Homework 4

Fundamental Algorithms, Spring 2003, Professor Yap

Due: Mon April 14, in class
Solutions prepared by: T.A. Igor Chikanian and Chee Yap.

INSTRUCTIONS:

- Please read the questions carefully. When in doubt, please ask.

---

1. (20 Points)
In Lecture V.2, we define a code tree $T = T_C$ corresponding to any prefix code C. Let f be the frequency function that assigns to each leaf u of T a frequency $f(u) \geq 0$. The COST of this tree with frequency f is defined to be $COST(T, f) = \sum_u f(u)d(u)$ where u ranges over the leaves of T and $d(u)$ is the depth of u in T. We noted (without proof) an alternative way to compute the $COST(T, f)$: first extend f so that it assigns a frequency to every node v in T: if v is a leaf, then $f(v)$ is already defined. Otherwise, if the two children v_L, v_R of v have been assigned frequencies, define $f(v) = f(v_L) + f(v_R)$. Alternatively, $f(v)$ is just the sum of all the frequencies of the leaves below v. When v is the root of T, we also write $f(T)$ instead of $f(v)$. Let $C(T, f) = \sum_v f(v)$ where v ranges over all the *internal* nodes of T. You are to prove that $COST(T, f) = C(T, f)$.
SOLUTION:

Proof by induction on number of nodes in the tree: For trees with one node (no internal nodes):

$$COST(T, f) = C(T, f) = 0$$

Suppose this is true for all trees with less than n nodes. Let T have n nodes. Also, let u_0 be the root of T and let T_L and T_R be the left and right subtrees of T. So, by induction hypothesis,

$$COST(T_i, f) = C(T_i, f)$$

where $i = L, R$. Note that for any leaf u_i of T_i, let $d_i(u_i) = d(u_i) - 1$ denote the depth of u_i in T_i. Then

$$\begin{aligned}
COST(T, f) &= \textstyle\sum_u f(u)d(u) \\
&= \textstyle\sum_{u_L} f(u_L)d(u_L) + \sum_{u_R} f(u_R)d(u_R) && (u_i \text{ ranges over leaves of } T_i,\ i = L, R) \\
&= \big(f(T_L) + \textstyle\sum_{u_L} f(u_L)d_L(u_L)\big) \\
&\qquad + \big(f(T_R) + \textstyle\sum_{u_R} f(u_R)d_R(u_R)\big) && (\text{using } d_i(u_i) + 1 = d(u_i)) \\
&= (f(T_L) + f(T_R)) + C(T_L, f) + C(T_R, f) && (\text{by induction hypothesis}) \\
&= f(T) + C(T_L, f) + C(T_R, f) && (f(T) = f(T_L) + f(T_R)) \\
&= C(T, f) && (\text{by definition of } C(T, f)\ )
\end{aligned}$$

2. (20 Points)
Consider the insertion of the following sequence of keys into an initially empty tree:

$$1, -1, 2, -2, 3, -3, \ldots, n, -n.$$

Let T_n be the final splay tree. In Lecture VI.2, we gave an insertion algorithm in which splay the tree before inserting the key. Call this Method 1. We noted in class an alternative, Method 2, where you first insert *as in an ordinary binary tree*, and then splay the newly inserted node.
(i) Show T_n for $n = 1, 2, 3, 4$ under Method 1.
NOTE: Please be careful – if T_i is wrong, then T_{i+1} is assumed wrong!
(ii) Show T_n for $n = 1, 2, 3, 4$ under Method 2.
(iii) Formulate a conjecture about the shape of T_n under Method 2. Prove it.

