COSC229 Assignment 2 -- Written Assignment on Data Structures and Algorithms

Question 1. If a new item can be inserted into the root of a binary search tree (BST), subsequent access to recent items can be done efficiently, as they will be near the root. This is based on the assumption that recent items are more likely to be accessed next. In a splay tree, a new item comes to the root after the splay operation. There is another scheme for insertion at the root. The idea is to find an insertion point as an ordinary binary search tree. Then by traversing back to the root on this path, we relocate smaller keys and greater keys to the left and to the right of this path respectively. See the following picture for an example. We deal with only keys for insertion. We call the new data structure the root BST.

Suppose we insert 6 into the first tree in the following picture. Dotted parts indicate the possible insertion point. We traverse the path from the dotted square to the root, and arrange the part of the tree smaller than 6 as the left sub-tree of the root, and the greater parts as the right sub-tree of the root. The roots of smaller part and greater part are given by pointers s and g.

Figure 1 Root BST

Suppose we are trying to insert key x. In the following program, pointers p and q are for establishing the path to the hypothetical insertion point. Pointer q is a follower of p. Pointers s and s1 are for keeping track of smaller parts, and g and g1 are for greater parts. Pointer s1 is the follower of s and g1 is that of g. At the end the key x is inserted to the new root, and pointer s becomes the left child of the root and g becomes the right child.
Let us walk on the path from the dotted square towards the root, using pointers \( u, v \) and \( w \), where \( u \) and \( v \) are used to identify direction, and \( w \) is used to detect a turning point. If \( v \) is the right (left) child of \( u \), key at \( u \) is smaller (greater) than \( x \). At the turning points we need to link smaller parts and greater parts using the follower pointers \( s_1 \) and \( g_1 \). The pointers \( s \) and \( g \) keep track of roots of smaller parts and greater parts. When we traverse in bottom-up manner on path, we have the right turning point (case 2) and the left turning point (case 4). Refer to the following program for case numbers. After the right turning point, \( s_1 \) is kept unchanged until the next left turning point is reached, and connected as the right child of \( u \). Similarly \( g_1 \) is kept unchanged after a left turning point, and connected as the left child of \( u \) at the next right turning point.

```c
insert(int x){
    int i;
    struct item*p, *q, *s, *g, *s1, *g1;
    s=nil; g=nil; p=proot; // proot is the pointer to the root
    while(p!=nil){
        q=p; m++; path[m]=q; // array path keeps track of the path to insertion point
        if (x<=(*p).key) p=(*p).left; else p=(*p).right;
    }
    w=malloc(sizeof(struct item)); path[m+1]=w; // w pointer to a dummy item
    if(x<=(*q).key) { (*path[m]).left=path[m+1]; g=q; }
    else { (*path[m]).right=path[m+1]; s=q; }
}
```

Figure 2. Path to the root
else         { (*path[m]).right=path[m+1]; s=q; }
i=m+1; gl=nil; sl=nil;
while(i>=3) { // back track on the path
    u=path[i-2]; v=path[i-1]; w=path[i];
    if((*u).left==v){g1=g; g=u;}  // save g to g1, g goes to u, case 1
    if(((*u).left==v)&&((*v).right==w)) (*u).left=g1;  // g1 connected to u, case 2
    if((*u).right==v){s1=s; s=u;}  // save s to s1, s goes to u, case 3
    if(((*u).right==v)&&((*v).left==w)) (*u).right=s1;  // s1 connected to u, case 4
    i=i-1;
}
if(x<=(*path[m]).key)path[m]->left=nil; else path[m]->right=nil;
proot=malloc(sizeof(struct item));
(*proot).key=x; (*proot).left=s; (*proot).right=g;
}

(A) Insert at least 10 random numbers of your choice into the root BST following the above
    algorithm. Draw the picture of the tree after each insertion.
(B) Insert the same set of numbers into a splay tree. Which is more balanced?

Question 2. The following is a mixture of radix sort and quick sort. It inspects the binary
    expansion of each key from most significant bit. It partitions the keys with 0 as the most
    significant bit to the left and those with 1 to the right. Thus the keys (53, 14, 27, 2, 31, 85, 30,
    11, 67, 50) are partitioned by the function partition in the following program into ((50, 14,
    27, 2, 31, 11, 30, 53), (67, 85)). Counting from the 0-th bit, the 6-th bits of the keys in the left
    list are 0 and those in the right list are 1. After partition, two recursive calls inspect the
    (k-1)-th bits.

Following the above example, partition the list of 10 random numbers of two digits of your
    choice with the 6-th bit. Note that the order in which keys appear in the left list and right
    list in your answer is important.

int bit(int x, int k){ // This is to return the k-th bit of key x
    int i;
    for(i=0; i<k-1; i++) x=x/2;
    if(x==((x/2)+2)) return 0; else return 1;
}
Question 3
The following is a graph with 6 vertices and 13 edges.

(1) Trace Dijkstra's algorithm for the single source shortest path problem with this graph.
(2) Trace Floyd's algorithm for the all pairs shortest path problem.
Question 4. Trace the KMP algorithm for pattern matching with
Pattern = abcdabcd
Text = abcdabccabcdababcdabcdb

Question 5. Trace "intersect" for segments s1 and s2, where
s1.p1=(1, 1), s1.p2=(3, 3), s2.p1=(3, 1), s2.p2=(2, 2), and
s1.p1=(1, 1), s1.p2=(3, 3), s2.p1=(2, 2), s2.p2=(2, 2)
For trace, show which cases to go through in the program and the truth values of turn functions.

Question 6. Trace the function "inside" in page 55 with p[1], ..., p[5] given by (0, 0), (5, 1), (4, 3), (6, 2), (3, 7) and v = (3, 2).
For a style, try
http://www.cosc.canterbury.ac.nz/tad.takaoka/cosc229/assi nsi de.c

Question 7. Trace Graham's algorithm for p1, ..., p9 given by
(7, 1), (9, 2), (10, 3), (14, 6), (11, 8), (10, 10), (4, 8), (2, 3), (5, 1)
Before trace, draw those points on a graph sheet. For a style, see exam06.doc at http://www.cosc.canterbury.ac.nz/tad.takaoka/cosc229/exam06.doc

Question 8. Trace the Euclidean algorithm for computing the greatest common divisor of two mutually prime numbers a and b of your choice with at least 2 digits each, and a⁻¹ mod b and b⁻¹ mod a. Note that the GCD is 1.
Due date: 15 October 2009, 5pm, drop due one week later. Worth 20%
Appendix: A large root BST

```
23
/  \
5    48
/  \
2    16
```

```
40
/  \
24    46
/    /  \
28    35    32
/    /  \
30    New
```