CS21003 Algorithms I, Autumn 2011–12

Class test 2

Maximum marks: 20 | Date: 08-Nov-2011 | Duration: 1 hour

Roll no: ____________________ **Name:** ____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. Let T be a B-tree with minimum degree $t = 8$. If T stores one billion (that is, $10^9 \approx 2^{30}$) keys, what are the possible heights of T? **(8)**

(**Hint:** Derive lower and upper bounds on the height h of T in terms of the number n of keys stored in T. Recall that in a non-empty B-tree with minimum degree t, the root contains at least two and at most $2t$ children, and a non-leaf non-root node contains at least t and at most $2t$ children. The number of keys in a node is one less than the number of its children—for a leaf, this is in the range $[t-1, 2t-1]$. Finally, all leaves occur at the same level.)

Solution For a given height h of T, the number n of keys in T is smallest when the root has exactly two children (one key), every non-root non-leaf node has exactly t children ($t-1$ keys), and each leaf node stores exactly $t-1$ keys. We, therefore, have

$$\begin{aligned} n &\geqslant 1 + 2(t-1) + 2t(t-1) + 2t^2(t-1) + \cdots + 2t^{h-1}(t-1) \\ &= 1 + 2(t-1)(1 + t + t^2 + \cdots + t^{h-1}) \\ &= 1 + 2(t^h - 1) = 2t^h - 1, \end{aligned}$$

that is,

$$h \leqslant \log_t \left(\frac{n+1}{2}\right).$$

On the other hand, n is largest when each non-leaf node in T has exactly $2t$ children ($2t-1$ keys), and each leaf node too stores exactly $2t-1$ keys. This implies that

$$\begin{aligned} n &\leqslant (2t-1)(1 + (2t) + (2t)^2 + \cdots + (2t)^h) \\ &= (2t)^{h+1} - 1, \end{aligned}$$

that is,

$$h \geqslant -1 + \log_{2t}(n+1).$$

Putting $t = 2^3$ and $n \approx 2^{30}$ gives

$$h \leqslant \lg\left(\frac{n+1}{2}\right) / \lg t \approx \lg 2^{29} / \lg 2^3 = 29/3 \approx 9.67.$$

Moreover,

$$h \geqslant -1 + \log_{2t}(n+1) \approx -1 + [\lg n / \lg(2t)] \approx -1 + [\lg 2^{30} / \lg 2^4] = -1 + (30/4) = 6.5.$$

Therefore, we approximately have

$$6.5 \leqslant h \leqslant 9.67.$$

Since h is an integer, the possible values of h are $7, 8, 9$.

2. You are given two alphabetic (lower case) strings S and T each of the same length n. Propose an $\mathrm{O}(n)$-time algorithm to decide whether S can be obtained by permuting the symbols of T. **(7)**

(**Examples:** The string *algorithm* is a permutation of *logarithm*, *retinae* is a permutation of *trainee* but not of *entrain* or *trainer*.)

Solution The total number of symbols in the alphabet is $t = 26$ which is a constant. Therefore, we can sort both S and T in $\mathrm{O}(n + t) = \mathrm{O}(n)$ time using counting sort. We then check whether the sorted S matches character by character with the sorted T. Indeed, it is not necessary to sort S and T explicitly. Counting the numbers of occurrences of the 26 possible characters in both S and T, and matching the two sets of counts suffice.

3. The following function bubble sorts an array of n pairs with respect to the first field x. Prove or disprove: This is a stable sorting algorithm. **(5)**

```
typedef struct { int x; int y; } pair;

void bubblesort ( pair A[] , int n )
{
   int i, j;  pair t;
   for (j=n-2; j>=0; --j) {
      for (i=0; i<=j; ++i) {
         if (A[i].x > A[i+1].x) { t = A[i]; A[i] = A[i+1]; A[i+1] = t; }
      }
   }
}
```

Solution *True.* The function never swaps two pairs with equal x values, so pairs with the same x value go to the output in the same order as they appear in the input.