Tentamen
Datastrukturer för D2
TDA 131

18 december 2004

- Tid: 8.30 - 12.30
- Ansvarig: Peter Dybjer, tel 7721035 eller 405836
- Max poäng: 60. Vart och ett av de sex problemen ger maximalt 10 p.
- Betygsgränser: 3 = 30 p, 4 = 40 p, 5 = 50 p
- Inga hjälpmedel.
- Skriv tydligt och disponera papperet på ett lämpligt sätt.
- Börja varje ny uppgift på nytt blad.
- Skriv endast på en sida av papperet.
- **Kom ihåg:** alla svar ska motiveras väl!
- Poängavdrag kan ges för onödigt långa, komplicerade eller ostrukturerade lösningar.
- Lycka till!

1. Vilka av följande påståenden är korrekta och vilka är felaktiga? Motivera! Svar utan korrekt motivering ger inga poäng.

   (a) Man sparar minne om man lagrar ett binärt träd i ett fält i stället för som en länkad struktur.

   (b) Sökning i ett splayträd tar $O(\log n)$ i värsta fall.

   (c) Hashtabeller med öppen adressering ("open addressing") använder mer minne än hashning med hinkar ("hashing in buckets" eller "chaining").

   (d) Grannmatriser använder mer minne än grannlistor om man har glesa grafer, dvs grafer som inte har så många bågar.

   (e) Om antalet noder i en riktad graf är n så är antalet bågar högst n^2 om vi förutsätter att det inte finns några parallella bågar. (Dock tillåter vi "self loops" dvs bågar med samma källa och mål.)

2. (a) Vad beräknar följande två Java-metoder?

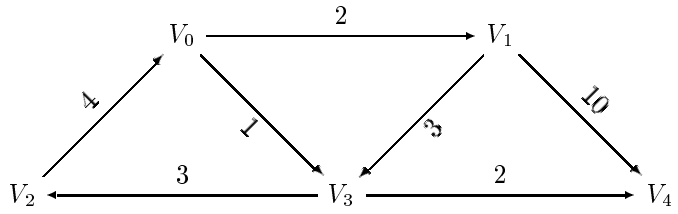
```
int f(int[] a){
   int s = a[0];
   for (int iter = 1; iter < a.length; iter++) {
      s = s + a[iter];
   }
   return s;
}

int g(int[] a, int k){
   for (int iter = 0; iter < a.length; iter++) {
      if (a[iter] == k) {
         return iter;
      }
   }
   return -1;
}
```

   (b) Låt $T_{\mathtt{f}}(\mathtt{n})$ och $T_{\mathtt{g}}(\mathtt{n})$ vara tiderna det tar att exekvera `f` och `g` på ett fält `a` med längden `n` i *värsta* fall. Ange lämpliga O-komplexitetsklasser för $T_{\mathtt{f}}(\mathtt{n})$ och $T_{\mathtt{g}}(\mathtt{n})$! För full poäng ska komplexitetsklasserna vara optimala, dvs så små som möjligt.

   (c) Låt $T'_{\mathtt{f}}(\mathtt{n})$ och $T'_{\mathtt{g}}(\mathtt{n})$ vara exekveringstiderna för `f` och `g` i *bästa* fall.

   - Är $T_{\mathtt{f}}(\mathtt{n}) = T'_{\mathtt{f}}(\mathtt{n})$?
   - Har $T_{\mathtt{f}}(\mathtt{n})$ och $T'_{\mathtt{f}}(\mathtt{n})$ samma O-komplexitet?
   - Är $T_{\mathtt{g}}(\mathtt{n}) = T'_{\mathtt{g}}(\mathtt{n})$?
   - Har $T_{\mathtt{g}}(\mathtt{n})$ och $T'_{\mathtt{g}}(\mathtt{n})$ samma O-komplexitet?

3. Betrakta följande graf:



   (a) Visa hur man representerar denna graf med hjälp av en grannmatris!

   (b) Visa hur man representerar denna graf med hjälp av en grannlista!

