Tentamen
Datastrukturer för D2
TDA 131

21 december 2002

- Tid: 8.45 - 12.45
- Ansvarig: Peter Dybjer, tel 7721035 eller 405836
- Max poäng: 60.
- Betygsgränser: 3 = 30 p, 4 = 40 p, 5 = 50 p
- Hjälpmedel. *En* av följande böcker får användas:
  - Kursboken Data Structures and Algorithms in Java, 2nd ed, av Michael T. Goodrich och Roberto Tamassia (G & T).
  - Fjolårets kursbok Classic Data Structures in Java av Timothy Budd.
  - Algorithms, Data Structures, and Problem Solving with C++ av Mark Allen Weiss.
  - Data Structures and Algorithm Analysis in Ada av Mark Allen Weiss.
  - Introduction to Algorithms, 2nd edition av Thomas H. Cormann, Charles E. Leiserson, Ronald Rivest och Clifford Stern.
- Skriv tydligt och disponera papperet på ett lämpligt sätt.
- Börja varje ny uppgift på nytt blad.
- Skriv endast på en sida av papperet.
- Alla svar ska motiveras väl.
- Poängavdrag kan ges för onödigt långa, komplicerade eller ostrukturerade lösningar.
- Om du använder kod från någon av de tillåtna böckerna, måste du kopiera koden! (Om du dock använder *årets kursbok* räcker det om du tydligt klargör vilket kodavsnitt som avses, genom att ange både sida och vilka rader på sidan som avses!)
- Lycka till!

1. Antag att du ska lagra talen i mängden $\{1, 2, 3, 4, 5, 6, 7\}$ i ett binärt sökträd. Först ska du använda algoritmen för insättning i binära sökträd (se t ex Goodrich och Tamassia 9.1 eller Budd sid 357-365) och sedan ska du använda algoritmen för insättning i AVL-träd (se t ex Goodrich och Tamassia 9.2 och Budd 14.2). Vilket binärt sökträd eller AVL-träd man får beror förstås på i vilken ordning man sätter in elementen. Vi ska nu titta på de bästa och värsta fallen.

   (a) Allmänna binära sökträd kan bli mycket obalanserade. Varför vill man undvika detta och se till att trädets höjd blir så liten som möjligt? (2p)

   (b) Vilken är den minimala höjden hos ett binärt sökträd som innehåller elementen $\{1, 2, 3, 4, 5, 6, 7\}$? Ange i vilken ordning man kan sätta in elementen så att du får denna minimala höjd! (Flera ordningar är möjliga, det räcker om du anger en!) Rita också det resulterande trädet! (2p)

   (c) Vilken är den maximala höjden hos ett binärt sökträd som innehåller elementen $\{1, 2, 3, 4, 5, 6, 7\}$? Ange i vilken ordning man kan sätta in elementen så att du får denna maximala höjd! (Flera ordningar är möjliga, det räcker om du anger en!) Rita också det resulterande trädet! (2p)

   (d) Algoritmen för insättning i AVL-träd ser till att träden alltid är höjdbalanserade genom att strukturera om dem när det behövs. Vilket är det maximala antalet omstruktureringar du kan behöva göra när du sätter in talen $\{1, 2, 3, 4, 5, 6, 7\}$? Ange vilken ordning elementen ska komma för att få detta värsta fall! Rita en bild på varje omstrukturering du behöver göra! (3p)

   (e) Vad är det värsta som kan hända om du sätter in talen $\{1, 2, 3, 4, 5, 6, 7\}$ i en skiplista? Med "värsta" menar vi här värsta fallet med avseende på sökning efter ett visst element i skiplistan. Vad är den asymptotiska komplexiteten (uttryckt i O-notation) för värsta fallet hos denna operation som funktion av antalet element n i den mängd som lagrats i listan? (Mängden $\{1, 2, 3, 4, 5, 6, 7\}$ innehåller alltså 7 element, även om skiplistan lagrar flera kopior av elementen.) Svaren på ovanstående frågor kan bero på exakt hur man valt att implementera skiplistorna. Diskutera! (3p)

2. (a) Personerna A, B, C, D, E och F bor tillsammans i ett hus. De vill koppla samman sina datorer, men vill använda så lite kabel som möjligt. Nätverket ska alltså bilda en *sammanhängande* graf, där summan av alla bågars vikter är så liten som möjligt. Minimala kabelavståndet mellan två datorer ges av följande grannmatris:

	A	B	C	D	E	F
A	0	6	1	5	7	5
B	6	0	5	10	3	9
C	1	5	0	5	6	4
D	5	10	5	0	8	2
E	7	3	6	8	0	6
F	5	9	4	2	6	0

Rita en bild som visar hur A, B, C, D, E och F bör koppla samman sina datorer! Förklara också hur algoritmen du använt fungerar genom att rita bilder som visar hur algoritmen steg för steg konstruerar kabelnätet. (5p)

(b) Man kan använda Dijkstras algoritm för att finna kortaste vägen mellan två noder i en viktad graf. Antag att alla vikterna (avstånden mellan två grannoder) är 1. Hur kan vi då förenkla Dijkstras algoritm? Vilken asymptotisk komplexitet har den förenklade algoritmen? Du ska ange O-komplexitet som en funktion av antalet bågar och antalet noder i grafen. (5p)

