Tietorakenteet 2018
Harjoitukset 3, ratkaisut (Viikko 39)

1.-2.
```
/*
   void insertLast (Object o)
   laittaa olion o jonon viimeiseksi
*/
public void insertLast (Object o)
{
  DoubleLinkNode n = new DoubleLinkNode (o);
  if (IsEmpty ())
    head = tail = n;
  else
    {
      n.prev = tail;
      tail.next = n;
      tail = n;
    }
}

/*
   Object removeLast ()
   Poistaa ja palauttaa jonon viimeisen alkion.
   Metodi palauttaa null-arvon, jos jono on tyhjä.
*/
public Object removeLast ()
{
  Object result = null;
  if (! IsEmpty ())
    {
      result = tail.data;
      tail = tail.prev;
      if (tail != null)
        tail.next = null;
      else
        head = null;
    }
  return result;
}
```

Toimintaperiaate: Lisätään ja poistetaan kaksoislinkitetyn listan lopusta solmuja. Loppuun päästään suoraan kelaamatta listaa läpi Deque-luokan olion tail-viitteen kautta, algoritmit ovat symmetrisiä alkuun lisäyksen ja poiston kanssa.

3.
```
/*
   void Reverse ()
   Kääntää jonon alkiot päinvastaiseen järjestykseen
*/
```

```
public void reverse ()
{
  DoubleLinkNode tmp;
  for (DoubleLinkNode i = head; i != null; i = i.prev)
    {
      tmp = i.next;
      i.next = i.prev;
      i.prev = tmp;
    }
  tmp = head;
  head = tail;
  tail = tmp;
}
```

Algoritmin toimintaperiaate: Vaihdetaan jokaisessa solmussa viitteet keskenään (toteutettu for-silmukassa, huom. eteneminen tehdään vasta viitteiden vaihdon jälkeen, joten edetään taaksepäin nykyisestä solmusta) ja vaihdetaan jonon alun ja lopun viitteet keskenään.

Kompleksisuus: For-silmukka suoritetaan n kertaa, missä n on listan solmujen lukumäärä. Silmukan sisällä suoritetaan vakiomäärä operaatioita, joten sen kompleksisuus on $n\mathcal{O}(1) = \mathcal{O}(n)$. Silmukan jälkeen suoritetaan vielä alun ja lopun viitteiden vaihto, johon tarvitaan vakiomäärä operaatioita, joten koko algoritmin kompleksisuus on $\mathcal{O}(n) + \mathcal{O}(1) = \mathcal{O}(n)$.

4. Vastausten notaatiossa $\mathcal{O}(f(n))$ kuvaa sekä kompleksisuusluokkaa $\mathcal{O}(f(n))$ että jotain luokkaan $\mathcal{O}(f(n))$ kuuluvaa funktiota. Näin ollen $\mathcal{O}(1) + \mathcal{O}(n) = \mathcal{O}(n)$ tarkoittaa lyhyemmin samaa kuin $f(n) \in \mathcal{O}(1)$, $g(n) \in \mathcal{O}(n)$ ja $h(n) = f(n) + g(n)$, siis $h(n) \in \mathcal{O}(n)$

   a) For-silmukka suorittaa vakiomäärän operaatioita ja ne suoritetaan n kertaa, joten algoritmin aikakompleksisuus on $\mathcal{O}(1) + n\mathcal{O}(1) = \mathcal{O}(n)$.

   b) Silmukat suorittavat sisimmän osan operaatioita, jotka vaativat ajan $\mathcal{O}(1)$ yhteensä

   $$1 + 2 + 3 + 4 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

   kertaa. Koska $n(n+1)/2 \in \mathcal{O}(n^2)$, on algoritmin aikakompleksisuus $\mathcal{O}(n^2)\mathcal{O}(1) = \mathcal{O}(n^2)$.

   c) kaksi sijoitusta alussa vaativat $\mathcal{O}(1)$ operaatiota ja while silmukan sisäosa vaatii $\mathcal{O}(1)$ operaatiota. Silmukka suoritetaan n kertaa joten algoritmin kompleksisuus on $\mathcal{O}(1) + n\mathcal{O}(1) = \mathcal{O}(n)$

   d) Ensimmäinen sijoitus vaatii $\mathcal{O}(1)$, a-kohdan perustein ensimmäinen silmukka vaatii $\mathcal{O}(m)$ ja vastaavin perustein toinen silmukka vaatii $\mathcal{O}(n^2)$. Algoritmin kompleksisuus on siis $\mathcal{O}(1) + \mathcal{O}(m) + \mathcal{O}(n^2) = \mathcal{O}(m + n^2)$.

