Assignment B — Binary Search Tree and Treasure Hunter

TA: Chen (rainywindsnow@yahoo.com.tw)

Deadline: Nov. 19th (Thursday) 11:59pm

1 Implement a Binary Search Tree with linked lists (45%)

A binary search tree (BST) is a sorted tree structure, where each node has two children. The left node always needs to contain a value smaller than the parent and the right node always needs to contain a value greater than the parent. Exception is when the node contains no data.

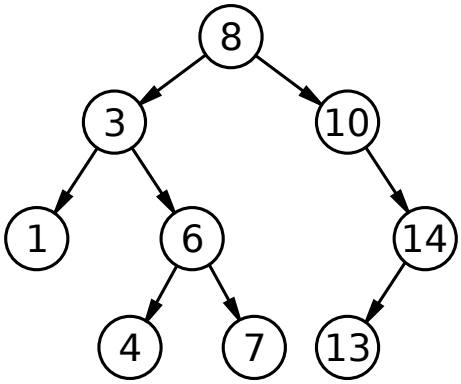

Figure 1: Example of a BST (From: `http://commons.wikimedia.org/wiki/File:Binary_search_tree.svg`)

In this homework, you need to implement a BST structure for integer numbers and provide the following functions:

insert(i) let the user insert a number **i**; error out when it is already existing.

delete(i) let the user delete a number **i**; error out when it is not existing. If the node with two child nodes is deleted, choose the smallest one on the right subtree to replace the deleted node.

search(i) search the tree; message out whether the number **i** is found or not.

printInfixOrder() print the whole tree in infix order (from left to right)

printLevelOrder() print the whole tree in level order (from up to down)

To help you getting started with this assignment, here is a piece of code that you can use. It is the code for a node in a tree. The node can contain two child nodes, `left` and `right`.

```
struct node
{
  int data;
  struct node *left;
  struct node *right;
};
```

2 Treasure Hunter (45%)

The function of treasure hunter is very similar to the part one. In addition to implementing a BST, you need to implement a treasure hunter function to look for a treasure on the tree. The scenario is described as follows:

There is a treasure hunter who goes into the maze represented by a binary search tree to find the treasure. The treasure hunter needs to find a key first, and then he can open the treasure box to get the treasure. The hunter goes into the maze with one entrance at the root node. In the maze, there are some bombs at the node with the number containing 8(such as 8, 28, 382, 818). When passing these nodes, bombs will be triggered. As a result, the maze would be changed as the nodes burn out due to bomb explosion, which can be treated as the node deletion from a binary search tree. If the node with the bombs v_i has two child nodes, the node with smaller number burns when the bombs are triggered, and the other one remains intact. Furthermore, if the burnt node(i.e., the deleted node) has two child nodes, choose the smallest node in the right subtree to replace the deleted node. If v_i is a leaf node, nothing burns when the explosion happens. Therefore, the maze would not be changed. Each bomb could only be triggered once. Therefore, while passing the same node twice, the bomb would not be triggered again.

In this part of homework, you are given a map file to construct the maze. The format of file is as follow.

```
8
3
1
13
10
6
4
12
9
7
18
15
2
19
```

Insert each number sequentially into the binary search tree. The illustration of the maze is shown in Figure 2. After constructing the maze, you need to enable user inputs to specify a key and a treasure.

