National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2019

Handout #
November 26, 2019
Iris Hui-Ru Jiang

Programming Assignment #3 (due on-line 1pm, December 29, 2019)

Submission URL & Online Resources:
https://cool.ntu.edu.tw/courses/469/assignments/4391 (Prof. James Chien-Mo Li's class)
https://cool.ntu.edu.tw/courses/527/assignments/4392 (Prof. Iris Hui-Ru Jiang's class)
https://cool.ntu.edu.tw/courses/543/assignments/4390 (Prof. Yao-Wen Chang's class)

Problem: Cycle Breaking

Cycle breaking serves as a fundamental technique for many applications, e.g., resolving resource deadlock or simplifying a problem instance. In graph theory, a cycle is a path where the first and the last vertices are the same. A graph without cycles is called an acyclic graph. Given a graph $G = (V, E)$ which may contain cycles, we want to remove some edges to make the graph acyclic with minimum total cost (weight). Such a problem is so-called the *cycle breaking* or *cycle removal* problem. For example, Figure 1 (a) is a weighted undirected graph G_u with cycles. We can remove e_{01} and e_{34} to make it acyclic with total cost $= 3 + 5 = 8$. For the weighted directed graph G_d in Figure 1 (b), there is only one cycle C_{143}. So we only need to remove e_{43} with total cost $= 5$. Cycle breaking for an (unweighted or weighted) undirected graph can be solved in polynomial time. The problem for a weighted directed graph, however, is also known as a *minimum feedback arc set* problem, which is a NP-hard problem. In this assignment, test cases contain three types of graph instances: 1) unweighted undirected graph, 2) weighted undirected graph, and 3) weighted directed graph.

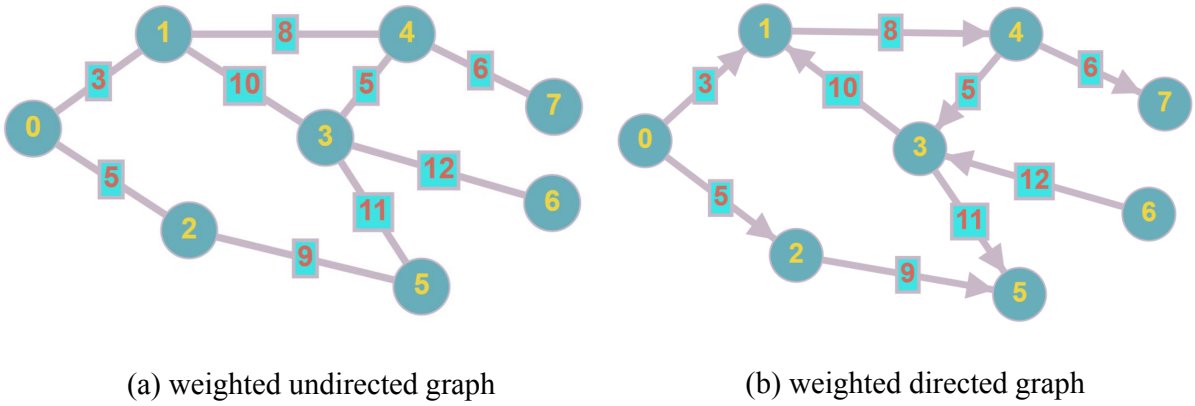

Figure 1: Graph with cycles.

Input

The input will be a graph with only one connected component which may contain cycles or not. The first line of the input is a character "u" or a character "d", which indicates the input graph is an undirected graph or a directed graph, respectively. The second line is an integer n, denoting the total number of vertices. The indices of these nodes will be continuous from 0 to $n - 1$. The third line is an integer m, denoting the total number of edges. In the following m lines, each contains three integers i, j and w, denoting an edge from vertex i to vertex j with weight w, $-100 \leq w \leq 100$. For test cases of unweighted undirected graph, the weight of all edges will equal to 1. Please note that the order of vertex i and j implies the direction of the edge in a directed graph, while it does not matter in an undirected graph. A single "0" (zero) in the input line signifies the end of input.

Output

The output file should report the total weight of removed edges to make the input graph acyclic, followed by a list of these removed edges and their weights. The output edges can be in arbitrary order. If the input graph has no cycles, you should output a line with single "0" (zero) in your output file.

Here are some input/output examples:

Sample Input 1	Sample Output 1	Sample Input 2	Sample Output 2
u 8 9 0 1 3 0 2 5 1 3 10 1 4 8 2 5 9 3 4 5 3 5 11 3 6 12 4 7 6 0	8 0 1 3 3 4 5	d 8 9 0 1 3 0 2 5 1 4 8 2 5 9 3 1 10 3 5 11 4 3 5 4 7 6 6 3 12 0	5 4 3 5

Command-line Parameter:
The executable binary must be named as **'cb"** and use the following command format.

```
./cb <input_file_name> <output_file_name>
```

For example, if you would like to run your binary for the input file `1.in` and generate a solution named `1.out`, the command is as follows:

```
./cb 1.in 1.out
```

Required Files:
You need to create a directory named <**student_id**>**_pa3/** (*e.g.* b06901000_pa3/) which must contain the following materials:

- A directory named **src/** containing your source codes (*e.g.* `cycleBreaking.cpp`): only `*.h`, `*.hpp`, `*.c`, `*.cpp` are allowed in src/, and no directories are allowed in src/;
- An executable binary named **cb**;
- A makefile named **makefile or Makefile** that produces an executable binary from your source codes by simply typing "make": the binary should be generated under the directory <student_id>_pa3/;
- A text readme file named **readme.txt** describing how to compile and run your program.

Then please use the following command to compress your directory into a *.tgz* file:

```
tar zcvf <filename>.tgz <your directory>
```

The submission file should be named as <**student_id**>**_pa3.tgz** (*e.g.* b06901000_pa3.tgz). For example, if your student ID is *b06901000*, then you should use the command below.

```
tar zcvf b06901000_pa3.tgz b06901000_pa3/
```

Please submit your *.tgz* file to the NTU COOL system before **1pm, December 29, 2019 (Sunday)**. To make sure that your submission satisfies all the requirements, we provide a script `checkSubmitPA3.sh` to check your *.tgz* file by the command below:

```
bash checkSubmitPA3.sh <your submission>
```

For example,

```
bash checkSubmitPA3.sh b06901000_pa3.tgz
```

Language/Platform:

1. Language: C or C++.
2. Platform: Linux.

Evaluation:
An individual score per test case is determined by the correctness of the output result as well as the file format for undirected graph instances. The score is determined by not only correctness but also quality (total weight) for directed graph instances. The runtime limit for each case is 60 seconds. For fair evaluation, please apply the **-O3** optimization for the compilation. Three input test cases are provided and more hidden test cases will be used for the final test.

References:
Breadth-first search, depth-first search, minimum spanning trees.

For any questions, please email Shang-Chien Lin at r07943106@ntu.edu.tw or Yi-Ting Lin at r07943102@ntu.edu.tw. Thank you so much for your cooperation. Have fun with this programming assignment!