

58053-7 Algoritmien suunnittelu ja analyysi (kevät 2004)

1. välikoe, ratkaisuja

Malliratkaisut ja pisteytysohje: Jyrki Kivinen

Tentin arvostelu: Jouni Siren (tehtävät 1 ja 2) ja Jyrki Kivinen (tehtävät 3 ja 4)

1. POP-UNTIL-operaatio toimii siis seuraavasti:

```
POP-UNTIL( $x$ ):  
  repeat  
    if EMPTY then  $y := x$   
    else  $y := \text{POP}$   
  until  $y = x$ 
```

Operaatioiden todelliset kustannukset ovat kertaluokan tarkkuudella

PUSH	1
POP	1
POP-UNTIL(x)	$1 + \min \{ k, \text{pinon koko} \}$

missä k on alkion x ensimmäisen esiintymän syvyys pinossa. (Sovitaan $k = \text{pinon koko} + 1$ jos alkioita ei löydy.) Nämä ovat oleellisesti samat kuin MULTIPOP-operaatiolla varustetussa pinossa (luennot s. 106), joten analyysikin sujuu samalla tavalla: joko kirjanpitomenetelmällä (s. 108) tai potentiaalimenetelmällä (s. 111).

Pisteytys:

1 piste toteutuksesta

3 pistettä tasoitetun analyysin peruskäsitteistön (kirjanpito tai potentiaali) käyttämisestä

2 pistettä analyysin loppuunviemisestä

Huomautuksia: Tehtävän tarkoitus oli testata tasoitetun analyysin ymmärtämistä, joten pelkästä toteutuksesta ei ole saanut enempää pisteitä. Yleensä tehtävä oli joko osattu (5–6 p.) tai sitten ei (0–1 p.)

2. Jos A on valmiiksi nousevassa järjestyksessä, PARTITION palauttaa $k = i$ ja rekursiivisissa kutsuissa osataulukkojen koot ovat $k - 1 - i + 1 = 0$ ja $j - (k + 1) + 1 = j - i$. Jos siis $T(k)$ on kutsun QUICKSORT(A, i, j) aikavaativuus kun $i - j + 1 = k$, saadaan (kun $n \geq 1$)

$$\begin{aligned} T(n) &\geq T(n-1) + cn \\ &= c \sum_{k=1}^n k + T(0) \\ &= c \frac{n(n+1)}{2} + T(0) \\ &= \Theta(n^2). \end{aligned}$$

Tässä c on PARTITION-kutsun vakiokerroin.

Pahimman tapauksen aikavaativuuden parantamiseksi voidaan valita

$$\text{pivot} := \text{SELECT}(A, \lceil n/2 \rceil),$$

missä $n = j - i + 1$ ja SELECT on luennoilla esitetty mediaaninetsimisalgoritmi, ja asettaa rekursiivisissa kutsuissa $k = \lceil n/2 \rceil$. Koska SELECT toimii lineaarisessa ajassa, saadaan

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil - 1) + T(\lfloor n/2 \rfloor) + \Theta(n) \\ &= 2T(n/2) + \Theta(n), \end{aligned}$$

mistä seuraa $T(n) = \Theta(n \log n)$ luennoilla esitettyjen tulosten mukaan (esim. master-teoreema).

Pisteytys:

2 pistettä siitä, että on todettu mitä algoritmi tekee järjestetyllä syötteellä ja analysoitu aika-vaativuus oikein

2 pistettä mediaanialgoritmin käyttämisestä ja sen lineaarinen aikavaativuuden toteamisesta

2 pistettä muokatun algoritmin aikavaativuusanalyysistä

Huomautuksia: Kuten monet olivat todenneetkin, mediaanialgoritmin käyttäminen ei ole käytännössä järkevää, koska siitä aiheutuu suuri vakiokerroin ja yksinkertaisempiakin $O(n \log n)$ -järjestämisalgoritmeja tunnetaan. Tässä esitetty kysymys kuitenkin vaatii vastaukseen juuri mediaania.

