Tentamen i ID1020 Algoritmer och Datastrukturer

Måndagen 2016-01-18 kl 09.00–13.00

Examinator: Johan Karlander, Jim Dowling
Hjälpmedel: Inga
Tentamensfrågorna behöver inte återlamnas efter avslutad tentamen.

Varje inlämnat blad skall förses med följande information:

1. Namn och personnummer
2. Nummer for behandlade uppgifter
3. Sidnummer

Tentamen består av två delar. Del I innehåller tio uppgifter för betyget E. Del II, som inte är obligatorisk, innehåller fyra uppgifter för betygen A-D. Maximal skrivtid är fyra timmar.

Varje lösning kommer att bedömas med 0-10 poäng.

Kravet för godkänt (E) är 50 poäng på del I. Du måste bli godkänd på del I för att kunna erhålla högre betyg. Poängen från del I förs inte vidare till del II.

Under förutsättning att du är godkänd på del I gäller följande betygsgränser för del II:

- Betyg D: 5-10 poäng
- Betyg C: 11-20 poäng
- Betyg B: 21-30 poäng
- Betyg A: 31-40 poäng

Lösningar anslås på kursens hemsida.

Uppgifter Del 1

Följande uppgifter avser betyget E. Krav på *fullständiga lösningar med motiveringar* gäller för samtliga uppgifter.

1. Ange tidskomplexitet - förväntade fallet - som en function av N för följande kodfragment. Använd tilde notationen. N är ett positiv stort heltal. Alla element in arrayn a är olika.

```
for (int i =0 ; i < N; i++)
   for(int j=i+1; j < N; j++)
     for(k=j+1;k<N;k++)
         if (a[i]+a[j]+a[k]== 0) cnt++;
```

   b)

```
for (int i =N ; i > 1; i=i/3) cnt++;
```

   c)

```
for (int i =0 ; i < N; i++)
    for(int j=i+1; j<N;  j++) {
         int x=j+1; int h=N-1; int k;
         while(x<h){
              k= (x+h)/2;
              if (a[i]+a[j]+a[k]== 0) { cnt++; break;}
              if (a[i]+a[j]+a[k]< 0) x=k+1;
              else h=k-1;
         }
    }
```

   d)

```
for (int i =0 ; i < N*N; i++)
    for(int j=1; j < N; j*2)
         cnt++;
```

2. a) Man kan implementer stackar med länkade listor (singly-linked). I en push operation can den nya noden läggas till vid början av listan eller vid slutet. Vad är tidskomplexiteten för push och pop operationer för de två alternativen?

   b) Köer kan också implementeras med länkade listor. Under en enqueue operation kan den nya noden läggas till I början av listan eller vid slutet. Vad är tidskomplexiteten för enqueue och dequeue operationer för de två alternativen?

   c) Stackar kan implementeras med arrayer. I en push operation kan elementet läggas till I slutet av arrayn eller i början. Vad är tidskomplexiteten för push och pop operationer för de två alternativen?

   d) Köer kan implementeras med arrayer. I en enqueue operation kan elementet läggas till I slutet av arrayn eller i början. Vad är tidskomplexiteten för enqueue och dequeue operationer för de två alternativen?

3. Där det fråga efter skall exempeln skall vara algoritmer från boken. Svaren skall vara korta (en mening eller två).

   a) Vad menas med en in-place sorteringsalgoritm?

   b) Ge ett exempel på en in-place sorteringsalgoritm.

   c) Ge ett exempel på en sorteringsalgoritm som har bättre minneskomplexitet än in-place.

   d) Ge ett exempel på en sorteringsalgoritm som har sämre minneskomplexitet än in-place.

   e) Kan en rekursiv divide-and-conquer algoritm vara in-place? Förklara varför eller varför inte.

4. Svaren skall vara korta (en mening eller två).

   a) Vad menas med en stabil sorteringsalgoritm?

   b) Vilka av följande sorteringsalgorithmer är stabila? Selection sort, insertion sort, shell sort, mergesort, och quicksort.

   c) Quicksort och mergesort är både exampel på divide and conquer algoritmer. Uppifrån och ner mergesort och quicksort delar mängden som skall sorteras i mindre delmängder och sortera delmängderna rekursivt. Vilken algoritm skulle kunna dela en mängd med 20 element i delmängder av storlek 5 och 15?

   d) Betrakta delmängderna [1,4,7,10] och[5,8,11,20]. Vilken algoritm skulle kunna dela den större mängd och producera dessa två delmängder?

   e) Beskriv vad som skall göras för att sätta ihop de sorterade delmängderna [1,4,7,10] och [5,8,11,20] för att producera den sorterade superset av 8 element.

