

Kirjoja, laskinta tms. ei saa olla tentissä. Tehtävällä on yksi tai useampi numero maksimipistemäärän mukaan. Muiden kuin koodaa tai piirrä -tehtävien mallivastaukset ovat melko lyhyet, tyypillisesti 1, ..., 3 riviä pistettä kohti.

1. Taulukon A sisältö on ['A', 'L', 'G', 'O', 'R', 'I', 'T', 'M', 'I'], ja se indeksoidaan nolasta alkaen. Mikä on suurin sellainen i , että $A[0 \dots i-1]$ on keko pienin ylinnä?

7

2. Miten A :n sisältö muuttuu, kun tehtävän 3 aliohjelma suoritetaan tehtävän 1 vastauksen mukaisella i :n arvolla? Vastaukseksi kelpaa esimerkiksi kuva jossa alkioiden paikkojen vaihtumiset on näytetty nuolilla, taikka A :n sisältö näytettynä jokaisen muutoksen jälkeen.

['A', 'L', 'G', 'O', 'R', 'I', 'T', ' ', 'I'] 'M' otettiin sivuun

['A', 'L', 'G', ' ', 'R', 'I', 'T', 'O', 'I'] 'O' siirtyi 'M':n paikalle

['A', 'L', 'G', 'M', 'R', 'I', 'T', 'O', 'I'] 'M' laitettiin 'O':n paikalle

- 3&4. Kirjoita aliohjelma lisää_kekoon(i), joka vaihtaa A :n alkioiden järjestystä siten, että jos alussa $0 \leq i < A.koko$ ja $A[0 \dots i-1]$ on keko pienin ylinnä, niin lopussa $A[0 \dots i]$ on keko pienin ylinnä.

```
1 void lisää_kekoon( unsigned i ){
2     char apu = A[i]; unsigned j = (i-1)/2;
3     while( i > 0 && A[j] > A[i] ){ A[i] = A[j]; i = j; j = (i-1)/2; }
4     A[i] = apu;
5 }
```

Taulukossa A on n alkioita ja taulukossa B on m alkioita. On löydettävä alkio, joka on molemmissa, tai palautettava tieto, että sellaista ei ole.

```
1 alkio_tyyppi yhteinen( taulukko A, taulukko B ){
2     for( i = 0; i < n; ++i ){
3         for( j = 0; j < m; ++j ){
4             if( A[i] == B[j] ){ return A[i]; }
5         }
6     }
7     return ei_ole;
8 }
```

- 5&6. Ilmoita oheisen aliohjelman yhteinen parhaan, jokaisen ja huonoimman tapauksen ajan kulutus O - ja/tai Θ -merkinnällä.

$\Theta(1)$, $O(nm)$ ja $\Theta(nm)$

- 7&8. Perustele kaksi tehtävän 5 vastauksesi O - ja/tai Θ -merkinnöistä.

Jos $A[0] = B[0]$, niin aliohjelma lopettaa vakioajassa. Mitään riviä ei suoriteta enemmän kuin $nm + n + 1$ kertaa. Jos vastataan "ei ole", niin rivi 4 suoritetaan nm kertaa.

9. Hajautustaulu saattaa olla käytännössä nopein tapa ratkaista aliohjelman yhteinen ratkaisema tehtävä. Miksi se ei silti ole tehtävän 10 oikea vastaus?

Hajautustaulu lupaa hyvän keskimääräisen ajan kulutuksen, mutta ei hyvää huonointa ajan kulutusta.

10. Miten aliohjelman yhteinen ratkaisema tehtävä voidaan ratkaista paljon paremmalla huonoimman tapauksen ajan kulutuksella olettaen, että $n \geq m$? Sanallinen vastaus riittää (ei tarvitse antaa koodia).

Järjestetään B . Selataan A ja etsitään kutakin sen alkioita B :stä puolitushaulla.

11. Mikä on tehtävän 10 vastauksesi huonoimman tapauksen ajan kulutus?

$\Theta(n \log m)$

Sekalaisia kysymyksiä

12. Quicksort on viime aikoina menettänyt suosiotaan eräälle toiselle järjestämisalgoritmillemme. Mikä tämä toinen algoritmi on ja miksi Quicksort on menettänyt sille suosiotaan?

Mergesort. Se on vakaa toisin kuin Quicksort. Uusissa oliokielissä taulukon sisältö ei tyypillisesti ole peräkkäisissä muistilokeroissa vaan niissä on osoittimia sinne tänne muistiin. Tämä vie osan Quicksortin nopeusedusta, joka vanhoissa kielissä perustui siihen, että Quicksort hyödynsi nopeaa välimuistia käsittelemällä suuren osan aikaa lähellä toisiaan olevia alkioita.

13&14. Minkälaisessa tietorakenteessa suunnattu graafi kannattaa esittää, kun on etsittävä mahdollisimman lyhyt reitti paikasta A paikkaan B maanteitä pitkin? Luettele tarvittavat olio-, tietue- tms. tyypit ja mitä tietoalkioita (kuten kaaren pituus) niissä on.