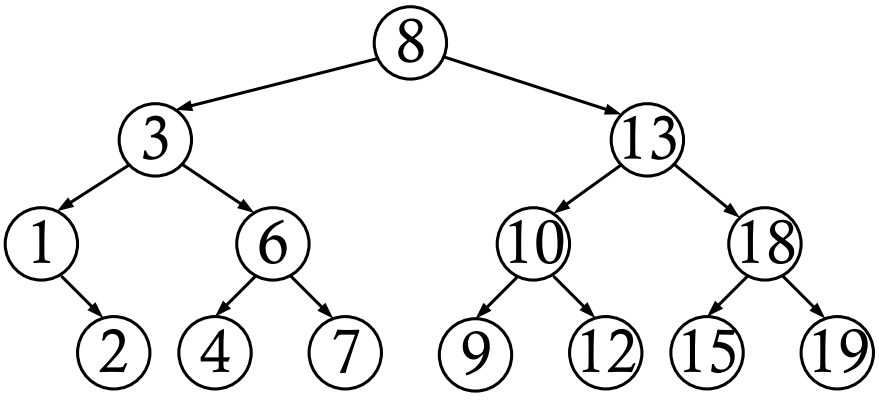

Figure 2: Example of a maze

For example, assume a user sets the key at node 7 and the treasure at node 19.

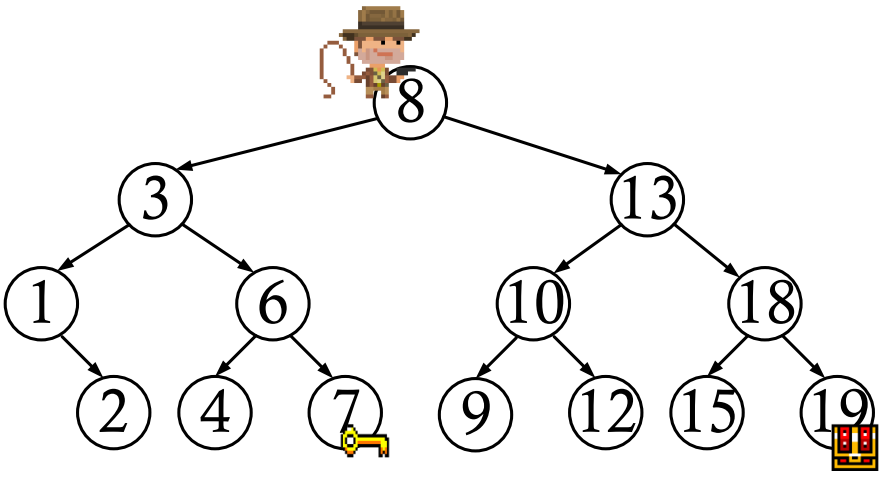

Figure 3: Example of a maze

When treasure hunter reaches node 8, the maze is changed as the rule described in the second paragraph by deleting node 3 and taking node 4 to replace the deleted node. The result is shown in Figure 4

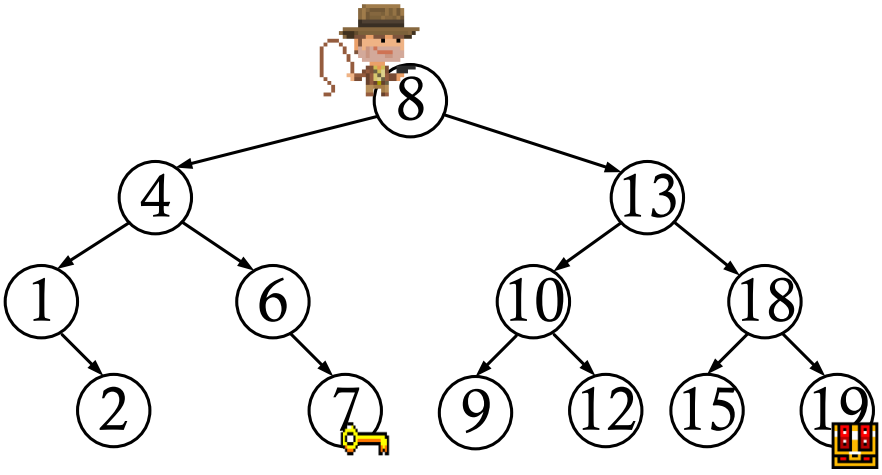

Figure 4: Example of a maze

Figure 5 shows the shortest path which the treasure hunter went through to find the treasure. The treasure hunter found the key in step 1. In step 2 to step 5, the treasure hunter found the treasure. The treasure hunter triggered the bomb in step 4, so node 15 was deleted. In the end, you need to print out the shortest path to find the treasure. (i.e. 8→ 4→ 6→ 7→ 6→ 4→ 8→ 13→ 18→ 19)