5. a) Anta att vi har ett perfekt balanserat binär sökträd. Elementen är heltalen från 1 till 7. Rita trädet. (4 pts)

   b) Vad är tidskomplexiteten för en sökningsoperation i ett röd-svart binär sökträd i värsta fallet? (2 pts)

   c) Vad är tidskomplexiteten för en sökningsoperation i ett binär sökträd i värsta fallet? (2 pts)

   d) Vanligtvis använder man en array av arrayer för att representera en matris. Men för stora glesa arrayer där de flesta element är 0 kräver detta för mycket mine. Beskriv ett bättre alternativ. (2 pts)

6. a) Standard receptet för att ändra storleken av den underliggande arrayen I hashtabeller håller antal lagrade element inom visa gränser. Vad är dessa gränser I termer av N - storleken av arrayn? (2 pts)

   b) Anta att hashfunktionen uniform hashar alla element till någon multipel av 32. Anta att du använder standard resizing receptet. Om tabellen använder linjär sondering (linear probing) vad är då det förväntat antal array accesser för en sökträff och sökmiss(2 pts)

   c) Anta att hashfunktionen uniform hashar alla element till någon multipel av 32. Anta att du använder standard resizing receptet. Om tabellen använder listkedjande (separate chaining) vad är då det förväntat antal array accesser för en sökträff och sökmiss(2 pts)

d) Jamför hashtabeller med röd-svarta BSTs för symboltabeller. Vad är fördelarna och nackdelarna med båda? (4 pts)

7. Nedan visas ett antal binära träd. Svaren till delfrågorna nedan kan vara ingetdera, alla träd, eller vissa träd.

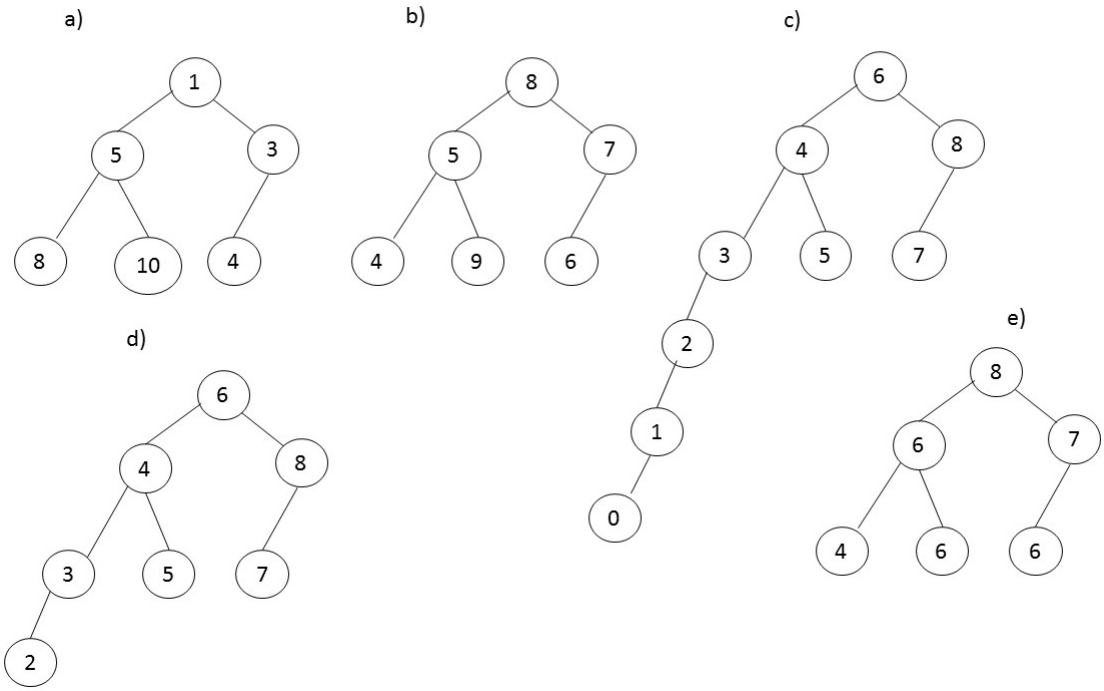
   a) Vilka träd skule kunna representera en binär heap ?

   b) Vilka skulle kunna vara binary sökträd.?

   c) Vilka skulle kunna representera röd-svarta binära sökträd. (Färgen av länken visas ej).

   d) Vilka träd skulle kunna representera det implicita trädet i quick-union.

   e) Vilka träd skulle kunna representera minimum-orienterade prioritetsköer?



8. Vi har en riktad graf G. Vi gör en DFS-sökning med s som startnod. Det ger oss ett sökträd T. Låt u och v vara två noder i grafen och antag att det går en riktad stig från u till v i grafen. Antag att u och v båda ligger i T och att algoritmen besöker u för första gången i steg i i algoritmen och v för första gången i steg j där $i < j$. Avgör nu följande: Är det alltid sant att trädet T är sådant att det innehåller en stig med s, u, v i precis denna ordning?

   Ge motivering eller motexempel.

9. Kruskals algoritm fungerar som bekant även om vissa kanter i en graf har samma vikt - fast det minimala spännande trädet behöver då inte vara unikt. Algoritmen fungerar även om man har negativa kantvikter. Låt oss nu anta att vi har en sammanhängande graf G där alla kanter har vikt 1 eller -1 (dvs vissa kanter har ena vikten och de övriga

den andra). Föreslå en modifiering av Kruskals algoritm som utnyttjar den enkla formen på kantvikterna och som ger bättre körtid än den vanliga varianten av Kruskal.