(c) Antag att du inte behöver returnera kortaste vägen, utan bara vill veta längden hos den kortaste vägen. Hur påverkas då den asymptotiska komplexiteten hos Dijkstras algoritm? (2p)

3. (a) Skriv en Java-metod

```
public int fib(int n) { ... }
```

så att `fib(n)` returnerar det `n`te fibonaccitalet, dvs resultaten ska uppfylla

```
fib(0) = 0
fib(1) = 1
fib(n+2) = fib(n) + fib(n+1)
```

Din metod ska ha asymptotisk tidskomplexitet $O(n)$ under antagandet att tiden det tar att utföra en addition är $O(1)$. (4p)

(b) Låt $T(\mathtt{n})$ vara antalet primitiva operationer din metod utför när den beräknar `fib(n)`. Vad är $T(0), T(1)$ och $T(2)$? Du ska alltså räkna antalet additioner, antalet tilldelningar, antalet jämförelser, osv! Skriv ner vilka antaganden du gör om vad som utgör en primitiv operation och skriv ner exakt vilka primitiva operationer din `fib`-metod använder! (3p)

(c) Ställ sedan upp rekursionsekvationer för $T(\mathtt{n})$! Dvs definiera $T(\mathtt{n})$ genom att säga vad $T(0)$ är och hur man kan räkna ut $T(\mathtt{n}+1)$ om man vet $T(\mathtt{n})$. Motivera varför $T(\mathtt{n})$ är $O(\mathtt{n})$! (3p)

(d) Man kan också mäta tidskomplexiteten som funktion av längden `m` av talet `n` i binär representation. Exempelvis har talet 3 binärrepresentationen 11 som har längden 2, och talet 12 har binärrepresentationen 1100 som har längden 4. Låt $T'(\mathtt{m})$ vara maximala antalet operationer din algoritm utför för att beräkna `fib(n)` där `m` är längden av `n` i binärrepresentation. Ange lämplig O-komplexitetsklass för $T'(\mathtt{m})$! (2p)

(e) Antagandet att tiden det tar att beräkna en addition är $O(1)$ förutsätter att summan verkligen kan lagras som en `int` utan att man får "overflow". Hur gör man om man vill addera godtyckligt stora heltal? Man vet alltså inte på förhand hur många binära siffror man behöver. Vilken O-komplexitet har addition is så fall? Uppskatta också O-komplexiteten för din `fib`-metod om den använder denna metod för att addera stora heltal? (2p)

4. (a) Antag att du ska göra en hashtabell för ett fastighetsregister för Göteborg. Söknycklarna är fastighetsbeteckningar som består av ett områdesnamn följt av ett 2-siffigt och ett 3-siffigt tal, t ex `Torp 44:106`. Du vet dock inte vilka områdesnamnen är, bara att de är representerade som strängar med maximalt 12 bokstäver. Vi antar vidare att hashtabellens storlek är ett 5-siffrigt primtal. Hitta på en bra hashfunktion! (4p)

   (b) Vilka av följande abstrakta datatyper är lämpliga och vilka är olämpliga att implementera med hashtabeller:

      i. prioritetsköer,
      ii. mängder,
      iii. multimängder?

   Motivera alla svar!

   Anm. Multimängder (multisets) kallas också påsar ("bags") och skiljer sig från mängder genom att de kan ha flera kopior av samma element. Multimängden $\{1, 1, 2\}$ skiljer sig alltså från påsen $\{1, 2\}$. Operationerna på multimängder motsvarar operationerna på mängder, t ex union, snitt, om ett visst element finns i mängden, osv. Skillnaden är dock att multimängdsoperationerna måste hålla reda på hur många förekomster det finns av ett visst element. (4p)

   (c) Du har bestämt dig för att använda en hashtabell, men du vill också veta värstafalls-komplexiteten för sökning, insättning, och borttagning av element. Hur beror värstafallskomplexiteten på antalet lagrade element n och på hashtabellens storlek N för

      i. öppen adressering; (Här får du förutsätta att belastningsfaktorn är mindre än 1, dvs att hashtabellen inte är full).
      ii. hashning med hinkar ("hashing in buckets, chained hashing"), där hinkarna implementerats med länkade listor. (4p)

5. Givet två sorterade vektorer, båda av längd n, konstruera en algoritm (i pseudokod) som returnerar det lägre medianelementet i den sammanslagna vektorn! Om du t ex har vektorerna $[1, 2, 5]$ och $[3, 4, 6]$ är alltså 3 "det lägre medianelementet" och 4 "det högre medianelementet", eftersom den sammanslagna (sorterade) vektorn är listan $[1, 2, 3, 4, 5, 6]$. För att få full poäng ska din algoritm ha komplexiteten $O(\log n)$ och du ska motivera varför den har denna komplexitet. Du kan dock få delpoäng för en mindre effektiv algoritm, om du gör en korrekt komplexitetsanalys. Antalet delpoäng beror då på hur effektiv din algoritm är. (10 p)