Laskennan vaativuus

T. Karvi

5.9.2006

Vaativuuden mittaaminen

- Tarkastellaan jotain kieltä A.
- Miten paljon Turingin kone kuluttaa aikaa tunnistaessaan A:n?
- Idea:
  - Laaditaan kielen A tunnistava Turingin kone (pseudokieli riittää).
  - Lasketaan, montako kertaa lukupää liikkuu nauhalla.
  - Koska tulos riippuu yleensä monesta parametrista, lausutaan tulos $\mathcal{O}$-notaatiolla.

Vaativuuden mittaaminen

Example

Tarkastellaan kieltä $A = \{0^k 1^k | k \geq 0\}$. Sen tunnistava Turingin kone toimii seuraavasti:

1. Selaa syötenauhaa ja hylkää syöte, jos 0 esiintyy ykkösen oikealla puolella.
2. Toista seuraavaa niin kauan, kuin sekä ykkösiä että nollia on nauhalla:
3. Selaa nauhaa ja poista yksi nolla ja yksi ykkönen.
4. Jos nauhalla on nollia tai ykkösiä, hylkää syöte, muuten hyväksy.

Vaativuuden mittaaminen

Määritelmä

Olkoon M deterministinen Turingin kone, joka pysähtyy kaikilla syötteillä. M:n laskenta-aika on funktio $f : \mathbf{N} \longrightarrow \mathbf{N}$, missä $f(n)$ on maksimimäärä askelia, joita M käyttää millä tahansa n:n pituisella syötteellä. Jos f on M:n laskenta-aika, sanotaan, että M toimii ajassa f ja että M on ajan f Turingin kone.

Vaativuuden mittaaminen

- Aika f riippuu Turingin koneesta: yksi- vai moninauhainen, -urainen.
- Lopullinen lauseke voi olla monimutkainen ja se voi vaihdella paljon eri syötteillä.
- Siksi tarkastellaan (tällä kurssilla) vain pahimman tapauksen analyysiä ja
- yksinkertaistetaan lausekkeita asymptoottisella $\mathcal{O}$-notaatiolla.

$\mathcal{O}$-notaatiot

Määritelmä

Olkoot f ja g funktioita $\mathbf{N} \longrightarrow \mathbf{R}^+$. Tällöin $f(n) = \mathcal{O}(g(n))$, jos on olemassa sellaiset positiiviset kokonaisluvut c ja n_0, että jokaisella $n \geq n_0$

$$f(n) \leq cg(n).$$

Jos $f(n) = \mathcal{O}(g(n))$, niin g on f:n yläraja tai täsmällisemmin asymptoottinen yläraja.

$\mathcal{O}$-notaatiot

- Voidaan myös tulkita, että $\mathcal{O}(g)$ on funktioluokka ja että $f \in \mathcal{O}(g)$ sen sijasta, että merkitään $f(n) = \mathcal{O}(g(n))$.

$\mathcal{O}$-notaatiot

- Voidaan myös tulkita, että $\mathcal{O}(g)$ on funktioluokka ja että $f \in \mathcal{O}(g)$ sen sijasta, että merkitään $f(n) = \mathcal{O}(g(n))$.
- Ideana $\mathcal{O}$-merkinnässä on, että vain nopeimmin kasvava termi tarvitsee ottaa huomioon. Jos esimerkiksi $f(n) = 6n^3 + 2n^3 + 20n + 45$, niin $f(n) = \mathcal{O}(n^3)$.

$\mathcal{O}$-notaatiot

- Voidaan myös tulkita, että $\mathcal{O}(g)$ on funktioluokka ja että $f \in \mathcal{O}(g)$ sen sijasta, että merkitään $f(n) = \mathcal{O}(g(n))$.
- Ideana $\mathcal{O}$-merkinnässä on, että vain nopeimmin kasvava termi tarvitsee ottaa huomioon. Jos esimerkiksi $f(n) = 6n^3 + 2n^3 + 20n + 45$, niin $f(n) = \mathcal{O}(n^3)$.
- Samoin jos $f_1(n) = 5n^3 + 2n^2 + 22n + 6$, niin $f_1(n) = \mathcal{O}(n^3)$, sillä voidaan valita $c = 6$ ja $n_0 = 10$. Nimittäin tällöin

$$5n^3 + 2n^2 + 22n + 6 \leq 6n^3$$

(diff. int. perustelu!).

$\mathcal{O}$-notaatiot

- Voidaan myös tulkita, että $\mathcal{O}(g)$ on funktioluokka ja että $f \in \mathcal{O}(g)$ sen sijasta, että merkitään $f(n) = \mathcal{O}(g(n))$.
- Ideana $\mathcal{O}$-merkinnässä on, että vain nopeimmin kasvava termi tarvitsee ottaa huomioon. Jos esimerkiksi $f(n) = 6n^3 + 2n^3 + 20n + 45$, niin $f(n) = \mathcal{O}(n^3)$.
- Samoin jos $f_1(n) = 5n^3 + 2n^2 + 22n + 6$, niin $f_1(n) = \mathcal{O}(n^3)$, sillä voidaan valita $c = 6$ ja $n_0 = 10$. Nimittäin tällöin

$$5n^3 + 2n^2 + 22n + 6 \leq 6n^3$$

(diff. int. perustelu!).
- Helpommin

$$5n^3 + 2n^2 + 22n + 6 \leq 5n^3 + 2n^3 + 22n^3 + 6n^3 = 73n^3 = \mathcal{O}(n^3),$$

kun $n > 1$.

$\mathcal{O}$-notaatiot

- Voidaan myös tulkita, että $\mathcal{O}(g)$ on funktioluokka ja että $f \in \mathcal{O}(g)$ sen sijasta, että merkitään $f(n) = \mathcal{O}(g(n))$.
- Ideana $\mathcal{O}$-merkinnässä on, että vain nopeimmin kasvava termi tarvitsee ottaa huomioon. Jos esimerkiksi $f(n) = 6n^3 + 2n^3 + 20n + 45$, niin $f(n) = \mathcal{O}(n^3)$.
- Samoin jos $f_1(n) = 5n^3 + 2n^2 + 22n + 6$, niin $f_1(n) = \mathcal{O}(n^3)$, sillä voidaan valita $c = 6$ ja $n_0 = 10$. Nimittäin tällöin

$$5n^3 + 2n^2 + 22n + 6 \leq 6n^3$$

  (diff. int. perustelu!).
- Helpommin

$$5n^3 + 2n^2 + 22n + 6 \leq 5n^3 + 2n^3 + 22n^3 + 6n^3 = 73n^3 = \mathcal{O}(n^3),$$

  kun $n > 1$.
- Myös $f_1(n) = \mathcal{O}(n^4)$.
- Sen sijaan $f_1(n) \neq \mathcal{O}(n^2)$.

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.
- Sen jälkeen nauhapää asetetaan alkuun, n askelta (at most!).

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.
- Sen jälkeen nauhapää asetetaan alkuun, n askelta (at most!).
- Alkuvalmistelut yhteensä $2n$ askelta eli $\mathcal{O}(n)$ askelta.

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.
- Sen jälkeen nauhapää asetetaan alkuun, n askelta (at most!).
- Alkuvalmistelut yhteensä $2n$ askelta eli $\mathcal{O}(n)$ askelta.
- Askelissa 2 ja 3 etsitään 0 ja 1; edelleen pahimmassa tapauksessa $\mathcal{O}(n)$ askelta. Etsintä tehdään $n/2$ kertaa. Siten aikaa kuluu $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$.

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.
- Sen jälkeen nauhapää asetetaan alkuun, n askelta (at most!).
- Alkuvalmistelut yhteensä $2n$ askelta eli $\mathcal{O}(n)$ askelta.
- Askelissa 2 ja 3 etsitään 0 ja 1; edelleen pahimmassa tapauksessa $\mathcal{O}(n)$ askelta. Etsintä tehdään $n/2$ kertaa. Siten aikaa kuluu $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$.
- Lopuksi vielä varmistetaan, ettei jäljellä ole bittejä, aikavaatimus $\mathcal{O}(n)$.

Turingin koneen aikavaatimus

- Alussa kone testaa, että syöte on muotoa 0^*1^*. Testi vaatii n askelta, jos syötteen pituus on n.
- Sen jälkeen nauhapää asetetaan alkuun, n askelta (at most!).
- Alkuvalmistelut yhteensä $2n$ askelta eli $\mathcal{O}(n)$ askelta.
- Askelissa 2 ja 3 etsitään 0 ja 1; edelleen pahimmassa tapauksessa $\mathcal{O}(n)$ askelta. Etsintä tehdään $n/2$ kertaa. Siten aikaa kuluu $(n/2)\mathcal{O}(n) = \mathcal{O}(n^2)$.
- Lopuksi vielä varmistetaan, ettei jäljellä ole bittejä, aikavaatimus $\mathcal{O}(n)$.
- Siten kokonaisaika on

$$\mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2).$$

$\mathcal{O}$-merkintä

- Huomattakoon, miten $\mathcal{O}$-merkintää käytetään aritmeettisissa lausekkeissa, esim. $\mathcal{O}(n^2) + \mathcal{O}(n)$.
- Tällaisessa lausekkeessa kukin $\mathcal{O}$-termi edustaa vastaavaa termiä, jonka kertoimesta ei välitetä. Termejä voidaan sieventää ottamalla huomioon vain dominoiva termi.

