IN2010 – Sensorveiledning

1 Diverse oppgaver med kort svar (automatisk rettet)

1(a) Kanter i et spenntre

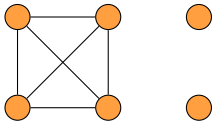
Løsningsforslag: 11.

G er sammenhengende og har 12 noder.

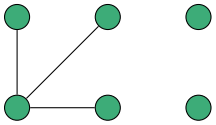
Sensorveiledning:

1(b) Kanter i en spennskog

Løsningsforslag: 3



G



F

Sensorveiledning:

1(c) Minimale spenntrær

Løsningsforslag: Borůvka, Kruskal og Prim.

DFS og BFS finner spenntrær men ikke minimale spenntrær.

Sensorveiledning: Det skal gis 1p hvis noen har kun én riktig, siden man har gjort 3 riktige valg (f.eks prim, ikke DFS og ikke BFS) og 2 gale (ikke Kruskal og ikke Borůvka). 3-2 = 1.

1(d) FEM

Løsningsforslag: Ja, fordi FEM er i P.

Vi vet at $P \subseteq NP$. De andre alternativene er feil/gir ikke mening.

Sensorveiledning:

1(e) SUDOKU-SJEKKER

Løsningsforslag: SUDOKU-SJEKKER kan løses i konstant tid fordi Aloku kun kan løse endelig mange brett.

Man kan sjekke i konstant tid om brettet er av orden 100 eller mindre (avbryt søket på et bestemt tidspunkt hvis input er for stor). Hvis brettet er "lite" slår man opp i en endelig tabell over alle brett med kun en unik løsning og svarer ja hvis brettet er i tabellen.

Siden vi har antatt at $P \neq NP$, vet vi at SUDOKU-SJEKKER ikke er NP-komplett.

Sensorveiledning:

1(f) Kodeanalyse

Løsningsforslag: $O(n^5)$.

Sensorveiledning: Sjekkes manuelt. n^5 gir 1p. Her skal kandidaten ha poeng hvis svaret er riktig, uavhengig av syntax-feil som ikke blir plukket opp av automatikken.

1(g) 5-CLIQUE

Løsningsforslag: 5-CLIQUE er i P.

Følgende pseudokode forklarer en brute-force algoritme som løser 5-CLIQUE i polynomiell tid. Koden er meget lik den som ble gitt i forrige deloppgave.

```
for all s in V:
  for all t in V:
    for all u in V:
      for all v in V:
        for all w in V:
          if st, su, sv, sw, tu, tv, tw, uv, uw, vw in E:
            return true
return false
```

Sensorveiledning:

1(h) Korteste avstander i grafer

Løsningsforslag:

	Bredde-først-søk	Topologisk Sortering	Dijkstra	Bellman-Ford
Uvektet	•			
Vektet DAG		•		
Ingen negative kanter			•	
Ingen negative sykler				•

Sensorveiledning:

1(i) Huffmankode

Løsningsforslag: GATTACA.

Sensorveiledning: Slurvefeil gir 0.5p. Sjekkes manuelt for feil i automatikken. For eksempel skal det gis 1p for lowercase og mellomrom/punktum.

1(j) Lukket hashing

Løsningsforslag: 1.

Setter inn 17, 98, 59, 32, 40.

1. 17 mod 5 = 2; 17 på indeks 2
2. 98 mod 5 = 3; 98 på indeks 3
3. 59 mod 5 = 4; 59 på indeks 4
4. 32 mod 5 = 2; 2 opptatt, 3 opptatt, 4 (siste indeks) opptatt, 0 ledig, 32 på indeks 0
5. 40 mod 5 = 0; 0 opptatt, 1 ledig, 40 på indeks 1

Sensorveiledning:

