811312A Tietorakenteet ja algoritmit 2018-2019, Harjoitus 2 ratkaisu

Harjoituksen aiheena on algoritmien oikeellisuus.

Tehtävä 2.1 ”Kahvipurkkiongelma”**.** Kahvipurkissa P on valkoisia ja mustia kahvipapuja, yhteensä vähintään kaksi kappaletta. Lisäksi saatavilla on suuri määrä mustia papuja. Osoita, että seuraava algoritmi päättyy. Merkitään purkissa P kulloinkin olevien papujen määrää N(P):llä.

```
Syöte: Kahvipurkki P, jossa on valkoisia ja mustia papuja,
alussa N(P) >= 2.
Tuloste: Purkin viimeinen papu.

PAVUT(P)
1.   while N(P) > 1
2.      valitse satunnaisesti kaksi papua purkista P
3.      if pavut samanväriset
4.         heitä pois pavut ja lisää purkkiin musta papu
5.      else
6.         palauta valkoinen papu purkkiin ja heitä pois musta papu
7.   return purkkiin jäänyt papu
```

Voidaanko viimeisen pavun väriä tietää ennakolta, jos tiedetään purkissa alussa olevien valkoisten ja mustien papujen lukumäärät?

Ratkaisu. Algoritmin päättyminen voidaan havaita seuraavasti: Jokaisella silmukan kierroksella purkista poistetaan kaksi papua ja palautetaan yksi papu. Näin ollen papujen määrä N(P) pienenee joka kierroksella ja saavuttaa lopulta arvon 1. Siten algoritmi päättyy (ja N(P) on siis konvergentti).

Tarkastellaan vielä viimeisen pavun väriä. Jos purkissa on kaksi papua, niin on kaksi mahdollisuutta:

1. Purkissa on joko kaksi mustaa tai kaksi valkoista papua. Tällöin viimeiseksi purkkiin jää musta papu.
2. Purkissa on valkoinen ja musta papu. Tällöin viimeiseksi purkkiin jää valkoinen papu.

Koodia tutkimalla huomataan, että valkoisia papuja vähennetään joko kahdella tai ei lainkaan. Näin ollen saadaan taas kaksi mahdollisuutta:

1. Purkissa on alun perin parillinen määrä valkoisia papuja. Koska jokaisen kierroksen jälkeen purkkiin myös jää parillinen määrä valkoisia papuja, niin kaksi viimeistä ovat joko kaksi mustaa tai kaksi valkoista, jolloin viimeiseksi purkkiin jää musta papu.
2. Purkissa on alun perin pariton määrä valkoisia papuja. Koska jokaisen kierroksen jälkeen purkkiin myös jää pariton määrä valkoisia papuja, niin kaksi viimeistä ovat musta ja valkoinen papu, jolloin viimeiseksi purkkiin jää valkoinen papu.

Siten laskemalla valkoiset pavut alussa voidaan lopputulos tietää: Jos alussa valkoisia papuja on parillinen määrä, niin viimeinen papu on musta. Jos taas alussa valkoisia papuja on pariton määrä, niin viimeinen papu on valkoinen.

Tehtävä 2.2 Seuraava ongelma tunnetaan **selkäreppuongelmana** (**knapsack problem)**: On annettu kokonaislukujen joukko S = {s_1,s_2,…,s_n} ja kokonaisluku T. Etsi joukon S sellainen osajoukko (mikäli sellainen osajoukko on olemassa), että sen alkioiden summa on tarkalleen T. Esimerkiksi, kun S = {5,10,1,2,11}, niin jos T=24, on olemassa ratkaisu ({1,2,10,11}), mutta jos T=25, ratkaisua ei ole. Näytä, että kumpikaan seuraavista algoritmeista ei ole oikea ratkaisu selkäreppuongelmaan:

a) Valitaan osajoukon alkiot joukosta S järjestyksessä pienimmästä suurimpaan (best-fit).
b) Valitaan osajoukon alkiot joukosta S järjestyksessä suurimmasta pienimpään.

Ratkaisu. Algoritmin vääräksi osoittamiseen tarvitaan ainoastaan yksi vastaesimerkki, ts. laillinen syöte, jolla algoritmi toimii väärin. Tässä tapauksessa on löydettävä sellainen joukko S ja summa T, että joukon S alkioista voidaan muodostaa summa T, mutta algoritmi ei löydä sitä. Valitaan nyt S = {8,5,4,11} ja T = 20. Tällöin ongelmalla on ratkaisu {4,5,11}. Tutkitaan miten annetut algoritmit käyttäytyvät.

a) Algoritmi valitsee luvut pienimmästä suurimpaan järjestyksessä {4,5,8} ja luku 11 ei mahdu. Algoritmi ei löydä ratkaisua.
b) Algoritmi valitsee luvut suurimmasta pienimpään järjestyksessä {11,8} ja lisää ei mahdu. Näin ollen tämäkään algoritmi ei löydä ratkaisua.

