Tietorakenteet, laskuharjoitus 4, 9.-12.2.

1. Tutustu Javan valmiisiin ArrayList- ja LinkedList-luokkiin.

   Ohjeita ArrayListin käyttöön löytyy mm. Ohjelmoinnin jatkokurssin materiaalista:
   `http://www.cs.helsinki.fi/u/wikla/Ohjelmointi/Sisalto/5/ObjListMap.html`

   Toteuta molempien avulla jono (ks. moniste sivu 88).

   Vertaile jonototeutustesi suorituskykyä empiirisesti, eli mittaa kuinka hyvin jonot toimiviat eripituisilla suurilla syötteillä. Suoritukseen kuluvan ajan voi mitata esim. seuraavasti:

   ```
   int kertaa = 100000;
   ArrayListJono j1 = new ArrayListJono();
   long alku = System.currentTimeMillis();
   for ( int i=0; i<kertaa; i++ ) j1.enqueue(i);
   for ( int i=0; i<kertaa; i++ ) j1.dequeue();
   long loppu = System.currentTimeMillis();
   System.out.println("aikaa kului :"+(loppu-alku)+" millisekuntia");
   ```

   Piirrä kuvaaja molempien versioiden käyttämästä ajasta suoritettujen operaatioiden lukumäärän suhteen.

   Mitä johtopäätöksiä jonojen suorituskykyjen empiirisestä analyysistä voi tehdä? Miten empiirinen analyysi suhtautuu O-analyysiin? Mistä suorituskykyero johtuu?

2. Toteuta Javalla tai jollain haluamallasi kielellä taulukkoon perustuva pino, jossa taulukon kokoa kasvatetaan jos push-operaation yhteydessä taulukossa ei ole enää tilaa uudelle alkiolle.

   Toteuta pinosta kaksi versiota: toisessa taulukon koko aina kaksinkertaistetaan, toisessa taas taulukon koko kasvaa aina sadalla. Molemmissa taulukon koko on aluksi 100.

   Vertaile empiirisesti toteutustesi suorituskykyä. Jälleen olemme kiinnostuneita lähinnä isoista syötteistä. Piirrä kuvaaja molempien versioiden käyttämästä ajasta suoritettujen push-operaatioiden suhteen.

   Mitä johtopäätöksiä pinojen suorituskykyjen empiirisestä analyysistä voi tehdä? Mistä suorituskykyero johtuu?

3. Toteuta Javalla (tai jollakin ohjelmointikielellä) yksisuuntainen linkitetty lista johon talletetaan kokonaislukuja. Yksisuuntaisessa linkitetyssä listassa solmuilla ei ole prev-attribuutteja, esim. tehtävän 6 kuvassa oleva lista on yksisuuntainen. Listalla on seuraavat metodit:

   - `delete(k)` poistaa listalta luvun k (tämä siis poikkeaa hieman luentomonisteen operaatiosta, jossa parametrina oli viite poistettavaan listasolmuun)
   - `search(k)` kertoo onko luku k listalla
   - `add(k)` lisää luvun k listalle
   - `list()` tulostaa listalla olevat luvut

   Tee myös pääohjelma jolla testaat listaa. Mikä on metodien aikavaativuus?

4. Muuta toteutustasi siten, että lista pidetään koko ajan suuruusjärjestyksessä. Miten metodien aikavaativuus muuttuu? Mikä etu suuruusjärjestyksen ylläpitämisessä on?

   Bonus: Toteuta listalle metodi `reverse()`, joka kääntää listan sisällön, eli aiempi listan alussa ollut menee viimeiseksi, jne. Metodin tulee toimia vakiotilassa. Aikaa se saa käyttää $\mathcal{O}(n)$, missä on n listalla olevien alkioiden määrä.

   Laskarirastiin ei tätä bonusosaa vaadita.

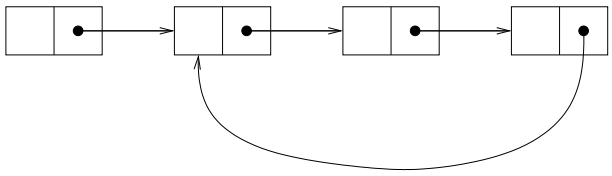
5. Jos ei haluta käyttää dynaamista muistinvarausta (**new**-operaatio tms.), myös linkitetyn listan voi toteuttaa taulukkona. Tällöin viitteet listan solmuihin esitetään yksinkertaisesti taulukon indekseinä. Esim. (järjestämätöntä kahteen suuntaan linkitettyä) listaa $(68, 24, 15, 17)$ voisi vastata seuraava taulukkoesitys:

	key	*next*	*prev*
1:			
2:			
3:	24	7	5
4:			
5:	68	3	0
6:			
7:	15	8	3
8:	17	0	7
9:			
10:			

head: 5

Esitä pseudokoodilla INSERT- ja DELETE-operaatioiden toteutus tälle talletusrakenteelle. Ratkaisusi pitäisi olla sellainen, että n-rivinen taulukko riittää, jos listan pituus millään yksittäisellä ajanhetkellä ei ole suurempi kuin n. Toisin sanoen DELETE-operaatioiden vapauttamat taulukon rivit pitää pystyä käyttämään uudelleen INSERT-operaatioissa. Koita myös välttää turhaa alkioiden siirtelemistä paikasta toiseen.

6. Esimerkiksi ohjelmointivirheen takia listarakenteeseen voi syntyä silmukka:



Esitä algoritmi, joka tutkii, onko syötteenä annetussa listassa tällainen silmukka. Algoritmisi ei saa muuttaa syötteenä annettua listaa. Analysoi myös algoritmisi aika- ja tilavaativuutta.

Lisähaaste: ongelma on mahdollista ratkaista lineaarisessa ajassa käyttäen vakiomäärä muistia. Rastiin riittää tehottomampikin algoritmi.