---

2 Spilldesign

2(a) Representasjon

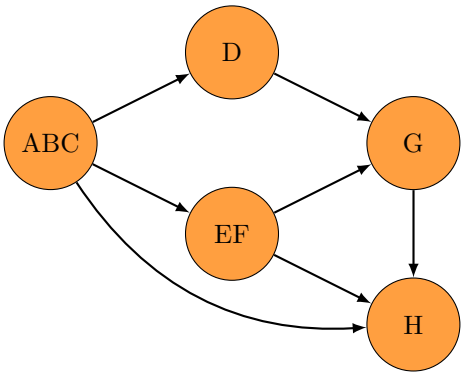
Løsningsforslag:

1. Kantene representerer overganger mellom rom. Det går en rettet kant fra u til v om v er på panelet til rom u.
2. Antall rom på panelet til noden.
3. 0, fordi et skattkammer ikke kan nå noen andre rom.
4. 1, kun G er på panelet til D

Sensorveiledning: Det gis 0,5p per riktig. Trekker hvis kandidaten kun snakker om veier og at noder kan nås. Trekkes for uklare svar.

2(b) Komponentgraf

Løsningsforslag:



Sensorveiledning: Trekker 1p for en manglende kant. Trekker 2p for feil noder. Kan gi 0.5p totalt hvis kandidaten har funnet noe riktig, men ellers feil.

2(c) Skattkammer I

Løsningsforslag:

1. $|C_k| = 1$. C_k inneholder kun k siden k ikke kan nå noen andre rom.
2. 0 fordi C_k kun består av k som har utgrad 0

Sensorveiledning: Trekker 0.5p for manglende begrunnelser. Hvis begrunnelsen står tydelig på et annet punkt trekkes det ikke.

2(d) Skattkammer II

Løsningsforslag: Hvis k er et skattkammer, må det finnes en sti fra alle andre noder til k (fra krav 2). De andre rommene kan derfor ikke være skattkammer, siden de kan nå minst ett annet rom (her brytes krav 1).

Sensorveiledning: Her finnes det flere mulige svar. Essensen skal være på plass. Trekkes for uklarheter og feil. Det kan trekkes hvis svaret er veldig langt.

2(e) Startrom

Løsningsforslag:

1. La $t \in C_s$. Da kan t nå s. Siden s kan nå alle rom (siden s er et startrom) kan t nå alle rom (enten direkte, eller via s). Derfor er t også et startrom.

2. Anta for motsigelse at startrommet t ikke er i C_s. Siden både s og t er startrom kan begge nå hverandre. Dette motsier at t ikke er i C_s.

Sensorveiledning: 2p per punkt. Trekkes for uklarheter som feil.

2(f) Algoritmedesign

Løsningsforslag:

Algoritmen som er mest rett frem er følgende:

For hver node i $v \in V$, bruk DFS/BFS for å sjekke om man kan nå alle noder fra v. Det skal være nøyaktig 3 slike noder.

Deretter snur vi alle kantene i grafen og gjør følgende for alle noder $v \in G_R$ (den reverserte grafen):

Hvis inngraden er null, bruk DFS/BFS til å sjekke om noden kan nå alle andre noder i G_R. Det skal være nøyaktig 1 slik node.

DFS/BFS fra en node tar $O(|E|)$ tid, i motsetning til f.eks $DFS/BFS - full$ som tar $O(|V| + |E|)$ tid. Kjøretiden blir $O(|V| * |E|)$ siden vi gjør DFS/BFS fra hver node.

Det finnes imidlertid en algoritme som kjører i lineær tid som forklart under.

Grov skisse:

0. Input er en rettet graf G
1. Finn/regn ut C, komponentgrafen til G

   Her skal det sies at algoritmen ikke finner kantene i C, se den mer detaljerte beskrivelsen

2a Bekreft at C inneholder nøyaktig én node med inngrad 0. Kall denne C_s

2b Bekreft at C inneholder nøyaktig én node med utgrad 0. Kall denne C_k

3a Bekreft at C_s inneholder nøyaktig 3 noder fra G

3b Bekreft at C_k inneholder nøyaktig 1 node fra G

Finere beskrivelse:

0. Input: $G = \langle V, E \rangle$ med inn- og utkanter for hver node i V.

   E består av par av noder fra V.

