Tentamen med lösningsförslag
Datastrukturer för D2
DAT 035

17 december 2005

- Tid: 8.30 - 12.30
- Ansvarig: Peter Dybjer, tel 7721035 eller 405836
- Max poäng på tentamen: 60. (Bonuspoäng från övningarna tillkommer.)
- 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 bra motivering ger inga poäng.

   (a) Om man implementerar en stack med hjälp av ett dynamiskt fält ("extendable array") så blir operationerna push (sätt in ett element överst i stacken) och pop (ta bort översta elementet i stacken) $O(1)$ i värsta fall.

   Svar. Nej, när fältet är fullt behöver man skapa ett nytt större fält och kopiera stacken till detta. Tidskomplexiteten för denna operation är $O(n)$, där n är antalet element i stacken.

   (b) Om man implementerar en stack med hjälp av en enkellänkad lista blir operationerna push och pop $O(1)$ i värsta fall.

   Svar. Ja. Tiden det tar att sätta in eller ta bort ett element i början av en enkellänkad lista är oberoende av hur stor den är.

   (c) Om ett binärt träd har n noder och höjden är h så är alltid $h \geq \log_2 n$. Ett träds höjd definieras som maximala noddjupet. Tänk på att roten ligger på djupet 0.

   Svar. Nej. Det finns ett binärt träd med 3 noder som har höjden 1 och $\log_2 3 > 1$.

   (d) Om ett (2,4)-träd har n noder (och alltså lagrar maximalt $3n$ element) och höjden är h så gäller att $h \leq \log_2 n$.

   Svar. Ja. Minimala antalet noder i ett (2,4)-träd med höjden h uppstår när vi bara har 2-noder, dvs när (2,4)-trädet är ett binärt sökträd. Eftersom alla löv ligger på samma djup har vi alltså $n \geq 2^{h+1} - 1$. Så $h \leq \log(n+1) - 1 \leq \log n$ om $n \geq 1$.

   (e) Det är lämpligt att använda en hashtabell för att lagra svenska ord som används av ett program som gör rättstavningskontroll.

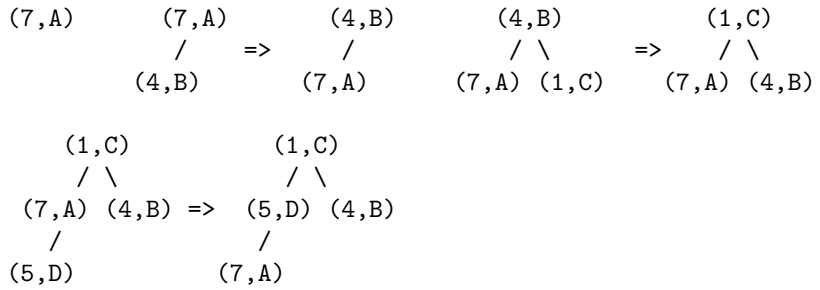
   Svar. Ja. Ett rättstavningsprogram använder en ordlista med korrekta ord och jämför orden i ett dokument med dessa. Eftersom man söker i ordlistan väldigt många gånger behöver man en datastruktur med mycket snabb sökning. En hashtabell är här lämplig eftersom sökning är $O(1)$ om vi kan bortse från kollisioner. (I riktiga rättstavningsprogram är i själva verket effektiviteten så viktig att man ofta tillåter sig att fuska med kollisionshanteringen. Visserligen gör då rättstavningsprogrammet fel ibland, men sökningen blir snabbare.)

   (10p)

2. Du ska lagra mängden $(7, A), (4, B), (1, C), (5, D)$ av par av nycklar och värden i fyra olika datastrukturer: heap, binärt sökträd, splayträd och skiplista. I alla fallen ska du använda standardalgoritmen för att sätta in nya element. Elementen ska sättas in ett efter ett i den ovan angivna ordningen.

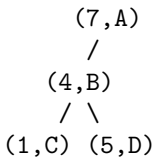
   Heap. Här ska du alltså använda heltalsnycklarna som prioriteter. Du ska börja med den tomma heapen och sedan visa steg för steg hur standardalgoritmen bygger upp heapen! Rita fyra bilder som visar hur heapen successivt byggs upp. Skriv också förklarande text så det framgår hur algoritmen fungerar.

   Svar. Följande figur visar hur heapen byggs upp genom att man successivt sätter in de fyra elementen sist i heapen och sedan "bubblar" upp dem till rätt plats.



