Tentamen
Datastrukturer D
DAT 035/INN960

19 december 2008

- Tid: 8.30 - 12.30
- Ansvarig: Peter Dybjer, 0736-341480 (innan 10.30). Håkan Burden, 0730-77287777, och Staffan Björensjö (after 10.30).
- Max poäng på tentamen: 60.
- Betygsgränser, CTH: 3 = 24 p, 4 = 36 p, 5 = 48 p, GU: G = 24 p, VG = 48 p.
- Hjälpmedel: *handskrivna* anteckningar på *ett* A4-blad. Man får skriva på båda sidorna och texten måste kunna läsas utan förstoringsglas. Anteckningar som inte uppfyller detta krav kommer att beslagtas!

  Föreläsningsanteckningar om datastrukturer i Haskell, av Bror Bjerner
- 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. Antag att du sätter in elementen 4, 3, 2, 1, 5 ett efter ett (i denna ordning) i följande datastrukturer (mha standardalgoritmerna för insättning)

   (a) Vanligt binärt sökträd, utan balansering. (1p)

   (b) Splayträd. (3p)

   (c) (2,4)-träd. Kom ihåg att i dessa träd har alla inre noder mellan 2 och 4 barn. Alla löv ligger på samma djup så trädet är perfekt balanserat. Varje nod lagrar mellan 1 och 3 element, och trädet har en sökträdsegenskap som generaliserar det binära sökträdets. (3p)

   (d) Heap (en vanlig binär "min-heap" som lagrar minsta elementet i roten). (3p)

   Rita hur träden byggs upp steg för steg efter varje insättning och förklara vad som händer! I (a) räcker det dock att du ritar det slutgiltiga binära sökträdet.

2. Avgör om vart och ett av följande påståenden är sant eller falskt! Endast svar med korrekt motivering godtages.

   (a) Om en algoritm har den asymptotiska komplexiteten $O(1)$ så tar den alltid lika lång tid att exekvera oavsett indata. (2p)

   (b) Höjden på ett binärt sökträd med n noder är alltid $O(\log n)$. (2p)

   (c) Det gäller alltid att $O(\log_2 n) = O(\log_4 n)$. (2p)

   (d) Höjden på ett (2,4)-träd som lagrar n element är alltid $O(\log n)$. (2p)

   (e) Man kan sortera ett fält på tiden $O(n \log n)$ i medeltal genom att först sätta in dem ett efter ett i en skipplista, och sedan plocka ut dem i ordning från understa raden i skipplistan. (2p)

3. (a) Följande program har som indata ett fält `A` av heltal.

```
int f(int[] A) {
  int n = A.length;
  int s = 0;
  for (int i = 0; i < n; i++) {
    s = s + A[0];
    for (int j = 1; j <= i; j++) s = s + A[j]
  }
  return s
```

Vilken är O-komplexiteten hos programmet uttryckt som funktion av `n`, längden hos indatafältet `A`? Motivera! För full poäng ska O-komplexiteten vara en "skarp" uppskattning. (3p)

(b) Följande program har som indata ett fält `A` av heltal $<$ `N`.

```
void f(int[] A, int N){
  int n = A.length;
  int[] B = new int[N];
  for (int j = 0; j < N; j++) B[j] = 0;
  for (int i = 0; i < n; i++) B[A[i]]++;
  for (int j = 0; j < N; j++) {
     int i = 0;
     while(B[j] > 0){
        A[i++] = j;
        B[j]--;
     }
  }
  return;
}
```

Vilken är algoritmens O-komplexitet uttryckt som funktion av fältets storlek `n` och `N`? Motivera! För full poäng ska O-komplexiteten vara en "skarp" uppskattning. (3p)

(c) Är det korrekt att programmet sorterar `A`? Om ja, motivera! Om nej, lokalisera felet! (4p)

4. Följande grannmatris representerar en viktad oriktad graf.

	A	B	C	D	E	F
A	-	1	4	5	-	-
B	1	-	2	-	2	-
C	4	2	-	5	6	4
D	5	-	5	-	-	2
E	-	2	6	-	-	6
F	-	-	4	2	6	-

Talen i matrisen representerar bågarnas vikter. Ett streck (-) betyder att bågen saknas.

