Project **Winter 2021/22**
Algorithm Engineering for hard problems **TU Berlin**
Nichterlein/Niedermeier **October 20, 2021**

1st Exercise Sheet

Weighted Cluster Editing I

Solver: November 01 **Handout:** November 02 **Presentation:** November 03

The task is to implement an exact algorithm to solve the Weighted Cluster Editing problem introduced in the lecture and to present the findings in a handout and a presentation. The deadlines for submitting the algorithm source code as well as the handout are at **12 noon** each, on the date given above.

Weighted Cluster Editing
Input: An undirected graph $G = (V, E)$ and a weight function $\omega\colon E \to \mathbb{Z}$.
Output: Find a set $S \subseteq V \times V$ of edges to add or delete such that $G' = (V, E \triangle S)$ is a cluster graph. The set S must be of minimum weight, i.e., $\sum_{e \in S} \omega(e)$ is minimized.

The goal of the 1st exercise sheet is to implement the simple search tree algorithm from the lecture.

All implementations **must** compile/run on one of our machines:

`X.akt.tu-berlin.de` with `X` $\in \{$`abc01`, `abc03`$\}$.

Java, C++, Python, etc. are installed. Please ask in the discussion forum in the ISIS course if you want further software installed.

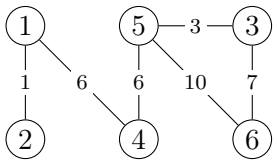
Input and output format. We use the following format for the input.

- Whitespaces at the beginning and end of every line are ignored.
- Text after # until the end of the line is ignored (treated as comment).
- Empty lines are ignored.
- The first line is the number $|V|$ of vertices in the input graph.
- The next $\binom{|V|}{2}$ lines define an edge or a non-edge by stating a start and an end vertex (indices are in the range $\{1, \ldots, |V|\}$) and its weight, all separated by a whitespace. If the weight is positive, then there is an edge between the two endpoints. If the weight is negative or zero[1], then there is no edge between the two endpoints.

[1] Only larger test instances (not expected to be solved in the first two exercises) contain weight zero. Strategies for dealing with weight zero will be discussed in the second lecture.

Example. (Weights are visualized only for edges.)

Input graph:

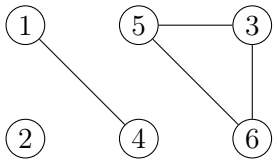


Input file:

```
6
1 2 1
1 3 -5
1 4 6
1 5 -1
1 6 -4
2 3 -4
2 4 -2
2 5 -10
2 6 -8
3 4 -5
3 5 3
3 6 7
4 5 6
4 6 -4
5 6 10
```

Your program should read the input file from `stdin`. It should be in the format described above; in case it is not, please notify us. The set S should be printed into `stdout`. Please output for each $\{u, v\} \in S$ a line containing u and v, separated by a white space. The order of the output does not matter. For the above example this should be:

Cluster graph:



Output:

```
1 2
4 5
```

Verification. We provide values of optimal solutions for most of the test instances in a separate `.solution` file.

Test data and benchmarks. The test instances, together with the solution files and a benchmark script are available for download:

https://fpt.akt.tu-berlin.de/alg-eng-data/wce-students.zip

There are three type of instances: `random`, `actionseq`, and `realworld`. The running time will be evaluated on those as well as some undisclosed instances.

Within the zip file you will find a bash script `benchmark-wce.sh`. It allows easy execution of your program on all the test instances under the following requirements: The computation of an instance is aborted if the program did not find a solution after

5 minutes. After *10 unsolved instances*, the script skips the remaining instances in the current subfolder.

To test your program, modify the following line so that it contains the command needed to run your program on a console:

```
PROGRAMM_NAME = "./path/to/your/solver"
```

For Java (assuming you created a jar called `mysolver.jar`), the string should be `"java -jar mysolver.jar"`. For Python, it should be something like `"python3 mysolver.py"`, and for C++ it should be something like `"./mysolver"`. *Hints.* If your program outputs the line "`#recursive steps:` x" (where x is the number of recursive calls counted by your program), then this value will be displayed when using the benchmark script. You might want to use this information in your presentation or handout.

```
1-random/000002_000000000002.dimacs 0.05 0 0 1 correct 0
1-random/000002_000000000006.dimacs 0.33 1 0 1 correct 0
...
```

The meaning of the strings in the columns are as follows:

Column 1. The filename of the tested instance.

Column 2. The time needed by your program.

Column 3. The solution size determined by your program (= number of lines not starting with #).

Column 4. The number of recursive steps (see hint above).

Column 5. Whether your program finished (value 1) or was terminated (value 0) due to hitting the time limit.

Column 6. Was the output of your program correct?

Your program will be terminated first with SIGTERM 2 (your program is asked somewhat nicely to shut down (it still can print something) and (as a fallback) 1 second later with SIGTERM 9 (hard kill, your program cannot do anything anymore). Knowing this might come in handy in future exercises.

Handout and presentation. Please submit a handout of your evaluation and present your findings. Your handout should give a short description of what you implemented as well as comparisons of the theoretical and practial running times (depending on the cost and the number of vertices). In later exercises it should also give a comparison to the results of the last exercise.

Find a LaTeX template for the handout in the ISIS course. There should be one page of double-column written text (including references) as well as some additional pages containing some nice figures (find templates for these in ISIS as well).

Your presentation should last at most 10 minutes and answer the following questions:

- How did you implement? What data structures and (possibly) other clever ideas did you use? In the first sheet also: Which programming language and packages did you use?
- How does the running time depend on the number of vertices of the input graph?
- How does the running time depend on the minimum cost to obtain a cluster graph?
- Is the theoretical running time bound reflected in the empirical running time? If not, what could be possible reasons?
- How do the answers of the above questions depend on the type of the instances, that is, are there differences between the synthetic and the real world instances? If yes, what could be possible reasons?

We recommend to use graphical representations of the data (e.g., diagrams) to answer these questions. (There is an example presentation with source code in the ISIS course and an exampe for making diagrams in LaTeX.) Please keep in mind that every member of your team should be able to answer the above questions, and every member of the team should have a fair share of presentation time during the five presentations.