$\mathcal{O}$-merkintä

- Huomattakoon, miten $\mathcal{O}$-merkintää käytetään aritmeettisissa lausekkeissa, esim. $\mathcal{O}(n^2) + \mathcal{O}(n)$.
- Tällaisessa lausekkeessa kukin $\mathcal{O}$-termi edustaa vastaavaa termiä, jonka kertoimesta ei välitetä. Termejä voidaan sieventää ottamalla huomioon vain dominoiva termi.
- Samoin tulkitaan vaikkapa $2^{\mathcal{O}(n)}$.

$\mathcal{O}$-merkintä

- $\mathcal{O}$-merkintä käsittelee logaritmeja kätevästi.
- Koska

$$\log_b n = \log_2 n / \log_2 b,$$

kantaluku ei ole tärkeä $\mathcal{O}$-merkinnässä ja voidaan kirjoittaa yksinkertaisesti $\mathcal{O}(\log n)$.

$\mathcal{O}$-merkintä

- $\mathcal{O}$-merkintä käsittelee logaritmeja kätevästi.
- Koska

$$\log_b n = \log_2 n / \log_2 b,$$

  kantaluku ei ole tärkeä $\mathcal{O}$-merkinnässä ja voidaan kirjoittaa yksinkertaisesti $\mathcal{O}(\log n)$.
- Esim. $f(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2$.
- Tällöin $f(n) =$O(n log n).

Aikavaativuusluokat

Määritelmä

Olkoon $t : \mathbf{N} \longrightarrow \mathbf{N}$ *funktio. Määritellään aikavaativuusluokka* $TIME(t(n))$ *sellaisten kielten* L *joukoksi, jotka Turingin kone tunnistaa ajassa* $\mathcal{O}(t(n))$.

Aikavaativuusluokat

Määritelmä

Olkoon $t : \mathbf{N} \longrightarrow \mathbf{N}$ *funktio. Määritellään aikavaativuusluokka* $TIME(t(n))$ *sellaisten kielten* L *joukoksi, jotka Turingin kone tunnistaa ajassa* $\mathcal{O}(t(n))$.

Example

Olkoon $A = \{0^k 1^k | k \geq 0\}$. Tällöin $A \in \mathrm{TIME}(n^2)$. Onko tunnistus mahdollista tehdä asymptoottisesti nopeammin?

Seuraava Turingin kone M_2 tunnnistaa A:n asymptoottisesti nopeammin.

1. Selaa nauhaa ja hylkää syöte, jos 0 löytyy 1:n oikealta puolelta.
2. Toista niin kauan kuin nollia ja ykkösiä on nauhalla:
3. Selaa nauhaa ja testaa, että nollien ja ykkösten yhteislkm on parillinen.
4. Selaa nauhaa ja pyyhi yli joka toinen 0 ja joka toinen 1.
5. Jos nollia ja ykkösiä ei enää esiinny, hyväksy, muuten hylkää.

- Todetaan ensin, että M_2 todella tunnistaa A:n.
- Askeleessa 4 nollien määrä puolittuu joka kerralla; mahdollinen jakojääännös hylätään.
- Siten jos alunperin nollia on 13, askeleen 4 jälkeen niitä on jäljellä 6. Seuraavissa askelissa nollia esiintyy 3, 1 ja 0 kpl.
- Samoin tapahtuu ykkösten kohdalla.
- Askeleessa 3 tutkitaan nollien ja ykkösten lukumäärää. Jos nollia on parillinen määrä, ykkösiä täytyy olla myös parillinen määrä. Jos nollia on pariton määrä, myös ykkösiä täytyy olla pariton määrä.
- Jos nollat puolitetaan, myös ykkösten täytyy puolittua samalla tavalla (even, odd). Jos alunperin ykkösiä ja nollia on eri määrä, jossain vaiheessa yhteismääräksi tulee pariton luku, ja syöte voidaan hylätä.

- Analysoidaan aikavaatimus.
- Jokainen kierros vaatii ajan $\mathcal{O}(n)$.
- Kierroksia tehdään $1 + \log_2 n$ kpl korkeintaan, sillä joka kierroksella nollien ja ykkösten määrä puolittuu.
- Siten aikavaatimus on $(1 + \log_2 n)\mathcal{O}(n)$ eli $\mathcal{O}(n \log n)$.

Lineaarisen ajan ratkaisu

- Aikavaatimus voidaan parantaa lineaariseksi, jos otetaan käyttöön kaksinauhainen Turingin kone.
- Yksinauhaisella koneella ei tulosta voi enää parantaa. Itse asiassa voidaan osoittaa, että mikä tahansa kieli, joka voidaan tunnistaa yksinauhaisella Turingin koneella ajassa $\mathcal{O}(n \log n)$, on itse asiassa säännöllinen.
- Kaksinauhainen kone yksinkertaisesti kopioi nollat toiselle nauhalle ja vertaa sitten nollien lkm:ää ykkösten lkm:ään.
- Selvästi näin saatu Turingin kone käyttää aikaa $\mathcal{O}(n)$ eli on lineaarinen.

1. Selaa nauhaa ja hylkää syöte, jos nollia on ykkösen vasemmalla puolella.
2. Selaa 1. nauhalla nollia kunnes kohdataan ensimmäinen ykkönen. Kopioi nollat samalla nauhalle 2.
3. Selaa ykköset 1. nauhalla nauhan loppuun. Pyyhi yli nolla nauhalla 2 jokaista ykköstä kohti. Jos nollat loppuvat ennen ykkösiä, hylkää syöte.
4. Jos kaikki nollat on pyyhitty yli, hyväksy. Jos nollia on jäljellä, hylkää.

Turingin koneiden väliset simuloinnit

Palautetaan mieleen keskeisiä tuloksia eri Turingin koneiden välisistä suhteista.

Lause

Olkoon $t(n)$ funktio, jolla $t(n) \geq n$. Tällöin yksinauhainen Turingin kone voi simuloida moninauhaista, ajan $t(n)$ käyttävää Turingin konetta ajassa $\mathcal{O}(t^2(n))$.

Todistus.

Laskennan mallit -kurssi. □

Tarkastellaan Turingin koneista vielä epädeterminististä versiota ja sen simulointia determinisellä koneella. Palautetaan ensin mieleen epädetermistisen koneen määritelmä.

Epädeterminstinen Turingin kone

Määritelmä

Epädeterministinen Turingin kone on muuten samanlainen kuin yksinauhainen Turingin kone, mutta siirtymäfunktio on muotoa

$$\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q \times \Sigma \times \{L, R\}).$$

Siis on useita vaihtoisia siirtymiä samalla lähtötilalla ja samalla merkillä.

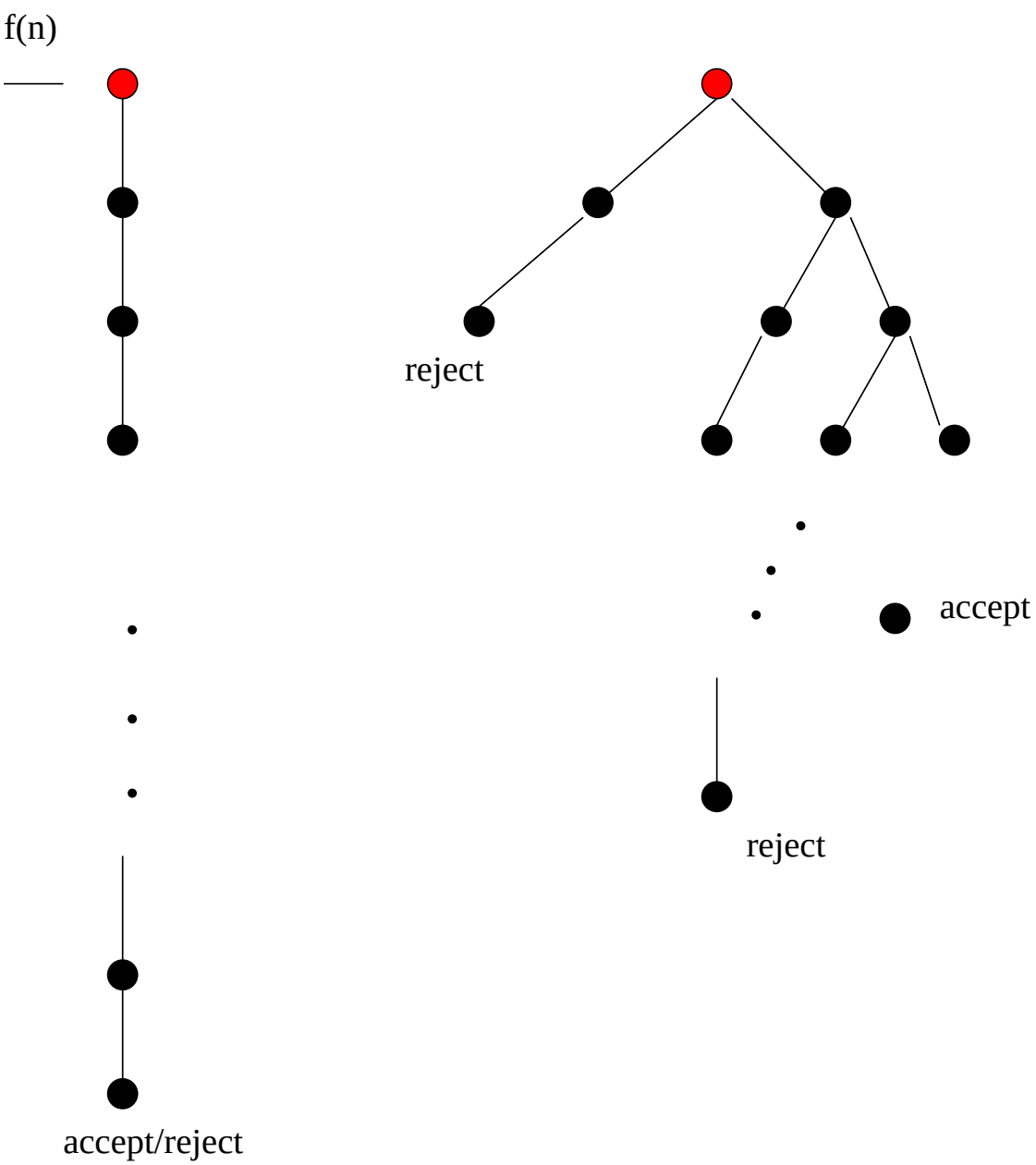
Epädeterministisen koneen laskentaa voidaan havainnollistaa puulla, jonka haarat kuvaavat koneen eri mahdollisuuksia kussakin tilanteessa siirtymäfunktion mukaan. Jos jokin haara johtaa hyväksyvään tilaan, syöte hyväksytään, muuten hylätään. Sekä formalismi että ajattelumalli on hyvin lähellä tavallista äärellistilaista epädeterminististä automaattia.

