Sensorveiledning/løsningsforslag IN2010/INF2220 Algoritmer og datastrukturer H2018

Ragnhild Kobro Runde, Stein Michael Storleer, Ingrid Chieh Yu

Generell informasjon

Alle besvarelser rettes av to sensorer. Som en hjelp under sensuren legges det opp til at det settes poeng per oppgave i tråd med vektingen angitt i oppgavesettet. I tillegg kan det foretas en skjønnsmessig helhetsvurdering både av den enkelte oppgave og av besvarelsen som helhet. Når hver sensor er ferdig med sin vurdering, slås resultatene sammen. Ved større sprik i vurderingen, diskuterer de to sensorene seg frem til et felles resultat.

Oppgave 1: O-notasjon (4 poeng)

1 poeng på hver av oppgavene.

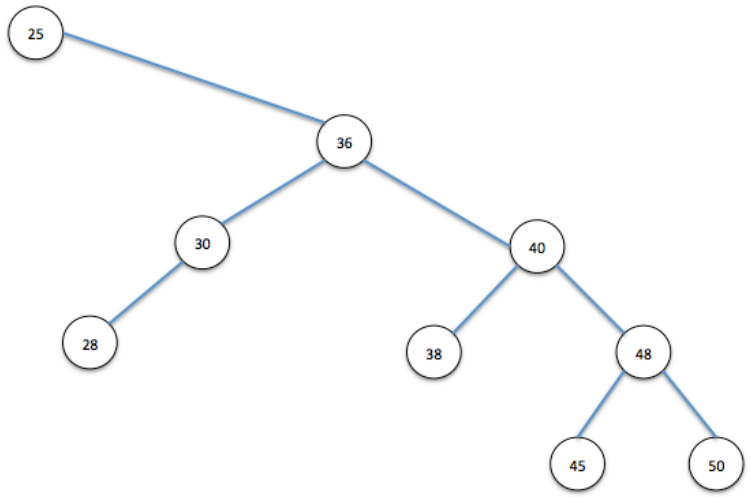
(i) Avhenger av hva som ligger i "gjør noe". Den angitte for-løkken går konstant antall ganger og er uavhengig av n, dvs O(1). Total tidskompleksitet for denne kodebiten blir da lik tidskompleksiteten til "gjør noe". Dette er et viktig aspekt ved O-notasjon, men noe de fleste sannsynligvis ikke vil ha tenkt på i forbindelse med denne oppgaven (selv om det er en studentoppgave).
0 poeng for O(8) eller å svare varianter av "O(n) for n = 8".
0,5 poeng for O(1).
1 poeng for O(1) med forbehold om innholdet i "gjør noe"

(ii) Denne bør være triviell. O(n^2).

(iii) Denne blir også O(n^2). Det spiller ingen rolle om i-løkken går til n eller 2*n siden det bare er en konstant forskjell. j-løkken går til i, og studentene bør ha sett en del eksempler på at siden i max kan være n kan også j max være n.

(iv) Denne er egentlig en kombinasjon av de to over. O(n^3). j-løkken går til i, dvs n. k-løkken går til i+j, dvs n+n=2n.

Oppgave 2: Binære søketreær (8 poeng)



a) Hvor mange kanter er det fra rota til dypeste bladnode/løvnode?
4
b) Hvor mange søskenpar finnes i treet?
3
c) Skriv i stigende orden verdiene til barnet/a til noden med verdi 30.
28
d) Er nodene med verdi 28 og 38 søsken?
nei
e) Hvor mange enebarn (noder uten søsken) finnes i treet?
3 ([25,] 28, 36). Godtar 2 dersom supplert med argument om at rota ikke er barn.
f) Hvor mange noder er løvnoder/bladnoder?
4
g) Skriv i stigende orden verdien(e) til nodene som har nøyaktig ett barn.
25, 30
h) Finnes det en eller flere noder uten forelder i treet?
ja, rota
i) Hva blir tekststrengen som definerer plasseringen til noden med verdi 38 i treet?
RRL 38 eller RRL
j) Hvor mange noder har strukturelt forskjellige subtrær?
4 (4 løvnoder har konforme subtrær, likeså 48: 9 - 5 = 4)
k) Hvilken verdi skal stå bak stien RRRL ?
45
l) Finnes det en node med stien L i treet?
nei
m) Hvilken bladnode ble satt inn først?
38
n) Skriv i stigende orden nodene med dybde 2.
30, 40
o) Hvor mange noder til må vi sette inn for å få et perfekt balansert tre?
22 (31 - 9 = 22)
p) Kan treet fargelegges som et rød-svart tre uten omstruktureringer?
nei