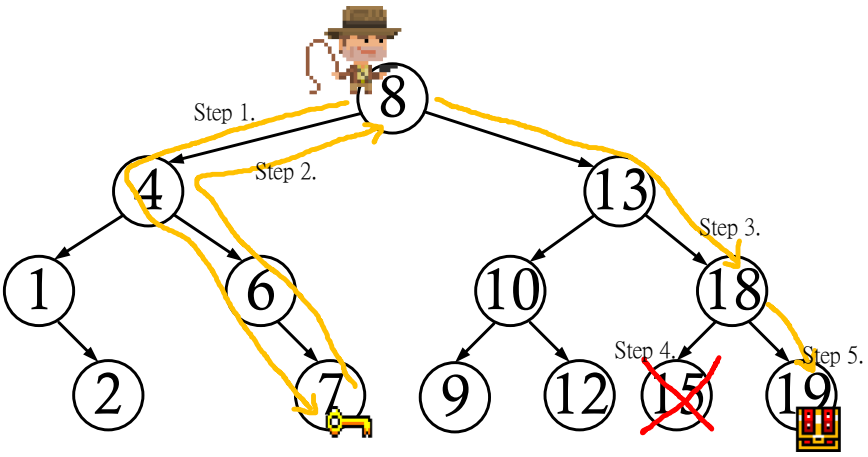

Figure 5: Example of a maze

Here is the summary of what you need to do for this assignment.

1. Use the map file to construct a BST maze.

2. Allow user inputs to specify a key and a treasure.

3. When passing the node with the number containing 8, change the maze based on the rules.

4. Print out the shortest path to the key and the treasure. If the treasure hunter cannot find the key print "Key is not found.". If the treasure hunter cannot find the treasure print "Treasure is not found.".

PS. It is possible that the treasure or the key cannot be found when the key and/or the treasure do not exist in the maze. Another possibility is that the node with key or treasure is deleted.

Download the example map files: (During the demo, we will use different map files with the same format)
exmap.txt `https://ecourse.ccu.edu.tw/35960/textbook/hw/HWB/exmap.txt`
exmap2.txt `https://ecourse.ccu.edu.tw/35960/textbook/hw/HWB/exmap2.txt`
exmap3.txt `https://ecourse.ccu.edu.tw/35960/textbook/hw/HWB/exmap3.txt`

Testing input and result:

1. Read exmap.txt, key at 3, treasure at 9.
Print: 7→ 4→ 2→ 3→ 2→ 4→ 7→ 12→ 8→ 9

2. Read exmap.txt, key at 10, treasure at 6.
Print: Key is not found.

3. Read exmap.txt, key at 9, treasure at 12.
Print: 7→ 12→ 8→ 9→ 8→ 12

4. Read exmap.txt, key at 1, treasure at 20.
Print: Treasure is not found.

5. Read exmap2.txt, key at 16, treasure at 67.
Print: 80→ 23→ 12→ 13→ 19→ 16→ 19→ 13→ 12→ 23→ 65→ 69→ 66→ 67

6. Read exmap2.txt, key at 20, treasure at 77.
Print: Key is not found.

7. Read exmap2.txt, key at 97, treasure at 41.
Print: 80→ 94→ 95→ 96→ 97→ 96→ 95→ 94→ 80→ 23→ 65→ 38→ 41

8. Read exmap2.txt, key at 10, treasure at 44.
Print: 80→ 23→ 12→ 7→ 8→ 10→ 8→ 7→ 12→ 23→ 65→ 38→ 41→ 45→ 44

9. Read exmap3.txt, key at 205, treasure at 1004.
Print: Key is not found.

10. Read exmap3.txt, key at 429, treasure at 1004.
Print: 559→ 393→ 537→ 454→ 406→ 448→ 428→ 429→ 428→ 448→ 406→ 454→ 537→ 393→ 559→ 1020→ 808→ 898→ 1016→ 932→ 1002→ 1011→ 1005→ 1004

