Assignment 1

COT 5405

January 28, 2021

1 Introduction

For the first assignment, you will be playing with some graph algorithms. Use a compiled programming language such as C++ or Java **(not python)** for coding. We describe the project using C++. Use corresponding files/structures if you are using a different language. For the first part, you will create a collection of general graph functions. Your functions will apply to any graph, provided it is presented in the correct form. Then, in the second part you will translate movie rating data from Netflix into graphs and analyze the graphs with your implemented functions. For this assignment you should make liberal use of the STL structures, like list, map, set, and vector - but not any external libraries. For implementing the functions, you are expected to use appropriate data structures such that the run-time and memory consumption is optimal. Report your experiments, results and observations in a file **results.txt/.doc/.pdf**. Before getting started, read through the whole document, so you understand what resources are available and what you are required to do.

2 Graph algorithms

You are to write three graph functions corresponding to three graph problems discussed in class: shortest paths, connected components and cycles. First, describe and implement an appropriate data structure for storing a graph. **You may **not** use any existing library for building and storing a graph. Instead, you must implement one using basic data structures (lists, arrays, etc).** The three functions you need to implement are *connected_components*(), *one_cycle*() and *shortest_paths*(). Each function takes the graph data structure as an argument and performs the following function.

- *connected_components*() : uses depth-first search to return the list of connected components. Each component is itself a list. The order of the components, and the order of vertices within a component, is not specified.
- *one_cycle*() : uses depth-first search to return a cycle, if there is one. If the graph is acyclic, the result is an empty list. Otherwise, the list specifies a cycle as a list of three or more vertices that starts and ends with the same vertex. If multiple cycles exist than any cycle may be returned.
- *shortest_paths*() : uses Dijkstra's algorithm and returns a map of shortest paths. Notice that we are using unweighted graphs so the edge weights are intrinsically 1, but we'd still like you to implement it using Dijkstra's. Suppose that *sp* is the list of paths returned by the function for a certain source node. Then for each vertex v which is reachable from source, $sp[v]$ is the list of vertices on a path from v back to source. A path in our case consists of a list of vertices and should include both v and the source. The path from source to source is just the single node source. There may be several shortest paths; your function will provide only one. The vertex v will not appear in *sp* if there is no path between v and source.

3 Testing - Simulated data

Below are seven easy-to-generate graphs that you can use for testing. Pick the appropriate ones for testing your code. Simulate at least three of these graphs to test out your methods. Note the running time and memory consumption of the experiments. Vary the number of nodes and observe how the run-time and memory grows. Report a short description of your experiments and observations in your submission file, **results.txt**. Include the simulation scripts in your submission directory.

1. An n-cycle: The vertices are integers from 0 through $n-1$. The vertices u and v are connected by an edge if $u-v=\pm 1$ or $u-v=\pm(n-1)$. There is one connected component, every shortest path has length at most $n/2$, and there is a unique cycle of length n.
2. A complete graph on n vertices: The vertices are integers from 0 through $n-1$. Every pair of distinct vertices forms an edge. There is one connected component, every shortest path has unit length, and there are many cycles.
3. An empty graph on n vertices: The vertices are integers from 0 through $n-1$. There are no edges. There are n connected components, no paths, and no cycles.
4. A heap: The vertices are integers from 0 through $n-1$. The neighbors of a vertex v are $(v-1)/2$, $2v+1$, and $2v+2$, provided those numbers are in the range for vertices. There is one connected component, the paths are short, and there are no cycles.
5. A truncated heap: The vertices are integers from m through $n-1$. The edge relationship is the same as for the heap. There are $n-1-2m$ edges, $m+1$ connected components, and no cycles. The paths, when they exist, are short.
6. Equivalence mod k: The vertices are integers from 0 to $n-1$, where $k \leq n$. The vertices u and v are connected by an edge if $u-v$ is evenly divisible by k. There are k components, and each component is a complete graph.

4 Testing - Real data

Now that you have some working methods, you will construct graphs that are much larger than the examples above by using data from the Netflix-prize challenge. The data has been archived and can be downloaded from the following link `https://tinyurl.com/y6y4y4kw`. The files **ratings_data_1.txt** to **ratings_data_4.txt** in the main directory contains information about movies and their ratings by anonymous users. The format of the data is described in the file **assignment_readme**. The movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. The ratings are on a scale from 1 to 5 (integral) stars.

In a file named **graph_make.xxx** (the extension to this and other files will depend on the programming language you choose), write a function that reads the ratings data files and generates the graph where each user corresponds to a node and two users are connected by an edge based on an adjacency criteria. The criterion for adjacency can change, and you should experiment with several different alternatives (you will need to provide results for three in the end). An easy one with which to begin is that two viewers are adjacent if there is at least one movie that both viewers have rated. Another, stricter criterion is that the two viewers have three movies in common and they have given the same rating to each of the three movies.

Finally, write a main function that uses the methods above (including some from the first part) to print the number of connected components and size of each component in the graphs that you create. Run your program with three different notions of adjacency. In the submission file, **results.txt** describe your experiments and results. The main driver function for this part should be included in a file **real_test.xxx** which should also contain your definition of adjacency that produces the results that you deem most interesting.

5 Submission

Your submission should include a zip file of the format
Lastname_Firstname_assignment1.zip which should contain the following files.

- **graph_operations.xxx** contains the three functions from the first part.
- **graph_simulator.xxx** contains functions for generating simulated data using the suggested methods. Atleast three of the functions should be implemented.
- **simulated_test.xxx** contains the main function for generating simulated data and running the operations on it. Output from graph operations can be written into a separate file or just printed out.
- **graph_make.xxx** contains functions to read in the movie reviews data and create a graph based on some adjacency criteria. At least three functions should be tried with a different adjacency criteria. The criteria should be mentioned as comments.
- **real_test.xxx** contains the main function for generating a real graph based on some adjacency criteria and running the operations asked in the second part. Output from graph operations can be written into a separate file or just printed out.
- **results.txt/.doc/.pdf** This file contains short descriptions of your experiments and some sample outputs from your tests of simulated and real data. **Make sure to include description of your data-structures used for various operations and report peak memory usage and CPU time for all experiments - real and simulated.**

Also include any additional files or instructions, that your submission may need. Add a readme file if necessary.

6 Grading

Your assignment will be scored based on your performance on these metrics.

Grading criterion	**Points**
Data-structure description and implementation for storing graph	10
Implementing the three functions in part 1	25
Graph simulations and testing	25
Building graph from real data and testing	20
Running time and peak memory analysis	15
Follow specifications and policies for the course & assignment [1]	5
Total	**100**

[1]See Lecture 1 for an overview of the policies