UNIVERSITY OF MINNESOTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
4041: ALGORITHMS AND DATA STRUCTURES
SPRING 2021

PROGRAMMING ASSIGNMENT 2, PART 1 :
Assigned: 3/29/21 Due: 4/14/21 at 11:55pm

Make sure you apply your code to different inputs to test the results. Copying code from others or the internet constitutes cheating and the University policies will be followed. For each problem submit these four things:

1. The code you create.
2. A "readme.txt" file explaining how your code works (though you should comment as well). If you were not able to complete a problem, describe your approach here.
3. A "run.sh" which runs the code (just put the commands to start your program in here).
4. A "make.sh" file that will both compile (if necessary) and anything else you need to do before running the code with "run.sh".

The "readme.txt" (2), "run.sh" (3) and "make.sh" (4) should all not be in sub-folders, but rather directly uploaded (you may have your "run.sh" and "make.sh" use sub-folders if you wish). You can either directly upload multiple files to gradescope or upload a zip with your files, but make sure you do not zip a folder (as then the "sh" files will be in a sub-folder).

You cannot use any advanced libraries (basic mathematical or input/output ones are acceptable), but rather need to code all parts from scratch. We will use files to both send input into your program and to read the output of your program. You can assume the the file "input.txt" in the same folder as the "run.sh" is where the input will be that you need to read. In each part, your code should create a text file named "output.txt" with your solution. Please make sure you exactly follow the formats shown for each problem.

Please ensure your code is easily readable and well structured. If the code is obfuscated, has poorly named variables/functions, not well documented, etc., then points will be taken off. You may choose to code in any of these languages: C/C++, Java, Python or Rust. For each problem the point breakdown will be roughly: 70% correct output, 10% documentation (including readme.txt and .sh file sufficient), and 20% coding style. The code will be tested on gradescope's machines, which are linux based (and very similar to the linux machines in Keller).

Problem 1. (20 points)
Implement a hash table (dictionary/HashMap) using only arrays and linked lists (and a class/struct if you want). You may choose any (reasonable) hashing function. You may assume the keys will be a string (name) and the value stored in the table will be a number (grade). Given a sequence of push/inserts then get/queries, so what values.

Sample input.txt (command then arguments (2 for put, 1 for get)):

```
put bob 99
get sally
put sally 100
get sally
```

```
put sally 72
get sally
put jack 100
put alex 88
put joe 5
put ajax 78
get bob
```

Corresponding sample output.txt file:

```
none
100
72
99
```

Problem 2. (20 points)
In this problem the input file will have classes listed with their prerequisites. Your output.txt should list a valid order for taking all of these classes while meeting all the prerequisites.

You may assume there are no circular prerequisites (i.e. class A requires class B. Class B requires class C. Class C requires class A). You may assume all classes that are prerequisites of another are listed. The courses can be any string (not necessarily numbers)

Sample input.txt (course identifier before ":". Prerequisites after ":" with spaces between):
1001:
1133:
2011:
1933:1133
4041:1933 2011

Sample output.txt (valid class ordering with spaces between (no space after final class)):

```
1133 2011 1933 4041 1001
```

Problem 3. (20 points)
Implement Dijkstra's algorithm to find the shortest path to every other vertex. The input text file will contain the graph represented as an adjacency matrix. Values of 2 million will represent "infinity" edge weight (when there is no edge between the verticies). You can assume all shortest paths will be less than 2 million.

The first line in the input file is the source vertex, one space, then the destination vertex (with the first row of the adjacency matrix being vertex "0", the second row being vertex "1", and so on). Afterwards is the adjacency matrix (which will be square). Your output.txt should contain the shortest path length from this source to destination vertex followed by a ":". After the ":" should be the actual list of vertexes you should travel through for this shortest path.

