Algoritmid ja andmestruktuurid
Arvutipraktikumi ülesanded I

Ajalise keerukuse empiiriline hindamine

Algoritmi efektiivsuse hindamisel on üks olulisi kriteeriume algoritmi täitmiseks tehtavate operatsioonide arv (mis määrab ka vastava programmi tööaja). Algoritmi ajaline keerukus on funktsioon k, mis algandmete mahule n seab vastavusse algoritmi sellise mahuga ülesannete täitmiseks tehtavate operatsioonide keskmise arvu $k(n)$. Ütleme, et algoritmi ajalise keerukuse k O-hinnang on $f(n)$ (lühemalt: k on $O(f)$), kui leiduvad konstandid $c > 0$ ja n_0 nii, et iga $n > n_0$ korral $k(n) < cf(n)$. Piltlikult: funktsiooni k graafiku kasvukõverus ei ole „ägedam“ kui funktsioonil f.

Algoritmi ajalise keerukuse hinnang määrab ka algoritmi praktilise rakendatavuse piiri: väga kiiresti kasvava keerukusfunktsiooniga algoritmi tasub realiseerida vaid väikesemahuliste ülesannete jaoks.

Ülesanne 1

Praktikumitöö.

a) Leida suurim indeks, millele vastava Fibonacci arvu on arvuti võimeline välja arvutama 1 sekundi jooksul rekursiivse meetodiga (vt [Kiho 2003, lk 89]).

b) Leitud väärtuse puhul teha kindlaks sellise meetodi tööaeg, kus Fibonacci arvude arvutamine on realiseeritud mitterekursiivse algoritmiga (vt [Kiho 2003, lk 89]).

c) Määrata ka uue algoritmi puhul suurim indeks, millele vastava Fibonacci arvu suudab arvuti välja arvutada 1 sekundi jooksul.

Ülesanne 2

Iseseisev töö.

a) Valida välja, realiseerida ja testida üks ruutkeerukusega (ajalise keerukuse hinnanguga $O(n^2)$) algoritm järjendi elementide ümberpaigutamiseks mittekahanevasse järjekorda. Sellisteks algoritmideks on näiteks mullimeetod (ingl k *bubble sort*), valikumeetod (*selection sort*), pistemeetod (*insertion sort*) jt. Testimisel mitte unustada nn äärejuhte (järjendid pikkusega 0, 1 või 2; ette antakse sorteeritud järjend, või siis vastupidiselt sorteeritud järjend).

b) Valida välja, realiseerida ja testida üks ajalise keerukuse hinnanguga $O(n \log n)$ algoritm järjendi elementide ümberpaigutamiseks mittekahanevasse järjekorda. Sellisteks algoritmideks on näiteks kiirmeetod (*quick sort*), ühildusmeetod (*merge sort*), Shelli meetod (*Shell sort*) jt. Testimisel mitte unustada nn äärejuhte!

c) Kujutada (soovitavalt vastava programmi abil) nende meetodite tööaegade graafikud, mis ilmekalt demonstreeriksid tööaegade kasvu vastavalt sorteeritavate järjendite pikkuste kasvule. Kolmandaks olgu joonisele lisatud ka süsteemse sorteerimismeetodi tööaja graafik (nt Pythoni korral meetod `sort()`).

Ajalise keerukuse empiiriliseks hindamiseks (ja graafikute joonistamiseks) tuleb mõõta programmi tööaega erinevate algandmemahtude korral.

Keeles Python saab seda teha näiteks moodulis `time` defineeritud funktsiooni `time()` abil:

```
from time import *

a = time()
# PROGRAMMIOSA, MILLE TÄITMISAEGA MÕÕDAME (algus)

# PROGRAMMIOSA, MILLE TÄITMISAEGA MÕÕDAME (lõpp)
b = time()

print(b - a) # PROGRAMMIOSA täitmisaeg sekundites
print((b - a)*1000) # PROGRAMMIOSA täitmisaeg millisekundites
```