CS 372/469
Fall 2018
Lab 5: Due 11/5, before lab

1 Basic graph class/methods

The purpose of this lab is to get some experience with graph algorithms, representing graphs, and storing graphs. In a graph, each vertex is represented as a class node with a name and an id number. To make things simple, we will consider representing the entire graph as an adjacency list (an adjacency matrix would've worked just as fine too). As an example, here is the C++ code for representing a single node, as well as methods for populating its adjacency list[1]. This is just given as an example of the classes/methods required, you can feel free to write similar classes and methods in any language of your choice.

```
class Node {

private :
string m_name; // a string that labels the node
int m_id; // a unique integer from 0 to n-1, where n is the total number of nodes
// i.e., n =|V|

public :
Node() {};
Node(const string & name, int id)
{m_name = name, m_id = id;};
int id() const { return m_id;};
const string & name() const { return m_name;};
```

Design a graph class using adjacency list, it must contain the following members and functions:

```
class Graph {
private :
vector< Node > m_nodes;
vector < list <Node> > m_adjList;

public :
// Construct the graph from a file of edges
Graph(const string & file );
```

[1]Code snippet from CS 372/469 Fall'17.

```
// Insert a edge (a, b) to m_adjList
void addEdge(const Node & a, const Node & b);

// Insert a node a to m_nodes
void addNode(const Node & a) { m_nodes[a.id()] = a;};

// Return node with id equal to i
const Node & getNode(size_t i) const { return m_nodes[i]; }

// Return reference of the adjacency list of node a
list <Node> & getAdjNodes(const Node & a) { return m_adjList[a.id ()];}

// Return constant reference to adjacency list of node a
const list <Node> & getAdjNodes(const Node & a) const
{ return m_adjList[a.id ()];}

// Return the total number of nodes in the graph
size_t num_nodes() const { return m_nodes. size (); }

// Create a graph from a tab-separated text edge list file to adjacency lists
void scan(const string & file );

// Save a graph from adjacency lists to a tab- separated text edge list file
void save(const string & file) const;
};
```

Once this is done, we need a file format to store into, and read graph data from in our program. One way to do this is to store the graph as tab separated text file containing a list of edges in the graph, specified by the source and destination nodes of each edge, one on each row, e.g.,

```
a b
a d
b d
c b
d c
```

There are several visualization tools such as R and Matlab for visualizing graphs, you can use them if you'd like, and are familiar with them, but are not required to do so for this lab.

2 DFS

Now, implement the DFS algorithm we had talked about in class – textbook algorithm 3.3, and 3.5. Use the graph text file as G. You need to make small changes or enhancements to the code above to remember the pre-visit (first time a node is visited), and post-visit (last time we depart from a node) numbers.

3 Testing the DFS algorithm

Generate five example graphs to test your code. The graphs do not have to be very large but try to represent a variety: cyclic/acyclic, directed/undirected.

4 Report: around 2 pages

Write a lab report to describe your lab work done in the following sections

1. Introduction (define the background, motivation, and the problem)
2. Methods (provide the solutions)
3. Results: show plots for empirical runtime on graphs. Discuss if they appear to be $O(|V| + |E|)$ – which is the expected time for DFS. If not, explain, or speculate on the possible reasons.
4. Discussions (general implications and issues)

How to submit: Upload your **pdf** file and source code on Canvas before the due date.