Luokat L ja NL

Määritelmä

L on niiden kielten luokka, jotka voidaan tunnistaa logaritmisessa tilassa deterministisellä Turingin koneella. Eli

$$L = \mathrm{SPACE}(\log n).$$

NL on niiden kielten luokka, jotka voidaan tunnistaa logaritmisessa tilassa epädeterministisellä Turingin koneella. Eli

$$NL = \mathrm{NSPACE}(\log n).$$

Luokat N ja NL II I

- Siis tutkitaan probleemoja, jotka voidaan ratkaista sublineaarisessa tilassa.
- Tilalla tarkoitetaan nyt työtilaa. Sitä varten tarvitaan hieman modifioitua Turingin koneen mallia, jossa on kaksi nauhaa.
- Toinen nauha on read only -nauha, jolla sijaitsee syöte. Toinen on työnauha.
- Tilankäytössä otetaan huomioon vain työnauhan käyttö, jolloin on siis mahdollista puhua sublineaarisesta tilasta, vaikka koko syöte luetaankin.
- Sen sijaan ei ole mielekästä tarkastella sublineaarista aikaa. Tällöinhän ei edes koko syötettä voisi lukea.

Example

Kieli $A = \{0^k 1^k \mid k \geq 0\}$ kuuluu L:ään.

- Aikaisemmin esitelty kone tunnisti kielen selaamalla edestakaisin syötettä ja merkkaamalla nollia ja ykkösiä. Se kulutti lineaarisen tilan.
- Jos sen sijaan otetaan käyttöön kaksi laskuria, joiden arvoa päivitetään työnauhalla, päästää log-tilaan.
- Toinen laskuri laskee nollien, toinen ykkösten lukumäärää. Laskurien arvo kirjataan binäärimuodossa.
- Näin päästään logaritmiseen tilaan.

Example (Polkuprobleema suunnatussa verkossa)

- Kieli PATH koostui alkioista $< G, s, t >$, missä G on suunnattu verkko, jossa on polku solmusta s solmuun t.
- PATH oli P:n alkio ja annettu algoritmi (syvyyssuuntainen etsintä) toimii lineearisessa tilassa.
- Ei tiedetä, voidaanko PATH tunnistaa deterministisesti logaritmisessa tilassa, mutta epädeterministisesti se voidaan seuraavasti.
- Kone arvaa epädeterministisesti seuraavan solmun matkalla s:stä t:hen.
- Vain polun viimeisin solmu pidetään muistissa.
- Tällä tavoin jatketaan, kunnes t kohdataan ja syöte hyväksytään tai on käyty läpi m solmua, missä m on verkon solmujen lukumäärä.

Konfiguraatio

Määritelmä

Jos M on Turingin kone, jolla on erillinen read-only syötenauha, ja w on sen syöte, niin M:n konfiguraatio tai tilanne syötteellä w koostuu koneen tilasta, työnauhan sisällöstä ja molempien lukupäiden asemasta. Syöte w ei ole osa M:n tilannetta.

Konfiguraatioiden lukumäärä

- Jos M vaatii $f(n)$-tilan ja jos w on n:n mittainen syöte, niin M:n konfiguratioiden lukumäärä w:llä on $n2^{\mathcal{O}(f(n))}$.
- Perustelu: Olkoon c M:n tilojen lkm ja g nauhasymbolien lkm.
- Työnauhalla voi esiintyä $g^{f(n)}$ merkkijonoa.
- Syötenauhan lukupää voi olla jossain n:stä paikasta ja työnauhan lukupää yhdessä $f(n)$:stä paikasta.
- Siten M:n konfiguraatioiden lkm on korkeintaan $cnf(n)g^{f(n)}$ eli $n2^{\mathcal{O}(f(n))}$.

- Rajoitumme tilavaativuuksiin f, missä $f(n) \geq \log n$.
- Aikaisempi väittämämme, että aikavaativuus on korkeintaan eksponentiaalinen tilavaativuuteen verrattuna, pätee edelleenkin asetetun kaltaisille tilafunktioille, koska $n2^{\mathcal{O}(f(n))}$ on $2^{\mathcal{O}(f(n))}$, jos $f(n) \geq \log n$.
- Savitchin lause sanoi, että epädeterminististä konetta voidaan simuloida deterministisellä koneella ja tällöin tilavaatimus kasvaa $f(n)$:stä vain neliöllisesti $f^2(n)$:een olettaen, että $f(n) \geq n$.
- Tulos pätee myös, jos sallitaan väljempi raja $f(n) \geq \log n$.
- Todistus sujuu samalla tavalla kuin normaalin Turingin koneen yhteydessä.

