

7. Aikavaativuus

- Edellä tarkasteltiin ongelmien ratkeavuutta kiinnittämättä huomiota ongelman ratkaisun vaatimaan aikaan
- Nyt siirrytään tarkastelemaan ratkeavien ongelmien aikavaativuutta
- Periaatteessa ratkeava ongelma voi olla käytännössä ratkeamaton
- TSP (kauppatkustajan ongelma) on löytää lyhin mahdollinen *Hamiltonin kehä* painotetusta verkosta
- Triviaalialgoritmi: n :n solmun verkossa kokeile kaikki $n!$ reittiä ja valitse niistä lyhin

- Koneessa, jossa yhden reitin tutkiminen kestää 0.001 s, vaatisi tämä algoritmi 22-solmuisella verkolla enemmän aikaa kuin maailmankaikkeuden tähänastinen ikä
- Biljoonakertainen koneen nopeus: vieläkkään ei kyettäisi maailmankaikkeuden eliniän puitteissa ratkaisemaan TSP:tä 31-solmuiselle verkolle
- Kaikki reittivaihtoehdot tutkivaa eksponentiaalista algoritmia ($n! \approx O(n^n)$) ei näin ollen voi nimittää ratkaisualgoritmiksi TSP:lle
- Algoritmia, jonka ajankäyttöä rajoittaisi n :n polynomi ei tunneta; ei osata osoittaa, että sellainen menetelmä olisi mahdoton

- Standardimallisen Turingin koneen $M=(Q, \Sigma, \Gamma, \delta, q_0, \text{accept}, \text{reject})$ laskennan

$$q_0 w \ggg_M u q av, \quad q \in \{ \text{accept}, \text{reject} \}$$

pituus on siihen sisältyvien siirtymäaskelten lukumäärä

- Aikavaativuus syötteellä w :

$$\text{time}_M = \begin{cases} \text{laskennan } q_0 w \ggg_M \dots \text{pituus,} & \text{jos se pysähtyy} \\ \infty, & \text{jos laskenta ei pysähdy } w \text{ : llä} \end{cases}$$

- Koneen M aikavaativuus on siis funktio

$$\text{time}_M: \Sigma^* \rightarrow \mathbb{N} \cup \{ \infty \}$$

- Yleensä vaatavuutta tarkastellaan syötteen pituuden n funktiona:

- *keskimääräisessä tapauksessa* kun n :n pituisia syötteitä saadaan todennäköisyysjakaumasta $P_n(w)$

$$\text{time}_M^{\text{avg}}(n) = \sum_{|w|=n} P_n(w) \cdot \text{time}_M(w)$$

- yleisemmin *pahimmassa tapauksessa*

$$\text{time}_M^{\text{max}}(n) = \max_{|w|=n} \text{time}_M(w)$$

- Yleensä myös merkitään yksinkert.

$$\text{time}_M: \mathbb{N} \rightarrow \mathbb{N} \cup \{ \infty \},$$

$$\text{time}_M(n) = \text{time}_M^{\text{max}}(n)$$

- Keskimääräisen tapauksen analysointi olisi mielenkiintoista, mutta yleensä se on niin hankalaa, että useimmiten tyydytään pahimman tapauksen tarkasteluun

Vaativuusfunktioiden kertaluokat

- Pahimpaan tapaukseen tyypistetyt vaativuusfunktiot ovat edelleenkin liian sotkuisia yksityiskohtaiseen käsittelyyn
- Tavallisesti tyydytään tarkastelemaan funktion **kertaluokkaa**
- Olkoot $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$ mv. funktioita.
 - $f = O(g)$: f on (korkeintaan) *kertaluokkaa* g , jos $\exists c, n_0 \in \mathbb{N}$:

$$f(n) \leq c \cdot g(n), \quad \forall n \geq n_0$$
 - $f = \Theta(g)$: f ja g ovat *samaa kertaluokkaa*, jos $f = O(g)$ ja $g = O(f)$

- $f = o(g)$: f on *alempaa kertaluokkaa* kuin g , jos $\forall c > 0: \exists n_c \in \mathbb{N}$:

$$f(n) < c \cdot g(n), \quad \forall n \geq n_c$$
- $f = \Omega(g)$: g on *alaraja* f :lle, jos $\exists c > 0$: äärettömän monella $n \in \mathbb{N}$:

$$f(n) \geq c \cdot g(n)$$
- $f = o(g) \Leftrightarrow f = O(g) \wedge g \neq O(f)$
 $f \neq \Omega(g) \Leftrightarrow f = o(g)$
- Usein puhutaan hieman epätäsmällisemmin sanoen
 - "funktio $n!$ on kertaluokkaa $O(n^n)$ " tai
 - " $2n^2$ on kertaluokkaa n^2 "

