Lab 4 Constructing and visualizing graphs

C S 372 Data Structures and Algorithms

September 11, 2019

In this lab, you will design data structures to represent graphs using adjacency lists, read and write text files that encode a graph, and visualize the graphs in R.

1 Data structure for graphs

The node class is provided to you and contains:

```
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

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);

  // 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;
};
```

The overloaded i/o operator “«” is also provided as follows

```
std::ostream& operator<<(std::ostream& out, const Graph & g)
{
  out << "Nodes in the graph:" << endl;
  for(unsigned i=0; i<g.num_nodes(); i++) {
    out << g.getNode(i).name() << ",";
  }
  out << endl;
  out << "Adjacency list of the graph:" << endl;
  for(unsigned i=0; i<g.num_nodes(); i++) {
    out << "Node " << g.getNode(i).name() << ":";
    const list<Node> neighbors = g.getAdjNodes(g.getNode(i));
    for(list<Node>::const_iterator itr = neighbors.begin();
      itr != neighbors.end(); ++itr) {
      out << itr -> name() << ",";
        }
        out << endl;
  }
  return out;
}
```

You must utilize the classes to generate all the required tests. If your code deviates from the given template, please explain in your lab report why your choice may be better in terms of time and space complexity.

2 File input and output for graphs

The graph file format is *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:

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

This graph is visualized in the next section.

3 Visualization graphs in R

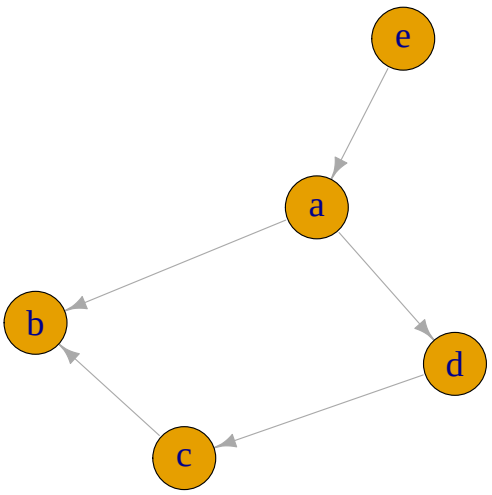
The interface between R and C++ code will be via graph text files, as the graph class cannot be easily passed between R and C++.

You can easily visualize your graph in R with the R package `igraph`. Please install the package first. If you want to use it on home computer, you will need to install it by calling`install.packages("igraph")`. R will install a few other R packages that `igraph` requires.

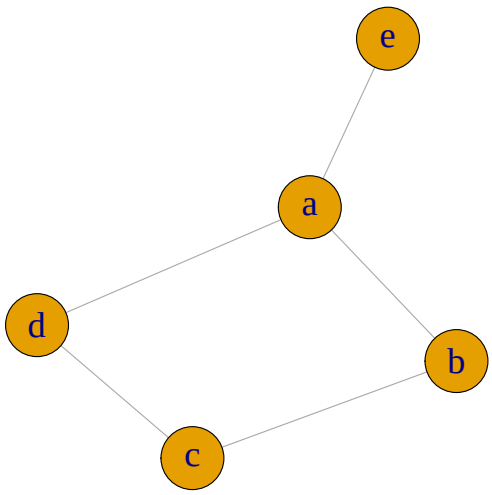
```
require("igraph")
links <- data.frame(from=c("a", "a", "c", "d", "e"),
                    to=c("d", "b", "b", "c", "a"))
net <- graph_from_data_frame(d=links, directed=T)
plot(net, vertex.size=30, vertex.label.cex=2)

net <- graph_from_data_frame(d=links, directed=F)
plot(net, vertex.size=30, vertex.label.cex=2)
```

Directed graph:



Undirected graph:



When the graph is large, you will use `read.table()` function in R to load the edges instead of typing them in your program.

You can read the online tutorial "Network visualization with R" by Katya Ognyanova for how to generate a variety of graph visualizations in R at

`http://kateto.net/network-visualization`

An R script file `random_graph.R` was provided to plot graphs from files and generate random graphs.

4 Test the classes

You will generate five graph text files containing a list of edges each. You must generate some large graphs containing at least 1,000,000 edges and 10,000 nodes, which can be randomly generated in R. Call you C++ program to scan and save it.

For small graphs, compare the visualization of the graph from the initial file and the one that was read from the saved file by your graph class. The graphs may not look identical due to the rearranged edge order, but must be identical in structure.

For large graphs, compare the text file and confirm they contain the same set of edges regardless of the order.

Your C++ program must include a main() function that calls the `testall()` function. When the program is compiled by a C++ compiler it generates a binary executable file that will run when invoked from the command line.

5 Submission

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
   (a) show figures for small graphs.
   (b) report runtime on reading and saving large graphs.
4. Discussions (general implications and issues),
5. Conclusions (summarize the lab and point to a future direction).

Submit the source code files and your lab report online.