Algoritmide keerukus

Algoritmi ja keerukuse mõiste.
Keerukuse hindamine. Keerukusklassid.

Algoritm

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

Algoritm on ülesande lahendamiseks täpselt määratletud reeglite lõplik korrastatud kogum (Infotehnoloogia sõnastik EVS-ISO/IEC 2382)

Igal käsul on täpne tähendus ja lõplik täitmise aeg.

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

Algoritm on probleemi lahendamise plaan (ingl *blueprint*). Kui algoritm olemas, võib selle tõlkida programmeerimiskeelde.

Algoritmist

Algoritm peab olema **lihtne** ja **lühike**.

Ta peab olema **toimiv** - soovitud tulemusi andev. Tulemus peab tekkima lõpliku ajaga.

Ta peab olema **üheselt kirjeldatud**, ei tohi olla mitmeti mõistetav ning loogika peab olema **selge**.

Ta peab olema **lõplik**, st algoritmi töö lõppeb peale lõpliku arvu sammude läbimist. Sammude arv võib olla väga suur.

Algoritm peab sobima kõigi andmete jaoks lubatud sisendite hulgast.

Algoritmi omadused

D. Knuth *(The Art of Computer Programming, vol 1):*

Sisend *(input):* algoritm aktsepteerib sisendandmeid tema jaoks kirjeldatud hulgast.

Väljund *(output):* algoritm tekitab tulemused, millel on täpselt määratud seos sisendiga.

Lõplikkus *(finiteness)*: iga siusendiga algoritm lõpetab töö lõpoliku arvu sammude pärast.

Selgus *(definiteness)*: kõik algoritmi sammud on määratud täpselt, selgelt ja ühemõtteliselt.

Toimivus *(effectiveness)*: lubatud sisendi korral loob järjekindlalt mõtestatud ja õige väljundi; algoritmi sammud peavad olema lihtsad ja lõpilku ajaga täidetavad.

Õigsus ehk veatus

Algoritm peab olema **õige** ehk **veatu** (ingl *correct*) – **iga** lubatud sisendi korral lõpetab ta töö õige väljundiga.

Veatu algoritm lahendab **arvutusprobleemi** (ingl *computational problem*).

Vigane algoritm võib osa sisendite korral saada õige tulemuse, mõne sisendi korral tööd mitte lõpetada ja osa sisendite korral anda vale vastuse.

Tõhusus, analüüsimine

Algoritm peab olema **tõhus** (ingl *efficient*)**.** Tõhususe peamiseks mõõdupuuks on ressursside kasutus.

Tõhusus on programmi täitmise kiiruse, mäluruumi tarbe või mõlema mõõt.

Ühte probleemi saab lahendada erinevate algoritmidega. Erinev tõhusus mõjutab tihti tööaega rohkem kui arvuti tehnilised näitajad.

Tõhusama algoritmi leidmiseks on vaja algoritmi kandidaate analüüsida (ingl *analysis of algorithms*).

Algoritmi analüüsimiseks nimetatakse tema ressursivajaduse ennustamist. Ressurssideks on tavaliselt mälu ja aeg.

Tööaja hindamine

Kuidas hinnata algoritmi täitmiseks kuluvat aega? Kuidas seda algoritmide võrdlemiseks mõõta?

- sekundites (paneme käima ja mõõdame; erineva kiirusega arvutid);
- koodiridades (erinevad stiilid ridade kirjutamisel);
- elementaartehetes (erinevad tehted - erinev aeg).

Parem on algoritmi tööaega hinnata sõltumatult keelest, riistvarast, op-süsteemist.

Algoritmi tööaeg on seotud tehtavate sammude arvuga.

Sammude arv on tavaliselt seotud sisendi ja/või väljundi suurusega.

Sisendi suurus -> aeg

Võrreldav on algoritmi tööaaja seos sisendi suurusega. Selle jaoks on vaja määrata lihtoperatsioonide arvu seos sisendi suurusega.

Operatsiooni piiritlemine on keeruline, nt üks võrdlemine (omistamine, tehe, ...) on üks operatsioon. Lihtsustatult üks lause on üks operatsioon.