NL-täydellisyys

- Samoin kuin probleema $P \stackrel{?}{=} NP$ myös probleema $L \stackrel{?}{=} NL$ on ratkaisematta.
- Etsimällä NL-täydellisiä probleemoja saadaan parempi käsitys edellä esitetystä probleemasta.
- Nyt täydellisyyden yhteydessä ei kuitenkaan voida käyttää polynomista palautusta ajan suhteen, koska se on liian voimakas.
- Nimittäin jokainen NL:n ongelma on ratkeava polynomisessa ajassa, joten mitkä tahansa kaksi NL:n ongelmaa voidaan tässä mielessä palauttaa toisiinsa.
- Sen sijaan käytetään log-tila -palautusta.

Log-tila -palautus

Määritelmä

- *Log-tila -muunnin (transducer) on read-only -syötenauhalla, write-only -tulostusnauhalla ja työnauhalla varustettu Turingin kone.*
- *Työnauha voi sisältää $\mathcal{O}(\log n)$ symbolia.*
- *Log-tila -muunnin M laskee funktion $f : \Sigma^* \longrightarrow \Sigma^*$, missä $f(w)$ on tulostusnauhalle jäävä merkkijono, kun M pysähtyy sen jälkeen, kun laskenta on aloitettu syötteestä w syötenauhalla.*
- *f on log-tilassa laskettava funktio.*
- *Kieli A voidaan palauttaa log-tilassa kieleen B, $A \leq_L B$, jos on olemassa log-tilassa laskettavissa oleva funktio f, jolla $x \in A$ jos ja vain jos $f(x) \in B$.*

NL-täydellisyys

Määritelmä

Kieli B on NL-täydellinen, jos

1. $B \in NL$,
2. *Jokainen NL:n kieli A voidaan palauttaa log-tilassa B:hen.*

Lause

Jos $A \leq_L B$ ja $B \in L$, niin $A \in L$.

Todistus.

- Nyt emme voi todistaa tätä tulosta samalla tavalla kuin ajan tapauksessa. Jos nimittäin lasketaan ensin $f(w)$ syötteestä w ja sen jälkeen lasketaan B:n algoritmilla vastaus $f(w)$:sta, niin $f(w)$:n pituus voi olla suurempi kuin $\log |w|$, joten A:lle asetettu tilavaatimus ylittyy.
- Sen sijaan tehdään niin, että A:n kone M_A laskee $f(w)$:n symboleja simuloiden B:n koneen M_B toimintaa.
- Simuloinnissa M_A pitää kirjaa siitä, minkä $f(w)$:n merkin kohdalla M_B:n syötenauhan lukupää kulloinkin on.
- Joka kerran kun M_B liikkuu, M_A aloittaa $f(w)$:n laskennan alusta ja unohtaa muut $f(w)$:n merkit kuin kyseisessä paikassa olevan merkin.
- Näin tehden joudutaan ehkä laskemaan $f(w)$:n osia useaan kertaan, joten laskenta on hidasta ajallisesti.
- Etuna on kuitenkin, että vain yksi $f(w)$:n symboli täytyy pitää muistissa kullakin hetkellä, joten tilaa säästyy ajan kustannuksella. □

Lause

PATH-ongelma on NL-täydellinen.

Todistus.

- Näytetään, miten jokainen NL:n probleema voidaan palauttaa log-tilassa PATH-probleemaan.
- Olkoon M epädeterministinen kone, joka tunnistaa A:n $\mathcal{O}(\log n)$-tilassa.
- Kun on annettu syöte w, konstruoidaan tapaus $< G, s, t >$ log-tilassa, missä G siis on suunnattu verkko, joka sisältää polun s:stä t:hen jos ja vain jos M hyväksyy w:n.
- G:n solmut ovat M:n konfiguraatioita w:n laskennassa.
- Jos c_1 ja c_2 ovat kaksi konfiguraatiota, niin (c_1,c_2) on kaari, jos M voi siirtyä c_1:stä c_2:een yhdellä askeleella.

