COSC3320: Graph Algorithms

1 Introduction

You will write a program with graph algorithms. The program reads the input graph from a csv file. The program writes the output to a csv file as well, only with the output graph. The program displays numbers of anything computed on the entire graph (e.g. MST weight).

- Language: Pascal
- Graph: undirected by default, meaning that for some algorithms you need to include the "reverse" edge.
- Edges: some problems assume binary edges and some problem assume a real number (distance, weight).
- Data structure: you should use list of edges with tuples i, j, v because graphs will be sparse. You can use 2D arrays (dense matrix) to store the graph, but they are discouraged because they waste space. Clarify in your documentation your choice.
- Subscripts for input and vertex numbering: vertex numbering $1 \dots n$ (0 unacceptable).
- Algorithms: listed below. You will choose a few algorithms to program. You need to choose one when calling your program.
- Minimum requirement: you can assume the entire input graph can be stored in main memory.
- Extra credit: low space complexity in memory (small RAM space). develop external algorithms. Reading the input graph block by block, not the entire graph.

2 Input and output

Notice that for most algorithms the input graph is assumed to be undirected, meaning you need to have two edges per vertex pair in a list of edges. That is, you need to add the reverse edge.

Input example 1

The input is a regular csv file with three columns (e.g. input1.csv as shown). The first line of the input should be line header, "i,j,v". The 'i' column and the 'j' column are integers. Each row is one edge with the source vertex id 'i', destination vertex id 'j', and edge weight 'v'. The third column is integer 1 for all edges(assuming the input graph is an un-weighted graph.).

```
# File:    input1.csv
i,j,v
1,2,1
2,3,1
1,3,1
4,5,1
5,7,1
7,4,1
10,11,1
```

Output example

The program should write the output into a csv file. Any diagnostic, progress or global metrics on the grpah should be displayed on the screen. The first line should be line hearder "G,i,j,v". 'G' is the index of output graphs. 'i', 'j' and 'v' are integers. There should be no extra spaces and characters before or after the number. If the output has only one graph(e.g the transitive closure, minimum spanning tree), the 'G' column value is '1' for all output. If the output are paths, rename the 'G' column to 'P', which is the subscript of each path (starting from 1, paths starting from the same source vertex have the same index). The output should be ordered by G, i, j.

For the input given in the previous paragraph, the output is shown as follows when the graph algorithm is connected components. Column 'G' is the index of output graphs.

```
# File: output1.csv
G,i,j,v
1,1,2,1
1,1,3,1
1,2,3,1
2,4,5,1
2,5,7,1
2,7,4,1
3,10,11,1
```

The output is shown as follows when the graph algorithm is single source shortest paths with source vertex is 1. The column 'G' is the index of output graphs.

```
# File: output2.csv
G,i,j,v
1,1,2,1
1,1,3,2
```

The output is shown as follows when the graph algorithm is all-pairs-shortest paths. The column 'G' is the index of output graphs.

```
# File: output3.csv
G,i,j,v
1,1,2,1
1,1,3,2
1,2,3,1
1,4,5,1
1,4,7,2
1,4,4,3
1,5,7,1
1,5,4,2
1,5,5,3
1,7,4,1
1,7,5,2
1,7,7,3
1,10,11,1
```

3 Program input and output specification, main call

The main program should be called **graph** [**input output algorithm**], to be called from the command line. Ech run will have different input and algorithm.

Examples with run syntax at the OS prompt:

```
graph input1.csv output1.csv sssp

graph input1.csv output1.csv apsp

graph input1.csv output1.csv triangle
```

4 Graph algorithms

You need to choose 2-4 graph algorithms from the following list: 2 is the minimum acceptable. We will discuss later if any extra credit can be applied if you program correctly four algorithms. The first keyword is the **algorithm** choice for your program.

- explore: decide if a directed a graph is: connected, tree, cyclic, complete Y/N for each item.
- sssp: Single-Source-Shortest paths (the source vertex is a random integer in V), compute distance.
- apsp: All-Pairs-Shortest paths(one graph, so the column 'G' is 1). Compute the length (distance) of the shortest path.
- triangle: triangle enumeration (multiple graphs, each triangle is a subgraph), count triangles
- mst: minimum spanning tree(one graph for connected graph. If the input graph is disconnected, you compute one MST for each connected component. Compute the weight of the MST.
- tc: transitive closure (one graph), binary or weighted.
- cc: connected components (multiple graphs), binary.
- tsp: traveling salesman problem (one graph), weighted, compute the distance.
- clique: maximum clique enumeration (multiple graphs, each clique is a graph), binary.

5 Evaluation

- You need to decide a data structure to store the graph. The index must starting from 1. For example, if you use arrays to store the graph, the first element whose index is 0 should not be used(just skip it). For example, the array size should be 11 for an input graph that contains 1 to 10 nodes.
- For the minimum requirement, you can assume the input graph can fit in the main memory. You can use any data structure you want, a 1D array is allowed, but you must explain why.
- You must experimentally test time complexity, counting operations and compare with the theory.
- Correctness is the most important requirement. Please verify your algorithm with different graphs. The TA will test your code with small and large input graphs. The large graphs can have thousands of vertices. You will get 70/80 to pass only small graphs. Your program should not crash or produce exceptions.
- Catch errors at runtime by default (dynamically). You should identify the error in a specific manner when feasible instead of just displaying an error message.
- Extra credit: you can develop external algorithms. That means you do not load the entire graph into main memory, but process the input graph block by block. Indicate if you did it or not.

6 Programming requirements

- The program must work on our Linux server. Your program must compile/run from the command line. There must not be any dependency with your IDE.
- Input graphs:

  synthetic: create a program to generate graphs with random edges as follows using n as a parameter: $m = \sqrt{n}$ a sparse disconnected graph, $m = n - 1$ resulting in a tree, $m = 2n$ a graph with cycles, several complete subgraphs with $k = 3 \ldots n-1$ vertices s.t. all n vertices are covered by some edge, $m = n(n-1)/2$ edges. Store the generated graph in a csv file. Tip: two random numbers i, j in $1 \ldots n$ give us an edge; just avoid $i = j$. Call this program "generategraph".

  real: you are encouraged to download graph data sets, like the SNAP repository.
- Test our program well with small n and large n. Your program will be tested for correctness and robustness with small n and for time with large n.
- Submission: Phase 1 to test compile/run with small graphs with 1-2 algorithms. Phase 2: full program with 4 algorithms.
- Create a README file with instructions to compile. Makefile encouraged.