Algoritmide keerukus

Algoritmi ja keerukuse mõiste. Hindamine. Keerukusklassid.

Algoritm

Algoritm on lõplik instruktsioonide (käskude, sammude) hulk, mis annab täpse operatsioonide järjestuse mingi probleemi (või probleemide klassi) lahendamiseks.

Arvutiteaduses on **algoritm** meetod probleemi lahendamiseks *(problem solving)*, mida saab realiseerida arvutiprogrammi abil.

Algoritmi omadused

Lõplikkus. Algoritmi töö lõppeb peale lõpliku arvu sammude läbimist (mis võib olla suur).

Määratletus. Algoritmi iga samm on täpselt määratud iga juhu jaoks.

Sisend ja väljund. Algoritmil on sisendandmed, mis on üheselt seotud väljundandmetega.

Efektiivsus. Kõik algoritmi sammud on nii lihtsad, et nad on paberi ja pliiatsi abil täidetavad lõpliku ajavahemiku jooksul, arvutil mõistliku ajaga.

Korrektsus. Algoritm lahendab etteantud ülesande õigesti.

Efektiivsus

- Esmase tingimusena peab algoritm peab olema **korrektne** (andma õige tulemuse).
- Lisaks peab algoritm olema **efektiivne** (leidma vastuse mõistliku ajaga).
- Programmi efektiivsusele hinnangu andmist nimetatakse **algoritmi keerukuse** uurimiseks.
- Hinnata saab algoritmi:
  - Ajalist keerukust *(time complexity)*
  - Mahulist keerukust *(space complexity)*
- Teisi näitajaid uuritakse harva.

Ajakulu hindamine

- Kui palju aega kulub algoritmi täitmiseks? Kuidas seda mõõta ja kuidas algoritme võrrelda?
  - sekundites (erineva kiirusega arvutid)
  - koodiridades (erinevad stiilid ridade kirjutamisel)
  - elementaartehetes (erinevad tehted, erinev aeg)
  - paneme käima ja mõõdame aega?
- Võrreldav on see, kuidas sõltub algoritmi tööaeg sisendi suurusest.

Sisendi suurus -> aeg

Mida rohkem andmeid, seda rohkem aega / samme kulub nende töötlemiseks.

Näide: otsimine

```
Korda kõigi andmetega:
  Kui andmeühik == otsitav:
    Teata "Leitud"
    Lõpeta otsimine
```

Aega saab tinglikult hinnata sammude arvuga:

1 võrdlemine = 1 samm

Kas antud näites sõltub aeg vaid sellest?

Sisendi iseloom -> aeg

- Kui otsitav on esimene?
- Kui otsitav on keskel?
- Kui otsitav on viimane?
- Kas saab olla veel kehvemat varianti?

Seega võib rääkida tööajast / sammude hulgast parimal, halvimal ja keskmisel juhul.

Algoritmi tööaeg

- Algoritmi tööaja hindamiseks võib teha eksperimendi:
  - realiseerida algoritm;
  - käivitada ta erinevate andmehulkadega;
  - mõõta aega.
- Puudus – ei saa ammendavaid tulemusi, sest igal eksperimendil on piirid.
- Parem oleks algoritmi analüüsida sõltumatult keelest, riistvarast, op-süsteemist ja arvestada erinevate andmetega.

Algoritmi keerukus kui funktsioon

Keerukus on funktsioon, mis seab andmete mahule vastavusse algoritmi sammude arvu.

- Sammu võib tõlgendada erinevalt – see võib olla üks tehe, üks tsüklitingimus või ka üks lause

Algoritmi **keerukusfunktsiooni kasvukiirus** näitab, kui kiiresti kasvab vastava programmi ressursivajadus töödeldavate andmete mahu kasvades.

Algoritmi keerukusfunktsiooni uurimine lubab anda hinnangu algoritmi efektiivsusele.

Algoritmi keerukus

- Analüüsides **algoritmi keerukust** saab anda hinnagu tema tööajale.
- Algoritmi keerukus:
  - Halvimal juhul *(worst-case complexity)*
  - Parimal juhul *(best-case complexity)*
  - Keskmisel juhul *(average-case complexity)*
- Kriitilistel juhtudel on kindlasti oluline keerukus halvimal juhul.

