Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

10 januari 2011

TDDC70/TDDC91 Datastrukturer och algoritmer
Tentamen 2011-01-10, 08–13 (T1, T2)

Examinator: Tommy Färnqvist
Jour: Tommy Färnqvist (telefon 070 4547668).
Max poäng: 29 poäng (betyg 5 = 25p, 4 = 20p, 3 = 14p)
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. Bevisa följande: (4 p)

   (a) Om x och y är reella tal sådana att $0 < x < y$, så gäller $n^x \in O(n^y)$ och $n^y \notin O(n^x)$. (2)

   (b) Logaritmisk tid beror inte på vilken bas som används för logaritmen; d.v.s. $\log_a(n) \in O(\log_b(n))$ för reella tal $a > 1$ och $b > 1$. (2)

2. Ett polynom av grad n är ett uttryck på formen (5 p)

$$a_0 + a_1 x + \cdots + a_n x^n$$

   där $a_0, \ldots, a_n$ är konstanter, $a_n \neq 0$ och x är en variabel över $\mathbb{R}$.

   (a) Ge pseudokod för en algoritm som, utgående från uttrycket ovan, beräknar värdet av ett polynom av grad n för ett givet värde på x. Analysera tidskomplexiteten för din algoritm under antagandet att x^i inte är en primitiv operation utan måste beräknas genom multiplikation. (2)

   (b) Beskriv en annan algoritm för att, under samma antaganden, beräkna värdet av ett polynom av grad n i tid $\Theta(n)$. Kan du koppla lösningen till ett av de algoritmiska paradigm som diskuterats i kursen? (3)

3. Antag att du har fått uppgiften att implementera ADT Stack med enbart två köer Q_1 och Q_2 som instansvariabler. (4 p)

   (a) Beskriv på svenska eller med pseudokod hur du skulle implementera metoderna push() och pop(). (2)

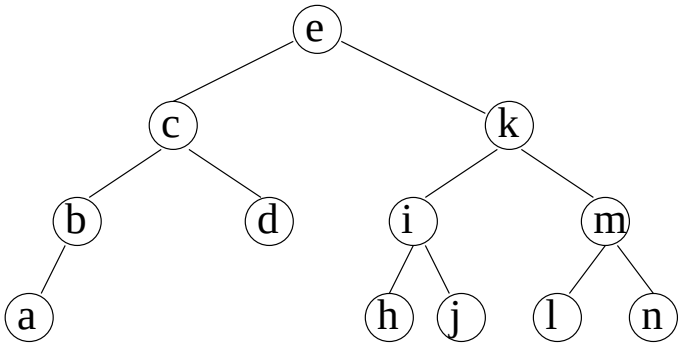
   (b) Karakterisera den asymptotiska exekveringstiden för dina implementationer av push() och pop() som funktioner av antalet element i stacken n. (2)

4. Den här uppgiften handlar om sökträd. (4 p)

   (a) I en sökning efter ett element x i ett binärt sökträd kallas listan av noder x jämförs med *nyckelsekvensen* för sökningen. (1)

   Antag att vi har talen från 1 till 1000 insatta i ett binärt sökträd och att vi vill söka efter talet 363. Förklara varför följande *inte* kan vara nyckelsekvensen för sökningen: 925, 202, 911, 240, 912, 245, 363.

   (b) Betrakta AVL-trädet T nedan. Utför följande operationer: $T_1 = Insert(g, T)$, $T_2 = Insert(f, T_1)$, $T_3 = Delete(d, T_2)$, $T_4 = Delete(j, T_3)$. Visa hur trädet ser ut efter varje operation samt ev. mellanliggande omstruktureringssteg. (3)



5. Den här uppgiften handlar om hashning. (6 p)

   (a) Rita en representation av en öppet adresserad hashtabell H med 11 celler och dess innehåll efter att vi använt avbildningen (3)

   $$h(i) = (2i + 5) \bmod 11$$

   för att sätta in nycklarna $12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5$ i den initialt tomma H. Antag att vi hanterar kollisioner med dubbel hashning. Använd

   $$h'(i) = 7 - (i \bmod 7)$$

   som sekundär hashfunktion.

   (b) Visa, för allmänna hashfunktioner $h(i)$ och $h'(i)$, att om hashtabellens kapacitet N inte är ett primtal är det möjligt att dubbel hashning inte hittar en tom cell även om en sådan existerar. *Tips:* Fundera på vilka villkor som måste vara uppfyllda för att en cell ska testas två gånger. (3)

6. Följande kodsnutt upphittades i en datorsal: (4 p)

```
public Object foo (Sequence S, Comparator C, int k, int a, int b)
{
  int c = partition(S,C,a,b);
  if (c == k) { return S.elemAtRank(k); }
  else if (c < k) { return foo(S,C,k,c+1,b); }
  else { return foo(S,C,k,a,c-1); }
}

public int partition (Sequence S, Comparator C, int left_bound, int right_bound)
{
  Object pivot = S.atRank(right_bound).element();
  int left_index = left_bound, right_index = right_bound-1;
  while (left_index <= right_index) {
    while ((left_index <= right_index) &&
           C.isLessThanOrEqualTo(S.atRank(left_index).element(),pivot)) {
      left_index++;
    }
    while ((right_index >= left_index) &&
           C.isGreaterThanOrEqualTo(S.atRank(right_index).element(),pivot)) {
      right_index--;
    }
    if (left_index < right_index) {
      S.swap(S.atRank(left_index),S.atRank(right_index));
    }
  }
  S.swap(S.atRank(left_index),S.atRank(right_bound));
  return left_index;
}
```

Olyckligtvis verkar inte författaren vara DALG-student, eftersom det inte finns några kommentarer och vissa variabelnamn är olyckligt valda. Kodens andra del känns lätt igen som partitioneringssteget i Quicksort, men den första delen är lite mystisk. Din uppgift är att lista ut vad den här algoritmen gör genom att göra följande saker:

   (a) Stega genom en exekvering av `foo(`S`,`C`,5,0,7)`, där S är sekvensen (C, H, B, E, J, G, D, A) och C är en jämförelseoperator för bokstäver som ordnar (2)

dem i alfabetisk ordning. Det första steget visas nedan; fortsätt genom att visa alla anrop till `foo` och `partition` och deras returvärden samt när platsbyten sker och hur de förändrar S. Vilket värde returneras till slut av anropet `foo(`S`,`C`,5,0,7)`?

```
foo(S,C,5,0,7)                      // S is (C,H,B,E,J,G,D,A)
  partition(S,C,0,7)
    swap elements at ranks 0 and 7 // S is now (A,H,B,E,J,G,D,C)
    return 0
  foo(S,C,5,1,7)                    // c is 0; choose c < k case since 0 < 5
```

(b) Vad gör `foo(`S`,`C`,`k`,`a`,`b`)` i allmänhet? Antag att k, a och b har giltiga värden. (2)

7. Använd Dijkstras kortaste väg-algoritm för att hitta den kortaste restiden från Providence till de andra städerna i grafen nedan. Bågvikterna är ungefärliga restider. Visa längden av de bästa vägarna funna så långt för varje nod efter varje relaxeringssteg. (2 p)