Huom. $\mathcal{O}(m)$ ei ole $\mathcal{O}(n^2)$, koska n ja m ovat toisistaan riippumattomia. Voisihan hyvin olla, että esimerkiksi $m = n^3$, jolloin kompleksisuus olisi $\mathcal{O}(n^3)$.

5. Pinon päällimmäinen oikeanpuoleisimpana. Jonon ensimmäinen vasemmanpuolimmaisena.

   a) $S = (a, b)$

   b) $S = (a, a, b, b)$

   c) $Q = (b, c)$

   d) $S = (a, a)$, $Q = (a)$

6. Jonon operaatiot voidaan toteuttaa kahden pinon avulla. Tarkastellaan pinoja A ja B. Kaikki lisäykset jonoon tehdään pinon A huipulle, ja muuttuja *count* pitää kirjaa jonossa olevien alkioiden määrästä. Poistettaessa jonon ensimmäistä alkiota (pinon A pohjalta), voidaan toimia seuraavasti.

   - Siirretään pinosta A $count - 1$ alkiota pinoon B.
   - Palautetaan (ja poistetaan) pinon A ainoa alkio.
   - Siirretään pinon B kaikki alkiot takaisin pinoon A.

   Pseudokoodit löytyvät alta (Dequeue ja Engueue):

Enqueue(x)
  A.Push(x)
  count ← *count* + 1
Dequeue(x)
  count ← *count* − 1
  for $i \leftarrow 1$ **to** *count* **do**
    B.Push(A.Pop())
  end for
  value ← *A.pop*()
  for $i \leftarrow 1$ **to** *count* **do**
    A.Push(B.Pop())
  end for
  return *value*

Alkion lisäys kahdella pinolla toteutettuun jonoon tapahtuu ajassa $\mathcal{O}(1)$, mutta jonon ensimmäisen alkion poisto vaatii ajan $\mathcal{O}(n)$.

7. Tyypillisesti liukuvaa keskiarvoa laskettaessa sovelletaan tavallisen jonon sijasta kehäpuskuria (circular buffer). Se on jonon kaltainen tietorakenne, missä uuden alkion lisääminen jonoon poistaa automaattisesti jonon vanhimman alkion. Ts. kehäpuskurin tapauksessa enqueue-operaatio sekä lisää uuden alkion että poistaa ja palauttaa puskurin vanhimman alkion. Kehäpuskuria käytettäessä algoritmia voidaankin yksinkertaistaa siten, että MovingAverage:n toisen rivin dequeue-operaatio korvataan kolmannen rivin enqueue-operaatiolla ja kolmas rivi poistetaan.

Algorithm 1 Liukuvan keskiarvon laskenta. Q on jono, jossa on n edellistä syötettä. S on jonossa olevien alkioiden keskiarvo. Alussa jono Q alustetaan siten, että kaikki n alkiota ovat nollia.

ComputeMovingAverage(X,n)
  $Q \leftarrow$ new Queue(n)
  $R \leftarrow$ new Queue()
  $S \leftarrow 0$
  $x \leftarrow X$.head()
  while $x \neq$ null **do**
    $S \leftarrow$ MovingAverage(x.val, n, Q, S)
    R.enqueue(S)
    $x \leftarrow x$.next()
  end while
  return R

MovingAverage (x, n, Q, S)
  $x \leftarrow x/n$
  $S \leftarrow S + x - Q$.dequeue()
  Q.enqueue(x)
  return S

Esimerkki, kun $X = (1, 1, 2, 3, 3)$ ja $n = 2$. Tällöin ensimmäisellä kierroksella jono on alustettu $Q = (0, 0)$ ja $S = 0$ eli kun lasketaan Moving Average, saadaan

$$\begin{aligned} S = \text{MovingAverage}(1, 2, (0, 0), 0) &= S + \frac{1}{2} - \text{Q.dequeue()} \\ &= S + \frac{1}{2} - 0 = 0 + \frac{1}{2} - 0 = \frac{1}{2}. \end{aligned}$$

