Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

25 augusti 2014

TDDI16 Datastrukturer och algoritmer
Tentamen (TEN1)
2014-08-25, 8–12 (TER3)

Examinator: Tommy Färnqvist
Jour: Tommy Färnqvist (telefon 070 4547668).
Max poäng: 19 poäng (betyg 5 = 17p, 4 = 14p, 3 = 10p)
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. Algoritmanalys (2 p)

Följande (Java-)kodsnutt är tänkt att köras på bakteriella genom som är ungefär 1 megabyte långa. Vad är den asymptotiska tidskomplexiteten för kodsnutten?

```
int N = Integer.parseInt(args[0]);
String[] genomes = new String[N];
for (int i = 0; i < N; i++) {
   In gfile = new In("genomFile" + i + ".txt");
   genomes[i] = gfile.readString();
}

for (int i = 1; i < N; i++) {
   for (int j = i; j > 0; j--) {
      if (genomes[j-1].length() > genomes[j].length())
         exch(genomes, j-1, j);
      else break;
   }
}
```

2. Binära sökträd (2 p)

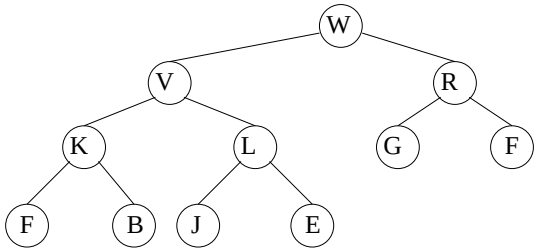
Förklara vad funktionen `mystery()` gör.

```
public boolean mystery() {
   return mystery(root, nullptr, nullptr);
}

private boolean mystery(Node* x, Key* a, Key* b) {
   if (x == nullptr) return true;
   if (a != nullptr && *(x->key) <= (*a)) return false;
   if (b != nullptr && *(x->key) >= (*b)) return false;
   return mystery(x->left, a, x->key) &&
          mystery(x->right, x->key, b);
}
```

3. Nycklarna i den här uppgiften om binära heapar är stora bokstäver och vi använder bokstavsordning för att sortera dessa nycklar. (2 p)
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Betrakta följande max-heap representerad som ett binärt träd.



Visa hur en arrayrepresentation av heapen ser ut efter ett anrop till `removeMax` har utförts. (Du får själv välja om du vill använda en 0-indexerad eller 1-indexerad arrayrepresentation.)

4. Kolumnen nedan till vänster innehåller strängar som ska sorteras, kolumnen nedan till höger är strängarna i sorterad ordning, övriga kolumner visar innehållet vid något mellanliggande steg i ett anrop till de fyra sorteringsalgoritmerna listade nedan. Para ihop varje algoritm med rätt kolumn. Använd varje algoritm exakt en gång. (2 p)

```
HELP AMTR AMTR WATC AMTR AMTR
IFYO APPE APPE SWAR APPE APPE
UARE DINS DINS UARE AREW AREW
READ EASE HELP SEND DINS DINS
INGT HELP HISI SORT EASE EASE
HISI HISI IDEA THEP ETYP ETYP
AMTR IDEA IFYO RATS EVIL EVIL
APPE IFYO INGA READ HELP HELP
DINS INGA INGT RATS HELP HELP
IDEA INGT READ RATS HELP HELP
SORT ITHM SORT MALL HISI HISI
INGA LGOR UARE OVER IDEA IDEA
LGOR OHPL LGOR LGOR LGOR IFYO
ITHM READ ITHM ITHM ITHM INGA
OHPL SORT OHPL OHPL OHPL INGM
EASE UARE EASE ETYP INGT INGT
SEND EVIL SEND APPE SEND ITHM
HELP HELP HELP HELP IFYO LACE
RATS MALL RATS DINS RATS LGOR
EVIL OVER EVIL EVIL READ MALL
RATS RATS RATS IDEA RATS OHPL
SWAR RATS SWAR INGT SWAR OVER
MALL SEND MALL IFYO MALL RATS
OVER SWAR OVER HISI OVER RATS
THEP AREW THEP INGA THEP RATS
LACE ETYP LACE LACE LACE READ
HELP HELP HELP HELP INGA SEND
RATS INGM RATS HELP RATS SORT
AREW LACE AREW AREW UARE SWAR
WATC RATS WATC AMTR WATC THEP
INGM THEP INGM INGM INGM UARE
ETYP WATC ETYP EASE SORT WATC
-----------------------------
1:a  2:   3:   4:   5:   6:b
```

(a) Indata

(b) Sorterad sekvens

(c) Selection-sort

(d) Insertion-sort

(e) Merge-sort

(f) Heap-sort (in-place)

5. Metoden `addBlock()` används för att lägga till M element av en typ med jämförelseoperator till en existerande sorterad datamängd innehållande N element av samma typ, där M är mycket mindre än N. (4 p)

DALG-studenten Frankie Halfbean gör två val. För det första väljer hen en sorterad array/vektor som datastruktur. För det andra väljer hen insertion sort som huvudalgoritm eftersom, förklarar hen, insertion sort är väldigt snabb för nästan sorterade arrayer. För att lägga till ett nytt block med M element skapar algoritmen helt enkelt en array av längd $N + M$, kopierar över de N gamla elementen till den nya arrayen, kopierar in de M nya elementen till slutet av arrayen och använder till sist insertion sort för att ordna alla element.

(a) Vad är den asymptotiska exekveringstiden för `addBlock()` i värsta fallet som en funktion av N och M? (1)

(b) Föreslå ett sätt att uppnå en bättre asymptotisk exekveringstid för `addBlock()` i värsta fallet. För full poäng krävs ett sätt som använder optimal tid och minne (inom konstanta faktorer). (3)

6. Symboltabeller (7 p)

(a) För var och en av symboltabellstillämpningarna nedan, välj den bästa symboltabellsimplementationen (A–F). (3)

1. Uppslagstabell för att beräkna $\sin(\theta)$, där θ är en av 1000000 möjliga vinklar jämnt fördelade mellan 0 och π.
2. Databas som avbildar ljuddata (från en fil) på artistnamn.
3. Snabbaste garantierna för `insert`, `delete` och `find` för en godtycklig uppsättning numeriskt data.

(A) Vanligt BST	(C) Hashtabell	(E) Oordnad array/vektor
(B) AVL-träd	(D) Ordnad array/vektor	(F) Heap

(b) Vi använder en array med indexen 0 till 8 för att implementera en hashtabell där kollisioner hanteras med hjälp av separat länkning (*separate chaining*). Hashfunktionen som används är $h(k) = k \bmod 9$. Visa en representation av tabellen efter att nycklarna 37, 13, 71, 25, 53, 7 satts in i den initialt tomma tabellen. (2)

(c) Vi använder en array med indexen 0 till 8 för att implementera en hashtabell där kollisioner hanteras med hjälp av linjär sondering (*linear probing*). Hashfunktionen som används är $h(k) = k \bmod 9$. Visa en representation av tabellen efter att nycklarna 37, 13, 71, 25, 53, 7 satts in i den initialt tomma tabellen. (2)