Lemma 7.1

1. $\log_a n = \Theta(\log_b n) \quad \forall a, b > 0,$
2. $n^a = o(n^b)$, jos $a < b$,
3. $2^{an} = o(2^{bn})$, jos $a < b$,
4. $\log_a n = o(n^b) \quad \forall a, b > 0,$
5. $n^a = o(2^{bn}) \quad \forall a, b > 0,$
6. $c \cdot f(n) = \Theta(f(n)) \quad \forall c > 0$ ja funktioilla f ,
7. $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$ kaikilla funktioilla f ja g sekä
8. jos $p(n)$ on aidosti astetta r oleva polynomi, niin $p(n) = \Theta(n^r)$

□

Lemma 7.2 Olkoot $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$ mv. funktioita. Jos raja-arvo

$$L = \lim_{n \rightarrow \infty} f(n)/g(n)$$

on olemassa ja

1. $0 < L < \infty \Rightarrow f = \Theta(g)$
2. $L = 0 \Rightarrow f = o(g)$
3. $L > \infty \Rightarrow g = o(f)$
4. $L < \infty \Rightarrow f = O(g)$
5. $L > 0 \Rightarrow f = \Omega(g)$

□



- Tarkastellaan kielen $A = \{ 0^k 1^k \mid k \geq 0 \}$ tunnistamista
- Paljonko aikaa yksinauhainen Turingin kone vaatii A :n tunnistamiseksi?
- Syötteellä w
 1. Käy w läpi, hylkää syöte, jos merkki 0 löytyy merkin 1 oikealta puolelta
 2. Toista niin kauan kuin molempia merkkejä on nauhalla Käy läpi nauhaa merkiten käsitellyksi yksi 0 ja yksi 1
 3. Jos kaikki nollat (ykköset) on merkitty ja ykkösiä (nollia) on vielä jäljellä, hylkää syöte. Muuten (kaikki nollat ja ykköset on merkitty) hyväksy syöte



- A :n tunnistajakoneen ensimmäinen vaihe vaatii n askelta syötteellä w , $|w| = n$
- Toiset n askelta kuluu, kun lukupää palautetaan nauhan alkuun
- Ensimmäinen vaihe siis vaatii $O(n)$ askelta
- Vaiheessa 2 0/1-parin merkitseminen vaatii $O(n)$ askelta ja korkeintaan $n/2$ tällaista pyyhkäisyä vaaditaan
- Kaikkiaan siis vaihe 2 vaatii $O(n^2)$ askelta
- Kolmas vaihe on taas lineaarisen askelten lukumäärän vaativa
- Kaikkiaan koneen toiminta siis vaatii

$$O(n) + O(n^2) + O(n) = O(n^2)$$

ajan



- Kielen A tunnistaminen ei kuitenkaan ole $\Omega(n^2)$ toiminto, vaan seuraava Turingin kone vaatii vain $O(n \log n)$ ajan
- Syötteellä w
 1. Hylkää syöte, jos merkki 0 löytyy merkin 1 oikealta puolelta
 2. Toista niin kauan kuin molempia merkkejä on nauhalla
 - a) Jos merkkejä on yhteensä pariton määrä, niin hylkää syöte
 - b) Poista joka toinen 0 alkaen ensimmäisestä 0 :sta. Toimi samoin merkkien 1 kanssa
 3. Jos kaikki merkit 0 ja 1 on poistettu, hyväksy syöte, muuten hylkää se



- Eo. Turingin kone puolittaa joka kierroksella käsittelemättömien nollien määrän (poistaen jakojäännöksen)
- Kunkin kierroksen pituus on $O(n)$ ja kierroksia on korkeintaan $1 + \log n$
- Kaikkiaan aikaa kuluu siis $O(n \log n)$
- Koneen oikeellisuus: jos alunperin esim. 7 nollaa ja 6 ykköstä, niin hylätään parittomuustestillä heti
- Jos 7 nollaa ja 5 ykköstä, niin määrien ensimmäisen puolituksen jälkeen on jäljellä 3 nollaa ja 2 ykköstä ja parittomuustesti ottaa kiinni
- 7 nollaa ja 3 ykköstä, vaatii kaksi puolituskierrosta ennen kuin parittomuustesti ottaa kiinni



- Yksinauhainen Turingin kone ei voi tunnistaa kieltä A asymptoottista aikaa $O(n \cdot \log n)$ tehokkaammin
- Sen sijaan kaksinauhaisella koneella tunnistaminen sujuu lineaarisessa ajassa
- Syötteellä w
 1. Hylkää syöte, jos merkki 0 löytyy merkin 1 oikealta puolelta
 2. Kopioi kaikki merkit 0 nauhalle 2
 3. Käy läpi merkit 1 , kunkin kohdalla merkiten yksi 0 nauhalta 2 käsittelyksi. Jos nollat loppuvat ennen ykkösiä, hylkää w
 4. Jos kaikki nollat on merkitty hyväksy w , jos nollija on vielä jäljellä, hylkää w