1a Beregn V_C, nodene i komponentgrafen til G.

```
V_C = SCC(G) // SCC: Strongly connected components
/* each node c ∈ V_C is a set of nodes from G
   the algorithm for finding scc's is known
   from the course and runs in O(|V|+|E|) time */
```

1b Marker hver node i V med nodens tilhørende komponent

```
for each c ∈ V_C:
  for each v ∈ c:
    v.component = C
// running time: O(|V|), as each node is tagged once
```

2 Sjekk om nodene i V_C har inn/utgrad mer enn 0

```
for each ⟨u,v⟩ ∈ E:
  cu = u.component
  cv = v.component
  if cu ≠ cv:
    cu.hasOutEdges = true
    cv.hasInEdges  = true
// running time: O(|E|), since we loop through E once
```

3 Finn noden(e) i V_C som har inngrad 0. Returner *false* hvis noden(e) ikke inneholder nøyaktig 3 noder fra G.

Finn noden(e) i V_C som har utgrad 0. Returner *false* hvis noden(e) ikke inneholder nøyaktig 1 noder fra G.

```
for each c ∈ V_C:
  if c.hasInEdges == false:
    zeroIns += 1
    if size(c) ≠ 3:
      return false
  if c.hasOutEdges == false:
    zeroOuts += 1
    if size(c) ≠ 1:
      return false
if (zeroIns == 1 and zeroOuts == 1):
  return true
else:
  return false
// running time: O(|V|), as |V_C| = O(|V|)
```

Den totale kjøretiden til algoritmen er $O(|V| + |E|)$. Beskrivelsen inneholder ikke oppretting av hjelpevariabler osv.

Korrektheten kommer av at komponentgrafen er en DAG (en rettet asyklisk graf). Hvis G ikke er sammenhengende, vil flere enn én node i C ha inngrad 0 og utgrad 0. Hvis nøyaktig én node i C har inngrad 0, må denne kunne nå alle andre noder i C. Tilsvarende gjelder hvis nøyaktig én node har utgrad 0. Da må alle noder i C kunne nå denne.

Sensorveiledning: Her gis det inntil 12p for versjoner av algoritmen som er mest rett frem og ikke ser på SCC. Det skal gis maksimalt 8p for varianter av den naive algoritmen som sjekker skattkammer feil, altså der det ikke sjekkes om alle noder kan nå skattkammeret.

2(g) Analyse

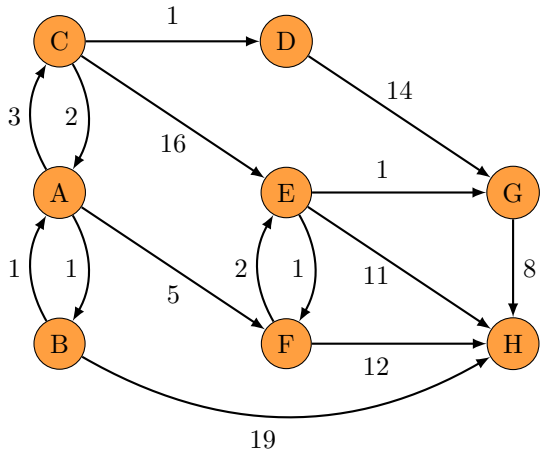
Løsningsforslag: $O(|V| + |E|)$

Se forrige deloppgave.

Sensorveiledning: Svaret skal stå til algoritmen som ble gitt på forrige deloppgave. For kandidater som har den naive algoritmen, trekkes det 0.5p hvis konklusjonen er at algoritmen kjører i $O(|V| * (|V| + |E|))$ tid, hvis de bruker DFS (og ikke DFS-full). DFS kjører i $O(|E|)$ tid, siden vi kun ser på nodene som kan nåes fra startnoden. Trekkes for bruk av n uten å forklare hva n er.

