OMTENTAMEN I
DATASTRUKTURER OCH ALGORITMER DVG B03

140610 kl. 08:15 – 13:15

Ansvarig Lärare: Donald F. Ross

Hjälpmedel: Inga. Algoritmerna finns i de respektive uppgifterna.

***** OBS *****

Betygsgräns:

Kurs:	Max 60p, Med beröm godkänd 50p, Icke utan beröm godkänd 40p, Godkänd 30p **(varav minimum 15p från tentamen, 15p från labbarna)**
Tentamen:	Max 30p, betyg 5: 26p-30p, betyg 4: 21p-25p, betyg 3: 15p-20p
Labbarna:	Max 30p, betyg 5: 26p-30p, betyg 4: 21p-25p, betyg 3: 15p-20p

SKRIV TYDLIGT – LÄS UPPGIFTERNA NOGGRANT

Ange alla antaganden.

---

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

(a) En definition av en mängd
(b) En definition av en sekvens
(c) En definition av ett träd
(d) En definition av en graf
(e) En definition av en samling

Förklara Dina exempel ((f)-(j)) med några meningar

(f) Ett exempel av ett fullt träd som inte är komplett
(g) Ett exempel av ett komplett träd som inte är fullt
(h) Ett exempel av ett träd som är varken fullt eller komplett eller perfekt
(i) Ett exempel av implementationsabstraktion
(j) Ett exempel av svansrekursion (tail recursion)

Totalt 5p

(2) Ge ett kortfattat svar till följande uppgifter.

(a) En definition av rekursion
(b) En förklaring av abstraktion
(c) Principerna bakom hashning
(d) Principerna bakom Kruskals algoritm
(e) Om man skulle lägga till sekvensen (1, 4, 22, 14, 34, 8, 44, 7) i ett hashutrymme (hash space) med 10 platser, vilken kollisionshanteringsmetod av linjär probning (linear probing) och kvadratisk probning (quadratic probing) är säkrare? **Förklara varför.**

Totalt 5p

(3) Rekursion

Skriv (pseudo)kod till en **rekursiv funktion** (eller **två rekursiva funktioner**) för att räkna fram antalet kanter (edges) i en graf. Ange alla antagande.

5p

(4) Grafoperationer

En graf kan beskrivas som en virtuell (eller abstrakt) maskin. I graflaborationen, filen ”begraph.h” är faktiskt en beskrivning av denna virtuell maskin. Innehållet av ”begraph.h” ges nedan.

```
/*************************************************************************/
/* function prototypes - operations on the Graph (a virtual machine)     */
/*************************************************************************/
/* Graph = (V, E) where V is a set of vertices/nodes and E a set of edges */
/* There are a limited number of operations (9) which can be applied to G */
/*************************************************************************/
void be_display_adjlist();             /* display G as an adjacency list  */
void be_display_adjmatrix();           /* display G as an adjacency matrix*/

void be_addnode(char c);               /* add a vertex (node) to G        */
void be_remnode(char c);               /* remove a vertex (node) from G   */

void be_addedge(char cs, char cd, int v);/* add an edge (with weight) to G*/
void be_remedge(char cs, char cd);     /* remove an edge from G           */

int  be_is_nmember(char c);            /* is a  node a member of G?       */
int  be_is_emember(char cs, char cd);  /* is an edge a member of G?       */

int  be_size();                        /* the number of nodes in G        */
/*************************************************************************/
```

I **front-end:en** kan man specifiera förvillkor (preconditions) till varje operation ovan med hjälp av en eller flera av dessa 9 funktioner.

Ange förvillkoret/förvillkoren till varje av de första åtta operationerna ovan. Dvs i (i) fe_display_adjlist(), (ii) fe_display_adjmatrix(), (iii) fe_addnode(), (iv) fe_remnode(), (v) fe_addedge(), (vi) fe_remedge(), (vii) fe_is_nmember(), (viii) fe_is_emember(). Kom ihåg att ifrån front-end:en kan man utföra ett “samtal” med användaren för att begära olika värden samt skriva felmeddelande.

(5p)

(5) Prims algoritm

a) Tillämpa **Prims algoritm** (nedan) på den **oriktade** grafen:

(a-3-b, a-3-c, a-3-d, b-3-c, b-3-e, c-3-d, c-3-e, c-3-f, d-3-f, e-3-f).

Börja med nod a. Ange alla antaganden och visa alla beräkningar och mellanresultat. Vad representerar resultatet? **Ange varje steg i din beräkning.** Anta att noderna lagras i ”närlistan” (adjacency list) i alfabetisk ordning samt att man söker efter det minimala värdet i alfabetisk ordning.

Rita delresultatet efter varje iteration.

(3p)

b) **Förklara principerna bakom Prims** algoritm.

(2p)

Totalt 5p

Ange *alla* antaganden och visa *alla* beräkningar och mellanresultat

```
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; }
   }
}
```

(6) Dijkstra + SPT (Shortest Path Tree)

Tillämpa **den givna Dijkstra_SPT algoritmen (nedan)** på **den riktade grafen**,

(a, b, 13), (a, d, 12), (a, e, 20), (b, c, 60), (c, e, 50), (d, c, 30), (d, e, 40)

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]
                 }
        }
}
```