11. Read exmap3.txt, key at 910, treasure at 9.
Print: Treasure is not found.

12. Read exmap3.txt, key at 598, treasure at 642.
Print: 559→ 1020→ 808→ 779→ 706→ 611→ 592→ 600→ 596→ 597→ 598→ 597→ 596→ 600→ 592→ 611→ 667→ 624→ 644→ 641→ 642

3 Readme, comments and style (10%)

An indicator for good source code is readability. To keep source code maintainable and readable, you should add comments to your source code where reasonable. A consistent coding style also help a lot in reading source code.

For this assignment, please also compose a small readme-file in *.txt format and name it "README.TXT". This file should contain a brief explanation of how to use your program. Please remember to have your source code comments and readme file in English.

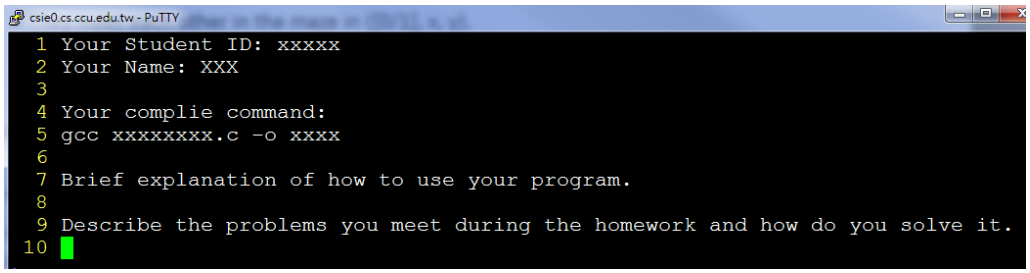

Figure 6: Readme file Format

4 How to submit

PuTTY(Figure 7) and **PieTTY** are free and open-source terminal emulator, serial console and network file transfer application. These two tools can help you to easily access the workstation of CSIE.
Host name: **csie0.cs.ccu.edu.tw**
To make sure that your program can run on the workstation, you **MUST USE a Makefile** to compile your program. If Makefile is not submitted or do not work well, you will not get the scores of HWB. Figure 8 shows the example of a Makefile.
The files you MUST submit:

1. Source Code

2. README.TXT

3. Makefile

To submit your file electronically, enter the following command in csie workstation:

- trunin ds.HWB [your files...]

To check the files you had turnin, enter the following command in csie workstation:

- trunin -ls ds.HWB

You can see other description about turnin from following link:
`https://www.cs.ccu.edu.tw/lab401/doku.php?id=turninhowto`