Epädeterministisen koneen aikavaatimus

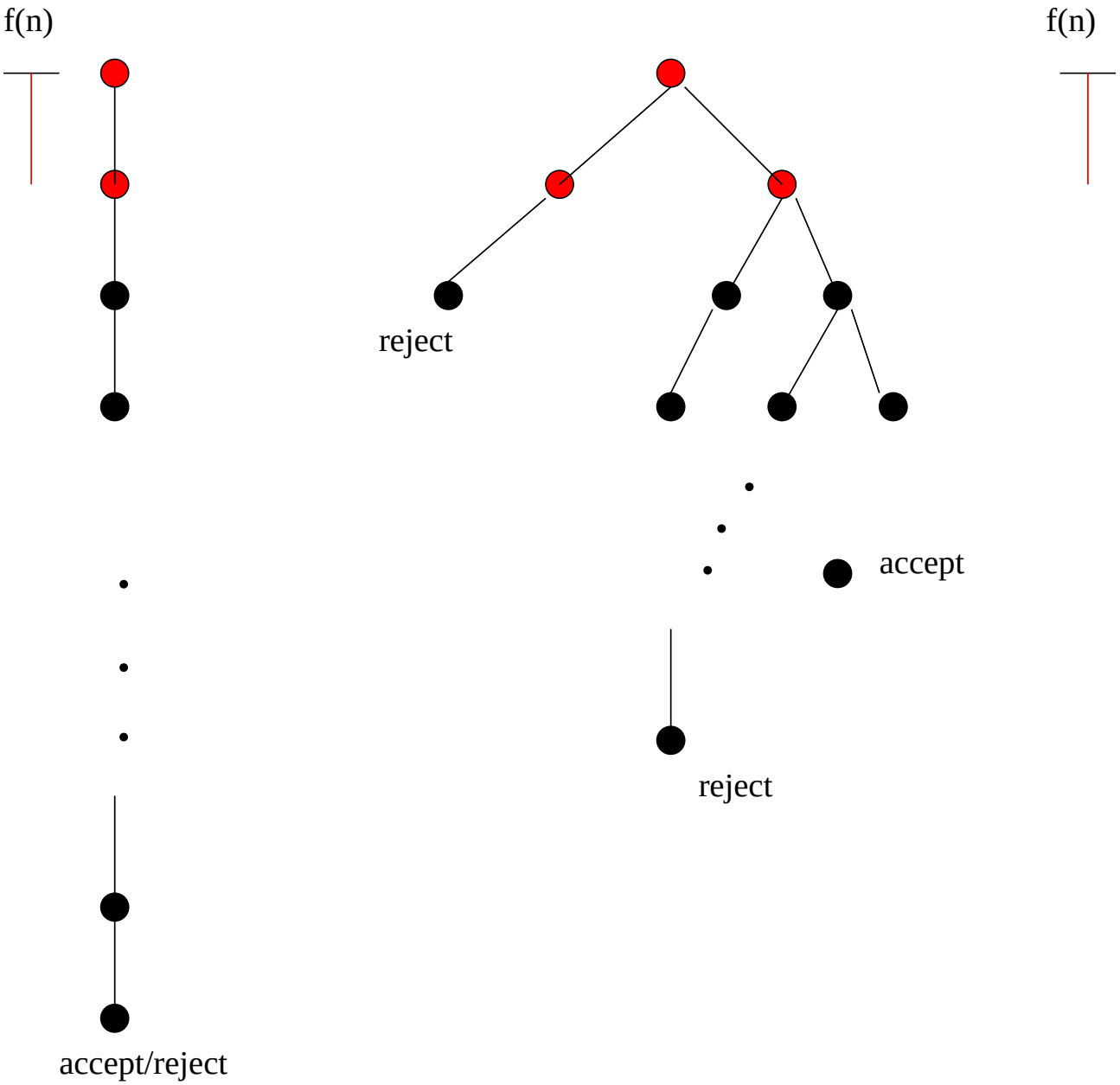
Määritelmä

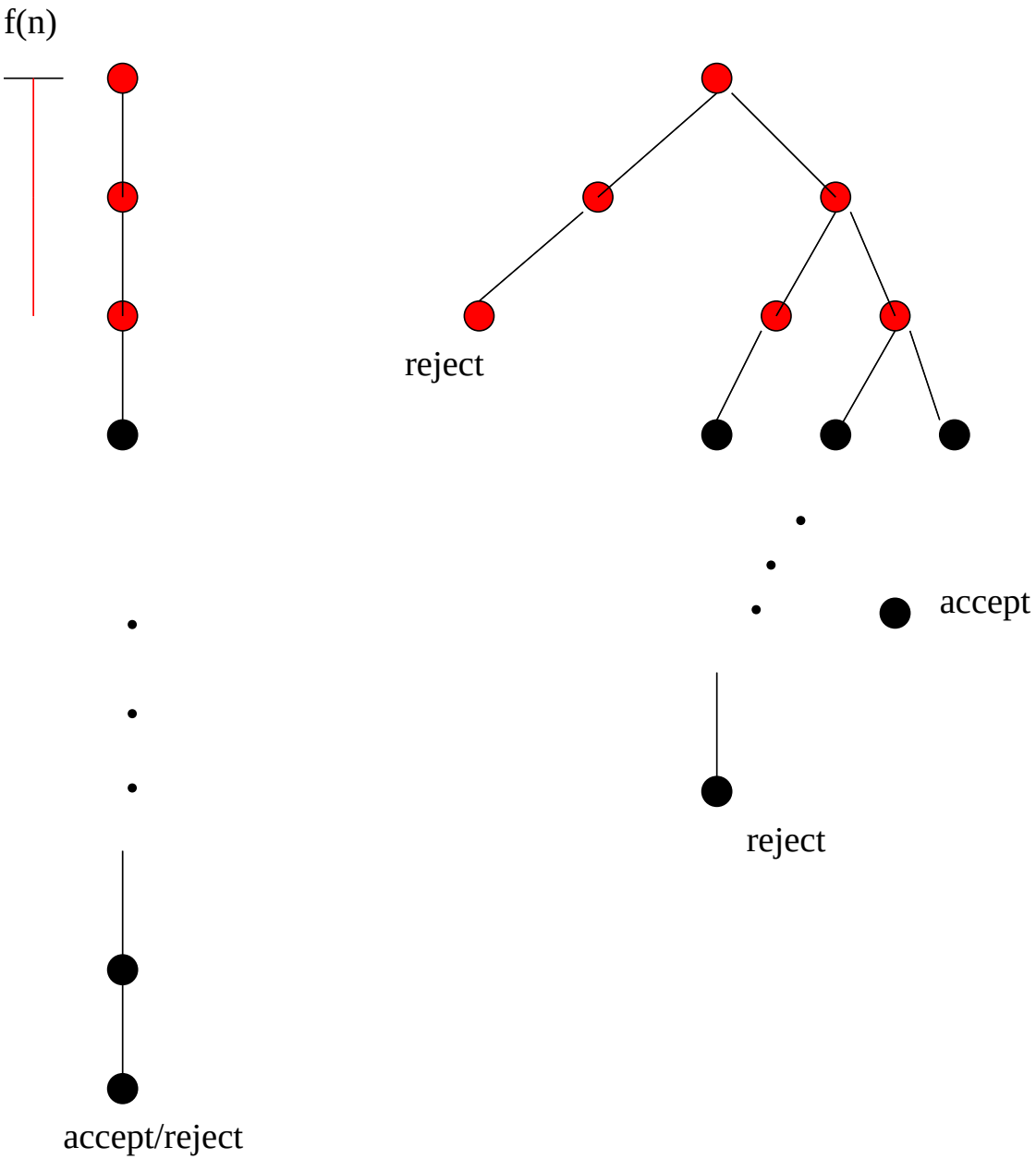
Olkoon N epädeterministinen Turingin kone, joka tunnistaa kielen. Tällöin N:n ajoaika on funktio $f : \mathbf{N} \longrightarrow \mathbf{N}$, *missä* $f(n)$ *on n:n pituisilla syötteillä käytettyjen askelien maksimaalinen määrä, kun huomioon otetaan kaikki epädeterministisen koneen laskentapolut kyseisellä syötteellä.*

