TARTU ÜLIKOOL
ARVUTITEADUSE INSTITUUT

Algoritmid ja andmestruktuurid

Ülesannete kogu

Versioon 1.2 26. august 2016. a. 07:58

Koostajad: Ahti Peder
Jüri Kiho
Härmel Nestra

Tartu 2016

Käesoleva õppevahendi väljaandmist on toetanud Hariduse Infotehnoloogia Sihtasutuse IT Akadeemia programm.

Kaanekujundus: –
Toimetamine: Jüri Kiho

Sisukord

ISBN XX-YY-ZZ

Saateks

Käesolev õppematerjal kujutab endast ülesannete kogu ülikoolikursuse *Algoritmid ja andmestruktuurid* tarbeks.

Kogumiku liigendus jaotisteks vastab üldiselt selle aine kavale. Erandina on juurde võetud ka ülesandeid sõnetöötlusest, algoritmi korrektsusest ja planimeetriast (jaotis 18). Jaotisse 17 on koondatud ülesandeid, mis ei seondu (ilmutatult) mõne eelmise jaotise teemaga.

Enamus ülesannetes esinevaid mõisteid leiavad käsitlemist aine õpikus [1]. Mõningaid vajalikke lisamõisteid on täiendavalt kirjeldatud jaotises 19.

Kogumiku ülesannetest põhiosa moodustavad ülesanded, mida kogumiku koostajad on sõnastanud ja kasutanud selle aine õpetamisel mitmete aastate jooksul. Mõningaid ülesandeid või nende ideid on saadud ka allikatest [2], [3], [4], [5] ja [6], aga ka praktikumijuhendajatelt ning üliõpilastelt.

Mõnede ülesannete jaoks on raamatu lõpus ära toodud vastused või/ja lahendamise suunised. Nende olemasolu märgivad ülesande numbri järel sulgudes olevad tähed, vastavalt *v* või/ja *s*. Suunised võivad olla üsna erineva detailsusega, alates paarisõnalisest vihjest kuni näidislahenduseni.

Terminoloogiat ei ole püütud kanoniseerida ega paralleelvorme täielikult vältida, näiteks funktsioonidevahelisi O- ja Θ-relatsioone väljendame nii sõnaliselt kui ka kuuluvussümboli abil: f on $\Theta(g)$ ehk $f \in \Theta(g)$, f ei ole $O(g)$ ehk $f \notin O(g)$ jmt.

Järgnevas tabelis on pikemalt selgitatud ülesannete tekstides formuleeritud lakoonilisi nõudeid.

Ülesandes nõutud:	Interpretatsioon:
Sõnastada algoritm ...	Kirjeldada vabas vormis, nö oma sõnadega, algoritm ... .
Kirjutada algoritm ...	Esitada algoritm ... mingis vabalt valitud pseudokoodis.
Sooritada ... (algoritmi järgi)	Teha „käsitsi“ läbi, „mängida läbi“ ...; ühtlasi (soovitavalt) tehes märkmeid, jooniseid.
Koostada programm ...	Kirjutada ja siluda-testida programm ... vabalt valitavas programmeerimiskeeles (Java, Python, C++ jmt).
Programmeerida funktsioon ...	Realiseerida funktsioon ... vabalt valitavas programmeerimiskeeles ja koostada vastav testprogramm.

Küsimuse vormis sõnastatud ülesannete korral tuleb vastust põhjendada, mitte piirduda lühivastusega.

Kui on nõutud leida algoritmi Θ-hinnang, siis tuleb vastusena anda võimalikult lihtne funktsioon f, nii et algoritmi keerukus on $\Theta(f)$; f on niimitme muutuja funktsioon, kuimitmest parameetrist sõltub algoritmi andmemaht, käesolevas peamiselt ühemuutuja funktsioon, mõnedel juhtudel ka kahemuutuja funkt-

sioon. Käesolevas programmina (Python-funktsioonina, Java-meetodina) esitatud algoritmi ajalise keerukuse hindamisel loetakse omistamised ja aritmeetilised ning võrdlemistehted reeglina elementaaroperatsioonideks (keerukusega $\Theta(1)$ ehk $O(1)$). Seda juhul, kui ei ole öeldud teisiti.

Käesoleva ülesannete kogu koostajate eriline tänu kuulub Reimo Palmile vormistusliku tarkvaratexnilise toe eest. Tänuväärne on ka Mare Koidult saadud keelenõu.

Ebakorrektsustest ja soovitustest palume teada anda aadressil
`Ahti.Peder@ut.ee`.

1. Funktsiooni asümptootiline hinnang

Olgu f ja g naturaalarvuliste argumentidega ja positiivsete väärtustega funktsioonid. Siis f on asümptootiliselt ülalt tõkestatud funktsiooniga g (seda tähistame $f \in O(g)$), kui leiduvad $c > 0$ ja $N > 0$ nii, et $f(n) < cg(n)$ iga $n > N$ korral.

Kui $f \in O(g)$ ja $g \in O(f)$, siis ütleme, et f ja g on asümptootiliselt ekvivalentsed ning tähistame $f \in \Theta(g)$. Kuna Θ-relatsioon on ekvivalents, siis saame rääkida ekvivalentsiklassidest – aine kontekstis nimetame neid edaspidi keerukusklassideks. Ühte keerukusklassi kuuluvad omavahel asümptootiliselt ekvivalentsed funktsioonid. Kuuluvusseoste $f \in O(g)$ ja $f \in \Theta(g)$ asemel kasutame tekstis ka vastavalt väljendeid „f on $O(g)$“ ja „f on $\Theta(g)$“.

Asjaolu g on $O(f)$ tähendab ühtlasi ka seda, et funktsioon f on altpoolt asümptootiliselt tõkestatud funktsiooniga g; sellisel korral öeldakse, et f on $\Omega(g)$. [1]

Käesolevas jaotises toodud ülesannetes peab tõestused läbi viima, toetudes eelpool toodud definitsioonidele. Aine kontekstis eeldame, et $O(1) = \Theta(1)$.

1.1. Tõestada O- ja Θ-relatsioonide omadused ([1], lk 15):

(a) iga $k > 0$ korral, kf on $O(f)$;

(b) kui f on $O(g)$ ja h on $O(g)$, siis $(f+h)$ on $O(g)$;

(c) kui f on $O(g)$ ja g on $O(h)$, siis f on $O(h)$;

(d) n^r on $O(n^s)$, kui $0 \leq r \leq s$;

(e) kui p on d-astme polünoom, siis p on $\Theta(n^d)$;

(f) kui f on $O(g)$ ja h on $O(r)$, siis $O(f \cdot h)$ on $O(g \cdot r)$;

(g) n^k on $O(b^n)$, kui $b > 1$, $k \geq 0$;

(h) $\log_b n$ on $O(n^k)$, kui $b > 1$, $k > 0$;

(i) $\log_b n$ on $\Theta(\log_d n)$ iga $b, d > 1$ korral.

1.2. Millised eelmises ülesandes toodud O-relatsiooni omadused kehtivad ka Θ-relatsiooni kohta? Tõestada need Θ-relatsiooni omadused.

1.3. *(s)* Näidata, et reas

$$\boxed{c_1 | c_2 \log n | c_3 n | c_4 n \log n | c_5 n^2 | c_6 n^3 | c_7 2^n}$$

iga funktsioon on O-relatsioonis talle järgneva funktsiooniga ega ole O-relatsioonis talle eelneva funktsiooniga. ([1], lk 12)

1.4. *(s)* Näidata, et

(a) $12n\log n - 120\log n$ on $\Omega(n\log n)$;

(b) $12n\log n + 120\log n$ on $\Omega(n\log n)$;

(c) $\log_3 n + \sqrt{n}$ on $\Theta(\sqrt{n})$;

(d) $1500n + n\log n$ on $\Theta(n\log n)$;

(e) $n!$ ei ole $O(2^n)$.

1.5. *(v)* Kontrollida järgmiste väidete paikapidavust:

(a) $2^n \in O(3^n)$;

(b) $2^n \in \Theta(3^n)$;

(c) $\max(m,n) \in \Theta(m+n)$;

(d) $\min(m,n) \in \Theta(m+n)$;

1.6. Leida võimalikult lihtne Θ-hinnang funktsioonile

(a) $2n^2 + 3(\log n)^3$;

(b) $3n^2 + 5n\log n^5$.

1.7. Algoritmi täitmisaja sõltuvust sisendi suurusest n väljendagu funktsioon

(a) $(n-1)!$;

(b) $n(n+1)$;

(c) $(3+5)^8$;

(d) 2^{n+1};

(e) $3\log_2(n)$;

(f) $3n^2 + 2n - 5$;

(g) $(n^2+1)^{10}$;

(h) $\log_2^2(n)$;

(i) $\log_2(n^2)$;

(j) $0.01n^3 + 100n^2$;

(k) $2000n^2 + 5\ln(n)$;

(l) $\ln(n) + \log_8(n)$.

Kontrollida, millistesse järgmistest hulkadest see funktsioon kuulub:
$O(1)$, $O(\log n)$, $O(n)$, $O(n^2)$, $O(n^5)$, $O(2^n)$, $O(5^n)$, $\Theta(\log n)$, $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^5)$, $\Theta(2^n)$, $\Theta(5^n)$.

1.8. *(v)* Antud on mõnede algoritmide ajalist keerukust väljendavad funktsioonid:

(a) $0.01n^4 + 3n^3 + 1$;

(b) $n^{0.5}$;

(c) $(n-2)!$;

(d) $5\log((n+100)^{10})$;

(e) $\ln^2 n$;

(f) 2^{2n};

(g) 3^n.

Järjestada need keerukusklassi järgi asümptootiliselt aeglasemalt kasvavast kiiremini kasvavani.

1.9. Järjestada keerukusklassid asümptootiliselt kiiremini kasvavast aeglasemani:
$\Theta(n^6)$, $\Theta(2^n)$, $\Theta(n\log n)$, $\Theta(n)$, $\Theta(n!)$, $\Theta(n^2)$, $\Theta(\log n)$.

1.10. *(v)* Olgu funktsiooni $f(n)$ üldkujuks $5n^8 + 8g(n)$, kus $g \in \Theta(n)$. Millised järgnevatest võrdustest võivad kehtida?

(a) $f(n) = 4n^8 + 8n$;

(b) $f(n) = 5n^7 + 8n$;

(c) $f(n) = 5n^8 + 8n^2$;

(d) $f(n) = 5n^8$;

(e) $f(n) = 5n^8 + 3n$;

(f) $f(n) = 5n^8 + 4$;

(g) $f(n) = 5n^8 + 8n - 5$.

1.11. *(v)* Millised järgnevatest väidetest on tõesed?

(a) kui f on $O(g)$, siis f ei ole kindlasti $\Theta(g)$;

(b) kui f on $\Theta(g)$, siis f on ka $O(g)$;

(c) kui f on $\Theta(g)$, siis f ei ole kindlasti $O(g)$;

(d) kui f on $O(g)$, siis f on ka $\Theta(g)$;

(e) kui f on $O(g)$, siis g on ka $O(f)$;

(f) kui f on $\Theta(g)$, siis g on ka $\Theta(f)$.

2. Algoritmi ajaline keerukus

Algoritmi parima, keskmise ja halvima juhu ajalise keerukuse hindamisel eeldatakse vaikimisi Θ-hinnangu leidmist, erijuhul aga võimalikult aeglaselt kasvava funktsiooniga O-hinnangu esitamist. Kõrvalekalded sellest nõudest tuuakse välja ilmutatud kujul ülesande tekstis. Edaspidi lihtsustavalt: kui ei räägita ilmutatult halvimast ega parimast juhust, siis tegeldakse keskmise ajalise keerukuse hindamisega.

Ülesannete lahendamisel võib toetuda faktile, et algoritmi ajalise keerukuse O- ja Θ-hinnangu leidmiseks piisab arvutada andmemahu n korral töös sooritatavate elementaaroperatsioonide koguarv $c(n)$. Suuremas plaanis võime suvalist algoritmi käsitleda elementaaroperatsioonina, kui selle sooritamise ajaline kestus ei sõltu algoritmi sisendparameetrite väärtustest ja on tõkestatud ülalt mingi positiivse konstandiga.

Mõnes ülesandes võetakse lihtsuse mõttes elementaaroperatsiooniks tegevus, mis tegelikult seda pole. Algoritmide esitamispõhimõtted skeemina leiab õpikust [1].

2.1. *(v)* Iga alljärgneva Python-funktsiooni korral leida, mitu liitmistehet sooritatakse antud funktsiooni täitmisel etteantud n korral.

(a)
```
def f1(n):
    i = 0
    while i < n:
        print(i+1)
        i = i + 1
```

(b)
```
def f2(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print(i+j)
            j = j + 1
        i = i + 1
```

(c)
```
def f3(n):
    i = 0
        while i < n:
            j = i
            while j < n:
                print(i+j)
                j = j + 1
            i = i + 1
```

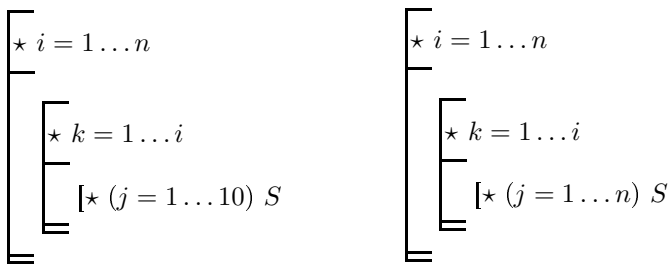
2.2. *(sv)* Iga alljärgneva Python-funktsiooni korral leida, mitu ekraanile väljastust sooritatakse antud funktsiooni täitmisel etteantud n korral.

(a)
```
def f4(n):
    i = n
    while i > 0:
        print(i)
        i = i // 2
```

(b)
```
def f5(n):
    i = n
    while i > 0:
        j = i
        while j > 0:
            print(i, j)
            j = j - 1
        i = i // 2
```

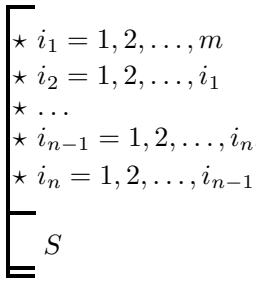
(c)
```
def f6(n):
    i = n
    while i > 0:
        j = i
        while j > 0:
            print(i, j)
            j = j // 2
        i = i - 1
```

2.3. *(v)* Hinnata joonisel 1 esitatud kahe algoritmi ajalist keerukust, kui alamalgoritmi S ajaline keerukus on $O(1)$.



Joonis 1: Kaks kolmekordset tsüklit.

2.4. Leida valem, mille järgi arvutada joonisel 2 kujutatud n-kordse tsükli ajaline keerukus antud m ja n korral (eeldades, et tsükli sisu S ajaline keerukus on $O(1)$).



Joonis 2: Mitmekordsete tsüklite pere üldkuju.

I Programmikoodi Θ-hinnangu leidmine

2.5. *(v)* Leida Python-funktsiooni

```
def f1(a):
   i = 0
   while i < len(a):
     print(a[i])
     i = i + 1
```

ajalise keerukuse Θ-hinnang, kui andmemahu määrab sisendlisti a pikkus.

2.6. *(v)* Leida Python-funktsiooni

```
def f2(a):
  i = 0
  while i < len(a):
    j = 0
    while j < len(a[i]):
      print(a[i][j])
      j = j + 1
    i = i + 1
```

ajalise keerukuse Θ-hinnang, kui andmemaht on määratud sisendtabeli a

(a) mõõtmetega (ridade arv *m*, veergude arv *n*);

(b) lahtrite arvuga.

2.7. *(v)* Eeldades, et Python-funktsiooni

```
def f3(a):
  i = 0
  while i < len(a) and i < len(a[i]):
    print(a[i][i])
    i = i + 1
```

sisendina kasutatakse vaid ruudukujulisi maatrikseid, leida selle funktsiooni halvima juhu, parima juhu ja keskmise ajalise keerukuse Θ-hinnangud, kui andmemaht

on määratud sisendmaatriksi a

(a) ridade arvuga;

(b) elementide arvuga.

2.8. *(v)* Leida Python-funktsiooni

```
def nullilejärgnevad(a):
    i = 0
    while i < len(a):
        while i < len(a) and a[i] != 0:
            i = i + 1
        i = i + 1
        if i < len(a):
            print(a[i])
```

parim ja halvim juht ning nende ajalise keerukuse Θ-hinnangud.

2.9. Leida Python-funktsiooni

```
def fun(a, i, j):
    if i == j:
        return a[i]
    kesk = (i + j) // 2
    return fun(a, i, kesk) + fun(a, kesk+1, j)
```

keskmise ajalise keerukuse Θ-hinnang.

2.10. *(sv)* Leida Java-meetodi

```
public static void m(int n){
  for (int i = 1; i <= n; i++){
    if (n < 10)
      for (int j = 1; j <= n; j++)
        System.out.println("Hello!");
    if (n < 100)
      for (int j = 1; j <= n; j++)
        System.out.println("Good Bye!");
  }
}
```

ajalise keerukuse Θ-hinnang.

2.11. *(v)* Leida Java-meetodi

```
public static void m1(int n){
  for (int i = 0; i < n; i++)
    for (int j = 0; j < i; j++)
      p(j);
}
```

ajalise keerukuse Θ-hinnang, kui meetodi `p` ajaline keerukus oma sisendi väärtuse j suhtes on

(a) $\Theta(j)$;

(b) $\Theta(2^j)$.

2.12. Leida Java-meetodi

```
public static int m2(int[] a){
  int i;
  for (i = 0; i < a.length; i++)
    if (a[i] % 2 != 0)
      break;
  return i;
}
```

halvima, keskmise ja parima juhu ajalise keerukuse Θ-hinnangud (sisendmassiivi `a` pikkuse suhtes).

2.13. *(v)* Leida Java-meetodi

```
public static void m3(int n){
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      q(i, j);
}
```

ajalise keerukuse Θ-hinnang, kui meetodi `q` ajaline keerukus oma sisendite väärtuste i, j suhtes on

(a) $\Theta(i+j)$;

(b) $\Theta(ij)$.

2.14. *(v)* Leida Java-meetodi

```
public static void m4(int n, int m){
  if (n > 0){
    for (int j = 1; j <= m; j++)
       p(j);
    m4(n-1, m);
  }
}
```

ajalise keerukuse Θ-hinnang sisendite väärtuste n, m suhtes , kui meetodi `p` ajaline keerukus oma sisendi väärtuse j suhtes on

(a) $O(1)$;

(b) $\Theta(\log j)$.

2.15. *(v)* Leida Java-meetodi

```
public static void m5(int n, int m){
  for (int i = n; i > 0; i /= 2)
     p(i*i);
  for (int i = m; i > 0; i -= 2)
     p(i+i);
}
```

ajalise keerukuse Θ-hinnang sisendite väärtuste n, m suhtes, kui meetodi `p` ajaline keerukus oma sisendi väärtuse j suhtes on

(a) $O(1)$;

(b) $\Theta(j)$.

2.16. Leida Java-meetodite `met1` ja `met2`

```
public static void met1(int n, int m){
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j += j)
       p(i, j);
}

public static void met2(int n, int m){
  if (m > 0){
    for (int i = 0; i < n; i++)
       p(i, m);
    met2(n, m-1);
  }
}
```

ajalise keerukuse Θ-hinnangud sisendite väärtuste n, m suhtes, kui meetodi `p(int i, int j)` ajaline keerukus on

(a) $O(1)$;

(b) $\Theta(j)$.

2.17. Leida Java-meetodi

```
public static void met3(int n, int m){
  for (int i = 1; i <= n; i++)
     p(i);
  for (int j = m * m; j > 0; j /= 2)
     q(n, j);
}
```

ajalise keerukuse Θ-hinnang sisendite väärtuste n, m suhtes, kui

(a) meetodid `p(int i)` ja `q(int i, int j)` on mõlemad ajalise keerukusega $O(1)$;

(b) meetodi `p(int i)` ajaline keerukus on $\Theta(i)$ ja meetodi `q(int i, int j)` ajaline keerukus $\Theta(i^2)$.

2.18. Leida Java-meetodi

```
public static void met4(int n, int m){
  int i, j, d;
  for (i = 2; i <= n; i ++){
    for (j = 1, d = 0; j <= m; j += ++d)
      p(i, j);
  }
}
```

ajalise keerukuse Θ-hinnang sisendite väärtuste n, m suhtes, kui meetodi `p(int i, int j)` ajaline keerukus on

(a) $O(1)$;

(b) $\Theta(\log i)$.

2.19. Leida Java-meetodi

```
public static void met5(int n, int m){
  if (m > 0)
    met5(n, m-1);
  else if (n > 0)
    met5(n-1, m);
  p(n, m);
}
```

ajalise keerukuse Θ-hinnang sisendite väärtuste n, m suhtes, kui meetodi `p(int i, int j)` ajaline keerukus on

(a) $O(1)$;

(b) $\Theta(i+j)$.

II Ajalise keerukuse hindamine

2.20. *(v)* Kui on teada, et algoritmi A keerukushinnang on $O(n)$, kas siis võib olla, et algoritmi A keerukushinnang on $\Theta(\log n)$?

2.21. *(s)* Kirjutada mitterekursiivne, järjendisiseselt töötav algoritm, mis kontrollib, kas antud sõnes kõik sümbolid on paarikaupa erinevad. Hinnata selle algoritmi ajalist keerukust.

2.22. Kirjutada rekursiivne algoritm järjendi summa leidmiseks, mis jagab järjendi neljaks enam-vähem võrdseks osaks ja siis tagastab nende osade summade summa. Hinnata selle algoritmi ajalist keerukust kasutades põhiteoreemi ([1], lk 20).

2.23. *(s)* Hinnata algoritmi ajalist keerukust, kui lahendusaeg T avaldub kujul

(a) $T(n) = 9T(n/3) + n$;

(b) $T(n) = 2T(n/2) + n^2$.

2.24. Leida ülesandes 2.7 nõutud halvima ja parima juhu keerukushinnangud juhul, kui sisendtabel ei tarvitse olla ruudukujuline.

2.25. Alljärgnevas on sõnastatud kaks algoritmi koostamaks antud järjendi nende elementide järjend, millest (antud järjendis) eespool paiknevate elementide seas ei ole sellest väiksemaid ega sellega võrdseid. (Näiteks $[3,4,5,1,6,7,0,2,8,2]$ korral on tulemuseks $[3,1,0]$.)

(a) Koostatav järjend on algselt tühi. Antud järjend läbitakse vasakult paremale; jooksev element lisatakse koostatava järjendi lõppu, kui koostatav järjend on tühi või vaadeldav element on koostatava järjendi viimasest elemendist väiksem.

(b) Koostatav järjend on algselt tühi. Antud järjend läbitakse paremalt vasakule; iga element lisatakse koostatava järjendi algusesse ja koos sellega eemaldatakse sealt kõik elemendid, mis paiknevad lisatu ja esimese temast väiksema elemendi vahel.

Leida kummagi algoritmi Θ-hinnang (eeldusel, et ühe kirje lisamine järjendisse ja ühe kirje eemaldamine järjendist on konstantse ajalise keerukusega).

2.26. *(sv)* Leida kahe n numbrist koosneva arvu käsitsi

(a) liitmise;

(b) lahutamise;

(c) korrutamise

tavaalgoritmi ajalise keerukuse Θ-hinnang.

2.27. *(s)* Kontroll, kas antud positiivne täisarv on arvu 2 aste, teostatakse järgmise algoritmi kohaselt.

Kui arv on paaritu, siis lõpetatakse. Kui arv on paarisarv, siis jagatakse see kahega ja korratakse tegevust tulemuseks saadud arvuga. Kui lõpetati tulemusega 1, siis on vastuseks JAH, vastasel korral on vastuseks EI.

Leida selle algoritmi halvima ja parima juhu Θ-hinnangud.

2.28. *(s)* Vaatleme algoritmi A, mis naturaalarvulise sisendi n korral toimetab järgmiselt:

- kui n on paarisarv, siis lõpetab töö;
- kui n on paaritu arv, siis teeb n korda tööd, mille ajaline keerukus on $O(1)$.

Tõestada, et selle algoritmi ajalise keerukuse hinnang

(a) ei ole $\Theta(n)$;

(b) ei ole $O(1)$.

2.29. *(sv)* Läbides maatriksi A mõõtmetega $n \times n$, töödeldakse iga elementi A_{ij} ajaga, mis kuulub keerukusklassi

(a) $O(1)$;

(b) $\Theta(\log n)$;

(c) $\Theta(i)$;

(d) $\Theta(ij)$;

(e) $\Theta(i+j)$.

Leida sellise algoritmi ajalise keerukuse Θ-hinnang.

2.30. *(sv)* Iga alljärgneva Python-funktsiooni korral kontrollida, millistesse järgmistest keerukusklassidest $O(1)$, $O(\log n)$, $O(n)$, $O(n\log n)$, $O(n^2)$, $O(2^n)$, $\Theta(\log n)$, $\Theta(n)$, $\Theta(n\log n)$, $\Theta(n^2)$, $\Theta(2^n)$ see funktsioon kuulub. Funktsiooni `fibo_arv` puhul eeldada, et arvu astendamine (a^n leidmine, a on konstant) on ajalise keerukusega $\Theta(\log n)$. Liitmistehte sooritus lugeda elementaaroperatsiooniks.

(a)
```
def fibo_arv(n): # Binet' valem
    phi1 = (1 + sqrt(5)) / 2
    phi2 = (1 - sqrt(5)) / 2
    return round((phi1**n-phi2**n)/sqrt(5))
```

(b)
```
def fibo_rek(n):
    if n < 3:
        return 1
    return fibo_rek(n-1) + fibo_rek(n-2)
```

(c)
```
def fibo_iter(n):
    if n < 3:
        return 1
    f1 = 1; f2 = 1
    for i in range(3, n+1):
        f3 = f1 + f2
        f1 = f2; f2 = f3
    return f3
```

2.31. *(s)* Leida funktsiooni `fibo_iter` (ülesandest 2.30) ajalise keerukuse Θ-hinnang, kui eeldada, et arvude liitmine on ajalise keerukusega $\Theta(max(p,r))$, kus p ja r on liidetavate kümnendkohtade arvud.

2.32. *(v)* Alljärgnevas on sõnastatud kaks algoritmi leidmaks järjendina antud n inimese masside järgi raskeima viiest inimesest koosneva rühma kogumass, kes saaks sõita 400 kg limiidiga liftis.

(1) Alustatakse jooksva massiga 0 kg. Koostatakse järjest kõik võimalikud komplektid järjendis antud kaaludest. Iga komplekti puhul, milles on täpselt 5 liiget, mille summa ei ületa 400 kg, võrreldakse summat jooksva massiga ja selle ületamisel loetakse jooksvaks massiks see summa.

(2) Alustatakse jooksva massiga 0 kg. Koostatakse järjest kõik võimalikud täpselt 5 liikmest koosnevad komplektid. Iga komplekti puhul, mille summa ei ületa 400 kg, võrreldakse summat jooksva massiga ja selle ületamisel loetakse jooksvaks massiks see summa.

Leida kummagi algoritmi keskmise ajalise keerukuse Θ-hinnang. Kumb algoritm on eelistatum, või on need võrdväärsed (st sama keerukusega)?

Leida teise algoritmi halvima juhu ajalise keerukuse Θ-hinnang.

2.33. Olgu tarvis kontrollida etteantud järjendi korral, kas selle mingi mittetühi algusosa kordub vahetult järgneva järjendiosa algul. Vaatleme järgmist (Python-funktsionina esitatud) naiivset algoritmi selle ülesande lahendamiseks.

```
def kontrolli_algused(a):
   pikkuse_piir = len(a) / 2
   seni_erinevad = True
   pikkus = 1
   while seni_erinevad and pikkus <= pikkuse_piir:
      i = 0
      seni_erinevad = False
      while not(seni_erinevad) and i < pikkus:
         if not(a[i] == a[pikkus+i]):
            seni_erinevad = True
         i = i + 1
```

```
        pikkus = pikkus + 1
    return not(seni_erinevad)
```

Leida selle algoritmi parima juhu ajalise keerukuse Θ-hinnang.

2.34. *(s)* Iga alljärgneva Python-funktsiooni jaoks leida võimalikult täpne keerukushinnang.

(a)
```
def aste1(a, n):
    tulemus = 1
    for i in range(n):
        tulemus = tulemus * a
    return tulemus
```

(b)
```
def aste2(a, n):
    if n == 0:
        return 1
    return a * aste2(a, n-1)
```

(c)
```
def aste3(a, n):
    if n == 0:
        return 1
    if n % 2 == 0:
        return aste3(a, n//2) * aste3(a, n//2)
    return aste3(a, n//2) * aste3(a, n//2) * a
```

(d)
```
def aste4(a, n):
    if n == 0:
        return 1
    pool_aste = aste4(a, n//2)
    if n % 2 == 0:
        return pool_aste * pool_aste
    return pool_aste * pool_aste * a
```

(e)
```
def aste5(a, n):
    if n == 0:
        return 1
    if n == 1:
        return a
    return aste5(a, n//2) * aste5(a, (n+1)//2)
```

2.35. Sõnastada mingi konkreetne algoritm, mille keskmise ja halvima juhu ajalise keerukuse hinnangud sisendparameetri n suhtes on mõlemad $\Theta(n^3)$.

2.36. Sõnastada mingi konkreetne algoritm, mille halvima juhu ajaline keerukus sisendparameetri n suhtes on $\Theta(n^3)$ ja keskmine ajaline keerukus on $\Theta(n^2)$.

2.37. Sõnastada mingi konkreetne algoritm, mis pole sorteerimismeetod, kuid mille keskmine ajaline keerukus sisendparameetri n suhtes on $\Theta(n \log n)$.

III Tööaja empiiriline hindamine

2.38. Koostada programm Fibonacci arvude $(F_0, F_1, \dots)$ leidmiseks kuluva aja mõõtmiseks, millega saaks konkreetsel arvutil katseliselt lahendada järgmised ülesanded.

(a) Leida suurim indeks k, millele vastava Fibonacci arvu F_k on arvuti võimeline välja arvutama 1 sekundi jooksul rekursiivse meetodiga (rekurrentse seose $F_n = F_{n-1} + F_{n-2}$ kohaselt).

(b) Teha kindlaks selle Fibonacci arvu F_k leidmise tööaeg, kui Fibonacci arvu leidmine on realiseeritud iteratiivse algoritmi kohaselt, vt ülesanne 2.30(c).

(c) Määrata selle iteratiivse algoritmi puhul suurim indeks, millele vastav Fibonacci arv leitakse 1 sekundi jooksul.

2.39. Automaat koostab ühikruutudest malelauda mõõtmetega $2^n \times 2^n$ nii, et teeb valmis 4 võrdset värvitud ruudustikku ja seejärel paneb need kokku. On teada, et nii ühe ühikruudu värvimine kui ka ruuduplokkide kokkutõstmine võtab ühe ajaühiku. Leida algoritmi ajalise keerukuse Θ-hinnang ja aeg, mis kulub 8×8 malelaua valmistamiseks.

2.40. *(sv)* Juku oli puudunud mõnedest tundidest, kus käsitleti bitivektorite loendamist. Seetõttu ei suutnud ta kuidagi uskuda, et näiteks 25-elemendilisi bitivektoreid on nii palju (33554432), kui õppejõud väidab. Selle kontrollimiseks kirjutas ta Python-funktsiooni

```
def bitt_gen(n, vektor = ""):
    if len(vektor) == n:
        return 1
    return bitt_gen(n, vektor+"0") + bitt_gen(n, vektor+"1")
```

mis leiab kõigi erinevate n-elemendiliste bitivektorite arvu. Oma arvutil sai Juku 25 elemendi korral vastuse kätte 10 sekundiga.

(a) Kui Jukul on loengu alguseni aega 60 minutit, siis mis on maksimaalne n, mille puhul Juku saaks vastuse kätte enne loengu algust?

(b) Tegelikult saab kõigi n-elemendiliste bitivektorite arvu leida palju lihtsamini ja märksa kiiremini. Kuidas?

2.41. *(s)* Priit, kes tegeleb DNA analüüsimisega, koostas Python-funktsiooni

```
def dna_mol_gen(n, genoomi_lõik = ""):
    if len(genoomi_lõik) == n:
        print(genoomi_lõik)
    else:
        for nukleotiid in ["A", "T", "C", "G"]:
            dna_mol_gen(n, genoomi_lõik+nukleotiid)
```

leidmaks kõik n nukleotiidi pikkused genoomilõigud. Käitades seda funktsiooni rakendavat programmi teadusarvutuste keskuse serveris, sai ta kätte kõik 20 nukleotiidi pikkused genoomilõigud 1 sekundiga.

(a) Mitu sekundit kulub Priidul aega, et saada samas serveris kätte kõik 26 nukleotiidi pikkused genoomilõigud?

(b) Kui Priidul oleks võimalik kasutada sama serverit 4 tundi, siis mis on maksimaalne pikkus, mille korral saaks Priit veel kõik sellise pikkusega genoomilõigud kätte?

2.42. *(s)* On teada, et järgnev Python-funktsioon väljastab konkreetsel arvutil 6-tähelise sisendsõne kõik permutatsioonid ühe sekundiga. Umbes mitu sekundit kulub aega 10-tähelise sisendi puhul?

```
def permutatsioon(sone, perm = ""):
  if len(sone) == 0:
     print(perm)
  else:
     for i in range(len(sone)):
        permutatsioon(sone[:i]+sone[i+1:], perm+sone[i])
```

2.43. Antud sõne sümbolite permutatsioonide leidmiseks on Pythonis koostatud järgmine rekursiivne generaator-funktsioon:

```
def gen_permut(s): # generaator-funktsioon
    # Antud: sõne s
    # Tulemus: antakse välja sõne s sümbolite
    #          järjekordne permutatsioon (sõnena)
    n = len(s)
    if  n <= 1:
       yield s
    else:
       for perm1 in gen_permut(s[1:]):
          # perm1 -- permutatsioon sõnest s ilma esimese
          #          sümbolita s[0]
          for i in range(n) :
             # i -- koht, kuhu sõnes perm1 lisada vahele
             #      sümbol s[0]
             yield perm1[:i] + s[0] + perm1[i:]
```

Sellele tuginedes programmeerida funktsioon $f(n)$, mille väärtuseks on n-sümbolilise sõne kõikide permutatsioonide genereerimiseks kuluv aeg (kasutataval arvutil). Tehes arvutil katseid, tabuleerida see funktsioon $n = 3,4,\ldots,12$ jaoks.

3. Hargnemistega algoritm. Rekursioon

Käesolevas jaotises loetakse n-sümbolilise sõne ekraanile väljastamise protseduuri ajaliseks keerukuseks $\Theta(n)$. Seda juhul, kui ei ole öeldud teisiti.

Jaotises kasutatava väljendiga „lõpetab töö“ mõeldakse programmi omadust täita kõik väljakutsutavad direktiivid ja tagastada tulemus (praktikas esineda võivat mälu- või ajapuudust arvestamata).

3.1. Millise väärtuse tagastab alljärgnev Python-funktsioon sisendi $n = 8$ korral?

(a)
```
def f(n):
    if n == 0:
        return 0
    return 2*n-1 + f(n-1)
```

(b)
```
def f(n):
    if n == 1:
        return 1
    return n + f(n-1)
```

3.2. *(v)* Mitu liitmistehet sooritatakse järgnevas Python-funktsioonis Fibonacci jada n-nda liikme leidmiseks rekursiivse definitsiooni järgi $n = 10$ korral?

```
def fibo_rek(n):
    if n < 3:
        return 1
    return fibo_rek(n-1) + fibo_rek(n-2)
```

3.3. *(v)* Milline sõne prinditakse järgnevate Python-funktsioonide poolt sisendi „REKURSIOON“ korral?

(a)
```
def tag(sõne):
    if len(sõne) == 0:
        return
    print(sõne[0], end=' ')
    tag(sõne[1:])
```

(b)
```
def tag(sõne):
    if len(sõne) == 0:
        return
    tag(sõne[1:])
    print(sõne[0], end=' ')
```

(c)
```
def tag(sõne):
    if len(sõne) == 0:
        return
    print(sõne[0], end=' ')
    tag(sõne[1:])
    print(sõne[0], end=' ')
```

(d)
```
def tag_a(sõne):
    if len(sõne) == 0:
        return
    print(sõne[0], end=' ')
    tag_b(sõne[1:])
```

(e)
```
def tag_b(sõne):
    if len(sõne) == 0:
        return
    tag_a(sõne[1:])
    print(sõne[0], end=' ')
```

3.4. *(v)* Mitu tõstmist tehakse (ehk mitu rida prinditakse alljärgnevas Python-funktsioonis) n kettaga Hanoi tornide ülesande lahendamisel rekursiivse definitsiooni järgi $n = 10$ korral? Mitu funktsiooni `tõsta` väljakutset sooritatakse?

```
def hanoi(n):
    def tõsta(n, kust, kuhu, ajutine):
        if n == 1:
            print("Tõsta ketas tornist",kust,"torni",kuhu+".")
        else:
            tõsta(n-1, kust, ajutine, kuhu)
            tõsta(1, kust, kuhu, ajutine)
            tõsta(n-1, ajutine, kuhu, kust)
    tõsta(n, "A", "B", "C")
```

3.5. Olgu ülesandeks väljastada ekraanile etteantud n ja k korral kõik pikkusega n bitivektorid, milles on täpselt k ühte. Kaks sellekohast Python-funktsiooni:

(a) Lootusetu haru äralõikamisega variant:

```
def bit1(n, k, yhtesid=0, tee=""):
    # n - vektori pikkus, k - lubatud '1'-de arv
    # yhtesid - '1'-de loendur, tee - bitivektori prefiks
    if yhtesid == k: # meil on täpselt k '1'-e
        lisaks = (n - len(tee)) * '0'
        print(tee + lisaks)
        return
    if len(tee) == n: return
    bit(n, k, yhtesid, tee+'0')
    bit(n, k, yhtesid+1, tee+'1')
```

(b) Kõikide juhtude läbivaatamisega variant:

```
def bit(n, k, yhtesid=0, tee=""):
    # n - vektori pikkus
    # k - lubatud '1'-de arv
    if len(tee) == n:
        if yhtesid == k:
            print(tee)
        return
    bit(n, k, yhtesid, tee+'0')
    bit(n, k, yhtesid+1, tee+'1')
```

Mitu funktsiooni `bit` väljakutset tehakse $n = 6$ ja $k = 2$ korral teises variandis rohkem (st kui ei lõpetata tööd harus, kus leitud vektoris on k ühte juba olemas)?

