Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

20 augusti 2012

TDDC70/TDDC91 Datastrukturer och algoritmer
Tentamen 2012-08-20, 14–19 (TER1)

Examinator: Tommy Färnqvist
Jour: Tommy Färnqvist (telefon 070 4547668).
Max poäng: 26 poäng (betyg 5 = 23p, 4 = 18p, 3 = 13p)
Hjälpmedel: INGA HJÄLPMEDEL TILLÅTNA!!!

VÄNLIGEN IAKTTAG FÖLJANDE

- Lösningar till olika problem skall placeras enkelsidigt på separata blad. Skriv inte två lösningar på samma papper.
- Sortera lösningarna innan de lämnas in.
- MOTIVERA DINA SVAR ORDENTLIGT: avsaknad av, eller otillräckliga, förklaringar resulterar i poängavdrag. Även felaktiga svar kan ge poäng om de är korrekt motiverade.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.
- SE TILL ATT DINA LÖSNINGAR/SVAR ÄR LÄSBARA.
- Lämna plats för kommentarer.

Lycka till!

1. Vilka av följande påståenden är sanna och vilka är falska? Svar utan motivering ger inga poäng. (2 p)

   (a) En lista L består av n heltal. Alla tal i listan ligger mellan 1 och n. Det går att sortera dessa n tal i tid $O(n)$. (1)

   (b) $n^{1+\epsilon/\sqrt{\log n}} \in O(n \log n)$, $\epsilon > 0$. (1)

2. Antag att du behöver generera en *slumpmässig* permutation av heltalen från 1 till N. Exempelvis är $\{4, 3, 1, 5, 2\}$ och $\{3, 1, 4, 2, 5\}$ giltiga permutationer, men $\{5, 4, 1, 2, 1\}$ är inte en giltig permutation eftersom ett tal saknas och ett tal förekommer två gånger. En sådan metod kommer ofta till användning i olika typer av simuleringar. Vi antar att vi har tillgång till en slumptalsgenerator, `r`, med metoden `randInt(i,j)`, som generar heltal mellan `i` och `j` med likformig sannolikhet. Här är tre algoritmer: (6 p)

   1. Fyll arrayen `a` från `a[0]` till `a[N-1]` enligt följande: För att fylla i `a[i]`, generera slumptal till ett dyker upp som inte redan finns i `a[0]`, `a[1]`, ..., `a[i-1]`.
   2. Samma som algoritm (1), men håll reda på en extra array `used`. När ett slumptal `ran` först sätts in i `a`, sätt `used[ran]=true`. Detta betyder att när `a[i]` ska fyllas i med ett slumptal kan vi testa i ett steg om slumptalet redan använts.
   3. Fyll i arrayen så att `a[i]=i+1` håller. Gör sedan följande:

      ```
      for ( int i = 1; i < N; i++ ) {
         swap( a[i], a[ randInt( 0, i )] );
      }
      ```

   Det är inte svårt att se att alla tre algoritmer ovan genererar giltiga permutationer. De första två algoritmerna har tester som garanterar att inga dubletter genereras och den tredje algoritmen blandar om i en array som initialt är fri från dubletter. Det är också lätt att se att de två första algoritmerna är helt slumpmässiga och att varje permutation är lika sannolik. Den tredje algoritmen, konstruerad av R. Floyd, är inte lika lätt att förstå sig på, men det går att visa att även denna algoritm är helt slumpmässig med hjälp av induktion.

   (a) Gör en asymptotisk analys av den *förväntade* körtiden för varje algoritm. (4)

   (b) Vad är exekveringstiden i värsta fall för respektive algoritm? (2)

3. Vi använder en array med indexen 0 till 6 för att implementera en öppet adresserad hashtabell av längd 7 med följande tabell som hashfunktion: (4 p)

nyckel	hashvärde
A	5
B	2
C	5
D	1
E	4
F	1
G	3

   Antag att linjär sondering (*linear probing*) används för att hantera kollisioner.

   (a) Visa hur arrayen ser ut efter att nycklarna har satts in i alfabetisk ordning: A, B, C, D, E, F, G. (2)

   (b) Vilka av följande skulle kunna vara innehållet i arrayen efter att nycklarna har satts in i någon ordning? (2)

I.

0	1	2	3	4	5	6
A	F	D	B	G	E	C

II.

0	1	2	3	4	5	6
F	A	D	B	G	E	C

III.

0	1	2	3	4	5	6
C	A	B	G	F	E	D

4. Länkade listor (3 p)

   (a) Skriv pseudokod för en algoritm som byter plats på två element som ligger precis bredvid varandra i en enkellänkad lista genom att justera länkarna (och inte datat som lagras). (1.5)

   (b) Skriv pseudokod för en algoritm som byter plats på två element som ligger precis bredvid varandra i en dubbellänkad lista genom att justera länkarna (och inte datat som lagras). (1.5)

5. Binära träd (4 p)

   (a) Låt T vara ett fullt binärt träd med rot r. Betrakta följande algoritm: (1.5)

   Input: Rot r av ett fullt binärt träd
   Output: ?
   function TRAVERSE(r)
   if r är ett löv **then return** 0
   else
   $t \leftarrow$TRAVERSE(vänster barn till r)
   $s \leftarrow$TRAVERSE(höger barn till r)
   return $s + t + 1$

   Vad gör algoritmen?

   (b) Låt T vara ett fullt binärt träd med 7 noder: a, b, c, d, e, f, g. En preordertraversering av T besöker noderna i ordningen b, g, d, a, c, f, e. En inordertraversering av T besöker noderna i ordningen d, g, c, a, f, b, e. Finns det några noder som är på avstånd 3 från roten av T? Vilka i så fall? (**Tips**: I trädet T är noden d vänster barn till g.) (1.5)

   (c) En mängd $K = \{k_1, \ldots, k_n\}$ av $n \geq 1$ nycklar ska lagras i ett AVL-träd. Vilket av följande påståenden är alltid sant? (1)

   (A) Om k_i är den minsta nyckeln i K så kommer vänster barn till noden som lagrar k_i i varje AVL-träd som lagrar K vara ett löv.

   (B) Alla AVL-träd som lagrar K har exakt samma höjd oavsett vilken ordning nycklarna sätts in i trädet.

   (C) En preordertraversering av ett AVL-träd som lagrar K besöker noderna i ökande nyckelordning.

   (D) I varje AVL-träd som lagrar K är nyckeln som lagras i roten av trädet samma, oavsett ordningen nycklarna sätts in i trädet.

   (E) Inget av påståendena ovan är alltid sant.

6. Givet två arrayer `a[]` och `b[]`, innehållandes M respektive N punkter i planet (med $N \geq M$), konstruera en algoritm som avgör hur många punkter som förekommer i båda arrayerna. Exekveringstiden för din algoritm ska vara proportionell mot $N \log M$ i värsta fallet och får (4 p)

bara använda en konstant mängd extra minne. Motivera att din algoritm klarar av dessa krav.

7. Rita en enkel, sammanhängande och viktad graf med 8 noder och 16 bågar, där varje båge har en unik (icke-negativ) bågvikt. Identifiera en nod som startnod och illustrera en exekvering av Dijkstras algoritm på din graf. (3 p)