Några svar till TDDC70/91 Datastrukturer och algoritmer

2012-08-20

Följande är lösningsskisser och svar till uppgifterna på tentan. Lösningarna som ges här ska bara ses som vägledning och är oftast inte tillräckliga som svar på tentan.

1. (a) **Sant**. Använd räknesortering.

   (b) **Falskt**. Antag motsatsen. Då växer $n^{\epsilon/\sqrt{\log n}}$ långsammare än $\log n$. Logaritmera $\Rightarrow$ $\log n \cdot \epsilon/\sqrt{\log n}$ växer långsammare än $\log\log n$. Men $\log n \cdot \epsilon/\sqrt{\log n} = \epsilon \cdot \sqrt{\log n}$. Om vi sätter $L = \log n$ får vi att $\epsilon\sqrt{L}$ växer långsammare än $\log L$ eller, ekvivalent, att ϵL växer långsammare än $\log^2 L$, men $\log^2 L$ växer strikt långsammare än L, så vårt ursprungliga antagande måste vara falskt.

2. (a) Algoritm 1: Iteration i: $O(i)$ tid för att kolla om tal ej redan finns. Förväntat antal slumptal som behöver genereras för `a[i]` är $N/(N-i)$ eftersom i av talen är dubletter (och sannolikheten att få icke-dublett är $(N-1)/N$). Vi får

$$\sum_{i=0}^{N-1} \frac{Ni}{N-i} \leq \sum_{i=0}^{N-1} \frac{N^2}{N-i} \leq N^2 \sum \frac{1}{N-i} \leq N^2 \sum_{j=1}^{N} \frac{1}{j} \in O(N^2 \log N).$$

   $\sum_{j=1}^{N} \frac{1}{j} = H_N$, det N:te harmoniska talet. Antingen känner man till att $\lim_{n\to\infty}(H_n - \ln n) = \gamma \approx 0.5772\ldots$ eller så kan man använda att $\int_1^N \frac{1}{x}\mathrm{d}x = \ln n$.
   Algoritm 2: Sparar faktor i för varje position $\Rightarrow$ $O(n \log n)$ förväntad tid.
   Algoritm 3 är alltid linjär.

   (b) Det finns ingen begränsning på värstafallstiden för algoritm 1 och 2 eftersom det alltid finns en nollskiljd sannolikhet att programmet inte terminerat vid en given tid T.

3. (a)

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

   (b) I är resultatet av insättning i ordning F, D, B, G, E, C, A. II är omöjlig, då första nyckeln som sätts in hamnar på en plats i tabellen som motsvarar dess hashvärde, men ingen nyckel har denna egenskap. III är omöjlig — B och G är i rätt position, så vi kan anta att de sattes in först. Men då skulle den tredje nyckeln också vara i rätt position.

4. (a)

```
tmp <- list.first()        |elem|next|--->|elem|next|--->
if (tmp == i1)                i1              i2
  i1.next <- i2.next
  i2.next <- i1
  list.setFirst(i2)
else
  while (tmp.next != i1)
    tmp = tmp.next
  tmp.next <- i2
  i1.next <- i2.next
  i2.next <- i1
```

(b) Pseudokod ej inkluderad.

5. (a) Algoritmen beräknar antalet interna noder i trädet.

   (b)

```
     b
   /   \
  g     e
 / \
d   a
   / \
  c   f
```

   (c) (E)

6. Sortera `a[]` med heapsort (genom att använda någon naturlig ordning för punkterna). För varje punkt `b[j]` använd binärsökning för att leta efter punkten i den sorterade arrayen `a[]`. Körtiden är $M \log M$ för sorteringen och $N \log M$ för de N binärsökningarna. Både heapsort och binärsökning kan göras in-place.

7. Lösning ej inkluderad.