Några svar till
TDDI16 Datastrukturer och algoritmer

2012-10-23

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) **Falskt**. Ett motexempel: $f(n) = g(n) = n^2$ och $h(n) = n^3$.

   (b) **Sant**. Enligt definitionen av Ω behöver vi hitta en reell konstant $c > 0$ och en heltalskonstant $n_0 \geq 1$ sådana att $n \log_2(n) \geq cn$ för $n \geq n_0$. Valet $c = 1$ och $n_0 = 2$ visar att $n \log_2(n) \geq cn$ för $n \geq n_0$ eftersom $\log_2(n) \geq 1$ i det intervallet.

   (c) **Sant**. $3^{\log(n^2)} = 3^{2\log(n)} = 9^{\log(n)} = n^{\log 9}$ som är polynomiskt.

2. (a)
```
res = 0;
for (i=0; i <= n; i++) {
   ai = a[i];
   xi = 1;
   for (j=0; j < i; j++) {
      xi = xi*x;
   }
   res = res + ai*xi;
}
return res;
```
   Vi gör $n + 1$ additioner och $1 + 2 + 3 + \ldots + (n + 1) = \frac{(n+1)(n+2)}{2}$ multiplikationer. Tidskomplexiteten blir $\Theta(n^2)$.

   (b) x^2 beräknas $n-2$ gånger, x^3 beräknas $n-3$ gånger, $\ldots$, x^i beräknas $n-i$ gånger. Undvik detta genom att skriva polynomet på form $a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + a_n x)\ldots))$.

```
res = a[n]*x;
for (i=n-1; n >= 1; i--) {
   res = res + a[i];
   res = res*x;
}
res = res + a[0];
return res;
```

3. (a)
```
5
5 3
5
5 2
5 2 8
5 2
5
5 9
5 9 1
```

```
5 9
5 9 7
5 9 7 6
5 9 7
5 9
5 9 4
5 9
5
```

(b)
```
5
5 3
3
3 2
3 2 8
2 8
8
8 9
8 9 1
9 1
9 1 7
9 1 7 6
1 7 6
7 6
7 6 4
6 4
4
```

Algorithm 0.1: REVERSE(Q)

(c)
```
S ← en tom stack
while ! Q.isEmpty()
  do S.push(Q.dequeue())
while !S.isEmpty()
  do Q.enqueue(S.pop())
```

4. (a)

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

(b) Endast I är möjlig.

- I är resultatet av insättning av nycklarna i ordning B D G A C E G.
- II är inte möjlig. Första nyckeln som sätts in hamnar på en plats i tabellen som motsvarar dess hashvärde. Men ingen nyckel har den egenskapen.
- III är inte möjlig. Både A och F hamnar på platser i tabellen som motsvarar deras hashvärden, så vi kan anta att de sattes in som första och andra element. Det betyder att den tredje nyckeln som sattes in också måste ha hamnat på en plats som motsvaras av dess hashvärde. Men inga nycklar (förutom A och F) har denna egenskap.

5. (a)

0	1	2	3	4	5	6	7	8	9	10	11	12
-	Y	X	H	G	T	C	A	F	B	Q	R	-

(b) Endast den den slutliga arrayrepresentation redovisas här.

0	1	2	3	4	5	6	7	8	9	10	11	12
-	Y	X	*P	G	T	*H	A	F	B	Q	R	*C

(c) `H I J K L M N`
Nyckeln är $\leq$ `N` eftersom den är ett barn till `N` i ursprungsheapen och $\geq$ `H` eftersom den är förälder till `H` i heapen efer anropet till `removeMax`.

6. Vi använder en algoritm som liknar binärsökning, där vi jämför nyckeln i mitten med nyckeln som ligger precis till höger om mittennyckeln. Beroende på resultatet av den här jämförelsen fortsätter vi rekursivt i vänster eller höger delarray (som också är bitonisk).

```
int max(int[] a, int lo, int hi) {
   if (hi == lo) return a[hi];
   int mid = lo + (hi - lo) / 2;
   if      (a[mid] < a[mid+1]) return max(a, mid+1, hi);
   else if (a[mid] > a[mid+1]) return max(a, lo, mid);
   else                        return a[mid];
}
```

Invarianten vi har är att delarrayen vi arbetar i innehåller den största nyckeln. I varje steg minskar storleken på delarrayen med en faktor 2, så antalet jämförelser blir logaritmiskt i N.