National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2022

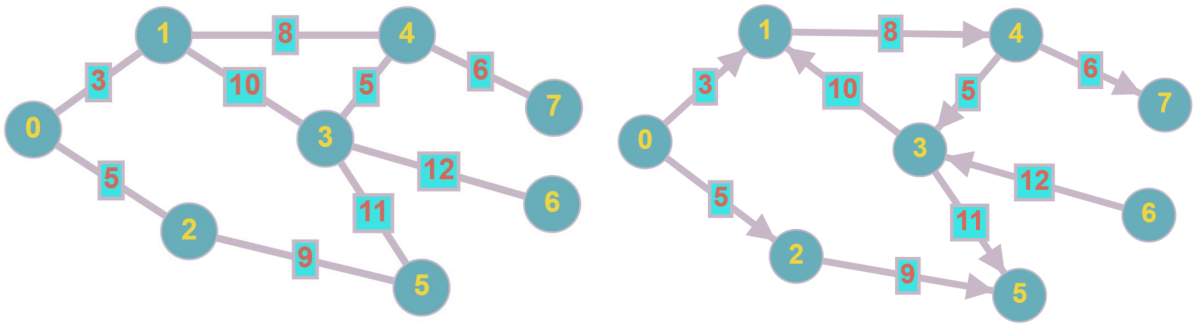
Handout #19
October 18, 2023
Chih-Hung Liu

Programming Assignment #3 (due on-line.)
Deadline: 1pm, December 27, 2023.

Submission URL & Online Resources:
https://cool.ntu.edu.tw/courses/32190/assignments/197552 (Prof. Yao-Wen Chang's class)
https://cool.ntu.edu.tw/courses/29152/assignments/197551 (Prof. James Chien-Mo Li's class)
https://cool.ntu.edu.tw/courses/30477/assignments/197550 (Prof. Chih-Hung Liu)

Problem: Cycle Breaking

Given a simple, weighted graph $G = (V, E, w)$, the *cycle breaking* problem is to make G *acyclic* by removing some edges in E such that the total weight of the removed edges is minimized. Recall that, in graph theory, a cycle is a path where the first and the last vertices are the same, and a graph without cycles is called an acyclic graph. 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 $= 5$.



(a) weighted undirected graph (b) weighted directed graph

Figure 1: Graphs with cycles.

In this assignment, test cases contain three types of graph instances: 1) unweighted undirected graph, 2) weighted undirected graph, and 3) weighted directed graph.

Additional information. The cycle breaking problem for an (unweighted or weighted) undirected graph can be optimally solved in polynomial time. However, the cycle breaking problem for a weighted, directed graph as known as the *minimum feedback arc set* problem, however, is a NP-hard problem. The so-far best polynomial-time approximation algorithm achieves an $O((\log n)(\log\log n))$ approximation factor, i.e., the value of the computed solution is $O((\log n)(\log\log n))$ times that of the optimal solution. It has also been proved that unless NP = P, no polynomial time can attain an approximation factor less than 1.3606. If the unique game conjecture is correct, there exists no constant-factor polynomial-time approximation algorithm.

Input

The input will be a simple 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 each edge will be 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.

Scope of n, m

- Undirected graph: $n \leq 10,000$; $m \leq 20,000,000$
- Directed graph: $n \leq 5,000$; $m \leq 10,000$

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, noted that the graph should still be connected after edges removal. 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
```

Compilation:
We expect that your code can be compiled and run as follows. Type the following commands under <**student_id**>**_pa3/** directory,

```
make
./bin/cb inputs/<input_file_name> outputs/<output_file_name>
```

Required Files:
You need to create a directory named <**student_id**>_**pa3/** (*e.g.* b10901000_pa3/) **(the student ID should start with a lowercase letter)** which must contain the following documents:

- A directory named **src/** containing your source codes (*e.g.* `cyclebreaker.cpp`): only `*.h`, `*.hpp`, `*.c`, `*.cpp` are allowed in src/, and no directories are allowed in src/;
- A directory named **bin/** containing your executable binary named **cb**;
- A directory named **doc/** containing your report;
- A makefile named **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/bin/;
- A text readme file named **README** describing how to compile and run your program.
- A report named **report.pdf** on the data structures used in your program and your findings in this programming assignment.

We will use our own test cases, so you do NOT need to submit the input files. Nevertheless, you should have at least the following items in your *.tgz file.

```
src/<all your source code>
bin/cb
doc/report.pdf
makefile
README
```

Submission Requirements:

1. Your program must be compilable and executable on EDA union servers.
2. The runtime limit for each test case is **1 minute**.
3. 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.* b10901000_pa3.tgz). For example, if your student ID is *b10901000*, then you should use the command below.

   ```
   tar zcvf b10901000_pa3.tgz b10901000_pa3/
   ```

4. Please submit your *.tgz* file to the NTU COOL system before **1pm, December 27, 2023 (Wednesday)**.
5. You are required to run the `checkSubmitPA3.sh` script to check if your *.tgz* submission file is correct. Suppose you are in the same level as the PA3 directory,

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

   For example,

   ```
   bash checkSubmitPA3.sh b10901000_pa3.tgz
   ```

Language/Platform:

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

Evaluation:
For undirected graph instances, an individual score per test case is determined by the correctness of the output result as well as the file format. For directed graph instances, the score is determined by not only correctness but also your rank of quality (total weight) compared with all submitted PAs. For fair evaluation, please apply the **-O3** optimization for the compilation. Seven 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 Yuan-Hsiang Lu at b07901030@ntu.edu.tw. Thank you so much for your cooperation. Have fun with this programming assignment!