Algoritmi keerukusfunktsioon

- Algoritmi keerukusfunktsiooni leidmiseks on võimalik kokku arvutada kõik sammud, mida arvuti teeb ülesande lahendamiseks.
- Seda ei ole võimalik väljendada konkreetse arvuna, vaid andmete hulgast (n) sõltuva valemina.
- Kui ühe operatsiooni kestvus oleks C0, siis n andme töölemisel kuluks aega C0*n. Sellele võivad näiteks lisanduda aega C1 võtvad operatsioonid, mis ei ole aga iga andmega seotud: C0*n + C1

Asümptootiline keerukus

- Liiga täpsete hinnangutega (sammude arvude lugemisega) ei ole suuremat pihta hakata.
- Analüüsi võib lihtsustada, kuid jõuda samas hinnanguni, mis annab üldise iseloomustuse.
- Hinnangud antakse suurte (piiramatult kasvavate) sisendite jaoks.
- Funktsiooni f(n) asümptootiline käitumine tähistab f(n) kasvu siis, kui andmete maht n piiramatult kasvab.

Funktsiooni asümptoot

- **Asümptoot** on sirge, millele funktsiooni graafik lõpmatult läheneb, kuid millega ta ei lõiku.

Leitud reaaleluline selgitus ;)

"Asümptoot on kui (sirge) elektri karjus. Ühel pool on krokodill ja teisel pool inimene. Krokodill ei tule inimese poolele, sest elektri karjus on vahel. Inimene tuleb lähemale uudistama, aga loomulikult krokodilli poolele ka ei lähe."

Asümptootilised hinnangud

Matemaatiliselt võib kohata tähistusi (c, c_1,c_2 on konstandid)

f(n) = Θ(g(n)) – asümptootiliselt täpne hinnang e f(n) kasv jääb c_1*g(n) ja c_2*g(n) vahele

- f(n) = O(g(n)) – ülemine hinnang ehk f(n) ei kasva kiiremini kui c*g(n)

Viimast algoritmide analüüsimisel kasutataksegi.

Asümptootilised hinnangud iseloomustavad keerukusfunktsioonide käitumist andmete mahu piiramatul kasvamisel.

Suure O tähistus

- Suure O tähistust (notatsiooni) *(Big O notation)* kasutatakse algoritmide keerukuse tähistamiseks.
- O(g(n)) on funktsioonide hulk, mis ei kasva kiiremini kui g(n).
- g(n) omakorda on funktsioon, mis kirjeldab algoritmi sammude arvu ja sellest tulenevalt tööaja seost sisendi mahuga (n).
  - Nt võib funktsiooniks g(n) olla n, n^2 jms

Keerukusklassid

Algoritmide süstematiseerimiseks on kasutusel O-tähistusele tuginevad **keerukusklassid:**

- O(1) – konstantne keerukus
- O(log n) – logaritmiline keerukus
- O(n) – lineaarne keerukus
- O(n log n) – linearitmiline (?) keerukus
- $O(n^2)$ – ruutkeerukus
- $O(n^3)$ – kuupkeerukus, üldisemalt polünomiaalne k
- $O(2^n)$ – eksponentsiaalne keerukus
- O(n!) – faktoriaalne keerukus

Keerukusklassi määramine

Algoritmi keerukusklassi määramiseks tuleb

- otsustada, millised operatsioonid kokku lugeda: nt lugeda sammuks lause;
- väljendada sammude arv sisendi suurust (n) arvestades: nt korrutades tsüklis tehtavad tegevused n-ga;
- lihtsustada tulemus - polünoomist jääb kõrgeim aste, likvideeruvad konstantsed kordajad jms:
  - Näiteks: $3n^2 + 5n + 2$ lihtsustub klassiks $O(n^2)$

Keerukusklassid

- Tuntud algoritmide jaoks on keerukusklassid määratud ja neid tuleks usaldada.
- Lihtsamatel juhudel saab ka ise koodi vaadates anda hinnagut keerukusklassile.
- Näiteks:
  - 1 tsükkel üle kõigi andmete – O(N),
  - 2 tsüklit üksteise sees üle kõigi andmete – $O(N^2)$

Aja kasv

Programmi tööaja kasv andmete mahu kasvades 5 x.

Funkts. c f(n)	Suhe f(25) / f(5)
c_1	1
$c_2 \log n$	2
$c_3\ n$	5
$c_4\ n \log n$	10
$c_5\ n^2$	25
$c_6\ n^3$	125
$c_7\ 2^n$	1 048 576

Andmed ja arvuti kiirus

Ajaline keerukus	1 MOPS	10 MOPS	100 MOPS
n	1 miljon	10 miljonit	100 miljonit
n^2	1 000	~ 3 162	~10 000
n^3	100	~ 215	~ 464
2^n	~20	~ 23	~ 26
MOPS – miljonit operatsiooni sek-s		Töödeldavate andmete mahu kasv arvuti kiiruse kasvades 10 x	Töödeldavate andmete mahu kasv arvuti kiiruse kasvades 100 x