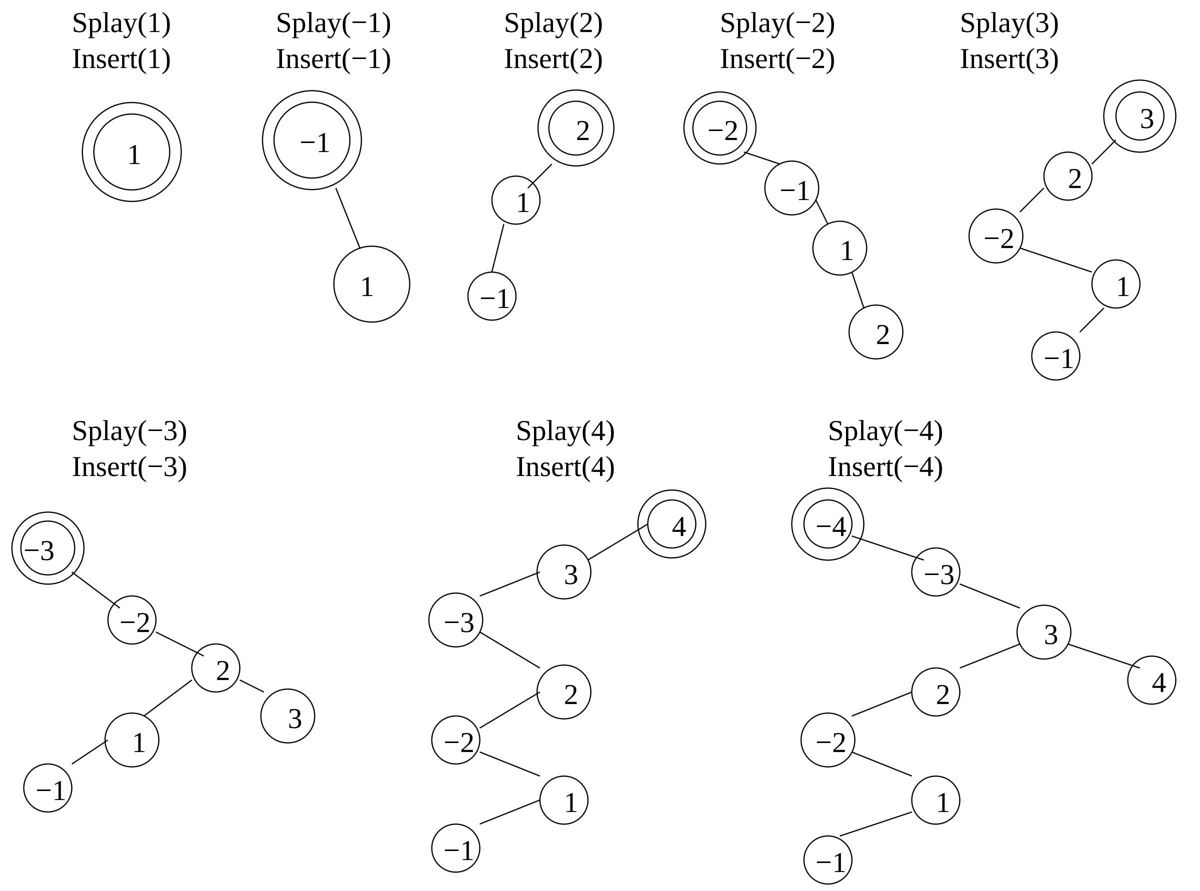


Figure 1: Q2(i) Method I

SOLUTION:

(i) See Figure 1.

(ii) See Figure 2.

(iii) The conjecture should be: T_n is a tree with a unique leaf u_0, and the path from this leaf to the root is a zig-zag path (alternating a left turn and a right turn). The root contains $-n$ and its right child is n. The left child of n is T_{n-1}. For the proof, see Figure 3.

3. (15 Points)
In many applications of binary search trees, we need to maintain more information than just the search key. Thus, in the convex hull application (Lecture VI.4), we assumed that each node u has a reference to its successor u.`succ` and predecessor u.`pred`. Assume that u.`pred` $=$ *null* if u contains the minimum node (i.e., the one with smallest key in the tree); similarly u.`succ` $=$ *null* if u is the maximum node. Consider maintaining such pointers in splay trees. What additional steps must be carried out to maintain these pointers when you perform the following splay tree operations?
(i) Rotation. (Careful – this could be considered a trick question.)
(ii) Insertion.
(ii) Deletion.

SOLUTION:
(i) Rotation preserves all keys and the successor/predecessor relations are not disturbed, so no additional action is needed.

