OMTENTAMEN I
DATASTRUKTURER OCH ALGORITMER DVG B03

150609 kl. 14:15 – 19:15

Ansvarig Lärare: Donald F. Ross

Hjälpmedel: Inga. Algoritmerna finns i de respektive uppgifterna eller i bilogarna.

***** OBS *****

Betygsgräns:	Kurs:	Max 60p, Med beröm godkänd 50p, Icke utan beröm godkänd 40p, Godkänd 30p (varav minimum 20p från tentan, 10p från labbarna)
	Tenta:	Max 40p, Med beröm godkänd 34p, Icke utan beröm godkänd 27p, Godkänd 20p
	Labbarna:	Max 20p, Med beröm godkänd 18p, Icke utan beröm godkänd 14p, Godkänd 10p

SKRIV TYDLIGT – LÄS UPPGIFTERNA NOGGRANT

Ange alla antaganden.

Studenter som har läst kursen from HT2014 ska svara på samtliga uppgifter.

Studenter som har läst kursen from HT2006 tom HT2013 ska välja uppgifter värt 30 poäng.

Studenter som har läst kursen tom HT2005 ska svara på samtliga uppgifter.

---

(1) Ge en kortfattad definition av följande begrepp ((a)-(j)).

(a) Mängd
(b) Sekvens
(c) Träd
(d) Binärt träd
(e) AVL-träd
(f) Graf
(g) Enkel cykel i en graf
(h) Artikuleringspunkt (articulation point)
(i) Prestande (performance)
(j) Big-oh (O(n))

Totalt 5p

(2) Rekursion

Beskriv **ingående** varför rekursion är så pass viktigt inom datastrukturer och algoritmer. Använd gärna **exempel.** Ge flera exempel där man använder sig av rekursion.

5p

(3) Heap

(a) Läs koden till heap operationer (**se Bilaga A**), Svara på följande uppgifter:-

i. Vad är parameter ”i” i **Heapify**(A, i)?
ii. Skriv (pseudo) kod till funktionen **Right**(i)
iii. I **Heapify**, vad gör koden nedan? Vad betyder koden?

```
if l <= A.size and A[l] > A[i] then largest = l else largest = i
if r <= A.size and A[r] > A[largest] then largest = r
```

iv. I **Heapify**, vad gör koden nedan? Vad betyder koden?

```
if largest != i then
    swap(A[i], A[largest])
    Heapify(A, largest)
end if
```

v. Varför skriver man **A.size/2** i funktion **Build**?
vi. Hur fungerar funktion **Remove**?

3p

(b) Givit heapen: **72, 66, 18, 44, 46, 9, 11, 33, 27, 15**
Förklara **stegvis** hur funktionen **Add** fungerar när man lägger till värdet **68** i heapen.
Rita en bild av den **trädrepresentationen** av heapen **vid varje steg**.

2p

Totalt 5p

(4) Binärt Träd

(a) Ge en rekursiv definition av ett binärt träd. 1p

(b) Utifrån din definition i (a) skriv pseudokod till en **rekursiv version** av den lägga-till funktion (add) för ett binärt träd. Anta att dubbletter inte tillåtas.

2p

(c) Att ta bort ett element från ett binärt träd kan reduceras till ett annat problem, nämligen att ta bort roten från ett binärt träd. Då finns det fyra möjligheter. Vilka är dessa? **Diskutera** utförligt vad händer i varje fall.

2p

Totalt 5p

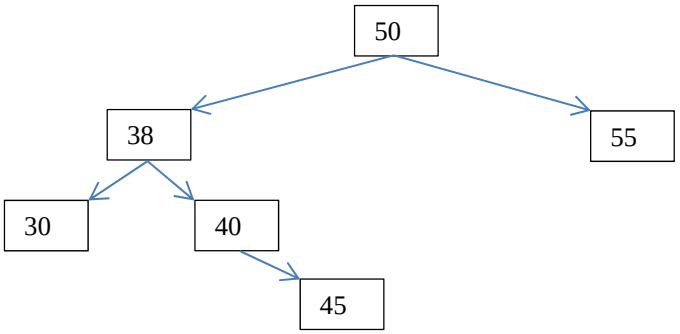
(5) AVL-Träd

Visa hur Du skulle ta fram de 4 rotationsfunktionerna från första principer.

