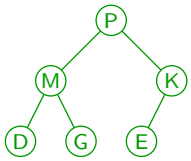
Vastaa tentin järjestäjän antamalle paperille (ei kysymyspaperille). Kirjoja, laskinta tms. ei saa olla tentissä. Kukin tehtävä on 1 tai 2 pisteen arvoinen. Muiden kuin koodaa tai piirrä -tehtävien mallivastaukset ovat melko lyhyet, 1, ..., 4 riviä.

1. Perustele, että [P, I, K, A, N, A] ei ole keko, jossa on suurin ylinnä.

   N:n vanhenpi on I, joka on N:ää pienenpi.

2. Piirrä keko [P, M, K, D, G, E] puuna.



3. Mikä on lopputulos, kun keosta [P, M, K, D, G, E] poistetaan yksi alkio?

   [M, G, K, D, E]

Arvioi algoritmin suoritusaikaa n:n funktiona sekä O- että Ω-merkinnällä. Perustele vastauksesi. Taulukko `A` indeksoidaan $0, \ldots, n-1$.

4&5. `i = 0; while( i < n && A[i] != x ){ i = i+1; }`

   $O(n)$, koska `i` selaa `n-1`:een saakka jos `A` on täynnä nollaa ja `x` on 1.
   $\Omega(1)$, koska algoritmi lopettaa heti jos `A` on täynnä nollaa ja `x` on 0.

6&7.
```
i = 0; tulos = 0;
while( i < n ){ if( A[i] > 0 ){ tulos += A[i]; } i = 2*i+1; }
```

   $O(\log n)$ ja $\Omega(\log n)$, koska `i` suunnilleen kaksinkertaistuu joka kierroksella, eikä suoritusaika riipu muusta kuin `n`:stä.

Alla on esitetty valintajärjestäminen C++-koodina. Olkoon $n =$ `A.size()`.

```
1  void Valintajarjesta( taulukko & A ){
2    for( unsigned i = 0; i+1 < A.size(); ++i ){
3      unsigned p = i;
4      for( unsigned j = i+1; j < A.size(); ++j ){
5        if( A[j].x < A[p].x ){ p = j; }
6      }
7      alkio apu = A[i]; A[i] = A[p]; A[p] = apu;
8    }
9  }
```

8. Mitä i:n arvosta tiedetään rivin 3 alussa?

   $0 \leq i < n-1$

9. Mitä i:n, j:n ja p:n arvoista tiedetään rivin 5 alussa?

   $0 \leq i \leq p < j < n$

10. Mitä i:n ja p:n arvoista tiedetään rivin 7 alussa?

    $0 \leq i \leq p < n$ ja $i < n-1$

11&12. Anna valintajärjestämisen ulommalle silmukalle invariantti!

Koko taulukossa on alkuperäiset alkiot. Osa $A[0 \ldots i-1]$ on kasvavassa järjestyksessä. Joko $i = 0$ tai jokainen osan $A[i \ldots n-1]$ alkio on vähintään yhtäsuuri kuin $A[i-1]$. Pätee $0 \leq i < n$.

UNION-FIND-tietorakenteen alkiossa on osoitin e kohti edustajaa ja kokonaisluku k tallettamassa korkeusarvion. Edustajan e-osoitin osoittaa edustajaan itseensä. Olkoot a ja b osoittimia alkioihin.

13&14. Kirjoita aliohjelma EDUSTAJA(a), joka palauttaa osoittimen a:n edustajaan. Yksi piste tulee siitä, että aliohjelma palauttaa oikean tuloksen ja toinen siitä, että se toteuttaa UNION-FIND:iin kuuluvan nopeutuskikan.

```
1 EDUSTAJA(a)
2 x := a
3 while x.e ≠ x do x := x.e
4 while a ≠ x do y := a.e ; a.e := x ; a := y
5 return x
```

15. Kirjoita aliohjelma SAMASSA(a,b), joka palauttaa true tai false sen mukaan, ovatko a ja b samassa joukossa (eli samassa graafin komponentissa).

```
1 SAMASSA(a,b)
2 if EDUSTAJA(a) = EDUSTAJA(b) then return true else return false
```

16&17. Kirjoita aliohjelma YHDISTÄ(a,b), joka yhdistää joukot, joihin a ja b kuuluvat. Yksi piste tulee toimivasta yhdistämisestä ja toinen nopeutuskikasta.

```
1 YHDISTÄ(a,b)
2 a := EDUSTAJA(a) ; b := EDUSTAJA(b)
3 if a.k < b.k then a.e := b else b.e := a
4 if a.k = b.k then a.k := a.k + 1
```

Sekalaisia kysymyksiä

18. Miten Dijkstran algoritmi ylläpitää reitin etsinnän aikana tietoa, jonka avulla reitti voidaan lopuksi tulostaa?

Jokaisessa solmussa (paitsi lähtösolmussa) on linkki reitillä edelliseen solmuun.