Typillinen virhe oli esittää satunnaista tai jotain heuristisesti valittua jakoalkiota. Erityisesti on huomattava, että keskiarvon käyttäminen ei ole lainkaan sama kuin mediaanin.

Toinen virhe oli, ettei ollut todettu mediaanialgoritmin lineaarisuutta.

3. Ahne algoritmi kohtaan (a):

```
I := ∅
c := 0
valitse i ∈ {1, ..., n} - I jolla s_i on pienin
while c + s_i ≤ L do
    c := c + s_i
    I := I ∪ {i}
    valitse i ∈ {1, ..., n} - I jolla s_i on pienin
return I
```

(Pienimmän arvon s_i etsiminen on yleisessä tapauksessa tehokkainta toteuttaa kasan avulla, mutta tämä ei tässä yhteydessä ole oleellista.)

Lause Ahne algoritmi palauttaa optimaalisen ratkaisun, ts. sellaisen $I \subseteq \{1, \dots, n\}$ että $\sum_{i \in I} s_i \leq L$ ja $|I|$ on pienin mahdollinen.

Todistus Osoitetaan induktiolla, että I on aina jonkin optimaalisen ratkaisun osajoukko. Aluksi $I = \emptyset$ ja väite pätee.

Olkoon $I \subseteq J$ jollain optimaalisella J , ja tarkastellaan tilannetta, kun algoritmi päivittää $I := I \cup \{i\}$. Tällöin siis erityisesti $I \cup \{i\}$ on laillinen ratkaisu, joten $|J| \geq |I| + 1$.

Jos $i \in J$, väite selvästi pysyy voimassa. Jos $i \notin J$, valitaan jokin $j \in J - I$. Alkioiden i valintaperiaatteen nojalla $s_i \leq s_j$. Siis $J - \{s_j\} \cup s_i$ on optimaalinen ratkaisu, jonka osajoukko $I \cup \{s_i\}$ on.

Siis I on aina jonkin optimiratkaisun osajoukko. Kun suoritus päättyy, mikään joukon I aito ylijoukko ei ole edes laillinen ratkaisu, joten I itse on optimaalinen ratkaisu. \square

Ahne algoritmi kohtaan (b):

```
I := ∅
H := {1, ..., n}
c := 0
b := min_i s_i
while c + b ≤ L do
    valitse i ∈ H jolla s_i on suurin
    H := H - {i}
    if c + s_i ≤ L
        c := c + s_i
        I := I ∪ {i}
return I
```

Algoritmi ei aina tuota optimaalista ratkaisua, esim. tapauksessa $n = 4$, $s_1 = 1/2$, $s_2 = 1/3$, $s_3 = s_4 = 1/4$ algoritmi tuottaa $I = \{1, 2\}$ vaikka optimaalinen olisi $\{1, 3, 4\}$.

Pisteytys: Sekä (a)- että (b)-kohdassa 1 piste algoritmista, 2 pistettä optimaalisuustarkastelusta. Algoritmit sinänsä ovat aika ilmeisiä, pääpaino oli optimaalisuuden tai ei-optimaalisuuden täsmällisessä toteamisessa.

Huomautuksia: Tyypillisessä vastauksessa kaikki muu oli oikein, mutta (a)-kohdan todistus hyvin epämääräinen. Tehtävässä nimenomaan pyydettiin todistuksia, ja kaikki muu olikin itse asiassa aika ilmeistä, joten tässä nimenomaan haluttiin täsmällinen päättelyketju (vaikka tietysti esim. tieteellisessä artikkelissa tämän tasoiset jutut sivuutetaan ilmeisinä). Monissa vastauksissa on oleellisesti todettu, että I minimoi summan $\sum_{i \in I} s_i$ kun $|I|$ on kiinnitetty. Miten tästä seuraa se mitä halutaan, eli että I maksimoi koon $|I|$ kun summa $\sum_{i \in I} s_i$ on rajoitettu? Voiko olla olemassa J jolla $|J| > |I|$ (vaikka jopa $|J| = |I| + 2$) ja $\sum_{i \in I} s_i < \sum_{i \in J} s_i \leq L$? Vastaus on tietysti, että ei voi, mutta tämä ei ole mitenkään sen ilmeisempää kuin se alkuperäinen väite, jota tässä ollaan todistamassa.