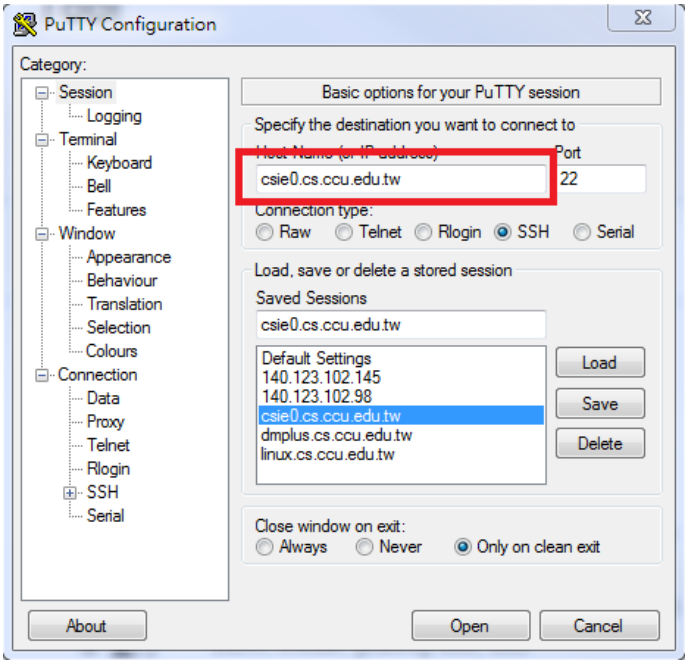

Figure 7: PuTTY

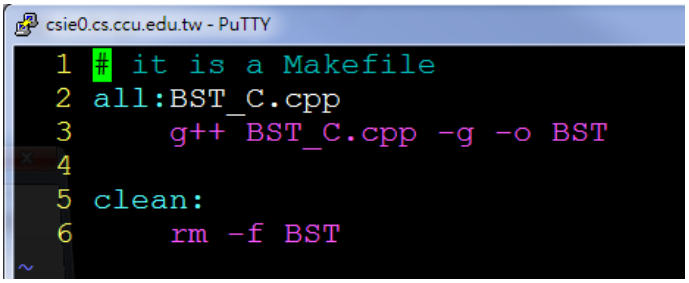

Figure 8: Makefile Example

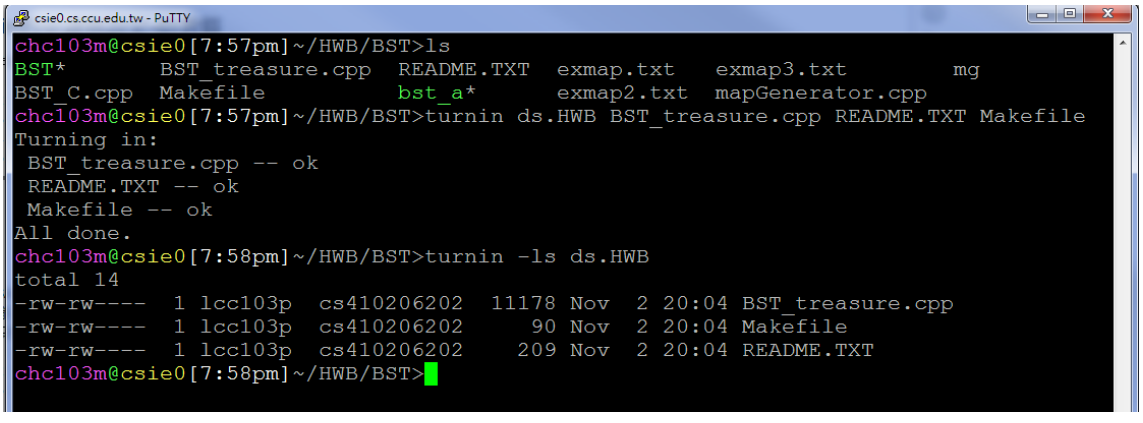

Figure 9: turnin Example

Grading (for TA)

The TA(s) will mark and give points according to the following grading polices:

BST.insert()	Implement insert function for BST. Handle duplicates and tree is always in correct order. The user gets a notice if a node already exists.	15%
BST.delete()	Implement delete function for BST. Tree must always be in correct order and correctly re-linked, if necessary. The user will be notified if a node could not be deleted.	10%
BST.search()	Implement search function for BST. Only the right sub-tree will be checked (no brute-force). The user will be notified if node exists or not.	10%
BST.printInfixOrder()	Prints the tree in inflix order. The order must be correct and all nodes must be printed.	5%
BST.printLevelOrder()	Prints the tree in level order. The order must be correct and all nodes must be printed.	5%
Treasure Hunter		45%
Readme, comments and style	Provide a README.TXT that contains information about the program. Source code is readable and has comments where reasonable.	10%