CSE 232 Spring 2014

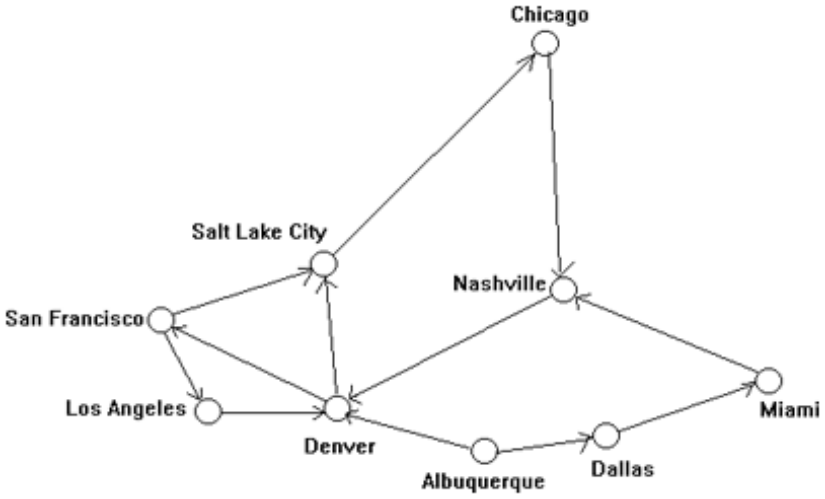
Programming Project #6

Assignment Overview
In this assignment you will practice using maps, along with previous stuff like vectors, random numbers, file input and output etc.

This assignment is worth 40 points (4.0% of the course grade) and must be completed and turned in before 11:59PM on Monday, March 10th . This is the first day of class after spring break.

Background
In class I showed you a recent experiment done that crawled for 3.5 billion web pages in a effort to see if they could measure those web locations that were the most **central**. http://webdatacommons.org/hyperlinkgraph/ . In a network like the web, centrality means something like "the most connected to node" in the network. We are going to write an algorithm to work with a very small subset of this data and make a determination of the most central nodes.

Graph
A graph is a data structure that is a natural way to represent a network like the web. A graph consists of nodes and arcs. A simple analogy is a city map. The nodes in the city map would be the cities and the arcs would be the roads that connect the cities. Not all cities are connected, and the connections might be directed, that is they might be one-way roads.



In the graph/map at the right, we see that there are 9 nodes/cities (Miami, Dallas, Nashville, Chicago, Albuquerque, Denver, Salt Lake, LA, San Fran) and 12 arcs/roads. The arcs are directed, that is one-way. For example, in our map the direct connection between Chicago and Nashville is one way. You can go from Chicago to Nashville by the arc, but not directly back.

Central
Which of the nodes in the graph above are the most central? By visual inspection we would likely say that Denver is the most central as it has the most arcs, either in or out. How could we write an algorithm to discover, for a large graph, which is the most central?

It turns out that there are a number of ways to determine centrality of a node in a graph, but here is an easy one and the one we will use: a random walk algorithm.

Random Walk
A random walk is an general computation approach that allows us to model processes like the stock market, molecular dynamics and other problems (http://en.wikipedia.org/wiki/Random_walk). We will use a random walk to gather statistics about node centrality. Here it is how it works on our city map. We will keep a **visit_count** data structure of how many times each city is visited if we take a random walk through our map. We limit our random walks to a particular length; let's say 3 for our example. Here we go.

- Select a random city, let's say San Francisco.
- Update the **visit_count** for San Francisco by +1
- randomly pick an arc out of San Francisco (there are two). Let's pick Los Angeles
- Update the **visit_count** for L.A. by +1
- randomly pick an arc out of L.A. There is only one, to Denver.
- update the **visit_count** for Denver by +1.
- randomly pick an arc out Denver (there are two). Let's pick Salt Lake
- Update the **visit_count** for Salt Lake by +1

That's the end of one random walk of length three. What would the **visit_count** look like for each city if we conducted hundreds or even thousands of length 3 random walks? In the end, the most central city would be that city that was visited the most on our walks, in this case the city that had the highest **visit_count** . In fact, you might say we could rate the centrality of each node in the graph by sorting the cities on their visit counts.

Assignment
You are going to determine the centrality of nodes in a graph of the web using a random walk algorithm. The data will be provided in two files (in the directory):

- *example_nodes.txt*
    - Each line of the file provides two pieces of information:
        - the name of the website (string)
        - an ID number to use for the website (long)
- *example_arcs.txt*
    - Each line of the file provides two pieces of information
        - an ID number where an arc begins (long)
        - an ID number where an arc ends (long)

You will read in both of the above files, providing all the information you need to do a random walk through the graph.

Three Data Structures
You will need three maps for this project, and here they are.

- `map<long, string> index_map`
    - map of ID to website name

- `map<long, vector<long>> arc_map`
    - map of a website ID to vector of all the website ID's it connects to.
- `map<long, long> visit_count`
    - map of ID to the count of visits

Functions

- `void` **`read_index`**`(map<long, string> &m, string &file_name)`
    - reads the `file_name` in as an index file (example_nodes.txt).
- `void` **`read_arc`**`(map<long, vector<long>> &m, string &file_name)`
    - reads the `file_name` in as an arc_file (example_arcs.txt)
- `long` **`select`**`(default_random_engine &dre, vector<long> &container)`
    - Returns a random value from a vector of long.
- `map<string, long>` **`do_walk`**`(map<long, vector<long>> &arc_map, map<long, string> &index_map, long total_walks, long walk_length, default_random_engine &dre)`
    - Does `total_walks` random walks of length `walk_length`. Returns a `visit_count` map with key `website_name` and value `visit_count`.
    - calls `select`

Requirements

As mentioned, we have provided a `main.cpp` which will test each of your functions. Your code must compile and produce correct output using our main **without modification.** We have provided `example_nodes.txt` and `example_arcs.txt`.

Deliverables

`rand_walk.h` and `rand_walk.cpp` -- your source code solution (remember to include your section, the date, project number and comments). Remember, you ***do not provide*** `main.cpp`

1. Please be sure to use the specified file names
2. Save a copy of your file in your CSE account disk space (H drive on CSE computers).
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient

Assignment Notes

- the arcs are one way.
- It might be the case that there is an arc into a node and but not an arc out.
    - under those circumstances you should end a particular walk.