0,5 poeng for hvert riktig svar

Oppgave 3: Trær: writePathToNode (6 poeng)

Ikke trekk for korrekt *iterativ*løsning.

2 poeng for oppbygging av korrekt tekststreng
1 poeng for korrekt håndtering av hver av de 4 mulighetene:
n.v == v
n.v > v
n.v < v
v finnes ikke

```
void writePathToNode(Node n, int v, String pathSoFar) {
   if ( n == null )
      System.out.println("Verdien "+ v
                         +" finnes ikke i treet.");
   else  if ( n.v == v )
          System.out.println(pathSoFar + " " + v);
   else if ( n.v > v )
          writePathToNode(n.l, v, pathSoFar+"L");
   else // if ( n.v < v )
          writePathToNode(n.r, v, pathSoFar+"R");
}
```

eller

```
public void writePathToNode(Node n, int v, String pathSoFar) {
   if ( n != null ) {
       if ( n.v == v )
          System.out.println(pathSoFar + " " + v);
       else if (n.v > v && n.l != null)
            writePathToNode(n.l, v, pathSoFar+"L");
       else if (n.v < v && n.r != null)
            writePathToNode(n.r, v, pathSoFar+"R");
       else
            System.out.println("Verdien "+ v
                           +" finnes ikke i treet.");
   }
   else System.out.println("Treet er tomt!");
}
```

Noen kandidater har kommet i skade for å forveksle n.l (left) med n.v (value) for venstre. Dette ser man hvis det for eksempel skrives noe à la if (n.v != null) og lignende. Det skal ikke trekkes for dette.

Oppgave 4: Trær: removeLessThan (6 poeng)

```
Node removeLessThan(Node n, int value) {
      if (n == null) return null;

      if (value <= n.v) {
            // n skal ikke fjernes, heller ikke noe i
            // høyre subtre, men kanskje i venstre?
          n.l = removeLessThan(n.l, value);
          return n;
      } else {
            // n.v < value og n skal fjernes
            // sammen med hele venstre subtre
            // og kanskje noe i høyre subtre også
          return removeLessThan(n.r, value);
      }
    }
```

2 poeng for riktig håndtering av noden selv
2 poeng for riktig håndtering av hver av subtrærne
Trekk 1 poeng hvis feil med ulikhetene

Oppgave 5: Hashing (5 poeng)

Benytter åpen hashing fordi det ikke er plass til alle tallene hvis lukket hashing velges. Dette ville også vært det beste valget hvis fyllingsgraden > 70%:

0: 99, 33
1: 78, 12
2: 57
3: 91
4:
5:
6: 61
7: 18
8: 74, 19
9: 42, 20
10:

Hvis kandidaten har valgt lukket hashing med linær prøving skal tabellen se slik ut:

0: 99
1: 78
2: 57
3: 12
4: 33
5: 19
6: 61
7: 18
8: 74
9: 42
10: 20

Det er ikke plass til 91 (h(91) = 3).

a)

2 poeng for åpen hashing med begrunnelse.

Trekk 2 poeng for å velge lukket hashing uten begrunnelse.
Trekk 1,5 poeng for å velge lukket hashing med begrunnelse.
Trekk 1 poeng for for åpen hashing uten begrunnelse.

b)

3 poeng

Trekk 1 poeng for hvert feilplassert tall.