Solmut ovat yhdessä taulukossa. Kaaret ovat toisessa taulukossa siten, että samasta solmusta lähtevät kaaret ovat peräkkäin. Solmussa on tieto siitä lähtevien kaarten sijainnista kaaritaulukossa, lyhimmän siihen mennessä löydetyn reitin pituus ja linkki edelliseen sillä reitillä. Kaaressa on pituus ja kärkipään solmun indeksi.

15. Kuten tehtävä 13, mutta nyt on lueteltava kaikki mahdollisimman lyhyet välimatkat kahden risteyksen välillä.

$|V| \times |V|$ -matriisin kohdassa $[i, j]$ on mahdollisimman lyhyen jo löydetyn solmusta i solmuun j vievän reitin pituus, sekä seuraavan solmun numeron ko. reitillä.

Syvyysshaun toimintaa havainnollistetaan usein solmujen väreillä valkoinen, harmaa ja musta.

16. Milloin solmun väri vaihdetaan, ja mistä väristä mihin väriin se vaihdetaan?

Solmun väri vaihdetaan valkoisesta harmaaksi kun solmu kohdataan ensimmäisen kerran. Se vaihdetaan harmaasta mustaksi kun kaikki solmusta lähtevät kaaret on tutkittu ja siitä peruutetaan pois.

17. Missä tapauksessa syvyyshaku etenee tutkimaansa kaarta pitkin ja missä tapauksessa ei etene?

Se etenee jos ja vain jos kaari vie valkoiseen solmuun.

18. Milloin syvyyshaku löytää silmukan?

Kun tutkittava kaari vie harmaaseen solmuun.

19. Perustele, että silloin on löytynyt silmukka.

Harmaat solmut muodostavat polun alkusolmusta nykyiseen solmuun, joten löydetystä solmusta on polku nykyiseen solmuun. Tutkittava kaari täydentää silmukan.

Binäärihakupuun solmussa on avain ja osoittimet vasen ja oikea. Oheiselle koodinpätkälle annetaan muuttujassa *s* osoitin binäärihakupuun juureen.

```
1  tulos = nil;
2  while( s != nil ){
3      if( s->avain < x ){ s = s->oikea; }
4      else{ tulos = s; s = s->vasen; }
5  }
```

20. Mihin *tulos* osoittaa lopuksi?

Ensimmäiseen solmuun, jonka avain on vähintään *x*.

21. Mihin *s* osoittaa lopuksi?

Ei minnekään.

22&23. Perustele, että oheinen koodinpätkä ei jää ikuisen silmukkaan.

Jokaisella kierroksella se etenee yhden kaaren verran puussa alaspäin.

24. Perustele, että aina rivin 3 alussa joko *tulos* on *nil* tai *s* osoittaa jotakin solmua osoittimen *tulos* osoittaman solmun vasemmassa alipuussa.

Rivin 3 alussa ei *s* voi olla *nil*, koska muuten sinne ei oltaisi tultu riviltä 2. Jos *tulos* ei ole *nil*, niin *else*-haara on suoritettu ainakin kerran. Kun se suoritettiin viimeksi, *s* siirtyi *tulos*:sen vasempaan alipuuhun. Sen jälkeen *s* pysyy siellä kunnes muuttuu *nil*:ksi, koska *s* liikkuu vain alaspäin.

25. Perustele, että jos etsittävää solmua ei ole olemassa, niin kun oheinen koodinpätkä on lopettanut, on sen vastaus oikea.

Jos etsittävää solmua ei ole olemassa, on jokaisen solmun avain pienempi kuin *x*, joten koodinpätkä suorittaa riviä 3 kunnes *s* on *nil*. Tällöin myös *tulos* on *nil*.

26&27. Perustele, että jos etsittävä solmu on olemassa, niin aina rivin 2 alussa pätee seuraava invariantti: joko etsittävä solmu on *s*:n osoittamassa alipuussa tai *tulos* osoittaa siihen.

Ensimmäisellä kerralla tämä on totta, koska *s* osoittaa koko puuta ja etsittävä solmu on olemassa. Jos *s*->avain < *x*, niin etsittävä solmu ei voi olla *s* eikä *s*:n vasemmassa alipuussa, joten invariantin vuoksi se on *s*:n oikeassa alipuussa tai *tulos* osoittaa siihen. Muussa tapauksessa *s*->avain ≥ *x*, joten etsittävä solmu on invariantin vuoksi *s* tai sen vasemmassa alipuussa tai *tulos*, mutta tehtävän 24 vuoksi se ei voi olla *tulos*.

28. Perustele, että jos etsittävä solmu on olemassa, niin kun oheinen koodinpätkä on lopettanut, on sen vastaus oikea.

Lopuksi *s* on *nil*, joten edellisen invariantin vuoksi etsittävä solmu on *tulos*.

29&30. Jokaisessa solmussa on myös osoitin vanh. Kirjoita aliohjelma, jolle annetaan muuttujassa *s* osoitin binääripuun solmuun ja joka muuttaa *s*:n osoittamaan edelliseen solmuun.

```
1  if( s != nil ){
2      if( s->vasen != nil ){
3          s = s->vasen;
4          while( s->oikea != nil ){ s = s->oikea; }
5      }else{
6          while( s->vanh != nil && s == s->vanh->vasen ){ s = s->vanh; }
7      }
8  }
```

Loppu