19. Millä algoritmilla voidaan selvittää, onko suunnatussa graafissa silmukoita? Algoritmin nimi riittää vastaukseksi.

syvyyshaku

20. Mikä etu Bellman–Ford-algoritmilla on kurssin muihin reitinetsintäalgoritmeihin verrattuna?

Se sallii silmukat, joiden kokonaispaino on negatiivinen.

21. Edmonds–Karp-algoritmi löytää maksimi vuon. Kuvaile lyhyesti, miten se toimii.

    Jokaisella kierroksella se etsii mahdollisimman lyhyen polun lähdöstä maaliin, jossa on käyttämätöntä kapasiteettia, ja kasvattaa voita ko. polulla sen verran kuin käyttämätön kapasiteetti ahtaimmillaan sallii.

22. Miksi Edmonds–Karp-algoritmi tarvitsee monimutkaisemman esityksen graafille kuin esimerkiksi Dijkstran algoritmi?

    Edellisen kohdan polulle hyväksytään myös takaperoisia kaaria.

23. Miten hajautustaulun kantataulukon koko kannattaa valita (eli sen taulukon, jota indeksoidaan hajautusfunktion tuloksella)?

    Suunnilleen odotettavissa oleva alkioiden määrä.

24. Minkä toiminnon tasapainotettu binäärihakupuu tarjoaa paljon tehokkaammin kuin hajautustaulu?

    Selaamisen avainten mukaisessa järjestyksessä.

DNA-sekvenssi voidaan esittää merkkijonona, jossa ei esiinny muita merkkejä kuin A, C, G ja T. Eliölajien sukulaisuutta voi mitata laskemalla niiden DNA-sekvenssien välisten erojen määrän oheisella aliohjelmalla. Olkoot `n1` = `dna1.size()` ja `n2` = `dna2.size()`. Alkuperäinen kutsu on `etaisyys( n1, n2 );`.

```
1  unsigned etaisyys( unsigned i1, unsigned i2 ){
2    if( i1 == 0 ){ return i2; }
3    if( i2 == 0 ){ return i1; }
4    unsigned tulos = etaisyys( i1-1, i2-1 );
5    if( dna1[ i1-1 ] != dna2[ i2-1 ] ){ ++tulos; }
6    tulos = std::min( tulos, 1 + etaisyys( i1-1, i2 ) );
7    tulos = std::min( tulos, 1 + etaisyys( i1, i2-1 ) );
8    return tulos;
9  }
```

25. Kirjoita mahdollisimman tiukka ehto, jonka luvut `i1` ja `i2` toteuttavat aina rivin 2 alussa.

    $0 \leq$ `i1` $\leq$ `n1` ja $0 \leq$ `i2` $\leq$ `n2`

26. Perustele, että `etaisyys` ei indeksoi taulukoita `dna1` ja `dna2` laittomasti eikä kutsu itseään laittomilla indekseillä.

    Riville 4 tultaessa `i1` > 0 ja `i2` > 0, koska muuten olisi lopetettu rivillä 2 tai 3. Rekursiivisissa kutsuissa `i1` ei muutu tai pienenee yhdellä, ja samoin `i2`. Siksi $0 \leq$ `i1` $- 1 <$ `i1` $\leq$ `n1` ja $0 \leq$ `i2` $- 1 <$ `i2` $\leq$ `n2` riveillä $4, \ldots, 7$.

27. Perustele, että DNA-sekvenssien AGAA ja GAC välinen etäisyys on 2.

    Optimireitillä alun A jätetään pois (rivi 6), G ei muutu (rivi 5), A ei muutu (rivi 5) ja lopun A korvataan C:llä (rivi 5).

28. Miksi `etaisyys` on hidas jo melko lyhyillä DNA-sekvensseillä?

    Se kutsuu itseään samoilla parametreilla uudelleen ja uudelleen.

29&30. Kirjoita olennaisesti nopeampi aliohjelma, joka laskee saman asian kuin `etaisyys`. Oleta, että sopiva taulukko on valmiiksi luotu ja alustettu.

```
1  unsigned taulukoiva( unsigned i1, unsigned i2 ){
2    if( paras[ i1 ][ i2 ] == ~0u ){
3      unsigned tulos;
4      if( i1 == 0 ){ tulos = i2; }
5      else if( i2 == 0 ){ tulos = i1; }
6      else{
7        tulos = taulukoiva( i1-1, i2-1 );
8        if( dna1[ i1-1 ] != dna2[ i2-1 ] ){ ++tulos; }
9        tulos = std::min( tulos, 1 + taulukoiva( i1-1, i2 ) );
10       tulos = std::min( tulos, 1 + taulukoiva( i1, i2-1 ) );
11     }
12     paras[ i1 ][ i2 ] = tulos;
13   }
14   return paras[ i1 ][ i2 ];
15 }
```

loppu