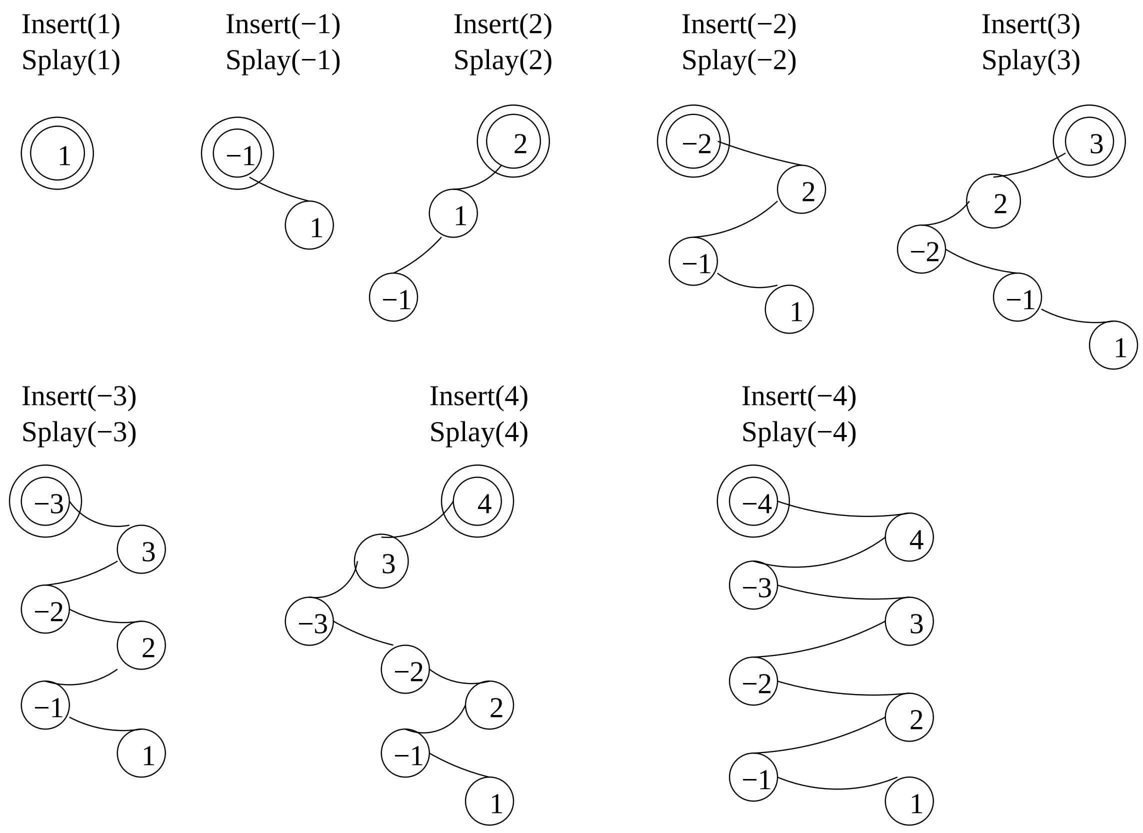


Figure 2: Q2(ii) Method II

(ii) Assume that node u is inserted as described in the Lecture Notes for inserting into splay tree. This means that we first splay the tree to get a new node u' and then u is made the parent of u'. Now there are two cases, depending on whether $u.Key > u'.Key$ (call this Case A) or $u.Key < u'.Key$ (call this Case B). We assume that you need to insert some code to above insertion algorithm remember which case occurred. We now perform some additional assignment of pointers:

Case A (See Figure 4)

