# COSC3320: graph algorithms

## 1 Introduction

You will write a C++ program that implements graph algorithms. You must choose two different algorithms from the list shown as follows ( you cannot choose both SSSP and APSP, or APSP and TC). The program reads the input from a csv file. The output should write into a csv file also.

**Minimum requirement**: you can assume the entire input graph can be stored in main memory.

**Extra credits**: develop external algorithms. Reading the input graph block by block, not the entire graph.

## 2 Input and output

### 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. 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, the 'G' column value is the index of paths(starting from 1, paths starting from the same source vertex have the same index). The output should be ordered by 'G'. If 'G' column is the same, it is ordered by 'i'.

For the input given in the previous paragraph, the output is shown as follows when the graph algorithm is connected graphs. The 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: output1.csv
G,i,j,v
1,1,2,1
1,1,3,1
```

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: output1.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.cpp**. The input argument is as "inPutFileName outPutFileName alg". The input file name can be changed when evaluating your program.

```
g++ graph.cpp -o graph
```

Run syntax at the OS prompt: Single Source Shortest path(the source vertex is 1):

```
./graph input1.csv output1.csv sssp
```

Other example of program call: All-pairs-shortest-path:

```
./graph input1.csv output1.csv apsp
```

triangle enumeration:

```
./graph input1.csv output1.csv tri
```

Minimum spanning tree:

```
./graph input1.csv output1.csv mst
```

Transitive closure:

```
./graph input1.csv output1.csv tc
```

connected components:

```
./graph input1.csv output1.csv cc
```

travelling salesman problem:

```
./graph input1.csv output1.csv travel
```

maximum cliques detection:

```
./graph input1.csv output1.csv clique
```

# 4 Graph algorithms

You need to choose two graph algorithm from the following list.

- Single-Source-Shortest paths(the source vertex is 1, one graph so the column 'G' is 1)
- All-Pairs-Shortest paths(one graph, so the column 'G' is 1)
- triangles enumeration(multiple graphs, each triangle is a graph)
- minimum spanning tree(one graph for connected graph. If the input graph is disconnected, you can have one MST for each connected component.
- transitive closure(one graph)
- connected components(multiple graphs)
- travelling salesman problem(one graph)
- maximum cliques detection(multiple graphs, each clique is a graph)

# 5 Evaluation

- You need to find 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 require, you can assume the input graph can fit in the main memory. You can use any data structure you want, an array is allowed.

- Extra credits: 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.
- 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.

# 6 Programming requirements

- Must choose two different graph algorithms. For example, you cannot choose both single-source-shortest paths and all-pairs-shortest-paths, or both all-pairs-shortest-paths and transitive closure.
- Must work on our Linux server
- Interpreter must be programmed in GNU C++. Using other C++ compilers is feasible, but discouraged since it is difficult to debug source code. TAs will not test your programs with other C++ compilers (please do not insist).
- You can use STL or any c++ libaries or you can develop your own C++ classes. You are required to disclose any code you downloaded/copied.
- Create a README file with instructions to compile. Makefile encouraged.
- Your program must compile/run from the command line. There must not be any dependency with your IDE.