National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2021

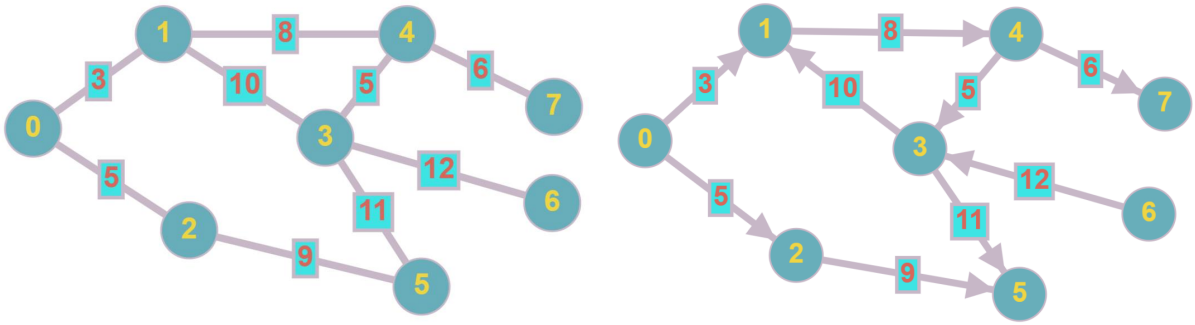
Handout #17
December 6, 2021
Iris Hui-Ru Jiang

Programming Assignment #3 (due on-line 1pm, January 16, 2022)

Submission URL & Online Resources:
https://cool.ntu.edu.tw/courses/8266/assignments/59010 (Prof. James Chien-Mo Li's class)
https://cool.ntu.edu.tw/courses/9152/assignments/59007 (Prof. Iris Hui-Ru Jiang'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 simple 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 optimally be solved in polynomial time. The problem for a weighted directed graph, also known as a *minimum feedback arc set* problem, however, 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.



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

Figure 1: Graphs with cycles.

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. For undirected graph instances, $1 \leq n \leq 10000$ and $1 \leq m \leq 50000000$. For directed graph instances, $1 \leq n \leq 5000$ and $1 \leq m \leq 10000$.

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 graph must remain connected (weakly-connected component for directed graph instances) after the edges are removed. 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.* b08901000_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. **Do not use the existing library (e.g. Boost Graph Library, Boost Disjoint Sets) in your implementation.** But, it is allowable to use sorting and heap in the existing C++ library.
3. The runtime limit for each test case is **1 minute**.
4. 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.* b08901000_pa3.tgz). For example, if your student ID is *b08901000*, then you should use the command below.

   ```
   tar zcvf b08901000_pa3.tgz b08901000_pa3/
   ```

5. Please submit your *.tgz* file to the NTU COOL system before **1pm, Janurary 16, 2022 (Sunday)**.
6. 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 b08901000_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. Six input test cases are provided and more hidden test cases will be used for the final test.

Grading Policy:

1. Report 20%
2. Correctness
    - Unweighted undirected graph 20%
    - Weighted undirected graph 30%
    - Weighted directed graph 6%
3. Performance
    - Weighted directed graph 24%

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

For any questions, please email Yi-Ting Hsieh at r09943097@ntu.edu.tw. Thank you so much for your cooperation. Have fun with this programming assignment!