Oppgave 6: Mapping-metoder (6 poeng)

```
boolean put (Element e) {
    int i  = e.k % hashTable.length;
    int firsti = i;

    while (hashTable[i] != null) {
        if ( ((Element)hashTable[i]).k == e.k ) {
              // element med samme nøkkel lå der fra før
              // skifter ut med ny e
             hashTable[i] = e;
        }
        i = (i+1) % hashTable.length;
        // sjekker at vi nå ikke har kommet helt rundt:
        if ( i == firsti) return false;
    }
     // hashTable[i] == null
    hashTable[i] = e;
    return true;
}
```

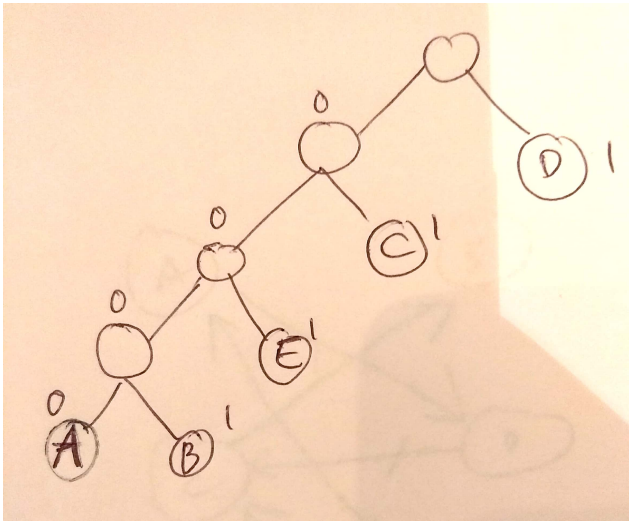
Typekonverteringen fra Object til Element er ikke påkrevd.

1 poeng for riktig lineær prøving
1 poeng for riktig while-betingelse
1 poeng for riktig innsetting
1 poeng for å bytte ut e hvis e.k lå i tabellen fra før
1 poeng for returtype
1 poeng for å sjekke at vi ikke får evig løkke

Oppgave 7: Heap (4 poeng)

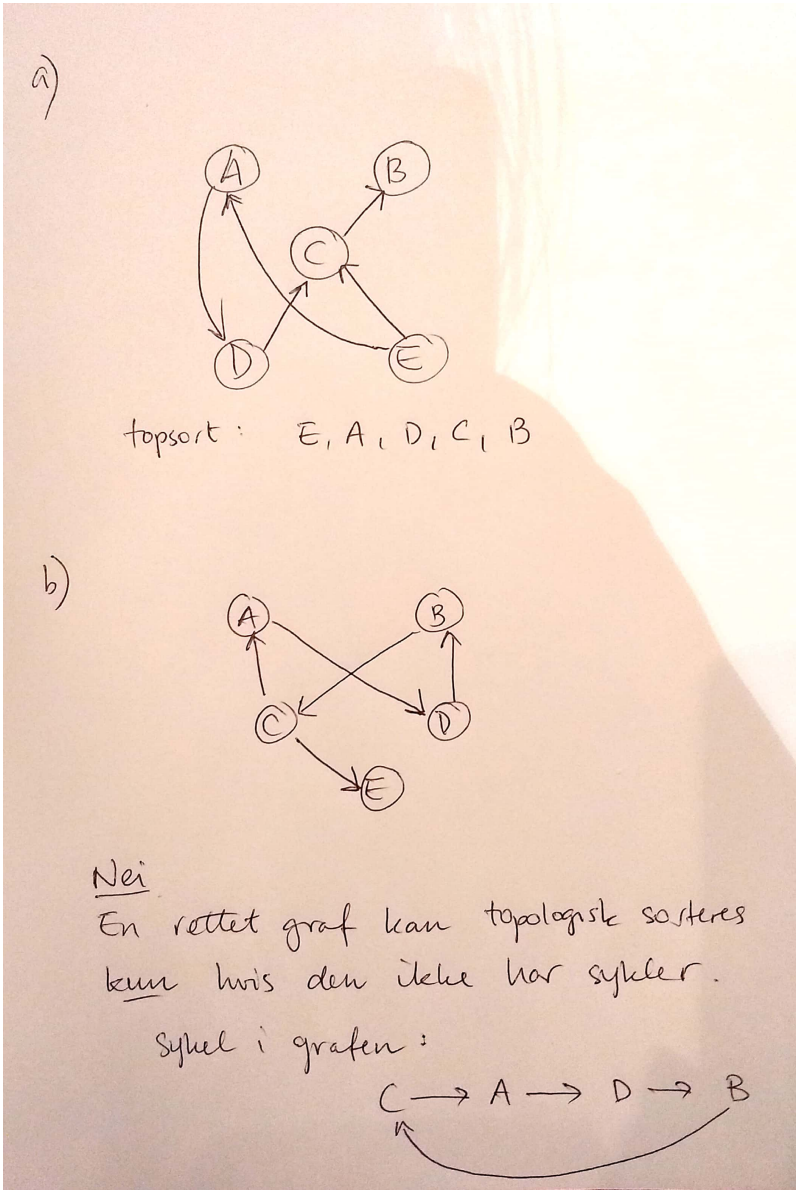
Nei. treet tilfredstiller strukturkravet til heap, men ikke ordningskravet. 9 er rotnoden som verken er det minst eller største tallet i arrayet. Så denne er verken en min heap eller en max heap.