3.6. *(v)* Milliste parameetritega tuleb teha järgnevates Python-funktsioonides rekursiivne väljakutse, et tulemuseks oleks järjestikuste naturaalarvude korrutis a-st b-ni?

(a)
```
def korrutis(a, b):
    if a == b:
        return a
    return a * korrutis(    ,    )
```

(b)
```
def korrutis(a, b):
    if a == b:
        return b
    return b * korrutis(    ,    )
```

(c)
```
def korrutis(a, b):
    if a == b:
        return b
    return a * korrutis(    ,    )
```

(d)
```
def korrutis(a, b):
    if a > b:
        return 1
    return a * korrutis(    ,    )
```

(e)
```
def korrutis(a, b):
    if a > b:
        return 1
    if a == b:
        return a
    return a * b * korrutis(    ,    )
```

3.7. *(v)* Millist ülesannet lahendab alljärgnev Python-funktsioon?

(a)
```
def f(n):
    if n < 0:
        return f(-n)
    if n == 0:
        return True
    if n < 1:
        return False
    return f(n-1)
```

(b)
```
def f(a, b):
    if b == 0:
        return 1
    return a * f(a, b-1)
```

(c)
```
def f(a, b):
    if b == 0:
        return 0
    return a + f(a, b-1)
```

(d)
```
def f(n):
    if n < 0:
        return False
    if n == 0:
        return True
    return f(n-1)
```

3.8. *(sv)* Piirdume siin seljakotiülesande ([1], lk 16) lihtsustatud variandiga, kus esemete hindu ei arvestata. Ülesanne seisneb võimalikult raske, kuid seljakoti kandevõimet mitteületava esemetekomplekti koostamises.

Olgu antud järgnevad kaks Python-funktsiooni seljakotiülesande lahendamiseks; a on esemete kaalude järjend ja k on seljakoti kandevõime. Mitu korda vaadeldakse kummagi funktsiooni täitmisel igat esemete alamhulka (mis ei asu lootusetus harus)?

(a)
```
def seljakott(a, k):
   if len(a) == 0 or k <= 0:
      return []
   a1 = [a[0]] + seljakott(a[1:], k-a[0])
   a2 = seljakott(a[1:], k)
   s1 = sum(a1)
   s2 = sum(a2)
   if s1 > k or s2 > s1:
      return a2
   return a1
```

(b)
```
def seljakott(a, k):
   if len(a) == 0 or k <= 0:
      return []
   parimA = []
   parimSumma = 0
   for i in range(len(a)):
      b = [a[i]] + seljakott(a[:i]+a[i+1:], k-a[i])
      s = sum(b)
      if s > parimSumma and s <= k:
         parimA = b
         parimSumma = s
   return parimA
```

3.9. Iga alljärgneva Python-programmi korral näidata funktsiooniväljakutsete magasini muutumine arvutuse käigus. Selleks joonistada magasini seis (ajalises järjestuses) pärast iga väljakutset ja pärast iga funktsioonist tagasipöördumist. Magasini elemendiks olgu funktsiooni nimi koos argumendi väärtusega. Täpselt korduvaid osi võib asendada kordusmärkidega.

(a)
```
def pr1(n):
  def abi(a):
    if len(a) < n:
      abi(a+[0])
      abi(a+[1])
    else: print(a)
  abi([])
pr1(3)
```

(b)
```
def pr2(n):
  def abi(a):
    if len(a) < n:
      abi(a+[0])
      abi(a+[1])
      abi(a+[2])
    else: print(a)
  abi([])
pr2(2)
```

(c)
```
def fib(n):
  if n >= 2: return fib(n-1) + fib(n-2)
  if n >= 0: return n
  a = fib(-n)
  if n % 2 == 0: return -a
  return a
fib(3)
```

(d)
```
def perm(a):
  def abi(x, y):
    if len(y) > 0:
      i = 0
      while i < len(y):
        z = y[:]
        z.pop(i)
        abi(x+[y[i]], z)
        i += 1
    else: print(x)
  abi([], a)
perm([1,2,3])
```

3.10. *(v)* Olgu antud järgmine rekursiivne Python-funktsioon, mis peaks tagastama kahe arvu korrutise.

```
def korruta(a, b):
    if b == 1:
        return a
    return a + korruta(a, b-1)
```

Kas see funktsioon peatub (lõpetab töö) kõigi täisarvuliste sisendipaaride korral?

3.11. *(v)* Olgu antud järgmine rekursiivne Python-funktsioon, mis peaks leidma arvu n kahega jagamisel tekkiva jäägi.

```
def mod2(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return mod2(n-2)
```

Määrata, kas see funktsioon peatub (lõpetab töö)

(a) iga täisarvulise sisendi korral;

(b) iga positiivse paarisarvulise sisendi korral;

(c) iga positiivse paaritu sisendi korral;

(d) iga naturaalarvulise sisendi korral.

3.12. Määrata, kas Python-funktsioon

```
def mod2(n):
    if n == 0:
        return 0
    return mod2(n-2)
```

peatub (lõpetab töö)

(a) iga positiivse paarisarvulise sisendi korral;

(b) iga positiivse paaritu sisendi korral;

(c) iga naturaalarvulise sisendi korral.

3.13. *(v)* Olgu antud järgmised Python-funktsioonid naturaalarvude paarsuse tuvastamiseks:

```
def paaris(n):
    if n == 0:
        return True
    return paaritu(n-1)

def paaritu(n):
    if n == 0:
        return False
    return paaris(n-1)
```

Selgitada, milliste täisarvude $k \geq 0$ korral leidub väärtus avaldisel

(a) `paaris(k)`;

(b) `paaritu(k)`.

Leida, milliste k väärtuste korral leidub neil avaldistel väärtus, kui funktsioonid `paaris` ja `paaritu` oleksid defineeritud kui

```
def paaris(n):
    if n == 0:
        return True
    return not paaritu(n-1)

def paaritu(n):
    if n == 1:
        return True
    return not paaris(n-1)
```

3.14. Sõnastada „jaga ja valitse“ tüüpi algoritm järjendist suurima ja vähima elemendi leidmiseks.

3.15. *(s)* Programmeerida funktsioon, mis väljastab antud naturaalarvu kõik-võimalikud lahutused

(a) liidetavate 1 ja 2 summadeks;

(b) liidetavate 2, 4 ja 6 summadeks.

Liidetavate järjekorra poolest erinevad summad lugeda erinevateks. Näiteks juhul (a) väljastataks (mingis järjestuses) argumendile 3 vastavalt $[1,1,1]$, $[1,2]$, $[2,1]$.

3.16. Pliiatseid tootva vabriku tööline kontrollib, kas igas tsehhist väljuvas karbis on täpselt nõutud arvul pliiatseid. Ta loendab pliiatseid rühmade kaupa, milles

on alati rohkem kui 1 pliiats. Näiteks 10 pliiatsiga karbi puhul saab ta kahekaupa loendades vahesummad 2, 4, 6, 8, 10; aga ta võib võtta ka nt rühmad suurusega 3, 3, 4, mispuhul saab ta vahesummad 3, 6, 10.

Programmeerida funktsioon sisendparameetriga n, mis väljastab kõik võimalused, millised vahesummade jadad saavad tekkida pliiatsite loendamisel, kui karbis on täpselt n pliiatsit. Näiteks $n = 10$ korral väljastatakse teiste hulgas `[2, 4, 6, 8, 10]` ja `[3, 6, 10]`, aga ka `[10]` (viimane vastab juhule, kui üksainus rühm sisaldab kõik pliiatsid). Arvestada, et ka viimasesse rühma peab alati jääma rohkem kui 1 pliiats.

3.17. *(s)* Programmeerida funktsioon, mis saab argumendiks kaks järjendit ja väljastab kõik sellised järjendid, mis sisaldavad parajasti mõlema antud järjendi kõik elemendid, nii et kummagi järjendi elementide omavaheline järjestus pole muutunud.

Näiteks sisendjärjendite `[„kas“, „mina“]` ja `[„olen“, „siin“]` korral väljastatakse

„kas mina olen siin“,
„kas olen mina siin“,
„kas olen siin mina“,
„olen kas mina siin“,
„olen kas siin mina“,
„olen siin kas mina“.

4. Variantide läbivaatamine

Selle jaotise ülesannete vastustes ei tohi kasutada funktsiooniväliseid ehk globaalseid muutujaid. Tuleb võimalikult minimeerida kasutute rekursiivsete väljakutsete arv. Eelistatud on üldisemad lahendused, milles andmemaht antakse sisendparameetrina, seda ka juhul kui ülesandes on tegemist konstandiga (bittide, isikute, toodete jmt arv).

4.1. Koostada programm, mis väljastab kõik bitivektorid, mille pikkus on 7.

4.2. Programmeerida sisendparameetriga $k > 0$ funktsioon, mis väljastab kõik pikkusega 30 bitivektorid, milles on parajasti k ühte.

4.3. Koostada programm, mis väljastab järjendina etteantud hulga kõik alamhulgad.

4.4. *(s)* Jukul on taskus rahamündid vääringutega $x_1, x_2, \ldots, x_n$, igas vääringus kaks münti. Ülesandeks on välja selgitada, kas Juku saab poes tasuda müntidega oma taskust täpselt summa s. Programmeerida vastav loogilist tüüpi funktsioon.

4.5. *(s)* Järjendisse on salvestatud 26 erineva toote hinnad. Programmeerida funktsioon, mis sellise etteantud järjendi puhul tagastab suurima toodete sellise valiku koguhinna, mis ei ületa 100 eurot ja kus igat toodet on võetud maksimaalselt 2 tükki.

4.6. Järjendisse on salvestatud 26 erineva ehitustööriista kaalud. Programmeerida funktsioon, mis sellise etteantud järjendi korral

(a) kontrollib, kas sellest järjendist mingi elementide valik, milles on mitte rohkem kui 13 tööriista, annab kogukaaluks täpselt 10 kg;

(b) leiab tööriistade komplekti, mille kaal ületab 10 kg võimalikult vähe.

4.7. *(s)* Programmeerida rekursiivne funktsioon, mis etteantud inimeste masse (kg) sisaldava järjendi korral väljastab ekraanile üksikkaalude järjendite näol kõik sellised inimeste valikud, mille kogumass jääb lõiku 950–1050 kg.

4.8. *(s)* Programmeerida funktsioon, mis etteantud raamatute hindu sisaldava järjendi korral leiab kõikide selliste raamatukomplektide arvu, mille koguhind jääb lõiku $[50, 100]$ eurot.

4.9. *(s)* Sõjaväerongi koosseisu komplekteerimisel kasutatakse nelja tüüpi vaguneid: A, B, C ja D. Ohutuse huvides kehtivad järgmised komplekteerimisnõuded:

- C-osa ei või vahetult järgneda B-osale ega B-osa C-osale;
- D-osa ei või vahetult järgneda A-osale;
- järjestikused ühte ja sama tüüpi osad pole lubatud.

Programmeerida funktsioon, mis etteantud vagunite arvu korral väljastab kõik võimalikud sellistele tingimustele vastavad järjestatud rongikoosseisud, ja ka nende koguarvu.

4.10. Raudteefirma soovib koostada rongi, mis koosneks kolme tüüpi vagunitest A, B, C vastavalt pikkustega 16, 18, 21 meetrit. Programmeerida funktsioon, mis argumendiks saadud mittenegatiivse täisarvu n jaoks tagastab listi kõigist võimalikest rongikoosseisudest, mille pikkus on suurem kui $2n$ ja väiksem kui $3n$. Tuleb arvestada, et vagunite kokkuhaakimisel jääb nende vahele ühemeetrine vahe.

4.11. *(s)* Programmeerida funktsioon, mis tagastab arvu, mitmel erineval viisil saab üles minna n-astmelisest trepist, kui iga sammuga võib võtta ühe, kaks või kolm astet, ja väljastab kõikvõimalikud sellised teekonnad listidena sammupikkustest.

4.12. Programmeerida funktsioon, mis etteantud täisarvuliste elementide järjendi korral tagastab kõikvõimalike alamjärjendite summad sorteeritud unikaalsete (summa esineb tulemuses täpselt üks kord) elementidega järjendina.

4.13. *(sv)* Päevapiltnik soovib pildistada rivis seisvast n õpilasest moodustatud kõikvõimalikke osarivisid (mingi arv järjest kõrvutiseisvaid õpilasi). Näiteks õpilaste rivi $[a_1, a_2, a_3, a_4]$ korral saab moodustada osarivisid $[a_1]$, $[a_1, a_2]$, $[a_1, a_2, a_3]$, $[a_1, a_2, a_3, a_4]$, $[a_2]$, $[a_2, a_3]$, $[a_2, a_3, a_4]$, $[a_3]$, $[a_3, a_4]$ ja $[a_4]$. Kirjutada programm, mis väljastab kõik osarivid.

Leida koostatud programmi ajalise keerukuse Θ-hinnang.

4.14. Koostada programm, mis väljastab antud sõne kõik permutatsioonid. Loodud programmi katsetades teha kindlaks, millise pikkusega sõne korral on see ülesanne teostatav veel alla ühe sekundi.

4.15. Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: järjend a (listina).

Tulemus: järjendi a elementide permutatsioonid (listide listina).

4.16. Koostada programm, mis leiab võimalikult paljude ratsude paigutuse $n \times n$ malelauale nii, et ükski ratsu ei oleks ühegi teise ratsu tules. Soovituslik lisaülesanne lisatingimusega: pooled (± 1, kui paigutatud ratsusid on paaritu arv) ratsudest peavad asuma valgetel ruutudel, ülejäänud mustadel ruutudel.

4.17. *(s)* n-liikmelisest treeningrühmast ($n = 10 \pm 2$) tuleb eelolevaks võistluseks välja valida neljase bobi meeskond. Treeningrühma iga liikme kohta on antud nimi, kaal ja naturaalarvuline reiting (mis näitab treenituse taset). Koostada programm, mis leiab sellise bobimeeskonna, mille kogukaal ei ületa 325kg ja mille liikmete reitingute summa on võimalikult suur.

4.18. 19 asutust moodustab asutuste juhtidest koosnevat viieliikmelist streigikomiteed. Iga liikmekandidaadi kohta on antud nimi, sugu ja vanus. Koostada programm, mis leiab sellise komitee, milles vähemalt kolm liikmetest on naissoost (soolise võrdõiguslikkuse ja võrdse kohtlemise voliniku nõue!) ja komitee keskmine vanus on võimalikult väike.

4.19. Eesti „ärimees“ tuleb USA-st ja tahab endaga kaasa tuua võimalikult suurt kasumit andva komplekti tollimaksuvabu (elektroonika)kaupu. Tema pagasi kaal ei tohi ületada 10 kg ja mitte ühtegi kaubaartiklit ei tohi olla rohkem kui üks. Antud on kaupade loetelu, igaühe hind USA turul, hind Eestis (käibemaksuga) ja kaal. Koostada programm, mis etteantud hindade ja kaaludega kaupade jaoks leiab suurimat vaheltkasu andva pagasi.

4.20. Antud on töötajate nimekiri, näiteks `1.Mari 2.Jüri 3.Ants 4.Rein 5.Kaia 6.Jaan` ja sobivusmaatriks (s_{ij}), mille element s_{ij} näitab töötajate i ja j vastastikuse sobivuse taset reaalarvuna lõigul $[0;1]$.

Koostada programm, mis leiab võimalikult suure rühma töötajaid, mille korral vastastikuste sobivuste keskmine on suurem kui 0.5.

4.21. *(s)* Programmeerida mitterekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: järjend a (listina) ning naturaalarv k.

Tulemus: väljastatud järjendi a elementide kombinatsioonid k kaupa (listidena).

4.22. Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: täisarvud n ja k, $1 \leq k \leq n$.

Tulemus: järjendi 0, 1, ..., n-1 elementide kombinatsioonid k kaupa (listide listina).

4.23. Programmeerida

(a) rekursiivne generaator-funktsioon;

(b) mitterekursiivne generaator-funktsioon

järgmise ülesande lahendamiseks.

Antud: hulk a (listina).

Tulemus: järjekordsel pöördumisel antakse välja a järjekordne alamhulk (listina).

4.24. Koostada programm seljakotiülesande ([1], lk 16) lahendamiseks, milles

(a) vaadatakse läbi kõikvõimalikud esemete komplektid (nn jõumeetod);

(b) „ülekaaluliste“ esemekomplektide läbivaatamine on tõkestatud.

5. Otsimisalgoritmid järjenditel

Otsingumeetoditest käsitletakse kursusel järjestik- ja kahendotsingut ning topeltkahendotsingut. Kahendotsingut kasutatakse vaid andmestruktuuridel, millel on realiseeritud elementidele otseligipääs läbi indeksite. Ka eeldatakse, et reeglina ei ole topeltkahendotsingul teada otsinguakna suurus ega elementide väärtusvahemik.

5.1. *(v)* Miks on kahendotsimisel soovitav otsinguala jaotada kaheks võimalikult keskelt?

5.2. *(s)* Milline on n-elemendilisel järjestatud kirjetega massiivil teostatava kahendotsingu parima juhu ja halvima juhu ajaline keerukus?

5.3. *(v)* Millist meetodit sobib kasutada etteantud võtmega kirje otsimiseks järjestatud lihtahelast?

5.4. Millist algoritmi on sobiv kasutada kindla võtmega kirje otsimiseks kahemõõtmelisest maatriksist, mille iga rida ja veerg kujutab endast kasvavat järjendit?

5.5. *(s)* Olgu antud massiiv

11	13	16	20	25	29	32	34	35	37	40	44	49	53	56	58	59	61	64	68

Näidata ajalises järjestuses kõik võrdlused, mis sooritatakse sellest massiivist arvu otsimise käigus kahendotsinguga, päripidi topeltkahendotsinguga ja tagurpidi topeltkahendotsinguga, kui otsitavaks on

(a) arv 51;

(b) arv 35.

5.6. *(v)* Olgu antud kahemõõtmeline massiiv

10	11	13	16	20	25	31	38	46	55	65	76	88	91	92	93	94	95	96	97
04	05	08	10	12	15	16	20	24	25	27	30	32	35	40	45	50	55	60	64
02	04	07	09	11	14	15	18	21	24	26	29	31	34	37	40	42	45	47	50
01	03	05	07	09	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39

(a) Näidata kõik kohad massiivis, kus toimub sadulameetodil arvu 41 otsimise käigus otsingu suuna muutus. Teha seda otsingu kõigi võimalike alguspunktide jaoks.

(b) Näidata ajalises järjestuses kõik sooritatavad võrdlused, mis sooritatakse sellest massiivist arvu 41 otsimise käigus sadulameetodil, kui pikema külje sihis kasutatakse topeltkahendotsingut. Teha seda otsingu kõigi võimalike alguspunktide jaoks.

5.7. Olgu antud kahemõõtmeline massiiv

01	02	04	05	08	10	12	13	14	15	16	17	18	20	21	24	25	26	27	28
02	04	06	08	12	16	20	24	32	40	48	56	60	64	68	72	80	88	92	96
09	10	12	15	19	24	30	37	45	54	64	73	81	88	94	95	96	97	98	99

(a) Näidata kõik kohad massiivis, kus toimub sadulameetodil arvu 21 otsimise käigus otsingu suuna muutus. Teha seda otsingu kõigi võimalike alguspunktide jaoks.

(b) Näidata ajalises järjestuses kõik sooritatavad võrdlused, mis sooritatakse sellest massiivist arvu 21 otsimise käigus sadulameetodil, kui pikema külje sihis kasutatakse topeltkahendotsingut. Teha seda otsingu kõigi võimalike alguspunktide jaoks.

5.8. *(s)* Olgu antud kahemõõtmelised massiivid

23	25	30	32	33	43	47	52	55	56	60	65	69	70	71	74	78	84	90	93
20	21	22	24	28	31	33	35	40	45	50	55	60	65	68	72	75	77	78	80
15	16	17	19	21	26	28	30	32	34	36	38	42	46	50	58	62	66	70	74
12	13	14	15	18	24	25	27	28	30	32	33	34	35	36	48	49	50	60	72

13	15	16	19	28	30	32	34	35	36	46	47	48	50	52	57	58	65	66	67
14	16	17	18	21	25	26	29	31	32	45	48	50	51	57	58	59	60	63	68
11	14	18	20	22	23	27	28	30	33	36	38	39	40	46	60	62	64	65	69
12	13	14	15	23	24	25	27	28	30	32	33	34	35	40	48	49	50	60	72

13	12	20	16	23	24	25	26	27	28	30	32	35	36	37	38	39	42	45	46
16	14	22	24	25	26	28	27	30	34	32	35	36	37	38	39	40	44	46	50
17	16	27	25	27	28	30	28	32	36	34	36	37	38	39	40	41	45	47	54
20	18	28	29	28	30	32	31	34	37	36	39	38	39	40	44	42	46	48	56

13	20	22	24	28	30	32	34	35	36	46	47	48	55	57	58	61	65	71	76
12	16	17	18	21	25	27	29	31	33	38	43	45	50	52	53	56	60	62	63
11	14	15	16	20	23	26	28	30	32	34	36	38	40	44	50	52	54	56	59
10	11	12	13	18	20	21	22	23	25	27	28	29	30	35	38	40	45	50	58

Teha kindlaks, millistes neist on võimalik rakendada arvu otsimiseks sadulameetodit. Seejärel sooritada vastava(te)st massiivi(de)st arvu 61 otsimine sadulameetodi sellise variandiga, mis üldjuhul on efektiivseim, kõigi võimalike alguspunktide jaoks. Näidata ajalises järjestuses kõik töö käigus tehtavad võrdlused.

5.9. Olgu antud kaks kahemõõtmelist massiivi

83	75	70	67	63	62	61	57	55	51	50	45	39	35	31	29	28	24	20	18
85	81	77	74	73	71	68	65	60	56	55	54	50	45	43	42	35	32	28	25
90	86	80	79	76	74	73	70	67	64	63	58	57	56	50	48	36	35	30	29
92	88	84	80	78	77	75	74	73	70	67	63	59	58	56	53	49	45	40	32

23	25	26	34	38	40	42	45	50	51	56	57	58	60	67	68	78	80	86	87
14	26	27	33	36	38	41	44	46	47	50	54	54	55	62	63	74	75	78	83
21	24	28	30	32	33	37	38	40	43	46	48	49	50	56	60	72	74	75	79
18	19	24	25	28	29	30	32	34	36	38	40	42	45	49	52	54	56	60	72

Kummas neist on võimalik rakendada arvu otsimiseks sadulameetodit? Sooritada sellest massiivist arvu 48 otsimine sadulameetodi variandiga, mis üldjuhul on efektiivseim, kõigi võimalike alguspunktide jaoks. Näidata võrdlemise ajalises järjestuses kõik massiivi elemendid, mida töö käigus otsitavaga võrreldakse.

5.10. Olgu antud naturaalarvuliste argumentide ja reaalarvuliste väärtustega kahemuutuja funktsioon f. On teada, et f on mõlema argumendi järgi monotoonne, st kui $x_1 \leq x_2$ ja $y_1 \leq y_2$, siis $f(x_1, y_1) \leq f(x_2, y_2)$. Veel on teada, et iga reaalarvu v korral leiduvad argumendid i ja j, mille korral $f(i, j) > v$. Sõnastada võimalikult efektiivne algoritm, mis suvalise etteantud arvu v jaoks teeb kindlaks, kas leiduvad argumendid i ja j, mille korral $f(i, j) = v$.

5.11. *(s)* Poiss kohtub uusaastaööl tüdrukuga võõralt planeedilt, kelle vanus võib olla kuitahes suur. Ta soovib teada saada tüdruku vanust, aga tüdruk ei taha seda öelda. Siiski on tüdruk nõus õigesti vastama, kui poiss pakub välja mingi aastaarvu ja sündmuse ning küsib, kas see sündmus leidis aset selle numbriga aastast hiljem. Sõnastada algoritm, mille abil poiss saab võimalikult kiiresti tüdruku vanuse (aasta täpsusega) teada.

5.12. *(v)* Lahkudes võõrriigist, tahab kodanik X kogu enda omanduses oleva selle riigi valuuta lennujaama kioskis ära kulutada. Sõnastada võimalikult efektiivne algoritm, mis teeb kindlaks, kas kodanikul X peaks see õnnestuma, kui kioskis on sel päeval müügil ainult

(a) 2;

(b) 3

artiklit kaupa.

5.13. On teada, et massiivi elemendid asetsevad kuni teatud kohani kahanevalt ja sealt edasi kasvavalt. Sõnastada võimalikult efektiivne algoritm, mis leiab massiivi väikseima elemendi indeksi. Leida selle algoritmi halvima juhu keerukuse Θ-hinnang. Võib eeldada, et massiivi elemendid on paarikaupa erinevad.

5.14. Juku sai topeltkahendotsingu ideest valesti aru ja programmeeris selle nii, et topeldamise etapil leitud vahemikus rakendatakse rekursiivselt sedasama (Juku) topeltkahendotsingut (tavalise kahendotsingu asemel). Kas Juku algoritm on üldjuhul kiirem või aeglasem kui õige topeltkahendotsing või pole vahet?

6. Järjendi ümberkorraldamine

Käesolevas jaotises kasutatavate andmestruktuuridega opereerimisel peab lähtuma andmestruktuuri abstraktsest definitsioonist ja eeldama, et andmetega manipuleerimiseks ei ole lisaoperatsioone.

Massiivioperatsioonide realiseerimisel eeldame, et massiiv on staatiline andmestruktuur ja sinna ei saa „vahele kirjutada“. Ahela puhul saab „vahele kirjutada“, kuid puudub otseligipääs (ajaga $O(1)$) suvalisele selle elemendile, v.a esimesele ja mõnes realisatsioonis ka viimasele (topeltahela korral). Ligipääs ahela komponentidele realiseerub ajaga $O(1)$ vaid eelneva või järgneva kaudu.

I Lahkme järgi ümbertõstmine

6.1. *(v)* Olgu antud massiivid

20	21	12	15	25	31	29	11	23	14	13	30	35	19	24	26	28	27	32	23

30	22	11	35	19	36	23	21	20	27	28	29	25	12	17	36	18	32	31	37

Sooritada mõlema massiivi elementide jaotamine nii, et arvust 22 väiksemad oleksid massiivi alguses ja ülejäänud nende järel. Näidata seis pärast iga vahetust.

6.2. *(s)* Sooritada massiivi

56	48	16	17	62	64	65	30	19	18	60	49	63	50	75	24	27	41	47	69	34	59	20

elementide ümberpaigutamine (efektiivseima algoritmi kohaselt) nii, et 5 vähimat oleksid massiivi alguses ja ülejäänud nende järel. Näidata joonisel seis koos asjakohaste piiridega pärast iga jaotamist.

6.3. *(s)* Sooritada massiivi

26	24	36	23	31	32	15	20	29	28	30	17	37	25	35	12	27	21	13	39	14	38	10

elementide ümberpaigutamine (efektiivseima algoritmi kohaselt) nii, et 7 suurimat oleksid massiivi lõpus ja ülejäänud nende ees. Näidata joonisel seis koos asjakohaste piiridega pärast iga jaotamist.

6.4. *(v)* Olgu antud massiivid

30	45	36	62	61	23	55	35	29	40	78	80	77	75	50

70	80	65	15	26	37	50	49	33	18	28	78	66	25	27

Sooritada vähima elemendi paigutamine massiivi algusse, kasutades tagurpidi mullina liikumist. Näidata töö seis pärast iga vasakule liikuva elemendi peatumist.

6.5. *(v)* Olgu antud massiivid

24	32	28	19	37	15	30	35	23	16	31	22	20	17	37	27	11	18

12	25	31	28	10	1	11	13	5	7	8	20	4	18	22	23	24	15	30	2

5	29	26	18	19	20	11	26	28	9	1	6	14	31	23	15	12	21	16	7

Sooritada iga massiivi puhul elementide jaotamine nii, et 6 väiksemat oleksid massiivi alguses ja ülejäänud nende järel. Näidata massiivi seis koos töövälja piiridega pärast iga kaheksjaotamist.

6.6. *(v)* Vaatleme algoritmi, kus n-elemendilisest massiivist k vähima elemendi massiivi algusesse ümberpaigutamiseks kasutatakse alamprotseduurina korduvalt lahkme järgi jaotamist. Milline sisend on selle algoritmi jaoks halvim juht? Leida selle juhu ajalise keerukuse Θ-hinnang.

6.7. Millised juhud on keerukuselt halvimad „jaga ja valitse“ tüüpi algoritmile kindla arvu väiksemate elementide viimiseks massiivi algusse?

6.8. Sõnastada võimalikult madala halvima juhu keerukusega algoritm, mis etteantud n-elemendilisest arvujärjendist leiab 32 väikseimat elementi (kui $n < 32$, siis niipalju kui on). Anda ka halvima juhu ajalise keerukuse Θ-hinnang.

6.9. Sõnastada võimalikult väikese aja- ja lisamälutarbega algoritm, mis saab sisendiks n-elemendilise massiivi a ning naturaalarvu i ($0 < i < n$) ja paigutab elemendid ümber selliselt, et esimesed i elementi oleksid lõpus (järjestust muutmata) ning viimased $n - i$ elementi oleksid alguses (samuti järjestust muutmata). Esitada ajalise keerukuse ja lisamälukasutuse Θ-hinnangud.

II Sorteerimismeetodi realiseerimine

6.10. Juku programmeeris lihtahela järjestamiseks rekursiivse funktsiooni, mis tühja ahela jätab muutmata, mittetühja ahela puhul aga järjestab kõigepealt sama funktsiooniga ahela saba (st ahela osa, mis järgneb esimesele elemendile) ja seejärel „pistab“ ahela esimese elemendi ahela sabas õigele positsioonile.

Sooritada selle funktsiooni tegevus ahelal elementidega 54, 39, 33, 19, 30, 28, 61. Näidata joonisel ahela seis pärast iga „pistet“.

6.11. Juku programmeeris lihtahela järjestamiseks meetodi, mis töötab järgmisel põhimõttel. Tühi ja üheelemendiline ahel jäetakse puutumata, pikema ahela puhul tuuakse kõigepealt vähim element mullina ahela algusse ja seejärel järjestatakse ülejäänud kirjed rekursiivselt samal meetodil. Mullina ettetoomiseks (rekursiivselt) ei tehta üheelemendilises ahelas midagi, pikema ahela puhul tuuakse kõigepealt ahela sabas (st ahela osas, mis järgneb esimesele elemendile) vähim element mullina ette ning kui seejärel on ahela teine element esimesest väiksem, siis vahetatakse esimesed kaks elementi omavahel.

Sooritada selle funktsiooni tegevus ahelal, milles on järjest elemendid 73, 65, 41, 55, 28, 56, 30. Näidata joonisel ahela seis pärast iga „mullina ettetoomist“.

6.12. Programmeerida sorteerimisfunktsioon, mis põhineb magasinidesse jaotamisel.

I osa: paigutada järjendi elemendid magasinidesse, igas mittekasvavalt, nt
[7, 4, 5, 3, 1, 2, 8, 6, 4] → [7, 4, 3, **1**], [5, **2**], [8, 6, **4**]
(poolpaksult on tähistatud magasini ava).

II osa: korjata magasinidest tulemusjärjend.

6.13. Programmeerida sorteerimisfunktsioon, mis põhineb järjekordadesse jaotamisel.

I osa: paigutada järjendi elemendid järjekordadesse, igas mittekahanevalt (suunas esimene->viimane), nt
[1, -7, 4, 5, 8, 3, 6, -2, 3, 4] → [**1**, 4, 5, **8**], [**-7**, 3, **6**], [**-2**, 3, **4**]
(poolpaksult on tähistatud järjekorra avad).

II osa: korjata järjekordadest tulemusjärjend.

6.14. *(s)* Programmeerida sorteerimisfunktsioon (nn listimeetod), mis põhineb listidesse jaotamisel. Antud juhul on tegemist listidega, kuhu saab lisada nii algusesse kui ka lõppu, eemaldada aga saab ainult algusest.

(a) Listimeetodi I osa: paigutada järjendi elemendid listidesse, igas mittekahanevalt (suunas esimene->viimane), nt

[1, -7, 4, 5, 8, 3, 6, -2, 3, 4] → [**-7**, 1, 4, 5, **8**], [**-2**, 3, **6**], [**3**, **4**]
(poolpaksult on tähistatud listi avad).

Listimeetodi II osa: korjata listidest tulemusjärjend.

(b) Millist tüüpi järjend on listimeetodi jaoks soodne ja milline ebasoodne?

(c) Koostada programm, milles on funktsioonina realiseeritud sorteerimise listimeetod ja mis väljastab kolm selle meetodi tööaegade graafikut: juhujärjendite korral, soodsat tüüpi järjendite korral ja ebasoodsat tüüpi järjendite korral.

(d) Koostada programm, milles on funktsioonina realiseeritud sorteerimise listimeetod selliselt, et ei oleks võimalik ette anda (konstrueerida) ebasoodsat tüüpi järjendit.

6.15. Kahemõõtmelist massiivi saab järjestada kahel viisil.

(a) Käsitledes tabelit kui üht pikka ühemõõtmelist massiivi, milles elemendid ridade kaupa rivis, sorteeritakse see mingit massiivi järjestamise meetodit rakendades.

(b) Järjestatakse elemendid igas tabeli reas ja seejärel järjestatakse elemendid igas tabeli veerus eraldi, rakendades iga kord mingit ühte ja sama massiivi järjestamise meetodit.

(Tulemused ei ole tingimata ühesugused, kuid mõlemal juhul on tulemuses iga rida ja iga veerg omaette järjestatud.)

Kumb nendest kahest moodusest on efektiivsem, kui massiivi järjestamiseks kasutatava meetodi keerukus massiivi pikkuse n suhtes on

(a) $\Theta(n^2)$;

(b) $\Theta(n \log n)$?

III Klassikalised meetodid

6.16. Kirjutada mulli-, valiku- ja pistemeetodi rekursiivsed algoritmid.

6.17. On programmeeritud funktsioon $piste(a,k,m)$ järgmise ülesande lahendamiseks:

Antud: massiiv $a_1, a_2, \ldots, a_n$ ning indeksid k ja m ($1 \leq k \leq m \leq n$).

Tulemus: element a_m pistetud kohale k massiivis a.

(a) Kirjutada kahendpistemeetodi rekursiivne algoritm, mis kasutab alamalgoritmi $piste()$ ja milles pistekoha otsimine toimub vastava funktsiooni abil.

(b) Kirjutada vastav pistekoha otsimise mitterekursiivne algoritm.

(c) Kirjutada vastav pistekoha otsimise rekursiivne algoritm.

6.18. Sooritada ahela järjestamine valikumeetodil ja pistemeetodil, kui ahel on elementidega

(a) 75, 10, 45, 95, 50, 15, 60;

(b) 62, 63, 67, 68, 65, 60, 59, 77, 69, 70;

(c) 80, 88, 92, 97, 90, 96, 81, 85, 95, 99.

Näidata joonisel töö seis pärast iga „valikut“/„pistet“.

6.19. Sooritada ahela järjestamine kiirmeetodil, kui ahel on elementidega

(a) 50, 10, 21, 36, 32, 20, 85, 78, 45, 80, 26, 31;

(b) 40, 83, 13, 20, 35, 25, 90, 18, 55, 65, 22, 38, 80, 82, 26, 36, 51, 78, 87, 30;

(c) 54, 16, 60, 15, 88, 84, 19, 75, 28, 43, 30, 91, 80, 69, 81, 92, 22, 46, 27, 67.

6.20. Sooritada ahela järjestamine klassikalisel ja optimeeritud põimemeetodil, kui ahel on elementidega

(a) 50, 10, 21, 36, 32, 20, 85, 78, 45, 80, 26, 31;

(b) 40, 83, 13, 20, 35, 25, 90, 18, 55, 65, 22, 38, 80, 82, 26, 36, 51, 78, 87, 30;

(c) 54, 16, 60, 15, 88, 84, 19, 75, 28, 43, 30, 91, 80, 69, 81, 92, 22, 46, 27, 67.

6.21. Olgu antud massiivid

75	10	45	95	50	15	60

62	63	67	68	65	60	59	77	69	70

80	88	92	97	90	96	81	85	95	99

Sooritada iga massiivi järjestamine

(a) mullimeetodil;

(b) vahetustega valikumeetodil;

(c) pistemeetodil.

Näidata joonisel töö seis pärast välimise tsükli sisu iga täitmist.

6.22. Sooritada massiivi

24	28	44	35	40	27	31	46	29	52	20	30

järjestamine kiirmeetodil.

6.23. Olgu antud massiivid

50	10	21	36	32	20	85	78	45	80	26	31

40	83	13	20	35	25	90	18	55	65	22	38	80	82	26	36	51	78	87	30

54	16	60	15	88	84	19	75	28	43	30	91	80	69	81	92	22	46	27	67

(a) Sooritada iga massiivi järjestamine kiirmeetodil;

(b) Sooritada iga massiivi järjestamine klassikalisel ja alt-üles põimemeetodil.

6.24. Millised järgmistest sorteerimismeetoditest

- kiirmeetod (kolmeharuline variant);
- valikumeetod;
- mullimeetod;
- ühildus- ehk põimemeetod;
- pistemeetod

on

(a) parima juhu ajalise keerukusega $\Theta(n)$;

(b) parima juhu ajalise keerukusega $\Theta(n \log n)$;

(c) parima juhu ajalise keerukusega $\Theta(n^2)$;

(d) halvima juhu ajalise keerukusega $\Theta(n \log n)$;

(e) halvima juhu ajalise keerukusega $\Theta(n^2)$?

