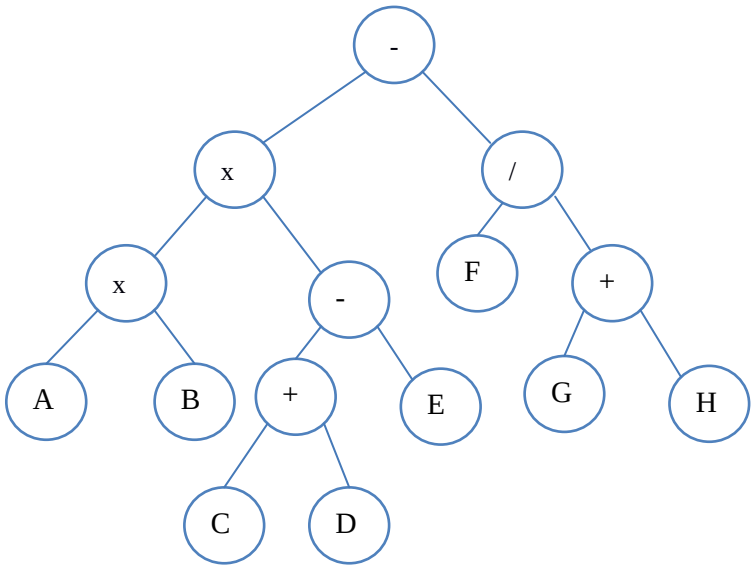
CS 202 Fundamental Structures of Computer Science II
Assignment 2 –Trees
Date : 19 October 2015
Due Date: 3 November 2015 23:59 (Tuesday)

Q1) [10 pts]

Give the prefix, infix, and postfix expressions obtained by preorder, inorder, and postorder traversals, respectively, for the expression tree below:



Q2) [10 pts]

Draw the initially empty Binary Search Tree after operations as follows (show all intermediate steps):

insert 42, 27, 31, 19, 89, 62, 100, 83, 22, 75, 52, 12, 72, 90; then delete 52, 22, 42.

Q3) Programming Assignment [60 pts]

You are to write a C++ program to count the frequency (number of occurrences) of n-grams in a text file. Definition of n-gram is simple: it is the number of consecutive letters in a given text. For example, for the word bilkent the 2-grams (bigrams) are bi, il, lk, ke, en, nt. You may ignore any capitalizations and assume that the text file contains only English letters 'a'...'z', 'A'…'Z', and the blank space to separate words. Your program should take the value of *n* as a parameter and construct the corresponding BST accordingly. While processing the input text, if your program encounters a word that has length smaller than the value of parameter *n*, you can simply ignore that word and process following words.

You are to use a pointer based implementation of a **Binary Search Tree** (BST) to store the n-grams and their counts. (You can use the source codes available in the course book as well as you can implement a BST yourself.) Each node object is to maintain the associated n-gram as a string, its current count as an integer, and left and right child pointers. On top of the regular operations that a BST has, you must implement the following functions:

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

- **getTotalNgramCount**: recursively computes and returns the total number of n-grams currently stored in the tree.
- **printNgramFrequencies**: recursively prints each n-gram in the tree in alphabetical order along with their frequencies.
- **isComplete:** computes and returns whether or not the current tree is a complete tree.
- **isFull:** computes and returns whether or not the current tree is a full tree.

Below is the interface of an `NgramTree` class for implementing the above functionality as well as a `main` function to test it with a sample input text file. These will be used for evaluation purposes. Make sure your code runs correctly against these. We will test your program extensively.

```
// hw2.h
…
//NgramTree class
class NgramTree
{
public:
      NgramTree();
      ~NgramTree();

      void addNgram(string ngram);
      int  getTotalNgramCount();
      void printNgramFrequencies();
      bool isComplete();
      bool isFull();
      void generateTree(string fileName, int n);

      …
private:
      …
};
…
```

```
// hw2.cpp
#include "hw2.h"
#include <stdlib.h>

…
// main function
int main(int argc, char **argv)
{
      NgramTree tree;
      string fileName(argv[1]);
      int n = atoi(argv[2]);
      tree.generateTree(fileName, n);

      cout << "\nTotal " << n << "-gram count: "
                << tree.getTotalNgramCount()) << endl;

      tree.printNgramFrequencies();

      cout << n << "-gram tree is complete: "
                << (tree.isComplete() ? "Yes" : "No") << endl;

      //Before insertion of new n-grams

      cout << "\nTotal " << n << "-gram count: "
                << tree.getTotalNgramCount()) << endl;

      tree.addNgram("samp");
      tree.addNgram("samp");
      tree.addNgram("zinc");
      tree.addNgram("aatt");

      cout << "\nTotal " << n << "-gram count: "
                << tree.getTotalNgramCount()) << endl;

      tree.printNgramFrequencies();

      cout<< n << "-gram tree is complete: "
                << (tree.isComplete() ? "Yes" : "No") << endl;

      cout<< n << "-gram tree is full: "
                << (tree.isFull() ? "Yes" : "No") << endl;


      return 0;
}


 // input.txt
 this is sample text
 and thise is all

 // Sample output
 Total 4-gram count: 6
```

