OMTENTAMEN I
DATASTRUKTURER OCH ALGORITMER DVG B03

170331 kl. 08:15-13:15

Ansvarig Lärare: Donald F. Ross

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

***** 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 Dijkstras algoritm?
(b) Vad är ”big-O” för Floyds algoritm?
(c) Vad är ”big-O” för lägga-till-operationen i hashning?
(d) Vad gör Warshalls algoritm?
(e) Vad gör en postorder traversering av ett binärt träd?
(f) Vad är en graf?
(g) Vad representerar kanterna i en graf?
(h) Vad är ett ”Free Tree”?
(i) Vad händer när man lägger till en kant till i ett ”Free Tree”?
(j) Vad är abstraktion?

Totalt 5p

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

(a) Skriv **rekursiv (pseudo)kod** för hitta-operation (find) i **ett binärt träd (OBS ej BST)**.
(b) Förklara hur lägga till operationen (add) i en heap fungerar. Se koden nedan. **OBS** – ge inte en översättning kod ➔ svenska! **Förklara principen och/eller ge ett exempel**.

```
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
```

(c) Skriv **abstrakt rekursiv (pseudo)kod** för hitta-operation (**find**) i en sekvens. Funktionen ska returnera en **referens** till elementet eller **NULLREF** om inte värdet finns.
(d) Beskriv hur man förvandla ett **generellt träd** till ett **binärt träd**.
(e) Ge definitionen av ett **komplett träd** (complete tree).

5p

(3) AVL-träd

(a) Ge en definition av ett AVL-träd **1p**
(b) Beskriv **med exempel** de fyra rotationsoperationerna på ett AVL-träd. **2p**
(c) Beskriv hut man bestämmer vilken rotationsoperation som behövs för att ombalancera ett AVL-träd. **2p**

Totalt 5p

(4) Topologisk sortering

Vid ett universitet har vissa kurser förkunskapskrav. I datavetenskap kräver kompilatorkonstruktion (DAV D02) programspråk (DAV C02) som förkunskap. Datastrukturer och algoritmer (DAV B03) är ett förkunskapskrav till programspråk, avancerad programmering i C++ (DAV C05), samt projektarbete i Java (DAV C08). Datastrukturer och algoritmer kräver diskret matematik (MAA B06) samt programutvecklingsmetodik (DAV A02). Operativsystem (DAV B01) kräver i sin tur programutvecklingsmetodik och datorsystemteknik (DAV A14) och är förkunskapskrav till C och UNIX (DAV C18), tillämpad datasäkerhet (DAV C17) samt realtidssystem (DAV C01). Objektorienterade designmetoder (DAV D11) kräver bägge avancerad programmering i C++ och software engineering (DAV C19).

Hur kan man **visa** att kompilatorkonstruktion och objektorienterade designmetoder kräver diskret matematik?

I vilken ordning ska en student som vill läsa på D-nivå ta alla de ovannämnda kurserna? Metoden som kan användas för att komma fram till en lösning heter topologisk sortering. En variant av topologisk sortering är följande algoritm:

Topological Sort

```
tsort(v) -- prints reverse topological order of a DAG from v
{       mark v visited
        for each w adjacent to v if w unvisited tsort(w)
        display(v)
        }
```

Tillämpa den givna algoritmen (ovan) till problemet för att hitta ordning i vilken en student ska läsa kurserna.

Använd denna ordning i kurslistan:
MB06, A02, A14, B01, B03, C01, C02, C05, C08, C17, C18, C19, D02, D11

4p

Beskriv ett annat sätt att utföra en topologisk sortering?

1p

Totalt 5p

(5) Rekursion

Förklara utförligt de för- och nackdelarna med rekursion. Förklara vad en rekursiv definition samt en rekursiv funktion är. Vilka **programmeringsmönster** finns som direkt resultat av de rekursiva definitionerna för en sekvens och ett binärt träd? Ge exempel och (pseudo)kod för att illustrera din diskussion.

5p

(6) Hashning

I hashning har följande metoder för kollisionshantering presenterats, nämligen

1) Open addressing (closed hashing/linear probing) f(i) = i
2) Quadratic probing f(i) = i * i
3) Double hashing f(i) = i * H_2(key)

Där ”i” är värdet på antalet kollisioner (d.v.s. 1, 2, …).

Allmänt sett kan man beskriva kollisionshantering som **H(key) + f(i)** i varje fall.
Anta att **H(key) = key mod 10**. **Hash space = array H[10].**
Anta att **H_2(key) = 7 – (key mod 7).**

Dessa metoder kan betraktas som en historisk utveckling d.v.s. att varje metod försöker att lösa problem med den föregående metoden men i sin tur kan introducera nya problem.

Tillämpa dessa 3 metoder på sekvensen **4, 36, 44, 5, 7, 64, 24** och **visa varje steg** i Dina beräkningar samt diskutera för- och nackdelarna med dessa metoder.

3p

Vilka resultat är **mätbara** när man tillämpar dessa metoder? **1p**

Under vilka omständigheter skulle dubbelhashningsmetoden ovan bete sig på ett likadant sätt som linjärprobning?

1p

Totalt 5p

(7) Graf algoritmer

(a) Tillämpa **Dijkstra_SPT algoritm** på den **oriktade** grafen nedan.

- **Visa varje steg** i Dina beräkningar.
- **Rita en bild av grafen och ge kostnadsmatrisen**.
- **Börja med nod a**.
- **Rita en bild av varje mellanresultatet**.

(a, b, 10), (a, d, 30), (a, e, 100), (b, c, 50), (c, e, 10), (d, c, 20), (d, e, 60)

```
Dijkstra_SPT ( a ) {
   S = {a}
   for (i in V-S) {
      D[i] = C[a, i]
      E[i] = a
      L[i] = C[a, i]
      }
   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]
      }
   }
}
```

3p

(b) Tillämpa **Prims algoritm** på den oriktade grafen nedan.

- **Visa varje steg** i Dina beräkningar.
- **Rita en bild av grafen och ge kostnadsmatrisen**.
- **Börja med nod a**.
- **Rita en bild av varje mellanresultatet**.
- **Anta att närlistan (adjacency list) skapas i alfabetisk ordning**.

(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).

```
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;
          }
       }
}
```

3p

(c) **Förklara ingående** principen bakom Prims algoritm.

OBS: Skriv inte en rad för rad "**översättning**" från koden till svenska (engelska) utom ge **en beskrivning** (dvs en tolkning) av hur algoritmen fungerar vid varje steg. Använd gärna bilder i Din beskrivning.

Sedan ge en sammanfattning av principen i två eller tre meningar.

4p

Totalt 10p