CS 202 Fundamental Structures of Computer Science II
Assignment 2 – Binary Search Trees

Assigned on: 2 March 2016 (Wednesday)
Due Date: 14 March 2016 (Monday)

Question-1: Tracing [20 pnts]

a) Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given? Draw the binary search tree after each insertion.

3, 5, 8, 7, 4, 6, 1, 2, 9

b) What binary search tree is formed when you delete the following values in the order given? Draw the binary search tree after each deletion.

5, 8, 7

Question-2: Programming Assignment [60 pts]

You are to write a C++ program to count the frequency (number of occurrences) of *shingles* in a text file. A *k-shingle* is defined as k consecutive words in a given text. Assume that the input text contains only English letters 'a'...'z', 'A'…'Z' and the blank space to separate words. All k-shingles should be stored as lower-case letters only. For example, consider the following text:

“The most effective way to represent documents as sets is to construct from the document the set of phrases that appear within it”

Some of the 3-shingles generated from this text should be as follows:

“the most effective”, “most effective way”, “effective way to”, “way to represent”, …, “phrases that appear”, “that appear within”, “appear within it”.

Note that capital letters are converted to lower case when shingles are generated.

Your program should take the value of k as a parameter and construct the corresponding binary search tree (BST) accordingly. You are to use a pointer based implementation of a BST to store the k-shingles and their counts. (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-shingle as a string array 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:

- **addShingle**: adds the specified k-shingle in the BST if not already there; otherwise, it simply increments its count.
- **generateTree**: reads the input text from a file and generates a BST of k-shingles. In this function you should detect all of the k-shingles in the input text and add them to the tree by using the addShingle function. This function also requires the parameter *k*.

- **getDistinctShingleCount**: recursively computes and returns the total number of distinct k-shingles currently stored in the tree.
- **getNumberOfShinglesThatStartWith**: recursively computes and returns the number of distinct shingles that start with the given word. This function requires the input parameter *firstWord*.
- **printAllShinglesAndFreqs**: recursively prints each k-shingle in the tree in alphabetical order along with their frequencies.
- **printShinglesThatStartWith**: recursively prints in alphabetical order each k-shingle in the tree that starts with the given word along with its frequency. This function requires the input parameter *firstWord*.
- **isComplete**: recursively computes and returns whether or not the current tree is a complete tree.
- **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.

Below is the interface of a ShingleTree class for implementing the above functionality as well as a main function to test it with a sample input text file. Make sure that your code runs correctly against these and other test cases. We will test your program extensively. Also, make sure that your output format is exactly the same as shown in this example.

```
// shingle.h
…
//ShingleTree class
class ShingleTree
{
public:
      ShingleTree();
      ~ShingleTree();
      void addShingle(string shingle[]) ; // string array of size k
      void generateTree(string fileName, int k);
      int getDistinctShingleCount();
      int getNumberOfShinglesThatStartWith(string firstWord) ;
      void printAllShinglesAndFreqs();
      void printShinglesThatStartWith(string firstWord) ;
      bool isComplete();
      int getHeight() ;
…
private:
…
};
…


// shingle.cpp
#include "shingle.h"
#include <stdlib.h>
…
// main function
int main(int argc, char **argv)
```

```
{
	ShingleTree tree;
	string fileName(argv[1]);
	int k = atoi(argv[2]);




	tree.generateTree(fileName, k);

	cout << “Distinct ” << k << “-shingle count: ”
	     << tree.getDistintShingleCount()) << endl;
	tree.printAllShinglesAndFreqs();

	cout << endl << k << ”-shingle tree is complete: "
	     << (tree.isComplete() ? "Yes" : "No") << endl;

	cout << “Height of the ” << k << “-shingle tree: ”
	     << tree.getHeight() ;

	string firstWord = “peppers” ;
	cout << endl << “There are ”
	     << tree.getNumberOfShinglesThatStartWith(firstWord) << “ ”
	     << k << “-shingles that start with ” << firstWord << “: ”
	     << endl ;
	tree.printShinglesThatStartWith(firstWord) ;

	firstWord = “rain” ;
	cout << endl << “There are ”
	     << tree.getNumberOfShinglesThatStartWith(firstWord) << “ ”
	     << k << “-shingles that start with ” << firstWord << “: ”
	     << endl ;
	tree.printShinglesThatStartWith(firstWord) ;

	return 0 ;
}

// input.txt
If Peter Piper picked a peck of pickled peppers where is the peck of
pickled peppers Peter Piper picked

// Sample output
Total distinct 2-shingle count: 13
a peck: 1
if peter: 1
is the: 1
of pickled: 2
peck of: 2
peppers peter: 1
peppers where: 1
peter piper: 2
```

```
picked a: 1
pickled peppers: 2
piper picked: 2
the peck: 1
where is: 1

2-shingle tree is complete: No
Height of the 2-shingle tree: 5

There are 2 2-shingles that start with peppers:
peppers peter: 1
peppers where: 1

There are 0 2-shingles that start with rain:

// End of example
```

Implementation Details: Put the implementation of your functions in **shingle.cpp** and their interfaces in **shingle.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. Also, make sure that your output format is exactly the same as shown above for each function. ***You will lose significant amount of points if you do not follow these guidelines exactly.***

Question-3: Analysis [20 pnts]

In this part, you are going to analyze the height of your BST as a function of the number of nodes. For this, find a large text with at least 10000 words (e.g. You can copy paste part of an e-book such as https://www.gutenberg.org/files/28054/28054-h/28054-h.html to a text file. You can filter out the non-letter characters such as punctuations or just ignore them.) Modify your main.cpp program in Question-2 to print out the following information periodically (e.g. after reading 1000, 2000, … 10000 words):

1) the number of distinct shingles, i.e. the number of nodes in the BST
2) the height of the BST

Then, with the help of a plotting tool, graphically present the BST height as a function of the number of nodes in the BST. What does this function look like? i.e., Is it a linear function or a logarithmic function? Is this the expected behavior? Why?

For this question, you need to include a plot as explained above and a few sentences about what the function looks like and whether it is the expected behavior. You do not need to submit any code for this question.

HAND-IN

- Before 23:59 of 14 March, 2016, upload your solutions using the following online submission form, http://www.cs.bilkent.edu.tr/~saksoy/courses/cs202-Spring2016/upload.html. You should upload a single zip file that contains

  - hw2.pdf, the file containing the answers to Questions 1 and 3
  - shingle.cpp, shingle.h, and main.cpp, the files containing the C++ source code of Question 2
  - readme.txt, the file containing anything important on the compilation and execution of your program in Question 2.
  - **<u>You should prepare the answers of Questions 1 and 3 using a word processor (in other words, do not submit images of handwritten answers).</u>**
  - Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities.
  - Do not forget to put your name, student id, and section number, in all of these files. Well comment your implementation.
- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work in a Linux environment (specifically using the g++ compiler). We will compile your programs with the g++ compiler and test your codes in a Linux environment. <u>Thus, you may lose a significant amount of points, if your C++ code does not compile or execute in a Linux environment</u>.
- Keep all the files before you receive your grade.
- This homework will be graded by your TA, Nabil AbuBaker (nabil.abubaker at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

<u>IMPORTANT:</u> For this assignment, you may use the codes given in your textbook and the slides. However, you are <u>NOT</u> allowed to use any codes from other sources (including the codes given in other textbooks, found on the internet, and belonging to your class mates). Furthermore, you are <u>NOT</u> allowed to use any data structure or function from the C++ standard template library (STL).