6.25. Koostada programm, milles on realiseeritud ülesandes 6.24 loetletud sorteerimismeetodid ja iga meetodi rakendamisel konkreetsele massiivile väljastatakse sorteerimise käigus tehtud massiivi elementide (väärtuste) omavaheliste võrdlemiste arv. Leida selle programmi abil meetodites tehtavate võrdlemiste arvud, kui sorteeritavateks massiivideks on juhujärjendid pikkusega

(a) 5;

(b) 12;

(c) 24;

6.26. *(v)* Vaatleme sorteerimise ühildus- ehk põimemeetodit:

```
def põimi(esim, teine):
    uus = []
    i = 0
    j = 0
    while len(uus) < len(esim)+len(teine): #ebaefektiivne
        if j>=len(teine) or i<len(esim) and esim[i]<=teine[j]:
            uus.append(esim[i])
            i += 1
        else:
            uus.append(teine[j])
            j += 1
    return uus

def põimesort(järjend):
    if len(järjend) <= 1:
        return järjend
    keskmine = len(järjend) // 2
    esimene = põimesort(järjend[:keskmine])
    teine = põimesort(järjend[keskmine:])
    uus = põimi(esimene, teine)
    return uus
```

Leida Python-funktsiooni `põimesort` väljakutsete arv 51-elemendilise sisendjärjendi puhul.

6.27. *(v)* Vaatleme sorteerimise kiirmeetodi modifikatsiooni, milles lahkmeks võetakse järjendi esimene element:

```
def kiirsort(järjend):
    if len(järjend) <= 1:
        return järjend
    # luua uus järjend, kus on järjendi esimene element:
    vordne = [järjend[0]]
    vaiksem = []
    suurem = []

    for elem in järjend[1:]:
        if elem < vordne[0]:
            vaiksem.append(elem)
        elif elem == vordne[0]:
            vordne.append(elem)
        else:
            suurem.append(elem)
    return kiirsort(vaiksem) + vordne + kiirsort(suurem)
```

Leida suurim võimalik Python-funktsiooni `kiirsort` väljakutsete arv 90-elemendilise sisendjärjendi puhul.

6.28. Millise järjendi korral on sorteerimise kiirmeetodi, mis valib lahkmeks järjendi esimese elemendi ning jaotab elemendid kolme ossa (väiksemad, võrdsed, suuremad), väljakutsete arv suurim ja millise korral vähim?

6.29. Koostada programm järgmise kolme sorteerimismeetodi tööaegade graafikute kujutamiseks (ühel joonisel):

- kiire sorteerimismeetod (keerukusklassist $\Theta(n \log n)$);
- aeglane sorteerimismeetod (keerukusklassist $\Theta(n^2)$);
- süsteemne sorteerimismeetod, nt Pythoni korral funktsioon `sorted`.

Sorteeritavateks olgu juhujärjendid.

6.30. *(sv)* Meil on arvuti, millega kulub 10 miljoni elemendilise järjendi sorteerimiseks põime- ehk ühildusmeetodiga aega keskmiselt 16000 sekundit. Umbes mitu sekundit võtab aega 10000-elemendilise järjendi sorteerimine sama meetodiga samal arvutil?

6.31. *(v)* Ühel arvutil kulub 10 miljoni elemendilise järjendi sorteerimiseks kiirmeetodil aega keskmiselt 32000 sekundit, teisel arvutil aga sama palju aega 10 miljoni elemendilise järjendi sorteerimiseks pistemeetodil. Umbes mitu sekundit võtab aega

(a) esimesel arvutil 10000-elemendilise järjendi sorteerimine kiirmeetodil;

(b) teisel arvutil 100000-elemendilise järjendi sorteerimine pistemeetodil.

6.32. Juku programmeeris massiivi järjestamise põimemeetodi alt-üles variandi nii, et kahe kõrvutiasetseva osa põimimiste asemel järjestatakse nende kahe osa

poolt kaetav ala valikumeetodil. Kas Juku versioon põimemeetodist on aeglasem või kiirem kui tavaline põimemeetod või pole vahet? Leida Juku järjestamisalgoritmi Θ-hinnang.

6.33. Olgu antud lihtahel, milles paiknevad järjest kirjed võtmetega 44, 25, 27, 16, 32, 50, 37. Sooritada selle ahela järjestamine valikumeetodil. Näidata joonisel seis iga „ettetoomise“ järel.

6.34. Olgu antud lihtahel, milles paiknevad järjest kirjed võtmetega 23, 31, 22, 18, 50, 45, 24, 25, 41, 10, 17, 52, 38, 39. Sooritada selle ahela järjestamine kiirmeetodil.

6.35. Olgu antud lihtahel, milles paiknevad järjest kirjed võtmetega 44, 56, 37, 45, 55, 54, 50, 38, 24, 19, 11, 33. Sooritada selle ahela järjestamine põimemeetodi klassikalise variandi kohaselt.

6.36. Olgu antud lihtahel, milles paiknevad järjest kirjed võtmetega 36, 44, 15, 43, 42, 48, 50, 22, 25, 26, 45, 30, 34, 23, 21, 55. Sooritada selle ahela järjestamine põimemeetodi efektiivseima variandi kohaselt.

6.37. *(s)* Selgitada, millistel juhtudel on alt-üles põimemeetod efektiivsem kui klassikaline põimemeetod.

6.38. Juku kasutab massiivi järjestamiseks sellist kiirmeetodi modifikatsiooni, kus kõik osamassiivid pikkusega kuni 10 elementi järjestatakse valikumeetodiga. Milline on selle meetodi keskmise juhu ajalise keerukuse Θ-hinnang?

6.39. *(s)* On teada, et klassikalise massiivisiseselt töötava järjestamise kiirmeetodi puhul võib funktsiooniväljakutsete magasin ületäituda. Millisel juhul see võib teostuda ja milliste meetmetega saaks sellist olukorda vältida? Eeldame, et väljakutsete magasini ei ole lubatud suurendada.

6.40. Programm realiseerib klassikalise kiirmeetodi variandi, mis jaotamisel võtab alati lahkmeks järjestatava massiiviosa esimese elemendi. Leida 9-elemendiline massiiv, mis sisaldab kõik elemendid $1,\ldots,9$ ja mille puhul selle programmi töö käigus jääb igal jaotamisel suurim element üksi omaette osasse.
Näidata joonisel ka selle massiivi sorteermisel tehtavad jaotamised.

6.41. Millisel juhul töötab massiivi järjestamine pistemeetodil kõige kiiremini?

6.42. *(sv)* Juku kasutab juhuslike elementidega järjendi $[a_1,\ldots,a_n]$ järjestamiseks järgmist algoritmi. Kõigepealt järjestatakse kiirmeetodil ühelemendiline alamjärjend $[a_1]$, seejärel järjestatakse samal meetodil alamjärjend $[a_1,a_2]$, siis alamjärjend $[a_1,a_2,a_3]$ jne kuni lõpuks samuti kiirmeetodil alamjärjend $[a_1,\ldots,a_n]$. Seejuures valitakse lahkmeks alati järjestatava järjendiosa esimene element.

Leida Juku järjestamisalgoritmi ajalise keerukuse Θ-hinnang.

6.43. Valida üks aeglane sorteerimismeetod, mille kohta on teada „hea“ juht – millist tüüpi järjendeid see siiski sorteerib kiiresti. Valida üks kiire sorteerimismeetod, mille kohta on teada „halb“ juht – millist tüüpi järjendeid see siiski sorteerib aeglaselt. Koostada programm, mis rakendab mõlemat meetodit kahel juhul: juhujärjendite korral ja eritüüpi (vastavalt soodsat ja ebasoodsat tüüpi järjendite korral) ning väljastab neid nelja juhtu iseloomustavad graafikud (ühel joonisel).

IV Tõkestamata massiiv

6.44. Lisada algselt tühja järjestatud tõkestamata massiivi järjest kirjed võtmetega 11, 15, 28, 17, 24, 13, 29, 20. Näidata joonisel seis koos tühjade lahtritega pärast iga lisamist. Seejärel eemaldada kirje võtmega 29 ja näidata lõppseis.

6.45. Lisada algselt tühja tõkestamata massiivi lõppu kirjed võtmetega

(a) 25, 36, 11, 15, 29, 20, 26, 19, 17;

(b) 7, 1, 3, 8, 5, 9, 6, 2, 0.

Näidata joonisel alusmassiivi seis koos tühjade lahtritega iga kirje lisamise järel.

Seejärel eemaldada tulemuseks olevaist tõkestamata massiividest kirjed samas järjekorras nagu neid lisati. Näidata joonisel alusmassiivi seis koos tühjade lahtritega iga kirje eemaldamise järel.

6.46. Tõkestamata massiivis on varem mingil hetkel olnud 100 kirjet, misjärel on kõik operatsioonid olnud eemaldamised. Kummast massiivist

45	56	76	11	25		

või

45	56	76	11	25										

on jutt, eeldusel et algselt oli massiiv tühi? Eemaldada sellest massiivist suurima võtmega kirje viiel korral, näidata joonisel seis koos tühjade lahtritega pärast iga eemaldamist.

6.47. Leida tõkestamata massiivi elementide hulka ühe elemendi lisamise protseduuri halvima ja parima juhu keerukuse Θ-hinnang.

7. Paisksalvestus

Kui paisktabeli iga rida mahutab kuni ühe elemendi ja põrke korral paigutatakse teine element kindla eeskirja alusel mujale tabelis, siis nimetatakse tabelit *lahtise adresseerimisega* paisktabeliks.

Kimpudega (kui kimbustruktuuriks on ahel, siis *välisahelatega*) paisktabeli korral on ridadeks kirjete nn kimbud, mis võivad sisaldada 0 või enam kirjet. Kirje võtmega v salvestatakse kimpu real $h(v)$ (sõltumata sellest, kas tegu on või ei ole põrkeolukorraga), kus h on kasutatav paiskfunktsioon. Käesoleva jaotise ülesannetes eeldatakse, et vaikimisi on kimbustruktuuriks järjestamata lihtahel. Samuti eeldatakse, et kimbumeetodis ühe kimbu sorteerimise meetod pole fikseeritud, kasutaja võib selle sobivalt ise valida. Valiku põhjendamisel on muuhulgas vajalik ülejäänute mittesobivuse põhjendamine.

Sorteerimismeetodi valiku ülesannetes eeldatakse sobivaima meetodi valimist järgnevatest:

- kiirmeetod (kaheharuline või kolmeharuline variant);
- valikumeetod;
- mullimeetod;
- ühildus- ehk põimemeetod (tavaline või alt-üles või optimeeritud alt-üles variant);
- pistemeetod (tavaline või kahendpiste);
- kuhjameetod;
- loendamismeetod;
- positsioonimeetod;
- kimbumeetod.

I Lahtise adresseerimisega paisktabel

7.1. *(s)* Olgu antud 1000 reaga lahtise adresseerimisega ja lineaarse kompimisega paisktabel. Millised väärtused hulgast $\{1,\ldots,6\}$ sobivad kompesammuks sellele tabelile?

7.2. Olgu antud 1000 reaga paisktabel kompesammuga 1 ja paiskfunktsiooni h kandidaadid kujul

(a) $h(k) = (k+a) \mod 1000$;

(b) $h(k) = (ak) \mod 1000$,

kus a võib olla 2, 3, 4 või 5. Milline neist neljast väärtusest a kohale on sobivaim?

7.3. *(v)* Miks lahtise adresseerimise lineaarse kompimise meetodis peab kompesamm olema tabeli pikkusega ühistegurita?

7.4. Milline operatsioonidest (otsimine, lisamine, eemaldamine) on lahtise adresseerimisega paisktabeli puhul kõige problemaatilisem?

7.5. Tuua näide 5-realisest lahtise adresseerimisega paisktabelist kompesammuga 1, milles on 4 elementi ja kus elemendi eemaldamisel realiseerub selle operatsiooni halvim juht. Näidata ka eemaldatav element.

7.6. Hinnata paisktabelist kirje eemaldamise algoritmi halvima juhu keerukust kujul $\Theta(f(n))$, kus n on tabeli suurus, kui paisktabel on lahtise adresseerimisega, lineaarse kompimisega ning tabeli rida saab kas olla tühi või sisaldada kirjet (st pole eraldi varianti kustutatud oleku näitamiseks). Kirjeldada lähemalt, millistel tingimustel see halvim juht realiseerub.

7.7. Olgu antud 1000 reaga lahtise adresseerimisega paisktabel, milles paiskfunktsiooniks on jääk ridade arvuga jagamisel ja kompesammuks -1.

1:	22001	1:	22001
120:	52721		
121:	14121	121:	14121
426:	90427	426:	90426
427:	32927	427:	32427
		999:	4999

Kummast kahest ülaltoodud tabelist on jutt (puuduvad read on tabelis vabad), kui loomisel oli tabel tühi ja kõik operatsioonid on sooritatud korrektselt. Lisada õigesse tabelisse järjest kirjed võtmega 48760, 50121 37000, 19427. Näidata joonisel tabeli seis iga lisamise järel.

7.8. Olgu antud 200 reaga lahtise adresseerimisega paisktabel, milles paiskfunktsiooniks on jääk ridade arvuga jagamisel ja kompesammuks -1.

0:	10200		
110:	52711	110:	52710
111:	49111	111:	49111
		112:	60912
113:	34913	113:	34913
199:	93600	199:	93600

Kummast kahest ülaltoodud tabelist on jutt (puuduvad read on tabelis vabad), kui loomisel oli tabel tühi ja kõik operatsioonid on sooritatud korrektselt. Lisada õigesse tabelisse järjest kirjed võtmega 36750, 56513, 20813, 9000. Näidata joonisel tabeli seis iga lisamise järel.

7.9. Koostada sobiv programm, et katsetada kolme erinevat paiskfunktsiooni (h_1, h_2, h_3) kuue andmefaili korral, ning täita katsetulemuste tabel

Nr.	Andmefaili nimi	Kirjete arv	M	h_1	h_2	h_3
1.				p_{11}	p_{12}	p_{13}
2.				p_{21}	p_{22}	p_{23}
3.				p_{31}	p_{32}	p_{33}
4.				p_{41}	p_{42}	p_{43}
5.				p_{51}	p_{52}	p_{53}
6.				p_{61}	p_{62}	p_{63}
		Keskmiselt:		kp_1	kp_2	kp_3

kus M on paisktabeli ridade arv, p_{ij} on j-nda paiskfunktsiooni unikaalsete väärtuste protsent i-nda kirjefaili korral ($i, j = 1,2,3$). Paiskfunktsioonideks olgu:

(a) lihtsaim – jääk võtme (arvkuju) jagamisel arvuga M;

(b) muu, loengust või internetist/kirjandusest;

(c) originaalne, omaloominguline.

7.10. Koostada programm paiskfunktsioonide uurimiseks, mis võimaldab katsetada kolme erinevat paiskfunktsiooni mingi reaalse tekstilise andmefaili korral. Iga paiskfunktsiooni korral leida vähim paisktabeli ridade arv M (täpsusega 100), mille korral vastav paiskfunktsioon jaotab kirjed nii, et ühtegi slotti (paisktabeli ritta) ei siseneks üle kahe kirje. Paiskfunktsioonideks olgu:

(a) lihtsaim – jääk võtme (arvkuju) jagamisel arvuga M;

(b) muu, loengust või internetist/kirjandusest;

(c) originaalne, omaloominguline.

7.11. Valida ja realiseerida üks paiskfunktsioon $h(v,M)$, kus võtmeks v on sõne ning arv M on paisktabeli ridade (slottide) arv.
Koostada ja rakendada seda paiskfunktsiooni sisaldav programm, mis arvutab mingi tekstilise andmefaili korral paiskfunktsiooni väärtused kõigi kirjete korral ning väljastab statistika analoogselt järgmise näitega:

```
Fail: protokoll133km.txt
Kirjete arv: 1970
M = 3743
2 kirjet ühte slotti 312 korda
3 kirjet ühte slotti 45 korda
4 kirjet ühte slotti 5 korda
5 kirjet ühte slotti 1 kord
Unikaalseid h väärtusi: 1186 (60.2%)
```

7.12. Joonistada seitsmerealine paisktabel pärast alljärgnevas tabelis antud kirjete lisamist (tabelis toodud järjekorras) paisktabelisse topeltpaiskamise meetodil ([1], lk 47), kasutades paiskfunktsiooni h ja teisese paiskfunktsiooni h_1 tabelis antud väärtusi.

Nr.	k	$info$	$h(k)$	$h_1(k)$
1.	785	Mart Must	1	6
2.	750	Sulev Sinine	0	1
3.	613	Peeter Punane	3	2
4.	801	Vello Valge	1	5
5.	730	Rein Roosa	3	2

7.13. *(s)* Olgu lahtise adresseerimisega paisktabelis 365 rida. Tabelis hoitakse inimeste kirjeid nii, et iga inimese kirje paiskfunktsiooni väärtuseks on tema sünnikuupäeva päeva järjekorranumber aastas.

Pannes tabelisse x kirjet, siis kui suure tõenäosusega esineb tabelis vähemalt 1 põrge? Leida ka kirjete arvu x selline suurim väärtus, kus see tõenäosus on väiksem kui 0.5.

II Välisahelatega paisktabel

7.14. Mis põhjendusega soovitatakse kimpudega paisktabeli realiseerimisel võimalikult lihtsat kimbustruktuuri – järjestamata ahelat?

7.15. Olgu antud 50-realine välisahelatega jääkpaiskamisega paisktabel.

1:	42351	1:	42401
38:	77838		
39:	61039	39:	77838, 61039
40:	17240	40:	17240
49:	39499	49:	33949

Kummast tabelist on jutt, kui loomisel oli tabel tühi ja kõik operatsioonid on sooritatud korrektselt. Lisada õigesse tabelisse järjest kirjed võtmega 45900, 26138 57889, 47250. Näidata joonisel tabeli seis iga lisamise järel.

7.16. *(s)* Lihtahelad, milles on järjest kirjed võtmetega

(a) 7.9, 9.9, 3.4, 0.9, 8.7, 6.6, 8.6, 3.5, 6.5;

(b) 0.22, 1.5, 0.5, -2.5, -1.16, -1.2, 0.97, -1.15, 0.95, 0.25,

järjestatakse kimbumeetodil. Kimpudeks on lihtahelad, mille järjestamiseks kasutatakse stabiilset meetodit, ning kimbumeetod on realiseeritud ka ise stabiilsena.

Joonistada mõlema juhu jaoks paisktabeli seis iga lisamise järel.

7.17. Olgu antud lihtahel, milles on järjest kirjed võtmetega -1.3, -0.2, -0.5, 3.8, -1.8, 2.9, 7.0, -1.9, 6.2, 0.9, 1.6, -1.8, 2.1, -2.0, 0.7. Sooritada selle ahela järjestamine kimbumeetodil, kus kimpudeks on lihtahelad. Kasutada varianti, mis on stabiilne (eeldusel et kimpude järjestamine on stabiilne). Näidata töö seis vahetult pärast 8 kirje töötlemist ja vahetult pärast kõigi kirjete töötlemist ning lõpptulemus.

7.18. Programmeerida funktsioon etteantud järjendi sorteerimiseks paiskamise kimbumeetodil. Paiskfunktsiooniks võtta ühtlane paiskamine.

Võrrelda selle funktsiooni töökiirust mõne kiirema sorteerimismeetodi (keerukusega $O(n \log n)$) kiirusega sorteeritava järjendi mitmesuguste pikkuste korral. Järjendi elementideks valida nt 1000, ... , 100000 juhuslikku täisarvu poollõigult $[1, 100000)$.

Saadud tulemuste põhjal kujutada ühel joonisel kaks graafikut, kus x-teljel on järjendi pikkus ja y-teljel sorteerimismeetodi tööaeg.

7.19. *(s)* Õppejõud Ahti sorteerib tudengite tööd järgmise kimbumeetodi modifitseeritud variandiga:

- paiskab n tudengi tööd kimpudesse, võttes paiskfunktsiooni väärtuseks tudengi perenime esimese tähe;
- kordab sama protseduuri igas tekkinud kimbus perenime teise tähega;
- järjestab iga eelmisel etapil tekkinud kimbu valikumeetodil;
- ühendab kõik sama perenime esitähega algavad järjestatud kimbud;
- ühendab eelmisel etapil tekkinud järjestatud kimbud.

Leida kirjeldatud sorteerimismeetodi keskmise juhu ja halvima juhu ajalise keerukuse Θ-hinnangud. Võib eeldada, et erinevaid perenimede esimesi ja teisi tähti esineb enam-vähem sama sagedusega.

7.20. Vaatleme algoritmi, mis järjestab n tudengi nimed järgmise kimbumeetodi variandiga:

- ühte kimpu pannakse perenime sama algustähega tudengid;
- kui kõik nimed on kimpudesse pandud, rakendatakse kimpudele kiirmeetodit;
- kõik kimbud algustähe järjestuses lisatakse tulemusjärjendisse, kulutades ühe kimbu lisamiseks konstantse aja tööd.

Leida algoritmi keskmise juhu ajalise keerukuse Θ-hinnang. Eeldada, et erinevaid perenimede algustähti on 28, mis kõik esinevad sama sagedusega.

7.21. *(s)* Juhuslike suuruste normaaljaotuse korral esineb keskmisele lähedasi väärtusi tihedamini ja keskmisest kaugeid väärtusi harva. Olgu vaja järjestada võtmete kasvamise järgi järjend, mille kirjete võtmed on teatud kindlates piirides, kuid jaotunud vastavalt normaaljaotusele (kusjuures keskmine langeb kokku piiride vahelise keskkohaga). Selgitada, mis on kimbumeetodi tavalise realisatsiooni puudus selles olukorras ning kuidas tuleks kimbumeetodit modifitseerida, et muuta see taolises olukorras efektiivsemaks.

7.22. Kas sorteerimise kimbumeetodi realiseerimisel paarikaupa erinevate võtmetega järjendi jaoks võivad kõik elemendid koonduda ühte kimpu?

III Segu

7.23. Olgu antud 100 reaga jääkpaiskamisega paisktabel.

I. Sooritada tegevus, kus tühja tabelisse lisatakse järjest kirjed võtmetega 16185, 77034, 65699, 19285, 37000, 85101, 10934, 92801, 38585 ja seejärel eemaldatakse kirje võtmega 85101, kui paisktabel on

(a) lahtise adresseerimisega, kompesammuga -1;

(b) välisahelatega.

Kujutada joonisel tabeli seis pärast iga lisamist ning lõppseis.

II. Sooritada sama tegevus, kui paisktabel on 50 reaga.

7.24. *(v)* Kas olukorras, kus paisktabelisse salvestatavate kirjete võtmed on täisarvud ja teadaolevalt hõivavad suuremaid järjestikuste täisarvude piirkondi, on mõistlikum ühtlane (lineaarne) paiskamine või jääkpaiskamine?

7.25. Programmeerida järgmised funktsioonid paisktabeli koostamiseks:

(a) Lahtine adresseerimine lineaarse kompimisega. Funktsiooni sisendiks olgu mittenegatiivsete täisarvude järjend. Funktsiooni töö tulemuseks olgu järjend, kuhu etteantud täisarvud on paigutatud lahtise adresseerimise meetodil, kasutades põrgete lahendamiseks lineaarset kompimist.

(b) Kimbumeetod. Funktsiooni sisendiks olgu mittenegatiivsete täisarvude järjend. Funktsiooni töö tulemuseks olgu järjend, mille elementideks on kirjete kimbud. Kimbu võib realiseerida näiteks omaette järjendina, millele uued elemendid lisatakse lõppu.

Paiskfunktsiooniks võib võtta funktsiooni

$$h(k) = \lfloor m(k \cdot T - \lfloor k \cdot T \rfloor) \rfloor,$$

kus k on võti, m tabeli pikkus ja $T = (\sqrt{5}-1)/2$.

Ülesandeks on võrrelda elemendi otsimise efektiivsust kummalgi meetodil realiseeritud paisktabelist. Selleks

- fikseerida paisktabeli pikkus (nt 1000);
- genereerida täisarvujärjendid pikkustega 10%, 20%, ... , 90%, 99% paisktabeli pikkusest, elementideks juhu-täisarvud (nt 1-st 10000-ni);
- paigutada järjendite elemendid paisktabelisse kummalgil meetodil;
- genereerida teatud arv (nt 1000) juhu-täisarve samades piirides nagu järjendi elemendid; iga täisarvu puhul loendada, mitu võrdlemist tehakse selle täisarvu otsimisel kummastki paisktabelist, eristades seejuures edukat ja mitteedukat otsingut.

Tulemused kujutada graafikutena, mis esitavad keskmise võrdlemiste arvu sõltuvust paisktabeli täituvusest, nii eduka otsingu kui ka mitteeduka otsingu korral

kummastki paisktabelist; x-teljel on paisktabeli täituvus (10%, 20%, ... , 90%, 99%)ja y-teljel keskmine võrdlemiste arv.

7.26. Mobiiltelefoni igale klahvile on kinnistatud väike sümbolirühm. Ühe sümboli saab rühmast valida, kui klahvi vajutada teatav arv kordi. Näiteks, kui klahvi sümbolirühmaks on `abc2ä`, siis kaks vajutust sellel klahvil annab `b`, viis vajutust aga `ä`.

Olgu klahvide sümbolirühmad esitatud sõnena `"+0 .,?!-&1 abc2ä def3 ghi4 jkl5 mno6ö pqrs7 tuv8ü wxyz9"`, kus rühmad on eraldatud tühikuga.

Koostada programm, mis leiab iga antud tekstis esineva sõna s korral suuruse $kkv(s)$: mitu klahvivajutust keskmiselt on tarvis sõna s ühe sümboli sisestamiseks. Näiteks, sõna $s =$`jääbki` sisestamiseks tuleb teha $1+5+5+2+2+3 = 18$ klahvivajutust, seega keskmiselt kulub ühe sümboli sisestamiseks $kkv(s) = 18/6 = 3.0$ klahvivajutust. Programmi töö tulemus printida järjendina, milles elemendiks on sõna koos vastava kkv väärtusega. Tulemusjärjend olgu korduvate elementideta ning sorteeritud kkv väärtuste järgi mittekahanevalt. Suur- ja väiketähti ei tule eristada.

Näiteks käesoleva ülesande ülaltoodud teksti korral, kui 'õ' asemel sisestati '6', on tulemuseks (siin osaliselt esitatud): `1.0 ja; 1.0 aga; 1.38 vajutada; 1.4 antud; 1.5 teha;... 1.73 mittekahanevalt; 1.75 elementideta; 1.75 mitu; ...; 2.83 sellel; ...; 3.5 siis; 3.67 töö.`

Programmis tuleb kasutada paisktabeleid <sümbol|vajutusi> ja <sõna|kkv>.

IV Positsioonimeetod

7.27. Lihtahel, milles paiknevad järjest kirjed võtmetega 16185, 77034, 65699, 19285, 37000, 85101, 10934, 92801, 38585, järjestatakse positsioonimeetodiga alusel 100. Kujutada joonisel kõik vahestruktuurid nende valmimise järjekorras ning lõpptulemus.

7.28. Lihtahel, milles paiknevad järjest kirjed võtmetega (kahendsüsteemis) 111111111, 1111010, 1101111, 100111101, 110010011, 110010100, 110010101, 110010010, järjestatakse positsioonimeetodiga alusel 2. Kujutada joonisel kõik vahestruktuurid nende valmimise järjekorras ning lõpptulemus.

7.29. On antud lihtahel, milles paiknevad järjest kirjed võtmetega 1464, 1187, 1031, 1880, 1556, 1908, 1115, 1797, 1200, 1886, 1099, 1551, 1765. Sooritada selle ahela järjestamine positsioonimeetodiga

(a) alusel 10;

(b) alusel 100,

kasutades positsioonipõhiseks järjestamiseks paisktabelit. Kujutada joonisel kõik vahestruktuurid nende valmimise järjekorras ning lõpptulemus.

7.30. On antud lihtahel, milles paiknevad järjest kirjed arvuliste võtmetega FFF_{16}, $1F2_{16}$ $19F_{16}$, $8F9_{16}$, $A23_{16}$, $B28_{16}$, $B29_{16}$ ja $B22_{16}$. Sooritada selle ahela jär-

jestamine positsioonimeetodiga alusel 16. Kujutada joonisel kõik vahestruktuurid nende valmimise järjekorras ning lõpptulemus.

V Sorteerimismeetodi valik

7.31. *(v)* Üliõpilaste arv valdkonnas on $n < 1500$. Üliõpilaste kirjed tuleb sorteerida võtme (1000×(sünniaasta -1900)+ 100×sünnikuu + (sünnipäev kuus)) järgi. Kas selleks on sobivam loendamismeetod või kimbumeetod? Miks?

7.32. *(v)* Ühe aastakäigu kutsealuste arv on $n > 5000$. Kutsealuste kirjed tuleb sorteerida võtme (100×sünnikuu + (sünnipäev kuus)) järgi. Kas selleks on sobivam loendamismeetod või kimbumeetod? Miks?

7.33. *(s)* Järjend koosneb kahte liiki elementidest suvalises järjestuses: pooled on juhuslikud täisarvud lõigult $[1,10]$ ja pooled juhuslikud reaalarvud lõigult $[1,10^9]$. Millised meetodid on eelistatud selle järjendi sorteerimiseks mittekahanevasse järjestusse?

7.34. *(s)* Kirjeldada mistahes etteantud n jaoks järjend pikkusega n selliselt, et selle järjestamiseks pole eelistatud ei kimbu- ega kiirmeetod, vaid mingi muu sorteerimismeetod.

7.35. Kirjeldada mistahes etteantud n jaoks järjend pikkusega n selliselt, et saadud järjenditel töötab nii kiir- kui ka kimbumeetod oma halvima juhu keerukusega.

7.36. *(s)* Juku kirjutas algoritmi, mis järjendi sorteerimiseks tõstab sealt elemente ühekaupa algselt tühja tulemusjärjendi lõppu ja järjestab iga elemendi lisamise järel tulemusjärjendi kohe meetodiga `lemmik_sort`. Millist järjestamismeetodit peaks Juku eelistama meetodi `lemmik_sort` rollis, et tema algoritm oleks võimalikult kiire?

7.37. Olgu antud massiiv Eesti elanike kirjetest (umbes 1.3 miljonit inimest). Sõnastada stabiilne ja võimalikult efektiivne sorteerimismeetod selle massiivi sorteerimiseks sünnipäeva (kujul `päev.kuu.aasta`) järgi.

7.38. Agentuuri WADA käsutuses on massiiv dopinguainete kirjetest, kusjuures kirjed on selles järjestatud aine keelamise aja järgi. Iga dopinguaine kirjes on salvestatud ka selle dopinguaine koostisesse kuuluva olulise toimeaine nimetus. Sõnastada võimalikult efektiivne algoritm dopinguainete massiivi sorteerimiseks leksikograafiliselt toimeaine nimetuse järgi nii, et sama toimeainet sisaldavad kirjed paikneksid tulemusmassiivis aine keelamise aja järgi. Toimeainete arvuks võtta 200. Anda ka algoritmi keskmise juhu ajalise keerukuse Θ-hinnang.

7.39. *(s)* Massiivi a on salvestatud ühe inimese taskus olevate euromüntide väärtused suvalises järjestuses. Näiteks kui tal on üks 2-eurone münt, kaks

10-sendist münti, üks 5-sendine münt ja kolm 2-sendist münti, siis võib a olla {2,5,2,2,10, 200,10}. Millist algoritmi kasutada, et keskmisel juhul võimalikult efektiivselt vääringu järgi järjestada järjend a? Leida selle algoritmi keskmise ajalise keerukuse Θ-hinnang.

7.40. Iga kahe sama vääringuga mündi kaalud on õige pisut erinevad, kahe erineva vääringuga müntide kaalud erinevad rohkem. Panga rahahoidja peab kotitäie juhuslikult valitud müntide kaalud järjendisse paigutama ja suuruse järgi järjestama. Kas efektiivsem on kasutada järjestamismeetodit keerukusega $\Theta(n \log n)$ kogu järjendi peal või sorteerida see kõigepealt mündi vääringu järgi rühmadesse, rakendada seda meetodit igal rühmal eraldi ning ühendada siis rühmad üheks järjendiks?

7.41. Millist ahela järjestamismeetodit eelistada, kui oodatavalt pooltel juhtudel on vaid mõned elemendid „paigast ära“?

7.42. Milline ahela järjestamismeetod on kõige eelistatum, kui võtmete väärtused on reaalarvud kindlates piirides ja seal ühtlaselt jaotunud?

7.43. Kehalise kasvatuse tundi tuli ükshaaval n õpilast. Iga õpilase saabumisel lisas õpetaja ta rivi lõppu ning järjestas siis kõik kohalolevad õpilased kiirmeetodil pikkuse järgi, seejuures esimesel korral pikkuse kahanemise, teisel juhul kasvamise, kolmandal korral jälle kahanemise, neljandal kasvamise järgi jne.

Leida õpetaja tegevusalgoritmi parima, keskmise ja halvima juhu ajalise keerukuse Θ-hinnangud. Eeldada, et kahe õpilase pikkuse võrdlemine ja asukoha vahetamine on elementaaroperatsioonid.

Millised sorteerimismeetodid on sobivamad kogu õpetaja töö sooritamiseks keskmise juhu ajalise keerukuse mõttes õpetaja kasutatava kiirmeetodi asemel?

7.44. Olgu tegemist arvujärjenditega, kus elementideks on parajasti 31-kohalised mittenegatiivsed kahendarvud (väiksemad täiendatud eelnullidega). Kirjutada selliste elementidega järjendi mittekahanevalt sorteerimise algoritm, mille halvima juhu ajaline keerukuse hinnanguks on $\Theta(n)$.

7.45. Raamatukogus asetatakse päeva jooksul tagastatud raamatud tagastusriiulile (tagastamise järjekorras). Töötaja Mari ülesandeks on need päeva lõpul tagastusriiulilt fondiriiulitele viia. Oma töö hõlbustamiseks (jalavaeva vähendamiseks) sorteerib Mari tagastatud raamatud eelnevalt nende inventarinumbrite järjekorras. Mari kasutab (korduvalt) ainult kahe raamatu vahetamise operatsiooni, vahetades kaks raamatut juhul, kui suurema numbriga raamat asub tagastusriiulil väiksema numbriga raamatust kuskil vasakul pool.

Sõnastada algoritm, mis pakub Marile võimalikult vähe vahetusi. Märgime, et kui tagastusriiulil on näiteks raamatud inventarinumbritega 35, 351, 13, 17 ja 213, siis saab need sorteerida kolme vahetusoperatsiooni teel, kui aga tagastusriiulil on raamatud inventarinumbritega 61, 121, 57, 70, 91, 10 ja 101, siis saab need sorteerida nelja vahetamise teel.

VI Paisksalvestuse rakendusi

7.46. *(s)* Sõnastada algoritm, mis suvalisest arvulise võtmega kirjete lihtahelast eemaldab kõik need kirjed, mille võti langeb kokku mõne eespool esineva kirje võtmega. Ülejäänud kirjete omavahelist järjestust muuta ei tohi. Püüda disainida algoritm keskmisel juhul võimalikult efektiivseks, eeldusel et ahela kirjete võtmed jaotuvad ühtlaselt.

7.47. *(s)* Sõnastada algoritm, mis suvalisest arvulise võtmega kirjete lihtahelast leiab seal esinevate unikaalsete (mittekorduvate) võtmeväärtustega kirjete arvu. Püüda disainida algoritm keskmisel juhul võimalikult efektiivseks, eeldusel et ahela kirjete võtmed jaotuvad ühtlaselt.

7.48. *(s)* Olgu ülesandeks leida antud järjendi kaks sellist elementi, mis on suuruse poolest teineteisele kõige lähemal. Sõnastada algoritm selle ülesande lahendamiseks, mis oleks halvimal juhul kõige efektiivsem. Esitada ka selle algoritmi halvima juhu ajalise keerukuse hinnang koos põhjendusega.

7.49. *(s)* Sõnastada võimalikult efektiivne algoritm, mis etteantud n-elemendilise positiivsete paarikaupa erinevate täisarvuliste võtmetega kirjete massiivi korral teeb kindlaks, kas selles massiivis leidub kaks kirjet, mille võtmete summa on n. Anda ka algoritmi keskmise juhu ajalise keerukuse Θ-hinnang.

7.50. *(s)* Täisarvuliste võtmetega kirjete massiivi kohta on teada, et selles on täpselt kaks sama võtmeväärtusega kirjet, kõik ülejäänud on paarikaupa erinevad. Sõnastada võimalikult efektiivne algoritm, mis leiab massiivist korduva võtme väärtuse. Võib eeldada, et võtmed on ühtlaselt jaotunud lubatud väärtusvarus. Anda ka algoritmi keskmise juhu ajalise keerukuse Θ-hinnang.

8. Magasin ja järjekord

Käesolevas jaotises eeldatakse, et magasiniga ja järjekorraga manipuleerimisel puudub otseligipääs nende kirjetele. Lubatud on vaid andmestruktuuri definitsiooni järgivad lisamise ja eemaldamise operatsioonid ning üle- ja alatäitumise kontrollid.

8.1. Koostada programm, mis magasinioperatsioone kasutades teeb kindlaks, kas sisendiks antud sõne on palindroom.

8.2. Sooritada avaldise

(a) `((2*7)+5)*(5-6)`

(b) `2+((((4+6)*8)-(7*5))-3)`

viimine pööratud poola kujule. Näidata joonisel töö algseis ja seis iga sümboli töötlemise järel. Seisus kujutada tehtemagasini olek, valmis tulemuse osa ja töötlemata sõneosa.

8.3. Sooritada pööratud poola kujul avaldise (kus operandideks on kümnendnumbrid)

(a) `27*5+56-*`

(b) `27*5+56-*`

(c) `5398*72+-*+`

(d) `12+34**5678+++-`

(e) `92-7*34++108*-+`

väärtuse arvutamine magasini kasutades. Näidata joonisel töö seis iga sümboli töötlemise järel. Seisuks lugeda magasini olek ja avaldise veel töötlemata osa.

