Algoritmi e Strutture Dati

The office (`the-office`)

Problem statement

Original slides at: asdlab.disi.unitn.it/asd25/prog2.pdf

The Problem

In one of the branches of the great **Dunder Malloc**, deep in the Povsylvania of Trentino, there is an air of significant change. Among colleagues, who never lack a passion for gossip, news of a new director has been spreading for weeks. In a world dominated by the best memory allocators, the company has always been on the brink of failure. The employees of Dunder Malloc, overcome by discouragement, work in an increasingly listless and inefficient manner, wasting precious time and skills.
Finally today is the big day: the new director **Albert Scott** enters the company, ready and convinced that he can change things. For Dr. A. Scott, the watchword is **optimization**. *"Can we do better than this?"* is his very famous motto. The new director, a lover of any gossip that can keep his "hard work" on hold, calls his collaborators into his office one by one for an interview. He asks about their **skills**, their **hourly availability**, and their **cohesion** with each of their colleagues.

Work cohesion. At Dunder Malloc, it is well known that not everyone gets along. For each pair of people p and q, Director Scott identifies 2 values w_{pq} and w_{qp}: the first represents how well p feels working with q (*p's cohesion with q*), and the second represents the vice versa.

The projects. Albert Scott's difficult role is to assign the right people to the projects that the company wants to carry out. Each project requires a certain number of hours to be completed and a set of skills.
Albert Scott has big plans, but the collaborators are hostile. Fearing they will be scattered across too many projects without being able to focus on any, they make several requests and win a small concession: it is agreed that **no one can work on more than K projects at the same time**.

Your task. Dr. Scott, suspecting a *Not-Exactly* easy resolution, admits he has no idea how to solve the problem of assigning people to projects: could you help him?
Your task is to help the director build a valid assignment of people to projects. The objectives, in order of priority, are:

- **cover the skills required by the projects**: a project is considered satisfied only if, among the people assigned to it, there is at least one person for each required skill;
- **minimize uncovered time**: only for projects with covered skills, we measure how much of the hourly demand remains unassigned; if a project does not cover all required skills, its entire demand is considered uncovered;
- **maximize team cohesion**: among solutions with similar coverage, teams with high overall cohesion are preferable.

Input

A file with $1 + 3 \cdot P + 2 \cdot J$ lines.

- The first line contains 4 integers: P (`int`) the number of people, J (`int`) the number of projects, S (`int`) the number of existing skills, and K (`int`) the maximum number of projects a person can work on.
- The following $3 \cdot P$ lines (from 2 to $1 + 3 \cdot P$) describe the people. For each person $p = 0, ...P - 1$:
  - the $(2 + 3 \cdot p)$-th line contains 2 integers, cap_p (`int` $\in [1, 40]$), the number of hours per week that person p can work, and σ_p (`int`), their number of skills;

  - the $(2 + 3 \cdot p + 1)$-th line contains σ_p integers, $\{s_i\}_{i=0}^{\sigma_p - 1}$ (`int`), the identifiers of the skills possessed by person p;
  - the $(2 + 3 \cdot p + 2)$-th line contains P integer values, $\{w_{pq}\}_{q=0}^{P-1}$ (`int`), where the q-th value w_{pq} is person p's work cohesion with person q.
- The following $2 \cdot J$ lines (from $2 + 3 \cdot P$ to $1 + 3 \cdot P + 2 \cdot J$) describe the projects. For each project $j = 0, ...J - 1$:
  - the $(2 + 3 \cdot P + 2 \cdot j)$-th line contains two integers dem_j (`int`) and $\hat{\sigma}_j$ (`int`), respectively the number of hours and the number of skills required for project j;
  - the $(2 + 3 \cdot P + 2 \cdot j + 1)$-th line contains $\hat{\sigma}_j$ integer values, $\{\hat{s}_i\}_{i=0}^{\hat{\sigma}-1}$ (`int`), the identifiers of the skills required by project j.

Output

A file containing your proposed solutions, where each proposal consists of $(2 + 3 \cdot J)$ lines. Each proposal:

- in the first line contains the `score` value (`int`) calculated as described in the next section;
- following this, for each project:
  - the first line contains the project index j, followed by the number of people num_j assigned to the project;
  - the next line contains the list of people assigned to the project;
  - the next line contains the respective working time (in weekly hours) of each person on the project, **which must always be greater than 0**;
- the proposal must end with a final line: `***`.

