Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

19 december 2014

TDDD86
Datastrukturer, algoritmer och programmeringsparadigm
Datortentamen (DAT1)
2014-12-19, 14–19

Examinator:	Tommy Färnqvist
Jour:	Tommy Färnqvist (telefon 013-282479).
Max poäng:	109 poäng (betyg 5 = 108p, 4 = 105p, 3 = 100p)
Hjälpmedel:	Inga hjälpmedel tillåtna, förutom OpenDSA!

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. Efter inloggning i datortentasystemet finns i startmenyn för Linux Mint: (100 p)

   - "OpenDSA" (öppnar URL för tentamensversion av OpenDSA i Chrome) och
   - "Inloggningsuppgifter OpenDSA" (öppnar sida med inloggningsuppgifter till tentamensversion av OpenDSA i Chrome).

   Starta tentamensversionen av OpenDSA, logga in med inloggningsuppgifterna enligt ovan och lös de anvisade uppgifterna. Kapitel 0.1 i tentamensboken anger vilka OpenDSA-övningar som måste genomföras. Genom att klicka på ditt inloggningsnamn kan du se i betygsboken hur många av de poänggivande uppgifterna du löst hittills. Om du inte löser samtliga poänggivande uppgifter får du noll poäng på den här tentamensuppgiften.

2. Följande (Java-)kodsnutt är tänkt att köras på bakteriella genom som är ungefär 1 megabyte långa. Vad är den asymptotiska tidskomplexiteten för kodsnutten? (2 p)

```
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())
         swap(genomes, j-1, j);
      else break;
   }
}
```

3. Metoden `addBlock()` används för att lägga till M element av typen `Comparable` till en existerande sorterad datamängd innehållande N `Comparable`-element, där M är mycket mindre än N. (4 p)

   DALP-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 `Comparable`-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)

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

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