8.4. Sooritada avaldise

(a) `5*2+7+3*1`

(b) `5+9*3*6-2*4`

(c) `(2*7+5)*(5-6)`

(d) `2+((4+6)*8-7*5-3)`

väärtuse arvutamine magasini kasutades. Näidata joonisel töö algseis, seis iga sümboli töötlemise järel ning lõppvastus. Seisuks lugeda magasini olek ja avaldise veel töötlemata osa.

8.5. *(v)* Olgu pööratud poola kujul avaldises n arvu ja m tehtemärki. Leida selle avaldise

(a) väärtuse leidmisel tehtavate elementaaroperatsioonide koguarv;

(b) väärtuse leidmise algoritmi ajalise keerukuse hinnang $\Theta(f(n,m))$.

Elementaaroperatsioonideks lugeda aritmeetilised tehted ja magasinioperatsioonid.

8.6. Programmeerida funktsioon, mis (magasine kasutades) teisendab sõnena etteantud infikskujul aritmeetilise avaldise vastavaks postfikskujul avaldiseks. Võib eeldada, et avaldises esinevad ainult binaarsed tehted ning infikskuju ja kõik tehete operandid peale arvude ning muutujate on sulgudesse võetud. Näiteks: $(a+(b\hat{}2))$.

8.7. Olgu meil 36-lehelist kaardipakki kujutav sõnede järjend:

```
"ruutu 6", "ruutu 7", ..., "ruutu äss",
"ärtu 6",  "ärtu 7",  ..., "ärtu äss",
"pada 6",  "pada 7",  ..., "pada äss",
"risti 6", "risti 7", ..., "risti äss".
```

- Koostada programm, mis kasutab järjekordi ja/või magasine paki ümberkorraldamiseks nii, et tulemus oleks järgmine:

```
"ruutu 6",   "ärtu 6",   "pada 6",   "risti 6",
"ruutu 7",   "ärtu 7",   "pada 7",   "risti 7",
...          ...         ...         ...
"ruutu äss", "ärtu äss", "pada äss", "risti äss".
```

- Lisada funktsioon sama operatsiooni antud arv kordi sooritamiseks.
- Rakendades eelmist funktsiooni, segada kaardipakki juhuslik arv kordi. Väljastada tulemus.
- Leida segamiste arv, mille tulemusena saadakse esialgne kaartide järjestus.

8.8. *(s)* Koostada programm, mis järjekorda kasutades läbib failisüsteemi, alates etteantud kataloogist, ja väljastab ekraanile kõik alamkataloogide ning failide nimed. Nõutav on, et algul väljastab programm kõik kataloogid ja failid, mis on „sügavusel“ 1, siis kõik sellised, mis on „sügavusel“ 2, jne.

Faili- ja kataloginimed väljastada sügavusele vastava taandega.

8.9. *(sv)* Koostada mitterekursiivne programm, mis magasini- ja/või järjekorraoperatsioone kasutades leiab n lipu erinevate paigutuste arvu $n \times n$ malelauale nii, et ükski lipp ei oleks ühegi teise lipu tules.

8.10. *(s)* Juveelipoe omanik korraldab aegajalt reklaamiürituse, kus ühele külastajale loositakse tasuta võimalus omandada üks ehe vaateaknal olevast ehetereast. Selline premeerimine toimub vahetult enne poe sulgemist, mil ehted vaateaknalt ära viiakse. Loosiga välja valitud isik peab ütlema ühe arvu k ($0 < k <$ ehete arv reas). Seejärel hakatakse ehteid aknalt eemaldama: vasakult paremale liikudes eemaldatakse iga k-s ese, rea lõppu jõudes jätkub loendamine rea algusest. Isik saab omale ehte, mis viimasena jääb vaateaknale. Selleks et loosiõnne

naeratamise puhuks olla valmis ütlema sellist arvu k, et saada kõige hinnalisem ehe, tuleb koostada (ja oma nutitelefoni laadida) programm, mis, saades ehete hindade järjendi, väljastab soovitud k.

8.11. Koostada programm järgmise ülesande lahendamiseks.

Antud: infiks-kujul aritmeetiline avaldis (sõnena), näiteks

$(((a+b)*(c-d*k)+2.5)/(pi*(1-e)))$

Tehete järjekorra määramiseks kasutatakse ümarsulge ja ka kogu avaldis on võetud sulgudesse.

Tulemus: väljastatakse arvutuseeskiri, mis näitab millises järjekorras tuleb (võiks) arvutada selles esinevate suluavaldiste (st sulgudega ümbritsetud) avaldiste väärtused; näiteks ülaltoodud avaldise korral väljastatakse eeskiri

```
Arvutada:
S1 := a+b
S2 := c-d*k
S3 := S1*S2+2.5
S4 := 1-e
S5 := pi*S4
S6 := S3/S5
```

Programm peab kasutama magasini.

Lisand: näha ette kasutaja teavitamine (süntaksi veast) sulgude ebakorrektse paiknemise puhul sisendina antud avaldises.

8.12. *(s)* Järgmises tabelis on antud linnadevahelised kaugused:

	Elva	Haapsalu	Kuressaare	Narva	Pärnu	Rakvere	Tallinn	Tartu	Valga	Viljandi	Võru
Elva	0	267	308	211	156	152	216	27	60	70	75
Haapsalu	267	0	155	314	111	203	101	258	254	199	310
Kuressaare	308	155	0	429	152	313	216	330	295	249	351
Narva	211	314	429	0	299	116	212	184	271	265	252
Pärnu	156	111	152	299	0	183	129	178	143	97	199
Rakvere	152	203	315	116	183	0	99	126	212	151	193
Tallinn	216	101	216	212	129	99	0	189	252	161	257
Tartu	27	258	330	184	178	126	189	0	87	73	68
Valga	60	254	295	271	143	212	252	87	0	91	71
Viljandi	70	199	249	265	97	151	161	73	91	0	128
Võru	75	310	351	252	199	193	257	68	71	128	0

Meil on ökonoomsusauto, millega saab ühe tankimisega sõita x kilomeetrit. Olgu nii, et sellele sõidukile sobivad tanklad asuvad vaid tabelis loetletud linnades. Koostada programm, mis etteantud piirangu x korral väljastab ekraanile tabeli M, mille element M_{ij} väljendab vähimat tankimiskordade arvu, millega saab sõita linnast i linna j. Kasutada tuleb järjekorda või magasini.

Leida ka piirangu x vähim väärtus, mille korral saaks liikuda iga kahe linna vahel.

8.13. *(s)* Olgu magasinide paari M_1 ja M_2 korral on lubatud järgmised operatsioonid

1) võtta element magasinist M_1 ja panna see magasini M_2;
2) eemaldada element magasinist M_2.

Programmeerida funktsioon, mis kahe sisendjärjendi a ja b korral kontrollib, kas ülaltoodud operatsioonide mingis järjestuses sooritamise teel on võimalik saavutada magasini M_2 seisuks järjend b, kui algselt on magasin M_2 tühi ja magasini M_1 seisuks on järjend a.

8.14. Lahendada ülesanne 8.13, kui magasinide paari asemel on järjekordade paar.

9. Puu ja kahendpuu

I Puu struktuur

Aine raames käsitletakse puud põhiliselt kui juurega puud. Puu P (või kahendpuu T) juurtippu tähistame $P.juur$ ($T.juur$). Kahendpuu igal tipul on maksimaalselt 2 alluvat: vasak ja parem alluv. Eeldame, et kahendpuu igas tipus t on määratud viidad alluvatele, vastavalt väljad $t.vasak$ ja $t.parem$; kui alluv puudub, siis on viidaks tühiviit, mida tähistab kas Λ (algoritmis) või `null` (Javas) või `None` (Pythonis).

Puu ja kahendpuu korral kasutame järgmisi mõisteid.

Tipu *tasemenumbriks* ehk *sügavuseks* puus (kahendpuus) loeme sammude arvu (ülemuselt alluvale), mis on vajalik selle tipuni jõudmiseks juurtipust lähtudes. Puu juurtipu sügavuseks on seega 0 (juurtipp asub tasemel 0), juurtipu vahetute alluvate sügavuseks on 1 jne. Puu *tasemeks* nimetame ühe ja sama tasemenumbriga tippude alamhulka. Puu *kõrguseks* on tema tasemete arv. Tühja puu kõrgus on seega 0.

Ka siinse jaotise ülesannete lahendustes ei või kasutada funktsioonivälise id muutujaid. Vaikimisi eeldatakse, et puu tippudel puudub viit oma ülemusele. Lisaviit ülemusele määrata vaid siis, kui teisiti ei ole võimalik. Küll aga võib arvestada, et puu iga tipuga t on seotud nn *tööväli* $t.x$, mida saab kasutada puu töötlemise käigus tipuga seotud info talletamiseks ja kasutamiseks. Puu puhul eeldame, et tipu t alluvate läbivaatamise võib kirjeldada tsüklina `[* Tipu` t `iga alluva` v `korral: ...]`.

9.1. *(s)* Programmeerida funktsioon, mis leiab antud kahendpuu viimasel (sügavaimal) tasemel olevate tippude arvu

(a) rekursiivsel meetodil, kahendpuud ühekordselt läbides;

(b) iteratiivsel meetodil.

9.2. Kirjutada algoritm antud puule vastava kahendpuu ([1], lk 28) konstrueerimiseks.

9.3. *(v)* Kirjutada algoritm puu iga tipu tööväljale selle tipu vahetute alluvate arvu salvestamiseks, kui on antud

(a) kahendpuu;

(b) puu;

(c) puu vastava kahendpuu kujul.

9.4. *(v)* Sõnastada järgmise ülesande lahendamise algoritm.
Antud: kahendpuu T ja selle tipp t.
Tulemus: tipust t algava (alam)kahendpuu kõrgus.

9.5. Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: kahendpuu ja selle tipp.
Tulemus: antud tipu kõigi (vahetute ja kaugemate) järglaste koguarv.

9.6. *(s)* Alljärgneva loetletuga on esitatud rida lihtsamaid puutöötluse ülesandeid, milles igaühes nõutakse antud puu iga tipu tööväljale omistada sellest tipust algava alampuu

1) tippude arv;
2) lehtede arv;
3) vahetippude arv;
4) kõrgus;
5) alampuude kõrguste aritmeetiline keskmine;
6) juurtipu tasemenumber antud puus.

Kirjutada algoritm iga ülalloetletud ülesande (1 – 6) jaoks, kui antud on

(a) kahendpuu;
(b) puu;
(c) puu vastava kahendpuu kujul.

9.7. *(s)* Programmeerida funktsioon, mis etteantud kahendpuu tippudest moodustab järjendi, milles tipud paiknevad kahendpuu tasemete kaupa.

9.8. Kirjutada algoritm väljastamaks antud kahendpuu lehttipud tasemete kaupa, taseme sügavuse

(a) kasvamise järjekorras;
(b) kahanemise järjekorras.

9.9. Programmeerida generaator-funktsioon järgmise ülesande lahendamiseks.
Antud: kahendpuu.
Tulemus: antakse välja järjekordse taseme tipud (listina).

9.10. *(s)* Kirjutada algoritm, mis kontrollib, kas etteantud kahendpuu on kompaktne ([1], lk 28).

9.11. Mitu tippu ja mitu lehte on täielikus kahendpuus, mille viimase taseme number on h? Tõestada need valemid.

9.12. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: loodava kahendpuu tasemete arv m.
Tulemus: täielik m-tasemeline kahendpuu.

9.13. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: loodava kahendpuu tippude arv n.
Tulemus: kompaktne n-tipuline kahendpuu.

9.14. Koostada programm järgmise ülesande lahendamiseks.
Antud: kahendpuu.
Tulemus: nende tippude järjend, mille alampuude kõrgus erineb rohkem kui 1 võrra.

9.15. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: kahendpuu.
Tulemus: uus kahendpuu – antud kahendpuu peegeldus üle juurt läbiva vertikaaltelje (tavajoonise mõttes).

9.16. Sõnastada algoritm järgmise ülesande lahendamiseks.
Antud: puu ja selle kaks tippu.
Tulemus: antud tippude lähim ühine eellane antud puus.

9.17. Inimeste põlvnemise andmestikku on salvestatud iga inimese kohta puukujulise infona n eelmise põlvkonna kõik isikud (ema, isa, emaema, emaisa, isaema, isaisa, emaemaema jne). Kahe inimese, A ja B, suguluse kindlakstegemise algoritmiga kontrollitakse, kas mingi A esivanem esineb ka B esivanemate hulgas. Leida sellise algoritmi halvima juhu ajalise keerukuse Θ-hinnang. Võib eeldada, et põlvnemise puu igal tipul on kas 0 või 2 alluvat.

9.18. *(s)* Programmeerida juhu-kahendpuu loomise funktsioon.
Antud: loodava puu tippude arv n.
Tulemus: juhuslikul moel genereeritud n-tipuline kahendpuu.

II Puu läbimisviisid

9.19. Kuidas selgitada asjaolu, et kahendpuudel on kolm standardset läbimisviisi, kuid üldiselt puudel vaid kaks?

9.20. Nummerdada joonisel 3 kujutatud kahendpuude tipud

(a) eesjärjestuses;

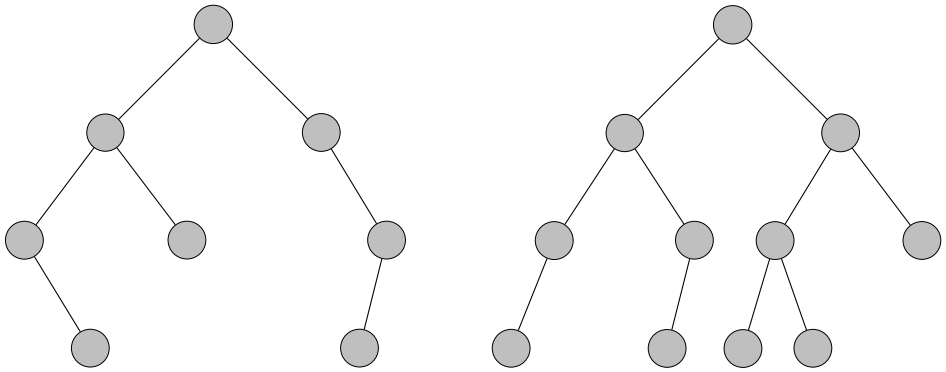
(b) keskjärjestuses;

(c) lõppjärjestuses.

9.21. Joonistada mingi 12-tipuline puu. Kirjutada see puu üles nii vasaku kui ka parema suluesitusena ([1], lk 43) ning koostada sellele puule vastava kahendpuu tippude loetelud ees-, kesk- ja lõppjärjestuses.

9.22. Kirjeldada võimalikult täpselt kõik kahendpuud, mida läbides nii kesk- kui ka eesjärjestuses, saadakse samad tippude järjestused.

9.23. Programmeerida funktsioon, mis kirjutab selle puu tippude tööväljadele tippude numbrid keskjärjestuses (st välja $t.x$ väärtuseks saab i, kui tipp t on kogu kahendpuu tippude keskjärjestuses i-ndal kohal).



Joonis 3: Kaks kahendpuud.

9.24. Joonistada võimalikult madal 12-tipuline kahendpuu ja nummerdada selle tipud lõppjärjestuses.

9.25. Joonistada võimalikult madal kahendpuu, mille tippudeks on sümbolid Teie ees- ja perekonnanimest ning mille tippude lõppjärjestus annaks Teie ees- ja perekonnanime.

9.26. Kirjutada algoritm, mille sisenditeks on kahendpuu ja tema tipp ning väljundiks selles puus antud tipule

(a) eesjärjestuses;

(b) keskjärjestuses;

(c) lõppjärjestuses

järgnev tipp (tühiviit, kui järgmist pole).

9.27. Programmeerida rekursiivne funktsioon, mille sisenditeks on kahendpuu ja selle tipp t ning tulemuseks tipust t algava (alam)kahendpuu tippude järjend

(a) eesjärjestuses;

(b) keskjärjestuses;

(c) lõppjärjestuses.

9.28. *(s)* Sõnastada mitterekursiivne algoritm kahendpuu läbimiseks keskjärjestuses.

9.29. *(s)* Kahendpuu-kujulise struktuuriga trassil tehakse järjestikku katseid külastada lõpp- ehk lehttippe, liikudes ülemuselt alluvale, iga katset juurtipust alustades. Kahendpuu juurtipus ja igas vahetipus paikneb kahevalentne foor, milles põleb kas roheline või punane tuli. Roheline tähendab, et on lubatud liikuda ainult vasakule alluvale; punane foorituli lubab liikuda ainult paremale alluvale. Tipu läbimisel muutub foori tuli vastupidiseks: rohelisest punaseks või punasest

roheliseks. Katse loetakse sooritatuks, kui jõutakse lehttippu või tippu, millel puudub see alluv, kuhu foor lubab liikuda; viimasel juhul foor ikkagi muudab värvi. Enne esimest katset on foorid lülitatud juhuslikul moel kas roheliseks või punaseks.

(a) Kirjutada algoritm leidmaks, mitu katset läheb tarvis, et kokkuvõttes oleks külastatud etteantud trassi iga lõpptipp.

(b) Tõestada, et trassi mistahes lõpptipp saab külastatud lõpliku arvu katsete käigus.

III Aritmeetilise avaldise puu

Märgendatud graafi (erijuhul puu või kahendpuu) igal tipul t leidub selle märgendit sisaldav lisaväli $t.m$.

Käesolevas piirdume selliste aritmeetiliste avaldistega, milles esinevad ainult binaarsed tehted ja operandideks on arvud. Niisugune aritmeetiline avaldis esitatakse kahendpuuna (aritmeetilise avaldise puuna), kus lehttipul on märgendiks arv ja juurel ning vahetipul – tehtemärk. Nii juurel kui ka igal vahetipul on parajasti kaks alluvat.

9.30. *(s)* Programmeerida funktsioon juhusliku aritmeetilise avaldise puu loomiseks.

Antud: loodava kahendpuu tippude maksimaalarv n.

Tulemus: juhuslikul viisil ehitatud vähemalt $n/2$-tipuline aritmeetilise avaldise puu; lehttipu märgendiks on juhuarv, ülejäänutel – binaarse tehte märk, juhuvalikuna märkide $+ - * /$ seast.

9.31. *(s)* Programmeerida

(a) funktsioon, mis leiab antud aritmeetilise avaldise puule vastava aritmeetilise avaldise väärtuse;

(b) funktsioon, mis leiab antud aritmeetilise avaldise puule vastava infikskujul aritmeetilise avaldise, kus iga binaarse tehte ümber on sulud, nt (3*2)+(5+3);

(c) funktsioon, mis leiab antud aritmeetilise avaldise puule vastava infikskujul aritmeetilise avaldise, kus tehete prioriteeti arvestades on mittevajalikud sulud ära jäetud, nt 3*2+5+3.

9.32. *(s)* Programmeerida funktsioon aritmeetilise avaldise puule vastava suluavaldise leidmiseks.

Antud: aritmeetilise avaldise puu T ja selle tipp t.

Tulemus: suluavaldis, mis vastab tipust t algava T alamkahendpuu poolt kujutatavale (osa)avaldisele.

9.33. Aritmeetilise avaldise väärtuse leidmiseks töödeldakse tema struktuurile vastavat kahendpuud kõrgusega h lõppjärjestuses. Hinnata:

(a) parima juhu ajalist keerukust kujul $\Theta(f(h))$;

(b) halvima juhu ajalist keerukust kujul $\Theta(f(h))$;

(c) keskmise juhu ajalist keerukust kujul $\Theta(f(h))$ (keskmise määramisel arvestada, et iga võimalik puu tippude arv esineb sama sagedusega).

Kõik hinnangud esitada võimalikult lihtsa funktsiooniga f ja põhjendada. Aritmeetilise tehte sooritamine lugeda elementaaroperatsiooniks.

9.34. *(s)* Kahendpuud, mille juurtipul ja igal vahetipul on parajasti kaks alluvat, nimetame *täis-kahendpuuks*. Sellise struktuuriga on ka käesolevas vaadeldav aritmeetilise avaldise puu. Kahelehelisi täis-kahendpuid on üks, kolmelehelisi – kaks.

Programmeerida funktsioon, mis argumendile n vastavalt leiab erinevate n-leheliste täis-kahendpuude arvu.
Näiteks $n = 4$ korral on tulemuseks 5, $n = 8$ korral on tulemuseks 429.

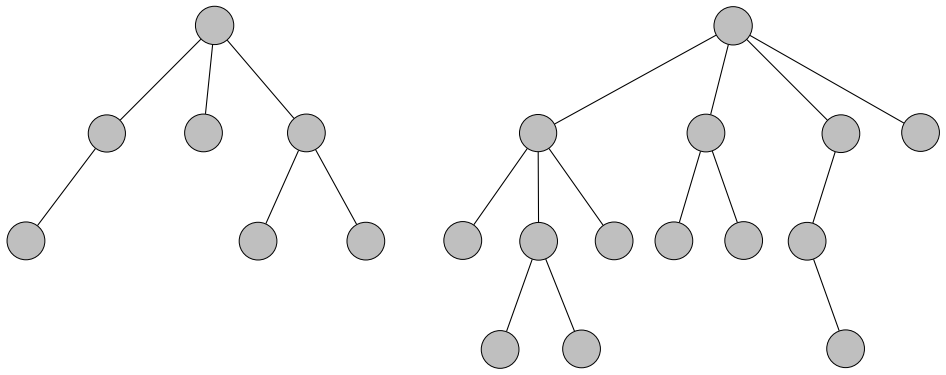
IV Mets

9.35. *(s)* Mitu puud on mets?

9.36. Nummerdada joonisel 4 kujutatud metsa tipud

(a) eesjärjestuses;

(b) lõppjärjestuses.



Joonis 4: Kahepuuline mets.

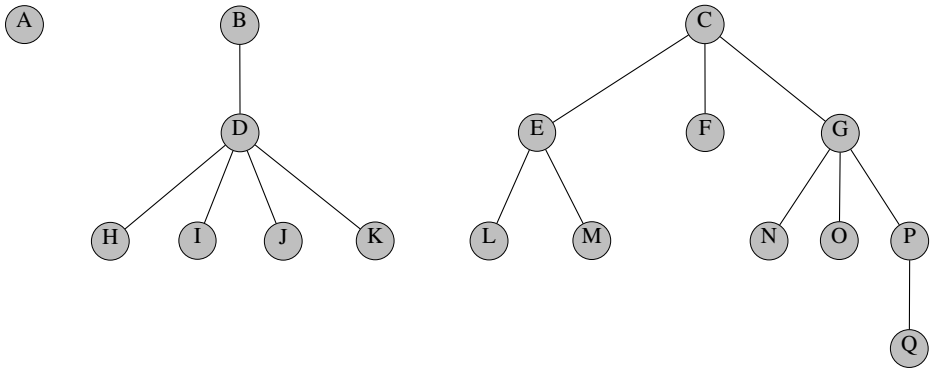
9.37. Kirjutada algoritm, mis antud kahendpuu korral konstrueerib vastava tavapuude metsa.

9.38. Kirjutada algoritm, mis antud metsa korral konstrueerib vastava kahendpuu.

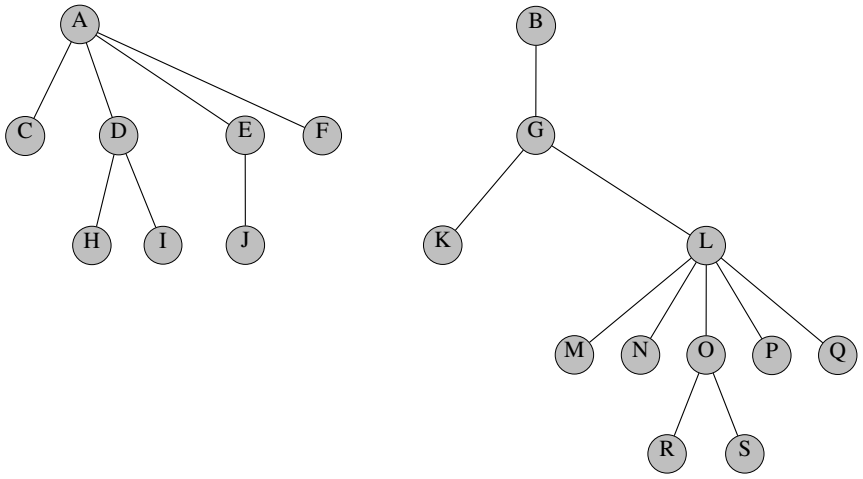
9.39. Esitada kahendpuu kujul

(a) joonisel 5 kujutatud mets;

(b) joonisel 6 kujutatud mets.



Joonis 5: Kolmepuuline mets.



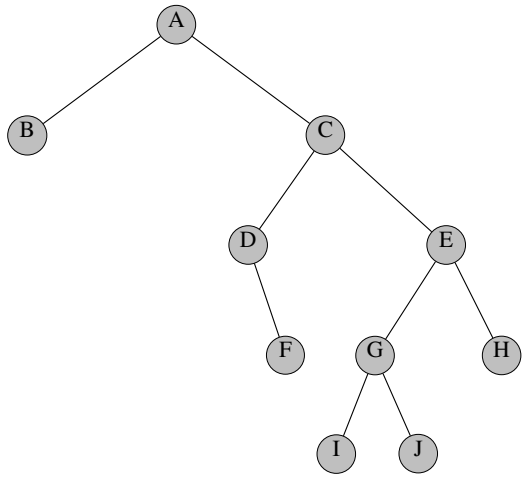
Joonis 6: Mets.

9.40. Milline mets esitub

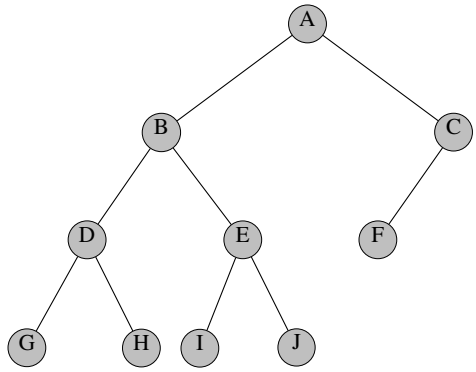
(a) joonisel 7 kujutatud kahendpuuna?

(b) joonisel 8 kujutatud kahendpuuna?

9.41. Tõestada, et kahendpuude ja metsade vahel on võimalik üksühene vastavus.



Joonis 7: Mets kahendpuuna.



Joonis 8: Mets kahendpuuna.

V Pakkimine

9.42. *(s)* Antud teksti

(a) `aajbcadefbabgihiiafh`;

(b) `vanapaganavanaema`

korral

- joonistada vastav Huffmani puu;
- koostada antud tekstis esinevate sümbolite Huffmani koodide tabel;
- kirjutada antud teksti Huffmani kodeering (bitijärjendina).

9.43. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: tekst *s* (sõnena).

Tulemus: Huffmani algoritmiga leitud koodipuu teksti *s* jaoks.

9.44. Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: koodipuu *kdp*, selle tipp *t* ja 0-1-rada (bitijärjend) *kdp* juurtipust tipuni *t*.

Tulemus: tipust *t* algava alampuu iga lehttipu *u* korral väljastatud 0-1-rada *kdp* juurtipust tipuni *u*.

9.45. Programmeerida funktsioon järgmise ülesande lahendamiseks.

Antud: tekst *s*.

Tulemus: Huffmani algoritmiga leitud koodipuu teksti *s* jaoks ja selle põhjal prefiks-kodeeritud *s*.

Võrrelda efektiivsust (pakkimata teksti ja pakitud teksti pikkuste suhet) mõne enamkasutatavama pakkimisprogrammiga.

9.46. Programmeerida funktsioon järgmise ülesande lahendamiseks.

Antud: prefiks-kodeeritud tekst *s* ja kodeerimisel kasutatud koodipuu.

Tulemus: teksti *s* dekodeerimisel saadud sõne.

9.47. Koostada prefiks-kodeerimisel põhinev „krüpto-programm“, mis saab kasutajalt kolme faili nimed ja tegevuse tunnuse. Failideks on *f_in*, *f_out*, *f_kdp*. Kui tegevuseks on *pakkida*, siis failis *f_in* olev tekst prefiks-kodeeritakse, tulemus pakitakse faili *f_out* ja kodeerimisel kasutatud koodipuu salvestatakse faili *f_kdp*. Kui tegevuseks on *lahtipakkida*, siis failist *f_in* saadav prefikskood dekodeeritakse tekstiks faili *f_out*, kasutades failis *f_kdp* olevat koodipuud.

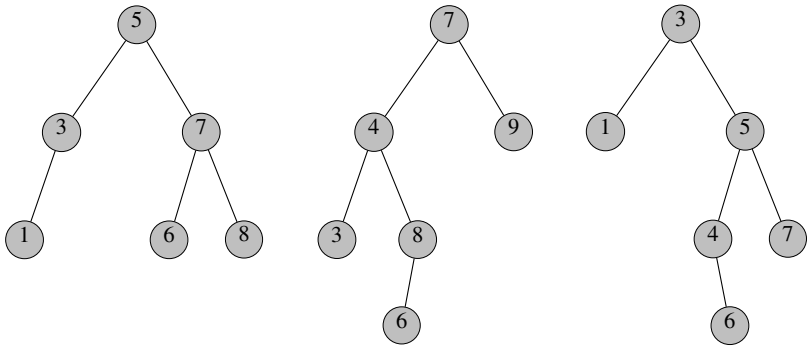
10. Otsimispuud

I Kahendotsimise puu

Kahendotsimise puu ([1], lk 28-32) ehk kahendotsimispuu igas tipus paikneb üks (andme)kirje. Etteantud võtmega kirje otsimine toimub tippudeks olevate kirjete seast nende võtmete järgi. Tipus t oleva kirje võtit tähistame $t.võti$. Joonisel näidatakse kahendotsimise puu igas tipus ainult vastava kirje võtmeväärtus. Kahendotsimise puu on kirjete hulka kujutav andmestruktuur, mille operatsioonivarusse kuuluvad nii kirje otsimise kui ka lisamise ja eemaldamise operatsioonid.

10.1. Millised joonisel 9 kujutatud puudest on kahendotsimispuud?

10.2. Miks ei saa sätestada, et kahendotsimise puu peab alati olema täielik kahendpuu ([1], lk 28)?



Joonis 9: Kolm kahendpuud.

10.3. *(s)* Programmeerida funktsioon kontrollimaks, kas antud kahendpuu on kahendotsimise puu.

10.4. *(s)* Joonistada kõik korduvate võtmeteta kahendotsimispuud, milles on parajasti kirjed võtmetega

(a) 1, 2, 3;

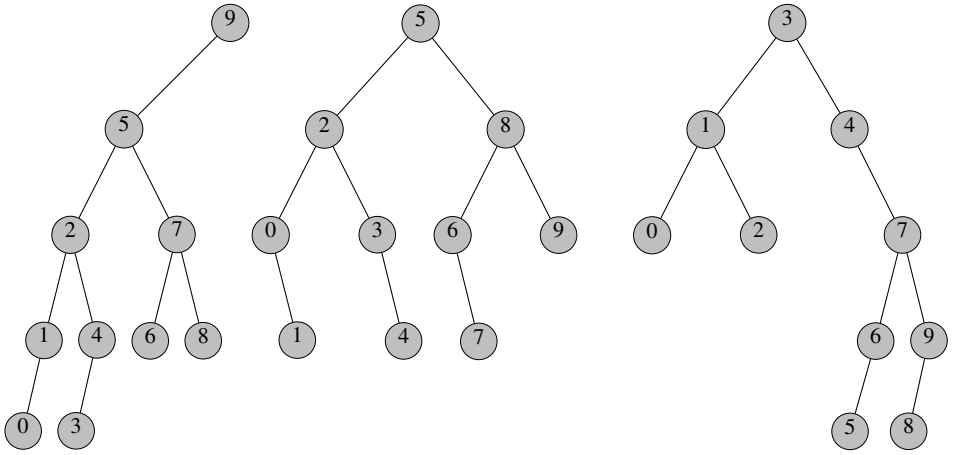
(b) 1, 2, 3, 4;

(c) 1, 2, 3, 4, 5.

10.5. Konstrueerida võimalikult madal kahendpuu, mille tippudeks on sümbolid Teie ees- ja perekonnanimest ning mille tippude keskjärjestus annaks Teie ees- ja perekonnanime. Asendada selles kahendpuus tähed paarikaupa erinevate arvuliste võtmetega nii, et moodustuks kahendotsimise puu nende võtmete järgi. Joonistada kahendotsimise puu, mis on saadud sellest puust juurtipu eemaldamisel.

10.6. *(s)* Programmeerida funktsioon, mis etteantud kahendotsimispuus T asendab selle puu tippudes olnud võtmed erinevate võtmetega $1,\ldots,n$, kus n on puu T tippude arv, nii et puu jääb kahendotsimispuuks.

10.7. *(s)* Eemaldada joonisel 10 kujutatud kahendotsimispuudest juure kirje, joonistada iga puu seis pärast seda operatsiooni.



Joonis 10: Kolm kahendotsimispuud.

10.8. Eemaldada joonisel 10 kujutatud kahendotsimispuudest kirjed näidatud järjekorras:

(a) esimesest puust kirjed võtmetega 8, 7, 2, 1, 3, 0, 9, 5, 6, 4;

(b) teisest puust kirjed võtmetega 7, 0, 8, 5, 2, 3, 4, 1, 9, 6;

(c) kolmandast puust kirjed võtmetega 8, 4, 1, 3, 2, 5, 7, 6, 9, 0.

Joonistada välja puu seis pärast iga eemaldamist.

10.9. Lisada algselt tühja kahendotsimispuusse järjest kirjed võtmetega

(a) 40, 35, 36, 60, 44, 41, 55, 44, 43, 45;

(b) 95, 10, 40, 10, 40, 35, 80, 75, 96, 30.

Eemaldada tulemuseks olevast kahendotsimispuust vähima võtmega kirje; joonistada puu seis pärast seda operatsiooni. Seejärel eemaldada tulemuspuust omakorda vähima võtmega kirje; joonistada puu seis selle operatsiooni järel.

10.10. Programmeerida funktsioon järgmise ülesande lahendamiseks.

Antud: kahendotsimispuu *kop* (võtmed unikaalsed) ja lisatav kirje võtmega *k*.

Tulemus: tipp võtmega *k* lisatud antud kahendotsimispuusse *kop*; kui võti *k* juba leidus, siis ei tehta midagi.

10.11. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: *kop* - kahendotsimispuu, *t* - selle tipp, *y* - tipu *t* ülemus (Λ, kui *t* on *kop* juurtipp).
Tulemus: tipp *t* eemaldatud kahendotsimispuust *kop*.

10.12. *(v)* Järjendit sorteeritakse järgmise stabiilse meetodi kohaselt: alguses paigutatakse järjendi elemendid kahendotsimispuusse, läbides järjendit päripidi, ja seejärel, läbides puu keskjärjestuses, leitakse tulemus. Paigutades kahendpuusse järjendi elementi *x* osutus, et selle võtme väärtus võrdub kahendpuu juurkirje võtme väärtusega. Kummasse puu harusse paigutatakse *x*?

10.13. Kasutades vahestruktuurina kahendotsimispuud, sorteerida ahel, milles on järjest kirjed võtmetega

(a) 55, 70, 80, 63, 24, 33, 72, 15, 25, 51, 47, 12, 60, 40, 70;

(b) 40, 88, 50, 15, 27, 25, 55, 56, 60, 75, 24, 12, 44, 90, 89, 37, 57, 81, 18, 36;

(c) 46, 20, 35, 26, 18, 71, 63, 50, 25, 33, 87, 11, 25, 80, 59, 40, 52, 11, 76, 55,

rakendades meetodi stabiilset varianti. Joonistada kahendotsimispuu lõppseis ning sorteeritud ahel.

10.14. *(sv)* Kuidas programmeerida järjestamine kahendotsimispuu abil nii, et oleks tagatud halvima juhu ajaline keerukus $O(n \log n)$, kus n on järjendi kirjete arv.

10.15. *(s)* Sõnastada algoritm, mille sisendiks on kaks kahendotsimispuud ja väljundiks üks kahendotsimispuu, mis sisaldab kõik sisendpuude kirjed. Algoritm peab töötama lineaarse ajalise keerukusega tippude koguarvu suhtes.

10.16. Kirjutada algoritm kahendotsimispuust suuruse poolest teise võtmega elemendi leidmiseks (puud muutmata).

10.17. Kahendotsimispuu juure kirje eemaldamise operatsiooni võib kirjeldada vastastikku rekursiivsena vähima võtmega kirje eemaldamise operatsiooniga; mitu rekursiivset pöördumist juure kirje eemaldamise operatsiooni poole sellisel juhul arvutuse käigus maksimaalselt toimub (algset väljakutset loendamata)?

10.18. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: $n > 0$.
Tulemus: n-tipuline juhu-kahendotsimispuu.
Võimalik skeem:

- luua n-tipuline juhu-kahendpuu *kp*;
- luua n-elemendiline juhuarvude sorteeritud järjend *a* (näiteks juhuarvudest lõigult [1; 99]);

- omistada arvud järjendist a kahendpuu kp tippude märgenditeks tippude keskjärjestuses.

10.19. Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud: kop - mittetühi kahendotsimise puu, t - selle tipp ja arv k, mille kohta (ja ülemust) otsida alates tipust t.

Tulemus: paar $<t1, t2>$, kus

- $t1$ on tipp, milles arv k leiti (või Λ, kui arvu ei leitud);
- $t2$ on $t1$ ülemus (või Λ, kui $t1$ on juurtipp);
- juhul, kui arvu ei leitud ($t1 = \Lambda$), siis $t2$ on tipp, millele arv k sobiks alluvaks.

10.20. *(s)* Tõestada, et ei leidu halvima juhu ajalise keerukusega $\Theta(n)$ algoritmi, mis lisab kahendotsimispuusse n-elemendilise sorteerimata järjendi kõik kirjed.

10.21. *(s)* Antud kahendotsimispuu langetab selles olnud lehed järjekorras vasakult paremale. Kui tulemuseks ei ole tühi puu, siis ka see langetab lehed samal moel. Kuni lõpuks on langetatud ka puu juur. Langetatud tippudes olnud kirjete võtmed paigutatakse loetellu langemise järjekorras.