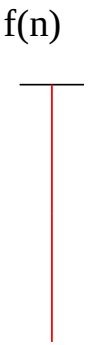
- Seuraavat kuvat valaisevat deterministisen ja epädeterministisen koneen laskenta-aikojen eroavaisuuksia.
- Huomattakoon, että epädeterministisen koneen laskenta-aika ei vastaa minkään todellisen koneen laskenta-aikaa.
- Se on "vain"matemaattinen käsite, jonka avulla laskennan vaativuutta voidaan luonnehtia kätevästi, jopa reaalisten, käytännöllisten probleemojen yhteydessä.

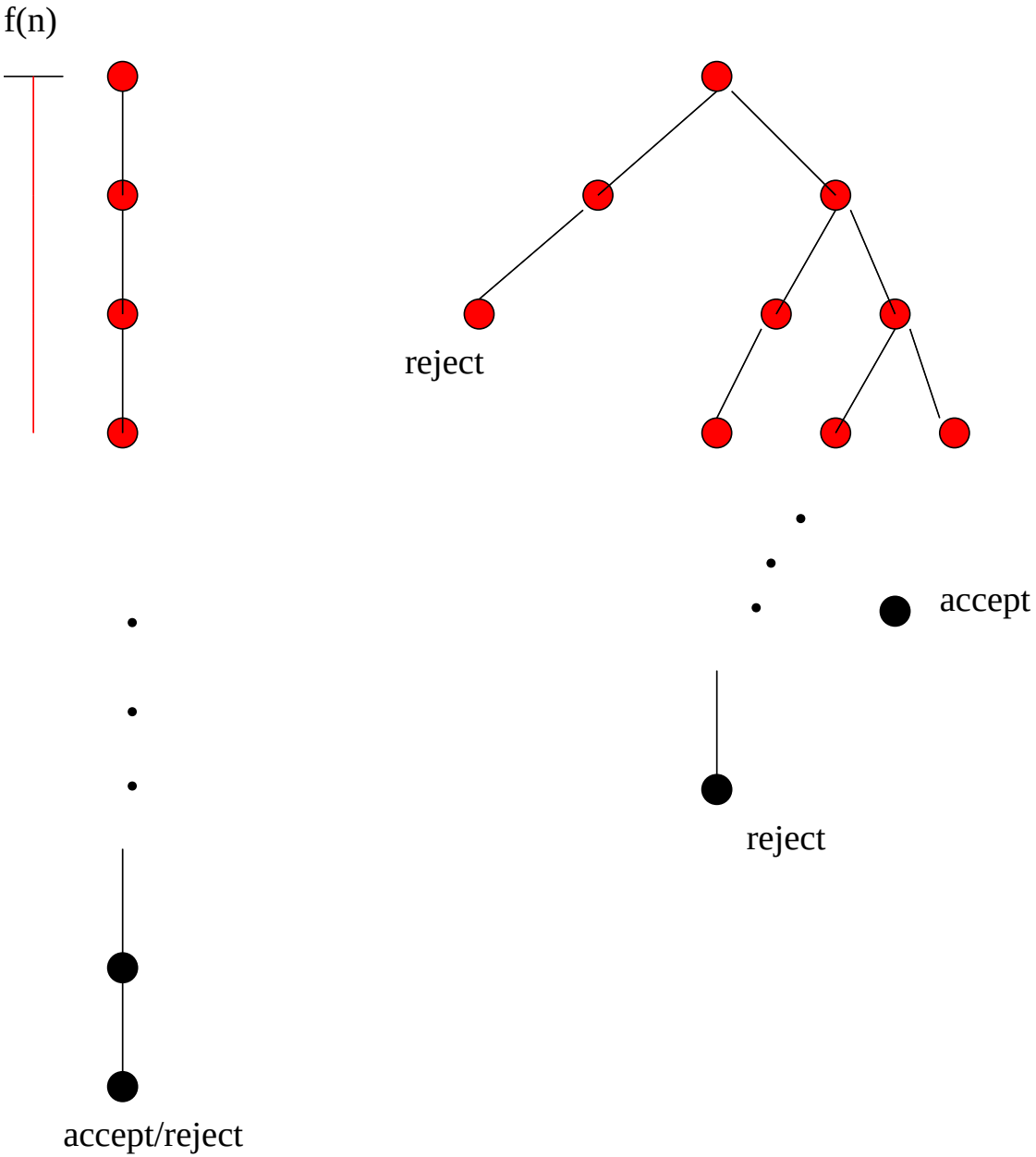


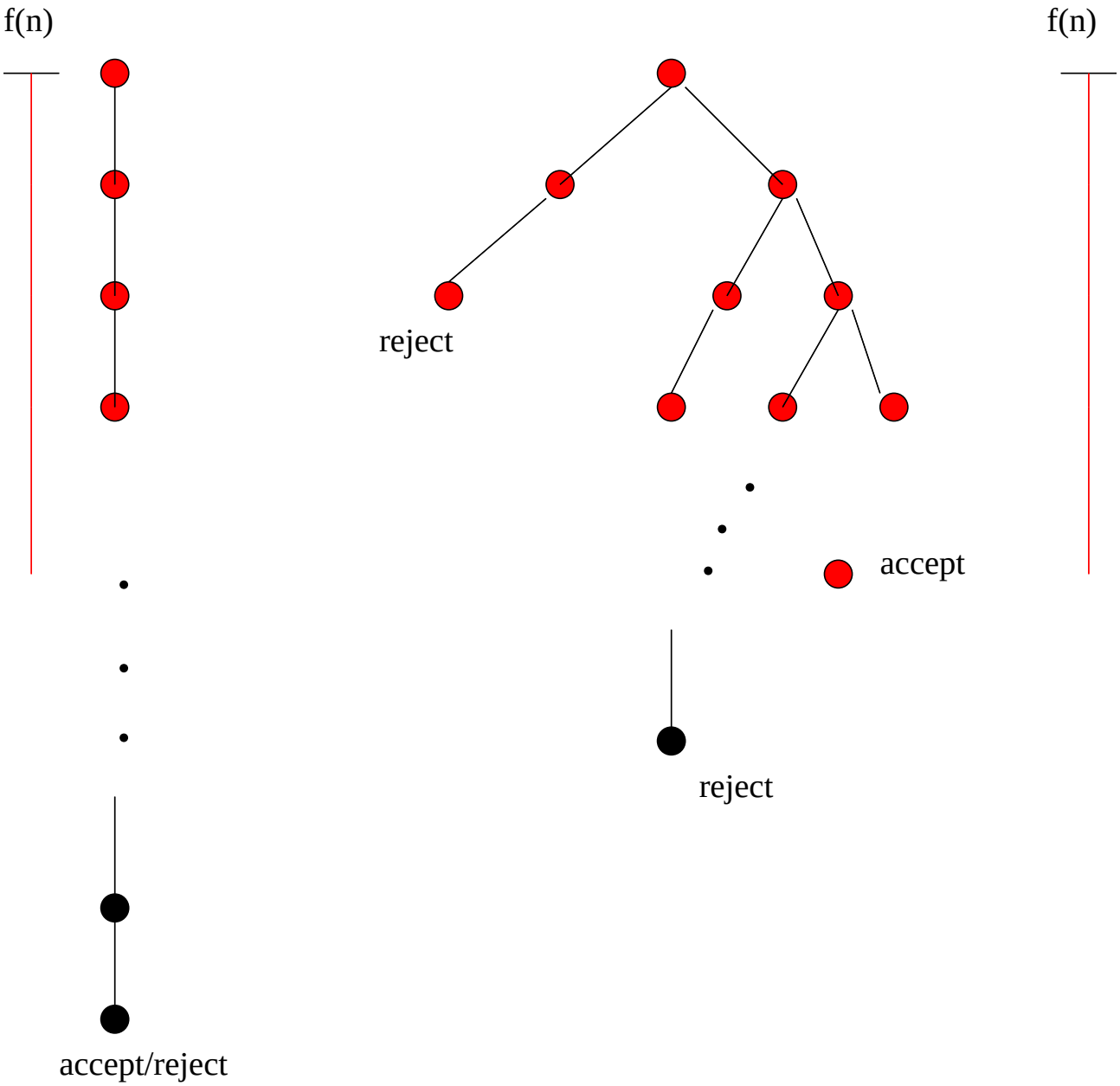
f(n)