- Laskennan universaalit mallit eivät siis ole yhtä tehokkaita
- Vaativuusteoriassa on merkitystä sillä mitä laskennan mallia käytetään
- Determinististen laskennan mallien tehokkuuserot eivät kuitenkaan ole dramaattisia
- Olkoon $t: \mathbb{N} \rightarrow \mathbb{R}^+$ mv. funktio
- Kieli A voidaan tunnistaa ajassa t , jos on olemassa *deterministinen* Turingin kone M s.e.
 - $L(M) = A$ ja
 - $\text{time}_M(n) \leq t(n)$ kaikilla n
- Formaalien kielten aikavaativuusluokka

$$\text{DTIME}(t) = \{ A \mid A \text{ voidaan tunnistaa ajassa } t \}$$

- Yksinauhaisella koneella voidaan simuloida moninauhaista konetta (lause 3.13)
- Yhden k -nauhaisen koneen askelen simulointi vaatii $O(t(n))$ ajan, missä $t(n) \geq n$ on moninauhaisen koneen vaativuusfunktio
- Askelia on kaikkiaan $O(t(n))$, joten

Lause 7.8 Jokaisella ajassa $t(n)$ toimivalla moninauhaisella Turingin koneella on olemassa ekvivalentti yksinauhainen kone, joka vaatii $O(t^2(n))$ ajan.

- Epädeterministisen Turingin koneen N laskennan vaativuus: $\text{time}_N(w)$ on pisimmän laskennan ($q_0 w \ggg_N \dots$) pituus

- Epädeterministisen koneen laskentoja syötteellä w on usein hyödyllistä ajatella laskentapuuna.
- Puun lehdissä laskenta pysähtyy. Kone hyväksyy $w:n$, jos jokin lehti vastaa hyväksyvää lopputilaa.
- Pahimman tapauksen aikavaativuus:

$$\text{time}_N(n) = \max_{|w|=n} \text{time}_N(w)$$

- Kieli A voidaan tunnistaa epädeterministisesti ajassa t , jos on olemassa epädeterministinen Turingin kone N s.e.

- $L(N) = A$ ja
- $\text{time}_N(n) \leq t(n) \quad \forall n$

- Epädeterministiset vaativuusluokat:

$$\text{NTIME}(t) = \{ A \mid A \text{ voidaan tunnistaa epädeterm. ajassa } t \}$$

Lause 7.11 Jokaisella ajassa $t(n)$ toimivalla yksinauhaisella epädeterministisellä Turingin koneella on olemassa ekvivalentti yksinauhainen deterministinen kone, joka vaatii $2^{O(t(n))}$ ajan.

Todistus. Lauseen 3.16 deterministinen Turingin kone M käy systemaattisesti läpi epädeterministisen koneen N laskentapuuta.

Olkoon N :n toiminta-aika $t(n)$, jolloin laskentapuun jokaisen haaran pituus n :n pituisella syötteellä on korkeintaan $t(n)$.

Solmujen haarautumisaste määräytyy N :n siirtymäfunktion tarjoamien siirtymävaihtoehtojen lukumäärän mukaan. Olkoon $b \geq 2$ yläraja tälle.

Puun lehtien lukumäärä on korkeintaan $b^{t(n)}$.

Puun läpikäyminen vaatii pahimmassa tapauksessa kaikkien sen solmujen käsittelyä. Solmuja on kaikkiaan alle kaksi kertaa lehtien maksimimäärä, eli luokkaa $O(b^{t(n)})$. Joten kolminauhaisen koneen M asympotoottisesti vaatima aika on $2^{O(t(n))}$.

Lauseen 7.8 perusteella yksinauhainen deterministinen kone voidaan toteuttaa $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$ ajassa. \square

- Yksi- ja moninauhaisten koneiden tehokkuusero on siis korkeintaan t :n neliö, siis polynominen t :n suhteen
- Determinististen ja epädeterminististen koneiden tehokkuusero puolestaan on eksponentiaalinen t :n suhteen

7.1 Luokka P

- Tässä yhteydessä ajatellaan polynomisen tehokkuuseron olevan pieni ja eksponentiaalisen suuri
- Vaativuusfunktioissa polynomisen ajan ajatellaan olevan käyttökelpoisen, kun taas eksponentiaalinen useimmiten on käyttökelvoton
- Tyypillisimmin eksponentiaaliseen vaativuuteen törmätään kattavaa hakua tehtäessä
- Kaikissa kohtuullisissa laskennan malleissa voidaan ratkoa samat ongelmat polynomisessa ajassa
- Mallit ovat tässä mielessä ekvivalentteja