Oppgave 8: Huffman (4 poeng)



A: 0000
B: 0001
C: 01
D: 1
E: 001

Oppgave 9: Topologisk sortering (4 poeng)



Oppgave 10: Dijkstra (10 poeng)

6 poeng for oppgave a, 4 poeng for oppgave b.

oppgave a)
Det viktigste å forstå med Dijkstra er:
- induktivt og grådig. Vises ved rekkefølge nodene ble gjort kjent.
- relaxation av avstand og oppdatering av sti. Vises i tabellen.

List nodene i den rekkefølge de ble gjort kjent: A B C G D E F H. max 3 poeng

max 3 poeng for tabellen:

Node	Kjent	Avstand	Sti
A	Y	0	-
B	Y	2	A
C	Y	5	A
D	Y	10	B
E	Y	~~24~~ 13	~~B~~D
F	Y	~~20~~ 13	~~AG~~
G	Y	9	C
H	Y	~~21 17~~14	~~G E~~ F

-trekk 1 poeng hvis tabellen er helt riktig når stegene ikke vises.

oppgave b)
Nei. For å bregne den riktig må vi bryte invarianten. Dijkstras algoritmen endrer ikke en node som er kjent og antar at en bedre sti fra startnoden til en behandlet node v ikke vil bli funnet i fremtiden. Vi må derimot bryte invarianten siden etter at v er kjent (f.eks E med avstand 7 fra A-B-E), kan det finnes en negativ kant til v som gir en sti med kortere vei (f.eks E med avstand 3 fra A-B-D-E).

max 1 poeng hvis svaret er nei uten begrunnelse f.eks "dijkstra fungerer ikke på slike grafer".
max 3 poeng for b) hvis studenten viser konkrete feil begrenset til selve grafen.
max 4 poeng for b) hvis studenten viser til invarianten også.

Oppgave 11: Urettet graf (10 poeng)

En urettet graf er et tre hvis den har følgende **2 graf**-egenskaper:
1. ingen ***sykler***.
2. grafen er ***sammenhengende***.

