

## 58131 Tietorakenteet (kevät 2007)

### 2. kurssikoe 7.5., ratkaisuja

1. Lisäysjärjestäminen on pienillä aineistoilla ( $n$  20–100 alkioita) paras mainituista algoritmeista (ja yleensäkin).

Kekojärjestäminen toimii ajassa  $O(n \log n)$  (toisin kuin lisäysjärjestäminen), myös pahimmassa tapauksessa (toisin kuin pikajärjestäminen), ja vakiotyötilassa (toisin kuin pikajärjestäminen ja lomitusjärjestäminen). Sen on suhteellisen helppo toteuttaa (luultavasti helpoin ajassa  $O(n \log n)$  toimivista algoritmeista).

Lomitusjärjestämistä voidaan käyttää myös tilanteissa, joissa järjestetään levytiedostoja tai linkitettyjä listoja; algoritmi ei tarvitse suoraa indeksointia taulukkoon.

Pikajärjestäminen on paras yleiskäyttöinen järjestämisalgoritmi suurille aineistoille.

**Pisteytys:** Kustakin algoritmista sai puoli pistettä jonkin hyvän puolen mainitsemisesta. oinen puoli pistettä sai, jos mainittu ominaisuus todella erotti algoritmin muista mainituista. (Puolikaspisteet on lopputuloksissa pyöristetty ylöspäin.)

2. Todetaan ensin, että viisi miljoonaa alkioita on niin paljon, että algoritmien aikavaativuuksien suuruusluokka-arviot antavat hyvän kuvan niiden todellisista suoritusajoista toisiinsa verrattuna.

(a) Helpoin ratkaisu on varata taulukko  $A[0..10000]$ , jonne merkataan, mitä alkioita on jo tulostettu. Käydään alkioit läpi järjestyksessä  $x_1, x_2, \dots$  ja alkion  $x_i$  kohdalla jos  $A[x_i] = 0$ , niin tulostetaan  $x_i$  ja asetetaan  $A[x_i] \leftarrow 1$ . Jos valmiiksi  $A[x_i] = 1$ , niin  $x_i$  on jo tulostettu, joten mitään ei tarvitse tehdä. Algoritmi toimii lineaarisessa ajassa ja on muutenkin tehokas.

(b) Nyt lukualue on niin iso, että yllä kuvattu taulukko  $A$  ei mahtuisi muistiin. Luvut  $x_i$  sen sijaan mahtuvat muistiin hyvin. (Suurin mahdollinen arvo on  $10^{12} \approx 2^{40}$ , joten yksi luku mahtuu hyvin kuuteen tavuun). Ongelma voidaan ratkaista ajassa  $O(n \log n)$  sijoittamalla luvut  $x_i$  taulukkoon, järjestämällä taulukko ja tulostamalla taulukko siten, että samaa lukua sisältävistä jaksoista tulostetaan vain ensimmäinen luku. Järjestämisalgoritmiksi voidaan valita esim. kekojärjestäminen.

Lisäoletuksia tekemättä ongelmaa ei itse asiassa voi ratkaista nopeammin kuin ajassa  $O(n \log n)$ , joten tämä on jossain mielessä optimaalista.

(c) Koska luvut ovat tasaisesti jakautuneita ja halutaan hyvä keskimääräinen aikavaativuus, käytetään hajautusta. Kun kohdataan luku  $x_i$ , etsitään sitä ensin hajautustaulusta. Jos se löytyi, ei tehdä mitään. Jos sitä ei löytynyt, tulostetaan se ja talletetaan hajautustauluun.

Minkä tahansa normaalin hajautusmenetelmän pitäisi annetuissa olosuhteissa toteuttaa haku ja lisäys keskimäärin vakioajassa. Esim. jakolaskumenetelmä, ketjuttaminen ja talletusalueen koko  $n \cdot 5 \cdot 10^6$  (eli  $\alpha \approx 1$ ) pitäisi kelvata. Tällöin koko ongelman aikavaativuus on siis keskimäärin lineaarinen.

Huomaa, että vaikka lukuja on vain  $n \cdot 5 \cdot 10^6$  ja ne ovat tasaisesti jakautuneet paljon suurempaan joukkoon  $\{0, \dots, 10^{12}\}$ , duplikaattien esiintyminen on todennäköistä (syntymäpäiväparadoksi).