Lisäksi lisätään jonoon Q alkio $\frac{1}{2}$ eli nyt $Q = (0, \frac{1}{2})$. Lisätään tulosjonoon R alkio S eli $R = (\frac{1}{2})$.
Toisella kierroksella $Q = (0, \frac{1}{2})$ ja $S = \frac{1}{2}$ eli kun lasketaan Moving Average,

saadaan

$$\begin{aligned}S &= \text{MovingAverage}(1,2,(0,\frac{1}{2}),\frac{1}{2}) = S + \frac{1}{2} - \text{Q.dequeue}()\\ &= S + \frac{1}{2} - 0 = \frac{1}{2} + \frac{1}{2} - 0 = 1.\end{aligned}$$

Lisäksi lisätään jonoon Q alkio $\frac{1}{2}$ eli nyt $Q = (\frac{1}{2}, \frac{1}{2})$. Lisätään tulosjonoon R alkio S eli $R = (\frac{1}{2}, 1)$.
Kolmannella kierroksella $Q = (\frac{1}{2}, \frac{1}{2})$ ja $S = 1$ eli kun lasketaan Moving Average, saadaan

$$\begin{aligned}S &= \text{MovingAverage}(2,2,(\frac{1}{2},\frac{1}{2}),1) = S + \frac{2}{2} - \text{Q.dequeue}()\\ &= 1 + \frac{2}{2} - \frac{1}{2} = \frac{3}{2}.\end{aligned}$$

Lisäksi lisätään jonoon Q alkio $\frac{2}{2}$ eli nyt $Q = (\frac{1}{2}, 1)$. Lisätään tulosjonoon R alkio S eli $R = (\frac{1}{2}, 1, \frac{3}{2})$.
Neljännellä kierroksella $Q = (\frac{1}{2}, 1)$ ja $S = \frac{3}{2}$ eli kun lasketaan Moving Average, saadaan

$$\begin{aligned}S &= \text{MovingAverage}(3,2,(\frac{1}{2},1),\frac{3}{2}) = S + \frac{3}{2} - \text{Q.dequeue}()\\ &= \frac{3}{2} + \frac{3}{2} - \frac{1}{2} = \frac{5}{2}.\end{aligned}$$

Lisäksi lisätään jonoon Q alkio $\frac{3}{2}$ eli nyt $Q = (1, \frac{3}{2})$. Lisätään tulosjonoon R alkio S eli $R = (\frac{1}{2}, 1, \frac{3}{2}, \frac{5}{2})$.
Viimeisellä kierroksella $Q = (1, \frac{3}{2})$ ja $S = \frac{5}{2}$ eli kun lasketaan Moving Average, saadaan

$$\begin{aligned}S &= \text{MovingAverage}(3,2,(1,\frac{3}{2}),\frac{5}{2}) = S + \frac{3}{2} - \text{Q.dequeue}()\\ &= \frac{5}{2} + \frac{3}{2} - 1 = \frac{6}{2} = 3.\end{aligned}$$

Lisäksi lisätään jonoon Q alkio $\frac{3}{2}$ eli nyt $Q = (\frac{3}{2}, \frac{3}{2})$. Lisätään tulosjonoon R alkio S eli $R = (\frac{1}{2}, 1, \frac{3}{2}, \frac{5}{2}, 3)$.

Algorithm 2 Listoilla esitettyjen lukujen yhteenlasku

```
Sum(A, B)
8.  a ← A.head
    b ← B.head
    R ← uusi lista
    R.head ← uusi solmu
    r ← R.head
    c ← a.val + b.val
    r.val ← c mod 10
    c ← ⌊c/10⌋
    a ← a.next
    b ← b.next
    while a ≠ null and b ≠ null do
        r.next ← uusi solmu
        r ← r.next
        c ← c + a.val + b.val
        r.val ← c mod 10
        c ← ⌊c/10⌋
        a ← a.next
        b ← b.next
    end while
    if b ≠ null then
        a ← b
    end if
    while a ≠ null do
        r.next ← uusi solmu
        r ← r.next
        c ← c + a.val
        r.val ← c mod 10
        a ← a.next
        c ← ⌊c/10⌋
    end while
    if c > 0 then
        r.next ← uusi solmu
        r.next.val ← c mod 10
    end if
    return R
```