The score

The value you have to compute and minimize is:

$$\texttt{score} = \alpha \cdot \texttt{total\_unmet} - \beta \cdot \texttt{total\_cohes}. \tag{1}$$

Weights: $\alpha = 7$, $\beta = 3$.

Uncovered time:

$$\texttt{total\_unmet} = \sum_{j=0}^{J-1} u_j, \qquad u_j = \begin{cases} dem_j & \text{if the skills of project } j \\ \text{uncovered time} & \text{are not all covered} \\ \text{in project } j & \text{otherwise} \end{cases}$$

Total cohesion:

$$\texttt{total\_cohes} = \left\lfloor \frac{1}{2P} \sum_{j=0}^{J-1} \sum_{p,q=0}^{P-1} y_{j,p} y_{j,q} w^*_{j,pq} \right\rfloor, \quad w^*_{j,pq} = \begin{cases} -|w_{pq}| & \text{if the skills of project } j \\ & \text{are not all covered} \\ w_{pq} & \text{otherwise} \end{cases}$$

$y_{j,p}$ is 1 if p is assigned to project j, and 0 otherwise. It is always true that $w_{pp} = 0$.

Examples

The examples refer to a single input case (Figure 1) and propose several solutions (valid or incorrect).

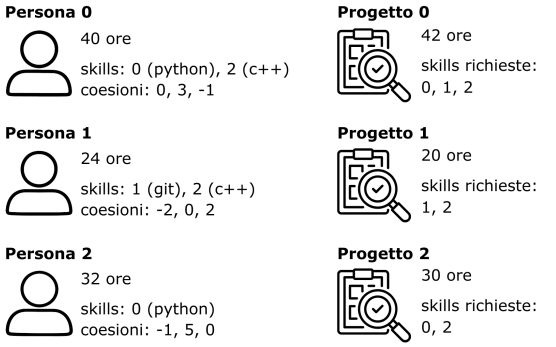

Figura 1: Example with $P = 3, J = 3, S = 3, K = 2$.

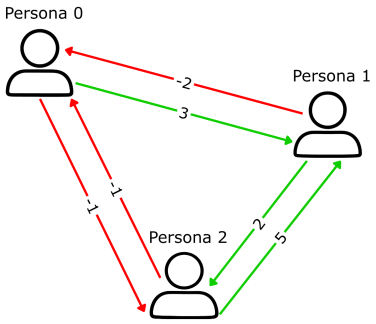

Figura 2: Cohesions between the people in Figure 1 visualized with a directed graph

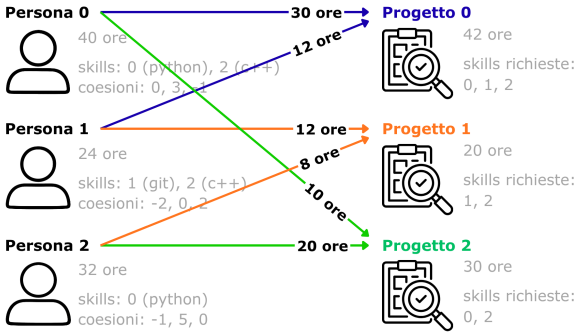

Figura 3: Valid solution for the input in Figure 1 (`score` = -3)

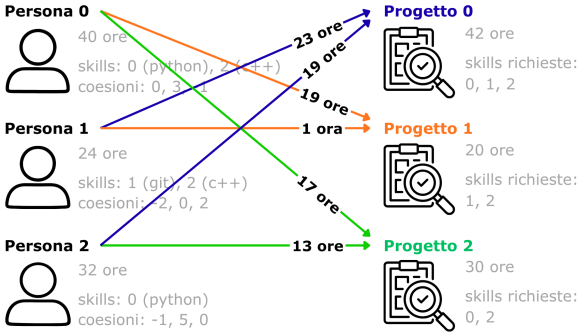

Figura 4: Valid solution for the input in Figure 1 (`score` = -3)

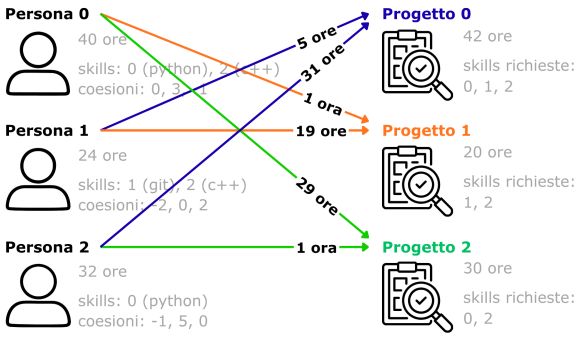

Figura 5: Valid solution, but worse than the previous ones (not all requested hours are covered), for the input in Figure 1 (`score` = 39)