   Binärt sökträd. Här behöver du bara rita det slutgiltiga trädet. Sökträdet ska vara ordnat med avseende på nycklarnas storlek. Det ska inte balanseras.

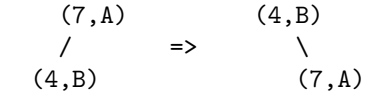
   Svar.



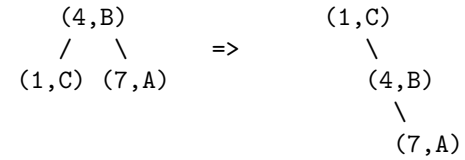
Splayträd. Rita fyra bilder som visar hur splayträdet successivt byggs upp. Bifoga förklarande text.

Svar. Först sätter vi in (7,A)

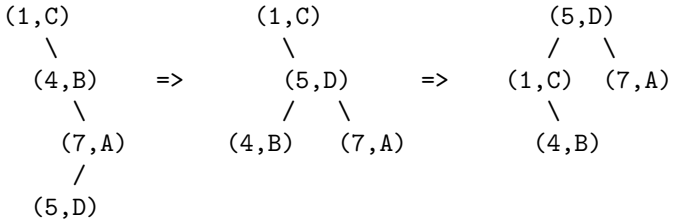
Sedan sätter vi in (4,B) och gör en "zig"-operation:



Sedan sätter vi in (1,C) och gör en "zig"-operation till:

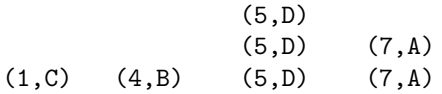


Stutligen sätter vi in (5,D) och gör en "zig-zag"-operation och en "zig"-operation



Skiplista. Här behöver du bara rita den slutgiltiga skiplistan, som är ordnad m a p nycklarnas storlek. De fyra paren ska ha multipliciteterna 2,1,1,3 respektive. Dvs paret $(7, A)$ ska ha en kopia, paren $(4, B)$ och $(1, C)$ ska inte ha några kopior, och paret $(5, D)$ ska ha två kopior.

Svar. Skiplistan ska ha följande principiella utseende (av typografiska skäl är inte länkarna inritade, men ett fullgott svar ska visa dem, se boken för fullständiga bilder på skiplistor):



(10p)

3. (a) Vad gör metoden `f` nedan? Indatafältet `a` är sorterat och har längden n.

```
public int f(int k, int[] a) {
   return g(k,a,0,a.length-1);
}

private int g(int k, int[] a, int i, int j) {
   if (i > j) return -1;
   int m = (i+j)/2;
   if (k == a[m]) return m;
   if (k < a[m]) {return g(k,a,i,m-1);};
   return g(k,a,m+1,j);
}
```

Svar. Metoden gär binärsökning efter nyckeln `k` i fältet `a`. Om nyckeln hittas returneras index där den lagrades, annars returneras -1.

(b) Vilken asymptotisk tidskomplexitet har metoden `f` i värsta fallet? Motivera!

Svar. Algoritmen arbetar rekursivt och halverar varje gång antalet element som man söker bland. (I själva verket går vi från n till $n/2$ eller $n/2 - 1$ element). Värsta fallet uppstår när man inte hittar elementet. Man måste då halvera $O(\log n)$ gånger.

(c) Vilken asymptotisk tidskomplexitet har den i bästa fallet? Motivera!

Svar. Bästa fallet är när man hittar nyckeln direkt. I detta fall exekverar man bara de tre första raderna i metoden en gång. Eftersom detta är oberoende av n blir komplexiteten $O(1)$.

(d) Vilken asymptotisk tidskomplexitet har följande Javametod?

```
public int h(int n) {
   int s = 0;
      for (i = 0; i < n; ++i) {
          for (j = i; j < n; ++j) {s += j;};
      }
   return s;
}
```

Svar. Instruktionen i den inre loopen exekveras $n + (n-1) + (n-2) + \cdots + 2 + 1$ gånger. Den exekveras alltså $O(n^2)$ gånger. Initialiseringar och uppdateringar och test av loopvariabler påverkar inte O-komplexiteten.

(10p)

4. (a) Lilla fågelboken med 100 uppslagsord och stora fågelboken med 10000 uppslagsord ligger lagrade i datorsökbar form. Det tar dubbelt så långt tid att söka i stora fågelboken som i lilla fågelboken. Vilken sökmetod har använts? Motivera.

