Homework 0

Prateep Mukherjee

December 14, 2013

1. (a)

   - $A(n) = \Theta\left(4^n\right)$
   - $B(n) = \Theta\left(n^2\right)$
   - $C(n) = \Theta\left(n^2\right)$
   - $D(n) = \Theta\left(n^{log_3 2}\right)$
   - $E(n) = \Theta\left(3^{\frac{n}{2}}\right)$

   (b) $7^n >> \sqrt{7^n} >> 7^{\sqrt{n}} >> 7^{lg(n)} >> 7^{lg(\sqrt{n})} \equiv \sqrt{7^{lg(n)}} >> 7^{\sqrt{lg(n)}} >> lg7^n >> lg\sqrt{7^n} >> n >> lg7^{\sqrt{n}} >> \sqrt{lg\,7^n} >> \sqrt{n} >> lgn >> lg\sqrt{n} > \sqrt{lg\,n}$

2. (a) List of nodes generated by a preorder traversal are:

   - Preorder: S Q V I R T Z A P H E B D X F L O G M N Y C K

   Proof: S is at the beginning of the inorder traversal and at the last of the post-order traversal. This implies that the left-subtree of S is NULL. Next, V is the immediate neighbor to S in inorder traversal. Hence, this should be the next left child of S. However, it is not directly attached to S. Its father, the node Q, comes immediately after S in post-order traversal. Thus, Q should be the immediate right child of S, and the father of V.

(b)

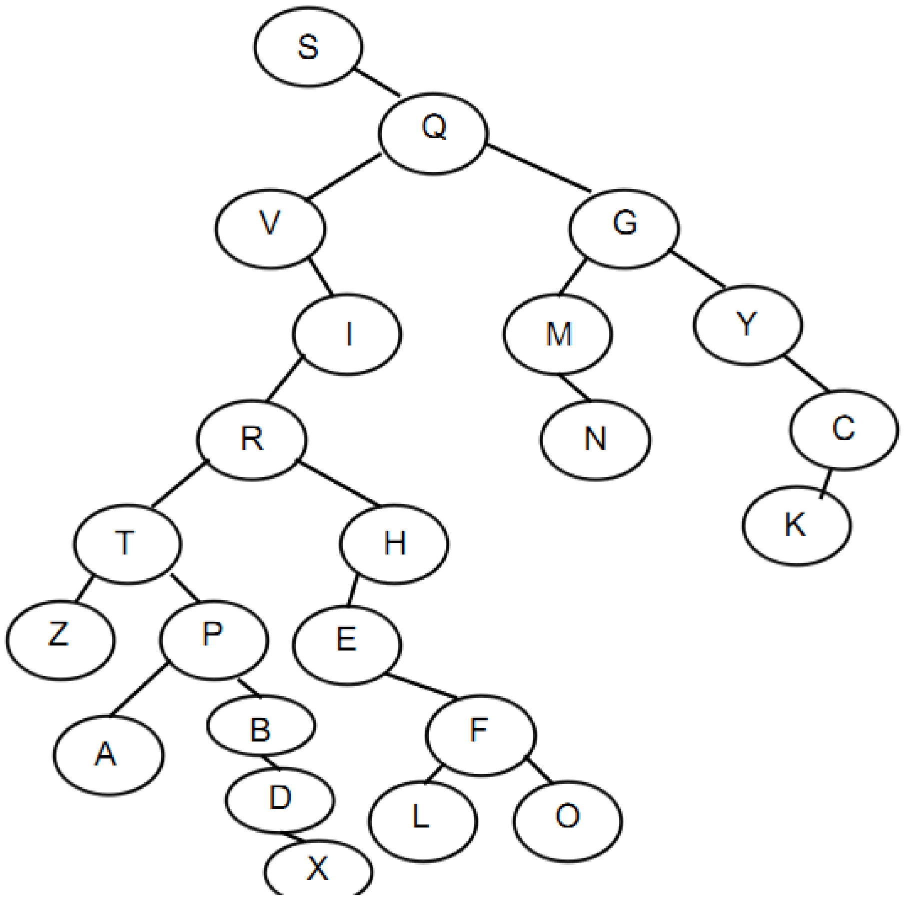

Figure 1: Prof. Giungla's tree

3. A *segment-tree* (`http://en.wikipedia.org/wiki/Segment_tree`) data-structure can be used to support the given queries. A segment-tree is a heap-like structure, where in each node we store the maximum value within a given range in an array. The segment-tree data structure supports two operations: *BuildSegmentTree(node,beg,end,T,A,n)* and *QuerySegmentTree(node,beg,end,T,A,i,j)*. Here, T is the array storing the segment-tree data-structure. Size of T is O(n), since height of the tree is $\log n$ and the maximum number of nodes stored is $2^{lg\, n} \equiv n$.

Both the functions above are performed in $\mathrm{O}(\log n)$, where n is the number of nodes in the tree. As the preprocessing step, we build two segment-trees, one on x-coordinate and the other on y-coordinate. The calling function for this operation is $BuildSegmentTree(1, 1, n, T, \mathcal{A}, n)$. Lets call these two arrays T_x and T_y respectively. Having defined the above functions, we can define the functions HighestToRight and *RightMostAbove* similarly, as follows.

