Några svar till
TDDC70/91 Datastrukturer och algoritmer

2011-01-10

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) Eftersom $x < y$, $n^x < n^y$ för $n \geq 1$. Alltså gäller $n^x \leq c\ n^x$ för alla $n \geq n_0 = 1$ och $c = 1$, dvs $n^x \in O(n^y)$.
   För alla $c > 0$ gäller $cn^x < n^y$ för $n > c^{1/(y-x)}$. Alltså finns inga värden $c > 0$ och $n_0 \geq 0$ s.a. $n^y < c\ n^x$. Alltså gäller inte $n^y \in O(n^x)$.

   (b) Enligt logaritmlagarna gäller för positiva tal $a, b \neq 1$ och alla $n > 0$ att

   $$\log_a n = \log_b n\ \log_a b.$$

   Välj $c = \log_a b$ och $n_0 = 1$. Vi får $\log_a n \leq c \log_b n$ för $n \geq n_0$ och alltså $\log_a n \in O(\log_b n)$.

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;
```
   Tidskomplexitet $\Theta(n^2)$.

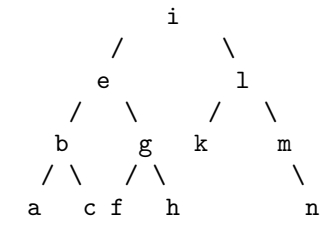
   (b)
```
res = a[n] * x;
for (i = n-1; i >= 1; i--) {
  res = res + a[i];
  res = res * x;
}
res = res + a[0] ;
return res;
```
   Detta är ett exempel på dynamisk programmering.

3. Push: gör enqueue på Q_1. Tar $O(1)$ tid.

   Pop: dequeue:a alla element i Q_1 och enqueue:a dem i Q_2, förutom sista elementet som vi sparar i en temporär variabel. För tillbaka elementen till Q_1 genom att dequeue:a dem från Q_2 och enqueue:a dem i Q_1. Returnera temporärvariabelns element. Pop kommer att ta $\Theta(n)$ tid.

4. (a) 240 finns efter 911, så vi letar i vänster delträd till 911. Där är alla element mindre än eller lika med 911 så alla nycklar i sekvensen måste vara mindre än 911. Men 912 besöks efter 240 vilket är en motsägelse.

(b) Endast T_4 visas nedan:



5. (a) Lista först ut $h(i)$ och $h'(i)$ för varje nyckel: Sedan hashtabellen:

key	$h(i)$	$h'(i)$
12	7	2
44	5	5
13	9	1
88	5	3
23	7	5
94	6	4
11	5	3
39	6	3
20	1	1
16	4	5
5	4	2

slot	key
0	11
1	23
2	20
3	16
4	39
5	44
6	94
7	12
8	88
9	13
10	5

(b) The buckets tested under double hashing are $h(i) \bmod N$, $h(i) + h'(i) \bmod N$, $h(i) + 2h'(i) \bmod N$, ... Let $0 < j \leq N$ be the smallest integer such that $jh'(i) \bmod N = 0$ (i.e., $jh'(i)$ is a multiple of N). Also let b_k be the bucket visited at the kth step. Then two important properties are true:

$$b_k = b_{k+j} = b_{k+2j} = \ldots$$

and

$$b_k \neq b_l \text{ for all } k < l < j$$

In other words, each bucket is repeated after exactly j steps and not before.

To show that buckets are skipped, all that is needed is to show that if N is not prime, there is at least one value of $h'(i)$ for which $j < N$ because then for the key i, it is not possible for all the buckets to be visited before the first bucket is repeated. If N is not prime then it has at least one factor n other than 1 and N — when $h'(i) = n$, then $j = N/n$ is an integer less than N and $jh'(i) = N$ is a multiple of N. (As a side note, if N is prime, then we need to observe that if $jh'(i)$ is a multiple of N, there is an integer $0 < k < N$ such that $jh'(i) = kN$. Since $h'(i) < N$, this works out only if $h'(i) = k$ and $j = N$. Since no bucket is repeated until after N tests are made, they must all be visited.)

6. (a)
```
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
    partition(S,C,1,7)
      swap elements at ranks 1 and 2    // S is now (A,B,H,E,J,G,D,C)
      swap elements at ranks 2 and 7    // S is now (A,B,C,E,J,G,D,H)
      return 2
    foo(S,C,5,3,7)                      // c is 2; choose c < k case since 2 < 5
```

```
      partition(S,C,3,7)
        swap elements at ranks 4 and 6 // S is now (A,B,C,E,D,G,J,H)
        swap elements at ranks 6 and 7 // S is now (A,B,C,E,D,G,H,J)
        return 6
      foo(S,C,5,3,5)                   // c is 6; choose c > k case since 6 > 5
        partition(S,C,3,5)
          swap elements at ranks 7 and 7   // S is now (A,B,C,E,D,G,H,J)
          return 5
      return G                         // c is 5; choose c = k case since 5 = 5
    return G
  return G
return G
```

Resultatet av `foo(`S`,`C`,5,0,7)` är G.

(b) Om $a = 0$ och $b = S$.size() $- 1$ så returnerar `foo(`S`,`C`,`k`,`a`,`$b`) elementet vid rank k i den sorterade sekvensen. (Annars finns två möjligheter. Om $k < a$ eller $k > b$ får vi en krasch. Om $a \leq k \leq b$ returneras elementet vid rank $k - a$ i den sorterade sekvensen bestående av element med rank mellan a och b i indatat S.)

7. Tabellvärdena är kortaste restiden från Providence till var och en av städerna i vänstra kolumnen. Restider visas med fet stil när slutvärdet fixerats.

lägg till →		PVD	Wor	Fitch	Hart	Spring	Green	North	Can	GtB	Lenox	Wmstown
Canaan	∞	∞	∞	∞	2:39	2:39	2:39	2:39	**2:39**	**2:39**	**2:39**	**2:39**
Fitchburg	∞	∞	1:25	**1:25**	**1:25**	**1:25**	**1:25**	**1:25**	**1:25**	**1:25**	**1:25**	bf 1:25
Great Barrington	∞	∞	∞	∞	∞	3:05	3:05	3:05	2:56	**2:56**	**2:56**	**2:56**
Greenfield	∞	∞	2:13	2:13	2:13	2:13	**2: 13**	**2:13**	**2:13**	**2:13**	**2:13**	**2:13**
Hartford	∞	1:38	1:38	1:38	**1:38**	**1:38**	**1:38**	**1:38**	**1:38**	**1:38**	**1:38**	**1:38**
Lenox	∞	∞	∞	∞	∞	2:57	2:57	2:57	2:57	2:57	**2:57**	**2:57**
Northampton	∞	∞	∞	∞	∞	2:24	2:24	**2:54**	**2:54**	**2:54**	**2:54**	**2:54**
Providence	**0:00**	**0:00**	**0:00**	**0:00**	**0: 00**	**0:00**	**0:00**	**0:00**	**0:00**	**0:00**	**0 :00**	**0:00**
Springfield	∞	∞	2:00	2:00	2:00	**2:00**	**2:00**	**2:00**	**2:00**	**2:00**	**2:00**	**2:00**
Williamstown	∞	∞	∞	∞	∞	$\in fty$	3:20	3:20	3:20	3:20	3:20	**3:20**
Worcester	∞	0:50	**0:50**	**0:50**	**0:50**	bf 0:50	**0:50**	**0:50**	**0:50**	**0:50**	**0:50**	**0:50**