- Luokka P on:

$$P = \bigcup \{ \text{DTIME}(t) \mid t \text{ on polynomi} \}$$

$$= \bigcup_{k \geq 0} \text{DTIME}(n^k + k)$$

- Luokka P on keskeinen, koska
 - se on laskennan mallista riippumaton (Turingin koneen kanssa polynomisesti ekvivalenttien mallien suhteen) ja
 - se jotakuinkin vastaa tietokoneella realistisesti ratkaistavissa olevien ongelmien joukkoa
- Korkea-asteiset polynomiset algoritmit eivät tietenkään ole kovin käyttökelpoisia, mutta niitä esiintyy vain harvoin
- Luokan P asema on luokan REC asemaa laskettavuusteoriassa vastaava

Ratkeavia ongelmia

- Olkoon G suunnattu verkko, jossa on solmut s ja t
- Ongelma PATH: Onko G :ssä suunnattu polku s :stä t :hen?
- Kattava haku kävisi kaikki G :n polut läpi tutkien onko jokin niistä etsitty polku
- Pahimmassa tapauksessa polkuja on kuitenkin eksponentiaalinen lukumäärä (verkon solmujen suhteen)
- Tarvitaan siis hienovaraisempi algoritmi
 - Asetetaan merkki solmuun s
 - Niin kauan kuin merkittyjen solmujen joukko kasvaa: tutki kaikki verkon kaaret, jos kaari (a, b) kulkee merkitystä solmusta a merkitsemättömään b , lisää b merkittyjen joukkoon

- Jos t kuuluu merkittyjen solmujen joukkoon hyväksy syöte, muuten hylkää se
- Olkoon m verkon G solmujen lukumäärä
- Edellinen algoritmin toista askelta toistetaan korkeintaan m kertaa, koska vain niin kauan voidaan lisätä uusia alkioita merkittyjen joukkoon
- Kaikkiaan siis $m + 2$ helposti polynomisessa ajassa toteutettavaa askelta
- PATH \in P



- Kaksi kokonaislukua ovat *keskenään alkulukuja* (relative primes), jos niillä ei ole lukua 1 suurempia kokonaislukutekijöitä
- Esim. 10 ja 21 ovat keskenään alkulukuja, vaikka kumpikaan ei ole alkuluku
- Sen sijaan 9 ja 21 ovat molemmat 3:lla jaollisia
- Tarkastellaan kokonaislukuparien, jotka ovat keskenään alkulukuja, tunnistamisongelmaa **RP**
- Jälleen suoraviivaisin haku osoittautuu tehottomaksi
- Kaikkien mahdollisten jakajien läpikäyminen on eksponentiaalisen ajan vaativaa, koska mahdollisia jakajia on annettujen lukujen binääriesityksen pituuteen nähden eksponentiaalinen määrä



- Ikivanha **Eukleideen algoritmi** lukuparin suurimman yhteisen tekijän (**syT**) määrittämiseksi antaa mahdollisuuden ratkaista **RP** tehokkaasti
- Esim. **syT**(18, 24) = 6
- Luonnollisesti x ja y ovat keskenään alkulukuja joss **syT**(x, y) = 1
- Olkoot x ja y luonnollisia lukuja binääriesityksenä
 1. Toista kunnes $y = 0$
 - a) $x \leftarrow x \bmod y$
 - b) vaihda x ja y keskenään
 2. Palauta x



- Esim. $x = 18, y = 24$
 - $x \leftarrow 18 \bmod 24 = 18$
 - $x \leftarrow 24, y \leftarrow 18$
 - $x \leftarrow 24 \bmod 18 = 6$
 - $x \leftarrow 18, y \leftarrow 6$
 - $x \leftarrow 18 \bmod 6 = 0$
 - $x \leftarrow 6, y \leftarrow 0$
- Palauta $x = 6$



- Kun askel 1 on suoritettu ensimmäisen kerran pätee varmasti $x > y$
- Sen jälkeen jokainen askelen 1(a) suoritus vähintään puolittaa x :n arvon
 - $x \geq 2y \Leftrightarrow x \bmod y < y \leq x/2$
 - $x < 2y \Leftrightarrow x \bmod y = x - y < x/2$
- Joka toinen askelen 1 suoritus puolittaa toisen alkuperäisistä arvoista x ja y , joten pahimmassa tapauksessa kierroksia on $\min\{2\log_2 x, 2\log_2 y\}$
- Lukujen binääriesitykset ovat logaritmisia pituudeltaan, joten Eukleideen algoritmi vaatii lineaarisen ajan