**Pisteytys:** Jokainen alakohta pisteytettiin erikseen seuraavasti:

- Jokin järkevä idea siitä, miten ongelman voisi ratkaista: 1/2 p
- Algoritmi on kuvattu toimivalla ja riittävän tarkalla tavalla: +1/2 p
- Algoritmi toimii tehokkaasti (a:  $O(n)$ , b:  $O(n \log n)$ , c:  $O(n)$ ): +1/2 p
- Perustelut: 1/2 p

Pisteet laskettiin yhteen ja pyöristettiin alaspäin.

3. Mallinnetaan ongelma suunnattuna painotettuna verkkona, jonka solmuina ovat tietokoneet  $a_i$  ja kaarina ne parit  $(a_i, a_j)$ , joiden välillä on yhteys. Kaaren  $(a_i, a_j)$  paino on  $w(a_i, a_j) = p(i, j)$ . Lisäksi annetaan paino myös solmuille:  $w'(a_i) = q(i)$ . Ongelmana on nyt löytää lyhimmat solmusta  $a_1$  alkavat polut, kun polun pituudeksi määritellään sillä olevien solmujen ja kaarten painojen summa.

Jos solmupainoja ei olisi, ongelma ratkeaisi suoraan Dijkstran algoritmilla. Olkoon  $d[v]$  lyhimmän polun pituus solmusta  $a_1$  solmuun  $v$ . Jos  $(a_1, v_1, v_2, \dots, v_k)$  on lyhin polku  $a_1 \rightsquigarrow v_k$  ja solmupainoja ei ole, niin

$$d[v_k] = d[v_{k-1}] + w(v_{k-1}, v_k).$$

Dijkstran algoritmi perustuu tähän yhtälöön. Kun otetaan huomioon solmupainot, saadaan

$$d[v_k] = d[v_{k-1}] + w(v_{k-1}, v_k) + w'(v_k).$$

Oletetaan, että syötteenä on annettu ensin yhtenä listana kaikki solmupainot ja sitten listana kaikki olemassaolevat yhteydet  $(a_i, a_j)$  painoineen  $p(i, j)$ . Algoritmista tulee seuraava:

#### DIJKSTRA-WITH-VERTEX-WEIGHTS

```

Lue viipeet  $q(i)$  ja talleta taulukkoon  $w'$ .
Lue yhteydet  $(a_i, a_j)$  ja viipeet  $p(i, j)$  ja muodosta
niistä vieruslistaesitys painotetulle suunnatulle verkolle  $G = (V, E, w)$ .
 $Q \leftarrow$  tyhjä prioriteettijono
for  $v \in V - \{a_1\}$ 
    do  $d[v] \leftarrow \infty$ 
        INSERT( $Q, v, d[v]$ )
 $d[a_1] \leftarrow w'[a_1]$ 
INSERT( $Q, a_1, d[a_1]$ )
while  $Q \neq \emptyset$ 
    do  $u \leftarrow$  DELETE-MIN( $Q$ )
        for  $v \in Adj[u]$ 
            do if  $d[v] > d[u] + w(u, v) + w'(v)$ 
                then  $d[v] \leftarrow d[u] + w(u, v) + w'(v)$ 
                     $p[v] \leftarrow u$ 
                    DECREASE-KEY( $Q, v, d[v]$ )

```

Algoritmin toimitaperiaate on sama kuin Dijkstran algoritmilla. Se pitää yllä invarianttia, että jos  $u$  kuuluu joukkoon  $S = V - Q$ , niin  $d[u]$  on lyhimmän polun pituus  $a_1 \rightsquigarrow u$ . Lisäksi  $p[u]$  on toiseksi viimeinen solmu tällä polulla. Joukkoa  $S$  laajennetaan ahneesti valitsemalla aina lähin sen ulkopuolella oleva solmu.

Kuten Dijkstran algoritmin perusversiossa, tässäkin suoritetaan  $O(|V| + |E|)$  prioriteettijono-operaatiota, mikä selvästi dominoi aikavaativuutta. Toteuttamalla prioriteettijono keon avulla yksittäiset operaatiot saadaan suoritetuksi ajassa  $O(\log|V|)$ , joten aikavaativuus on  $O((|V| + |E|) \log|V|)$ . Jos verkko on yhtenäinen, pätee  $|V| = O(|E|)$  ja aikavaativuus yksinkertaistuu muotoon  $O(|E| \log|V|)$ .

