Indian Institute of Technology Kharagpur

CS29003: Algorithms Laboratory, Spring 2023

Assignment 4: Balanced Binary Search Trees

2PM – 5PM 31st January, 2023

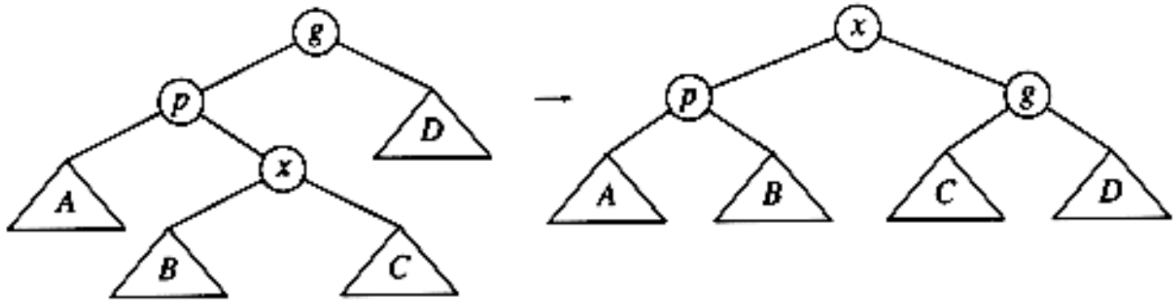
General Instructions (to be followed strictly)

Submit a single C/C++ source file.
Do not use global variables unless you are explicitly instructed so.
Do not use Standard Template Library (STL) of C++.
Use proper indentation in your code and include comments.
Name your file as `<roll_no>_a4.<extn>`
Write your name, roll number, and assignment number at the beginning of your program.

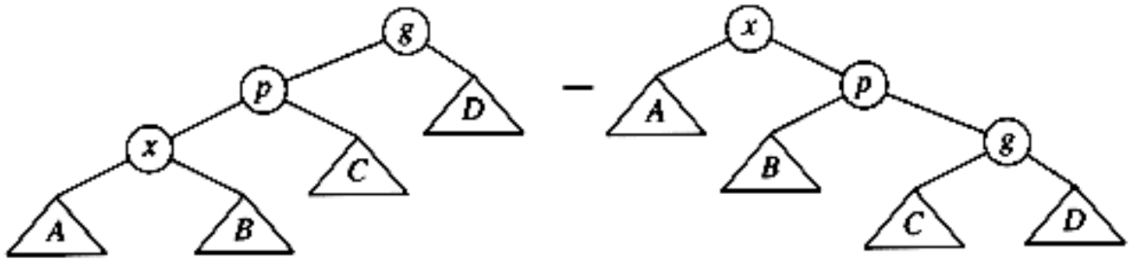
Often data access exhibits locality of reference — data accessed now is likely to get accessed again in near future. In this assignment, you implement a special kind of binary search tree which works well when data access pattern exhibits such locality of reference. The idea is to rearrange the binary search tree every time a key is searched/inserted in such a way that the searched/inserted key becomes the root. This is achieved by simply rotating all nodes visited from the root to the node which contains the search/insert key — let us call this path "search path." Let us call this tree "local tree." The kind of rotations that we perform at each node in the search path are decided by the following.

Let x be the key searched/inserted. If the parent of x is the root of the tree, we perform appropriate AVL tree single-rotation at the root so that x becomes the root node. Otherwise, let us assume p and g be respectively the parent and grand-parent of x in the current tree.