(a) Programmeerida funktsioon leidmaks etteantud kahendpuu langenud lehtede võtmete loetelu.

(b) Kirjutada algoritm leidmaks mingi kahendotsimispuu tippude (võtmete) eesjärjestus, kui antud on vaid selle langenud lehtede võtmete loetelu.

Võib eeldada, et kirjete võtmed on unikaalsed.

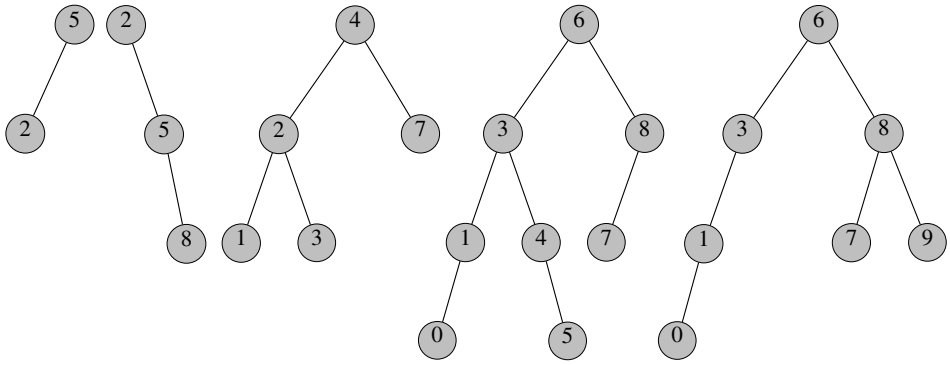
10.22. *(s)* Programmeerida funktsioon, mis (ilma kahendpuud rekonstrueerimata) kontrollib, kas antud paarikaupa erinevate võtmete järjend esindab mingi kahendotsimise puu lõppjärjestust.

II AVL-puu

Kokkuleppeliselt, AVL-puu topeltpöörded on vasakparem- (algul vasakus harus vasakpööre, seejärel kogu puus parempööre) ja paremvasakpööre (algul paremas harus parempööre, seejärel kogu puus vasakpööre). Mõiste „kahekordne pööre“ on sünonüümiks mõistele „topeltpööre“.

10.23. *(s)* Selgitada, mille poolest on AVL-puu eelistatum kui suvaline kahendotsimispuu.

10.24. *(v)* Millised joonisel 11 kujutatud puudest on AVL-puud?



Joonis 11: Viis kahendotsimispuud.

10.25. *(s)* Olgu puu juurtipu sügavuseks 0, selle otseste alluvate sügavusteks 1 jne. Kas AVL-puus saab olla lehttippe nii sügavustel 2 kui ka 5? Kui selline puu eksisteerib, siis tuua näide. Vastasel juhul põhjendada, miks sellist pole.

10.26. Joonistada kõrgusega 4 AVL-puu, milles leidub ainult üks lehttipp, mille eemaldamine ei rikuks tasakaalu. Märgistada see graafi tipp ning põhjendada, miks just see on ainus.

10.27. *(s)* Joonistada üks minimaalse tippude arvuga viietasemeline AVL-puu.

10.28. Tuua näide AVL-puust, mille lehttippude seas on enamuses sellised, mille eemaldamisel läheb vaja puu tasakaalustamist.

10.29. *(sv)* Millise kõrgusega saab olla 60-tipuline AVL-puu? (Puu kõrguseks lugeda selle viimase taseme number.)

10.30. Programmeerida funktsioon, mis koostab järjendi antud AVL-puu kõigi lehttippude sügavuste hulgast (iga võimalik sügavus esineb tulemusjärjendis parajasti üks kord).

10.31. Joonistada AVL-puu, milles leidub lehti vähemalt kolmel eri tasemel.

10.32. *(v)* Ühe AVL-puu kohta on teada, et selles leidub lehttippe sügavustega 50 ja 100. Millistel sügavustel veel leidub selles puus kindlasti lehttippe?

10.33. Programmeerida rekursiivne funktsioon kõrgusega h AVL-puu genereerimiseks järgmisel viisil:

- kui kõrgus on 1, luua lehttipp;
- kui kõrgus on 0, tagastada tühiviit;

- kui kõrgus $h > 1$, siis luua juurtipp ning sellele kaks alampuud vastavalt juhuslikult valitud kõrgustega: kas $(h-1, h-1)$, $(h-1, h-2)$ või $(h-2, h-1)$;
- funktsioon tagastab viida juurtipule.

10.34. Programmeerida funktsioon etteantud AVL-puu tippude arvu leidmiseks.

10.35. *(s)* Programmeerida funktsioon etteantud tippude arvuga ja struktuuriga AVL-tasakaalus oleva kahendpuu tippude täitmiseks mingite täisarvudega nii, et tekiks AVL-puu.

10.36. *(s)* Programmeerida funktsioon AVL-puu väljastamiseks konsoolile (igas reas puu ühe taseme tipud sobiva taandega).

10.37. Programmeerida funktsioon antud võtmeväärtusega kirje otsimiseks AVL -puust.

10.38. *(s)* Milline on suurim puu tippude arv x, mille jaoks leiduvad AVL-puud on kõik ühe ja sama kõrgusega?

10.39. *(s)* Leida vähim tippude arv x, mille jaoks leidub kolm erineva kõrgusega AVL-puud.

10.40. *(sv)* Tähistagu $c(h)$ kõrgusega h AVL-puu minimaalset tippude arvu. Leida rekurrentne seos $c(h)$ jaoks.

10.41. *(s)* Tõestada, et AVL-puu kõrgus $h < \log_c n + 1$, kus $1 < c < 2$.

10.42. *(s)* Programmeerida võimalikult efektiivne funktsioon, mis etteantud kahendpuu korral kontrollib, kas selle igas tipus on rahuldatud AVL-tasakaalu tingimus. Lisavälja, kus oleks salvestatud puu kõrgus, tippudes ei ole.

10.43. *(s)* Programmeerida rekursiivne funktsioon järgmise ülesande lahendamiseks.

Antud:

- kahendpuu kp, milles iga tipuga on seotud kõrguseväli $.h$;
- kp tipp t;
- tipu t ülemus (Λ, kui t on kp juurtipp).

Tulemus: kp tasakaalu rikkekohtade list paaridest $< v, y >$, kus

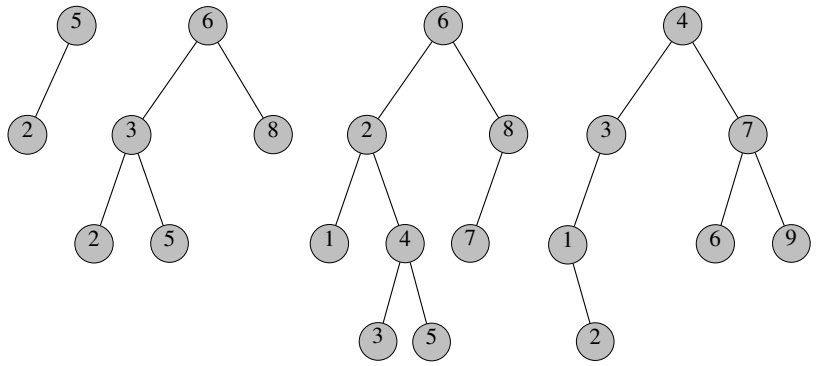
- v on AVL-rikkega tipp, st v vasaku ja parema alampuu kõrgused erinevad rohkem kui ühe võrra;
- y on selle tipu (v) ülemus (või Λ , kui v on kp juurtipp).

10.44. *(s)* Tõestada, et AVL-puu otsimis- ja lisamisprotseduurid on ajalise keerukusega $O(\log n)$, kus n on puu tippude arv.

10.45. *(s)* Kirjutada programm, mis sisendina antud kahe AVL-puu tipukirjetest moodustab uue AVL-puu. Programmi ajaline keerukus peab olema $O(m+n)$, kus m ja n on sisendpuude tipukirjete arvud.

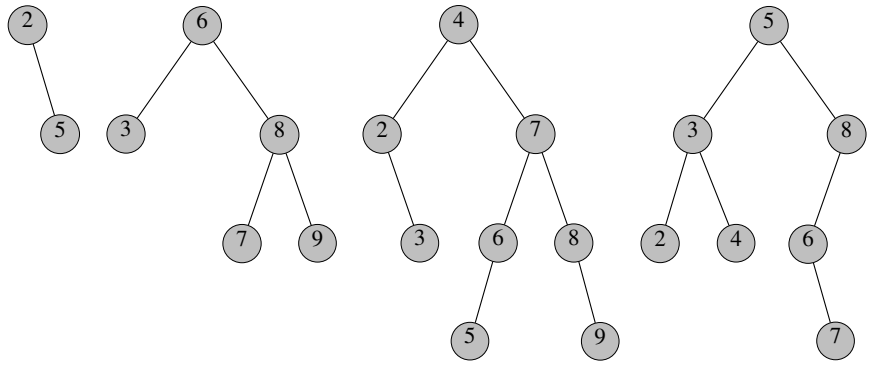
III AVL-puu pöörded

10.46. Sooritada joonisel 12 kujutatud kahendotsimispuudes parempööre.



Joonis 12: Neli kahendotsimispuud.

10.47. Sooritada joonisel 13 kujutatud kahendotsimispuudes vasakpööre.



Joonis 13: Neli kahendotsimispuud.

10.48. Millistes kahendotsimispuudes on võimalik teha vasakparempööret? (Vastust mitte anda teiste pöörete võimalikkuse kaudu.)

10.49. Sooritada

(a) vasakparempööre;

(b) paremvasakpööre

neis joonistel 12 ja 13 antud kahendotsimispuudes, milles võimalik.

10.50. Joonistada

- mingi 11-tipuline AVL-puu;
- AVL-puu, mis saadakse sellest pärast juurtipus oleva kirje eemaldamist (vajadusel tasakaalustamist rakendades).

10.51. Esitada näide AVL-puu tasakaalustamisest kahekordse pöördega.

10.52. Joonistada AVL-puu, millest tipu eemaldamisel saadakse (enne pöördeid) tasakaalustamata puu, kus kõik lehed on

(a) eri tasemetel;

(b) samal tasemel,

näidates ka eemaldatava tipu.

Tasakaalustada see puu. Joonistada saadud kahendpuule vastav harilik puu (mets).

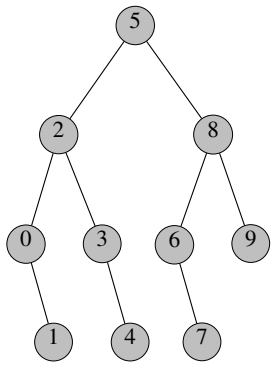
10.53. *(s)* Algselt tühja AVL-puusse lisatakse järjest kirjed

(a) (reaalarvuliste) võtmetega 11, 13, 16, 14, 12, 19, 18, 17, 16.5 ja 13.5;

(b) võtmetega 10, 28, 88, 56, 58, 44, 33, 32, 16 ja 29.

Kujutada joonisel puu seis pärast iga lisamist.

10.54. Eemaldada joonisel 14 kujutatud AVL-puust järjest kirjed võtmetega 8, 9, 5, 0, 2, 1. Kujutada joonisel puu seis pärast iga eemaldamist.



Joonis 14: AVL puu.

10.55. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud:

- kahendotsimispuu *kop* AVL-rikkega ühes tipus ([1], joonis 2.4 või selle peegeldus);
- kolm tippu – rikkega tipp a (allaviidav), selle alluv b (ülesviidav), tipu a ülemus y (Λ, kui a on *kop* juurtipp).

Tulemus: antud kahendotsimispuus *kop* sooritatud tasakaalustamise võte (ehk pööre).

10.56. Programmeerida funktsioon kirje lisamiseks (koos tasakaalustamisega) AVL-puusse.

10.57. Programmeerida funktsioon kõikide antud võtmeväärtusega kirjete eemaldamiseks (koos tasakaalustamisega) AVL-puust.

10.58. Programmeerida funktsioon, mis lahendab järgmise ülesande.
Antud: kahendotsimispuu *kop*, mis on saadud AVL-puust selle ühe tipu eemaldamisel või ühe tipu lisamisel.
Tulemus: kui esineb AVL-rike, siis see parandatud, st *kop* teisendatud AVL-puuks. (Vt [1], joonis 2.4, joonis 2.5.)

IV Mitmerajaline otsimispuu

Mitmerajalise otsimispuu praktikas tähtsamaks erijuhuks on *B-puu* ([1], lk 35).

10.59. *(v)* Mitu kirjet minimaalselt ja mitu kirjet maksimaalselt saab olla

(a) kolmandat järku B-puu;

(b) üheksandat järku B-puu

juurtipus; vahetipus?

10.60. Joonistada kõik viiendat järku B-puud, milledes on parajasti seitse kirjet võtmetega 1, 2, 3, 4, 5, 6 ja 7. Põhjendada, et rohkem selliseid B-puid ei ole.

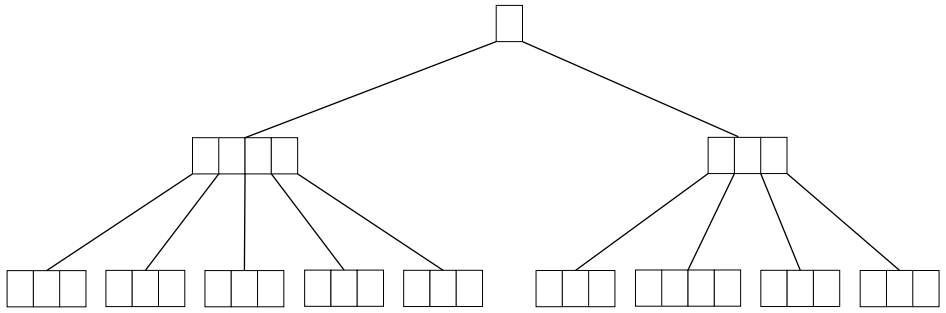
10.61. Konstrueerida 5-tipulise seitsmendat järku B-puu näide.

10.62. Joonisel 15 kujutatud B-puu struktuuris täita tühjad võtmekohad sobivate väärtustega. Mis võiks olla selle B-puu järk?

10.63. Kas AVL-puu on struktuurilt B-puu alaliik, kus järk on 2?

10.64. Kas B-puud on võimalik defineerida induktiivselt?

10.65. Kas m-järku B-puu määratluses võiks juurest erinevate tippude kirjete arvu alampiiriks lugeda $\lfloor \frac{m-1}{2} \rfloor$?



Joonis 15: B-puu näitestruktuur.

10.66. Esitada näide kirje lisamisest (vähemalt kolmetasemelisse) B-puusse, mille korral selle juur ületäitub.

10.67. Tuua näide kolmandat järku vähemalt kahe tasemega B-puust, millest elemendi eemaldamisel väheneb tasemete arv. Joonistada töö algseis, eemaldatav element ja lõppseis.

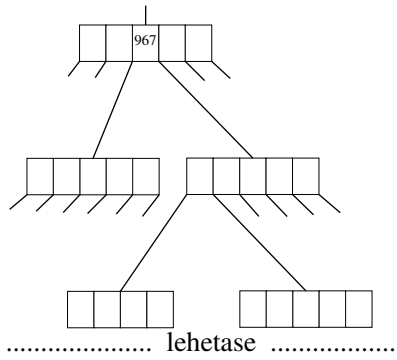
10.68. Lisada algselt tühja neljandat järku B-puusse järjest kirjed võtmetega 18, 50, 55, 17, 25, 33, 34, 47, 26, 38, 27, 10, 20. Joonistada puu seis iga struktuurimuutuse eel ja järel.

10.69. Lisada algselt tühja viiendat järku B-puusse

(a) kirjed võtmetega 55, 70, 80, 63, 24, 33, 72, 15, 25, 51, 47, 12, 60, 65, 90, 11, 23;

(b) kirjed võtmetega 40, 88, 50, 15, 27, 25, 55, 56, 60, 75, 24, 12, 44, 90, 89, 37, 57, 81, 18, 36.

Joonistada seis puustruktuuri iga muutuse (kui tippude arv muutub) eel ja järel.



Joonis 16: B-puu fragment.

10.70. Lahendada ülesanne 10.69, kui B-puu on

(a) kolmandat järku;

(b) neljandat järku.

10.71. Eemaldada kirjed

(a) ülesandes 10.69;

(b) ülesandes 10.70

koostatud B-puudest samas järjekorras, nagu nad sinna olid pandud. Joonistada seis iga puustruktuuri muutuse eel ja järel.

Eemaldamisoperatsiooni naabrilt laenamise osas eelistada vasakult naabrilt laenamist, selle puudumisel paremalt naabrilt võtmist.

10.72. Joonisel 16 kujutatud üheksandat järku B-puu fragmendis

- täita tühjad võtmekohad sobivate väärtustega;
- kirjeldada, kuidas eemaldatakse kirje võtmega 967;
- joonistada pärast eemaldamist saadud fragment.

10.73. *(s)* Tõestada, et B-puu kõrgus on $\Theta(\log n)$, kus n on kirjete arv puus.

11. Kuhjad

I Kahendkuhi

Kahendkuhja esituses massiivina paiknevad kirjed (tipud) järjestikku kahendpuu tasemete kaupa ([1], lk 52). Kui kahendkuhja kujutava massiivi a elementide indekseerimine algab nullist, siis tipu a_k alluvad (kui need leiduvad) on a_{2k+1} ja a_{2k+2} ning ülemus (kui see leidub) on $a_{\lfloor (k-1)/2 \rfloor}$.

11.1. Millised joonisel 17 esitatud puudest on kahendkuhjad või kahendpöördkuhjad? Eemaldada neist kuhjadest juurtipu kirjet niikaua, kuni alles jääb 6 tippu; seejärel lisada kirjed võtmetega 46 ja 25. Kujutada joonisel puuna kuhja seis pärast iga eemaldamist/lisamist.

11.2. Kirjutada algoritm kontrollimaks, kas antud kirjete massiiv on kahendkuhi.

11.3. Kirjutada algoritm sellise kirje võtmiseks kahendkuhjast, mille võti on suuruselt teine.

11.4. Millised järgmistest massiividest esitavad kahendkuhja?

50	55	52

61	50	60	20

80	90	50	79	60	41	22

40	33	22	30	25	24	21	19	17	16	15	14	13	12	10

96	78	86	69	70	81	80	50	61	49	59	79	75	76

10	20	27	25	24	32	28	35	29	40	41	30	39

Korrata neis kahendkuhjades suurima elemendi eemaldamist kuni kuhja tühjakssaamiseni. Joonistada seejuures puuna välja kuhja algseis ning seis iga kirje eemaldamise järel.

11.5. Lisada algselt tühja kahendkuhja järjekorras kirjed võtmetega

(a) 10, 15, 12, 18, 16, 11, 20, 13, 14, 17;

(b) 60, 45, 65, 50, 30, 35, 40, 70, 55, 25.

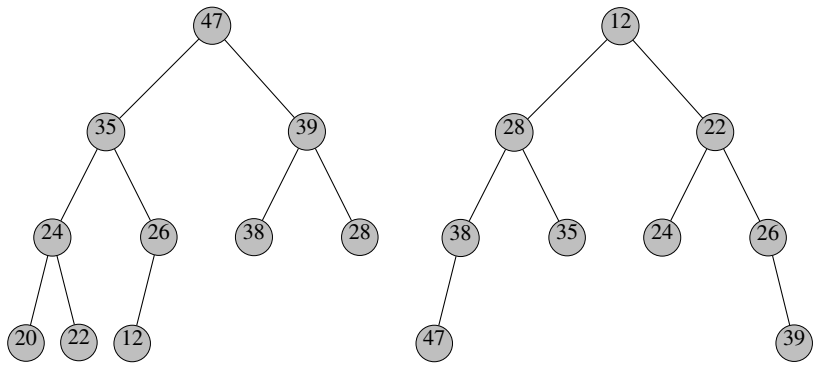
Joonistada puuna välja kuhja seis iga lisamise järel.

Korrata tulemuseks olevais kahendkuhjades suurima elemendi eemaldamist, kuni kuhi saab tühjaks. Joonistada puuna välja kuhja seis iga eemaldamise järel.

11.6. *(v)* Sooritada massiivide

75	44	35	56	38	36	49	10	70	64	32	72	37	18	16	14	21	28	19	74

51	60	48	90	88	49	75	59	61	91	76	80	85	81	70	71	68	55	58	50



Joonis 17: Kuhjad.

kuhjastamine. Esitada joonisel algseis ja vaheseisud peale igat sellist allaviimist, kus toimus vähemalt kahe tipukirje asukoha muutus, ja lõppseis puu kujul.

11.7. Kirjutada

(a) rekursiivne;

(b) mitterekursiivne

algoritm antud massiivi kuhjastamiseks ülesviimist kasutades.

11.8. Sooritada massiivi

10	15	20	25	35	30	45

järjestamine kuhjameetodil. Joonistada massiivina välja vaheseis pärast iga vahetust.

11.9. Kirjutada rekursiivne algoritm kahendkuhja kujutava massiivi mittekahanevalt sorteerimiseks.

11.10. Kirjutada algoritm järgmise ülesande lahendamiseks: kui antud kahendkuhi ei ole täielik kahendpuu, siis eemaldada viimasel tasemel olevad tipud.

11.11. Kirjutada algoritm järgmise ülesande lahendamiseks: eemaldada antud kahendkuhjast tipud, mis asuvad $\lfloor h/3 \rfloor$ viimasel tasemel, kus h on kahendkuhja kõrgus.

11.12. Programmeerida võimalikult efektiivne funktsioon, mis võtab sisendiks kahendkuhja kujutava massiivi ja tema mingi indeksi ning väljastab vastavast tipust lähtuva alampuu kõikide tippude kirjed tasemete kaupa.

11.13. Algselt tühja andmestruktuuri lisatakse n kirjet, millest pooltel on võti ligikaudu võrdne maksimaalse võtmega ja ülejäänute võtmed jaotuvad ühtlaselt

üle kogu võtmete piirkonna. Võrrelda selliste kirjete lisamise efektiivsust nii kahendkuhja kui ka kahendotsimispuu korral kummagi juhu Θ-hinnangu (n suhtes) alusel.

11.14. *(s)* Sõnastada võimalikult efektiivne algoritm saamaks kahendotsimispuust kahendkuhja, mis sisaldab täpselt samad kirjed. Leida koostatud algoritmi ajalise keerukuse Θ-hinnang (kahendotsimispuu tippude arvu suhtes).

11.15. *(s)* Programmeerida võimalikult efektiivne funktsioon, mis saab sisendiks kaks massiivina antud kahendkuhja ja tagastab mõlema kuhja kirjeid sisaldava kahendkuhja massiivina. Anda loodud funktsiooni Θ-hinnang sisendkuhjade suuruste suhtes.

11.16. Kuhja lisatakse ühekaupa järjendi elemendid järjekorras vasakult paremale. Kirjeldada iga võimaliku pikkuse n jaoks sisendjärjend, mis realiseerib kogu protsessi ajalise keerukuse halvima juhu.

11.17. Joonistada viis erineva struktuuriga mittetühja kahendpuud, mis on korraga nii kahendotsimispuud kui ka kahendkuhjad.

11.18. *(s)* Juku kaalub massiivi kuhjastamiseks järgmisi protseduure:

(a) igale massiivi elemendile järjekorras vasakult paremale rakendada protseduuri `viia_alla`;

(b) igale massiivi elemendile järjekorras vasakult paremale rakendada protseduuri `viia_üles`;

(c) igale massiivi elemendile järjekorras paremalt vasakule rakendada protseduuri `viia_alla`;

(d) igale massiivi elemendile järjekorras paremalt vasakule rakendada protseduuri `viia_üles`.

Millistel neist juhtudest saadakse alati kahendkuhi? Põhjendada.

11.19. *(s)* Tõestada, et kuhjastamise rekursiivse variandi halvima juhu ajaline keerukus on $\Theta(n)$, kus n on kirjete arv puus.

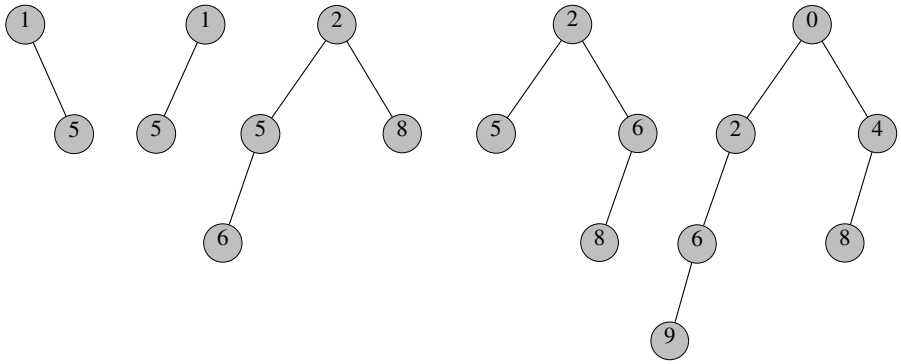
II Vasakkalduvad kuhjad

11.20. Millised puudest joonisel 18 on vasakkalduvad kuhjad, kui alampuu kaaluks loetakse

(a) tippude arv;

(b) paremkõrgus?

Vasakkalduvates kuhjades korrata vähima võtmega kirje eemaldamist kuni kuhja tühjakssaamiseni. Joonistada kuhja seis iga kirje eemaldamise järel.



Joonis 18: Kahendpuud.

11.21. Lisada algselt tühja vasakkalduvasse kuhja, mille alampuude kaaluks loetakse tippude arv, järjest kirjed võtmetega

(a) 10, 15, 12, 18, 16, 11, 20, 13;

(b) 60, 45, 65, 50, 30, 35, 40, 70, 55.

Joonistada kuhja seis iga lisamise järel.

Seejärel korrata tulemuseks olevaist vasakkalduvatest kuhjadest vähima võtmega kirje eemaldamist kuni kuhja tühjakssaamiseni. Joonistada kuhja seis iga kirje eemaldamise järel.

11.22. Olgu antud ahel, milles on järjest kirjed võtmetega

(a) 75, 44, 35, 56, 38, 36, 49, 10, 70, 64;

(b) 51, 60, 48, 90, 88, 49, 75, 59, 61, 91, 76.

Sooritada ahela kuhjastamine vasakkalduvaks kuhjaks, kus alampuu kaaluks loetakse selle tippude arv.

III Binomiaalkuhi

11.23. *(v)* Mitu tippu on binomiaalpuus, mille kõrgus on 8?

11.24. *(v)* Mitmendat järku binomiaalpuudest koosneb 168-kirjeline binomiaalkuhi?

11.25. Olgu n=(Teie sünniaasta)+(Teie sünnikuu)+(Teie sünnipäev kuus). Millist järku binomiaalpuud esinevad n-kirjelises binomiaalkuhjas?

11.26. Joonistada binomiaalkuhi,

(a) milles on kirjed võtmetega $1,2,\ldots,23$;

(b) milles on kirjed võtmetega $1,2,\ldots,26$;

(c) mis saadakse eelmises alamülesandes (b) tehtud kuhjast suurimat järku puu juurtipu eemaldamisel.

11.27. Joonistada binomiaalkuhi, milles on 13 kirjet ja vähima võtmeväärtusega kirje asub juurahela suurima astmega puu juurtipus. Seejärel

- kirjeldada, kuidas toimub vähima võtmeväärtusega kirje võtmine sellest binomiaalkuhjast;
- joonistada pärast vähima võtmeväärtusega kirje võtmist saadud binomiaalkuhi.

11.28. Lisada algselt tühja binomiaalkuhja kirjed võtmetega

(a) 14, 19, 29, 24, 23, 11;

(b) 10, 15, 12, 18, 16, 11, 20, 13, 14, 17;

(c) 60, 45, 65, 50, 30, 35, 40, 70, 55, 25;

(d) 10, 15, 20, 25, 35, 30, 45, 40, 50, 55, 60, 65, 80, 70, 75.

Joonistada kuhja seis iga lisamise järel.

11.29. Lisada algselt tühja binomiaalkuhja järjest kirjed võtmetega 12, 45, 30, 16, 34, 13, 17, näidates joonisel kuhja seisu pärast iga lisamist.

Seejärel eemaldada saadud kuhjast kolm korda järjest vähima võtmega kirje, näidates joonisel kuhja seisu pärast iga eemaldamist.

11.30. Sooritada ülesande 11.28(a) tulemuseks oleva binomiaalkuhja ühendamine

(a) ülesande 11.28(d);

(b) ülesande 11.28(c)

tulemuseks oleva binomiaalkuhjaga. Näidata kõik operatsioonid, mis binomiaalpuid muudavad (kujutada joonisel kuhja seis enne ja pärast).

11.31. Alustades

(a) ülesande 11.28(a);

(b) ülesande 11.28(c)

tulemuseks olevast binomiaalkuhjast, korrata vähima võtmega kirje eemaldamist kuni kuhja tühjakssaamiseni. Joonistada välja kuhja seis iga eemaldamise järel.

11.32. Olgu antud ahel, milles on järjest kirjed võtmetega

(a) 75, 44, 35, 56, 38, 36, 49, 10, 70, 64, 32, 72, 37, 18, 16, 14, 21, 28, 19, 74;

(b) 51, 60, 48, 90, 88, 49, 75, 59, 61, 91, 76, 80, 85, 81, 70, 71, 68, 55, 58.

Sooritada ahela kuhjastamine binomiaalkuhjaks, tehes ka selgitava joonise.

11.33. Millises olukorras tekib kahe binomiaalkuhja ühendamisel sama järku puude kolmik?

11.34. *(s)* Millised juhud on üksiku kirje lisamisel suure kirjete arvuga binomiaalkuhja halvimad (algoritm suhteliselt kõige aeglasem) ja millised parimad?

11.35. Millised kuhjadel teostatavad operatsioonid on kahendkuhjal ja binomiaalkuhjal erineva halvima juhu keerukusega? Põhjendada, tuues ära vastavad keerukushinnangud.

12. Klasside kujutamine

12.1. Selgitada, miks on ebaefektiivne klassijaotuse hoidmine

(a) listidena, igas listis ühe klassi elemendid;

(b) metsana, kus klassi elemendid paiknevad puus, mille juurtipuks on kanooniline esindaja ja igast ülejäänud tipust osutab viit juurtipule.

12.2. Miks hoitakse kõrgusoptimeeringuga Galler-Fischeri meetodis kõrgusi mälus?

12.3. Miks kasutatakse topeltoptimeeringuga Galler-Fischeri meetodis kõrguse asemel heuristikut?

12.4. Olgu antud Galler-Fischeri meetodil esitatud klassijaotus, kus alguses moodustavad kirjed a, b, c, d, e, f, g, h, i, j, k, l igaüks omaette klassi. Teostada järjekorras järgmised klasside ühendamised:

1) a klass b klassiga;
2) c klass d klassiga;
3) a klass d klassiga;
4) e klass f klassiga;
5) g klass h klassiga;
6) f klass g klassiga;
7) d klass h klassiga;
8) a klass e klassiga;
9) i klass j klassiga;
10) k klass l klassiga;
11) j klass l klassiga;
12) a klass l klassiga.

Joonistada lõppseis.

12.5. Olgu antud Galler-Fischeri meetodil esitatud klassijaotus, kus alguses moodustavad kirjed a, b, c, d, e, f, g, h, i, j igaüks omaette klassi. Teostada järjekorras järgmised klasside ühendamised:

1) a klass b klassiga;
2) b klass c klassiga;
3) c klass a klassiga;
4) d klass a klassiga;
5) d klass e klassiga;
6) f klass g klassiga;
7) j klass h klassiga;
8) b klass f klassiga;
9) j klass i klassiga;
10) d klass j klassiga.

Joonistada välja lõppseis.

12.6. Olgu antud teede õgvendamisega kõrgusoptimeeringuta Galler-Fischeri meetodil esitatav klassijaotus, kus algselt on kirjed a, b, c, d, e, f, g, h, i kõik eraldi klassides. Teostada järjekorras järgmised ühendamised:

1) a klass i klassiga;
2) b klass a klassiga;
3) f klass e klassiga;
4) e klass d klassiga;
5) a klass c klassiga;
6) a klass e klassiga;
7) h klass i klassiga.

Kujutada joonisel seis iga ühendamise järel. Mitmesse klassi jagunevad kirjed töö lõpul? Esitada kirjete nimekirjad lõppseisu klasside kaupa.

12.7. Kasutades kõrguse järgi optimeeritud ühendamist, lahendada

(a) ülesanne 12.5;

(b) ülesanne 12.4.

12.8. Olgu antud Galler-Fischeri meetodil esitatud klassijaotus, kus alguses moodustavad kirjed a, b, c, d, e, f, g, h, i, j igaüks omaette klassi. Teostada järjekorras järgmised klasside ühendamised, kasutades kõrguse järgi optimeeritud lähenemist:

1) a klass b klassiga;
2) b klass c klassiga;
3) d klass f klassiga;
4) i klass a klassiga;
5) b klass b klassiga;
6) c klass d klassiga;
7) f klass i klassiga;
8) e klass g klassiga;
9) j klass h klassiga;
10) g klass j klassiga;
11) d klass g klassiga;
12) a klass c klassiga.

Joonistada lõppseis.

12.9. Kasutades kõrgusoptimeeringu asemel teede õgvendamist, teostada operatsioonid

(a) ülesandest 12.5;

(b) ülesandest 12.4;

(c) ülesandest 12.8.

Joonistada seis pärast iga ühendamist.

12.10. Kasutades teede õgvendamist koos ühendamisega pseudokõrguse järgi, teostada operatsioonid

(a) ülesandest 12.5;

(b) ülesandest 12.4.

Joonistada seis pärast iga ühendamist.

13. Graafi läbimine

Graafi läbimisel liigutakse tipust tippu mööda graafi kaari. Tipus, kuhu jõutakse, töödeldakse tipu info (nt prinditakse tipu number) parajasti üks kord. Tippude töötlemise järjekord määrab tippude järjestuse konkreetse läbimisviisi korral.

I Läbimise liigid

Graafi *sügavuti läbimise rekursiivne algoritm* on esitatud õpikus ([1], lk 108). Kuna selles tipu (a) töötlemine ($f(a)$) on ette nähtud enne tsüklit, siis on tegemist sügavuti läbimisega eesjärjestuses. Kui töötlemisfunktsiooni rakendataks pärast tsükli täitmist, oleks tegemist sügavuti läbimisega lõppjärjestuses.

Graafi *läbimise mitterekursiivses protseduuris* hoitakse abihulgas Q (nn frondis) tippe, mis on juba vaadeldud (milleni on juba jõutud), kuid millest tuleb veel püüda edasi liikuda. Protseduuri põhiosaks on tsükkel, milles võetakse üks tipp frondist välja ning fronti pannakse väljavõetud tipu kõik veel vaatlemata naabrid, ühtlasi seades need vaadelduteks. Läbimise liigi määrab frondi andmestruktuur: kui Q on magasin, siis on tegemist graafi sügavuti läbimisega (eesjärjestuses), kui Q on järjekord, siis laiuti läbimisega. Front algatatakse sinna ühe (lähte)tipu lisamisega. Tsükkel lõpeb, kui front on tühjaks saanud.

Märgime, et tippude järjestus graafi läbimisel (nii sügavuti kui ka laiuti) ei ole üheselt määratud, sõltudes sellest, millises järjekorras toimub tsüklis tipu naabrite vaatlusele võtmine.

13.1. Kas graafi, milles on palju tippe, kuid vaid mõni üksik kaar, on otstarbekam esitada ahelstruktuurina või naabrusmaatriksina ([1], lk 44)?

13.2. *(v)* Võttes lähtetipuks tipu a, läbida joonistel 19 ja 20 esitatud graafid

(a) laiuti;

(b) sügavuti eesjärjestuses;

(c) sügavuti lõppjärjestuses.

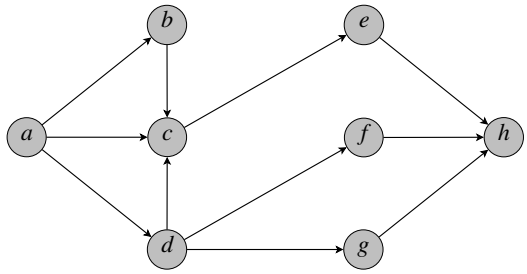
Kõigil juhtudel kirjutada üles tipud töötlemise järjekorras.

13.3. Kirjutada graafi laiuti läbimise algoritm, mis leiab kõik tipud antud graafis, mis on saavutatavad antud tipust lähtudes.

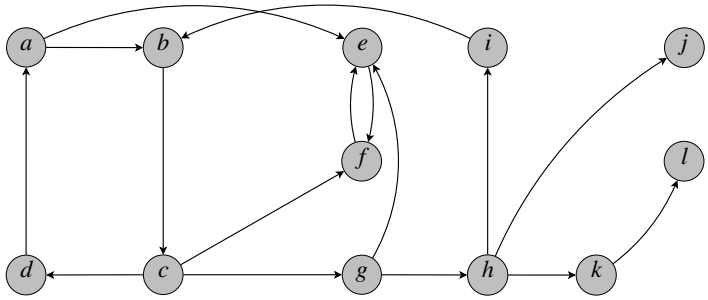
13.4. *(s)* Kirjutada graafi laiuti läbimise algoritm, milles frontide andmestruktuurideks on järjendid, kuhu saab tippe ainult lisada.

13.5. *(v)* Mitmel erineval viisil saab laiuti läbida joonisel 19 toodud graafi, kui alustada tipust a?

13.6. *(s)* Kirjutada algoritm leidmaks, mitmel erineval viisi saab antud graafi



Joonis 19: Graaf ülesannetele 13.2 ja 13.5.



Joonis 20: Graaf ülesannetele 13.2 ja 13.27.

(a) laiuti läbida;

(b) sügavuti läbida,

alustades antud lähtetipust.

13.7. *(sv)* Kas graafi sügavuti läbimise algoritm on halvimal juhul lineaarse keerukusega (tippude arvu suhtes)?