Sivuhuomautus: Kohdan (b) ongelma sisältää osatapauksenaan NP-täydellisen ongelman PARTITION:

Annettu: s_1, \dots, s_n

Kysymys: päteekö $\sum_{i \in I} s_i = \frac{1}{2} \sum_{i=1}^n s_i$ jollain $I \subseteq \{1, \dots, n\}$

Tarkan polynomisen algoritmin löytyminen olisi siis ei-luultavaa.

4. *Perusratkaisu:* Muodostetaan taulukot $L[1 \dots n]$ ja $P[1 \dots n]$, missä $L[k]$ on pisimmän alkioon $A[k]$ päättyvän kasvavan jonon pituus ja $P[k]$ sen toiseksi viimeisen alkion indeksi. Siis ("reunoja" lukuunottamatta)

- $L[i] = L[P[i]] + 1$ ja
- $P[i]$ on sellainen j , että $A[j] < A[i]$ (jolloin alkioon $A[j]$ päättyvä jono voidaan jatkaa alkioon $A[i]$) ja $L[j] < i$ on suurin mahdollinen.

Muodostetaan taulukot vasemmalta alkaen, ja pidetään samalla muuttujissa ℓ ja k kirjaa pisimmän löydetyn jonon pituudesta ja päätepisteestä. Lopuksi luetaan pisin löydetty jono taulukkoon $S[1 \dots \ell]$.

```

ℓ := 1
for i := 1 to n do
  L[i] := 1
  P[i] := 0
  for j := 1 to i - 1 do
    if A[j] < A[i] and L[j] + 1 > L[i] then
      L[i] := L[j] + 1
      P[i] := j
    if L[i] > ℓ then
      ℓ := L[i]
      k := i
  for j := 1 to ℓ do
    S[ℓ - j + 1] := k
    k := P[k]

```

Aikavaativuus on selvästi $O(n^2)$.

Bonusratkaisu: Aikavaativuus on väistämättä $\Omega(n^2)$, jos halutaan laskea koko taulukko $L[1 \dots n]$. Osa arvoista $L[i]$ on kuitenkin "turhia" sikäli, että jo arvoa $L[i]$ laskettaessa voidaan todeta, että pisin kasvava polku ei missään tapauksessa voi kulkea alkion $A[i]$ kautta. Näin on, mikäli jokin j toteuttaa kaikki seuraavat kolme ehtoa:

- (a) $j < i$,
- (b) $A[j] < A[i]$ ja
- (c) $L[j] > L[i]$.

Nimittäin ehtojen (a) ja (b) nojalla millä tahansa alkion $A[i]$ kautta kulkevalla kasvavalla jonolla voidaan alkioon $A[i]$ päättyvä alkusegmentti korvata alkioon $A[j]$ päättyvällä, ja ehdon (c) nojalla näin voidaan pidentää jonoa.

Pidetään edelleen kiinni käsittelyjärjestyksestä vasemmalta oikealle. Siis alkion $A[i]$ tullessa käsittelyvuoroon on jo löydetty kaikki ”tarpeelliset” jonot osataulukosta $A[1 \dots i - 1]$. Edellisen perusteella ”tarpeellisiksi” jonoiksi riittää tulkita kullakin jononpituudella k se jono, jonka viimeinen arvo on pienin. Olkoon tämän jonon viimeinen indeksi talletettu arvoksi $R[k]$. Siis alkion $A[i]$ tullessa käsittelyvuoroon seuraavan pitää päteä kaikilla k :

- (a) jos osataulukossa $A[1 \dots i - 1]$ on kasvava k alkion jono, niin $R[k] < i$,
- (b) jokin k alkion kasvava jono päättyy alkioon $R[k]$ ja
- (c) jos jokin k alkion kasvava jono päättyy alkioon $j < i$, niin $A[j] \geq A[R[k]]$.

Sovitetaan, että jos k alkion kasvavia jonoja ei vielä ole löydetty, niin $R[k] = n + 1$.