   Svar. Eftersom logaritmen för 10000 är dubbelt så stor som logaritmen för 100 handlar det om en logaritmisk algoritm, t ex binärsökning i fält, balanserat sökträd eller skiplista.

   (b) Man bestämmer sig nu för att byta lagrings- och sökmetod. Nu tar det lika långt tid att söka i båda fågelböckerna. Vilken är den nya sökmetoden? Motivera.

   Svar. Här handlar det om en $O(1)$-algoritm, dvs det måste vara en hashtabell.

   (5p)

5. Antag att du har en graf där noderna representerar städer och bågarna representerar vägar mellan städerna och hur långa dessa vägar är. Handelsresandeproblemet ("the Travelling Salesperson Problem") består i att finna den kortaste rundturen som besöker alla noder. En rundtur är en cykel, dvs en väg med samma start- och slutnod.

   (a) Betrakta följande algoritm. Börja i en godtycklig startnod. Gå sedan till den närmaste grannen. Gå därefter till denne grannes närmaste obesökte granne. Osv.

   Ger algoritmen en korrekt lösning till handelseresandeproblemet? Om du svarar ja måste du motivera ditt svar. Om du svarar nej ska du ge ett motexempel.

   Svar. Nej. Se t ex motexemplet i bild 19 från föreläsning den 24 november 2005.

   (b) Det finns ett antal välkända algoritmdesignmetoder som togs upp i kursen. Vilken metod är ovanstående algoritm ett exempel på? Motivera.

   Svar. Det är en girig algoritm. Den väljer hela tiden det som verkar bäst för stunden, dvs den närmaste obesökta noden.

   (5p)

6. (a) Du vill sortera telefonnummerna i Göteborgskatalogen i lexikografisk ordning. Nummerna är från början helt osorterade. Antag att alla telefonnummer har 6 eller 7 siffror. Vi betraktar ett 6-siffrigt nummer som ett 7-siffigt nummer med ett sista blanktecken. Lexikografisk ordning innebär att man i första hand sorterar efter första siffran i numret, i andra hand efter andra siffran, osv. Blanktecknet betraktas som mindre än varje annan siffra. Här är ett exempel på korrekt sorterade telefonnummer.

```
405837
4058372
405839
406100
4061000
511099
```

Du ska ange två sorteringsalgoritmer som du betraktar som huvudkandidater för uppgiften och jämföra dem. Du måste motivera ditt svar med utgångspunkt från algoritmernas asymptotiska komplexiteter och att antalet telefonnummer som ska sorteras är ca 300 000.

Svar. Eftersom vi har ett stort antal osorterade element $n = 300000$ kan vi utesluta $O(n^2)$-algoritmer och begränsa oss till $O(n \log n)$ algoritmer som quicksort (vi använder oss här av tidsåtgången i medeltal) och jämföra dem med radix sort med komplexiteten $O(d(n+N))$ med $d = 7$ och $N = 11$ och bucket sort $O(n+N)$ med $N = 11000000$. Låt oss som första approximation anta att konstanterna för de tre algoritmerna är lika. Om vi t ex sätter dem alla till 1 får vi följande grova uppskattningar:

- quick sort: ca 18 x 300 000 = 5 400 000 tidsenheter
- radix sort: ca 7 x 300 000 = 2 100 000 tidsenheter
- bucket sort: ca 11 000 000 tidsenheter

För att bedöma om radix sort verkligen är bättre än quick sort och bucket sort behöver vi sedan uppskatta deras konstanter. Vi gör inte det i detalj utan nöjer oss med att konstatera att det handlar om att räkna sammanlagda tiden det tar att exekvera de instruktioner som utförs flest gånger. Vad som dominerar tidsåtgången i radix sort är att lägga elementen i rätt ”hinkar”. Vad beträffar in-place quick sort är det upprepade omkastningar av element och flytt av index som man gör när man delar upp elementen genom att jämföra dem med pivotelementet. Vad beträffar bucket sort dominerar tiden det tar att plocka ut elementen ur hinkarna. Det förefaller alltså som om radix sort har en bra konstant och sannolikt är den klart bästa algoritmen för ändamålet.

(b) Efter det att du har sorterat katalogen ska du sätta in 1000 nummer till. Vilken sorteringsalgoritm är mest effektiv för denna uppgift? Även här ska du motivera ditt svar utifrån asymptotiska komplexiteter.