2p

Förklara vad ”balansfaktorn” (balance factor) är och använd denna för att ta fram balansfunktionen från första principer. **Skriv balansfunktionen i pseudokod.** Tillämpa Din pseudokod på trädet nedan

3p



(6) Rekursiva funktioner

Skriv pseudokod till **två rekursiva funktioner** (alltså en funktion plus en hjälpfunktion) för att räkna fram antalet kanter (”edges”) i en graf. Ange alla antagande.

5p

(7) Prim

Tillämpa **den givna Prims algoritm (nedan)** på **den oriktade grafen**,

(a-20-b, a-36-c, a-34-d, b-22-c, b-24-e, c-28-d, c-30-e, c-38-f, d-26-f, e-36-f).

Prim's Algoritm **-- antagande: att det finns en kostnadsmatrix C**

```
Prim ( node v) -- v is the start node

{   U = {v}; for i in (V-U) { low-cost[i] = C[v,i]; closest[i] = v; }

    while (!is_empty (V-U) ) {
        i = first(V-U); min = low-cost[i]; k = i;
         for j in (V-U-k) if (low-cost[j] < min) { min = low-cost[j]; k = j; }
        display(k, closest[k]);
        U = U + k;
        for j in (V-U) if ( C[k,j] < low-cost[j] ) )  { low-cost[j] = C[k,j]; closest[j] = k; }
   }
}
```

Börja med nod "a".
Visa varje steg i dina beräkningar.
Ange *alla* antaganden och visa *alla* beräkningar och mellanresultat

(3p)

Visa att man har kollat på varje kant under algoritmens exekvering.
Använd bilder. I vilka ordning har algoritmen kollat på kanterna?
Vilka kanter **ersätter** andra kanter under algoritmensexekvering?

(2p)

Totalt 5p

(8) Dijkstra

Algoritmen kan skrivas som nedan

```
Dijkstra( a )
{
        S = {a}
        for (i in V-S) D[i] = C[a, i];

        while(!is_empty(V-S)) {
                choose w in V-S such that D[w] is a minimum
                S = S + {w}
                foreach ( v in V-S )  D[v] = min(D[v],  D[w] + C[w,v]);
        }
}
```

Visa hur du skulle **utöka algoritmen** för att bygga den ”Shortest Path Tree” (kortaste väg trädet) samtidigt som man utför algoritmen.

(2p)

Tillämpa **den givna Dijkstra algoritmen (ovan)** på **den riktade grafen**,

(a, b, 12), (a, d, 11), (a, e, 9), (b, c, 7), (c, e, 5), (d, c, 3), (d, e, 1)

Börja med nod ”a”.
Visa varje steg i Dina beräkningar.
Ange *alla* antaganden och visa *alla* beräkningar och mellanresultat

(3p)

Totalt 5p

Bilaga A

Heap Algoritmer

```
Heapify(A, i)
        l = Left(i)
        r = Right(i)
        if l <= A.size and A[l] > A[i] then largest = l else largest = i
        if r <= A.size and A[r] > A[largest] then largest = r
        if largest != i then
                swap(A[i], A[largest])
                Heapify(A, largest)
                 end if
end Heapify

Build(A)
         for i = [A.size / 2] downto 1 do Heapify(A, i)
end Build
```

```
Remove (H, r)
        let      A = H.array
        A[r] = A[A.size]
        A.size--
        Heapify(A, r)
        end Remove
```

```
Add (H, v)
        let A = H.array
        A.size++
        i = A.size
        while i > 1 and A[Parent(i)] < v do
              A[i] = A[Parent(i)]
               i = Parent(i)
               end while
        A[i] = v
end Add
```