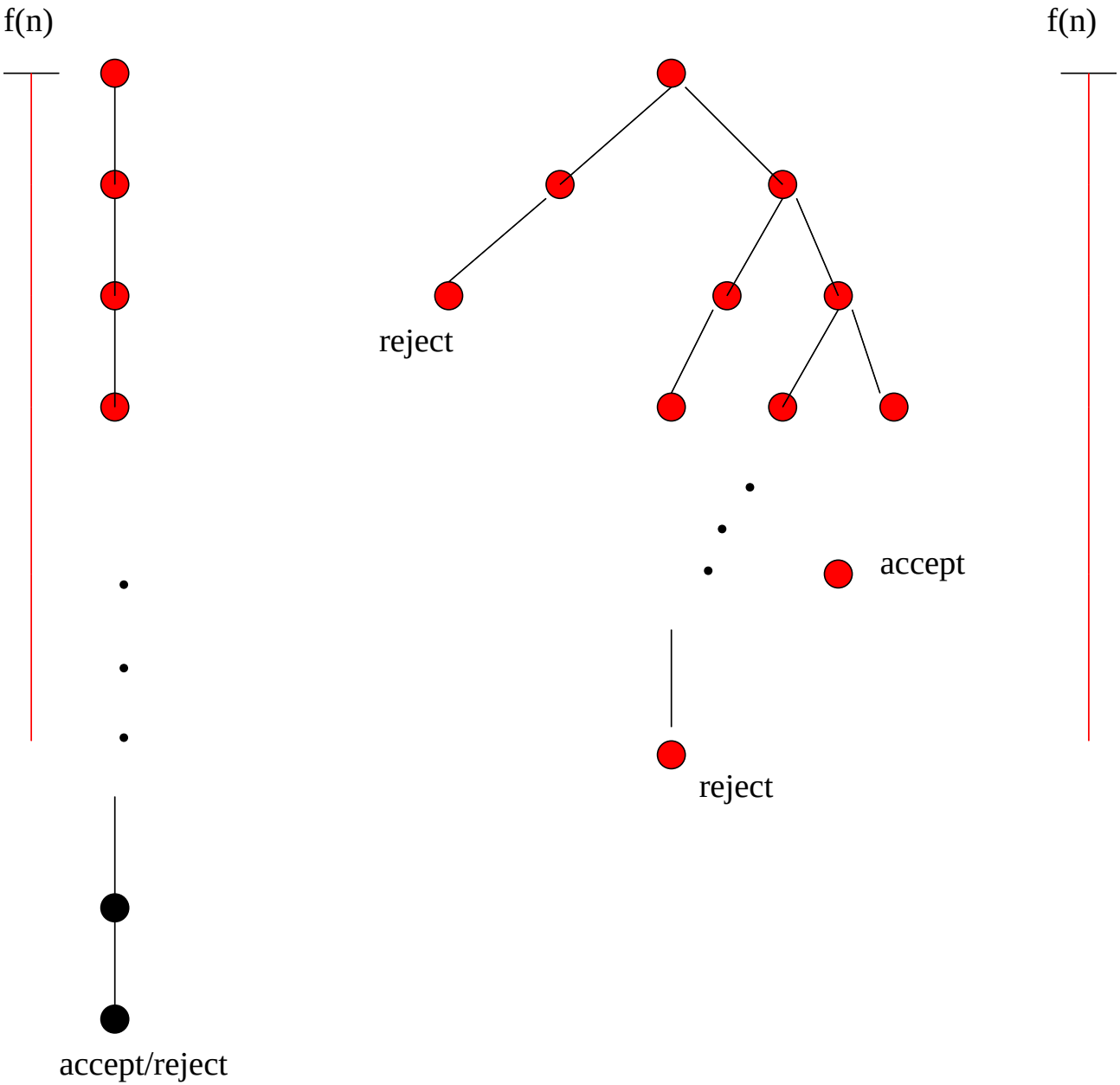

f(n)
reject
accept
reject
f(n)
accept/reject

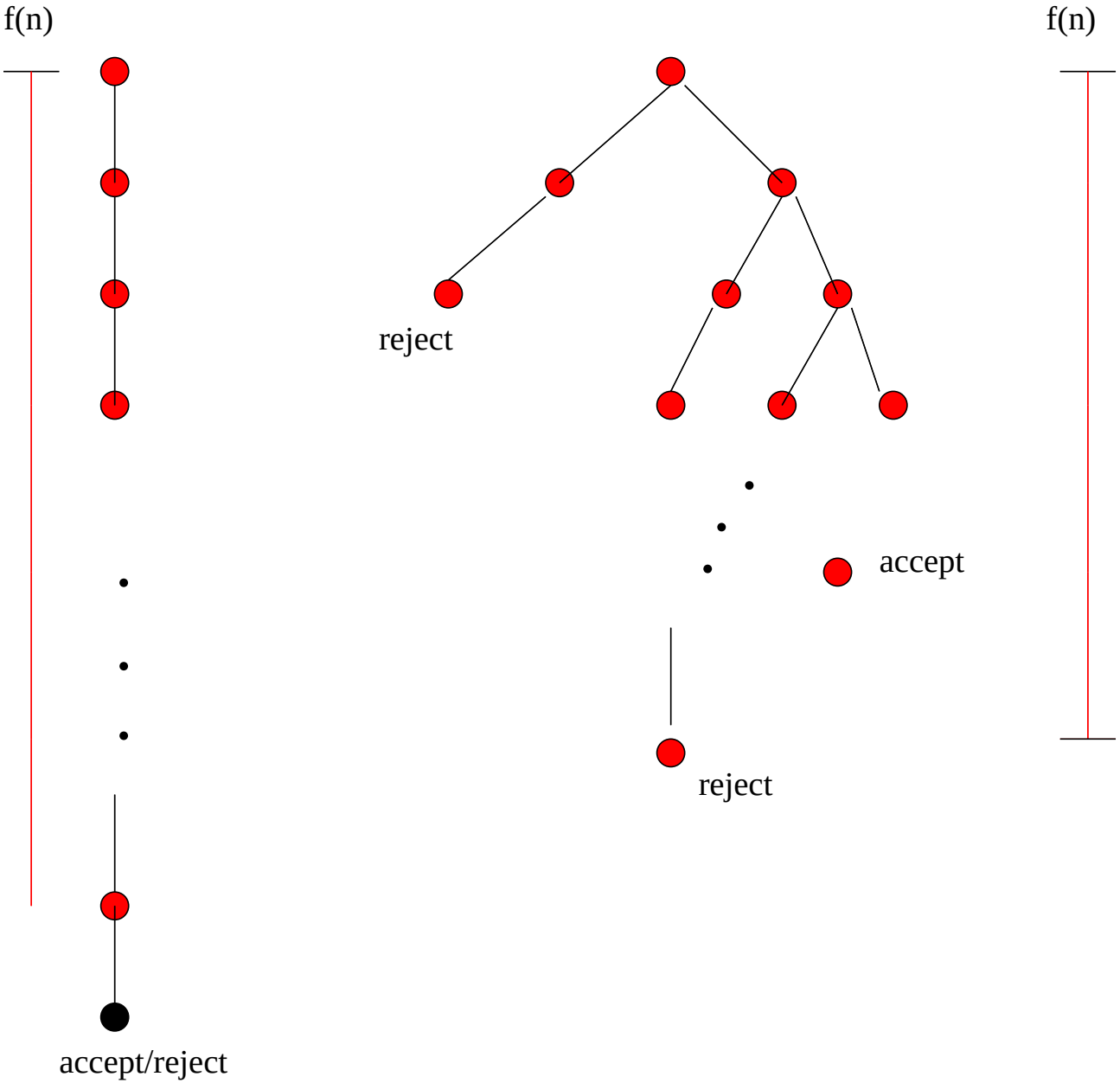

f(n)
reject
accept
reject
accept/reject

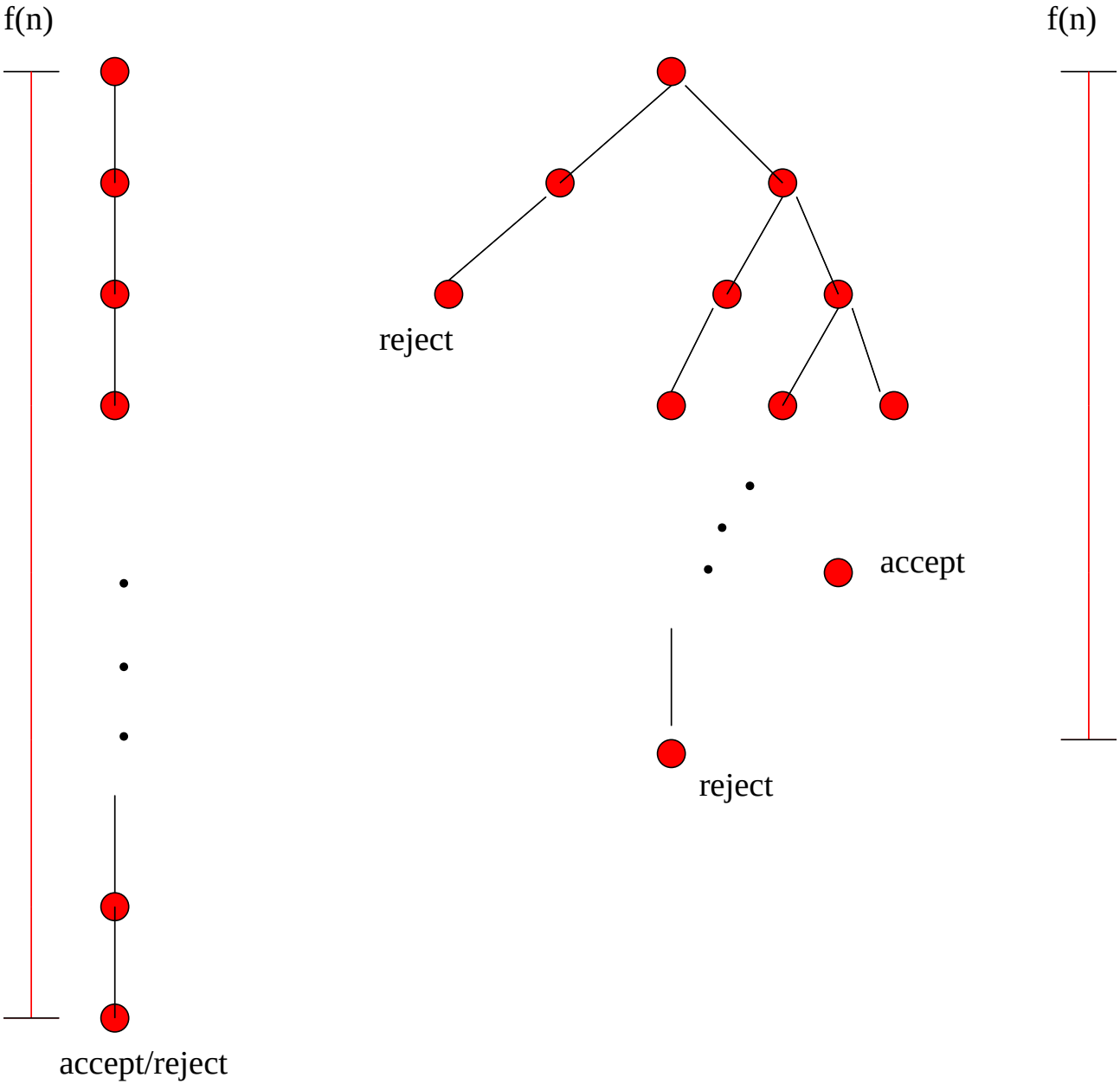

f(n)


f(n)
reject
f(n)
accept
reject
accept/reject


f(n)
reject
accept
reject
f(n)
accept/reject


f(n)
reject
accept
reject
f(n)
accept/reject


f(n)
reject
accept
reject
f(n)
accept/reject


f(n)
reject
accept
reject
f(n)
accept/reject

Epädeterministisen ja deterministisen koneen vastaavuus

Tiedämme jo Laskennan mallit -kurssilta, että deterministisellä koneella voidaan simuloida epädeterminstistä. Palautetaan mieleen simuloinnin vaativuus.

Lause

Olkoon $t(n)$ funktio, jolla $t(n) \geq n$. Tällöin jokaista ajan $t(n)$ epädeterminististä yksinauhaista konetta vastaa yksinauhainen deterministinen kone, jonka aikavaatimus samoilla syötteillä on $2^{\mathcal{O}(t(n))}$.

Todistus.

Laskennan mallit -kurssi. □

Luokka P

- Kohta määrittelemäämme luokkaan P tulee kuulumaan ne probleemat, jotka voidaan algoritmisesti ratkaista polynomisessa ajassa syötteen pituuden suhteen.
- Jos tällainen polynominen ratkaisualgoritmi on olemassa, niin katsotaan, että probleema on ratkaistavissa algoritmisesti, myös käytännössä.

Luokka P

- Kohta määrittelemäämme luokkaan P tulee kuulumaan ne probleemat, jotka voidaan algoritmisesti ratkaista polynomisessa ajassa syötteen pituuden suhteen.
- Jos tällainen polynominen ratkaisualgoritmi on olemassa, niin katsotaan, että probleema on ratkaistavissa algoritmisesti, myös käytännössä.
- Sen sijaan eksponentiaalisia algoritmeja ei voi käytännössä ajaa suurilla syötteillä.
- Eksponentiaalisia algoritmeja syntyy erityisesti silloin, kun ratkaisu perustuu raa'an voiman etsintään, jolloin käydään systemaattisesti eri vaihtoehtoja.
- Laskennan vaativuusteoriassa probleemat jaetaan suuriin luokkiin, joista yhtenä tärkeänä esimerkkinä toimii juuri P.

- Huomattakoon kuitenkin, että käytännössä polynomin asteluvulla ja jopa vakioilla on merkitystä.

- Huomattakoon kuitenkin, että käytännössä polynomin asteluvulla ja jopa vakioilla on merkitystä.
- Esimerkiksi kielioppien jäsennys voitaisiin tehdä yleisellä Earlyn algoritmilla, jonka aikavaatimus on $\mathcal{O}(n^2)$. Näin ei kuitenkaan käytännössä tehdä, sillä kielioppia hieman rajoittamalla päästään lineaarisiin tai lähes lineaarisiin jäsennysalgoritmeihin ja nämä erot näkyvät käytännössä laajojen ohjelmien kohdalla.

- Huomattakoon kuitenkin, että käytännössä polynomin asteluvulla ja jopa vakioilla on merkitystä.