13.8. *(s)* Kirjutada algoritm mis leiab ühe tee antud graafis ühest antud (lähte)tipust teise (siht)tippu

(a) graafi sügavuti läbides;

(b) graafi laiuti läbides.

13.9. *(s)* Kirjutada graafi sügavuti läbimisel põhinev rekursiivne algoritm leidmaks kõik teed antud graafis ühest antud (lähte)tipust teise (siht)tippu.

13.10. Joonistada kolm erineva struktuuriga 5-tipulist graafi, millest igaühe nii sügavuti kui ka laiuti läbimisel töödeldakse tipud samas järjekorras.

13.11. *(v)* Leida tugevalt sidusa graafi laiuti läbimise algoritmi parima juhu ja halvima juhu keerukuse Θ-hinnangud tippude arvu suhtes.

13.12. Sõnastada üks originaalne graafitöötlusülesanne, mille (efektiivne) lahendamine võiks põhineda graafi sügavuti läbimisel. Kirjutada selle ülesande lahendusalgoritm.

II Läbimise rakendusi

13.13. *(s)* Kirjutada laiuti läbimisel põhinev algoritm, mis kontrollib, kas etteantud orienteerimata graaf on tsükliteta.

13.14. *(s)* Kirjutada sügavuti läbimisel põhinev algoritm, mis kontrollib, kas etteantud orienteeritud graaf on tsükliteta.

13.15. *(s)* Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: orienteeritud graaf g ja selle tipp t.
Tulemus: kontrollitud, kas graaf g on puu juurtipuga t.

13.16. Olgu antud geograafiliste punktide graaf, mille iga kaarega (x, y) on seotud väli $.l$ – langus liikumisel punktist x punkti y. *Ohutuimaks teeks* kahe punkti vahel nimetame sellist teed, millel suurim lokaalne langus (punktist järgmisse punkti) on minimaalne. Kirjutada graafi sügavuti läbimisel põhinev algoritm, mille käigus leitakse ohutuim tee antud punktist teise antud punkti.

13.17. Olgu antud geograafiliste punktide graaf, mille iga tipuga on seotud vastava punkti kõrgus merepinnast.

(a) Kirjutada graafi laiuti läbimisel põhinev algoritm, mis kontrollib, kas kahe antud punkti vahel leidub samakõrgustee, st. tee, millel asuvad punktid on kõik ühe ja sama kõrgusega.

(b) Kirjutada graafi sügavuti läbimisel põhinev algoritm leidmaks sellist teed antud punktist teise antud punkti, mille kõrgeim tipp on võimalikult madalal.

13.18. *(s)* Olgu labürint esitatud 0-1-maatriksina, kus 0-element tähistab vaba kohta. Liikuda vabalt kohalt teisele kohale saab maatriksi piires üles, alla, vasakule, paremale, tingimusel, et sihtkoht ei ole 1-element. Programmeerida funktsioon leidmaks labürindis teid antud lähtekohast antud sihtkohta.

13.19. *(s)* Programmeerida funktsioon, mis leiab (kui võimalik) ühe tsükli, mis läbib antud graafi antud tippu. Leitud tsüklis võib olla korduvaid tippe, kuid ei tohi olla korduvaid kaari.

13.20. *(s)* Koostada programm leidmaks (kui võimalik) Euleri tsükkel antud orienteerimata graafis. Euleri tsüklis läbitakse iga serv ainult üks kord. Euleri tsükkel leidub parajasti siis, kui graaf on sidus ja iga tipu aste on paarisarv.

13.21. Sõnastada järgmise ülesande algoritm.
Antud: kaarepikkustega varustatud graaf ja selle tipp v.

Tulemus: leitud (kui võimalik) tippu v läbiva lühima tsükli pikkus (kaarte pikkuste summa).

13.22. *(s)* Programmeerida funktsioon järgmise ülesande lahendamiseks.

Antud: graaf, milles võib esineda ka sümmeetrilisi (edas-tagasi) kaari, ning selle graafi üks tipp.

Tulemus: otsitakse üles üks päristsükkel ([1], lk 40) $[v_1, v_2, \ldots, v_k, v_1]$ läbi antud tipu; leitud tsükkel „orienteeritakse“, eemaldades antud graafist selle tsükli tagasi-kaared (kui neid peaks olema) $(v_i, v_{i-1}), i = 2, 3, \ldots k$ ja (v_1, v_k); funktsiooni väärtusena tagastatakse *tõene*, kui tsükkel leidus, vastasel korral – *väär*.

13.23. *(s)* Linna tänavate plaan on esitatud sidusa orienteerimata graafina, milles iga serv kujutab kahesuunalist tänavalõiku linna kahe punkti (graafi tipu) vahel.

Kirjutada algoritm muutmaks plaanil võimalikult palju tänavalõike ühesuunaliseks (asendamaks servi kaartega), nii et säiliks võimalus liikuda igast punktist igasse teise punkti.

13.24. *(s)* Mägikuurordi suurel tablool kuvatakse mäenõlvalt laskumiste plaan. Ülaservas on kujutatud lähtepunkt (milleni viib tõstuk), alaservas aga kõigi teekondade lõpupunkt. Mäenõlval asuvad kümned vahepeatus- ja puhkepaigad on kujutatud samuti punktidena. Punktist punkti viivaid laskumislõike kujutavad nooled. Seejuures on punase noolega tähistatud mõnevõrra ohtlikumad laskumislõigud. Ühte vahepunkti võib siseneda ja sellest väljuda rohkem kui üks nool. Igal hommikul värskendatakse laskumisvõimaluste plaan, vastavalt ilmastiku- ja hooldusoludele. Kuurordi külalistele kavatsetakse edaspidi pakkuda veel võimalus teha päring, leidmaks kõik marsruudid lähtepunktist lõpupunkti, millel on parajasti külalise poolt sisestatud arv punaseid laskumislõike.

Kirjutada kavandatavat lisavõimalust realiseeriv algoritm.

III Sidus graaf

13.25. *(s)* Programmeerida funktsioon, mis kontrollib, kas etteantud orienteerimata sidus graaf on puustruktuuriga.

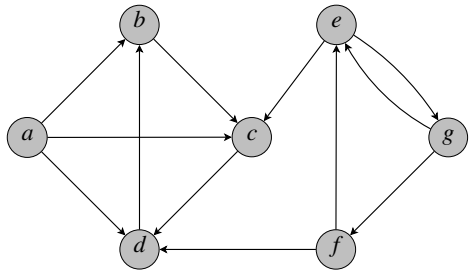
13.26. Kirjutada algoritm, mis etteantud orienteeritud graafi korral kontrollib, kas graaf on tugevalt sidus. Graafi töö käigus muuta pole lubatud.

13.27. Sooritada tugeva sidususe komponentide leidmine Kosaraju algoritmi kohaselt

(a) joonisel 20 esitatud graafi korral;

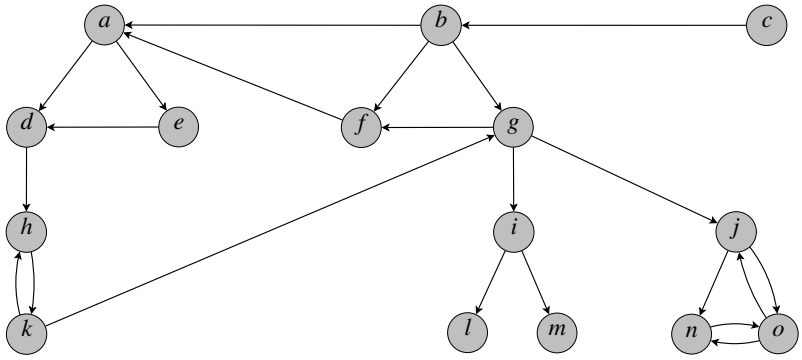
(b) joonisel 21 esitatud graafi korral.

Näidata lõpptulemus, esitades joonisel komponendid nende leidmise järjestuses ning samuti tugeva sidususe komponentide graafid.

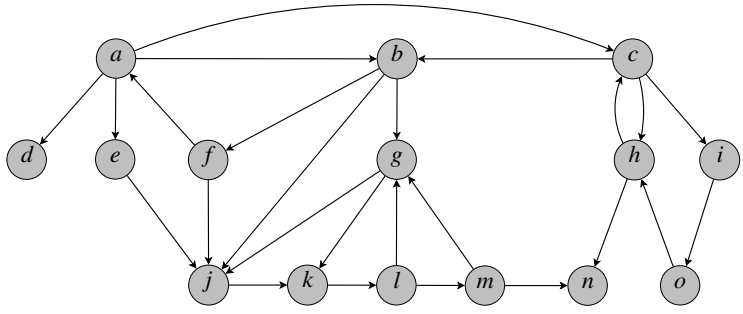


Joonis 21: Graaf ülesandele 13.27.

13.28. Leida joonistel 22 ja 23 esitatud graafide sidususe komponendid Kosaraju algoritmiga, võttes graafi läbimisel mitme võimaluse korral aluseks tähestiku järjestuse. Näidata tippude läbimise järjestus ning komponendid nende leidmise järjestuses.



Joonis 22: Graaf ülesandele 13.28.



Joonis 23: Graaf ülesandele 13.28.

13.29. Sõnastada algoritm, mis leiab etteantud orienteeritud graafi korral selle tugevalt sidusate komponentide arvu.

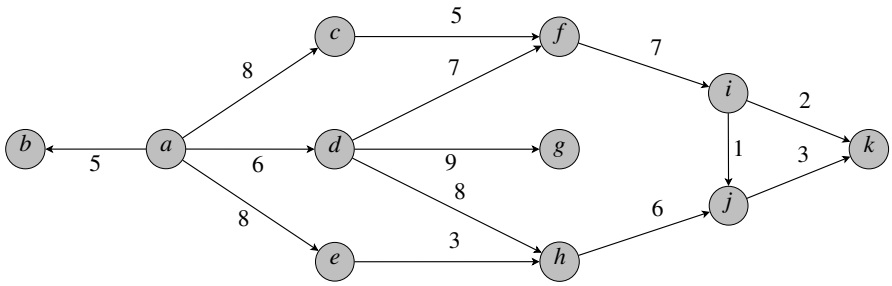
13.30. Miks graafi läbimisel põhinevas nõrga sidususe komponentide leidmise algoritmis tuleb töödelda ka kaared, mis suunduvad juba läbitud tippu?

13.31. Olgu arvutivõrk kujutatud orienteerimata graafina, milles tippudeks on võrgus olevad arvutid ja servadeks arvutitevahelised (võrgu)ühendused. Algolukorras saab iga arvutipaar omavahel kommunikeeruda, kas vahetult või läbi mõne(de) teis(t)e arvuti(te). Koostada programm leidmaks (kui võimalik) selline arvuti (tipp), mille rivist väljalangemine (nt sulgemine) põhjustaks olukorra, kus ülejäänutest vähemalt kaks ei saa enam omavahel kommunikeeruda.

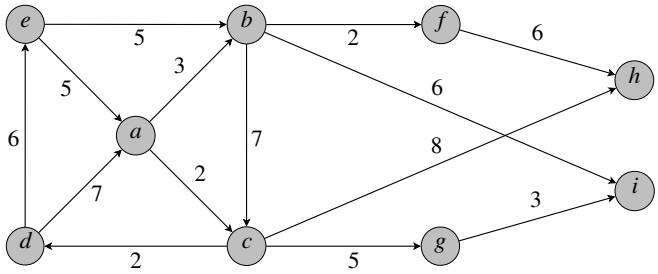
14. Kaugusalgoritmid graafidel

I Algoritmi realiseerimine

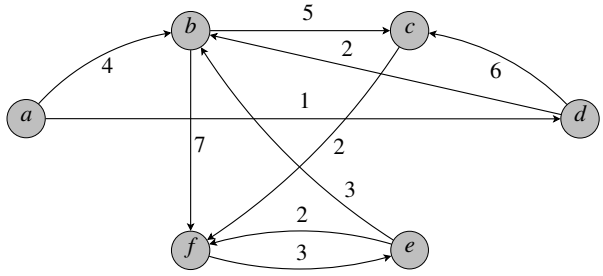
14.1. Leida iga joonistel 24 – 30 kujutatud graafi korral kauguste puu tipu a suhtes Dijkstra algoritmiga. Esitada kaarte lisamise järjekord ja joonistada tulemuseks olev kauguste puu koos kõigi tippude kaugustega tipust a.



Joonis 24: Graaf ülesandele 14.1.

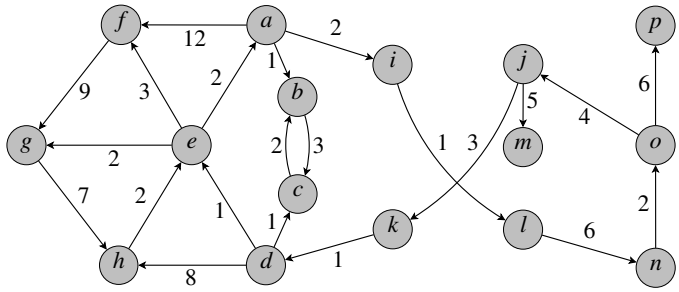


Joonis 25: Graaf ülesandele 14.1.

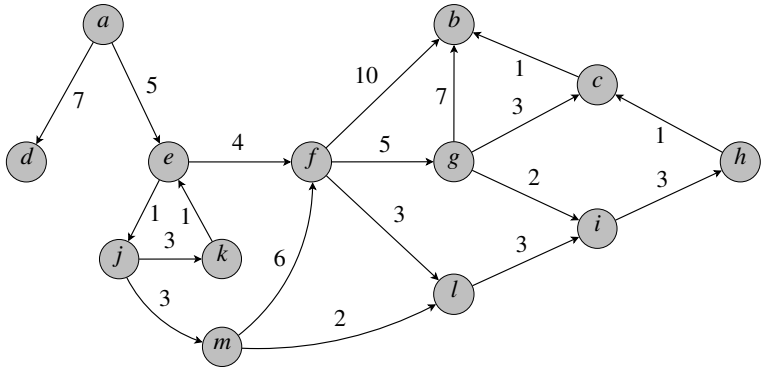


Joonis 26: Graaf ülesandele 14.1.

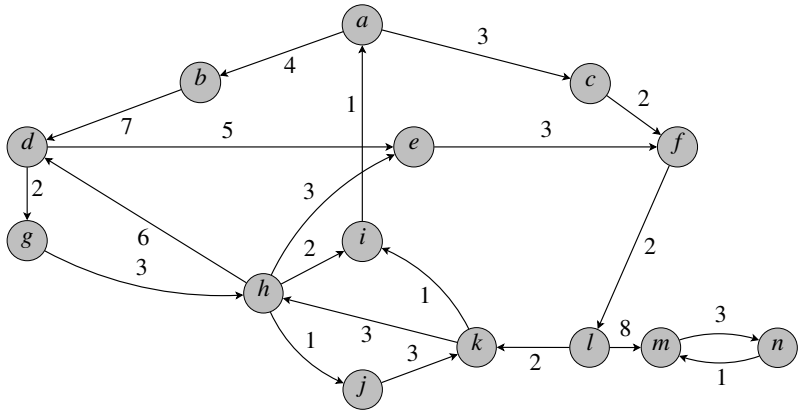
14.2. Kas Dijkstra algoritmi abistruktuurina sobib kasutada kahendkuhja või kahendpöördkuhja? Leida Dijkstra algoritmiga joonisel 31 kujutatud graafi kau-



Joonis 27: Graaf ülesandele 14.1.



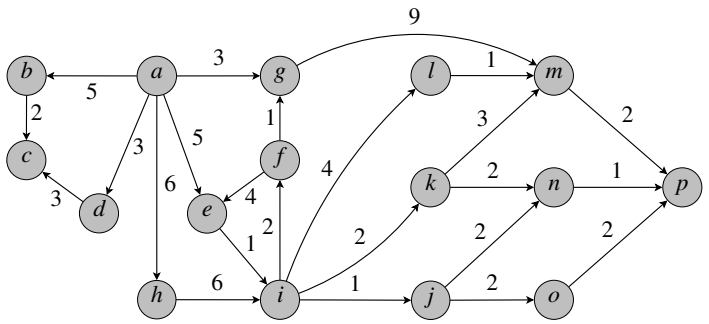
Joonis 28: Graaf ülesandele 14.1.



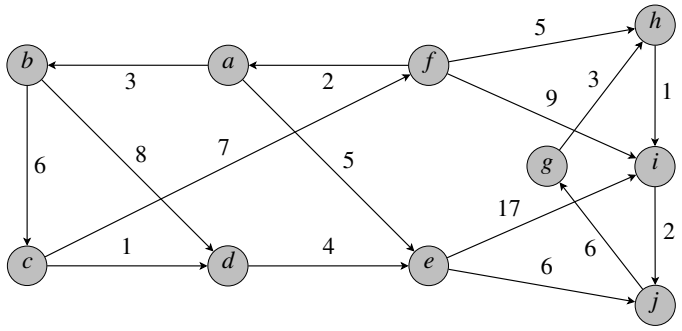
Joonis 29: Graaf ülesandele 14.1.

guste puu tipust a, kasutades abistruktuurina sobivat liiki kuhja. Näidata kuhja seis pärast iga lisamist ja eemaldamist ning algoritmi lõpptulemus.

14.3. Joonistada 8-tipuline võimalikult väheste kaartega graaf, mille ühest ti-



Joonis 30: Graaf ülesandele 14.1.



Joonis 31: Graaf ülesandele 14.2.

pust a on kõik 7 ülejäänud tippu saavutatavad ja, alustades Dijkstra algoritmi tipust a, toimub töö käigus 6 kauguste parandust. Sealjuures näidata, millises järjekorras ja millises ulatuses toimuvad kauguste parandused.

14.4. Tuua näide kaalutud kaartega orienteeritud graafist (kasutada võib ka mittepositiivseid kaale), kus Dijkstra algoritm ei arvuta õigeid tippude kaugusi etteantud tipust. Näidata ka algustipp, mille korral see juhtub.

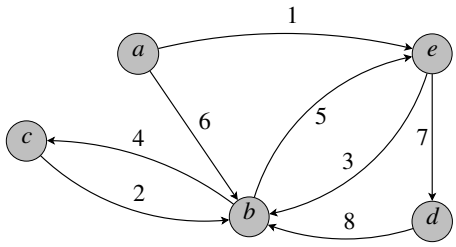
14.5. Leida kaugused igast tipust igasse teise tippu joonisel 32 toodud graafis

(a) Floyd-Warshalli;

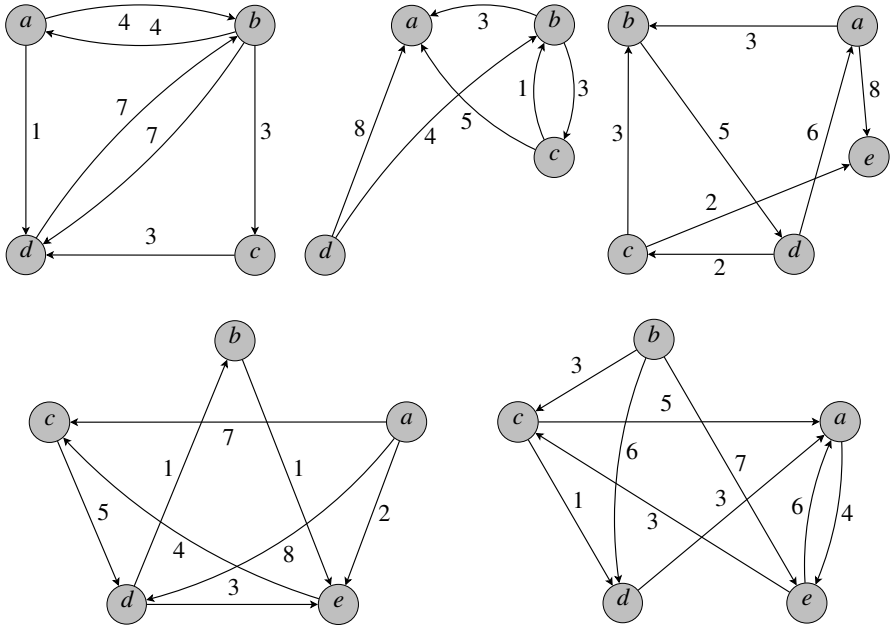
(b) Bellmann-Fordi

algoritmiga. Näidata töö seis enne välimise tsükli sisu iga täitmist ning lõppseis.

14.6. Sooritada joonisel 33 esitatud graafides tippudevaheliste kauguste leidmine Floyd-Warshalli algoritmiga. Kõigil juhtudel kujutada joonisel töö seis enne välimise tsükli iga sammu, ning lõppseis.



Joonis 32: Graaf ülesandele 14.5.



Joonis 33: Graafid ülesandele 14.6.

14.7. *(s)* Tuua näide vähemalt 6-tipulisest graafist, millele Floyd-Warshalli algoritmi rakendades toimub mingi tipupaari jaoks maksimaalne võimalik kauguseparanduste arv. Näidata see tipupaar, millel see kauguste paranemine niiviisi toimub, samuti algoritmi välimise tsükli käigus käsitletavate tippude järjekord ja kauguse muutumise protsess selle tipupaari jaoks. Ülejäänud tipupaaride jaoks kaugusi arvutada pole tarvis.

14.8. Joonistada 4-tipuline graaf, millele Floyd-Warshalli kauguste leidmise algoritmi rakendades toimub välimise tsükli igal sammul vähemalt ühe elemendi parandus kauguste tabelis.

14.9. Joonistada võimalikult väikese kaarte arvuga, vähemalt 6-tipuline graaf, milles kauguste leidmisel Floyd-Warshalli algoritmiga toimub välimise tsükli igal sammul vähemalt ühe kauguse parandus.

14.10. Programmeerida alljärgnevalt spetsifitseeritud funktsioonid.

```
def naabrusmaatriks(g):
#Antud: graaf g, mille igal kaarel (v,w) on pikkus c(v,w)>=0
#Tulemus: tagastatakse naabrusmaatriks ([1], lk 102)

def Floyd_Warshall(a): # ([1], joonis 6.11)
#Antud: graafi naabrusmaatriks a
#Tulemus: a on teisendatud  kauguste maatriksiks
```

14.11. Leida Dijkstra algoritmi ajalise keerukuse hinnang, kui frondi Q rollis kasutada järjestamata massiivi.

14.12. Milline invariant kehtib Dijkstra algoritmi rakendamisel enne kirje võtmist frondist Q?

14.13. Milline invariant kehtib Bellmann-Fordi algoritmi rakendamisel pärast välimise tsükli k-ndat sammu?

14.14. *(sv)* Graafis lühimas tees tipust a tippu b on k kaart. Kas võib väita, et Bellmann-Fordi algoritmi rakendamisel see lühim tee leitakse mitte enne kui k-ndal algoritmi välimise tsükli sammul?

14.15. *(s)* Kuidas Bellmann-Fordi algoritmi abil tuvastada, et graafis leidub negatiivse pikkusega tsükkel?

II Algoritmi valik

Käesolevas jaotises eeldatakse põhjendatud valiku tegemist järgmiste kaugusalgoritmide seast:

- Dijkstra algoritm (rakendatud iga tipu jaoks);
- Floyd-Warshalli algoritm;
- Bellmann-Fordi algoritm (rakendatud iga tipu jaoks).

Eelistuse leidmiseks võrrelda algoritmide ajalise keerukuse hinnanguid antud ülesande korral. Võimalusel arvestada elementaaroperatsioonide arvu hinnangu konstantidega.

14.16. *(v)* Graafis, milles on n tippu ja iga tipu väljundaste on

(a) $\left\lfloor \frac{n}{100} \right\rfloor$;

(b) 100

tuleb leida kõikide tipupaaride vahelised kaugused. Milline algoritm on eelistatuim, kui n võib olla kuitahes suur?

14.17. Graafi G kohta on teada, et selle igast tipust läheb kaar vähemalt $\frac{1}{4}$, kuid mitte rohkem kui $\frac{2}{3}$ ülejäänud tippudest. Milline algoritm on eelistatuim, kui on vaja leida kõikide tipupaaride vahelised kaugused ja graafi tippude arv võib olla kuitahes suur?

14.18. Milline algoritm on eelistatuim selleks, et suure tippude arvuga graafis leida kõigi tippude kaugused kahest fikseeritud tipust?

14.19. Sõnastada võimalikult efektiivne algoritm, mis etteantud orienteeritud, tsükliteta graafi etteantud tipu jaoks leiab pikimate teede pikkused kõigisse sellest tipust saavutatavatesse tippudesse.

Milline on selle algoritmi halvima juhu ajalise keerukuse hinnang $\Theta(f(m,n))$, kus f on võimalikult lihtne funktsioon, n on graafi tippude arv ja m kaarte arv?

14.20. *(s)* Sõnastada võimalikult efektiivne algoritm, mis etteantud orienteeritud, tsükliteta graafi etteantud tipu jaoks leiab pikima sellest tipust algava tee pikkuse.

14.21. Koostada programm leidmaks antud tugevalt sidusa orienteeritud graafi diameeter, st pikima otsetee ([1], lk 106) pikkus selle graafi kahe tipu vahel.

14.22. Olgu Eesti linnade plaan kujutatud orienteerimata graafina, milles tippudeks on linnad ja servadeks linnadevahelised maanteed. Serva märgendiks on vastava maantee pikkus (km). Koostada programm leidmaks

(a) millise kahe linna vahelisel lühimal teel on kõige rohkem linnu;

(b) millist linna läbib kõige rohkem kahe linna vahelisi lühimaid teid;

(c) selline linn, mille korral teistest linnadest kauguste (lühimate ühendusteede pikkuste) summa on kõige väiksem.

14.23. Olgu G n-tipuline tugevalt sidus orienteeritud graaf. Nimetame graafi tippu v graafi G *tsentriks*, kui ülejäänud tippude kauguste summa tipust v on minimaalne (tsentrit ei pruugi leiduda). Sõnastada võimalikult efektiivne algoritm leidmaks antud graafi G tsentrit, eeldusel et antud graafi iga tipu väljundaste on 100. Leida loodud algoritmi keskmise ajalise keerukuse Θ-hinnang (tippude arvu n suhtes).

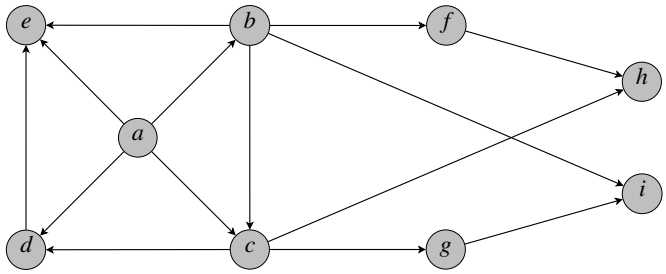
15. Eeldusgraaf

I Topoloogiline järjestus

15.1. Sooritada joonisel 34 kujutatud graafi tippude topoloogiline järjestamine

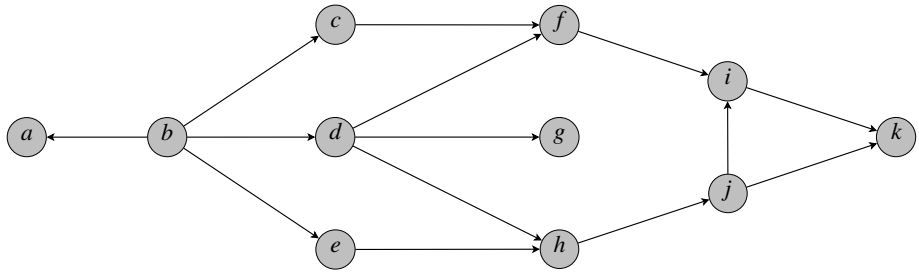
(a) lõppjärjestuse kaudu;

(b) Kahni algoritmiga.



Joonis 34: Graaf ülesannetele 15.1 ja 15.14.

15.2. Sooritada joonisel 35 kujutatud graafi tippude topoloogiline järjestamine Kahni algoritmiga.

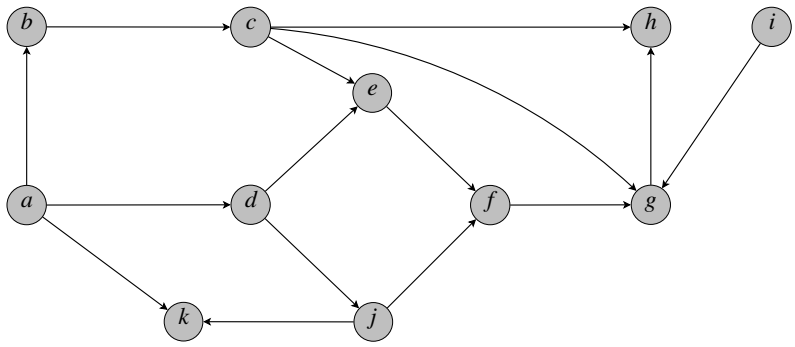


Joonis 35: Graaf ülesannetele 15.2 ja 15.15.

15.3. *(v)* Milline on vähim ja milline suurim võimalik n-tipulise tsükliteta graafi topoloogiliste järjestuste arv?

15.4. Leida joonisel 36 kujutatud graafi tippude kolm erinevat topolooglist järjestust.

15.5. Joonistada kolm erineva struktuuriga 5-tipulist graafi, millest igaühel on täpselt kaks erinevat topoloogilist järjestust.



Joonis 36: Graaf ülesandele 15.4.

15.6. *(v)* Olgu graafi G kaarteks $a \to b$, $c \to d$, $e \to f$, $g \to h$ ja $i \to j$. Leida G erinevate topoloogiliste järjestuste arv. Kas Kahni algoritmi sobivalt rakendades saame need kõik selle algoritmi abil ka leida?

15.7. Sõnastada võimalikult efektiivne algoritm, mis leiab etteantud tsükliteta graafi tippude kõikvõimalikud topoloogilised järjestused.

15.8. *(v)* Mis juhtub, kui Kahni algoritmi rakendada graafile, milles on orienteeritud tsükkel? Millisel algoritmi töö hetkel saame olla kindlad, et graafis leidub tsükkel?

15.9. *(s)* Sõnastada võimalikult efektiivne algoritm, mis etteantud graafi korral teeb kindlaks, kas selles leidub tsükkel või mitte.

15.10. *(v)* Kas võib väita, et kui tsüklit sisaldavale graafile rakendatud Kahni algoritm peatub, siis iga tipp, mille lisaväli pole 0, kuulub mingisse tsüklisse? Mida saab öelda tsüklite arvu kohta?

15.11. Kirjutada algoritm, mis leiab graafis antud tippu läbivate tsüklite seast lühima pikkuse.

15.12. Orienteeritud graafis G on kõik kaared erineva kaaluga. On teada, et tipud jaotuvad ühisosata mittetühjadesse alamhulkadesse $A_1, \ldots, A_{10}$ nii, et igast hulga A_i tipust läheb igasse hulga A_{i+1} tippu kaar (seda iga $i = 1, \ldots, 9$ jaoks) ning rohkem kaari graafis G ei ole. Kirjeldada võimalikult täpselt selle graafi kõikvõimalikud tippude topoloogilised järjestused.

15.13. Instituudi juhataja andis korralduse kõikide instituudis õpetatavate ainete vastutavatel õppejõududel esitada (iga) oma aine kohta selle eeldusainete täielik loetelu (st loetleda kõik ained, mis peavad olema läbitud enne seda ainet).

Koostada programm, mis

- teisendab õppejõududelt laekunud inforidade <aine, eeldusaine1, eeldusaine2, ... > hulga graafiks, kus tipp esindab ainet, kaar tipust w tippu v aga seost „aine w on eeldusaineks ainele v“;
- redutseerib selle graafi eeldusainete liiasusteta graafiks, tehes ühtlasi valiidsuse (tsüklite puudumise) kontrolli.

II Eeldusgraafi analüüs

15.14. Olgu joonisel 34 antud graaf tehtud eeldusgraafiks, määrates tippude läbimisajad järgmiselt:

a	b	c	d	e	f	g	h	i
3	5	5	7	6	8	3	5	1

.

Teostada eeldusgraafi analüüs.

15.15. Olgu joonisel 35 antud graaf tehtud eeldusgraafiks, määrates tippude läbimisajad järgmiselt:

a	b	c	d	e	f	g	h	i	j	k
2	1	4	3	9	1	5	2	2	1	1

.

Teostada eeldusgraafi analüüs.

15.16. Eeldusgraafis on tipud $a(5)$, $b(6)$, $c(8)$, $d(1)$, $e(1)$, $f(1)$, $g(2)$, $h(2)$, $i(4)$, $j(4)$, $k(3)$, $l(3)$, $m(5)$ (sulgudes tipu läbimisaeg) ning kaared $a \to c$, $b \to a$, $d \to b$, $d \to g$, $e \to f$, $e \to h$, $g \to e$, $g \to h$, $h \to f$, $f \to k$, $i \to d$, $i \to j$, $i \to m$, $j \to k$, $k \to m$, $l \to i$, $l \to k$, $l \to m$. Teostada eeldusgraafi analüüs.

15.17. *(v)* Olgu eeldusgraafis n tippu ja m kaart. Millised järgnevad hinnangud sobivad eeldusgraafi analüüsi ülesande ajalise keerukuse hinnanguks?

- $O(m+n)$;
- $\Theta(m+n)$;
- $O(n)$;
- $\Theta(n)$;
- $O(m)$;
- $\Theta(m)$?

15.18. Jukul on vaja ülikooli läbimiseks sooritada ained $a_1, a_2, \ldots, a_n$, osad neist on mingitele teistele eeldusaineteks. Nende ainete eeldusgraaf Q sisaldab parajasti tipud $a_1, \ldots, a_n$ ja kaared $a_i \to a_j$, kus aine a_i on ainele a_j eeldusaineks. Eeldame, et kõiki aineid loetakse igal semestril.

Sõnastada algoritm, mis etteantud Q korral leiab ühe sellise ainete läbimise semestrite kaupa, mille korral ülikooli stuudiumi läbimise aeg on vähim.

III Kriitilised tööd

15.19. Olgu antud üheksa tööd kaaludega $1, 2, \ldots, 9$. Joonistada kaks erinevat, kõiki neid töid sisaldavat eeldusgraafi, mis pole ahelad, aga kus kõik tööd on kriitilised. Ühes graafis olgu pikima tee pikkus võimalikult suur, teises võimalikult väike.

15.20. Olgu antud kümme tööd kaaludega $1, 2, \ldots, 10$. Koostada neist eeldusgraaf, millel on täpselt üks tipp sisendastmega 0, täpselt üks tipp väljundastmega 0 ning milles kriitilisi töid on võimalikult vähe.

15.21. Olgu antud kümme tööd kaaludega $1, 2, \ldots, 10$. Koostada neist eeldusgraaf, mis ei ole ahel ja kus kõik tipud on kriitilised tööd.

15.22. *(s)* Koostada eeldusgraaf, milles on tööd kaaludega 1, 2, 3, 4, 5, kõik tööd on kriitilised ja kus topoloogilisi järjestusi on võimalikult palju.

15.23. *(v)* Miks leidub eeldusgraafis alati vähemalt üks kriitiline tee ja milline see on?

15.24. *(v)* Kas alati, kui eeldusgraafi tipul leidub eellasi ja kõik vahetud eellased on kriitilised tööd, on ka see tipp ise kriitiline?

15.25. *(v)* Kas eeldusgraafis saab leiduda töö sisendastmega vähemalt 1, mille eellaste hulgas ei ole ükski töö kriitiline?

15.26. *(v)* Eeldusgraafis on kolm tippu sisendastmega 0 ja kolm tippu väljundastmega 0. Leida esimest liiki tipust teist liiki tippu viivate kriitiliste teede vähim võimalik arv.

16. Graafi toes

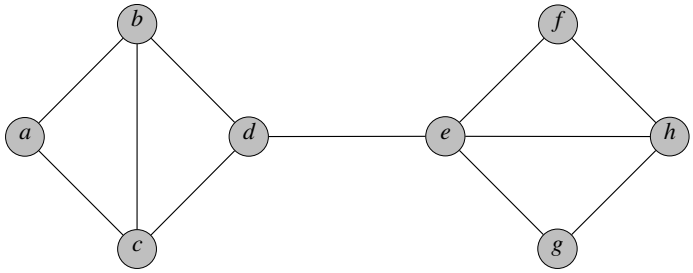
I Toes üldisemalt

16.1. Joonistada välja joonistel 37 ja 38 esitatud graafide toesed, mis saadakse graafi

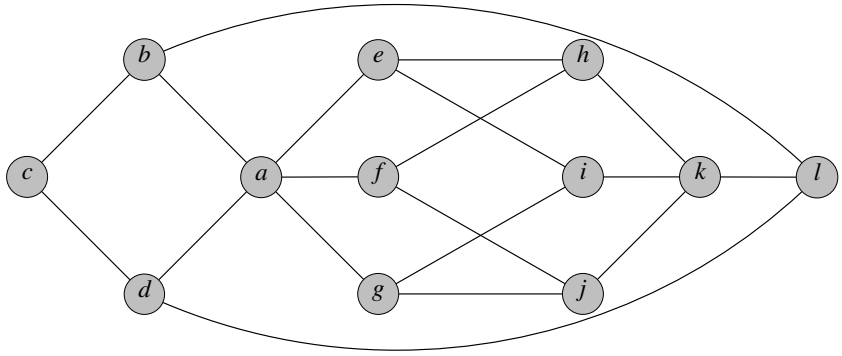
(a) laiuti läbimisel;

(b) sügavuti läbimisel,

võttes juurtipuks a.



Joonis 37: Graaf ülesannetele 16.1 ja 16.2.



Joonis 38: Graaf ülesandele 16.1.

16.2. *(v)* Mitu erinevat toest leidub joonisel 37 toodud graafil?

16.3. Tuua näide sidusast graafist, millel leidub täpselt 3 erinevat toest. Joonistada need toesed ja põhjendada lühidalt, miks rohkem ei ole.

