CS202, Spring 2023

Homework 2 - Binary Search Trees

Due: 10/11/2023

Before you start your homework, please read the following instructions carefully:

FAILURE TO FULFIL ANY OF THE FOLLOWING REQUIREMENTS WILL RESULT IN A GRADE SCORE OF 0 (zero) WITHOUT ANY CHANCE OF REDEMPTION.

- See the course page for late submission policies and the Honor Code for Assignments.
- This assignment has 3 questions, please make sure to read all pages carefully.
- Upload your solutions in a single ZIP archive using the Moodle submission form. Name the file as studentID_name_surname_hw2.zip. Your ZIP archive should contain **only** the following files:
  - **studentID_name_surname_hw2.pdf**, the file containing the answers to Questions 1, and 3.
  - `main.cpp`, `BSTNode.h`, `BSTNode.cpp`, `BST.h`, `BST.cpp`, `KmerTree.cpp`, and `KmerTree.h` files which contain the C++ source codes and the **Makefile**. Do not add unnecessary files like the auxiliary files generated from your preferred IDE.
  - Do not forget to put your name, student ID, and section number in these files. Comment on your implementation well. Add a header (see below) to the beginning of each file:

```
/**
 * Title: Binary Search Trees
 * Author: Name & Surname
 * ID:
 * Section:
 * Homework: 2
 * Description: description of your code
 */
```

- Your code must compile and be complete.
- Your code must run on the **dijkstra.cs.bilkent.edu.tr** server.
- You can submit your answer for Question 1 as scanned files, but make sure that the answers are legible. You may lose points if the handwriting or drawings are unclear.
- For any questions related to the homework, you can send an email to your TA: *zulal.bingol@bilkent.edu.tr* with the subject line “CS-202: HW-2”.

Question 1 (25 points)

- **a.** *(5 points)* Insert 82, 3, 6, 28, 4, 34, 87, 5, 8, 85, and 9 into an empty binary search tree in the given order. Show the resulting BST after every insertion.
- **b.** *(5 points)* What are the preorder, inorder, and postorder traversals of the BST you have after (a)?
- **c.** *(5 points)* Delete 28, 5, 82, 6, and 4 from the BST you have after (a) in the given order. Show the resulting BST after every deletion.
- **d.** *(5 points)* Write a recursive pseudocode implementation for finding the minimum element in a binary search tree.
- **e.** *(5 points)* What is the maximum and minimum height of a binary search tree that contains n items?

Question 2: Programming Assignment (55 points)

(a) *(15 points)* Implement a pointer-based Binary Search Tree whose keys are *strings*. Implement methods for insertion, deletion, and inorder traversal. The traversal method must return an ordered vector of keys. Put your code into the `BSTNode.h, BSTNode.cpp, BST.h, BST.cpp` files. Prototypes of the methods must be the following:

- `void BST::searchTreeInsert(string key); // 5 points`
- `void BST::searchTreeDelete(string key); // 5 points`
- `vector<string> BST::inorderTraversal(BSTNode* root); // 5 points`

(b) *(30 points)* You are to write a C++ program to count the frequency (number of occurrences) of *k-mers* in a text file. A k-mer is defined as *k* consecutive non-overlapping characters in a given text. Assume that the input text contains only English letters 'a'...'z', 'A'...'Z' and has no blank space. All k-mers should be stored as lower-case letters only. For example, consider the following text:

“ComputerScienceisFun”

All of the 3-mers generated from this text should be as follows:
“com”, “put”, “ers”, “cie”, “nce”, “isf”

Note that capital letters are converted to lowercase when k-mers are generated.

Your program should take the value of *k* as a parameter and construct the corresponding binary search tree (BST) according to the lexicographical order. You should use a pointer-based implementation of a BST to store the k-mers and their frequencies. (You can use the source codes available in the course book, or you can implement a BST yourself.) Each node object should maintain the associated k-mer as a string of size *k*, its current count as an integer, and left and right child pointers. On top of the regular operations that a BST has, you should implement the following functions:

- *(5 points)* `addKmer(string kmer)`: adds the specified k-mer in the BST if not already there; otherwise, it simply increments its count.

- *(5 points)* `generateTree(string fileName, int k)`: reads the input text from a file and generates a BST of k-mers. In this function, you should detect all the k-mers in the input text and add them to the tree using the add-kmer function. This function also requires the parameter *k*.
- *(5 points)* `getUniqueKmerCount()`: recursively computes and returns the total number of unique k-mers currently stored in the tree.
- *(5 points)* `getNumberOfKmerThatStartWith(char ch)`: recursively computes and returns the number of distinct k-mers that start with the given character. This function requires the input parameter for the first character.
- *(5 points)* `printAll()`: recursively prints each k-mer in the tree in alphabetical order along with their frequencies.
- *(5 points)* `getHeight()`: recursively computes and returns the height of the current tree.

For all these operations, your implementation should be efficient and should work on the BST directly. For example, you **cannot** copy all entries to a linear array, sort them, and return the results.

Implementation Details: *(0 points, but mandatory)* In the end, write a basic **Makefile** which compiles all your code and creates an executable file named **`hw2`**. Your program should take two command-line arguments: `<k>` and `<input_file>` in the same order. Please ensure your **Makefile** works properly; otherwise, you will not get any points from Question 2. Put the implementation of your functions in **`KmerTree.cpp`** and their interfaces in **`KmerTree.h`**. Do not include your main function in these files. Instead, submit your main function inside another file called **`main.cpp`**. Use the class names and function names exactly as listed above. Your homework will be tested with different testing scenarios. An example test file (`test0.cpp`) and the expected output file (`output0.log`) are attached to this assignment. Note that the given `test0.cpp` will not work as-is, without your KmerTree.h file. Please check that your output format is the same as shown above for each function as much as possible. You can use `diff` command in Linux to show the differences between your output file and the expected output. You will lose a significant amount of points if you do not follow these guidelines strictly.

(c) *(10 points)* In this part, you will analyze the time performance of the pointer-based implementation of binary search trees. For this, you need a large text input with at least 45000 characters. You can either write a function that creates a random text, or you can use an online text generator (e.g., https://onlinestringtools.com/generate-random-string).

Add a global function **`void timeAnalysis(string inputfilename, int k)`** into your `main.cpp` file to print out the following information periodically (e.g., after adding 1000, 2000, ... 10000 kmers) by calling `addkmer` function:

1. the number of unique kmers (i.e., the number of nodes in the BST)
2. the height of the BST
3. Time taken in each period (you can use the clock from **`ctime`** for calculating elapsed time)

Repeat this experiment with at least two different k values.

Question 3: Analysis (20 points)

After running your programs, you are expected to prepare a single-page report about the experimental results obtained in Question 2(c). With the help of a spreadsheet program (Google Sheets, Matlab, or other tools):

1. plot BST height versus the number of nodes in the BST. What does this function look like? Is it a linear function or a logarithmic function? Is this the expected behavior? Why?
2. plot the number of kmers added (not unique) versus elapsed time after each period of insertions to the BST. How would the time complexity of your program change if you inserted sorted kmers into it instead of randomly generated kmers?