CS 202, Fall 2022

Homework 4 – Balanced Search Trees and Hashing

Due: 23:55, December 23, 2022

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:55, December 23, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw4.zip.
2. Your ZIP archive should contain the following files:
    - Your solution for Question 1 and Question 2 as a **pdf**. You should prepare and upload **handwritten** answers for Question 1 and Question 2 (in other words, do not submit answers prepared using a word processor). Use the exact algorithms shown in lectures.
    - Code files (only the "**.cpp**" and "**.h**" files that you write for the homework) of Question 3 and a "**Makefile**" for the compilation of your code that produces the executable.
    - Your Makefile should build an executable called "**converter**", so write your Makefile accordingly (i.e. at the end, when you type "make" in terminal on your working directory, it should produce "**converter**" executable)
    - In the end, your zip file should contain ONLY **code files**, "**Makefile**" and a **pdf**; any violation of these causes a significant loss from your grade.
3. Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as given below to the beginning of each file:

```
/*
 * Title: Balanced Search Trees
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 4
 * Description: description of your code
 */
```

4. This homework will be graded by your TA, Hakan Sivük. Thus, please contact him directly (hakan.sivuk at bilkent edu tr) for any homework related questions.

- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. If your C++ code does not compile or execute on that server, you will lose points.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

1) Question Part (25 points)

Assume that we have the following balanced-searched tree implementations:

a) 2-3 tree
b) 2-3-4 tree
c) Red-Black tree

Starting with an empty balanced search tree, we would like to insert the following keys into the tree in the given order:

2 3 5 4 6 7 8 1

and then, delete the keys 5 and 6 (in the given order) from the tree. Note: while deleting an internal node, its inorder successor should be used as the substitute if needed.

Show the structure for the 2-3, 2-3-4, and Red-Black trees after insertions and deletions with sufficient detail. For Red-Black tree, show only insertions. For other tree types above other than Red-Black trees, show both insertions and deletions.

2) Question Part (15 points)

Assume that we have a hash table with a size of 17 (index range is 0 . . . 16), and we use the mod operator as a hash function to map a given numeric key into this hash table. Draw the contents of the hash table after inserting each of the following keys

20 36 2 19 53 34 37

in the given order for each of the following methods:

a. open addressing with linear probing,
b. open addressing with quadratic probing,
c. separate chaining.

3) Programming Part (60 points)

The red-black tree, which is a special binary search tree, can be used to represent the 2-3-4 tree, so that we can retain the advantage of the 2-3-4 tree without storage overhead. In this question, you are asked to implement a program that ***converts a given red-black tree to its equivalent 2-3-4 tree.***

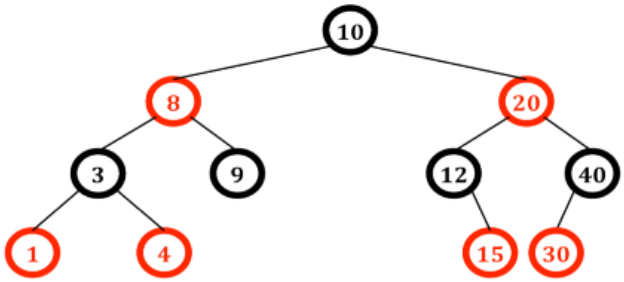
You will read the contents of a red-black tree from an input file, convert it to its equivalent 2-3-4 tree, and then write the contents of the resulting 2-3-4 tree into an output file. The formats of the input and output files are explained below.

The input file format:

The contents of the input red-black tree are stored in a plain text file[1]. The first line of this file contains the number of nodes and the id of the root node. The subsequent lines contain four integers, each separated by one or more whitespace characters (space or tab). These denote, respectively, the node id, the id of the left child, the id of the right, and the node type (0 for red nodes and 1 for black nodes). For the input, you may assume the following:

- The data file will always be valid. All data are composed of integers.
- The id of a node is used as the search key of the node. All ids are unique and non-negative integers.
- -1 corresponds to a NULL value. For example, if the id of the right child of a node is -1, then you should understand that this node does not have a right child.

For example, the following red-black tree is represented with the file content given below. From this file content, we understand that the red-black tree has 11 nodes and the root id is 10. The first node with id 12 is a BLACK node and has only a single right child with id 15 (it has no left child since the id of the left child is -1). The second node with id 10 is also a BLACK node and has two children (the left child has id 8 and the right child has id 20). The third node with id 20 is a RED node and has two children (the left child has id 12 and the right child has id 40). The fourth node with id 15 is also a RED node and does not have any children (there are -1 for both the ids of the left and right children). Here, note that the lines in the input file may be shuffled, (i.e. there is no relation between the order of the nodes in the red-black tree and the order of the nodes in the input file).




```
11    10
12    -1    15    1
10    8     20    1
20    12    40    0
15    -1    -1    0
8     3     9     0
3     1     4     1
9     -1    -1    1
1     -1    -1    0
30    -1    -1    0
40    30    -1    1
4     -1    -1    0
```

[1] The file is a UNIX-style text file with the end-of-line denoted by a single \n (ASCII 0x0A)

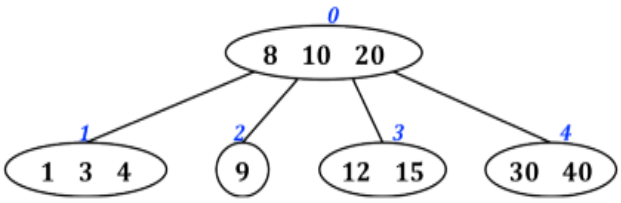
The output file format:

The contents of the equivalent 2-3-4 tree should be written in a plain text file. The first line of this file contains the number of nodes and the id of the root node. The subsequent lines contain eight integers, each separated by one or more whitespace characters (space or tab). These denote, respectively, the node id, the small search key, the middle search key, the large search key, the id of the left child, the id of the left-middle child, the id of the right-middle child, and the id of the right child.

Here, you will have to create IDs for each node instead of using the node search keys as IDs, since each node will potentially hold more than one key. In other words, as opposed to our red-black tree notation, in which we assume that the node IDs correspond to the node search keys; in the 2-3-4 tree notation, each node should have a unique virtual ID and could have up to three search keys. Please examine the following example to understand this notation better. In your implementation, you should create these node IDs (which will be auto-incrementing numbers) and you should use these node IDs to access the children of the nodes.

Similar to the red-black input file, -1 corresponds to a NULL value. In other words, you will use -1 to indicate that a given node does not have a particular child. Also use -1 to indicate unused values. For example, for a 2-node, we could have the small search key as well as two children (left and left-middle children). Thus, you should use -1 for the middle search key, the large search key, the right-middle child, and the right child.

For example, the 2-3-4 tree, which is equivalent to the above red-black tree, is shown as follows; the ID that is assigned to a particular node is represented by an integer written at the top of this particular node in blue. The contents of the output file are also given below.



5	0						
0	8	10	20	1	2	3	4
1	1	3	4	-1	-1	-1	-1
2	9	-1	-1	-1	-1	-1	-1
3	12	15	-1	-1	-1	-1	-1
4	30	40	-1	-1	-1	-1	-1

The name of the input file and the name of the output file will be provided as command line arguments to your program. Thus, we call your programs using two command line arguments in the following format:

username@dijkstra:~>./converter <inputFileName> <outputFileName