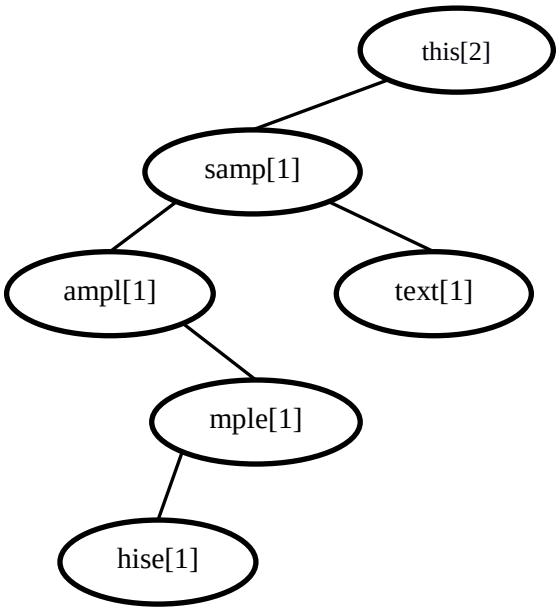
```
"ampl" appears 1 time(s)
"hise" appears 1 time(s)
"mple" appears 1 time(s)
"samp" appears 1 time(s)
"text" appears 1 time(s)
"this" appears 2 time(s)

4-gram tree is complete: No
4-gram tree is full: No


Total 4-gram count: 8
"aatt" appears 1 time(s)
"ampl" appears 1 time(s)
"hise" appears 1 time(s)
"mple" appears 1 time(s)
"samp" appears 3 time(s)
"text" appears 1 time(s)
"this" appears 2 time(s)
"zinc" appears 1 time(s)

4-gram tree is complete: No
4-gram tree is full: No
```

The following is the BST constructed for the input text whe $n = 4$.



Q4) [20 pts]

Analyze the worst-case running time complexities of the **addNgram** and **printNgramFrequencies** functions in the previous question using the big-oh notation.

HAND IN

- Before **23:59** of November 3, 2015, upload your solutions using Moodle .You should upload a single zip file that contains
    - hw2.pdf, the file containing the answers to Questions 1,2 and 4, and the sample output of the program.
    - hw2.cpp, hw2.h, and main.cpp, and any additional files if you wrote additional classes in your solution, and
    - readme.txt, the file containing anything important on the compilation and execution of your program in Question 3.
    - You should be able to compile your program on a Linux terminal with the following command:
        - g++ *.cpp –o hw2
    - Do not forget to put your name, student id, and section number, in all of these files. Well comment your implementation.

- For this assignment, you must use your own implementation of binary search trees. In other words, you cannot use any existing binary search tree code from other sources such as the binary search tree class in the C++ standard template library (STL). However, you can adapt the binary search tree codes in the Carrano (5th ed.) or Carrano-Henry (6th ed.) book. You will get no points if you do not use binary search trees as indicated.

IMPORTANT: Although you may use any platform and any operating system in implementing your algorithms and obtaining your experimental results, your code should work in a Linux environment with the g++ compiler. We will test your codes in a Linux environment. Thus, you may lose a significant amount of points, if your C++ code does not compile or execute in a Linux environment.

- Keep all the files before you receive your grade.

- This homework will be graded by your TA Cem Orhan (cem.orhan at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!