Kasutatakse hüpoteetilise arvuti mõistet, kus kõik lihtoperatsioonid loetakse võrdseteks sammudeks.

Mida rohkem andmeid, seda rohkem operatsioone kulub nende töötlemiseks.

Sisendi iseloom -> aeg

Näide: otsimine

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

Millest sõltub operatsioonide arv?

Kas on erinevust ajakulus, kui:

otsitav on esimene?

otsitav on keskel?

otsitav on viimane?

Algoritmi tööaeg ja keerukus

Algoritmi tööajaks (ingl *running time*) loetakse konkreetse sisendi mahu korral lihtoperatsioonide arvu.

Hindamiseks leitakse funktsioon, mis iseloomustab andmehulga seost töötlemissammude hulgaga

Sarnaselt kirjeldatakse ka **algoritmi ajalist keerukust** (ingl *time complexity*)

Võib rääkida algoritmi tööajast parimal, halvimal ja keskmisel juhul.

Määrata tuleb enamasti **algoritmi tööaeg halvimal juhul** (ingl *worst-case running time*), sest nii saame tööaja ülemise piiri.

Algoritmi tööaja / sammude arvu leidmine on suur üldistus.

Algoritmi ajaline keerukus

Analüüsides **algoritmi ajalist keerukust** saab anda hinnagu tema tööajale.

Algoritmi ajaline keerukus (analoogia tööajale):

- halvimal juhul *(ingl worst-case complexity);*
- parimal juhul *(ingl best-case complexity);*
- keskmisel juhul *(ingl average-case complexity).*

Kriitilistel juhtudel on kindlasti oluline keerukus halvimal juhul.

Algoritmi keerukus kui funktsioon

Algoritmi ajaline 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 (nt tööaeg) töödeldavate andmete mahu kasvades.

Algoritmi keerukusfunktsiooni uurimine lubab anda hinnangu algoritmi tõhususele.

Algoritmi keerukusfunktsioon

Algoritmi keerukusfunktsiooni leidmiseks on võimalik kokku arvutada kõik sammud, mida arvuti teeks ülesande lahendamiseks.

Seda ei ole võimalik väljendada konkreetse arvuna, vaid andmete hulgast (n) sõltuva funktsiooniga.

Hüpoteetiliselt on ühe operatsiooni kestvus üks samm ajaga C_1, n elemendi töölemisele kulub aega $C_1 n$. Sellele võib lisanduda aega C_0 võttev operatsioon, mis ei ole iga andmega seotud:

$$C_1 n + C_0$$

Asümptootiline keerukus

Liiga täpsete hinnangutega (sammude arvude lugemisega) ei ole suuremat pihta hakata. Seda enam, et iga sammu täpne täitmise aeg ei ole teada ning sõltub taas riistvarast.

Analüüsi võib lihtsustada ja saada hinnang, mis annab piisavalt üldise iseloomustuse.

Hinnangud antakse suurte (piiramatult kasvavate) sisendite jaoks.

Funktsiooni `f(n)` **asümptootiline käitumine** kirjeldab `f(n)` käitumist, kui andmemaht `n` piiramatult kasvab.

Asümptootiline hinnang

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

$f(n) = \Theta(g(n))$ – (kreeka täht theeta) asümptootiliselt täpne hinnang ehk $f(n)$ kasv jääb $c_1 * g(n)$ ja $c_2 * g(n)$ vahele

$f(n) = O(g(n))$ – ülevalt piiratud hinnang ehk $f(n)$ ei kasva kiiremini kui $c * g(n)$

Viimast kasutatakse algoritmide hindamisel.

Algoritmi asümptootilise keerukuse hindamisel uuritakse, kuidas ta hakkab käituma **väga suurte andmemahtude korral,** alates mingist n_0-st.

