ORDINARIE TENTAMEN I
DATASTRUKTURER OCH ALGORITMER DVG B03

160119 kl. 08:15 – 13:15

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.

---

(1) Ge ett kortfattat svar till följande uppgifter ((a)-(j)).

(a) Vad är ”big-O” för en funktion som skriver ut en adjacency matrix? Varför?
(b) Vad gör Dijkstras algoritm?
(c) Vad gör Floyds algoritm?
(d) Vad gör Warshalls algoritm?
(e) Vad gör Topologisk sortering?
(f) Vad är en heap?
(g) Vad är fördelen med hashning?
(h) Vad är en rekursiv definition?
(i) Vad är ett AVL-träd?
(j) Vad är dubbel hashning?

Totalt 5p

(2) Ge ett kortfattat svar till följande uppgifter ((a)-(e)).

(a) Skriv **abstrakt rekursiv pseudokod** för en back-end Boolsk ”find” funktion för att hitta ett värde i en sekvens. Skriv om funktionen så att den returnerar en referens till ett element eller NULLREF i stället för sant eller falsk.

2p

(b) Skriv **abstrakt rekursiv pseudokod** till en funktion för att lägga till (add) ett värde i en sekvens. Skriv om denna funktion med bara två rader i stället för de tre rader som man får om man programmerar enligt sekvensmönstret. Dvs hur kan man kombinera 2 operationer?

2p

(c) Skriv **abstrakt rekursiv pseudokod** till en funktion för att ta bort (remove) ett värde från en sekvens. Förklara hur denna funktion fungerar om inte värdet finns i sekvensen.

2p

(d) Skriv **abstrakt rekursiv pseudokod** till funktionen **T2Q** från labb 1. **T2Q** förvandlar ett binärt träd till en array.

2p

(e) Skriv den **rekursiva definitionen** av ett AVL-träd.

2p

Totalt 10p

(3) Heap

Diskutera ingående hur koden till heap operationer (**se Bilaga A**) fungerar. Använd sekvensen 46, 13, 18, 33, 72, 9, 11, 44, 27, 15, 66 som ett exempel. Anta att det största värdet hamnar i roten. **Visa varje steg i Dina beräkningar.**

5p

(4) Rekursion

Skriv **abstrakt rekursiv pseudokod** till en funktion som tar bort (remove) ett värde från ett binärt sökträd. Vilka aspekter måste man ta hänsyn till när man skriver en sådan funktion? **Ange alla antagande**.

5p

(5) Diskussionsuppgift

Man använder Big-oh när man pratar om en algoritmens prestanda. Vad betyder Big-oh? Vilka aspekter måste man ta hänsyn till när man tillämpar Big-oh. Vilka är de för- och nackdelarna?
Svara ingående. Ange alla antagande.

5p

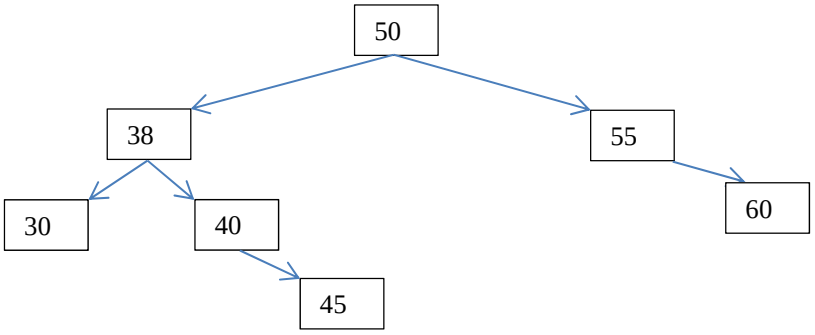
(6) AVL-Träd

Skriv **abstrakt pseudokod** till de 4 rotationsfunktionerna från första principer.

2p

Skriv **abstrakt pseudokod** för att ombalancera (re-balance) ett AVL-träd.
Tillämpa dessa funktioner på AVL-trädet nedan efter att man har tagit bort 60 från trädet.
Visa varje steg i dina beräkningar fram till lösningen. Förklara hur du använder din pseudokod vid varje steg

3p



Totalt 5p

(7) Dijkstra + SPT (Shortest Path Tree)

Tillämpa **den givna Dijkstra_SPT algoritmen (nedan)** 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)

SPT = Shortest Path Tree - dvs kortaste väg trädet (KVT) från en nod till alla de andra.

Börja med nod ”a”.
Visa varje steg i Dina beräkningar.
Ange *alla* antaganden och visa *alla* beräkningar och mellanresultat
Rita **varje steg** i konstruktionen av SPT:et – dvs visa till och med de noder och kanter som läggs till men sedan tas bort.

(3p)

Förklara **principerna** bakom **Dijkstras_SPT** algoritm.

(2p)

Totalt 5p

Dijkstras algoritm med en utökning för SPT

```
Dijkstra_SPT ( a )
{
    S = {a}

        for (i in V-S) {
          D[i]   = C[a, i]              --- initialise D - (edge cost)
          E[i]   = a                    --- initialise E - SPT (edge)
          L[i]   = C[a, i]              --- initialise L - SPT (cost)
        }

        for (i in 1..(|V|-1)) {
                choose w in V-S such that D[w] is a minimum
                S = S + {w}
                foreach ( v in V-S )  if (D[w] + C[w,v] < D[v]) {
                        D[v] = D[w] + C[w,v]
                        E[v] = w
                        L[v] = C[w,v]
                 }
        }
}
```

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
```