16.4. Konstrueerida 5-tipuline graaf, millel on täpselt 3 erinevat minimaalse kaaluga toest. Joonistada need toesed.

16.5. Tuua näide kaalutud servadega, võimalikult väikese tippude arvuga graafist, kus toes, mis saadakse graafi laiuti läbimisel abistruktuurina järjekorda kasutades, ei tarvitse realiseerida kõiki lühimaid teid algustipust teistesse tippudesse.

16.6. Tuua näide 5-tipulisest kaalutud kaartega graafist, kus iga kahe tipu vahel leidub kaar (ühes või teises suunas) ja mille puhul graafi laiuti läbimisel saadav toes sisaldab lühimad teed algustipust kõigisse tippudesse.

16.7. *(v)* Sidusa graafi G kohta on teada, et selle kõik servad on paarikaupa erineva kaaluga. Kas võib kindlalt väita, et:

(a) vähima kaaluga serv kuulub G minimaalse kaaluga toesesse;

(b) suurima kaaluga serv ei kuulu G minimaalse kaaluga toesesse?

16.8. Olgu graaf G saadud mingi riigi linnadevaheliste teede graafist kui minimaalse kaaluga toes. Sõnastada võimalikult efektiivne algoritm, mis leiab graafis G kaks teineteisest kõige kaugemal asetsevat linna. Esitada ka vastav ajalise keerukuse hinnang.

16.9. *(s)* Sõnastada algoritm, mis etteantud sidusa graafi minimaalsest toesest leiab alglähendi rändkaupmehe ülesandele.

16.10. Olgu rändkaupmehe ülesande lahenduse S servade kogukaaluks T ja vaadeldava teedegraafi minimaalse kaaluga toese servade kogukaal K. Tõestada, et

$$K + k \leq T \leq 2 \cdot K,$$

kus k on suurim serva kaal graafis S.

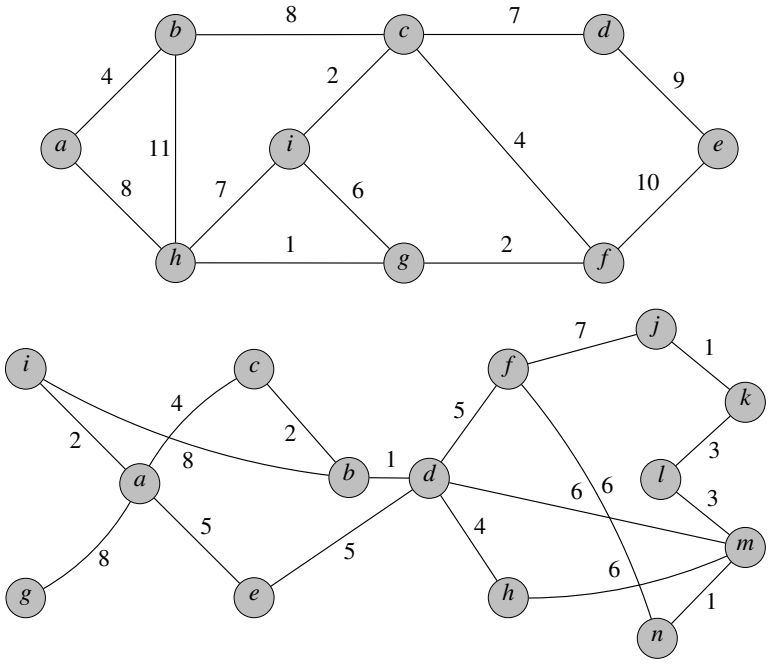
II Primi ja Kruskali algoritmid

16.11. Kas Primi algoritmi abistruktuurina sobib kasutada kahendkuhja või pöördkuhja? Sooritada joonisel 39 kujutatud graafide minimaalsete toeste leidmine Primi algoritmiga, kasutades abistruktuurina sellist kuhja. Näidata joonisel kuhja seis puu kujul pärast iga lisamist ja eemaldamist, ning algoritmi lõpptulemus.

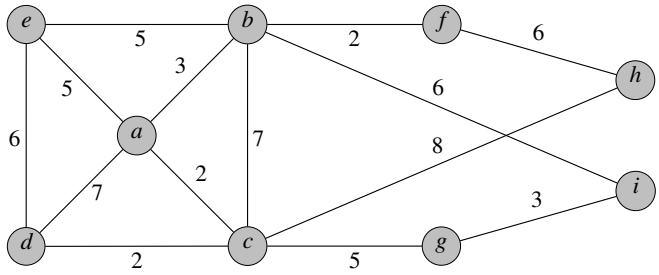
16.12. Leida iga joonistel 40 - 43 kujutatud graafi korral minimaalne toes Primi algoritmi kohaselt, kui lähtetipuks on a.

16.13. Tuua näide graafist, kus leidub tipp a, millest alustades annavad Dijkstra algoritm ja Primi algoritm erinevad toesed. Põhjendada, näidates ära algustipu a ja servade lisamise järjekorra kummagi algoritmi korral.

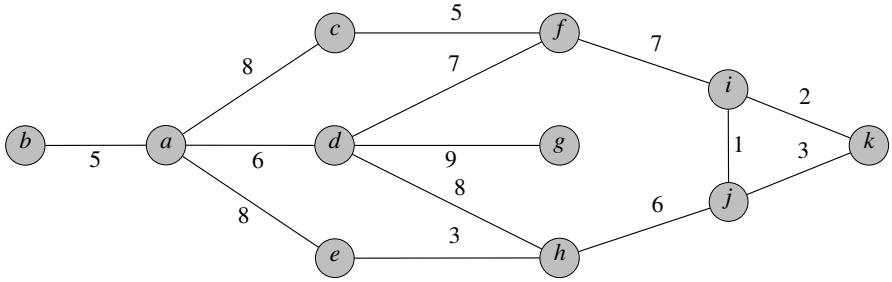
16.14. Leida joonisel 39 kujutatud graafide minimaalsed toesed Kruskali meetodil, kirjutades seejuures üles ka igal sammul tekkivad tipuklassid.



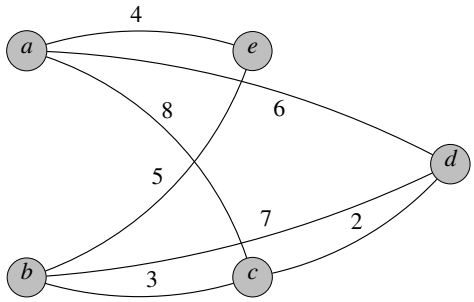
Joonis 39: Graafid ülesannetele 16.11 ja 16.14.



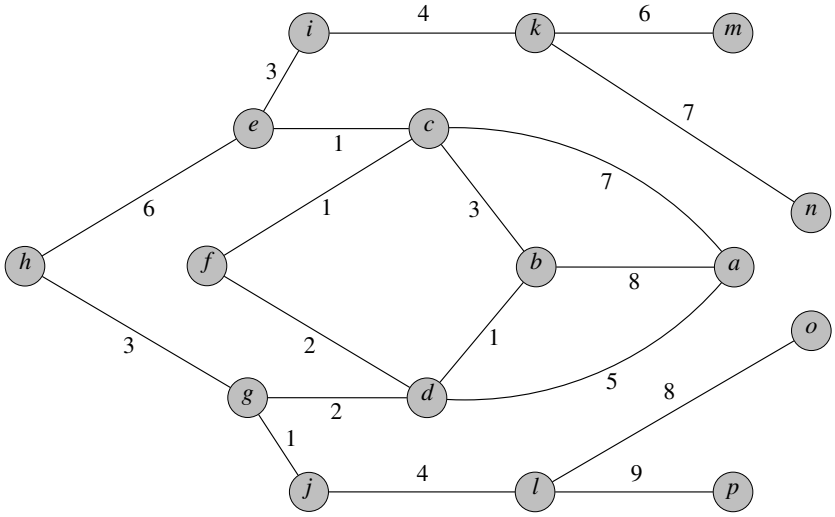
Joonis 40: Graaf ülesandele 16.12.



Joonis 41: Graaf ülesandele 16.12.

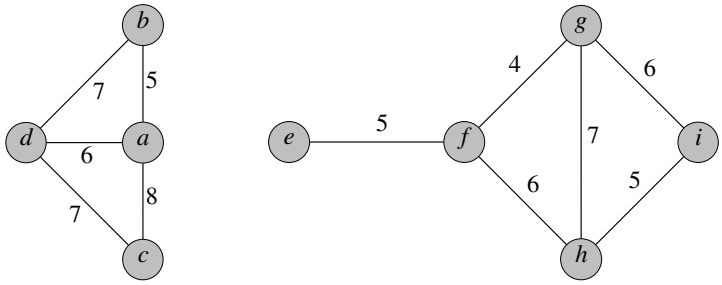


Joonis 42: Graaf ülesandele 16.12.



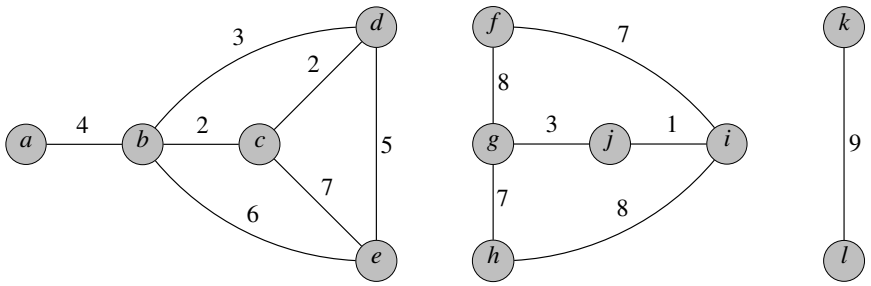
Joonis 43: Graaf ülesandele 16.12.

16.15. Leida joonisel 44 esitatud graafi minimaalne toes Kruskali meetodil. Näidata servade lisamise järjekord ja lõpptulemus.



Joonis 44: Graaf ülesandele 16.15.

16.16. Sooritada joonisel 45 esitatud graafi minimaalse toese leidmine Kruskali algoritmi kohaselt. Sidususkomponente hoida optimeeringuteta Galler-Fischeri meetodil. Näidata joonisel töö algseis ja seis koos klassipuudega iga serva lisamise järel.



Joonis 45: Graaf ülesandele 16.16.

16.17. Koostada programm, milles on

- realiseeritud Galler-Fischeri klassikäitluse meetod;
- realiseeritud Kruskali algoritm (tipuklasside käitlus – Galler-Fischeri meetodil);
- kirjeldatud testiosa.

16.18. Asulate plaan on antud orienteerimata graafina, milles tipud esindavad asulaid ja servad nendevahelisi ühendusteid. Tipu märgendiks on asula nimi, serva nimeks – ühendustee pikkus km. Koostada programm, mis leiab rännaku eeskirja kõigi asulate külastamiseks mööda selle graafi minimaalse toese servi, lähtudes etteantud lähteasulast.

16.19. Milline graafi läbimisel põhinev algoritm sobib kõige paremini sidusa graafi toese leidmiseks?

16.20. Joonistada 5-tipuline täisgraaf, mille kõik servad on paarikaupa erineva kaaluga ja millel annavad nii Primi (selles sobivalt algtippu valides) kui ka Kruskali algoritm sama toese sama kaarte valiku järjekorraga.

16.21. Lähtume ülesandes 8.12 toodud Eesti linnade vaheliste kauguste tabelist. Eeldame, et elektriliini maksimaalne pikkus kahe alajaama vahel, ilma et elektrikaod liiga suureks muutuksid, on d kilomeetrit. Alajaamad asuvad ainult tabelis nimetatud linnades. Koostada programm, mis leiab

(a) Primi algoritmiga;

(b) Kruskali algoritmiga

liinide vähima kogupikkuse, mida on vaja kogu Eesti „elektrifitseerimiseks“.

Selleks moodustada antud kauguste faili põhjal sobiv graaf ja rakendada sellele nõutud algoritmi. Ühtlasi leida vähim d väärtus, mille korral kõigi linnade (sidusasse) elekrivõrku ühendamine on veel võimalik.

16.22. *(s)* Olgu graafi G tippudeks ülesandes 8.12 toodud tabelis nimetatud linnad ja kaarte kaaludeks linnadevahelised kaugused selles tabelis.

Lähtudes G minimaalse kaaluga toesest, konstrueerida G jaoks lähend rändkaupmehe ülesandele. Katsetada teekonna algpunktidena kõikvõimalikke toese tippe.

16.23. *(s)* Koostada programm labürindi genereerimiseks antud mõõtmetega ristkülikusse, rakendades

(a) graafi sügavuti läbimist;

(b) juhuvalikuga Kruskali toese leidmist;

(c) juhuvalikuga Primi toese leidmist.

17. Varia

17.1. Sõnastada võimalikult efektiivne algoritm, mis saab sisendiks kaks järjestatud massiivi ja väljastab sisendmassiivide ühised elemendid. Võib eeldada, et kumbki sisendmassiiv ei sisalda korduvaid elemente.

17.2. Sõnastada võimalikult efektiivne algoritm, mis saab sisendiks rea veebiaadresse ja leiab, mitu saadud veebiaadressidest on unikaalsed (ei kordu). Leida selle algoritmi Θ-hinnang.

17.3. Millist andmestruktuuri sobib kõige paremini kasutada, kui vajalikud operatsioonid on kirje lisamine, vähima võtmega kirje eemaldamine ja kahe struktuuri ühendamine?

17.4. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: järjend a (listina).
Tulemus: info järjendi a elementide korduste kohta – listina, mille elemendiks on paar kujul (<element>,<selle elemendi korduste arv>).
Näiteks järjendi a = [9, 0, -2, 9, 8, 6, 0, 8, 9, -1] korral on tulemuseks list
[(0, 2), (6, 1), (8, 2), (9, 3), (-2, 1), (-1, 1)].

17.5. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: järjend a (listina).
Tulemus: list elementidega (n, vn), kus vn on järjendis a leiduvate (pikimate) võrdsetest elementidest koosnevate ennikute arv, $n > 1$, $vn > 0$. Näiteks järjendi [9, 0, 5, -2, 9, 8, 6, 0, 8, 5, 9, -1, 5, 4, -1, 5, 9] korral on tulemuseks list
[(2, 3), (4, 2)], st
võrdsete paare on 3 (nendeks on (0,0), (8,8) ja (-1,-1))
(võrdsete kolmikuid ei ole)
võrdsete nelikuid on 2 (nendeks on (9,9,9,9) ja (5,5,5,5)).

17.6. Programmeerida funktsioon järgmise ülesande lahendamiseks.
Antud: järjend a (listina) ja positiivne arv k.
Tulemus: järjend (list), kuhu on lisatud iga vähemalt k korda esineva elemendi korral paar (<elemendi väärtus> , <selle esinemisi>).
Näiteks, kui a = [9, 0, -2, 9, 8, 6, 0, 8, 9, -1], siis
$k = 2$ korral on tulemuseks [(0, 2), (8, 2), (9, 3)].
$k = 1$ korral on tulemuseks [(0, 2), (6, 1), (8, 2), (9, 3), (-2, 1), (-1, 1)].

17.7. Lõputööde kaitsmisele on lubatud 60 üliõpilast. Kaitsmine toimub neljas komisjonis kolmel päeval. Igal päeval tuleb igas komisjonis kaitsmisele 5 üliõpilase lõputöö. Leida, mitu võimalust on kaitsmiste ajakava koostamiseks. Ajakavas on iga komisjoni jaoks näidatud iga kaitsmispäeva kohta, millised 5 üliõpilast millises järjekorras kaitsevad oma lõputööd sellel päeval.

17.8. Lõputööde kaitsmisele on lubatud 60 üliõpilast. Kaitsmine toimub neljas komisjonis kolmel päeval. Igal päeval tuleb igas komisjonis kaitsmisele viie üliõpilase lõputöö. Ühe töö kaitsmiseks on ette nähtud 30 min. Võib eeldada, et ühel tööl on parajasti 1 juhendaja ja 1 retsensent.

Andmed:

Üliõpilane/töö[60]: nimi, töö pealkiri, töö spetsiaalsus, juhendaja, retsensent.

Komisjon[4]: number, liige[5], esimees, esimehe spetsiaalsus.

Koostada programm kaitsmiste ajakava koostamiseks, milles on järgitud alljärgnevaid nõudeid.

Ranged nõuded:

Kui töö retsensendiks on üks komisjoni esimeestest, siis see töö peab tulema kaitsmisele selle esimehe komisjonis. Samaaegseks ei tohi olla kavandatud nende tööde kaitsmine, millel on üks ja sama retsensent.

Soovitatavad nõuded (võimalikult palju neist võiksid olla rahuldatud, võimaluse korral), alaneva prioriteediga:

Kui töö spetsiaalsus langeb kokku mõne komisjoni esimehe spetsiaalsusega, siis see töö peaks tulema kaitsmisele selle esimehe komisjonis. Kui töö retsensendiks on üks komisjoni liikmetest, siis see töö peaks tulema kaitsmisele selles komisjonis. Kui töö juhendajaks on üks komisjoni liikmetest, siis see töö peaks tulema kaitsmisele selles komisjonis. Samaaegseks ei tohiks olla kavandatud nende tööde kaitsmine, millel on üks ja sama juhendaja.

Koostada lõputööde graafiku loomise programm.

17.9. Rahvastikuregistrisse kantud isikute kirjete läbivaatamisel saadi paaride järjend (<isiku nimi>, <elukoha asula nimetus>). Sõnastada algoritm leidmaks, mitu isikut elab igas asulas; st tulemuseks peab olema paaride järjend (<asula nimetus>, <isikute arv, kelle elukohaks see asula on>).

17.10. Sõnastada algoritm leidmaks suure hulga 4-baidiste (32-bitiste) „arvutisõnade“ seast

(a) kõik täiendsõnade paarid (<sõna>, <täiendsõna>), st kus <sõna> ja <täiendsõna> kõikides vastavates bittides on erinevad väärtused;

(b) kõik sõnade paarid (<sõna1>, <sõna2>), kus <sõna1> ja <sõna2> vastavates bittides on erinevaid väärtusi ülimalt kahel juhul.

17.11. Korvpalliliiga meistrivõistluste finaalis kohtuvad võistkonnad *A* ja *B*. Reglemendi kohaselt mängitakse omavahelisi mänge seni, kuni üks võistkond saab 4 võitu (tulles seega liiga meistriks). Koostada programm, mis

(a) väljastab kõikvõimalikud lõpptulemused; näiteks tulemus *AAABBA* vastab olukorrale, kus esimesed kolm mängu võitis *A*, seejärel *B* sai kaks võitu, kuid siis võitis jälle *A*;

(b) väljastab kõikvõimalikud lõpptulemused süstemaatiliselt (mingi kindla reegli kohaselt) järjestatuna.

17.12. *(s)* Tartu ühe gümnaasiumi abituriendid otsustasid tellida peotantsukursuse algajatele. Leiti juhendaja (tantsupedagoog), kes oli nõus selle kursuse läbi viima. Juhendaja nõudel tuli registreeruda (sega)paaride kaupa. Abituriendid koostasid kursuslaste nimekirja: n poissi ja n tütarlast. Kuid paaride moodustamisel ei jõutud omavahel kuidagi kokkuleppele. Matemaatikaõpetaja, kellelt nõu küsiti, ütles, et asi on väga lihtne: iga poiss hinnaku kõiki tütarlapsi numbritega 1, 2, ..., n (tantsupartnerina) meeldivuse järjekorras ja iga tütarlaps hinnaku kõiki poisse numbritega 1, 2, ..., n (tantsupartnerina) meeldivuse järjekorras. Sobivad paarid annab hulkade P ja T (poiste ja tütarlaste) selline üksühene vastavus, et iga kahe poisi p_1,p_2 ja neile vastavate tütarlaste t_1, t_2 korral

((poiss p_1 hindab tütarlast t_1 kõrgemalt kui tütarlast t_2) või
(tütarlaps t_2 hindab poissi p_2 kõrgemalt kui poissi p_1))

ja

((poiss p_2 hindab tütarlast t_2 kõrgemalt kui tütarlast t_1) või
(tütarlaps t_1 hindab poissi p_1 kõrgemalt kui poissi p_2)).

Pärast seda, kui kõik olid üles kirjutanud oma hinnangud, paluti abi kooli IT-juhilt, kes koostas vastava programmi, sisestas hinnangud ja printis programmi abil üksühese vastavuse paaride listina. Viimasega jäid kõik rahule ja peotantsukursus võis alata. Millise programmi oleksite Teie kirjutanud?

17.13. Rangelt kasvav järjend a sisaldab nii negatiivseid kui ka positiivseid elemente. Sõnastada algoritm leidmaks (kui võimalik) selline indeks i, mille korral $a_i = i$.

17.14. Koostada programm järgmise ülesande lahendamiseks. Olgu antud n-elemendiline järjend a ning indeksite $1,2,...,n$ permutatsioon p. Korraldada a (kohapeal) ümber vastavaks permutatsiooniks. Näiteks, kui a = [„koer“, „kana“, „lammas“] ning p = [3,1,2], siis a uueks väärtuseks saab [„lammas“, „koer“, „kana“]. Programm võib kasutada abimälu ainult konstantses mahus ($O(1)$).

17.15. Sõnastada algoritm leidmaks suurim antud erineva n arvu seast, kui võib kasutada ainult liitmist ja summade võrdlemist.

17.16. Antud on järjend, mis võib sisaldada nii täisarve kui ka (eestikeelseid) sõnu. Koostada programm sellise järjendi sorteerimiseks nii, et tulemusjärjendi arvudest koosnev alamjärjend oleks mittekahanevalt sorteeritud ja sõnedest koosnev alamjärjend oleks leksikograafiliselt sorteeritud. Lisaks peab olema rahuldatud tingimus: kui lähtejärjendi i-ndal kohal oli arv, siis ka tulemuse i-ndal kohal on arv, ning kui lähtejärjendi i-ndal kohal oli sõna, siis ka tulemuse i-ndal kohal on sõna.

Näide: $(aken, 20, -30, uks, 199, 11, katus) \rightarrow (aken, -30, 11, katus, 20, 199, uks)$.

17.17. Börsimaaklerite omavaheline suhtlusvõrgustik on kujutatud graafina, milles iga tipp esindab ühte maaklerit. Kaar tipust u tippu v märgendiga m näitab, et maakler u edastab info maaklerile v ajaga m minutit. Koostada programm leidmaks (kui võimalik) selline maakler (graafi tipp) w, et kui maaklerile w „sööta“

mingi info (nt kuuldus turgude seisust), siis see info levib kõikide teiste maakleriteni kõige lühema ajaga.

17.18. Selgitada, milline andmestruktuur oleks kohane Eesti linnade asukohaandmete (vt nt http://www.htg.tartu.ee/phys/linnad.html ; 24.08.2016) kujutamiseks, kui ülesandeks on leida antud geograafilisele punktile kõige lähem (linnulennult) linn.

17.19. *(s)* Mõnevõrra kulunud metallketi iga lüli jaoks on leitud selle tugevus (arvulise väärtusena). Ketist on taaskasutuse eesmärgil tarvis välja lõigata pikim osa, milles nõrgima lüli tugevus on suurem kui teatav kriitiline ühe lüli tugevus.

Sõnastada efektiivne algoritm, leidmaks (võimaluse korral) väljalõigatava osa alguslüli ja lõpulüli indeksid, kui on antud mingi keti lülide tugevuste järjend ja kriitiline lüli tugevus.

17.20. Algarvude hulgas on täheldatavad nn algarvuringid. *Algarvuring* on kümnendsüsteemis ühepikkuste algarvude järjend $x_1, x_2, \ldots, x_n$, $n > 2$, milles

1) x_{i+1} on saadud arvust x_i selle viimase kümnendnumbri lõpust alguses-se ümberpaigutamise teel ($i = 1, 2, \ldots n-1$);

2) $x_n = x_1$.

Kaks näiteringi: $[971, 197, 719, 971]$ ja $[3119, 9311, 1931, 1193, 3119]$.
Algarvuringi *esindajaks* loeme tema vähima elemendi; ülaltoodud näiteringide puhul siis vastavalt arvud 197 ja 1193. Loomulik on ühe ja sama esindajaga algarvuringid lugeda samasteks; seega esindajale vastab parajasti üks algarvuring.

Koostada programm järgmise ülesande lahendamiseks.

Antud: arv a.

Tulemus: leitud viis algarvuringi arvust a suuremate algarvude hulgas; vastus esitada esindajate kasvava järjendina.

Näide: $a = 16$, vastus $[17, 37, 79, 113, 337]$.

18. Eriteemad

I Sõnetöötlus

18.1. Leida prefiksfunktsiooni ([1], lk 78) väärtused sõne $s = 'aaaaab'$ jaoks.

18.2. Konstrueerida sõne, mille prefiksfunktsiooni väärtusteks on

0 1 0 1 2 3 0 1 2 3 4 0.

18.3. *(s)* Loendada, mitu korda võrreldakse sümboleid sõne
$s = 'aabcaabcd'$
esinemise otsimisel tekstis
$t = 'aabcaabcaaabcaabcdbaaaaaabab'$

(a) Knuth-Morris-Pratti algoritmis (prefiksfunktsiooni leidmist arvestamata);

(b) lihtsas otsimisalgoritmis ([1], joonis 5.1).

18.4. Antud sõnepaaride s ja t jaoks konstrueerida maatriks c vastavalt algoritmile *kavandada* ([1], lk. 90). Kriipsutada alla c elemendid, mis vaadatakse läbi nende sõnede pikima ühissõne leimisel.

(a) $s = 'aabcbdba'$; $t = 'abbdcbba'$;

(b) $s = 'aabcbdbaa'$; $t = 'abbdcbba'$;

(c) $s = 'aabcbdbaab'$; $t = 'abbdcbab'$;

(d) Sõne s saamiseks kirjutage oma ees- ja perekonnanimi järjestikku, ilma tühikuteta ja ainult väiketähtedega, ning võtke sellest 10 esimest tähte. Kui tähti oli vähem kui 10, siis lisage lõppu vajalik arv algusest võetud tähti. Näiteks, *Jüri Kiho* jaoks s = *'jürikihojü'*. Sõne t saate sõne s transponeerimisel (sõne s tähed vastupidises järjekorras). Näiteks, *Jüri Kiho* jaoks t = *'üjohikirüj'*.

18.5. Programmeerida funktsioon kontrollimaks, kas antud sõna on teise antud sõna anagramm.

18.6. Eestikeelsest tekstikorpusest on välja eraldatud kõigi viietäheliste sõnade hulk. Koostada programm, mis leiab sellest kõik anagrammide alamhulgad. Näiteks, kaks võimalikku alamhulka:
{kaust, kutsa, skaut, tasku, takus, tuska, tukas};
{ketas, kaste, katse, kesta, teaks, teksa}.

18.7. Täht-number mõistatus antakse valemina, milles erinevad suurtähed tähistavad erinevaid numbreid. Tuleb leida, milline täht millist numbrit tähistab. Näiteks

(a) K x K + Y = YZ (on mitmeid lahendeid, üks neist: K=6, Y=3, Z=9);

(b) PI x R x R = AREA (1 lahend);

(c) GAIUS + JULIUS = CAESAR (3 lahendit).

Koostada programm etteantud täht-number mõistatuse kõigi lahendite leidmiseks.

18.8. Koostada programm järgmise ülesande lahendamiseks.
Antud on n indiviidi ühepikkused genoomijärjestused (sõnedena).
Leida kõige väiksema Hammingu kaugusega (`https://en.wikipedia.org/wiki/Hamming_distance` 24.08.2016) indiviidide paar.

18.9. *(s)* Keelehuviline Valdur töötas välja järgmised definitsioonid.
Sõna *S valdur-tüveks* ehk *v-tüveks* nimetame sõna S lühimat mittetühja alamsõna S' (järjestikust S osa), milles ei ole ühtegi unikaalset (alamsõna S' piires) sümbolit ja (paarisarvulise pikkusega S' korral) sõna S' üks pool ei ole teise poole anagramm. Kaugeltki mitte igal sõnal ei leidu v-tüve. Sõna, millel on vähemalt üks v-tüvi, nimetame *v-sõnaks*.
Defineeritud mõisted on üldiselt rakendatavad keelest sõltumata. Näiteid v-sõnadest (poolpaksuna esile tõstetud v-tüvedega):
n**uumama**ta, r**ahaah**nuse, ongelma**llisis**ta (soome k.), emb**arrass**ing (ingl. k.), s**trutture** (it. k.), verteidigungsminis**tertreff**ensmittelm (saksa k.).
Mõnel v-sõnal võib olla mitu v-tüve, nt k**elle**legi, ke**llele**gi. Valdur pani ka tähele, et soomekeelsed tekstid paistavad silma v-sõnade erilise rohkuse poolest.
Programmeerida funktsioon

(a) antud sõna kõigi (erinevate) v-tüvede leidmiseks;

(b) v-sõnade arvu leidmiseks antud tekstis.

II Planimeetria

18.10. Kirjutada algoritmid kontrollimaks punkti kuuluvust

(a) kolmnurka;

(b) nelinurka.

18.11. Eeldusel, et on olemas algoritm punkti kolmnurka kuulumise kontrollimiseks $kuulubKolmnurka(punkt, A, B, C)$, kus A, B, C on kolmnurga tipupunktid,

(a) kirjutada algoritm punkti nelinurka kuulumise kontrollimiseks (nelinurk ei pruugi olla kumer);

(b) kirjutada lineaarse keerukusega algoritm punkti kumerasse hulknurka kuulumise kontrollimiseks.

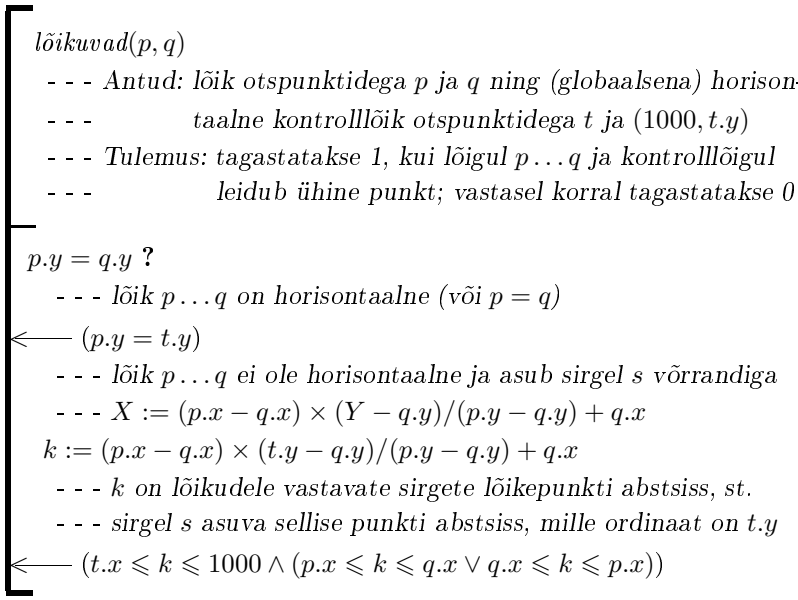
18.12. Kirjutada algoritm järgmise ülesande lahendamiseks.
Antud on neli punkti $p1$, $p2$, $p3$ ja $p4$. Analüüsida tuleb kinnist murdjoont $M = p1$-$p2$-$p3$-$p4$-$p1$.
Tulemus: tagastatakse

0, kui M ei ole nelinurk;
1, kui M on mittekumer nelinurk;
2, kui M on trapets;
3, kui M on rööpkülik.

18.13. Kirjutada lineaarse keerukusega algoritm, mis antud punktihulgast elimineerib kumerasse kattesse mittekuuluvad punktid.

18.14. Kirjutada detailne algoritm kontrollimaks punkti kuuluvust hulknurka, kui hulknurga tipud on antud ringahelana (loomulikus järjekorras).

18.15. Olgu punkti hulknurka kuulumise praktilise ülesande korral teada, et vaadeldavate punktide koordinaadid ei ole absoluutväärtuselt suuremad kui 999. Kas algoritmis *punkt_hulknurgas* ([1], lk. 117) võib siis alamalgoritmi *lõikumine* asemel kasutada joonisel 46 toodud algoritmi *lõikuvad*?



Joonis 46: Lõikumise erikontroll.

18.16. *(s)* Tasandil paikneb mingi arv vertikaalseid lõike. Ütleme, et kaks lõiku on *silmsides*, kui neid saab ühendada horisontaalse lõiguga, mis ei lõiku ühegagi ülejäänud vertikaalsetest lõikudest. Koostada programm, mis leiab antud vertikaalsete lõikude hulgast kõik sellised lõikude kolmikud, milles iga kaks liiget on omavahel silmsides.

18.17. *(s)* Tasandil paikneb ristkülik R, mille küljed on paralleelsed koordinaattelgedega. Ristkülikus on märgitud mingi arv punkte. Koostada programm leidmaks suurima pindalaga ristkülik R', nii et

- R' küljed on paralleelsed ristküliku R külgedega;
- R' paikneb ristküliku R sees;
- R' ei sisalda (oma sisealas) ühtegi märgitud punktidest.

III Algoritmi korrektsus

Käesolevas jaotises esitatud ülesannete lahendamiseks on tarvilik tunda algoritmi korrektsuse tõestamise põhimõtteid ([1], ptk 8).

18.18. *(s)* Leida omistamisdirektiivi nõrgim eeltingimus P joonisel 47.

$$
\begin{array}{|l}
\hline
\quad \text{- - -}\ P \\
x := x^2 - y \\
\quad \text{- - -}\ x < y \\
\hline
\end{array}
$$

Joonis 47: Omistamisdirektiiv.

18.19. *(s)* Tõestada alljärgnevate algoritmide (a) – (e) korrektsus.

(a) Lihtmurdude summa:

$$
\begin{array}{|l}
\hline
\quad \text{- - -}\ n > 5 \\
s := 0.5 \\
k := 2 \\
\quad \begin{array}{|l}
\hline
k < n\ \textbf{?} \\
\quad m := k + 1 \\
\quad s := s + \dfrac{1}{km} \\
\quad k := m \\
\hline
\end{array} \\
\quad \text{- - -}\ s = 1 - \dfrac{1}{n} \\
\hline
\end{array}
$$

(b) Lineaarne algoritm:

- - - $\sqrt[3]{3} \leqslant n^m \leqslant \sqrt[2]{3}$
$m := 1 - m$
$n := n + 1$
$\alpha := 1 - 2n$
$b := 1 + \cos^2 \alpha$
$b := b - \log_3 (n^2 + \alpha) + sin^2\alpha - 2$
$x := \log_3 (n^2 + \alpha)^2$
$y := mb$
$x := x + 2y$
$x := x/2$
- - - $0,5 \leqslant x \leqslant 1$

(c) Lineaarne algoritm:

- - - $0,5 \leqslant a \leqslant 0,7$ *ja*
- - - $0,5 \leqslant \log_3 n \leqslant 0,7$
$n := n + 1$
$\alpha := 1 - 2n$
$a := 1 - a$
$b := 2 + \cos^2 \alpha$
$b := b - \log_3 (n^2 + \alpha) + sin^2\alpha - 3$
$x := \log_3 (n^2 + \alpha)^2$
$y := ab$
$x := x + 2y$
$x := x/2$
- - - $0,5 \leqslant x \leqslant 1$

(d) Faktoriaalide jagatis:

- - - $0 \leqslant k < n$
$m := k + 1$
$t := m + 1$
$t \leqslant n$ **?**
$m := m * t$
$t := t + 1$
- - - $m = \frac{n!}{k!}$

(e) Suurem faktoriaal:

- - - $n > 5$
$m := 6$
$t := 7$
$t \leqslant n$ **?**
$m := m * t$
$t := t + 1$
$m := m \times 120$
- - - $m = n!$

19. Lisa: täiendavaid mõisteid

alusstruktuur – abstraktsema andmestruktuuri kaudseks mäluesituseks kasutatav konkreetsem struktuur. Nt klassikalise kahendkuhja puhul on alusstruktuuriks tõkestamata massiiv, tõkestamata massiiv esitatakse omakorda tavalise massiivi abil (st alusstruktuur on tavaline massiiv) jne.

alusmassiiv – vt alusstruktuur.

eeldusgraafi analüüs – eeldusgraafi iga tipu varaseima võimaliku lõpuaja ja hiliseima lubatava algusaja ning kriitiliste tippude algoritmiline tuvastamine.

GRAAF

- *indutseeritud alamgraaf* – graafi alamgraaf, mis sisaldab selle graafi kõik need kaared, mille algus- ja lõpptipp on selles alamgraafis;
- *kaalutud (servadega/kaartega) graaf* – graaf, mille servadel/kaartel on (arvulised) kaalud/pikkused vmt;
- *nõrgalt sidus graaf* – orienteeritud graaf, mille kaarte asendamisel servadega tekib sidus suunamata graaf
- *nõrgalt sidus komponent* – orienteeritud graafi selline nõrgalt sidus mittetühi indutseeritud alamgraaf, mis ei sisaldu üheski suuremas nõrgalt sidusas alamgraafis;
- *tugevalt sidus graaf* – orienteeritud graaf, milles leidub (suunatud) tee igast tipust igasse tippu;
- *tugevalt sidus komponent* – orienteeritud graafi selline tugevalt sidus mittetühi indutseeritud alamgraaf, mis ei sisaldu üheski suuremas tugevalt sidusas alamgraafis;

generaator-funktsioon – programne vahend rea väärtuste (objektide) järjestikku ükshaaval genereerimiseks.

generaator-funktsioon, Python – funktsioon, mis rakendamisel konstrueerib ja annab välja (*yield*) järjekordse väärtuse reas. Tüüpiline kasutamine:

```
for v in gen:  # kus gen on generaator-funktsioon
    # muutujale v on omistatud järjekordne väärtus
    ...
```

generaator-funktsioon, Java (üks võimalusi) – parameetriteta isendimeetod (tavaliselt nimega *next*) generaatori klassis, näiteks klassis nimega Gen. Viimases defineeritakse veel parameetriteta isendimeetod *hasNext*, ja ka isendiväli jooksva oleku salvestamiseks.