Kumpikaan algoritmeista ei siis ole oikea ratkaisu ongelmaan.

Tehtävä 2.3 Seuraavan algoritmin oletetaan laskevan syötteenä saamansa taulukon alkioiden summan.

```
Syöte: Taulukko A[1,..,n], n >= 1
Tuloste: Taulukon alkioiden summa

ALKIOIDEN_SUMMA(A)
1.   sum = 0
2.   i = 1
3.   while i<=n
4.       sum = sum + A[i]
5.       i = i+1
6.   return sum
```

Laske algoritmia käyttäen taulukon A = {3,4,2} alkioiden summa (vastaus:9). Osoita algoritmi oikeaksi. Osoita algoritmin päättyminen sopivan konvergentin avulla. Muotoile ja todista oikeaksi silmukkainvariantti, joka vahvistaa algoritmin tuottavan oikean tuloksen.

Ratkaisu.

Lasketaan algoritmilla aluksi taulukon A = {3,4,2} alkioiden summa. Nyt siis A[1]=3, A[2]=4 ja A[3]=2.

Nyt n=3 ja aluksi sum=0 sekä i=1. Silmukan ensimmäisellä kierroksella sum päivittyy arvoon 0+A[1]=3 ja i arvoon 2. Seuraavalla kierroksella sum päivittyy arvoon 3+A[2] = 3+4 = 7 ja i arvoon 3. Kolmannella kierroksella sum päivittyy arvoon 7+A[3] = 7+2 = 9 ja i arvoon 4. Nyt i > n, joten seuraavalle kierrokselle ei enää mennä vaan algoritmi päättyy ja palauttaa arvon 9, kuten pitääkin.

Todistetaan nyt algoritmi oikeaksi. Algoritmin varsinaisen työn tekee silmukka. Sen ulkopuolella on vain kaksi sijoituskäskyä ja return. Tällaisessa algoritmissa oikeellisuus liittyy keskeisesti silmukkainvarianttiin.

Algoritmin päättymisen takaa konvergenttina *i*, joka alkuarvolla 1 mahdollistaa pääsyn silmukkaan, kasvaa jokaisella silmukan kierroksella ja saavuttaessaan arvon n+1 lopettaa silmukan suorittamisen. Sen jälkeen algoritmi päättyy ja palauttaa tuloksen.

Osoitetaan sitten algoritmin osittainen oikeellisuus.
Määritellään silmukkainvariantti: *Rivillä 3 muuttujan sum arvo k:nnen kierroksen jälkeen on taulukon arvojen A[1], ..., A[k] summa A[1]+...+A[k] ja i=k+1.*

Aluksi *i=1* ja *sum=0*. Ensimmäisen kierroksen jälkeen *sum = A[1]* ja *i=2*. Invariantti on siis alussa voimassa.

Invariantin ylläpito: Tehdään oletus, että k < n+1 ja että invariantti on voimassa k:nnen kierroksen jälkeen eli *sum=A[1]+...+A[k]* ja *i=k+1*. Seuraavalla (eli k+1:nnellä) kierroksella rivillä 4 sijoitetaan

sum = sum + A[k+1] = A[1]+...+A[k] + A[k+1].

Rivillä 5 kasvatetaan i:n arvoa yhdellä joten kierroksen jälkeen *sum = A[1]+...+A[k]+A[k+1]* ja *i=k+2*. Invariantti pysyy siis voimassa.

Loppu: Invariantti on alussa voimassa ja säilyy jokaisella kierroksella. Siten lopussa, kun *i=n+1,* on *sum = A[1]+...+A[n]* eli taulukon arvojen summa.
Algoritmin osittainen oikeellisuus on todistettu.

Algoritmi on oikeellinen, koska se on osittain oikeellinen ja päättyy.

Tehtävä 2.4 Seuraavan algoritmin tulisi kääntää syötetaulukon alkioiden järjestys päinvastaiseksi:

```
Syöte: Taulukko A[0,1,..,n-1], n >= 1 (n siis taulukon koko)
Tuloste: Kääntää taulukon A alkiot päinvastaiseen järjestykseen
REVERSE(A,n)
1.   i = 0
2.   while i <= n/2
3.       temp = A[i]
4.       A[i] = A[n-i-1]
5.       A[n-i-1] = temp
6.       i = i+1
7.   return
```

Algoritmi ei kuitenkaan ole korrekti. Implementoi algoritmi ja yritä löytää virhe. Korjaa tämän jälkeen ohjelma (ja algoritmi) toimimaan oikein.

Ratkaisu.

Suoritetaan algoritmia aluksi kolmepaikkaisella taulukolla A = [1,2,3]. Nyt siis n=3. Näin ollen while-silmukan ensimmäisellä kierroksella i=0 ja n-1-i=2; tällöin vaihdetaan taulukon alkiot A[0] ja A[2], eli taulukko on [3,2,1]. Silmukka suoritetaan vielä muuttujan i arvolla 1, jolloin i=1 ja n-1-i=1; vaihdetaan taulukon alkiot A[1] ja A[1]. Vaihdolla ei ole vaikutusta ja taulukko jää tilaan [3,2,1]. Algoritmi toimii siis tässä tapauksessa oikein.

