Tentamen
Datastrukturer D
DAT 035/INN960

22 december 2006

- Tid: 8.30 - 12.30
- Ansvarig: Peter Dybjer, tel 7721035 eller 405836
- Max poäng på tentamen: 60. (Bonuspoäng från övningarna tillkommer.)
- Betygsgränser, CTH: 3 = 30 p, 4 = 40 p, 5 = 50 p, GU: G = 30 p, VG = 50 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!
- 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? Diskutera! Det kan vara viktigare att belysa frågan på ett allsidigt sätt än att ge rätt svar. När O-komplexiteter för relevanta operationer är betydelsefull för frågans svar ska de anges.

   (a) Borttagning av *minsta* elementet i en heap tar $O(1)$ i värsta fall. (2p)

   (b) Borttagning av *minsta* elementet i ett AVL-träd tar $O(n)$ i värsta fall. (2p)

   (c) Splayträd är en mycket effektiv datastruktur för lagring av prioritetsköer. (2p)

   (d) Det är alltid bättre att använda hashtabeller än balanserade sökträd (t ex AVL-träd, rödsvarta träd, B-träd) som lagringsmedium för avbildningar ("maps"). (4p)

2. Grannmatriser kan användas för att representera grafer. De kan implementeras antingen med hjälp av *vanliga fält* som har en viss storlek eller med hjälp av *dynamiska fält* som kan utvidgas vid behov.

   (a) När är det lämpligt att använda den ena implementeringsmetoden och när är det lämpligt att använda den andra? (2p)

   (b) Förklara varför det är intressant att diskutera den *amorterade* komplexiteten i det här sammanhanget! (3p)

3. Antag att du vill lagra epostmeddelanden i en hashtabell, så att man snabbt kan leta upp ett meddelande från en viss avsändare en given dag. Konstruera en bra hashfunktion för uppgiften! (5p)

4. Betrakta aritmetiska uttryck som består av positiva heltal, operationerna + och * och parenteser. Mer precist säger vi att antingen är ett aritmetiskt uttryck ett positivt heltal n eller så har det någon av formerna $(e + e')$ eller $(e * e')$, där e och e' redan är aritmetiska uttryck. (Notera att vi alltid sätter ut parenteser kring en summa eller produkt.) Exempel på aritmetiska uttryck är $7, 12, (3 + 4), (5 * 8), ((2 * 9) + 11)$, osv.

   (a) Aritmetiska uttryck representeras lämpligen i datorn som äkta binära träd med operationerna + och * i de inre noderna och tal i löven. Rita det träd som representerar uttrycket $(4 + ((2 * 9) + 11))$! (2p)

   (b) Skriv en Javaklass som lagrar äkta binära träd som representerar aritmetiska uttryck enligt ovan. (Det räcker med att ange tillståndsvariablerna i de klasser du använder.) (4p)

   (c) Rita sedan hur uttrycket i (a) lagras i minnet som ett objekt som tillhör *din* Javaklass i (b). Du ska rita ut *alla* minnesceller som används och även deras innehåll. (2p)

   (d) Skriv en metod

   ```
   public int valueOf()
   ```

   som returnerar värdet av trädet som ligger lagrat i din Javaklass! Om `valueOf()` anropas från det objekt som lagrar trädet $(4+((2*9)+11))$ ska det alltså returnera 33. (Avdrag kommer inte att ges för smärre syntaxfel i Java. Om du skriver i Java-liknande pseudokod är det dock viktigt att den är så detaljerad och lik Java som möjligt.) (4p)

5. I kursen använde vi skiplistor för att implementera avbildningar och lexika ("maps" och "dictionaries").

   (a) Man kan också använda skiplistor för att sortera. Ge pseudokod för en sorteringsalgoritm som använder sig av skiplistor! (Pseudokoden behöver inte vara detaljerad men de viktiga stegen i algoritmen måste framgå klart.) Algoritmen ska ta ett fält med heltal som indata och returnera ett sorterat fält som utdata.

   Vilken O-komplexitet har din algoritm i värsta fallet och medelfallet? (5p)

   (b) Ytterligare en tänkbar användning av skiplistor är för att implementera mängder. Hur gör man om man vill implementera unionen av två mängder med skiplistor? Vilken komplexitet har din algoritm? Är det en bra eller dålig implementering i jämförelse med de alternativ som finns? (5p)

6. (a) I spelprogram representeras ofta spel som träd eller grafer. Varje ställning representeras som en nod, och varje drag representeras som en riktad båge från ställningen innan draget till ställningen efter draget. En del spel har oändligt många ställningar och kommer på så sätt att representeras av *oändliga* grafer. Ett exempel är *luffarschack* som spelas på ett obegränsat stort rutnät. (Spelarna turas om att markera rutor med "o" respektive "x". Den spelare vinner som först får fem rutor i rad, vågrätt, lodrätt, eller diagonalt.)

   Djupet-först och bredden-först är två populära metoder för att söka i grafer. Djupet-först är oftast den mer effektiva metoden av de två. Den har dock en viktig nackdel när den används på oändliga grafer vars noder har ändlig grad (ändligt många grannar). I så fall finns situationer då djupet-först sökning inte kommer att genomsöka alla de noder som bredden-först sökning gör. Ge ett exempel på en sådan situation! (Om du vill kan du använda den oändliga grafen i deluppgift (b), men du får gärna konstruera ett eget exempel.) (3p)

   (b) *Iterativ fördjupning* är en intressant metod som försöker kombinera fördelarna hos djupet-först och bredden-först sökning. Iterativ fördjupning består i att göra upprepade djupet-först sökningar. Första gången söker man till djupet 1, andra gången till djupet 2, osv. (Med djup menar vi här avstånd från startnoden.) På så sätt kommer metoden att finna samma noder som bredden-först sökning.

   Betrakta den oändliga graf du får genom att ha en nod för varje positivt heltal och bågar mellan talen n och $2n$ och mellan n och $2n + 1$ för alla n. Räkna upp de *sju* första noder som besöks för djupet-först sökning, bredden-först sökning, och iterativ fördjupning om du börjar med noden med talet 1! (Observera att iterativ fördjupning besöker samma nod flera gånger och att du ska räkna upp den varje gång den besöks.) (3p)

   (c) Slutligen ska du analysera den asymptotiska komplexiteten hos iterativ fördjupning i *ändliga* grafer. Gäller det alltid att iterativ fördjupning är $O(m + n)$ i en graf med n noder och m bågar? Motivera! (2p)

7. Din uppgift är att skriva effektiva algoritmer för att räkna röster i ett val. Antag att det finns n väljare och k kandidater. Vilken algoritm (och datastruktur) som är bäst att använda beror på förhållandet mellan k och n.

   (a) I vanliga val är k mycket mindre än n. Vilken algoritm och datastruktur är då lämplig att använda? Du behöver inte ge pseudokod, men måste förklara tydligt de olika stegen i algoritmen. Du ska även ange O-komplexitet för din algoritm som funktion av n och k. (5p)

   (b) Om k är mycket större än n är det dock bättre att använda en annan metod än i (a). Föreslå även här en lämplig algoritm och datastruktur, samt ange O-komplexitet för din algoritm som funktion av n och k. (5p)

   Poängsättningen kommer att bero både på hur pass effektiv din algoritm är och hur bra din komplexitetsanalys är.