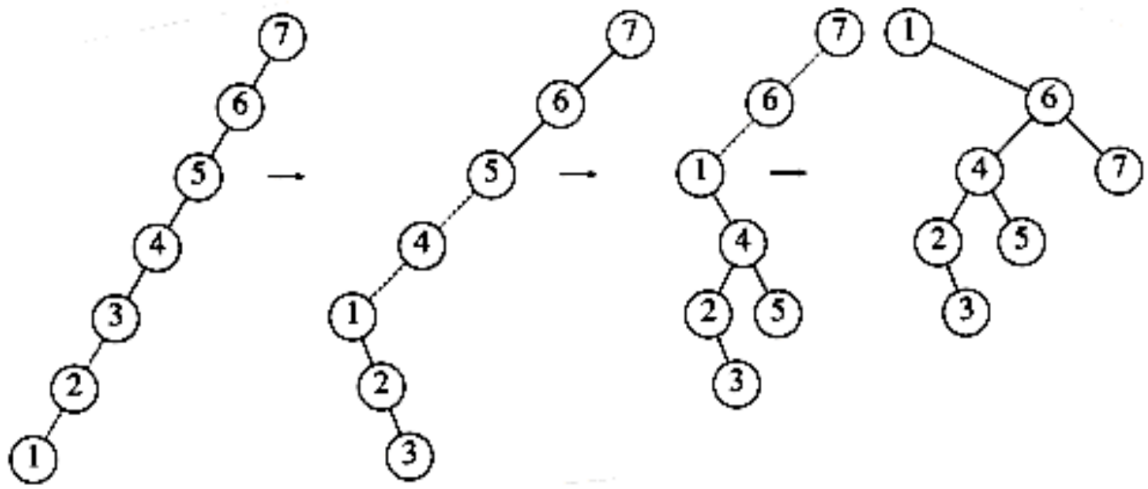
Case I: If x is a right child and p is a left child (or vice versa), then we perform appropriate AVL tree double-rotation at g (refer to the picture below).



Case II: If x and p are both right (or left) children, then we perform the operation as shown in the picture below.



We repeat the above operations as applicable until x becomes the root node. The example below shows how a binary search tree changes when we search for 1.



For this assignment, you store parent pointer also in the tree node. More concretely, use the following structure for storing all the information at each node.

```
typedef struct local_node *local_ptr;
struct local_node
{
int data;
local_ptr left;
local_ptr right;
local_ptr parent;
};
```

Write the following functions:

1. Insertion (including modification of the tree appropriately)
2. Print pre-order traversal of a binary search tree
3. Search for a given key in a binary search tree (including modification of the tree appropriately). If the key is not present, there is no need to alter the tree.

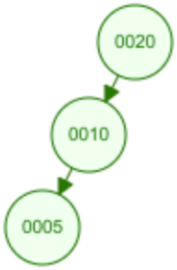
Sample Output

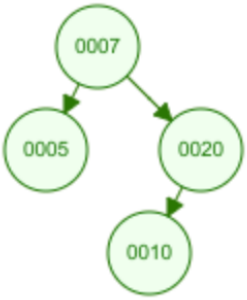
```
1. insert
2. search
3. exit

1
Write key to be inserted: 10
Pre-order traversal: 10
1
Write key to be inserted: 5
Pre-order traversal: 5, 10
1
Write key to be inserted: 20
Pre-order traversal: 20, 10, 5
1
Write key to be inserted: 7
Pre-order traversal: 7, 5, 20, 10
2
Write key to be searched: 10
Pre-order traversal: 10, 7, 5, 20
1
Write key to be inserted: 1
Pre-order traversal: 1, 10, 5, 7, 20
1
Write key to be inserted: 9
Pre-order traversal: 9, 1, 7, 5, 10, 20
1
Write key to be inserted: 30
Pre-order traversal: 30, 9, 1, 7, 5, 20, 10
3
```

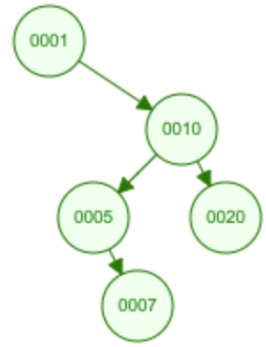
Pictures of the local-tree after each step of the above operations are shown below. These pictures are for your visualization. You do not need to output these pictures from your code.

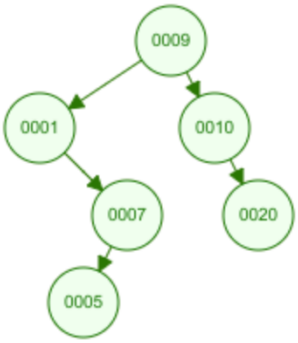

0005
0010

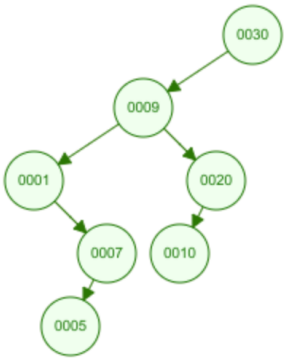

0020
0010
0005


0007
0005
0020
0010


0010
0007
0020
0005


0001
0010
0005
0020
0007


0009
0001
0010
0007
0020
0005


0030
0009
0001
0020
0007
0010
0005