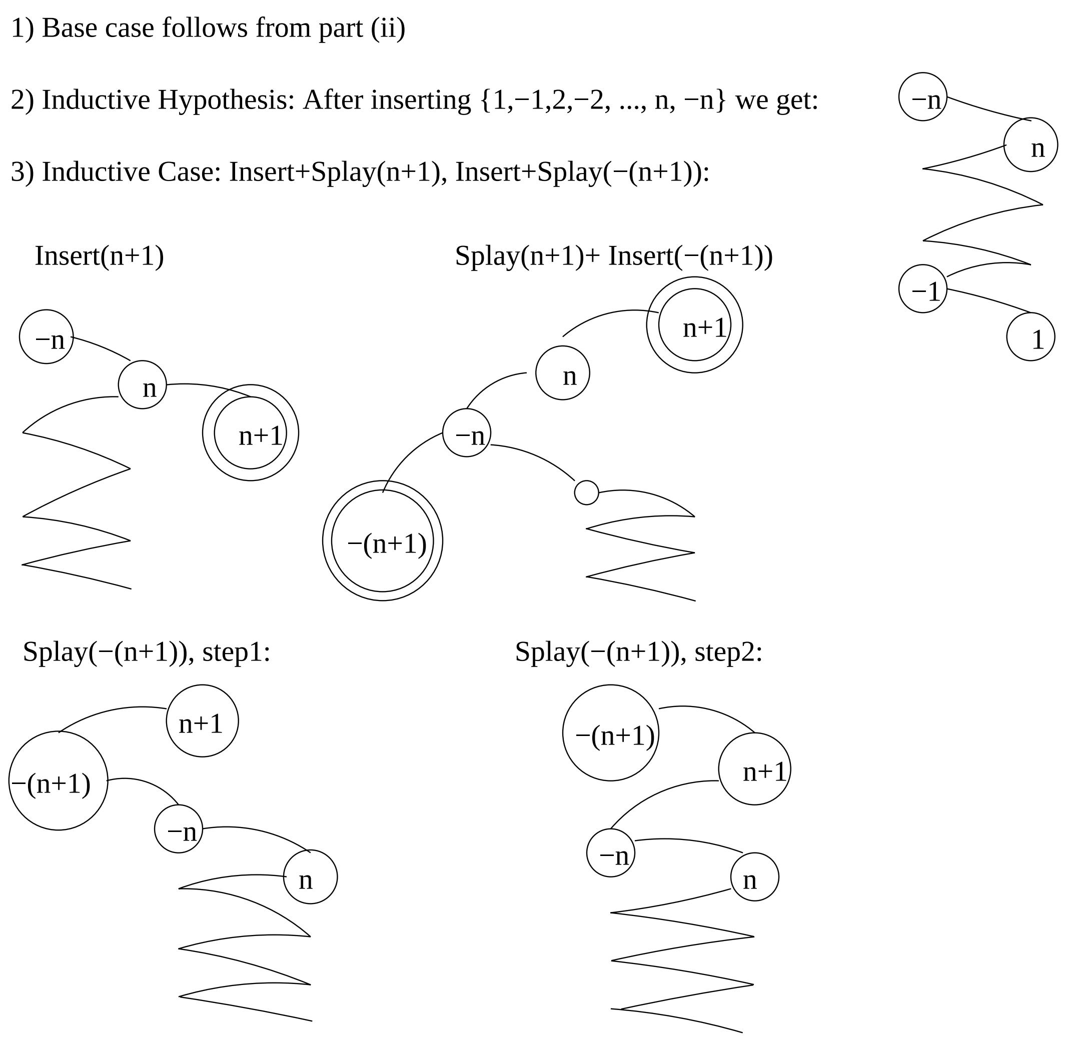
```
Suppose node $u$ was inserted as usual, $u'$ was root after splaying.
if ($u'$.succ $\neq$ $null$) then $u'$.succ.pred := $u$;
$u$.succ := $u'$.succ;
$u'$.succ := $u$;
$u$.pred := $u'$;
```

Case B (See Figure 4)

```
Suppose node $u$ was inserted as usual, $u'$ was root after splaying.
if ($u'$.pred $\neq$ $null$) then $u'$.pred.succ := $u$;
$u$.pred := $u'$.pred;
$u'$.pred := $u$;
$u$.succ := $u'$;
```

(iii) For deletion of node d, we only have to do the following:

```
if ($d$.pred $\neq$ $null$) then $d$.pred.succ := $d$.succ;
if ($d$.succ $\neq$ $null$) then $d$.succ.pred := $d$.pred;
delete($d$) as usual
```



