Homework #7 & #8

AVL Tree & STL

Yunmin Go

School of CSEE

HANDONG GLOBAL UNIVERSITY

HW#7

- Write a delete function for the AVL Tree.
  - Implement a delete function for AVL Tree in the AVLTree.cpp we have implemented in Practice #14.
  - Use the following declaration for the Delete()
    - `void Delete(element x);  // Delete a node whose key is x.key from tree`
      - `ex) x.key = 6; avl->Delete(x);`
      - Please refer to AVLMain2.cpp.
  - You can use Insert() and Print() implemented in the AVLTree.cpp already.

  - File name: AVLTree2.cpp

HW#8

- Re-implement the Practice #11 using STL
  - In the Practice #11, we have implemented a Graph.cpp for undirected graph class. In Graph.cpp, we used the array for adjacency list and the stack and queue classes implemented by us.
  - In this homework, use proper STL to implement Graph.cpp again.
  - Your program should also display the results of DFS and BFS and the all adjacency lists as shown in Practice #11.

  - File name: Graph2.cpp

Requirements

- All of C-style functions and headers are allowed.
  - E.g., printf, fopen, fgets, etc.

- Write clean source code
  - Add proper comment in your source code
  - Consider code indentation for enhancing readability

- Submit your screenshots.

Requirements

- For unmentioned requirements, you can implement freely.

- Test your source codes with many cases for self verification.

- Upload ZIP file on LMS by compressing all your source codes and screenshots
    - File name: hw7&8_student id.zip (ex: hw7&8_20400022.zip)

- Due date: 11pm, 6/22 (Tue)

HANDONG GLOBAL UNIVERSITY

PRACTICE #11

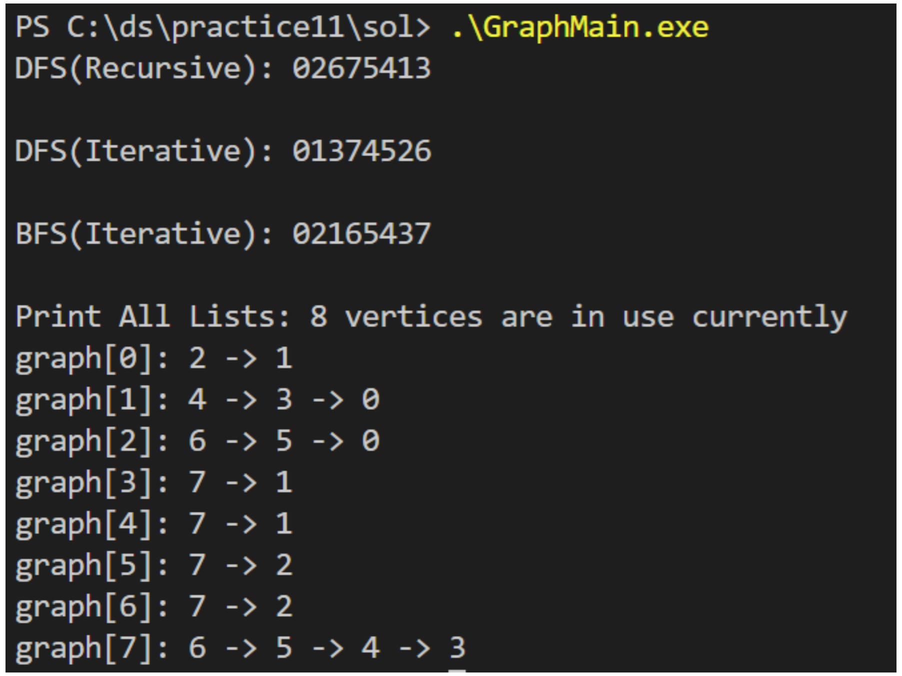
Graph DFS & BFS

- Implement a graph class for undirected graph
  - Complete a Graph.cpp (GraphMain.cpp: no need to change)
    - Graph.cpp defines a Graph class and its member functions
  - Refer to p.33, 35, 39, Chapter 6
  - We use adjacency list for graph representation.
  - Implement following member functions. You can modify the source codes and add additional member functions.
    - InsertEdge(int *src*, in *dest*): insert an edge between vertex *src* and vertex *dest*
      - Add new node at head of list (i.e., graph[])
    - DFS_recur(int *v*): iterative DFS algorithm (starts from vertex *v*)
    - DFS_iter(int *v*): iterative DFS algorithm (starts from vertex *v*)
    - BFS_iter(int *v*): iterative BFS algorithm (starts from vertex *v*)
    - PrintAdjList(): print all adjacency lists in graph[]

※ You can use given Stack.cpp and Queue.cpp for DFS and BFS.

Graph DFS & BFS

- Expected results

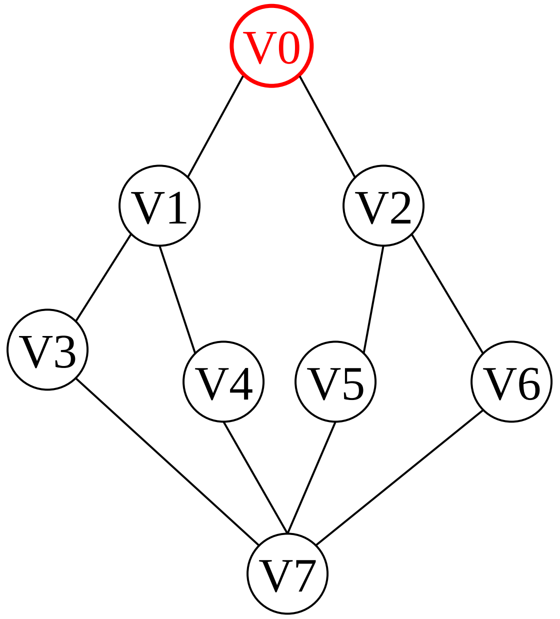


```
PS C:\ds\practice11\sol> .\GraphMain.exe
DFS(Recursive): 02675413

DFS(Iterative): 01374526

BFS(Iterative): 02165437

Print All Lists: 8 vertices are in use currently
graph[0]: 2 -> 1
graph[1]: 4 -> 3 -> 0
graph[2]: 6 -> 5 -> 0
graph[3]: 7 -> 1
graph[4]: 7 -> 1
graph[5]: 7 -> 2
graph[6]: 7 -> 2
graph[7]: 6 -> 5 -> 4 -> 3
```


V0
V1
V2
V3
V4
V5
V6
V7