- Solmu s on M:n aloituskonfiguraatio. M:ää modifioidaan siten, että sillä on yksikäsitteinen hyväksyvä loppukonfiguraatio, joka edustaa sitten solmua t.
- Tämä kuvaus palauttaa A:n PATH-ongelmaan, sillä aina kun M hyväksyy syötteen, jokin haara päätyy hyväksyvään tilaan, mikä tarkoittaa, että löytyy polku s:stä t:hen.
- Kääntäen jos on olemassa jokin polku s:stä t:hen, niin jokin haara hyväksyy syötteen w.
- Palautus toimii log-tilassa. Jotta tämä nähtäisiin, täytyy konstruoida muunnin, joka laskee syötteestä w verkon G.
- G:n kuvailemiseen täytyy antaa solmulista ja kaarten lista.
- Solmujen listaus on helppoa, koska jokainen solmu vastaa jotain M:n konfiguraatiota ja solmu voidaan esittää $c \log n$-tilassa, missä c on jokin vakio.
- Muunnin käy läpi sarjallisesti kaikki mahdolliset $c \log n$:n pituiset merkkijonot, testaa, että jokainen on M:n laillinen konfiguraatio w:n laskennassa, ja tulostaa ne jonot, jotka läpäisevät testin.

- Muunnin listaa kaaret samalla periaatteella.
- Log-tila on riittävä sen verifioimiseen, että konfiguraatiosta c_1 päästään konfiguraatioon c_2, koska muuntimen tarvitsee vain tutkia nauhojen senhetkinen sisältö nauhapäiden kohdalta konfiguraatiossa c. Tämän perusteella voidaan päättää, pääseekö M c_2:een c_1:stä.
- Muunnin tutkii kaikki parit (c_1, c_2) peräjälkeen selvittääkseen, mitkä ovat G:n kaaret. Nuo parit, jotka läpäisevät testin, tulostetaan tulostusnauhalle. $\square$

Lause

$NL \subset P$.

Todistus.

- Edellinen lause osooitti, että jokainen NL:n kieli on palautettavissa log-tilassa PATH-ongelmaan.
- Muistetaan, että Turingin kone, joka käyttää $f(n)$-tilaa toimii $n2^{\mathcal{O}(f(n))}$-ajassa, joten muunnin, joka toimii log-tilassa toimii polynomisessa ajassa.
- Siten jokainen NL:n kieli on palautettavissa polynomisessa ajassa PATH-ongelmaan, joka on P:ssä.
- Jos kieli on palautettavissa polynomisessa ajassa johonkin P:n kieleen, niin se on itsekin on P:ssä. □

CoNL I

Lause

NL = coNL.

Todistuksen idea

- Tulos on yllättävä.
- Osoitetaan, että $\overline{\text{PATH}}$ on NL:ssä, mikä osoittaa, että jokainen coNL:n kieli on myös NL:ssä, koska PATH on NL-täydellinen.
- NL:n algoritmi M päätyy hyväksyvään tilaan täsmälleen silloin, kun verkossa G ei ole polkua s:stä t:hen.
- Tarkastellaan ensin helpompaa tehtävää. Olkoon c niiden solmujen lkm, jotka voidaan saavuttaa s:stä. Oletetaan, että c on annettu, ja käytetään tätä hyväksi ongelman $\overline{\text{PATH}}$ ratkaisussa. Myöhemmin näytetään, miten c määrätään.
- On siis annettu G, s, t ja c ja laaditaan kone M, joka tunnistaa $\overline{\text{PATH}}$:n.