(a) Rita grafen! Sätt ut vikterna på bågarna. (1p)

(b) Man kan använda Dijkstras algoritm för att finna kortaste vägen mellan två noder i en graf. Som i lab 3 vill man ofta både veta vilka noder som ligger på den kortaste vägen och hur lång vägen är (dvs summan av de ingående bågarnas vikter). Beskriv i ord hur Dijkstras algoritm gör för att beräkna dessa två saker. (Du får delpoäng om du bara kan beräkna längden men inte nodlistan.) Du behöver inte ge fullständig pseudokod - det räcker om du förklarar vilka datastrukturer du använder och förklarar noggrannt vilken roll de spelar under beräkningen. (4p)

(c) Dijkstras algoritm arbetar stegvis. Visa hur algoritmen fungerar genom att visa vilka mellanresultat (tillstånd hos datastrukturerna) som Dijkstras algoritm (enligt din beskrivning ovan) lagrar för att kunna returnera den kortaste vägen och denna vägs längd mellan A och F i grafen ovan. (3p)

(d) Är det i allmänhet lämpligt att använda en grannmatris för att representera en graf när man implementerar Dijkstra! Varför eller varför inte? Motivera med hjälp av O-komplexiteter. (2p)

5. Klassen `ArrayList` i Java Collections kan lagra godtyckligt stora listor. Här är ett utdrag ur beskrivningen av klassen

> Resizable-array implementation of the `List` interface. ... The `size`, `isEmpty`, `get`, `set`, `iterator`, and `listIterator` operations run in constant time. The add operation runs in amortized constant time, ... Each `ArrayList` instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an `ArrayList`, its capacity grows automatically.

(a) Förklara betydelsen av "The add operation runs in amortized constant time"! (2p)

(b) Visa hur man lämpligen kan implementera `ArrayList` genom att skriva en klass i Java eller i detaljerad pseudokod. Det räcker om du visar vilka tillståndsvariabler ("fields") man behöver samt hur man implementerar följande operationer beskrivna i "constructor summary" och "method summary":

i. `ArrayList()` - Constructs an empty list with an initial capacity of ten.

ii. `E get(int index)` - Returns the element at the specified position in this list.

iii. `void add(E o)` - Appends the specified element to the end of this list.

iv. `void add(int index, E element)` - Inserts the specified element at the specified position in this list.

(8p)

6. En min-max-heap är en datastruktur med effektiva metoder både för att ta ut minsta och största elementet i en datasamling. Precis som en vanlig heap är den ett fullständigt binärt träd ("complete binary tree"), men elementen i en min-max-heap är ordnade på ett annat sätt. Alla noder som ligger på jämnt djup (exempelvis roten) har nycklar som är mindre än (eller lika med) sina ättlingar, medan alla noder som ligger på udda djup har söknycklar som är större än (eller lika med) sina ättlingar. (En ättling till noden n är en nod som finns i delträdet med roten n.)

   (a) Rita en min-max-heap som innehåller elementen 1,2,3,4,5,6,7,8,9,10! Observera att det finns många sådana min-max-heapar. (2p)

   (b) Hur implementeras operationen `max` som returnerar det största elementet i min-max-heapen (utan att ta bort det)? Vilken är den asymptotiska tidskomplexiteten hos `max`. (3p)

   (c) Hur implementeras operationen `deleteMin` som tar bort det minsta elementet i en min-max-heap? Visa resultatet av att utföra `deleteMin` på min-max-heapen i (a). Asymptotisk komplexitet hos `deleteMin`? (5p)

   Du ska beskriva implementeringarna här med hjälp av pseudokod. (Javakod accepteras också.)

   Observera att i t.ex. G & T är heapar en datastruktur för lexika där man lagrar par av söknycklar och element. Men man kan förstås även lagra datasamlingar där söknyckeln och elementet är samma sak.