Suoritetaan algoritmia seuraavaksi nelipaikkaisella taulukolla A = [1,2,3,4]. Nyt siis n=4. Näin ollen while-silmukan ensimmäisellä kierroksella i=0 ja n-1-i=3; tällöin vaihdetaan taulukon alkiot A[0] ja A[3], eli taulukko on [4,2,3,1]. Silmukka suoritetaan seuraavaksi muuttujan i arvolla 1, jolloin i=1 ja n-1-i=2; vaihdetaan taulukon alkiot A[1] ja A[2]. Taulukko on [4,3,2,1], mikä olisi oikea lopputulos. Silmukka suoritetaan kuitenkin vielä, kun i=2, jolloin n-i-1=1; jälleen vaihdetaan taulukon alkiot A[1] ja A[2] ja taulukko jää tilaan [4,2,3,1]. Algoritmi tuottaa siis tässä tapauksessa väärän tuloksen.

Yleistämällä yllä oleva päättely huomataan, että algoritmi toimii oikein, kun taulukossa on pariton määrä alkioita. Sen sijaan, kun taulukossa on parillinen määrä alkioita, algoritmi vaihtaa taulukon keskimmäiset alkiot takaisin alkuperäiseen järjestykseen ja tuottaa (yleensä) väärän tuloksen. Tätä voi tarkastella empiirisesti implementoimalla algoritmin. Edellisen tarkastelun perusteella voidaan päätellä, että algoritmi toimii muuten oikein, mutta while-silmukkaa suoritetaan liian pitkälle. Näin ollen algoritmin voi korjata helposti: oikea versio on

```
Syöte: Taulukko A[0,1,..,n-1], n >= 1 (n siis taulukon koko)
Tuloste: Kääntää taulukon A alkiot päinvastaiseen järjestykseen
REVERSE(A,n)
1.   i = 0
2.   while i < n/2
3.       temp = A[i]
4.       A[i] = A[n-i-1]
5.       A[n-i-1] = temp
6.       i = i+1
7.   return
```

Tehtävä 2.5 Ns. **puolitushaun** avulla voidaan hakea annettu arvo taulukosta, jossa alkiot ovat suuruusjärjestyksessä. Algoritmi toimii siten, että aluksi luetaan taulukon puolivälissä oleva arvo, jonka perusteella tiedetään, onko haettu arvo alku- vai loppupuolella taulukkoa. Tämän jälkeen puolitetaan jäljellä oleva osa, minkä jälkeen tiedetään missä taulukon neljänneksessä haettu arvo on. Näin jatkamalla löydetään arvo taulukosta tai havaitaan, että arvoa ei ole taulukossa. Seuraava iteratiivinen ohjelma pyrkii toteuttamaan puolitushaun järjestettyyn taulukkoon. Implementointi on kuitenkin virheellinen. Yritä löytää virhe. Implementoi ohjelma (joko C- tai Python-kielellä) ja yritä löytää testitapaus, jolla ohjelma toimii virheellisesti.

```
Syöte: Taulukko A[1,..,n], n >= 1, taulukon alkiot ovat kasvavassa
järjestyksessä A[1] <= A[2] <= … <= A[n]. Luku x jota haetaan taulukosta.
Tulostus: Alkion x indeksi taulukossa tai arvo -1, jos x ei esiinny
taulukossa.
HAKU(A,x)
 1. low=1
 2. up=n
 3. while low <= up
 4.     r = ⌊(low + up)/2⌋
 5.     if A[r] < x
 6.         low = r
 7.     else if A[r] > x
 8.         up = r
 9.     else
10.         return r
11. return -1
```

Merkintä $\lfloor x \rfloor$ (”floor”) tarkoittaa suurinta kokonaislukua, joka on korkeintaan yhtäsuuri kuin x.

Ratkaisu. Algoritmin ongelma on, että hakuintervalli ei pienene kaikilla syötteillä. Esimerkiksi, jos taulukko on [10,20,30,40,50] ja siitä haetaan arvoa 50, niin indeksimuuttujat päivitetään seuraavasti:

low = 1 up = 5

low = 3 up = 5

low = 4 up = 5

low = 4 up = 5

jne. Algoritmi jää siis ikuiseen silmukkaan. Samoin käy haettaessa mitä tahansa arvoa, joka ei ole taulukossa. Algoritmi voidaan korjata huomaamalla, että arvo A[r] on jo tarkastettu, joten sitä ei tarvitse enää ottaa huomioon. Näin ollen algoritmissa voidaan muuttaa rivi 6 muotoon

low = r+1

ja rivi 8 muotoon

up = r-1.

Tällöin algoritmi toimii oikein. Nämä asiat voi havaita kirjoittamalla algoritmista ohjelma.