- Esimerkiksi kielioppien jäsennys voitaisiin tehdä yleisellä Earlyn algoritmilla, jonka aikavaatimus on $\mathcal{O}(n^2)$. Näin ei kuitenkaan käytännössä tehdä, sillä kielioppia hieman rajoittamalla päästään lineaarisiin tai lähes lineaarisiin jäsennysalgoritmeihin ja nämä erot näkyvät käytännössä laajojen ohjelmien kohdalla.
- Toinen esimerkki saadaan symbolisen matematiikan ohjelmistoista. On tavallista, että saman probleeman ratkaisuun ohjelmoidaan useampia algoritmeja. Hieman tehottomampaa, mutta pienempiä vakioita sisältävää algoritmia käytetään pienten tapausten ratkaisuun. Suuriin tapauksiin käytetään asymptoottisesti parempaa, mutta suurempia vakioita sisältävää algoritmia. Näin ohjelmisto toimii tehokkaimmalla tavalla joka tilanteessa.

- Huomattakoon kuitenkin, että käytännössä polynomin asteluvulla ja jopa vakioilla on merkitystä.
- Esimerkiksi kielioppien jäsennys voitaisiin tehdä yleisellä Earlyn algoritmilla, jonka aikavaatimus on $\mathcal{O}(n^2)$. Näin ei kuitenkaan käytännössä tehdä, sillä kielioppia hieman rajoittamalla päästään lineaarisiin tai lähes lineaarisiin jäsennysalgoritmeihin ja nämä erot näkyvät käytännössä laajojen ohjelmien kohdalla.
- Toinen esimerkki saadaan symbolisen matematiikan ohjelmistoista. On tavallista, että saman probleeman ratkaisuun ohjelmoidaan useampia algoritmeja. Hieman tehottomampaa, mutta pienempiä vakioita sisältävää algoritmia käytetään pienten tapausten ratkaisuun. Suuriin tapauksiin käytetään asymptoottisesti parempaa, mutta suurempia vakioita sisältävää algoritmia. Näin ohjelmisto toimii tehokkaimmalla tavalla joka tilanteessa.
- Huomattakoon myös, ettei käytännössä näytä esiintyvän probleemoja, joiden polynominen aikavaatimus on hyvin suuri, esim. n^{100}. Tavallisimmin eksponentti on 1, 2 ja 3, joissakin harvinaisissa tilanteissa 5 tai 6.

Määritelmä

P on niiden probleemien luokka, jotka voidaan ratkaista polynomisessa ajassa yksinauhaisella deterministisellä Turingin koneella. Toisin sanoen

$$P = \bigcup_k \mathrm{TIME}(n^k).$$

P on tärkeä, koska

- se on invariantti kaikille laskennan malleille, jotka ovat polynomisesti ekvivalentteja deterministisen yksinauhaisen Turingin koneen kanssa, ja
- se suurin piirtein vastaa tietokoneella käytännössä ratkaistavien probleemojen joukkoa.

Luokan **P** probleemoja

Seuraavien probleemojen ratkaisualgoritmit kuuluvat luokkaan **P**:

- *Ongelma*: Olkoon G suuntaamaton tai suunnattu verkko ja s, t verkon solmuja. Onko olemassa polkua s:stä t:hen?