CoNL II

- M valitsee epädeterministisesti c solmua – t ei ole mukana – ja verifioi, että solmut ovat saavutettavissa s:stä. Jos näin on, M tietää, ettei t ole saavutettavissa ja hyväksyy syötteen.
- Jotta tilaa säästyy, arvaus etenee solmu solmulta. Siis ensin otetaan solmu ja arvataan, onko solmu saavutettavissa vai ei. Sen jälkeen verifioidaan ja siirrytään seuraavaan solmuun. Jos t:n arvataan olevan saavutettavissa, syöte hylätään heti.
- Seuraavaksi täytyy näyttää, miten c määrätään. Laaditaan epädeterministinen log-tilainen algoritmi, jonka vähintään yksi haara laskee c:n oikein.
- Kaikilla arvoilla $i = 0, ..., m$ olkoon A_i niiden solmujen joukko, jotka ovat korkeintaan etäisyydellä i s:stä. Siten $A_0 = \{i\}$, $A_i \subset A_{i+1}$ ja A_m käsittää kaikki s:stä saavutettavat solmut.
- Olkoon c_i A_i:n koko. Näytetään, miten c_{i+1} saadaan laskettua c_i:stä. Toistamalla tätä saadaan laskettua c_m.

CoNL III

- Käytetään samaa ideaa kuin äsken. Algoritmi käy läpi kaikki G:n solmut ja määrää, onko solmu A_{i+1}:n alkio vai ei, sekä pitää kirjaa hyväksytyistä alkioista.
- Tarkemmin: Kun ratkaistaan, onko solmu v A_{i+1}:ssä vai ei, käydään silmukassa kaikki solmut läpi ja arvataan, onko solmu A_i:ssä vai ei. Jokainen positiivinen arvaus verifioidaan arvaamalla korkeintaan i:n pituinen polku ja testaamalla, että se on todella polku solmusta s kyseiseen solmuun.
- Jokaista verifioitua solmua u kohti testataan tämän jälkeen, onko (u, v) kaari G:ssä. Jos se on kaari, v on A_{i+1}:ssä.
- Lisäksi verifioitujen solmujen lukumäärä lasketaan. Jos lopussa verifioitujen solmujen lkm ei ole c_i, laskenta tässä haarassa hylätään.
- Jos lkm on c_i, eikä v:tä ole vielä saatu A_{i+1}:hen, niin v ei kuulukaan sinne. Siirrytään seuraavaan v:hen.
- Seuraavassa vielä algoritmin toiminta algoritmimuodossa:

Algoritmi $\overline{\text{Path}}$-ongelmalle I

1. $c_0 := 1$;
2. **for** $i = 0$ **to** $m - 1$ **do**
3. $\quad c_{i+1} = 1$;
4. $\quad$ **for** each node $v \neq s$ G:ssä **do**
5. $\quad\quad$ d:=0;
6. $\quad\quad$ for each node u in G **do**
7. $\quad\quad\quad$ epädeterministisesti joko tee tai sivuuta seuraavat askeleet
8. $\quad\quad\quad\quad$ epädeterministisesti seuraa korkeintaan i:n mittaista polkua s:stä ja hylkää, jos se ei pääty u:hun;
9. $\quad\quad\quad\quad$ $d := d + 1$;
10. $\quad\quad\quad\quad$ **if** (u, v) kaari, $c_{i+1} := c_{i+1} + 1$ ja

Algoritmi $\overline{\text{Path}}$-ongelmalle II

go to step 5 seuraavan v:n kanssa;

11. **if** $d \neq c_i$ **then** *hylkää*;
12. $d := 0$;
13. **for** jokaiselle solmulle u **do**
14. epädeterministisesti joko suorita tai
    sivuuta seuraavat askeleet
15. seuraa epädeterministisesti korkeintaan
    m:n mittaista polkua s:stä ja *hylkää*,
    jos se ei pääty u:hun;
16. **if** $u = t$ **then** *hylkää*;
17. $d := d + 1$;
18. **if** $d \neq c_m$ **then** *hylkää*

else *hyväksy*.

- Algoritmin tarvitsee tallettaa vain u, v, c_i, c_{i+1}, d, i ja osoitin polun päähän.
- Siten se toimii logaritmisessa tilassa. $\square$

Tiedetään siis seuraavat luokkien suhteet:

$$\mathrm{L} \subset \mathrm{NL} = \mathrm{coNL} \subset \mathrm{P} \subset \mathrm{PSPACE}.$$

Aitoudesta tiedetään vain

$$\mathrm{NL} \subsetneq \mathrm{PSPACE}.$$

Useimmat olettavat, että kaikki kuuluvuudet ovat aitoja.