Algoritmitutkimus: helppoja, vaikeita ja mahdottomia ongelmia

Jyrki Kivinen

Tietojenkäsittelytieteen esittely 15.9.2004

Algoritmi on täsmällisesti esitetty jono toimenpiteitä, joilla syötteestä saadaan haluttu tuloste.

- ”täsmällinen” tarkoittaa suunnilleen samaa kuin matematiikassa
- syöte ja tuloste ovat yleensä symbolisessa muodossa esitettyä dataa (kirjainjonoja, lukuja, . . .)

Esim. kertolaskuongelma

syöte: kaksi luonnollista lukua
34986921 ja 7729527

tuloste: syötelukujen tulo
270432350516367 ($= 34986921 \cdot 7729527$)

algoritmi: ”allekkain kertolasku”

Käytännössä usein algoritmi on periaateratkaisu jonka perusteella kirjoitetaan tietokoneohjelma.

Esimerkkiongelma: hahmonsovitus

Syötteenä teksti (pitkä kirjainjono, esim. `muuttolaatikkolasti`) ja hahmo (lyhyempi kirjainjono, esim. `tola`)

Löydettävä hahmon esiintymäkohdat tekstissä jos on (esim. `muuttolaatikkolasti`.

Perusalgoritmi on melko ilmeinen (kokeillaan systemaattisesti joka paikassa, sopiiko).

- Perusalgoritmi on liian hidas hyvin pitkillä teksteillä ja hahmoilla.
- Entä jos sallitaan pienet virheet? Esim. `muuttolaatikkolasti`
- Entä jos etsitään useita hahmoja yhtaikaa?

Usein käytettyjä algoritmeja

- reititysalgoritmit tietoliikennepakettien ohjaamiseen tietoverkoissa
- tiedonhakualgoritmit tietokantajärjestelmissä
- salakirjoitusalgoritmit
- geometriset algoritmit käyttöliittymissä ja peleissä
- numeeriset algoritmit esim. differentiaaliyhtälöiden ratkaisemiseen
- ohjelmointikielten kääntäjät

Mitä tarvitaan algoritmin lisäksi

- syötteen kerääminen, tulosteen hyväksikäyttö (tietokannat, informaatiojärjestelmät, tietokoneverkot)
- algoritmin toteutus tietokoneohjelmana (laitteisto, käyttöjärjestelmä, ohjelmointikieli)
- ohjelman sijoittaminen osaksi suurempaa kokonaisuutta (ohjelmistotuotanto)
- vuorovaikutus ihmisten kanssa (käyttöliittymä)
- ym.

Algoritmitutkimuksen peruskysymyksiä

- Miten annetulle ongelmalle löydetään algoritmi?
- Toimiiko algoritmi aina oikein?
- Onko algoritmi tehokas (nopea, vie vähän muistia, . . .)?
- Onko ongelmia, joille ei ole olemassa tehokasta/minkäänlaista ratkaisualgoritmia?
- Jos on, niin mitä niille tehdään?

Käytännössä tärkeitä lisäkysymyksiä

- Miten algoritmi toteutetaan tietyllä ohjelmointikielellä?
- Miten yksittäisen algoritmin tehokkuus vaikuttaa koko järjestelmän tehokkuuteen?
- Miten algoritmi (oik. sen toteuttava tietokoneohjelma) toimii tietyssä ympäristössä (laitteet, käyttöjärjestelmä, . . .)?
- Miten algoritmi toimii niillä syötteillä joista varsinaisesti ollaan kiinnostuneita?

Ja ennen kaikkea, mikä oikeastaan on se ongelma joka halutaan ratkaista?

Toisaalta . . .

Computational problems are not only things that have to be solved, they are also objects that can be worth studying.

Christos Papadimitriou

Helppo ongelma: puhelinluettelo

Puhelinluettelossa on nimiä ja niihin liittyviä puhelinnumeroita.

`Autio`	`44602`	`Lokki`	`44245`
`Helin`	`44604`	`Nurmi`	`44250`
`Karhu`	`44124`	`Palin`	`44449`
`Lahti`	`44247`	`Rinne`	`44478`

Halutaan

- löytää numeroja nimen perusteella
- lisätä ja poistaa (nimi, numero) -pareja

(Yleisemmin tietysti (haku)avain voi olla nimen sijaan tilinumero, matkapuhelinnumero, atk-laitteen looginen nimi jne. ja siihen liittyvä tieto tilin saldo, puhelimen sijainti verkossa, laitteen fyysinen osoite jne.)