  Ratkaisu: Syvyyssuuntainen etsintä s:stä alkaen. Selvästi lineaarinen.

Luokan **P** probleemoja

Seuraavien probleemojen ratkaisualgoritmit kuuluvat luokkaan **P**:

- *Ongelma*: Olkoon G suuntaamaton tai suunnattu verkko ja s, t verkon solmuja. Onko olemassa polkua s:stä t:hen?

  Ratkaisu: Syvyyssuuntainen etsintä s:stä alkaen. Selvästi lineaarinen.

- *Ongelma*: Annettu kontekstiton kielioppi G ja merkkijono u. Päteekö $u \in L(G)$?

  Ratkaisu: Esimerkiksi Cocke-Younger-Kasamin yleinen jäsennysalgoritmi toimii kuutiollisessa ajassa. Earlyn algoritmi toimii neliöllisessä ajassa.

Luokan **P** probleemoja

Seuraavien probleemojen ratkaisualgoritmit kuuluvat luokkaan **P**:

- *Ongelma*: Olkoon G suuntaamaton tai suunnattu verkko ja s, t verkon solmuja. Onko olemassa polkua s:stä t:hen?

  Ratkaisu: Syvyyssuuntainen etsintä s:stä alkaen. Selvästi lineaarinen.

- *Ongelma*: Annettu kontekstiton kielioppi G ja merkkijono u. Päteekö $u \in L(G)$?

  Ratkaisu: Esimerkiksi Cocke-Younger-Kasamin yleinen jäsennysalgoritmi toimii kuutiollisessa ajassa. Earlyn algoritmi toimii neliöllisessä ajassa.

- *Ongelma*: Annettu kaksi luonnollista lukua a ja b. Ovatko luvut keskenään jaottomia?

  Ratkaisu: Seuraavassa esitettävä Euklideen algoritmi, joka siis etsii kahden luvun suurimman yhteisen tekijän. Luvut ovat keskenään jaottomia, jos tekijä on 1.

Euklideen algoritmi

Input: Luonnolliset luvut a ja b.

Menetelmä:

1. while $b \neq 0$ do
2. $\quad a := a \mod b$;
3. $\quad$ vaihda a ja b;
4. end while;
5. output a;

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.
- Nimittäin jos $a/2 \geq b$, niin $a \mod b < b < a/2$, joten a vähintään puolittuu.

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.
- Nimittäin jos $a/2 \geq b$, niin $a \mod b < b < a/2$, joten a vähintään puolittuu.
- Jos taas $a/2 < b$, niin $a \mod y = a - b < a/2$ ja jälleen a vähintään puolittuu.

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.
- Nimittäin jos $a/2 \geq b$, niin $a \mod b < b < a/2$, joten a vähintään puolittuu.
- Jos taas $a/2 < b$, niin $a \mod y = a - b < a/2$ ja jälleen a vähintään puolittuu.
- a:n ja b:n arvot vaihdetaan joka kierroksella askeleessa 3, joten lukujen arvot puolittuvat ainakin joka toisella kierroksella.

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.
- Nimittäin jos $a/2 \geq b$, niin $a \mod b < b < a/2$, joten a vähintään puolittuu.
- Jos taas $a/2 < b$, niin $a \mod y = a - b < a/2$ ja jälleen a vähintään puolittuu.
- a:n ja b:n arvot vaihdetaan joka kierroksella askeleessa 3, joten lukujen arvot puolittuvat ainakin joka toisella kierroksella.
- Siten kierroksia tehdään pienemmän luvuista $2 \log_2 a$ ja $2 \log_2 b$ verran.

Euklideen algoritmin aikavaatimus

- Aluksi huomataan, että joka kerran rivillä 2 a:n arvo vähintään puolittuu.
- Nimittäin jos $a/2 \geq b$, niin $a \mod b < b < a/2$, joten a vähintään puolittuu.
- Jos taas $a/2 < b$, niin $a \mod y = a - b < a/2$ ja jälleen a vähintään puolittuu.
- a:n ja b:n arvot vaihdetaan joka kierroksella askeleessa 3, joten lukujen arvot puolittuvat ainakin joka toisella kierroksella.
- Siten kierroksia tehdään pienemmän luvuista $2 \log_2 a$ ja $2 \log_2 b$ verran.
- Huomattakoon, että lukuteoreettisissa algoritmeissa syötteen pituus on sama kuin luvun pituus, esim. binäärijärjestelmässä. Siten $\log_2 a$ on syötteen a koko, joten algoritmin aikavaatimus on lineaarinen, $\mathcal{O}(n)$, syötteen koon mukaan.

Luokka **NP**

- Tulemme nyt mielenkiintoiseen luokkaan, joka sisältää monia käytännöllisiä ongelmia, joita ei osata ratkaista polynomisessa ajassa.
- Ehkä näille ongelmille ei ole olemassakaan polynomista ratkaisua tai sitten ratkaisu perustuu toistaiseksi tuntemattomille periaatteille.
- Luokka **NP** on helppo määritellä formaalisti, mutta määrittelemme sen myös hieman havainnollisemmin, jotta luokan intuitiivinen merkitys tulee selkeämmäksi.
- Eräs keskeinen käsite tässä yhteydessä on polynominen verifioitavuus. Katsotaan tästä ensin pari esimerkkiä.

Example (Hamiltonin polku)

- Suunnatun verkon G Hamiltonin polku on verkon polku, joka kulkee jokaisen solmun kautta täsmälleen kerran.
- Määrittelemme kielen HAMPATH, joka koostuu kolmikoista $\langle G, s, t \rangle$, missä G on suunnattu verkko ja G:n solmujen s ja t välillä on Hamiltonin polku.
- Ei tunneta polynomista algoritmia, joka etsii Hamiltonin polun. Raakaan voimaan perustuva eksponentiaalinen algoritmi on sen sijaan helppo esittää.

Example (Hamiltonin polku)

- Suunnatun verkon G Hamiltonin polku on verkon polku, joka kulkee jokaisen solmun kautta täsmälleen kerran.
- Määrittelemme kielen HAMPATH, joka koostuu kolmikoista $\langle G, s, t \rangle$, missä G on suunnattu verkko ja G:n solmujen s ja t välillä on Hamiltonin polku.
- Ei tunneta polynomista algoritmia, joka etsii Hamiltonin polun. Raakaan voimaan perustuva eksponentiaalinen algoritmi on sen sijaan helppo esittää.
- Jos tarjotaan ehdokasta Hamiltonin poluksi, niin on kuitenkin helppo tarkistaa polynomisessa ajassa, onko tarjokas Hamiltonin polku vai ei. (Käydään polku kerran läpi ja katsotaan, kuljetaanko kaikkien solmujen kautta täsmälleen kerran.)
- Eli Hamiltonin polku on polynomisesti verifioitavissa.

Example (Hamiltonin polun komplementti)

- Oletetaan, että olisi menetelmä, jolla ratkaistaan, ettei kahden solmun välillä ole Hamiltonin polkua.
- Ei ole kuitenkaan yhtään yksinkertaisempaa verifioida vastausta kuin ajaa algoritmi uudestaan.

Verifioitavuuden määritelmä

Määritelmä

Verifioija kielelle A on algoritmi V, jolla

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

Polynomisen ajan verifioija on sellainen algoritmi V, joka toimii polynomisessa ajassa w:n suhteen. Kieli on *polynomisesti verifioitavissa*, jos sillä on polynominen verifioija. Symboli c on *sertifikaatti* tai *todistus*, että w kuuluu kieleen.

Example (HAMPATH ja todistus)

HAMPATH-ongelmalle $\langle G, s, t \rangle$ todistus on yksinkertaisesti Hamiltonin polku G:ssä s:stä t:hen.

Example (COMPOSITES)

Määritellään

$$\text{COMPOSITES} = \{x \mid x = pq, \;\; p, q \in \mathbf{N}, \;\; p, q > 1\}.$$

COMPOSITES-ongelmalle todistus luvulle x on yksi sen tekijä. Jakolasku voidaan tehdä polynomisessa ajassa. Huomattakoon, että luvun jakaminen tekijöihin on huomattavan vaikeaa (niin että siihen pohjautuu julkisen avaimen salaus RSA). Myös alkulukutestaus on hankalaa, joskin viime vuosina on löydetty sille polynominen algoritmi. (Aikaisemmin tunnettiin vain satunnaisalgoritmeja.)

Luokka **NP**

Määritelmä

NP *on niiden kielten luokka, joille on olemassa polynominen verifioija.*

- Luokka **NP** on tärkeä, koska se sisältää monia käytännön probleemoja, kuten HAMPATH ja COMPOSITES.
- $\mathbf{P} \subset \mathbf{NP}$.
- NP tulee sanoista "nondeterministic polynomial time". Termi johtuu luokan vaihtoehtoisesta määrittelystä, jossa käytetään epädeterministisiä Turingin koneita.
- Määritellään luokka **NP** myös tällä tavalla. Ensin esimerkki.