10. Antag att vi har en oriktad, sammanhängande graf G. Vi säger att om vi ger varje kant en riktning så har vi gett G en *orientering*.

    a. Beskriv en effektiv algoritm som hittar en orientering som gör att den nya riktade grafen är acyklisk.
    b. Låt G vara den kompletta oriktade grafen med 6 noder. Hur många orienteringar finns det som gör grafen acyklisk?

Uppgifter Del 2

Följande uppgifter avser betygen A-D. Krav på *fullständiga lösningar med motiveringar* gäller för samtliga uppgifter.

1. Skriv ett program (pseudokod eller Java) som loser följande perkolationsproblem. For denna fråga får du bara använda grundläggande programmerings konstruktioner så om du vill använda en algoritm från boken behöver du koda algoritmen. Du har en N * N rutnät och en input fil som består av en sekvens av element av formen (i,j,Dir) där i är rad nummer, j är kolumn nummer och där Dir är en av down,up,left,right. Elementet(i,j,Dir) skall skapa en förbindelse mellan elementet (I,j) i rutnätet till ett annat element (antingen nedandör, ovanför, till höger eller till vänster). (T.ex. (4,5,down) I en 10 * 10 rutnät betyder at ten förbindelse skall skapa mellan elementen (3,5) och (4,5). Initiallt finns det inga förbindelser. Programmet skall läsa från input sekvensen element efter element, skapa en förbindelse och kontrollera om systemet perkolerar. Systemet perkolerar när det finns en väg av förbindelser mellan ett element på nedersta raden (vilken som helst) och ett element på översta raden. Programmet skall sluta när perkolation uppnås och returnera antalet skapade förbindelser (d.v.s antalet element från inputfilen som har behandlats).

   Ge tidskomplexiteten för den kombinerad operation av skapande av en förbindelse och själva perkolationskontrollen. För full poäng bör tidskomplexitetn vara kvadratiskt. Bonuspoäng ges om det är avsevärt bättre.

2. Den maximum-orienterade prioritetskö implementationen som presenterades under kursen stödjer metoderna delMax och insert som visas nedan.

```
public Key delMax() {
    Key max = pq[1];
    exch(1,N--);
    pq[N+1] = null;
    sink(1);
    return max;}

public void insert(Key v){
   pq[++N] = v;
   swim(N);}
```

a) Vad är syftet med raden pq[N+1]= null I delMax

b) Vad är invarianten får maximum-orienterade prioritetsköer?

c) Programmera ytterligare en metod - changeKey(int I, Key key) vilket ändrar på värdet av nyckeln I den i:te elementet. Du kan behöva använda hjälpmetoderna sink eller swim (eller båda) för att bebehålla invarianten.

d) Programmera ytterligare en metod - delete(int i) som plockar bort det i:te elementet (d.v.s ett godtyckligt element i kontrast mot bara maximum).

3. Du har beslutat att controller om din smak i literature beror mer på författarens ordförråd än själva historian. Du har valt ut 3 textfiler från din favoriter och 3 textfiler som du hatar, alla av längden N. Första steget är att hitta alla ord som används av alla dina favoriter tillsammans med antalet förekomster (summerat ihop). Denna ordmängd betecknas Fav. Andra steget är att hitta alla ord som används av alla dina icke favoriter, summera antalet förkomster för att producera ordmängden nonFav. Tredje steget är att filrera Fav genom att ta bort alla ord som också förekommer i nonFav, för att producera mängden filFav. Fjärde steget att skriva ut filFav sorterat efter antalet förekomster (det mest använda först). Eventuellt gillar du dessa ord till den graden att du därmed gillar verken. Femte steger är att filtrera nonFav genom att ta bort alla ord som också finns i och producera ordmängden filNonFav. Sjätte steget är att skriva ut order i filNonFav sorterat efter antalet förekomster. Kan det vara så att du ogillar dessa ord till den graden att det påverkar ditt omdöme om verken.

   Du skall programmera (Java or pseudo-code) en lösning till detta problem. För denna upgift kan du direct använda algoritmerna som har presentera I kurser, men var tydlig vilka metoder och version du änvänder (t.ex., du skulle kunna skriva Quick.sort(a) och nämna att du använder versionen som använder insertion sort när längden av listan är mindre än 10). Du skall ange tidskomplexitet för din lösning för alla steg i termer av N, U som är en övre gräns på antalet unika ord I alla text filer, och F som är en övre gräns för antalet ord i de två filtrerade mängderna filFav och filNonFav.

4. Om G är en riktad graf säger vi att G är *semisammanhängande* om det för varje par a, b av noder gäller att det finns en riktad väg från a till b eller en riktad väg från b till a eller båda. Lägg märke till att detta begrepp inte är samma sak som starkt sammanhängande. Hur avgör man om en graf är semisammanhängande? Beskriv en effektiv algoritm som löser detta problem. För att du skall få full poäng krävs det att algoritmen inte bara gör en DFS-sökning från varje nod. Det krävs en effektivare lösning.