Alkion $A[i]$ käsittelyssä pitää siis huolehtia, että tämä invariantti saadaan voimaan, kun i korvataan arvolla $i + 1$. Siis jos jokin k alkion kasvava jono päättyy alkioon $A[i]$, ja $A[i] < A[R[k]]$, pitää vaihtaa $R[k] := i$. Kuinka monella arvolla k tämä vaihto voidaan joutua tekemään?

Tarkastellaan jonoa $A[R[1]], \dots, A[R[\ell]]$, missä ℓ on pisimmän löydetyn kasvavan jonon pituus, eli löydettyjen ”tarpeellisten” jonojen viimeisiä alkioita jonon pituusjärjestyksessä. Keskeinen havainto on, että tämä jono on aidosti kasvava. Nimittäin jos jokin k alkion kasvava jono päättyy alkioon $A[R[k]]$ missä $R[k] \leq i$, niin varmasti jokin $k - 1$ alkion kasvava jono päättyy johonkin alkioon $A[j]$ jolla $A[j] < A[R[k]]$ ja $j < R[k] \leq i$.

Siis on olemassa korkeintaan yksi sellainen k , että $A[R[k]] < A[i] \leq A[R[k + 1]]$. Nyt alkio $A[i]$ ei kelpaa minkään yli k -alkioisen jonon jatkoksi, koska nämä kaikki menevät alkion $A[i]$ ”yli”. Pituuksia $k' < k$ oleville jonoille $A[i]$ kyllä kelpaisi jatkoksi, mutta saatavat jonot olisivat ”turhia”, koska parempiakin on jo löydetty. Ainoastaan tällä yhdellä k alkio $A[i]$ voi aidosti parantaa $(k + 1)$ -alkioisten jonojen tilannetta (eli ”alentaa” loppupistettä).

Ottamalla tarkemmin huomioon, mitä ”reunoilla” tapahtuu, saadaan seuraava algoritmi:

```

A[0] := -∞
A[n + 1] := +∞
R[0] := 0
R[1] := 1
for i := 2 to n do R[k] := n + 1
ℓ := 1
for i := 2 to n do
    etsi binäärihaulla k ∈ {0, ..., ℓ} jolla A[R[k]] < A[i] ≤ A[R[k + 1]]
    if A[i] < A[R[k + 1]] then
        R[k + 1] := i
        P[i] := R[k]
        if k = ℓ then ℓ := ℓ + 1
i := R[ℓ]
for j := 1 to ℓ do
    S[ℓ - j + 1] := A[i]
    i := P[i]

```

Edellä esitetyn perusteella binäärihaku todella löytää yksikäsitteisen oikean arvon k . Binäärihaun aikavaativuus on $O(\log \ell) = O(\log n)$, joten koko algoritmin aikavaativuus on $O(n \log n)$.

Pisteytys ja huomautuksia: Selvästi tyypillisin virhe oli, että oli ratkaistu rajoitetumpi ongelma, jossa kasvavan jonon vaaditaan olevan yhtenäinen; siis $i_{j+1} = i_j + 1$. Joko oli selvästi luettu tehtävä väärin, tai muuten päädytty algoritmiin joka ratkaisee vain em. rajoitetun ongelman. Tämä rajoitus muuttaa tehtävän helpoksi ohjelmointitehtäväksi, ja lisäksi jo tehtävänannosta olisi pitänyt arvata, että oikea ratkaisu ei voi olla helppo $O(n)$ algoritmi. Tämän takia tämän rajoitetun ongelman ratkaisevista algoritmeista on saanut nolla pistettä.

Jos edellinen virhe oli vältetty, oli tyypillisesti löydetty jokin toimiva taulukointi-idea. Toteutuksen korrektiuden mukaan tästä on tullut 4–6 pistettä.

Jos on selvästi yritetty tehdä jotain ei-yhtenäisten jonojen ongelmalle, mutta ei ole löydetty taulukointi-ideaa, on saanut 1–2 pistettä.

Bonustehtävään ei ollut varteenotettavia ratkaisuyrityksiä, mutta se onkin hyvin vaikea ratkaista tenttitilanteessa.