Perusratkaisu: taulukko aakkosjärjestyksessä

Etsintä tehokasta binaarihaulla. Esim. etsitään *Karhu*.
(Merk. *nimi1* $<$ *nimi2* jos *nimi1* on aakkosjärjestyksessä ensin.)

Autio	44602		Autio	44602		xxxxx	xxxxx
Helin	44604		Helin	44604		xxxxx	xxxxx
Karhu	44124		Karhu	44124		**Karhu**	44124
Lahti	44247		xxxxx	xxxxx		xxxxx	xxxxx
Lokki	44245	$\Rightarrow$	xxxxx	xxxxx	$\Rightarrow$	xxxxx	xxxxx
Nurmi	44250		xxxxx	xxxxx		xxxxx	xxxxx
Palin	44449		xxxxx	xxxxx		xxxxx	xxxxx
Rinne	44478		xxxxx	xxxxx		xxxxx	xxxxx
Karhu $<$ *Lahti*			*Karhu* $>$ *Helin*			löytyi	

Etsintä on melko nopeaa:

- tarkastetaan aina jäljellä olevien alkoiden puoliväli
- aakkosjärjestyksen perusteella jokainen tarkastus eliminoi ainakin puolet jäljelläolevista alkioista
- ol. taulukossa n alkiota
- k tarkastuksen jälkeen jäljellä kork. $n/2^k$ alkiota
- pitää olla $n/2^k \geq 1$, joten tarkastuksia kork. $k \leq \log_2 n + 1$
- jos esim. $n = 10$ milj. niin $k \leq 25$.

Ongelma: lisäykset ja poistot vaativat taulukon uudelleenorganisointia.

Kehittyneempi ratkaisu: hajautustaulu

- varataan taulukon kooksi m suunnilleen oletettu alkioiden maksimimäärä
- laaditaan sopiva hajautusfunktio h joka liittää jokaiseen nimeen x hajautusarvon $h(x) \in \{1, \ldots, m\}$
- nimeä x vastaava puhelinnumero talletetaan oletusarvoisesti taulukon riville $h(x)$
- valitaan sopiva tapa käsitellä yhteentörmäykset eli tilanteet $h(x) = h(y)$ eri nimille $x \neq y$

Hajautuksen edut:

- lisäykset ja poistot helppoja
- etsimiseen kuluva aika (suunnilleen) sama riippumatta alkioiden lukumäärästä n

Johtopäätös: helpoissakin ongelmissa oikeat algoritmit ja tietorakenteet voivat johtaa merkittäviin säästöihin

Tämä on tärkeää jos

- dataa on paljon (esim. verohallinto),
- vastaukset tarvitaan nopeasti (esim. puhelinkeskus) tai
- kyselyjä tehdään paljon (esim. osana jotain monimutkaisempaa algoritmia)

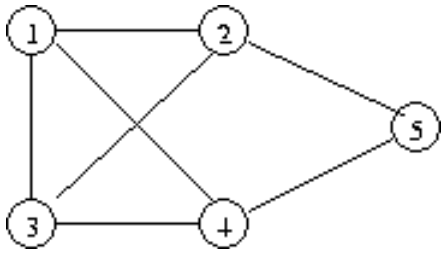
Huomautus 1: Tämä on klassinen ongelma johon tunnetaan lukuisia ratkaisumenetelmiä.

Huomautus 2: Itse algoritmin lisäksi toteutusyksityiskohdat ovat tärkeitä.

Verkko

koostuu solmuista ja solmuja yhdistävistä kaarista.

Verkko voidaan esittää esim. numeroimalla solmut ja merkkaamalla taulukkoon, minkä solmujen välillä on kaari.

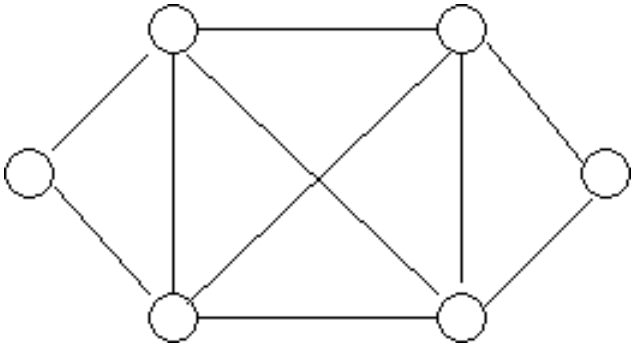


	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	0	1
3	1	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