1) Base case follows from part (ii)

2) Inductive Hypothesis: After inserting {1,−1,2,−2, ..., n, −n} we get:

3) Inductive Case: Insert+Splay(n+1), Insert+Splay(−(n+1)):

Figure 3: Q2(iii) Proof by Induction

4. (20 Points)
In the problem of maintaining an upper hull H, we represent H as a splay tree T. Lecture VI.4 gives an outline of the procedure called `FindLeftTangentPoint`(p, T) where we already know that p (the query point) lies outside the upper hull. You are to flesh out the details of this procedure, reducing all numerical detail computation to just the **left turn predicate**. This predicate is defined[1] for any three points $p, q, r \in \mathbb{R}^2$:

$$LeftTurn(p, q, r) = \begin{cases} +1 & \text{if the path } (p, q, r) \text{ represents a left turn} \\ -1 & \text{if the path } p, q, r) \text{ represents a right turn} \\ 0 & \text{if } p, q, r \text{ are collinear} \end{cases}$$

Note that `FindLeftTangent` could return the special point $v_S = (0, -\infty)$ (South Pole).

ASSUMPTIONS: For simplicity, you may assume that the input points are **non-degenerate** in the sense that $LeftTurn(p, q, r)$ on any *input* points p, q, r will never return a 0, and that the x-coordinate

[1] Note that in computational geometry, "predicates" are often 3-valued as in this instance. Contrast this to the 2-valued (true/false) logic that is more common elsewhere. The LeftTurn predicate is only one example of the class of "orientation predicates". See class notes and Exercise 4.1 for how to implement the LeftTurn predicate as the sign of a determinant. But this information is irrelevant for the present problem.

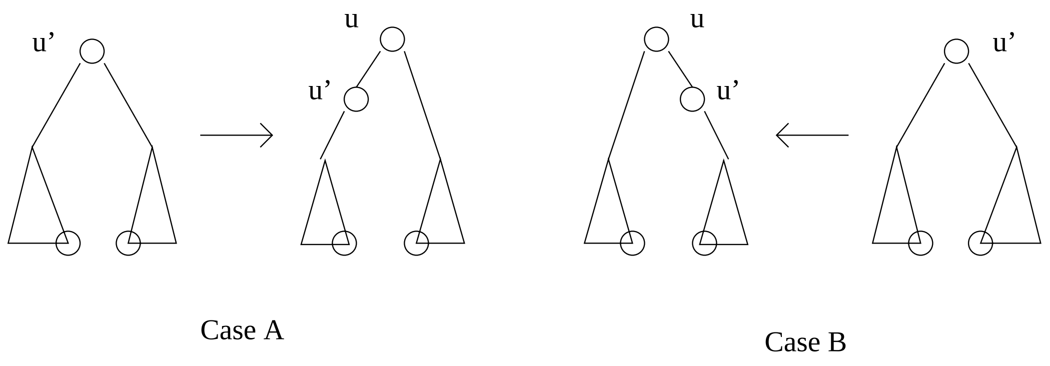


Figure 4: Q3: Insert

of input points and query point are all distinct. However, your solution must take care of what happens when a node u has no successor or no predecessor.

SOLUTION:

```
FindLeftTangentPoint(p,T)
Let $u := root(T)$;
Repeat Forever
    $v := u$.succ;
    $w := u$.pred;
    if ($p.x < u.x$)
        if ($w = null$) return $SouthPole$;
        $u := u.LeftChild$; break;
    if ($w \neq null$) and ($LeftTurn(w, u, p)$)
        $u := u.LeftChild$; break;
    if ($v \neq null$) and (Not $LeftTurn(u, v, p)$)
        $u := u.RightChild$; break;
    return $u$; //check that this is correct whether or not $v$ or $w$ is $null$.
```

5. (0 Points)

THIS PROBLEM IS NOT REQUIRED, so do not submit for grading. It is for your practice, and we will publish solution.