Asümptootilised hinnangud

S. Skiena, Algorithm Design Manual

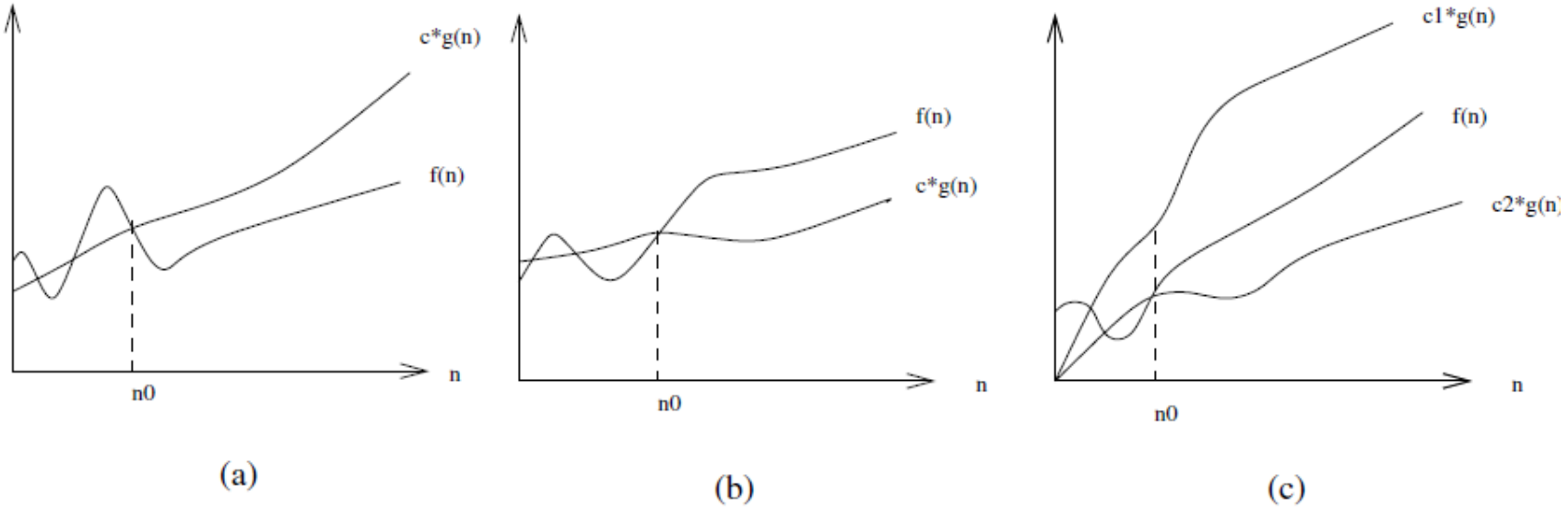

Figure 2.3: Illustrating the big (a) O, (b) Ω, and (c) Θ notations

Suure O tähistus

Kõige olulisem meie jaoks on **suure O tähistus** *(ingl Big Oh notation),* kasutatakse algoritmide keerukusklasside tähistamiseks.

`O(g(n))` on funktsioonide hulk, mis ei kasva kiiremini kui `g(n)`.

`g(n)` on omakorda funktsioon, mis kirjeldab algoritmi sammude arvu seost sisendi mahuga `(n)`.

- Nt võib funktsiooniks `g(n)` olla `n`, n^2 jms

Keerukusklassi määramine

Algoritmi keerukusklassi määramiseks tuleb

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

Keerukusklassid

Algoritmide tõhususe kirjeldamiseks ja süstematiseerimiseks kasutatavad peamised **keerukusklassid** on**:**

- 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

Keerukusklassid

Tuntud algoritmide jaoks on keerukusklassid määratud ja neid tasub 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 – $O(n^2)$

Programmi tööajast

Näide CLRS-raamatust *("Introduction to Algorithms")*.

On kaks arvutit: A teeb 10 miljardit (10^{10}) op ja B 10 miljonit (10^7) op sekundis.

Arvutil A käivitatakse üliefektiivselt kirjutatud $O(n^2)$ sorteerimisalgoritm, kus `C=2`. Arvutil B kehvalt kirjutatud $O(n \log n)$ algoritm, kus `C=50`.

10 miljoni arvu sorteerimiseks kulub arvutil A:

$(2*(10^7)^2)/10^{10} \approx 20\ 000$ sek

Ja arvutil B:

$(50*10^7 * \lg 10^7)/10^7 \approx 1163$ sek