Kun ylläoleva algoritmi on suoritettu, lyhin polku  $a_1 \rightsquigarrow u$  saadaan kutsulla  $PATH(u)$ , missä

```

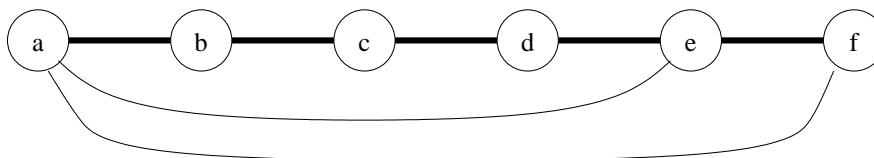
PATH( $u$ )
if  $u = a_1$ 
    then print  $u$ 
elseif  $d[u] = \infty$ 
    then print ”ei polkua”
else PATH( $p[u]$ )
    print  $u$ 

```

**Pisteitys:** Yhden pisteen sai, jos oli mallintanut ongelman verkkojen lyhimmän polun etsintänä. Toisen pisteen sai mainitsemalla Dijkstran algoritmin (myös Bellman-Ford hyväksyttiin) ja kolmannen, kun osasi selvittää algoritmin perustoimintaperiaatteen. Pseudokoodista sai 1–2 pistettä ja eksplisiittisesti selvitetystä aikavaativuudesta viimeisen pisteen.

4. (a) Ehdotettu ratkaisumenetelmä on virheellinen.

Tarkastellaan esim. allaolevaa verkkoa, jossa on suoritettu syvyysuuntainen läpikäynti solmusta  $a$  alkaen ja valitut puukaaret on paksunnettu:



Nyt verkon lyhin sykli on  $(a, e, f, a)$ , mutta ehdotettu algoritmi ei löydä sitä.

- (b) Ehdotettu ratkaisumenetelmä toimii oikein.

Olkoon  $(V, T)$  jokin pienin virittävä puu ja  $p$  maksimipainoltaan pienin polku  $s \rightsquigarrow t$  puun kaaria pitkin. Tehdään vasta oletus, että on olemassa maksimipainoltaan pienempi polku  $p'$  solmusta  $s$  solmuun  $t$ . Valitaan polun  $p$  painavin kaari; olkoon se  $e$ . Tarkastellaan verkkoa  $(V, T - \{e\})$ . Verkossa on kaksi komponenttia, joista toinen sisältää solmun  $s$  ja toinen solmun  $t$ . Polku  $p'$  sisältää ainakin yhden kaaren  $e'$ , joka yhdistää näitä komponentteja. Siis  $(V, T \cup \{e'\} - \{e\})$  on virittävä puu. Oletuksen mukaan kaikki polun  $p'$  kaaret ovat kevyempiä kuin  $e$ . Erityisesti tämä koskee kaarta  $e'$ , joten

$$w(T \cup \{e'\} - \{e\}) = w(T) + w(e') - w(e) < w(T).$$

Siis verkolla on pienempi virittävä puu kuin  $(V, T)$ ; ristiriita.

**Pisteitys:** Maksimipistemäärä kohdasta (a) oli 3 pistettä. Yhden pisteen sai, jos ehdotettu menetelmä oli todettu toimimattomaksi, mutta perustelut olivat pahasti puutteelliset tai virheelliset, tai jos menetelmää luultiin toimivaksi, mutta perusteluosuudessa oli sinänsä oikeita havaintoja syklien etsimisestä.

Maksimipistemäärä kohdasta (b) oli 5 pistettä. Yhden pisteen sai, jos ehdotettu menetelmä oli todettu toimivaksi tai jos menetelmää luultiin toimimattomaksi, mutta perusteluissa oli jotain oikeita havaintoja pienimmästä virittävästä puusta. Kolme pistettä sai toteamalla menetelmän toimivaksi ja esittämällä väitteen tueksi hyviä huomioita virittävästä puusta. Täydet viisi pistettä edellyttivät, että perustelut oli muotoiltu kunnolliseksi loogiseksi argumentiksi.