Sample input.txt (fire line always source and destination vertex, afterwards is adjacency matrix):
0 1

```
0 2000000 4
```

```
2 0 7
2000000 3 0
```

Sample output.txt (7 is the shortest path length (before ":"). 0 2 1 is the actual path (after ":")):
```
7: 0 2 1
```

Problem 4. (20 points)
Implement Johnson's algorithm to fin all pairs shortest paths. The input text file will contain the graph represented as an adjacency matrix. Values of 2 millino will represent "infinity" edge weights (i.. when there is no edge between vertexes). You can assume all pairs of shortest paths will be less than 2 million.

Your output.txt should contain a matrix of all pairs shortest paths (just the distances). If there is a negative cycle present, output.txt should contain just the words (without quotes): "Negative cycle". The matrix for shortest paths should have spaces between the numbers and a single row on one line (similar to the input file). The vertex order must also be the same as the input text file, so the first row in the input text file must correspond to the shortest paths from the first vertex in output.txt.

Sample input.txt #1 (contains adjacency matrix):
```
0 2000000 4
2 0 7
2000000 3 0
```

Sample #1 output.txt (no trailing spaces after last number):
```
0 7 4
2 0 6
5 3 0
```

Sample input.txt #2 (contains adjacency matrix):
```
0 2000000 -4
-1 0 7
2000000 -2 0
```

Sample #2 output.txt (just this line and no other words):
```
Negative cycle
```

Problem 5. (20 points)
Assume you ran your problem 4 code on an undirected graph to find all pairs shortest path, but afterwards new vertexes are added to the graph and you want to compute the new all pairs shortest paths. In other words, find an efficient way to find all pairs shortest paths when new vertexes and edges are added for an **undirected** graph. Here "efficient" means that the runtime should be smaller than rerunning your problem 4 on the new graph from scratch. Your runtime should be approximately runtime(# old vertexes) + runtime(# added vertexes) = runtime(# total vertexes), where "old vertexes" and "total vertexes" could be computed directly from problem 4's algorithm.

The first thing in the input file will be the adjacency matrix (same setup as problem 4, except you can assume it will correspond to an undirected graph). After this will the list of new vertexes in the format of their corresponding rows in the adjacency matrix (see below for an example).

Your output.txt should contain two all pairs shortest paths: one using only the original matrix (same as the output would be for problem 4) and one with all the vertexes. (Again, you should not re-run problem 4 from scratch to get the all pairs shortest paths with all vertexes.) If there are no extra vertexes, just simply output the same as problem 4 with a single matrix. For each "all pairs shortest paths", if a negative cycle is present simply output (without quotes): "Negative cycle". If this happens in both the original and new graph, this output should appear twice.

Sample input.txt #1 (contains adjacency matrix with no "new vertexes"):

```
0 2 1
2 0 4
1 4 0
```

Sample #1 output.txt (no trailing spaces after last number):

```
0 2 1
2 0 3
1 3 0
```

Sample input.txt #2 (contains adjacency matrix, with two "new" vertexes):

```
0 2 1
2 0 4
1 4 0
2000000 4 6 0 1
2 1 3 1 0
```

Sample #2 output.txt (two matrices, one 3x3 and one 5x5):

```
0 2 1
2 0 3
1 3 0
0 2 1 3 2
2 0 3 2 1
1 3 0 4 3
3 2 4 0 1
2 1 3 1 0
```

Sample input file #3 (contains adjacency matrix):

```
0 2 1
2 0 4
1 4 0
```

```
2000000 4 6 0 1
-2 1 3 1 0
```

Sample #3 output.txt:

```
0 2 1
2 0 3
1 3 0
Negative cycle
```

Sample input.txt #4 (contains adjacency matrix):

```
0 2 1
2 0 -4
1 -4 0
2000000 4 6 0 1
-2 1 3 1 0
```

Sample #4 output.txt:

```
Negative cycle
Negative cycle
```

Sample input.txt #5 (contains adjacency matrix):

```
0 2 1
2 0 -4
1 -4 0
```

Sample #5 output.txt (just this line and no other words):

```
Negative cycle
```