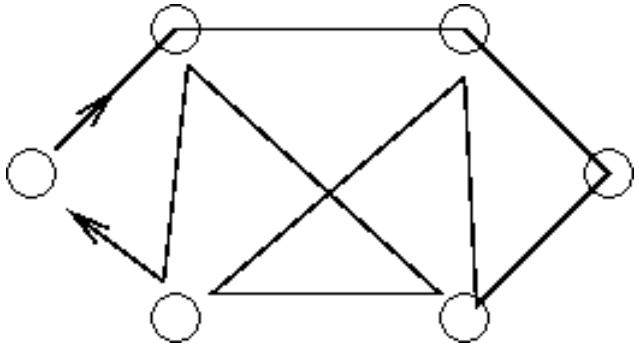
Verkkoja esiintyy tietenkin tietoliikenteessä, mutta lisäksi ne ovat erittäin yleinen tapa mallintaa tilanteita joissa on olioita ja niiden välisiä suhteita.

Helpohko verkko-ongelma

Eulerin kehä on polku joka käyttää *tasan kerran* jokaista verkon kaarta ja palaa lähtösolmuunsa.



Verkko



Eräs Eulerin kehä

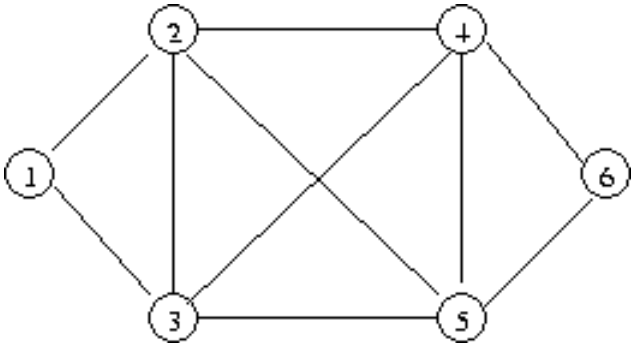
Eulerin kehä on olemassa jos ja vain jos verkko on yhtenäinen ja jokaiseen solmuun liittyy parillinen määrä kaaria (Euler 1736).

("Yhtenäinen" tarkoittaa että minkä tahansa kahden solmun välillä on polku, ts. verkko ei koostu useasta erillisestä palasta.)

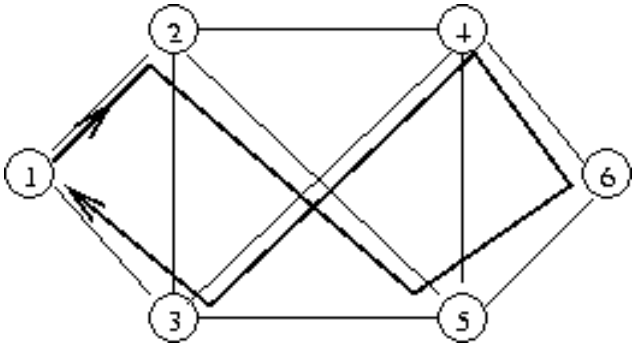
Tämän perusteella Eulerin kehä on myös melko helppo löytää jos sellainen on olemassa.

Vaikea (?) verkko-ongelma

Hamiltonin kehä on polku joka käy *tasan kerran* jokaisessa verkon solmussa ja palaa lähtösolmuunsa.



Verkko



Eräs Hamiltonin kehä

Hamiltonin kehä voidaan esittää luettelemalla solmu siinä järjestyksessä kuin ne esiintyvät polulla, esim. $1, 2, 5, 6, 4, 3$.

Hamiltonin kehän löytämiseen, tai sellaisen olemassaolon toteamiseen, ei tunneta yhtään *tehokasta* algoritmia.

Naiivi algoritmi Hamiltonin kehä -ongelmaan

- Käy läpi kaikki mahdolliset solmujen permutaation (eli järjestykset)
- Testaa kullakin permutaatiolla, esittääkö se Hamiltonin kehää.

Yhden permutaation testaaminen on helppoa: katsotaan onko aina kahden perättäisen solmun välillä kaari.

Ongelma on, että pahimmassa tapauksessa joudutaan testaamaan kaikki permutaatiot yksitellen läpi.

n solmua $\Rightarrow$ $n!$ permutaatiota — liian paljon

Esim. $50! \approx 3 \cdot 10^{64}$.

Hamiltonin kehä -ongelma on NP-täydellinen

Käytännössä NP-täydellisyys suunnilleen tarkoittaa, että

- ei tunneta algoritmia joka toimisi nopeasti hyvin suurilla ongelmilla,
- jos tällainen algoritmi löytyisi, suuri joukko muitakin samanlaisia ongelmia ratkeaisi sitä käyttämällä ja
- enemmistö alan asiantuntijoista veikkaa ettei tällaista algoritmia ole olemassakaan, mutta
- algoritmien kehittäminen "keskisuurille" syötteille on haastava ja käytännössä tärkeä ongelma.