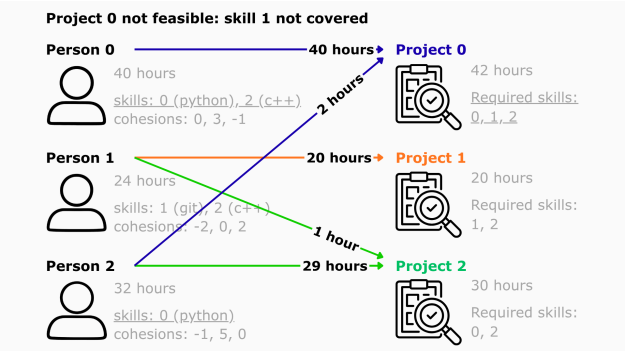

Figura 6: Valid solution, but worse than the previous ones (not all skills of a project are covered), for the input in Figure 1 (`score` = 294)

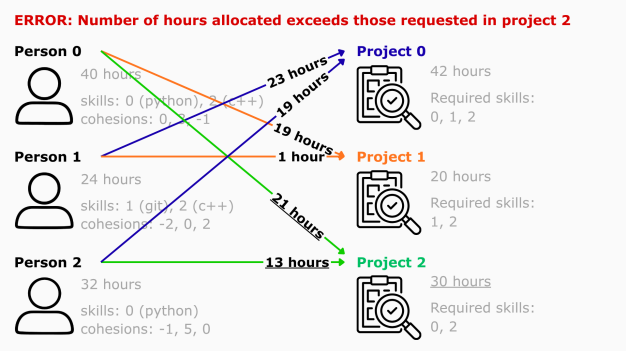

Figura 7: Incorrect proposed solution for the input in Figure 1. Project 2: allotted time (34) great than the required time (30).

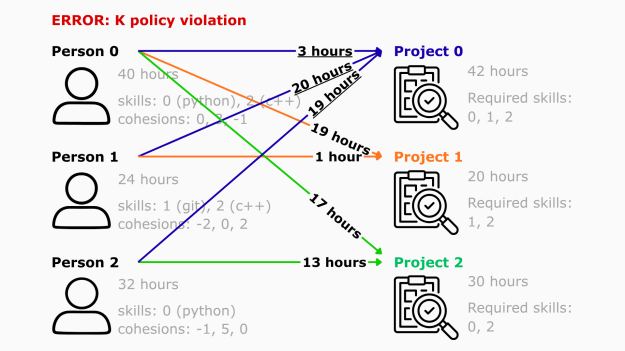

Figura 8: Incorrect proposed solution for the input in Figure 1, K-policy violation. Persona 0 assigned to 3 project (over the limit of 2)

Assumptions and test cases

Input assumptions

Parameters:

- $1 \leq P \leq 5\,000$
- $1 \leq J \leq 5\,000$
- $1 \leq S \leq 500$
- $1 \leq K \leq 40$
- $-10 \leq w_{pq} \leq 10$ for each pair of people p and q
- $w_{pp} = 0$ for each person p
- $1 \leq cap_p \leq 40$ for each person p
- $1 \leq dem_j \leq 5\,000$ for each project j

People are identified by values $0...P-1$, projects by values $0...J-1$, and skills by values $0...S-1$. Each person has at least one skill. Each project requires at least one skill.

Output assumptions

- If a project is assigned more time than it requires, it is considered an error
- If a person is assigned with 0 hours to a project, it is considered an error

Test cases

There are 20 test cases in total:

- In 6 out of 20 cases, $K = J$, $w_{pq} = 0$ for every p and q, all people possess the *same* skill and all projects require *only* that skill;
- In 9 out of 20 cases, $w_{pq} = 0$ for every p and q;
- In the remaining cases there are no particular limitations.

Time and memory limits

- Maximum time limit: 3 seconds (*soft*): 3.4 seconds (*hard*).
- Maximum memory: 48 MB.
- There is a limit of 50 submissions per group.

Evaluation

The passing threshold is set at **30 points**. The ranking is available at https://asdlab.disi.unitn.it/arena/ranking/. You can test with an equivalent dataset on your own machine. **Note:** in the output files in the sample dataset you will find two numbers: `worst_score` (`long long int`) and `best_score` (`long long int`). These are the values needed to calculate the score of each solution starting from `score`.