Data: point arrays $\mathcal{A}$ (**x**,**y**), l
Result: HighestToRight(l)
sort $\mathcal{A}$ based on x-coordinate ;
binary-search in $\mathcal{A}$ for i s.t. $(x_i) \geq l$;
if $\neg\exists i$ **then**
  return NONE ;
end
ind = QuerySegmentTree(1, 1, n, T_x, A, i, n);
if $ind == -1$ **then**
  return NONE;
else
  return $\mathcal{A}_{ind}$;
end

Algorithm 1: HighestToRight(l)

Data: point arrays $\mathcal{A}$ (**x**,**y**), l
Result: RightmostAbove(l)
sort $\mathcal{A}$ based on y-coordinate ;
binary-search in $\mathcal{A}$ for i s.t. $(y_i) \geq l$;
if $\neg\exists i$ **then**
  return NONE ;
end
ind = QuerySegmentTree(1, 1, n, T_y, A, i, n);
if $ind == -1$ **then**
  return NONE;
else
  return $\mathcal{A}_{ind}$;
end

Algorithm 2: RightmostAbove(l)

Size of the segment-tree T is O(n). Building the segment-tree is $O(\log(n))$. Each query is one search in the segment-tree, for both algorithms. So total query time is $O(\log(n))$.

4.

Lemma. *Any arithmetic expression tree can be decomposed into equivalent arithmetic expression tree in normal form.*

Definitions: Let, E denote an expression or non-terminal node and $\mathcal{A}$ denote a variable or terminal node. Therefore, we define the following semantic rules.

- $E = E \quad | \quad E + E \quad | \quad E \times E \quad | \quad \mathcal{A}$
- $\mathcal{A} = a, \cdots, z \quad | \quad \mathcal{A}$

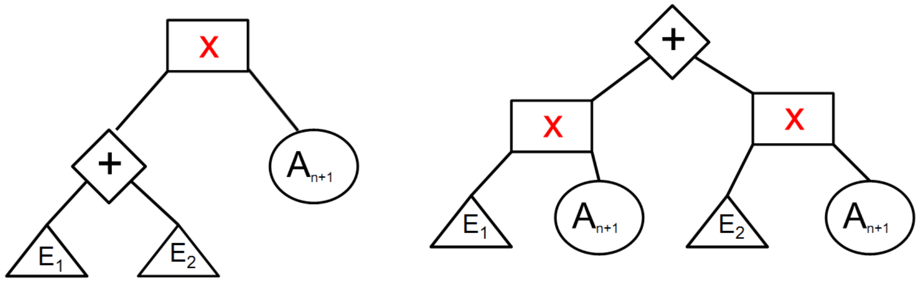

Figure 2: The expression tree(Left) which is transformed to generate the expression tree in normal form(Right).

Henceforth, we define the leaf nodes of the tree by $\mathcal{A}$ and the non-leaf nodes by E. Let us also denote

Proof. We prove the following lemma using the principle of induction.

[**Basis step**] We prove that the lemma is true for $n = 3$. The diagram given in the figure shows the two possible positions of the operators, $+$ and $\times$. From each of the two positions we can generate an expression tree in normal form.

[**Inductive step**] Let us assume the above lemma holds true for n nodes, $\mathcal{A}_1, \cdots \mathcal{A}_n$. Let us call this tree T_n, which is in normal form. Now, we have to add another term, $\mathcal{A}_{n+1}$ to the expression to see if we can generate an expression tree in normal form. The new term $\mathcal{A}_{n+1}$ can be appended to T_n in two ways. (i) $T_n + \mathcal{A}_{n+1}$, or (ii) $T_n \times \mathcal{A}_{n+1}$. If we append the term $\mathcal{A}_{n+1}$ at the front, we can prove our lemma using a similar argument.
It is evident that an expression tree will not be in its normal form if parent of a $+$-node is **not** a $+$-node. That is, the root of T_n is $+$-node and we multiply the term $\mathcal{A}_{n+1}$ to that, which is shown in the Fig. 2.
As seen in Fig. 2, we can transform a given expression tree, not in normal form, to an equivalent expression tree, in normal form, using the above transformation. Also here, the expressions E_1 and E_2 are in normal form. Hence, the new expression tree T_{n+1} is also in normal form. Hence, proved.

$\square$

5. (a) The cards are thrown away in a pair. So, each time either 0, or 2, or 4, or 8 etc cards are thrown away. Therefore,

$$E(\#\, of\ cards\, thrown) = 0 \times P(0) + 2 \times P(2) + \cdots + 102 \times P(102)$$

The expected number of cards that are hurled is $\frac{1751}{52}$.

(b)

i $\frac{1}{169}$

ii $\frac{1}{52}$

iii $\frac{1}{4}$

iv 0.0145

(c)

i $\frac{1}{13}$

ii $\frac{17}{52}$

iii 0

iv $\frac{3}{11}$