Professor Dr. Jens Vygen
Matthias Kaul

Programming Exercise 2

Exercise P.2. *Task:* Implement approximation algorithms for Submodular Function Maximization for the case of Exercise 11.3. You should implement both the algorithm presented in exercise 11.2, as well as the randomized 2-approximation algorithm presented in the lecture.

Usage: Your program should be called as follows:

```
program input_file
```

Input: The argument `input_file` is mandatory, i.e. your program should exit with an error message if it is not present. The file `input_file` encodes the input instance as follows: All lines beginning with a `c` are comments. Ignoring any comment lines the first line has the format

`p` n m

where $n = |F|$ and $m = |D|$. We implicitly identify the facilities F with $\{1, \dots, n\}$ and the clients D with $\{1, \dots, m\}$. The next line has the format

`f` c

representing the facility opening costs c.

The following $n \cdot m$ lines have the format

`d` i j x

representing that $p(i, j) = x$.

Output: Your program should compute a set $X \subseteq F$ approximately maximizing

$$f(X) := \sum_{d \in D} \max\{0, \max_{x \in X} p(d, x)\} - c \cdot |X|.$$

Implement *both* the algorithm from Theorem 14.26 in the book and the derandomization from Exercise 11.2 to solve this problem. Your program should ouput the two solutions X_1, X_2 to the standard output on two separate lines, each containing the ids of the facilities in X_i separated by spaces. It should output in the first line the set computed by the algorithm from Theorem 14.26, and on the second line the one computed by the algorithm from exercise 11.2.

Programming conditions: Your program should be written in C, C++, or Python 3, although the use of C++ is strongly encouraged. By default, your program will be compiled using g++ 13.2.0 using C++20. Different compilers or compiler versions are available upon request. Your program will be compiled using `-pedantic -Wall -Wextra -Werror`, i.e., all warnings are enabled and each remaining warning will lead to compilation failure. Program evaluation will be performed on Linux. The standard library can be used as you wish. No other libraries are allowed.

Submission Format: Your submission should consist of a single archive file in the .zip, .tar.gz or .tar.bz2 format, which contains all contents of your top level directory (but not the directory itself). For easier testing, your submission must contain a bash script `compile.sh` in its top level directory, which builds the executable (e.g. by directly calling the compiler or by executing some make command) when called without any arguments.

Code evaluation: Running time in practice will also be evaluated, as well as the elegance, cleanness and organization of your code. Make sure to add good documentation and give the variables, functions and types meaningful names that make their role clear. Break your complicated functions into small simple ones, break your program into a few units etc. Of course, your program should not trigger undefined behavior. In particular, your program should be `valgrind`-clean, i.e. it should not leak memory and should not perform invalid operations on memory.

Help: The website for the exercise class contains a set of test instances for testing your code.

(14 points)

Exercise P.3. Discuss the results of your implementation. Which of the two algorithms performs better in practice for the given instances? How much does this depend on the order of the facilities, or on the random choices? Submit the discussion of the results alongside the program itself as a PDF document.

(6 points)

Deadline: January 13th AoE, submissions should be sent by e-mail to mkaul@uni-bonn.de. The websites for lecture and exercises can be found at:

`http://www.or.uni-bonn.de/lectures/ws24/co_exercises_ws.html`