2(h) Minimal gjennomføringstid

Løsningsforslag: ∞, 20,17,16



A	B	C	D	E	F	G	H
0	∞	∞	∞	∞	∞	∞	∞
	1	3	4	19	5	18	20
				7		8	17
							16

Sensorveiledning: Her hadde mange misforstått formatet vi ønsket svaret på. Let etter lista til H. Det gis trekk hvis lista inneholder verdier som er høyere enn forrige estimat. Trekker ikke for kandidater som har listet opp estimater til andre noder

eller som oppgir stiene, så lenge de har kommet frem til riktig svar. Trekkes for slurvefeil. Fornuftige repetisjoner trenger man ikke trekke for.

2(i) Ukomprimerbare verdener

Løsningsforslag: Vi vet at C må være en DAG (en rettet asyklisk graf), som vil si at hvis G er ukomprimerbar så må G også være en DAG. Dette tillater oss å bruke en topologisk sortering av G for å finne korteste avstander i lineær tid.

Sensorveiledning: Her kan må få 2p for å ha skrevet at G er en DAG. Man trenger ikke forklare hvorfor C er en DAG, dette er kjent fra pensum. Man kan få 2p for topologisk sortering uten begrunnelse.

3 Prioritetskø og heap

3(a) PQ bruker utplukkssortering

Løsningsforslag: 17,43,98,11,43,56

Ved utplukkssortering blir elementene lagt inn i PQ uten at de ordnes. add blir $O(1)$. Da må removeMin hver gang lete gjennom hele arrayen, $O(n)$. Derfor er riktig svar her samme rekkefølge som tallene ble satt inn. Omvendt rekkefølge (som vi kan få hvis vi bruker en stack implementert som lenkeliste) er ikke aktuelt her siden den interne strukturen er en array.

Sensorveiledning: Pga uklarheter gis det 1p for besvart, 2p for riktig svar og 0p for ubesvart. Slurvefeil trekkes 0.5p.

3(b) PQ bruker innstikkssortering

Løsningsforslag: 11,17,43,43,56,98 eller 98,56,43,43,17,11

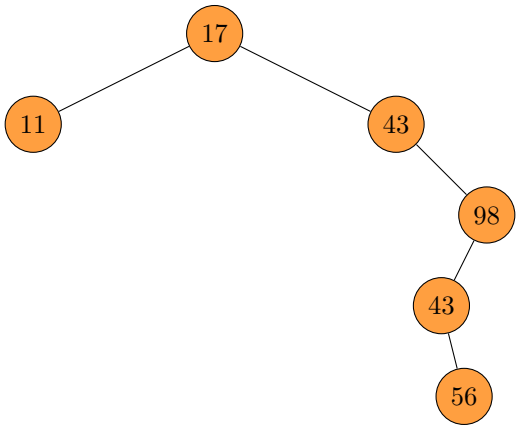
Ved innstikksortering vil add legge elementene i stigende orden, med minste element først, eventuelt sist, slik at removeMin blir $O(1)$.

Sensorveiledning: Slurvefeil trekkes 0.5p.

3(c) PQ bruker BST

Løsningsforslag: 17,11,43,98,43,56

Merk at oppgaven sier at like verdier skal til høyre. Det blir et ganske skjeivt tre (høyde 4):

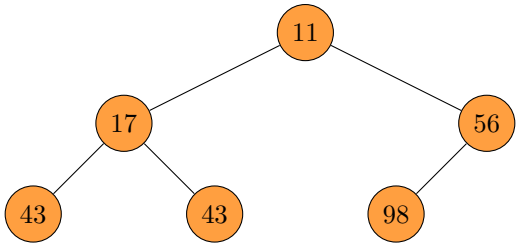


Sensorveiledning: Slurvefeil trekkes 0.5p.

3(d) PQ bruker heapsortering

Løsningsforslag: 11,17,56,43,43,98

add må legge elementet inn, slik at vi hele tida har en minHeap (minste element alltid på indeks 0). Etter de seks elementene er satt inn får vi følgende heap tegnet som en binærtre:



Sensorveiledning: Slurvefeil trekkes 0.5p.