Kasutamise näide:

```
Gen gen = new Gen(...);
...
while(gen.hasNext()){
   Object v = gen.next();
   // muutujale v on omistatud järjekordne väärtus
```

```
        //   gen.next() konstrueerib ja tagastab (return)
        //   järjekordse väärtuse ning salvestab jooksva oleku
        //   isendi gen vastavale väljale

          ...
    }
```

Kahni algoritm – õpikus ([1], ptk 6.1) kirjeldatud algoritm graafi tippude topoloogiliseks järjestamiseks.

KLASSID

- *kõrgusoptimeering* – reegel, mille järgi kahe erineva kõrgusega klassipuu ühendamisel pannakse alati madalam puu kõrgema alampuuks;
- *topeltoptimeering* – kõrgusoptimeering teede õgvendamisega Galler-Fischeri meetodi puhul, kus klassipuude kõrguse rollis kasutatakse pseudokõrgust;
- *pseudokõrgus* – klassipuuga seonduv efektiivselt arvutatav kunstlik atribuut, mille väärtus muutub teede õgvendamisega klassipuude ühendamisel samamoodi, nagu ilma õgvendamiseta klassipuude ühendamisel puu kõrgus;

Kosaraju algoritm – graafi tippude lõppjärjestuses läbimisel põhinev algoritm tugevalt sidusate komponentide leidmiseks. Algoritm rakendab iteratiivselt järgmist sidusa komponendi leidmise sammu: leida lõppjärjestuses viimane tipp u, mida pole veel sidusasse komponenti paigutatud, ning luua uus sidus komponent, kuhu kuulub tipp u koos kõigi nende veel komponentidesse paigutamata tippudega, millest u on graafis saavutatav.

parem raag – kahendpuu juurest algav ahel, mille iga järgmine tipp on jooksva tipu parem alluv ning viimasel tipul parem alluv puudub.

põimemeetod, optimeeritud variant – järjestamise põimemeetodi alt-üles variant, kus kahekaupa põimimise algseisuks võetakse järjendi juba alguses järjestatud segmentide list.

põimemeetod, alt-üles variant – järjestamise põimemeetodi variant, kus kõik järjendi jagamised osadeks on programmeeritud töö algul tehtava eraldi etapina, sellele järgneb saadud osade puukujuline kahekaupa kokkupõimimine. Massiivil realiseeritakse alt-üles variant tavaliselt tsüklitega, erinevalt klassikalisest rekursiivsest põimemeetodi definitsioonist. Järjendi osadeksjagamise võib teha triviaalselt, lugedes iga elemendi eraldi osaks. Vt ka põimemeetod, optimeeritud variant.

rändkaupmehe ülesanne – ülesanne leida antud kaalutud servadega täisgraafis lühim (vähima kaalude summaga) kinnine tee, mis läbib kõik tipud.

sadulameetodil otsing – efektiivne meetod järjestatud ridade ja veergudega kahemõõtmelises tabelis antud võtme esinemiste otsimiseks. Alustades tabeli nurgast, milles oleva kirje võti on ühes dimensioonis suurim ja teises vähim, otsib mööda rida või veergu (kasutades ühemõõtmelise otsingu meetodeid) vastavalt sellele, kummas suunas liikudes kirjete võtmete erinevus otsitavast väheneb. Punktis, kus uuritava reas (veeru) võtmete erinevus otsitavast hakkab jälle suu-

renema, muudetakse suunda ja toimitakse samamoodi edasi kuni otsitava võtme leidmise või tabeli piirest väljumiseni. Sama meetod rakendub nt kahe muutuja funktsiooni kindla väärtuse otsingul, kui funktsioon on mõlema argumendi järgi kasvav.

topeltkahendotsing – kahendotsing, millele eelnevalt otsitakse lõigu korduva topeldamise teel lõik, mis sisaldab otsitava kõik võimalikud asukohad. Kahendotsing sooritatakse topeldamisetapil leitud lõigus.

tõkestamata massiiv – massiiv, kuhu on võimalik elemente lisada ja kust elemente eemaldada. Realiseeritakse tavalise massiivi (nn alusmassiiv) ja täisarvulise lisaväljaga, mis näitab, mitu elementi on massiivi salvestatud (see võib olla massiivi pikkusest väiksem, ülejäänud elemendikohad loetakse tühjaks). Kui massiivi mahutavus (pikkus) ei luba uusi elemente lisada, asendatakse alusmassiiv (tavaliselt kaks korda) suuremaga ja kopeeritakse kõik elemendid sinna üle. Analoogselt, massiivi elementide arvu vähenemisel asendatakse alusmassiiv järjest väiksemaga kindla seaduspära alusel.

vasakkalduv puu – kahendpuu, mille iga mittetühja alampuu vasaku haru kaal on vähemalt niisama suur kui parema haru kaal. Alampuu kaaluna võib kasutada tippude arvu või parema rao pikkust.

vasakkalduv kuhi – kuhi, mille alusstruktuuriks on vasakkalduvad puud. Sarnaselt binomiaalkuhjadega defineeritakse kõik vasakkalduvate kuhjade operatsioonid kahe kuhja ühendamisoperatsiooni kaudu: kirje lisamiseks vasakkalduvasse kuhja ühendatakse ainult seda kirjet sisaldav ühetipuline vasakkalduv kuhi olemasoleva kuhjaga; vähima võtmega kirje eemaldamiseks eemaldatakse juurtipp ning ühendatakse vasak ja parem haru, tagastatakse juurtipu kirje. Kahe vasakkalduva kuhja ühendamiseks põimitakse nende puude paremad raod koos vasakute harudega võtme väärtuse alusel (säilitamaks kuhjatingimust), misjärel taastatakse vasakkalduva puu struktuur, liikudes suunaga alt üles ja vahetades vasakud ja paremad harud kõigis parema rao tippudest lähtuvates alampuudes, kus vasaku haru kaal on parema haru kaalust väiksem.

Suunised

1. Funktsiooni asümptootiline hinnang

1.3. Tõestuse teises pooles oletada vastuväiteliselt, et leidub selline konstant. **1.4.** (e) Vt Stirlingi valem ([1], lk 71).

2. Algoritmi ajaline keerukus

2.2. Uurida väljastamisi parameetri n väärtuste 8, 9, 15 ja 16 korral. Antud funktsioonide (a), (b) ja (c) täitmise ajalise keerukuse hinnangud on vastavalt $\Theta(\log n)$, $\Theta(n)$ ja $\Theta(n \log n)$. **2.10.** Θ-hinnang peab kehtima alates mingist naturaalarvust N. **2.21.** Parima algoritmi halvima juhu ajaline keerukus on $\Theta(n \log n)$. Paisktabelit kasutades saab praktikas seda kontrolli teha $\Theta(n)$ ajaga. **2.23.** Vt põhiteoreem ([1], lk 20). **2.26.** Fikseerida elementaaroperatsioonid ja seejärel leida nende koguarv. **2.27.** Halvima ja parima juhu ajalise keerukuse hinnangud langevad kokku (ega ole $O(1)$). **2.28.** Kui n fikseerida, siis algoritmi täitmisel sama n jaoks saame alati sama sammude arvu. **2.29.** Kui elemendi A_{ij} töötlemise aeg on $\Theta(f(i,j))$, siis see tähendab, et selle elemendi töötlemise aeg jääb vahemikku $c_1 f(i,j) \ldots c_2 f(i,j)$, kus c_1 ja c_2 on mingid konstandid ja $c_1 < c_2$. **2.30.** (b) Vaid üks õige vastus. (c) Viis õiget vastust. **2.31.** Leida järjestikuste Fibonacci arvude F_n ja F_{n+1} kümnendkohtade arvu sõltuvus indeksist n. **2.34.** (c), (d) ja (e): vt põhiteoreem ([1], lk 20). **2.40.** (a) Θ-hinnang selle programmi käitamisele ei ole $\Theta(2^n)$. (b) Astendamise teel. **2.41.** Funktsiooni täitmise ajalise keerukuse hinnanguks on $\Theta(n \cdot 4^n)$. **2.42.** Funktsiooni täitmise ajalise keerukuse hinnanguks on $\Theta(n \cdot n!)$.

3. Hargnemistega algoritm. Rekursioon

3.8. (b) Selle funktsiooni ajaline keerukus on $\Theta(n!)$, kus n on a elementide arv. **3.15.** (a) Binaarne rekursioon. **3.17.** Rekursiooni väljakutsete puu hargneb vastavalt sellele, kummast sõnejärjendist võetakse esimene element ära.

4. Variantide läbivaatamine

4.4. Võimalik, kuid ebaefektiivne oleks luua $2n$-elemendiline järjend, milles algse järjendi iga elementi on täpselt kaks eksemplari, ja seejärel sooritada saadud järjendi kõikide variantide läbivaatust. **4.5.** Trinaarne rekursioon. NB!Tekitades 52-elemendilise toodete järjendi ja valides sellest, ei saa me kõiki valikuid ühekordselt. **4.7.** Leidnud mingi sobiva valiku, peame üldjuhul selles rekursiooni harus otsingut jätkama. **4.8.** Seda leidev rekursiivne funktsioon peab tagastama selles harus leiduvate sobivate komplektide arvu. **4.9.** Ülesannet saab lahendada ka teatavate maatriksite astendamise teel. **4.11.** Rekursiivne funktsioon $ast(n)$: Antud: trepi astmete arv n, $n > 1$. Tulemus: tagastatakse sõnede järjend, milles iga sõne on üks ülesminemise moodus, koosnedes sümbolitest 1, 2 ja 3 (tähendustes – samm vastavalt kas järgmisele, ülejärgmisele või üle-ülejärgmisele astmele).

Baasjuht: kui $n = 2$, siis ülesminekumoodusteks on "11" ja "2". Üldjuht – annab astmele nr n ülesminekumoodused : [Iga $v \in ast(n-1)$ korral: $/v$ on üks astmele nr $n-1$ ülesminekumoodus/ 1) ühe mooduse saame, lisades sõnele v sümboli 1; 2) veel ühe mooduse saame, kui asendame sõnes v viimase sümboli: kui see oli 1, siis sümboliga 2; kui see oli 2, siis sümboliga 3.]. Ülesminekumooduste arv on funktsiooni $ast(n)$ poolt tagastatud järjendi pikkus. **4.13.** Uurida, mitmel viisil saab osarivi otspunkte määrata. Osarivi on määratud esimese elemendi ja viimase elemendi indeksitega kogu massiivis (õpilaste rivis). **4.17.** Üks võimalus – neljakordse tsükliga. **4.21.** Iga kombinatsioon n-elemendilisest järjendist k kaupa on määratud järjendi ühe maskiga; maskideks on kõikvõimalikud bitijärjendid pikkusega n, milles igaühes parajasti k ühte.

5. Otsimisalgoritmid järjenditel

5.2. Kahendotsingu algoritmis võrreldakse (võrduse mõttes) elemendi väärtust otsitavaga alles siis, kui otsinguakna suuruseks on 1. **5.5.** (a) Topeltkahendotsingu akna parema piiri rollis on järgepidi 11, 16, 32 ja 56. Aknas $[34,\ldots,53]$ sooritame tavalise kahendotsingu. **5.8.** Võimalik on see neist kahes. **5.11.** Kahendotsingut saab kasutada vaid siis, kui otsinguakna raamid (mõlemad ääred) on teada.

6. Järjendi ümberkorraldamine

6.2. Ümberpaigutamise tulemusel peab massiivi esimesel viiel kohal olema alammassiiv $[18,19,16,17,20]$. Terve massiivi lihtsalt sorteerimine ei ole kõige efektiivsem. **6.3.** Ümberpaigutamise tulemusel peab massiivi viimasel seitsmel kohal olema alammassiiv $[31,32,37,39,36,38,35]$. **6.14.** (d) Aitab juhuslikkus. **6.30.** Esimesest ajamõõtmise tulemusest määrata konstant. **6.37.** Vaadelda, millistel juhtudel nimetatud meetodites alamjärjendeid enam osadeks ei jagata. **6.39.** Väljakutsete magasinis hoitakse ka neid katkestuskohti, mille korral selles harus enam sisulist tööd ei tehta. **6.42.** Alati realiseerub kiirmeetodi halvim juht.

7. Paisksalvestus

7.1. Sobivad vaid need, mis võimaldavad kõik read läbi käia. **7.13.** Algul proovida leida, mitmel viisil on võimalik x kirjet paigutada tabelisse nii, et ei tekiks ühtegi põrget. Sealjuures tõenäosuse arvutamisel arvestada, et kirje võib sattuda suvalisse tabeli ritta. **7.16.** (a) Lihtsama paiskfunktsiooni saamiseks määrata: (a) $k \in [0,10)$ ja $h(k) = \lfloor 9k/10 \rfloor$; (b) $k \in [-3,2)$ ja $h(k) = \lfloor 2k+3 \rfloor$. **7.19.** Erinevaid perenime esitähti on mingi konstatne arv. **7.21.** Igasse kimpu peaks sattuma keskeltläbi võrdne arv kirjeid. **7.33.** Leidub algoritm, mis siinkohal töötab ajalise keerukusega $\Theta(n \log n)$. **7.34.** Leida nimetatud meetodite halvad juhud. **7.36.** Algoritm, mis alustab järjendi läbimist lõpust. **7.39.** Leidub mitmeid lineaarse keerukusega algoritme. **7.46.** Algoritmi töökiiruse võtmekohaks on sama väärtuse eespool esinemise kontrollimine. **7.47.** Piisab eelnevate elementide võimalikult efektiivselt meelespidamisest. **7.48.** Kui elemendid paisata ühtlaselt kimpudesse, siis leida,

millistel positsioonidel saavad asuda suuruse poolest lähimad elemendid. **7.49.** Salvestades paisktabelisse kirjet võtmega m, kontrollida, kas tabelis juba leidub kirje võtmega $n-m$. **7.50.** Tuleb võimalikult efektiivselt kontrollida, kas vaadeldav element on juba varem esinenud.

8. Magasin ja järjekord

8.8. Rekursiooniga seda efektiivselt lahendada ei saa. **8.9.** (a) Magasini sisuks võtta selliste lippude asukohad oma horisontaalil, mis pole omavahel tules; seejärel proovida lisada järgmist. **8.10.** Nn Josephuse probleemi saab lihtsasti lahendada, kujutades rea järjekorrana, millest igal sammul esimesed $k-1$ viiakse järjekorra algusest järjekorra lõppu, k-s aga lihtsalt eemaldatakse. **8.12.** Leida kõik alglinnast kaugusel 1 olevad linnad, seejärel kaugusel 2 olevad jne. **8.13.** Piisab järjendi a analüüsimisest järjendit b silmas pidades.

9. Puu ja kahendpuu

9.1. (b) Järgmise taseme moodustavad jooksva taseme tippude alluvad. **9.6.** Iga ülesande lahenduseks on rekursiivne protseduur, millel sisendparameetriteks puu ja tipp; vaid ülesande nr 6 korral tuleks lisada ka kolmas parameeter: arv, mis peab saama antud tipu tasemenumbriks. Mõnevõrra keerulisem on ülesanne nr 3, mille puhul võiks tööväljal "meeles pidada" (lisaks vahetippude arvule) ka alampuu tippude arvu ja lehtede arvu. Ülesandes nr 5 arvestada kahendpuu tipus ka puuduva alluvaga (selle kõrgus on 0). **9.7.** Kasutada abijärjekorda. **9.10.** Tasemed kuni eelviimaseni koosnevad üksnes kahe alluvaga tippudest, eelviimase taseme sabatippudest väljub kokku 0 või enam tühiviita. **9.18.** Algselt ühetipuline. Seejärel (kuni pole veel n tippu) valida juhuslikult üks olemasolevatest ilma kahe alluvata tippudest. Kui valitud tipul on üks alluv, siis lisada uus tipp puudunud alluva kohale. Kui valitud on lehttipp, siis lisada uus tipp sellele kas vasakuks või paremaks alluvaks (juhuslikul moel otsustades). **9.28.** Sisuliselt rekursiivseid pöördumisi alampuudesse tuleb realiseerida magasinitöötluse abil. **9.29.** (b) Induktsiooniga: trassi mistahes tipp saab külastatud etteantud arv kordi lõpliku arvu katsetega. **9.30.** Juhuslikul moel genereerida $\lceil n/2 \rceil$-tipuline kahendpuu, vt ülesanne 9.18; selles igale ühe alluvaga tipule lisada teine (puudunud) alluv, seejärel seada tippude märgendid. **9.31.** Läbimine lõppjärjestuses. **9.32.** /Algoritm, läbimine keskjärjestuses/ `suluav(T,t): kui t=Λ, siis tühisõne, vastasel korral "("+ suluav(T,t.vasak) + t.m + suluav(T,t.parem) + ")"`. **9.34.** Catalani arv. **9.35.** Vt [1], lk 27. **9.42.** Bittide arv antud teksti Huffmani kodeeringus: (a) 62; (b) 39.

10. Otsimispuud

10.3. Asjaolu, et juurtipu kirje on suurem oma kõikidest vasakus alampuus olevatest tippude kirjetest ja ei ületa ühegi parema alampuu kirje võtit, ei garanteeri veel kahendotsimispuuks olemist. Vt ka asjakohane teoreem, [1], lk 31. **10.4.** (a)

5 puud; (b) 17 puud; (c) 48 puud. **10.6.** Seda saab teha vaid ühel moel. **10.7.** Juurkirjeks saab, näiteks: (a) 5; (b) 6; (c) 4. **10.14.** Kitsaskohaks on võimalik puu kõrguse ja tippude arvu lineaarne sõltuvus. **10.15.** Leida, millised operatsioonid kõikide puu kirjetega saab realiseerida lineaarse ajalise keerukusega algoritmi abil. Vaja on mõnda neist kasutada mitu korda. **10.20.** Oletada vastuväiteliselt, et selline algoritm leidub. Seejärel leida, millise massiivioperatsiooni saab siis lahendada kiiremini, kui tõestatult teada. **10.21.** (b) Tipp langeb oma alluvatest hiljem. **10.22.** Viimasel kohal järjendis asub puu juur, eelmised peavad jagunema kaheks alamjärjendiks, millest üks koosneb juure vasaku alampuu tippudest lõppjärjestuses, teine – juure parema alampuu tippudest lõppjärjestuses. **10.23.** AVL-puu struktuuri säilitamine on odavam kui halvasti ehitunud tavalise kahendotsimispuu kasutamine. Miks? **10.25.** Ei saa olla. **10.27.** 20-tipuline. **10.29.** On mitu võimalust. **10.35.** Juba olemasolevasse AVL-puu struktuuri on võimalik etteantud järjendi kirjeid võimalik paigutada vaid ühel viisil. **10.36.** Iga tipu kauguse konsooli (ekraani) vasakust äärest saame, kui läbime puu keskjärjestuses. **10.38.** Leida kõrgusega h AVL-puu minimaalne ja maksimaalne kirjete arv. **10.39.** Vt. vihje ülesandele 10.38. **10.40.** Juurtipu mõlemas harus peab olema minimaalne võimalik tippude arv. **10.41.** Vt. vihje ülesandele 10.40. **10.42.** Tuleb garanteerida, et iga alampuu kõrgust leitaks täpselt üks kord. **10.43.** Kahendpuu läbimine toimetada eesjärjestuses. **10.44.** Väide ülesandest 10.41. **10.45.** Puu läbimine keskjärestuses on lineaarse ajalise keerukusega tippude arvu suhtes. **10.53.** (a) Lõppseis: `(16(13(11( ,12), 14(13.5, )), 17(16.5, 18( ,19))))`. **10.73.** B-puu definitsioonis on puu tipu kirjete arvule seatud nii alam- kui ka ülempiir.

11. Kuhjad

11.14. Kahendotsimispuust saame $\Theta(n)$ ajaga selle puu kõiki kirjeid sisaldava sorteeritud järjendi. **11.15.** Kuhjastamise rekursiivne variant töötab lineaarse ajalise keerukusega tippude arvu suhtes. **11.18.** Kahel juhul saadakse kahendkuhi, kahel juhul alati mitte. **11.19.** Leida iga tipu jaoks keskmine vahetuste arv allaviimisel. **11.34.** Juhud, kus sama liiki puude ühendamisi on võimalikult palju.

12. Klasside kujutamine

13. Graafi läbimine

13.4. Välimise tsükli igal sammul moodustatakse järgmine front Q', lähtudes sammu algul antud frondis Q olevatest tippudest. Samm lõpeb omistamisega $Q := Q'$. **13.6.** (a) $\prod_{t \in V(G)} p(t)!$, kus $p(t)$ on tipu t korral fronti lisatavate (vaatlemata) naabrite arv. **13.7.** Leida servade arvu ja tippude arvu suhte piirid. **13.8.** Tipu töötlemine graafi läbimise algoritmis seisneb siin järjekordse vaadeldava tipu (lisaväljale) eellase märkimises, et „pidada meeles“, kust vaadeldavasse tippu tuldi. Kui tee leidub, siis saab selle kätte mööda eellasi „tagurdades“, lähtudes sihttipust. **13.9.** Sobivasse kohta rekursiivses algoritmis, mis leiab ühe tee graafi sügavuti läbides, lisada

tipult tunnuse „vaadeldud“ mahavõtmine. **13.13.** Kui järjekordse vaadeldava tipu mõni naabritest on juba varem vaadeldud, siis leidub antud graafis tsükkel. Tuleb arvestada, et graaf ei pruugi olla sidus. **13.14.** Lähedane kõigi teede otsimisele, vt ülesanne 13.9. Arvestada, et graaf ei pruugi olla sidus. **13.15.** Laiuti läbimine. Igasse tippu peab juurest jõudma täpselt ühel viisil. **13.18.** Teede otsimine antud maatriksi põhjal konstrueeritud graafis. **13.19.** Sügavuti läbimine, kus märgistatakse (läbitud) kaari, mitte (vaadeldud) tippe. **13.20.** Korratav tegevus, eeldusel, et graafis on juba leitud mingi (ilma korduvate servadeta) tsükkel C: leida C tipp v, millest lähtuvate servade hulgas on veel läbimata serv; leida tsükkel C' läbi tipu v (vt ka 13.19); muuta C, pookides selles tipu v asemele tsükli C'. Korratakse seni, kuni kõik servad on kantud tsüklisse C. **13.22.** Sügavuti. **13.23.** Ühessegi tsüklisse mittekuuluvad tipud võib kohe kõrvale jätta; lugeda need vaadelduks. Korrata: veel vaatlemata tipu puhul leida (sügavuti) seda läbiv tsükkel (võib sisaldada ka vaadeldud tippe); sellel olevad servad orienteerida (kui pole veel orienteeritud), tipud lugeda vaadelduks. **13.24.** Üks võimalus: graafis teede sügavuti leidmise, lisaparameetriga varustatud rekursiivne protseduur. Kõiki teid ei pruugi läbi vaadata. **13.25.** Arvestada puu tippude arvu ja servade arvu vahelist seost.

14. Kaugusalgoritmid graafidel

14.7. Selle tipupaari kaugus peab realiseeruma läbi kõigi nelja ülejäänud tipu. **14.14.** Kaarte töötlemise järjekord on suvaline. **14.15.** Kahe tipu vaheline lühim tee kasutab maksimaalselt $n-1$ kaart, kus n on tippude arv. **14.20.** Taolisel graafil leidub tippude topoloogiline järjestus ([1], lk 93).

15. Eeldusgraaf

15.9. Topoloogilise järjestuse olemasolu. **15.22.** Topoloogiliste järjestuste suurim võimalik arv on 30.

16. Graafi toes

16.9. Läbides puu iga serva täpselt 2 korda, jõuame alati teekonna lähtetippu. **16.22.** Lähendi konstrueerimine – vt ülesande 16.9 vihje. **16.23.** Vt ka: `https://en.wikipedia.org/wiki/Maze_generation_algorithm` (24.08.16).

17. Varia

17.12. Soovitatav tutvuda: Gale-Shapley algoritm. **17.19.** Ühekordne tsükkel üle tugevuste järjendi.

18. Eriteemad

18.3. Soovitus: teksti t alla järgnevatesse ridadesse kirjutada otsisõne s sobiva nihkega; otsisõnes kriipsutada alla vastaval sammul võrdlemisele tulevad sümbolid.

18.9. (b) Vaadeldavateks sõnadeks lugeda tühiku(te)ga eraldatud tekstiosad. Vastuseks on nende sõnade arv, millel leidub ainult tähtedest koosnev v-tüvi. **18.16.** Alamülesanne: kontrollida, kas kaks antud vertikaalset äärelõiku on silmsides, kui nende vahel paikneb n antud vertikaalset vahelõiku ($n \geq 0$). Tarvitseb vaadelda vaid lõikude otspunktide ordinaate, st lõik on antud oma otspunktide paarina. Olgu äärelõikudeks (a_0, b_0) ja (a_1, b_1), kus paari esimene liige tähistab lõigu alumist, teine aga ülemist otspunkti. Ülesanne taandub kontrollimisele, kas nende omavaheline kattumisala ehk kontroll-lõik $(a,b) = (\max(a_0,a_1), \min(b_0,b_1))$ eksisteerib ja on vahelõikude poolt täielikult kaetud või mitte. Kaetuse kontrolli saab esitada nii „jaga ja valitse“ kui ka iteratiivset laadi alamalgoritmina. Esimesel juhul võtta jaotuspunktiks üks vahelõikudest (u,v); seejärel lahendada kaks väiksemat ülesannet: kontrollida lõigu (a,u) on kaetust vahelõikudega v.a (u,v) ja lõigu (v,b) on kaetust vahelõikudega v.a (u,v). Iteratiivse lähenemise korral „projekteerida“ vahelõikude otspunktid vertikaalile nt nurksulgude paaridena, kontrollides saadava nurksuluavaldise terviklikkust. **18.17.** Püstitame üldisema ülesande: leida (konstrueerida) kõigi selliste alam-ristkülikute hulk RH, nii et iga ristkülik $r \in RH$ rahuldab esialgse ülesande tingimusi ja on pindalalt maksimaalne, st on piiratud mõnede märgitud punktidega ja/või R küljega (külgedega). Hulka RH kuuluvad ristkülikud võivad osaliselt kattuda. Esitatud üldisema ülesande saab lahendada tsüklis, mille igal sammul võetakse vaatlusele üks järjekordne märgitud punkt p; iga sellise ristküliku $r \in RH$ korral, mis sisaldab punkti p eemaldatakse r hulgast RH ja hulka RH lisatakse neli uut maksimaalset (ja osaliselt kattuvat) alam-ristkülikut, milledeks punkt p jaotab ristküliku r. Hulga RH algväärtus enne tsüklit: $RH = \{R\}$. Esialgse ülesande lahenduseks on suurima pindalaga alam-ristkülik(ud) hulgas RH. **18.18.** $x < y \xRightarrow{x:=x^2-y} x^2 - y < y \Leftrightarrow x^2 < 2y \Leftrightarrow P$. **18.19.** (Algoritm a) Näidata, et tingimus R on tsükli invariant, $R \Leftrightarrow (k < n+1) \wedge (s = 1 - \frac{1}{k})$. Selleks (1) näidata R invariantsus: kui enne tsükli sisu (kolme omistamise) täitmist kehtib $(k < n) \wedge R$, siis pärast tsükli sisu täitmist kehtib R; tõestus – alt üles, rakendades kolm korda omistamise tuletusreeglit ([1], lk 130); (2) näidata R sihipärasus: $(k \geq n) \wedge R \Rightarrow (s = 1 - \frac{1}{n})$. Seejärel näidata tsükli lõplikkus: enne tsüklisse sisenemist on avaldis $n-k$ positiivne, sest $n > 5$ ja $k = 2$; kui enne tsükli sisu (kolme omistamise) täitmist mingi konstandi c korral kehtib $0 < n-k = c$, siis pärast tsükli sisu täitmist kehtib $0 \leq n-k < c$; tõestus – samuti alt üles, rakendades kolm korda omistamise tuletusreeglit. Lõpuks näidata, et enne tsüklisse sisenemist kehtib R. (Algoritm b) Leida kogu algoritmi nõrgim eeltingimus (R): $0,5 \leq x \leq 1 \xRightarrow{x:=x/2} 0,5 \leq x/2 \leq 1 \Leftrightarrow 1 \leq x \leq 2 \xRightarrow{x:=x+2y} 1 \leq x+2y \leq 2 \ldots \xRightarrow{m:=1-m} R$ ja näidata, et konkreetsest antud eeltingimusest järeldub R. (Algoritm d) Kõigepealt näidata, et tingimus R, $R \Leftrightarrow t \leq (n+1) \wedge m = \frac{(t-1)!}{k!}$ on tsükli invariant.

Vastused

1. Funktsiooni asümptootiline hinnang

1.5. (b) Jah. (c) Jah. (d) Ei. **1.8.** Järjestus aeglasemalt kasvavast funktsioonist alates: (d), (e), (b), (a), (g), (f), (c). **1.10.** (a)-; (b)-; (c)-; (d)-; (e)+; (f)-; (g)+ . **1.11.** (a)-; (b)+; (c)-; (d)-; (e)-; (f)+ .

2. Algoritmi ajaline keerukus

2.1. (a) $2n$; (b) $2n^2+n$: (c) n^2+2n. **2.2.** (a) $\lfloor \log_2 n \rfloor + 1$. **2.3.** Vastavalt $\Theta(n^2)$ ja $\Theta(n^3)$. **2.5.** $\Theta(n)$. **2.6.** (a) $\Theta(mn)$; (b) $\Theta(n)$. **2.7.** Halvim, parim ja keskmine juht ühtivad. (a) $\Theta(n)$; (b) $\Theta(\sqrt{n})$. **2.8.** Halvim ja parim juht erinevad vaid konstandi võrra, seega on mõlema hinnanguks $\Theta(n)$. Halvim juht realiseerub, kui kõik järjendi elemendid on nullid, parim juht aga siis, kui järjendi kõik elemendid on nullist erinevad. **2.10.** $\Theta(n)$. **2.11.** (a) $\Theta(n^3)$; (b) $\Theta(2^n)$. **2.13.** (a) $\Theta(n^3)$; (b) $\Theta(n^2 n!)$. **2.14.** (a) $\Theta(mn)$; (b) $\Theta(n\log(m!))$. **2.15.** (a) $\Theta(m+\log n)$; (b) $\Theta(m^2+n^2)$. **2.20.** Jah. **2.26.** (a) $\Theta(n)$; (b) $\Theta(n)$; (c) $\Theta(n^2)$. **2.29.** (a) $\Theta(n^2)$; (b) $\Theta(n^2\log n)$; (c) $\Theta(n^3)$; (d) $\Theta(n^2 n!)$; (e) $\Theta(n^3)$. **2.30.** (a) $\Theta(n)$ ja selles sisalduvad O-klassid. (b) $O(2^n)$. (c) $\Theta(n)$ ja selles sisalduvad O-klassid. **2.32.** (b) $\Theta(n^5)$. **2.40.** (a) Programmi käitamisaja ajaline keerukus on $\Theta(n2^n)$; seega maksimaalne n on 36, aega kulub arvestuslikult 2949 sekundit.

3. Hargnemistega algoritm. Rekursioon

3.2. 54. **3.3.** (a) REKURSIOON; (b) NOOISRUKER; (c) REKURSIOONNOOISRUKER; (d) RKRIONOSUE; (e) EUSONOIRKR. **3.4.** Prinditakse 1023 rida ja sooritatakse 1534 funktsiooni `tõsta` väljakutset; üldjuhul vastavalt 2^n-1 ja $3\cdot 2^{n-1}-2$. **3.6.** (a) $a+1,b$; (b) $a,b-1$; (c) $a+1,b$; (d) $a+1,b$; (e) $a+1,b-1$. **3.7.** (a) Kontrollib, kas sisendiks antud arv on paarisarv. (b) Leiab naturaalarvuliste sisendite a ja b korral arvu a^b. (c) Leiab naturaalarvude a ja b korrutise. (d) Kontrollib, kas sisendiks antud arv on naturaalarv. **3.8.** (a) 1 kord. (b) $n!$ korda, kus n on vaadeldava alamhulga elementide arv. **3.10.** Negatiivsete täisarvuliste sisendite korral ei peatu. **3.11.** (a) Ei. (b) Jah. (c) Jah. (d) Jah. **3.13.** (a) Alati. (b) Alati.

4. Variantide läbivaatamine

4.13. $\Theta(n^2)$.

5. Otsimisalgoritmid järjenditel

5.1. Siis on suurim sammude arv otsingul teiste jaotamisversioonidega võrreldes vähim. **5.3.** Järjestikotsingut. **5.6.** (a) Ülevalt vasakult alustades, suuna muutus toimub elementidel 10, 46, 24, 45, 40, 42 ja 33. Alt paremalt alustades toimub

suuna muutus elementidel 39, 50, 40, 45, 40, 92 ja 38. **5.12.** (a) Kasutada sadulameetodit täiendusega, et otsingusuuna muutuse koht tuleb ise arvutada. (b) Analoogselt sadulameetodiga, „lõigata“ kolmemõõtmeline otsinguruum „viiludeks“ ja igal viilul rakendada sadulameetodit.

6. Järjendi ümberkorraldamine

6.1. Esimene massiiv peab töö lõppedes olema järjestuses $[20,21,12,15,19,13,14,11,23,29,31,30,35,25,24,26,28,27,32,23]$. **6.4.** Esimese massiivi lõppseisuks on $[23,30,45,36,62,61,29,55,35,40,50,78,80,77,75]$. **6.5.** Esimese massiivi lõppseisuks on $[16,11,17,15,18,19,22,20,23,35,31,30,37,28,37,27,32,24]$. **6.6.** Kui lahkmeks valitud element on tööalal vähim. **6.26.** 101, üldjuhul $2n-1$. **6.27.** 189, üldjuhul $2n-1$. **6.30.** Ligi 9 sekundit. **6.31.** (a) Ligi 18 sekundit. (b) Ligi 4 sekundit. **6.42.** $\Theta(n^3)$.

7. Paisksalvestus

7.3. Ühisteguri puudumine tagab, et tabelis rea otsimisel saab jõuda iga reani. **7.24.** Jääkpaiskamine. **7.31.** Kimbumeetod. **7.32.** Loendamismeetod.

8. Magasin ja järjekord

8.5. (a) $4m+3n$; (b) $\Theta(m+n)$. **8.9.** (a) Paigutuste arvud $n = 1,2,\ldots,12$ jaoks on $1,0,0,2,10,4,40,92,352,724,2680,14200$.

9. Puu ja kahendpuu

9.3. (a) $aste(T,t)$: kui $t.vasak = \Lambda$ ja $t.parem = \Lambda$, siis $t.x := 0$, vastasel korral, kui $t.vasak \neq \Lambda$ ja $t.parem \neq \Lambda$, siis $t.x := 2$, muidu $t.x := 1$; $aste(T,t.vasak)$; $aste(T,t.parem)$. (b) $aste(P,t)$: $t.x = 0$`;[* Tipu` t `iga alluva` v `korral:` $t.x++$`;` $aste(P,v)$`]`. (c) $asteP(T,t)$: $t.x = 0$`;` $v := t.vasak$`;[* kuni` $v \neq \Lambda$`:` $t.x++$`;` $asteP(T,v)$`;` $v := v.parem$`]`. **9.4.** Rekursiivne algoritm, t – vaadeldav tipp. Baasjuht: kui t on olematu, siis vastuseks 0 (tühja puu kõrgus). Põhitegevus: rekursiivselt rakendades leiame t vasaku alampuu kõrguse h_0 ja t parema alampuu kõrguse h_1, vastuseks on $1+$(suurim kõrgustest h_0 ja h_1).

10. Otsimispuud

10.12. Paremasse harusse. **10.14.** Kasutada AVL-puud. **10.24.** Kõik peale viimase. **10.29.** Kõrgusega 5 või 6 või 7. **10.32.** Lehttippe peab leiduma ka igal tasemel $51\ldots99$. **10.40.** $c(h) = c(h-1)+c(h-2)+1$, $c(1) = 1$, $c(2) = 2$. **10.59.** (a) Juurtipus 1 ja 2; vahetipus 1 ja 2. (b) Juurtipus 1 ja 8; vahetipus 4 ja 8.

11. Kuhjad

11.6. Lõpptulemus teise massiivi jaoks on [91, 90, 85, 71, 88, 80, 81, 68, 61, 60, 76, 48, 49, 75, 70, 59, 51, 55, 58, 50]. **11.23.** 256. **11.24.** 7, 5, 3.

12. Klasside kujutamine

13. Graafi läbimine

13.2. Joonis 19 (kui naabrid tähestiku järjekorras): (a) *abcdefgh*; (b) *abcehdfg*; (c) *hecbfgd*. **13.5.** 12. **13.7.** Ei ole. Keerukuse hinnanguks on $\Theta(n^2)$, kus n on tippude arv. **13.11.** Parim juht: $\Theta(n)$; halvim juht: $\Theta(n^2)$.

14. Kaugusalgoritmid graafidel

14.14. Ei. **14.16.** (b) Dijkstra algoritm.

15. Eeldusgraaf

15.3. 1 ja $n!$. **15.6.** 113400; jah. **15.8.** Mingil hetkel tööjärjekord on tühi, aga kõik tipud pole veel tööjärjekorda pandud. **15.10.** Jah. Midagi ei saa öelda. **15.17.** Esimesed kaks. **15.23.** Kriitiliseks teeks on kindlasti selline ahel, milles tippude töötlusaegade summa on suurim. **15.24.** Ei. **15.25.** Jah. **15.26.** 1.

16. Graafi toes

16.2. 40. **16.7.** (a) Jah. (b) Ei.

17. Varia

18. Eriteemad

Viited

[1] J. Kiho. *Algoritmid ja andmestruktuurid.* Kolmas, parandatud ja täiendatud trükk. TÜ, 2003, 147 lk.

[2] J. Kiho. *Algoritmid ja andmestruktuurid. Ülesannete kogu.* TÜ, 2005, 31 lk.

[3] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1976.

[4] D. E. Knuth. *The Art of Computer Programming, III*. Addison-Wesley, 1998

[5] *ACM-ICPC Live Archive.* `https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8` (24.08.2016)

[6] *UVa Online Judge.* `https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8` (24.08.2016)