Let T be a splay tree on n nodes, and let T' be the result of inserting a new key into T using the standard insertion algorithm. So, the new key appears as a leaf u in T' but in all other respects T and T' are identical. The following will step you through a proof of

$$\Phi(T') - \Phi(T) = O(\lg n). \tag{1}$$

Consider the path $\pi = (u_0, u_1, \ldots, u_k)$ from the root u_0 of T' to $u = u_k$.

(i) Let v_i be the sibling of u_i for $i = 1, 2, \ldots, k$, and let s_i be the size of the subtree rooted at v_i. If u_i has no sibling, then v_i is undefined and s_i is 0. Let Δ_i be the increase in potential of u_i in going from T to T'. Express Δ_i in terms of the s_j's ($j = i, i+1, \ldots, k$).

(ii) Express $\Phi(T') - \Phi(T)$ in terms of the Δ_i's.

(iii) Try to bound the expression from (ii) as a telescoping sum, and deduce (1),

SOLUTION:

(i) Δ_i is the change in potential of u_i after we inserted u_k, so:

$\Delta_{k-1} = \lfloor \log(s_k + 2) \rfloor - \lfloor \log(s_k + 1) \rfloor$
$\Delta_{k-2} = \lfloor \log(s_k + s_{k-1} + 3) \rfloor - \lfloor \log(s_k + s_{k-1} + 2) \rfloor$
$\Delta_{k-3} = \lfloor \log(s_k + s_{k-1} + s_{k-2} + 4) \rfloor - \lfloor \log(s_k + s_{k-1} + s_{k-2} + 3) \rfloor$

... ...

$\Delta_i = \left\lfloor \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i+1)) \right\rfloor - \left\lfloor \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i)) \right\rfloor$
$\Delta_{i-1} = \left\lfloor \log(\sum_{j=i}^{k} s_j \ + \ (k-i+2)) \right\rfloor - \left\lfloor \log(\sum_{j=i}^{k} s_j \ + \ (k-i+1)) \right\rfloor$

... ...

$\Delta_1 = \left\lfloor \log(\sum_{j=2}^{k} s_j \ + \ k) \right\rfloor - \left\lfloor \log(\sum_{j=2}^{k} s_j \ + \ (k-1)) \right\rfloor$
$\Delta_0 = \left\lfloor \log(\sum_{j=1}^{k} s_j \ + \ (k+1)) \right\rfloor - \left\lfloor \log(\sum_{j=1}^{k} s_j \ + \ k) \right\rfloor$

Notice that we have:

$$\begin{aligned}
\sum_{j=i}^{k} s_j \ + \ (k-i+1) \quad &\geq \quad \sum_{j=i+1}^{k} s_j \ + \ (k-i+1) \\
\log(\sum_{j=i}^{k} s_j \ + \ (k-i+1)) \quad &\geq \quad \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i+1)) \\
\left\lfloor \log(\sum_{j=i}^{k} s_j \ + \ (k-i+1)) \right\rfloor \quad &\geq \quad \left\lfloor \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i+1)) \right\rfloor
\end{aligned}$$

Therefore, $\Delta_{i-1} \leq \Delta'_{i-1} = \left\lfloor \log(\sum_{j=i}^{k} s_j \ + \ (k-i+2)) \right\rfloor - \left\lfloor \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i+1)) \right\rfloor$
Similarly, $\Delta_i \leq \Delta'_i = \left\lfloor \log(\sum_{j=i+1}^{k} s_j \ + \ (k-i+1)) \right\rfloor - \left\lfloor \log(\sum_{j=i+2}^{k} s_j \ + \ (k-i)) \right\rfloor$

(ii),(iii) Let us compute $\Phi(T') - \Phi(T) = \sum_{i=0}^{k-1} \Delta_i$:

$\sum_{i=0}^{k-1} \Delta_i \leq \sum_{i=0}^{k-1} \Delta'_i$

By telescoping the last sum we get:
$\Phi(T') - \Phi(T) \leq \left\lfloor \log(\sum_{j=1}^{k} s_j \ + \ (k+1)) \right\rfloor - \lfloor \log(s_k + 1) \rfloor = \mathcal{O}(\log n)$, since the summation of subtree sizes s_j plus the length $(k+1)$ of the path π equals the number of nodes in the tree.