Studentene skal kunne sjekke disse når vi har adjacency list som datastruktur.
- maks 6 poeng med riktig implementasjon av å finne *sykler*.
- maks 4 poeng med riktig implementasjon for å sjekke at grafen er *sammenhengende*.
- NB. Alle urettet kanter forekommer i 2 lister: adj[v].add(w) og adj[w].add(v);) Riktig sjekk av sykler må ha en sjekk f.eks 'elseif(i != parent)'. Det er ikke nok med kun å sjekke om noden er besøkt. 4 av 6 poeng hvis de finner sykler uten denne sjekken.

Løsningsforslaget starter traverseringen fra node 0, men kan også starte med andre noder.

For en urettet graf kan vi enten bruke BFS eller DFS:

```
Class Graph
{
  private int V;   // No. of vertices
  private LinkedList<Integer> adj[]; //Adjacency List

  // Constructor
  Graph(int v)
  {
    V = v;
    adj = new LinkedList[v];
    for(int i=0; i<v; ++i)
      adj[i] = new LinkedList();
  }
```

```
// Function to add an edge into the graph
  void addEdge(int v,int w)
  {
    adj[v].add(w);
    adj[w].add(v);
  }

  // A recursive function that uses visited[] and parent
  // to detect cycle in subgraph reachable from vertex v.
  Boolean isCyclic(int v, Boolean visited[], int parent) {

    visited[v] = true;
    Integer i;

    // Recur for all the vertices adjacent to this vertex
    Iterator<Integer> it = adj[v].iterator();
    while(it.hasNext())
    {
      i = it.next();
      if(!visited[i]) {
          if(isCyclic(i, visited, v))
            return true;
      }

      // If an adjacent is visited and not parent of
      // current vertex, then there is a cycle.
      else if(i != parent)
        return true;
    }
    return false;
  }

  // Returns true if the graph is a tree, else false.
  Boolean isTree()
  {
    // Mark all the vertices as not visited

    Boolean visited[] = newBoolean[V];
    for(int i = 0; i < V; i++)
      visited[i] = false;

    // isCyclic returns true if graph reachable from a vertex (e.g. vertex 0 )
    // contains cycles
    if(isCyclic(0, visited, -1))
      return false;

    // return false if there is a vertex which is not reachable from 0

    for(int u = 0; u < V; u++)
      if(!visited[u])
        returnf alse;

    return true;
  }
}
```

Oppgave 12: Emner (14 poeng)

Denne løses ved hjelp av topologisk sortering. Videre er det viktig å forstå at dersom man får inngrad = 0 for et kurs w fordi kurs v er blitt behandlet nå, kan ikke kurs w tas før tidligst semester etter. Dvs. semester-telleren må økes før w kan tas. Holder med en standard Java FIFO kø. Må passe på antall poll-operasjoner i form av for-løkker i hver iterasjon (while løkke).

- maks 9 poeng hvis de har implementert topologisk sortering. Men ikke modifikasjon med antall semester slik oppgaven ber om.
- maks 5 poeng hvis de kun har fått med seg datastruktur som trengs, har initiert inngraden til noder, har initiert køen, men uten algoritme/topologisk sortering

Topologisk-sortering med tilhørende modifikasjon er nok til full score selv om løsningen ikke er garantert optimal.

```
int numberOfSemesters(){
    int indegree[] = new int[V];

    for(int  i = 0; i < V; i++) {
      LinkedList<Integer> temp = (LinkedList<Integer>) adj[i];
      for(int node : temp) {
        indegree[node]++;
      }
    }

    Queue<Integer> q = newLinkedList<Integer>();
    for(int i = 0;i < V; i++) {
      if(indegree[i]==0)
        q.add(i);
    }

    int semesters = 0;

    while(!q.isEmpty()){
      int size = q.size();
      for (int i = 0; 3>i && i< size; i++){ //max 3 kurs per semester. Her må vi ikke ha q.size()
        int v = q.poll();
        for(int node : adj[v]) {
              if(--indegree[node] == 0)   q.add(node);  }//end for
      }//end for
      semesters ++;
    }
  return semesters;
}
```

Oppgave 13: Shell-sortering (6 poeng)

4 poeng for a, 2 poeng for b.