Svar. Man sorterar först elementen i den nya listan med $m = 1000$ element och sedan utför man "merge"-operationen på de båda listorna. Sorteringen tar $O(m \log m)$ (eller $O(d(m + 11))$ om radix sort används). Merge-operationen har komplexiteten $O(m + n)$. Om vi som i (a) gör en grov uppskattning genom att sätta konstanterna till 1 och sätter in $m = 1000$ och $d = 7$ så ser vi att merge-operationen kommer att dominera exekveringstiden:

- radix sort: ca 7 x 1000 = 7000 tidsenheter
- merge: ca 300000 tidsenheter

Vi ser alltså att detta tillvägagångssätt är överlägset bättre än att sortera om listan.

(c) Ange två datastrukturer som är lämpliga för att lagra Göteborgskatalogen i och jämför dem. Även här ska du motivera ditt svar genom att ange asymptotiska komplexiteter för sökning, insättning och borttagning.

Svar. Bäst tidskomplexitet får vi genom att använda hashtabeller där alla tre operationerna har komplexiteten $O(1)$. Alternativet är att använda en datastruktur där de tre operationerna är $O(\log n)$, t ex balanserade sökträd eller skip listor. Splayträd är också ett bra alternativ trots $O(n)$ som värstafallskomplexitet eftersom vi kan förvänta oss att vissa telefonnummer är mycket mer eftersökta än andra.

(10p)

7. Skriv en algoritm som löser Sodoku-problem!

   Ett Sodoku-problem består av en 9 x 9 - matris där en del av cellerna är ifyllda med siffrorna 1 - 9. Att lösa ett Sodoku-problem innebär att fylla i de tomma rutorna så att

   - varje rad innehåller alla siffrorna 1-9 precis en gång.
   - varje kolumn innehåller alla siffrorna 1-9 precis en gång.
   - var och en av de nio 3 x 3-delmatriserna (som uppstår om man delar 9 x 9-matrisen i 3 x 3 delar, se figur) innehåller alla siffrorna 1-9 precis en gång.

   Du kan utgå från att det givna Sodoku-problemet har en entydig lösning. Här är ett exempel på ett Sodoku-problem.

```
- - 8   5 7 3   - 9 -
- - 5   - - -   - - 7
- 2 6   - 4 9   - 8 -

- - -   7 1 8   - - -
- - -   - - -   - - -
- - -   9 6 5   - - -

- 7 -   3 9 -   6 1 -
3 - -   - - -   4 - -
- 1 -   6 2 7   5 - -
```

   Du ska använda maximalt en sida (med normalstor text) för att beskriva en algoritm som löser detta problem. Din lösning bör tala om vilka datastrukturer och algoritmer du använder, och hur de implementeras. Om du vill kan du använda pseudokod på "hög nivå". Du kan förutsätta att algoritmer och datastrukturer från kursen är kända.

   Poängsättningen kommer att grundas på hur klar och fullständig beskrivning du ger. Det är också viktigt att du väljer de mest lämpliga datastrukturerna och algoritmerna, så att problemet kan lösas på ett effektivt sätt. (10p)

Svar. Man kan använda grafsökning för att lösa Sodoku-problem. Noderna är partiellt ifyllda 9 x 9-matriser, som uppfyller de tre kraven. Det finns en riktad båge mellan två noder om man går från den ena matrisen till den andra genom att man fyller i *en* siffra. På detta sätt kan man t ex använda djupet-först sökning för att söka efter en väg från en given startnod till en slutnod som representerar en lösning till Sodoku-problemet. Vidare bör man uppmärksamma att man inte kan bygga upp hela sökgrafen från början, eftersom det finns alltför många noder i grafen. I stället beräknar man möjliga efterföljarnoder till en given nod under sökningens gång.

För att göra sökningen effektiv kan man använda diverse regler för att minska sökrymden. Man håller lämpligen reda på vilka siffror som är möjliga att fylla en viss cell med och fyller omedelbart celler med bara en möjlighet. Det finns många intressanta regler som man kan använda för att dra slutsatser om de möjliga siffrorna för en viss cell. Om man t ex vet att två celler i samma rad båda måste fyllas med antingen 1 eller 2, vet man att övriga celler i raden vare sig kan innehålla 1 eller 2. Osv. Med hjälp av sådana regler kan man ofta helt eliminera behovet av sökning om man utgår från ett enkelt Sodoku-problem.