Example (HAMPATH-ongelman ratkaisu NTM:llä)

Ratkaisut epädeterministisellä koneella perustuvat siihen, että ensin arvataan ratkaisu tai sen osa ja sitten testataan, onko arvaus oikea.

Syöte: Kolmikko $\langle G, s, t \rangle$, missä G on suunnattu verkko ja s ja t G:n solmuja.

1. Kirjoita lista lukuja $p_1, \cdots, p_m$, missä m on G:n solmujen lkm; jokainen luku valitaan epädeterministisesti väliltä $[1, m]$.
2. Tarkista, ettei sama luku toistu listassa. Jos toistuu, hylkää syöte.
3. Tarkista, että $s = p_1$ ja $t = p_n$. Jos ei ole, hylkää syöte.
4. Jokaisella i, $1 \leq i \leq m - 1$, tarkista, että (p_1, p_{i+1}) on G:n kaari. Jos jokin pari ei ole, hylkää syöte.
5. Jos kaikki testit on viety läpi onnistuneesti, hyväksy.

Joka askel toimii polynomisessa ajassa eikä toistoja ole, joten koko algoritmi toimii polynomisessa ajassa.

Lause

Kieli on luokassa **NP**, *jos ja vain jos se voidaan tunnistaa epädeterministisellä Turingin koneella polynomisessa ajassa.*

Todistus.

(Verifioijasta Turingin kone) Näytetään, kuinka verifioija muutetaan epädeterminiseksi Turingin koneeksi. Turingin kone simuloi verifioijaa arvaamalla todisteen.
Olkoon $A \in \mathbf{NP}$ ja näytetään, että on olemassa polynomisessa jassa toimiva NTM N, joka tunnistaa A:n. Olkoon V verifioija. Voidaan olettaa, että V on deterministinen Turingin kone, joka toimii ajassa n^k syötteen pituuden n suhteen.

Syöte: w, jonka pituus n.

1. Valitse epädeterministisesti merkkijono c, jonka pituus on korkeintaan n^k.
2. Aja V syötteellä $\langle w, c \rangle$.
3. Jos V hyväksyy, hyväksy, muuten hylkää.

Todistus.

(Turingin koneesta verifioija) Todistetaan nyt toiseen suuntaan. Eli oletetaan, että kielen A tunnistaa epädeterministinen Turingin kone N polynomisessa ajassa, ja konstruoidaan polynomiaalisen ajan verifioija.

Syöte: Merkkijonopari $\langle w, c \rangle$.

Algoritmi V:

1. Simuloi N:ää syötteellä w siten, että c tulkitaan epädeterministisissä haarautumakohdissa ohjeeksi, mikä haara seuraavaksi valitaan. (Jos c on epäkelpo ohje, syöte hylätään.)
2. Jos tämä N:n näillä valinnoilla valittu haara hyväksyy, *hyväksy*, muuten *hylkää*.

Probabilistinen Turingin kone I

- Epädeterministisen Turingin koneen hyväksymä kieli voidaan määritellä suoraviivaisesti myös probabilistisen Turingin koneen avulla.
- Probabilistisen koneen siirtymäfunktio kuvaa siirtymän todennäköisyyttä:

$$\delta : \ Q \times \Sigma \times Q \times \Sigma \times \{L, R\} \longrightarrow [0, 1].$$

- Tulkinta: $\delta(q_1, a_1, q_2, a_2, d)$ tarkoittaa todennäköisyyttä, että kone tilassa q_1 lukiessaan symbolin a_1 päätyy tilaan q_2 ja kirjoittaa nauhalle a_2:n sekä siirtyy d:n suuntaan.
- Vaatimuksena on, että

$$\sum_{(q_2, a_2, d) \in Q \times \Sigma \times \{L, R\}} \delta(q_1, a_1, q_2, a_2, d) = 1.$$

Probabilistinen Turingin kone II

- Lisäksi vaaditaan, että siirtymäfunktion todennäköisyysarvot ovat rationaalilukuja, jotta vältyttäisiin laskettavuuteen liittyviltä periaatteellisilta ongelmilta.
- Olkoon probabilistisen koneen aikavaatimus syötteellä c pisimmän haaran ($tn > 0$) aikavaatimus.
- Nyt luokka NP voidaan myös määritellä niiden kielten luokaksi, jotka probabilistinen kone hyväksyy polynomisessa ajassa nollaa suuremmalla todennäköisyydellä.

Luokka NTIME

Voimme nyt esittää luokan **NP** traditionaalisen määritelmän, joka on formaalimpi kuin aikaisemmin esittämälle, verifioijiin perustuva määritelmä.

Määritelmä

NTIME$(t(n))$ *on niiden kielten joukko, jotka voidaan tunnistaa epädeterministisellä Turingin koneella ajassa* $\mathcal{O}(t(n))$.

Saadaan välittömästi

Corollary

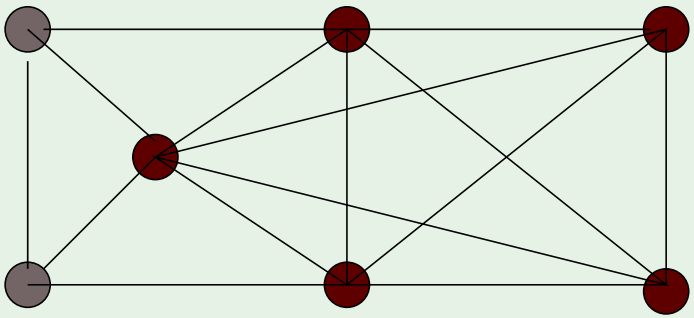
$$NP = \cup_k \mathbf{NTIME}(n^k).$$

Example

Suuntaamattoman verkon G klikki on aliverkko, jonka minkä tahansa kahden solmun välillä on särmä. Jos klikki sisältää k solmua, puhutaan k-klikistä. Klikkiprobleemana on määrätä, sisältääkö verkko tietynsuuruisen klikin. Formaalisti siis kysymyksessä on kieli

$$\mathrm{CLIQUE} = \{\langle G, k\rangle \mid G \text{ on suuntaamaton verkko, jossa } \mathrm{k}-\text{klikki}\}.$$

Esimerkiksi seuraavan verkon punaisella merkityt solmut muodostavat 5-klikin.



Lause

CLIQUE kuuluu **NP***:hen.*

Todistus.

Konstruoidaan verifioija klikkiongelmalle. Seuraavassa syötteenä on $\langle\langle G, k\rangle, c\rangle$.

1. Testaa, onko c G:n k:n solmun joukko.
2. Testaa, sisältääkö G särmän jokaisen c:n solmuparin välillä.
3. Jos molemmat ehdot voimassa, *hyväksy*, muuten *hylkää*.

□

Todistus.

Vaihtoehtoinen todistus, perustuen epädeterministiseen Turingin koneeseen.
Syötteenä on nyt verkko G ja luku k.

1. Valitse epädeterministisesti k:n solmun joukko G:stä.
2. Testaa, muodostavatko nämä k solmua klikin.
3. Jos muodostavat, *hyväksy*, muuten *hylkää*.

□

Example (SUBSET-SUM)

On annettu lukujoukko $X = \{x_1, \cdots, x_k\}$ ja erillinen luku t (kohdeluku). Ongelmana on päättää, löytyykö lukuja $y_1, \cdots, y_n \in X$, joiden summa on t. Formaalisti voidaan määritellä, että

$$\mathrm{SUBSET-SUM} = \{\langle S, t\rangle\},$$

missä siis $S = \{x_1, \cdots, x_k\}$ ja jollakin osajoukolla $\{y_1, \cdots, y_n\} \subset S$ pätee $\sum_{i=1}^{n} y_i = t$. Esimerkiksi $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in \mathrm{SUBSET-SUM}$, sillä $4 + 21 = 25$.

Huomattakoon, että S ja $\{y_1, \cdots, y_n\}$ voivat olla monijoukkoja eli niissä voi olla useampia samoja alkioita.

Lause

SUBSET-SUM kuuluu NP:hen.

Todistus.

Todistuksen idea perustuu siihen, että osajoukko on sertifikaatti eli todistus. Konstruoidaan verifioija V seuraavasti. Syötteenä on pari $\langle\langle S, t\rangle, c\rangle$.

1. Testaa, onko c:n lukujen summa t.
2. Testaa, kuuluuko c S:ään.
3. Jos molemmat testit ok, *hyväksy*, muuten *hylkää*.

□

Todistus.

Vaihtoehtoinen todistus, jossa konstruoidaan epädeterminististinen Turingin kone. Syötteenä nyt $\langle S, t \rangle$.

1. Valitse epädeterminininistisesti S:n osajoukko c.
2. Testaa, onko c:n lukujen summa t.
3. Jos testi ok, *hyväksy*, muuten *hylkää*.

- Huomattakoon, että näiden probleemajoukkojen komplementit $\overline{CLIQUE}$ ja $\overline{SUBSET - SUM}$ eivät ilmeisesti ole NP:n alkioita.
- Otetaan käyttöön merkintä coNP, jolla siis tarkoitetaan NP:n probleemojen komplementtiprobleemoita.
- Ei tiedetä, ovatko NP ja coNP erillisiä.