a) Shell-sortering er ikke stabil. Hvis feks a[2] og a[8] begge er mindre enn a[1] vil sorteringen av a[1] og a[8] ødelegge rekkefølgen på a[2] og a[8]. Poenget her er at selv om innstikksorteringen i seg selv er stabil, gjør sorteringen av del-arrayer at den ikke er stabil på tvers av del-arrayene. Studenter som har svart at shell-sortering er stabil fordi den bruker innstikksortering som er stabil, gis ett poeng.

b) Innstikk-sortering er O(n^2), som betyr at Shell-sort har bedre worst-case kjøretid. For sortert input fungerer Shell-sort som innstikksortering, dvs det foretas ingen ombyttinger. Men det foretas

flere sjekkinger for å konstatere dette. For å få full score er det tilstrekkelig å si at innstikksortering er O(n^2) og at Shell-sort dermed er bedre.

Oppgave 14: Sortering (7 poeng)

Dette er en tilpasning av studentoppgave 30. Poenget er selvfølgelig at det skal brukes bøttesortering og ikke compareTo.

Studenter som bruker bøttesortering uten begrunnelse får maksimalt 6 poeng. Bruk av andre algoritmer på en fornuftig måte og en fornuftig begrunnelse kan gis inntil 5 poeng.

Å kun angi bruk av bøttesortering uten å forklare hvordan/hvorfor gir 2 poeng. For full score forventes en kort begrunnelse (heltall fra 1 til 6, dvs 6 bøtter) sammen med tekst eller kode som viser forståelse for hvordan selve bøttesorteringen fungerer.

Tilpasning av algoritme 9.2 fra læreboken:

```
let B be an array of 6 lists, each of which is initially empty
for each elev x in elever do
   let k be x.karakter
   remove x from elever and insert it at the end of bucket (list) B[k]
for i from 6 down to 1 do
   for each elev x in list B[i] do
      remove x from B[i] and insert it at the end of the output array (elever)
```

Studentene forventes ikke å kunne denne ordrett, men de forventes å kunne prinsippene/hovedtrekkene, dvs:
1) Gjennomløp av elever-arrayen for å sette inn i riktig bøtte ved hjelp av karakter-attributtet.
2) Gjennomløp av bøttene (i motsatt rekkefølge!) for å få den ferdige-sorterte arrayen. Trekk 2 poeng hvis det ikke sies noe om dette.

Oppgave 15: NP-kompletthet (6 poeng)

To poeng for a, fire poeng for b.

Dette er ment som en krevende avslutningsoppgave, og det gis her ikke "trøstepoeng" for å skrive generelt om NP uten å svare riktig på noen av de konkrete spørsmålene. (Det hadde da vært spurt eksplisitt om å forklare NP.)

a) Reduksjon fra A til B, som vil bety at A kan løses ved hjelp av B og dermed har en algoritme med polynomisk kjøretid dersom B har det. (Gitt at reduksjonen er polynomisk - det er ikke krav om at studentene skal angi det for å få full score.)

b) Reduksjon av Hamiltonsk løkke hvor kanter = trives sammen. Studenter som har svart feil eller ikke svart på a, kan likevel få fire poeng her. Poenget her er å se sammenhengen mellom problemet i oppgaven og Hamiltonsk løkke. Godt beskrevet, kan det gi full score her selv om reduksjonen går feil vei. 1 poeng for å angi ("gjette") at dette har med Hamiltonsk løkke å gjøre uten nærmere angivelse. Det er mer krevende å benytte andre problemer (selv om det i teorien er mulig), så her kreves det en begrunnelse for å kunne få poeng i det hele tatt. Flere har brukt TSP - med begrunnelse gir det full score, også om begrunnelsen viser at det egentlig er Hamiltonsk løkke de tenker på (dvs uten vekter

på kantene). De som har prøvd seg på Vertex-cover er inne på noe, men kommer ikke helt i mål og vil typisk få to poeng.