Tietorakenteet, laskuharjoitus 10, 12-16.4

Seuraava TRAKLA-deadline 11.4.

1. Seuraavissa ongelmissa algoritmin syötteenä on taulukko T, jossa on joukko kokonaislukuja. Esitä kuhunkin ongelmaan ajassa $O(n \log n)$ toimiva ratkaisualgoritmi.

   (a) Esiintyykö jokin luku taulukossa monta kertaa?

   (b) Mikä luku esiintyy taulukossa useimmiten?

   (c) Mikä on pienin ero kahden taulukossa olevan luvun välillä?

2. Algoritmi saa syötteekseen taulukon T, jossa on joukko kokonaislukuja sekä kokonaisluvun k. Esitä ajassa $O(n \log n)$ toimiva algoritmi, joka tutkii onko taulukossa kahta lukua, joiden summa on k? Oletetaan tässä, että sama taulukon luku ei saa kuulua summaan kahta kertaa.

   Lisähaaste (ei edellytetä rastiin):

   (a) Esitä ajassa $O(n^2)$ toimiva algoritmi, joka tarkistaa, onko taulukossa kolmea lukua, joiden summa on k.

   (b) Esitä ajassa $O(n^2 \log(n))$ toimiva algoritmi, joka tarkistaa, onko taulukossa neljää lukua, joiden summa on k.

   Oletetaan myös näissä, että sama taulukon luku ei saa kuulua summaan kuin kerran.

3. Toteuta haluamallasi ohjelmointikielellä lomitusjärjestäminen tai pikajärjestäminen tai mieluiten molemmat.

   Java API:n `Arrays`-luokassa on määritelty metodi `sort` jonka avulla on mahdollista järjestää taulukollinen mitä tahansa olioita[1].

   Vertaile tällä ja edellisellä viikolla toteuttamiesi järjestämisalgoritmien sekä Javan Arrays.sort:in suorituskykyä erikokoisilla suurilla syötteillä, tapauksissa, joissa syötteenä oleva taulukko on

   - satunnaisessa järjestyksessä
   - valmiina järjestyksessä
   - valmiina käänteisessä järjestyksessä

   Jos käytät jotain muuta kieltä kuin Javaa, ota vertailukohdaksi kielesti standardissa oleva järjestämisalgoritmitoteutus.

   Huom: jos toteutat pikajärjestämisen suoraan sivujen 349 ja 351 mukaan, pahin tapaus sisältää hyvin syvän rekursion ja joudut kasvattamaan ajoaikaisen pinon kokoa välttyäksesi `java.lang.StackOverflowError`:ilta. Pinoa voi kasvattaa antamalla käynnistyksen yhteydessä parametrin `-Xss`, jonka yhteydessä määritellään ajonaikaisen pinon koko. Esim. `java -Xss100m Quicksort` määrittelee pinon kooksi 100 megatavua.

[1] olioiden on toteutettava rajapinta Comparable tai niille on määriteltävä Comparator-rajapinnan toteuttava vertailija. Alkeistyyppien käärinluokat Integer, Long ja Double sekä String toteuttavat Comparable-rajapinnan.

Jos valitset partition-operaation jakoalkioksi `a` keskimmäiseksi suurimman arvoista `A[p]`, `A[(p+q)/2]` ja `A[q]` vältyt pahimmalta tapaukselta jos taulukko on valmiiksi järjestetyssä.

4. Taulukon järjestäminen onnistuu ajassa $O(n \log n)$. Entä kuinka nopeasti taulukon voi sekoittaa? Määrittele tehokas algoritmi taulukon sekoittamiseen. Mikä on algoritmisi aikavaativuus ja tilavaativuus? Voit olettaa, että käytössäsi on funktio, joka palauttaa satunnaisen kokonaisluvun väliltä $1 \ldots n$ vakioajassa. Sekoittaako algoritmisi taulukon niin, että kaikki lukujen järjestykset ovat yhtä todennäköisiä?

5. Suunnatun verkon $G = (V, E)$ *transpoosi* G^{T} saadaan vaihtamalla kaikkien verkon G kaarten suunta. Siis $G^{\mathrm{T}} = (V, E^{\mathrm{T}})$, missä $E^{\mathrm{T}} = \{(v, u) \mid (u, v) \in E\}$. Esitä ajassa $O(|V| + |E|)$ toimiva algoritmi, joka muodostaa verkon G vieruslistaesityksen perusteella vieruslistaesityksen transpoosille G^{T}.

6. Toteuta haluamallasi ohjelmointikielellä labyrintin ratkaisija. Jos et halua ohjelmoida, riittää rastiin myös hyvin mietitty kohtuullisen tarkka pseudokoodi.

   Ohjelmalle annetaan labyrintin kuvaus, jossa `#` tarkoittaa seinää, `.` tarkoittaa lattiaa ja `X` tarkoittaa aloituskohtaa. Ohjelman täytyy tulostaa uudestaan labyrintti niin, että siihen on merkitty lyhin reitti ulos. Jos labyrintista ei ole reittiä ulos, ohjelman täytyy ilmoittaa siitä. Labyrintista pääsee ulos kaikista kohdista, joissa reunassa on lattiaa.

   Esimerkiksi ohjelmalle voidaan antaa seuraava syöte:

```
####.##.#
...#....#
#.#.###.#
#....X..#
#########
```

   Tällöin ohjelma tulostus on seuraava:

```
####.##o#
...#...o#
#.#.###o#
#....Xoo#
#########
```

   Lyhin reitti ulos muodostuu 5 askeleesta, joiden kohdalla tulosteessa on `o`.

   Labyrintti kannattaa tulkita sopivana verkkona, jossa mahdolliset olinpaikat ovat solmuja ja mahdolliset kulkureitit solmujen välisiä kaaria. Huomaa, että ohjelmassa verkkoa ei ole välttämätöntä esittää vieruslista- tai vierusmatriisimuodossa, vaan joku muu esitystapa saattaa olla ohjelmoinnin kannalta suoraviivaisempi tapa verkon ja sen kaarien esittämiseen.