CS221-02: Comp Sci II: Data Structures

Spring 2019

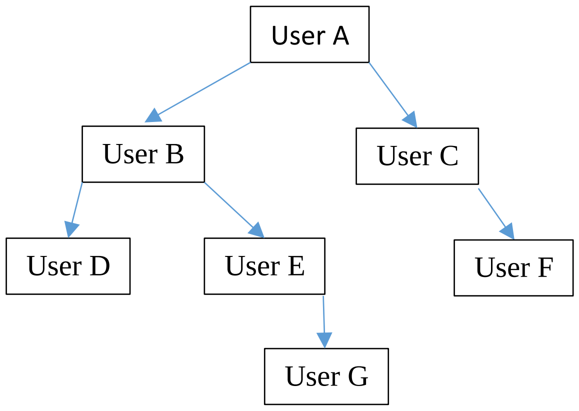
Programming Assignment 4

1.0 Overview

The graph data structure can be used to analyze complex problems in a simpler way. Graphs have various applications from actual computer networks to social networks. This programming assignment will help you utilize the graph data structure.

Main idea:

Assume that the users are inserted into a binary search tree using their birthdays. We would like to analyze kinship based on the built tree. In the following tree, User D and User E are siblings. User B is an ancestor of both User D and User G.



While searching in a tree is effective, other operations on a tree are not always easy using a tree structure. You will map the tree relationships to a graph. Graph G=(V,E) is composed of vertices V and a set of edges between vertices. In this assignment, vertices are users. You will build a directed graph. An edge from user u_1 to u_2 indicates that user u_1 is the parent of user u_2. Edges are not weighted.

You will represent the graph as an adjacency matrix. Instead of maintaining weights for edges, maintain a boolean value for the presence of edges between users. You need to use 1D and 2D dynamic arrays, since the number of users is not known ahead of time.

2.0 Requirements

2.1 (Class GraphType) Implement GraphType whose specification is provided in Chapter 13 (pages 740-744). There will be some differences between your implementation and the one in your book. Create GraphType.h and GraphType.cpp files.

- **(Member Functions)**
  - **GraphType(int noVertices)**: Constructor to allocate the memory for vertices and edges assuming noVertices is the number of users. (Similar to the book)
  - **void AddVertex(User* user):** Function to add a user to the graph. (Similar to the book)
  - **void AddEdge(User* fromVertex, User* toVertex):** Function to add an edge from one user to another user. (Similar to the book)
  - **User* findSibling(User* user1):** This function returns the sibling of user 1 (in the tree) using the graph.
  - **void DisplayAncestors(User* user1):** This function prints the ancestors of user 1 (in the tree) using the graph.
  - <u>******BONUS (15%):**</u> **User* FindEarliestCommonAncestor(User* user1, User* user2):** This function returns the earliest common ancestor between any two users. The earliest common ancestor for User A and User G is User B, whereas the earliest common ancestor for User D and User F is User A.
  - <u>******BONUS (10%):**</u> Just come to my office (preferably during my office hours) ask a question or tell me your progress on the assignment. If you cannot meet me during my office hours, I'll have some extra limited time after lectures.
- **(Member Attributes)**
  - **int numVertices:** number of vertices (users) in the graph
  - **int maxVertices:** maximum number of vertices (users)
  - **User** vertices:** 1D dynamic array of user pointers
  - **bool** edges:** 2D dynamic array of edges
- **(Helper Function)**
  - **int IndexIs(User** vertices, User* vertex):** returns the index of user in vertices list.

2.2 (Enhance Class TreeType)

- Add the following function:
  - **void BuildVertexGraph(GraphType& net):** This will add users to the graph calling AddVertex function and add tree edges by calling AddEdge function.

2.3 (Dynamic Arrays)

You will allocate space for dynamic arrays in the GraphType constructor. I also provide a partial code for you.

- 1D dynamic array of user pointers:
  - vertices = new User*[maxVertices];
  - To access a specific user's last name: vertices[i]->GetLastName(lname);

- 2D dynamic array of edges: allocate memory for the first dimension and then allocate space for the second dimension in a loop.
    - edges = new bool*[maxVertices];
    - for (int i = 0; i < maxVertices; i++)
    -     edges[i] = new bool[maxVertices];

  to access edges, call edges[i][j]

2.7 (Testing) Enhance TestDriver class. Create the specification file (TestDriver.h) and implementation file (TestDriver.cpp). This file should have

- **int Populate (const char* input, TreeType& treeUsers)** method is a similar method to the one in Assignment 3, but it returns the number of users as an output.

  Sample text file is available at this link.

  Note: Read the following file from your project: "C:\tmp\cs221\hw1\hw1samplefile.txt"

  Also write your outputs to the directory "C:\tmp\cs221\hw3\". Each output file should start with your username (e.g., raygunTesthw3sampleoutput.txt).

- Add one more function as **void TestGraph(std::ofstream& outFile,GraphType& userNet)** for testing experiments regarding the graph. In order to test the class functions TestGraph should evaluate the functions of the Graph class.
- Add one function **void Test(char* outfilename,GrapghType& userNet)** to TestDriver class. This function will call void TestGraph(std::ofstream& outFile,GraphType& userNet) and write the output to a file.
- The main() function should create an object of TestDriver and call Test function.

I'll provide a sample main function for you as follows:

```
int main()
{
    TestDriver tester;
    TreeType treeUsers;
    GraphType userNet;
    Int noUsers;

  noUsers=tester.Populate("C:\\tmp\\cs221\\hw1\\hw1samplefile.txt"
  , treeUsers);

   GraphType userNet(noUsers);
   treeUsers.BuildVertexGraph(userNet);

   tester.Test("C:\\tmp\\cs221\\hw4\\raygunhw4outputs.txt",
     userNet);

   return 0;
}
```

Note: If you had difficulty of completing assignment regarding building the tree, please let me know. The code is provided in this form assuming you completed the previous assignment.

Note: Neither your instructor nor the course GTA is your tester. You should develop a testing strategy and test every method in your code. Error detected during grading will result loss of points. Make sure that you test your program carefully. We will go over testing strategies in class.

***** NOTE *** (Helping Partner)** You may choose a helping partner for this assignment. Helping partner may look into your code and provide suggestion and feedback on your code. You helping partner cannot share his or her code with you and cannot code the assignment for you. Everyone should still work on his or her assignment.

3.0 Commenting Code

Your code should be commented well. Each method should have preconditions and post conditions listed after method declaration in the implementation files.

4.0 Deliverables

These products shall be delivered to the instructor electronically via Canvas as specified below.

4.1 Program source files -- The student shall provide fully tested electronic copies of the .cpp and .h files. These files should be uploaded to Canvas as a **single zip file**.

4.2 Software Description Document – The student shall provide general description of the software in a structured way as mention in Programming Assignments document.

4.3 Software Test Plan Document – Results of test experiments should be provided. If class members are updated, they should be displayed after being updated.

4.4 NOTE about Software Documents: You do not need to include sections from old documents. You may just provide the new additions to keep software documents for this assignment. We will focus on grading sections regarding this assignment.