   (c) Dijkstras algoritm kan användas för att bestämma kortaste vägen mellan två noder. I figuren ovan är V_0 startnod och V_4 målnod. Din uppgift är att förklara hur algoritmen fungerar genom att visa steg för steg hur algoritmen arbetar på grafen ovan och hur de kortaste vägarna successivt bestäms. Du ska alltså rita en följd av grafer som visar vilka mellanresultat som beräknats efter varje iteration. Som vanligt ska du också ge motiveringar; det räcker inte att bara rita figurer.

4. Analysera effektiviteten hos insertion sort och quicksort för små och stora listor!

   (a) Antag först att vi vill sortera en lista med precis 4 element. Hur många jämförelser gör de båda algoritmerna i *värsta* fall och i *bästa* fall? Motivera ditt svar på följande sätt:

   - Ge exempel på en indatalista med 4 element som ger *maximalt* antal jämförelser för *insertion sort* och förklara vilka jämförelser som görs.
   - Ge exempel på en indatalista med 4 element som ger *minimalt* antal jämförelser för *insertion sort* och förklara vilka jämförelser som görs.
   - Ge exempel på en indatalista med 4 element som ger *maximalt* antal jämförelser för *quicksort* och förklara vilka jämförelser som görs.
   - Ge exempel på en indatalista med 4 element som ger *minimalt* antal jämförelser för *quicksort* och förklara vilka jämförelser som görs.

   För att få full poäng ska du förutsätta att algoritmerna är implementerade på ett optimalt sätt. (Du behöver dock inte visa hur de är implementerade.)

   (b) Är insertion sort eller quicksort effektivast om man vill sortera 1000 element? Analysera både värsta fallet och bästa fallet. Här kan du förstås inte beräkna den exakta tidsåtgången utan det räcker att du utgår från din kunskap om algoritmernas asymptotiska komplexitet i bästa och värsta fallet.

5. Här är ett litet gränssnitt (i Java) för ändliga mängder av heltal:

```
public interface Set {
   public boolean member(int element);
   public void remove(int element);
}
```

Metodanropet `member(element)` ska returnera `true` om `element` finns i mängden och `false` annars. Metodanropet `remove(element)` ska ta bort elementet `element` från mängden.

(a) Implementera gränssnittet `Set` med hjälp av hashtabeller! Skriv antingen Javakod eller detaljerad pseudokod. Du behöver bara implementera de två metoderna och de hjälpmetoder och hjälpklasser du behöver. Du behöver heller inte implementera någon konstruerare. Du kan själv välja storleken på hashtabellen. Som hashfunktion kan du använda modulofunktionen (`k` modulo `N` skrivs `k % N` i Java). Du får också själv välja hur du implementerar hashtabellen (öppen adressering, hashning med hinkar, ...). Du måste dock förklara vilken implementeringsmetod du använt.

(Du kan få delpoäng på denna uppgift även om du inte ger en fullständig implementering i Java eller fullständig pseudokod. T ex får du avdrag om du skriver lösningen i ofullständig pseudokod eller om du förutsätter att det finns en viss hjälpmetod eller hjälpklass utan att implementera den.)

(b) Visa sedan hur du har tänkt dig att lagra följande mängd av element om hashtabellens storlek är $N = 11$!

$$19, 53, 30, 64, 31$$

Hur ser din datastruktur ut om du sedan tar bort elementet 19 från mändgen? (Du kan få poäng på denna uppgift även om du inte gjort en fullständig implementering i (a).)

6. (a) Heapar brukar lagras i fält. Skriv en algoritm i detaljerad pseudokod eller i Java som har som indata ett fält (med storleken n) med heltal och returnerar `true` om fältet representerar en heap med storleken n och `false` annars. Även en mer skissartad beskrivning av algoritmen kan ge delpoäng om den är tillräckligt klar.

(b) Vilken tidskomplexitet har ditt program uttryckt som funktion av antalet element i heapen? Motivera!

För full poäng på uppgiften ska algoritmen ha korrekt angiven optimal asymptotisk komplexitet.