Compilation instructions

Below are the instructions for testing your programs on various systems. It is assumed that the source file containing your code is called `the-office.cpp`. The files `the-office.cpp`, `grader.cpp`, and `the-office.h` must be in the same folder.

GNU/Linux systems

```
/usr/bin/g++ -DEVAL -std=c++11 -O2 -pipe -static -s -o the-office grader.cpp the-office.cpp
```

Mac OS X systems

On Mac OS X systems, use the following compilation command:

```
/usr/bin/g++ -DEVAL -std=c++11 -O2 -pipe -o the-office grader.cpp the-office.cpp
```

If you get an error such as `use of undeclared identifier quick_exit`, replace the instruction \ `quick_exit(EXIT_SUCCESS);` in `grader.cpp` with `exit(EXIT_SUCCESS);`.

Windows systems

For Windows 10, you can install the "Windows Subsystem for Linux"[1]. Then you can install the necessary tools to use Visual Studio Code[2] or Visual Studio 2017[3] by following the corresponding guides linked in the notes. Using this system, pay attention to where you save the files and to the names you give them, because you may have difficulties with paths that contain spaces and special characters.
Alternatively, or for systems earlier than Windows 10, you can install *Cygwin*[4], a fully POSIX-compatible environment for Windows. In this case as well, there are guides for configuring the common editors available on Windows so that they use the Cygwin environment, such as Visual Studio[5].
Once Cygwin is installed, you can simulate what happens on arena by compiling your source without including the `the-office.h` header and the `grader.cpp` grader:

```
/usr/bin/g++ -DEVAL -std=c++11 -O2 -pipe -static -s -o the-office the-office.cpp
```

and run the command as follows:

```
timeout.exe 3 ./the-office
```

`timeout.exe` will stop the program after 3 seconds.

[1] `https://docs.microsoft.com/en-us/windows/wsl/install-win10`
[2] `https://code.visualstudio.com/docs/cpp/config-wsl`
[3] `https://devblogs.microsoft.com/cppblog/targeting-windows-subsystem-for-linux-from-visual-studio/`
[4] `https://www.cygwin.com/`
[5] `https://devblogs.microsoft.com/cppblog/using-mingw-and-cygwin-with-visual-cpp-and-open-folder/`

Esempi di input/output

File `input.txt`	File `output.txt`

`3 3 3 2`
`40 2`
`0 2`
`0 3 -1`
`24 2`
`1 2`
`-2 0 2`
`32 1`
`0`
`-1 5 0`
`42 3`
`0 1 2`
`20 2`
`1 2`
`30 2`
`0 2`
	`-3`
	`0 2`
	`0 1`
	`30 12`
	`1 2`
	`1 2`
	`12 8`
	`2 2`
	`0 2`
	`10 20`
	`***`

File `input.txt`	File `output.txt`

`3 3 3 2`
`40 2`
`0 2`
`0 3 -1`
`24 2`
`1 2`
`-2 0 2`
`32 1`
`0`
`-1 5 0`
`42 3`
`0 1 2`
`20 2`
`1 2`
`30 2`
`0 2`
	`-3`
	`0 2`
	`1 2`
	`23 19`
	`1 2`
	`0 1`
	`19 1`
	`2 2`
	`0 2`
	`17 13`
	`***`

File `input.txt`	File `output.txt`

`3 3 3 2`
`40 2`
`0 2`
`0 3 -1`
`24 2`
`1 2`
`-2 0 2`
`32 1`
`0`
`-1 5 0`
`42 3`
`0 1 2`
`20 2`
`1 2`
`30 2`
`0 2`
	`39`
	`0 2`
	`1 2`
	`5 31`
	`1 2`
	`0 1`
	`1 19`
	`2 2`
	`0 2`
	`29 1`
	`***`

File `input.txt`	File `output.txt`

`3 3 3 2`
`40 2`
`0 2`
`0 3 -1`
`24 2`
`1 2`
`-2 0 2`
`32 1`
`0`
`-1 5 0`
`42 3`
`0 1 2`
`20 2`
`1 2`
`30 2`
`0 2`
	`-6`
	`0 3`
	`0 1 2`
	`10 23 9`
	`1 2`
	`1 2`
	`1 19`
	`2 1`
	`0`
	`30`
	`***`