3(e) Krav til en heap

Løsningsforslag:

Strukturkrav: En (binær) heap er et komplett binærtre. På dypeste nivå er alle nodene til venstre. Ingen node har en nullpeker til venstre for seg. Når nodene ligger i en array (uten 'hull') er dette kravet tilfredsstilt.

Ordningskrav: En node har en verdi som er større enn eller lik verdien til nodens mor/far/forelder.

Eventuelt: En node er mindre enn eller lik verdien til barna og varianter av dette.

Sensorveiledning: Kandidaten må ha begge kravene på plass. Trekk 0.5p for småfeil.

3(f) heapSort-forklar

Løsningsforslag: Fordi nodene på indeks (n/2 + 1) og utover ikke kan ha barn. De er løvnoder. De ligger allerede riktig (i forhold til sine ikke-eksisternede barn) når vi gjør downheap.

Bevis: Når rota ligger på indeks 0 vil venstre barn til (n/2 + 1) ligge på indeks 2*(n/2 +1) +1 = n + 2 + 1 = n+3. Dette er utenfor arrayen/heapen. Vi ser av dette at indeks n/2 også må være en løvnode, slik at løkka kunne ha startet med indeks (n/2 - 1).

Sensorveiledning: Det gis poeng hvis kandidaten viser forståelse for hvorfor dette fungerer.

3(g) heapSort-kode

Løsningsforslag:

```
// Tar med omgivelser for testformål

public class HeapSort {

    public static void main(String []args){

        int [] a = new int[11];
        a[0] = 1;
        a[1] = 4;
        a[2] = 3;
        a[3] = 2;
        a[4] = 5;
        a[5] = 6;
        a[6] = 13;
        a[7] = 12;
        a[8] = 0;
        a[9] = -6;
```

```
        a[10] = 555;

        for(int i = 0; i < a.length; i++)
            System.out.println(a[i]);
        a = heapSort(a);
        for(int i = 0; i < a.length; i++)
            System.out.println(a[i]);

    }

    // *** Oppgave 3(g) ****

    //static ikke nødvendig. Er med for kompileringens skyld
    static int[] heapSort(int [] A) {

        int n = A.length;
        for(int i = n/2; i >= 0; i--)
            downHeap(A,i,n-1);

        for(int i = 0; i < A.length; i++)
            System.out.println(A[i]);

        System.out.println();
        for(int i = n-1; i > 0; i--){
            int temp = A[i];
            A[i] = A[0];
            A[0] = temp;
            downHeap(A, 0, i-1);
        }

        return A;
    }
```

Sensorveiledning: Hvis besvarelsen er skrevet uten kode gis det maksimalt 2p på denne oppgaven. Det trekkes ikke hvis kandidaten har kalt på swap uten å skrive koden som swapper. Trekkes for småfeil.

3(h) downHeap-kode

Løsningsforslag:

```
    // *** Oppgave 3(h) ****

    static void downHeap(int [] A, int i, int n) {

        int left = 2*i+1;  // indeks til eventuelt venstrebarn
        int right = 2*i+2; // indeks til eventuelt høyrebarn
        boolean rightExists = ( right <= n );

        // i er løvnode hvis left >= n. Da er algoritmen ferdig.
        if ( left < n ){
            if ( A[i] < A[left] || rightExists && A[i] < A[right] ) {
                // Bytt verdiene i A[i] med største barn
                if ( ! rightExists || A[left] > A[right] ) {
                    int temp = A[i];
                    A[i] = A[left];
                    A[left] = temp;
                    downHeap(A, left, n);
                }
                else {
                    // A[right] >= A[left]
```

```
                    int temp = A[i];
                    A[i] = A[right];
                    A[right] = temp;
                    downHeap(A, right, n);
                }
            }
        }
    }
}
```

Sensorveiledning: Hvis besvarelsen er skrevet uten kode gis det maksimalt 2p på denne oppgaven. Det gis maksimalt 10p hvis kandidaten ikke sjekker om barna finnes.