Huomaa että NP-täydellisyys on itse ongelman ominaisuus, ei minkään yksittäisen algoritmin. (Tarkemmin ottaen se on täsmällinen matemaattinen termi, jonka määritteleminen tässä olisi turhan hankalaa.)

Mahdoton ongelma

Syöte: ohjelman lähdekoodi (esim. C-kielellä)

Kysymys: Tulostaako ohjelma mitään, kun (se käännetään ja) sille annetaan syötteenä sen oma lähdekoodi?

Tämä ongelma on ratkeamaton, toisin sanoen sille ei ole olemassa ratkaisualgoritmia.

- Tässä oletetaan että ohjelma ei kaadu muistin täyttymiseen tms.
- Ohjelmointikielen valinnalla ei ole väliä; teoreettisissa tarkasteluissa käytetään esim. *Turingin koneita*
- Vaikea tapaus on tietysti se, että ohjelma ei tulosta mitään ja ajautuu ikuiseen silmukkaan, mitä ei osata varmasti todeta. Tästä ongelman nimi pysähtymisongelma.

Missä nyt mennään?

- Perinteinen algoritmitutkimus liikkuu suurelta osin "helpon" ja "vaikean" välimaastossa.

- Modernissa tietojenkäsittelyssä (esim. tietokoneverkoissa, tekoälyssä) esiintyy myös paljon algoritmisia ongelmia, jotka eivät sovi tälle skaalalle, tai perinteiseen kaavaan "syöte $\rightarrow$ algoritmi $\rightarrow$ tuloste"

- Matematiikassa tutkitaan paljon "mahdottomia" (ja sitä vaikeampia) ongelmia, jotka ovat tärkeitä logiikan tutkimukselle. (Tämän takia algoritmin käsite ja ratkeamattomuus tunnettiin ennen elektronisten laskulaitteiden keksimistä.)

Viimeaikaisia algoritmiteorian läpimurtoja

- todistusten satunnainen tarkistaminen
- kvanttilaskenta
- julkisen avaimen salakirjoitus
- alkulukutestaus

Mielenkiintoisia sovellusalueita

- bioinformatiikka
- vaikea tieteellinen laskenta

Tutkimuksen luonteesta

- perustutkimuksessa ryhmät usein pieniä (2–5 henkeä)
- paljon kansainvälistä yhteistyötä joka perustuu henkilökohtaisiin suhteisiin
- algoritmianalyysi vaatii monipuolisia matemaattisia taitoja
- algoritmien testaaminen, matematiikkaohjelmistot yms.
- varsinaisessa soveltavassa tutkimuksessa isompi ryhmä, jossa sovellusalueen spesialisteja ja ohjelmoijia

Algoritmitutkimusta laitoksellamme

C-BRAHMS (Content-Based Retrieval and Analysis of Harmony and other Music Structures): Kjell Lemström, (Veli Mäkinen); City University, Lontoo

NAPS (Networking and Architecture for Proactive Systems): Patrik Floréen, (Pekka Orponen); HIIT, TKK

SYSFYS (Experimental and computational analysis of physiological regulation at transcriptome, proteome and metabolome level): Esko Ukkonen, (Juho Rousu); VTT

merkkijonomenetelmät: Juha Kärkkäinen, Esko Ukkonen, (Veli Mäkinen)

koneoppiminen: Jyrki Kivinen, (Tapio Elomaa, Juho Rousu); UC Santa Cruz, Australian National University; PASCAL

. . . plus jatko-opiskelijat

Muutkin hankkeet (esim. CoSCo, FDK) sisältävät kiinnostavia algoritmisia komponentteja.

Algoritmiopetusta laitoksellamme

Algoritmien peruskursseja: Tietorakenteet, Algoritmien suunnittelu ja analyysi

Matemaattisia perusteita: Ohjelmoinnin ja laskennan perusmallit, Laskennan teoria

Erikoiskursseja: Merkkijonomenetelmät, Tietokonegrafiikka, Geometriset menetelmät, Kombinatorinen optimointi, Koneoppiminen

Hyödyllisiä sivuaineita

- matematiikan perusopinnot
- lisää matematiikkaa: todennäköisyyslaskenta, logiikka, lineaarialgebra, differentiaali- ja integraalilaskenta, Diskreetti matematiikka II
- Fysiikan matemaattiset menetelmät?
- sovelluspuolen sivuaineet (esim. biologia)?