C++ Fundamentals: Exam 2

The following tasks should be submitted to the SoftUni Judge system, which will be open starting **Sunday, 29 July 2018, 09:00** (in the morning) and will close on **Sunday, 29 July 2018, 15:00**. Submit your solutions here: https://judge.softuni.bg/Contests/Compete/Index/1117.

For this exam, the code for each task should be a single C++ file, the contents of which you copy-paste into the Judge system.

Please be mindful of the strict input and output requirements for each task, as well as any additional requirements on running time, used memory, etc., as the tasks are evaluated automatically and not following the requirements strictly may result in your program's output being evaluated as incorrect, even if the program's logic is mostly correct.

You can use C++03 and C++11 features in your code.

Unless explicitly stated, any integer input fits into **`int`** and any floating-point input can be stored in **`double`**. On the Judge system, a C++ **`int`** is a **`32-bit`** signed integer and a C++ **`double`** is a **`64-bit`** IEEE754 floating point number.

NOTE: the tasks here are NOT ordered by difficulty level.

Task 4 – Social (Exam-2-Task-4-Social)

The NTSocial company has developed a new, very simple, professional social network. It consists of users, each of which has an id, a profession, and a list of friends by ids. However, they are having trouble with growing the social network, because users have difficulties finding friends of the same profession. Users do have friends, however analysis shows that they can have many more friends of the same profession, which are in some way connected to their friends.

The company wants to implement a **suggestion system**, which can **give a user** a list of **ids of users** with the **same profession**, which are **friends of friends**, or **friends of friends of friends**, or **friends of friends of friends of friends**… you get the idea.

More formally, if we say that each user **U** is "**connected**" to all their friends, then a **suggestion** for **U** contains **the users**, for which all the following is true:

- They have the **same profession** as **U**
- They are **not friends** of **U** (i.e. not directly connected to **U**)
- They are all **connected to other users**, some of which are **either connected to U**, **or some of their friends are connected to U** and so on. That is, there is a "**path**" that **goes through friend connections** starting **from each suggested user** and ending **with U**.

Write a program that, given a list of users, and their professions, then given a list of friendships between users (as pairs of ids which are friends), and then given a list of user ids searching for friends, prints the suggested ids (of users as described above), in alphabetical order, or indicates that no friends can be suggested if there are no users that meet the all requirements above.

Input

The input will be separated into 3 parts, each ending with a line containing the string **`"---"`**.

Each **line** in the **first part** will describe the **users**, each on a separate line that **containing exactly 2 strings, separated by a single space**. The **first string** in each line will be the **id** of the user. The **second string** will be the **profession** of the user.

The **second part** will describe the current **friendships**, each friendship on a separate line, containing the **ids of the two friends**, separated by a **single space**.

The **third part** will contain the **ids** of **users seeking friends**, each on a separate line.

Output

For **each line** in the **third part** of the input, print the **ids of the suggested** friends, sorted **lexicographically**. If nothing can be suggested for a user, print **`'-'`** (dash).

Restrictions

There will be no more than **`1000`** users. No two users will have the same id.

There will be no more than **`1000`** friendships. No user will be a friend of themselves.

There will be no more than **`50`** unique professions.

The ids and professions will only contain **English letters** and be no more than **8** characters long.

The users for which suggestions have to be printed will be no more than **`100`**.

The total running time of your program should be no more than **`0.2s`**

The total memory allowed for use by your program is **16MB**

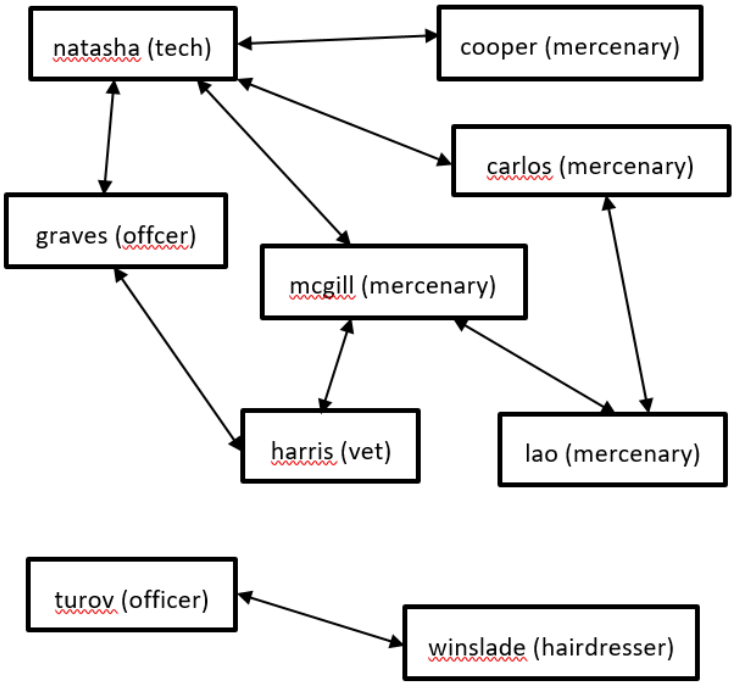
Hints

Rust, BFS.

This task isn't really focused on performance (the input is not very large).

Example I/O

Example Input	Expected Output	Explanation

`mcgill mercenary`
`turov officer`
`natasha tech`
`graves officer`
`harris vet`
`cooper mercenary`
`lao mercenary`
`carlos mercenary`
`winslade hairdresser`
`---`
`cooper natasha`
`turov winslade`
`natasha mcgill`
`carlos natasha`
`lao carlos`
`graves harris`
`graves natasha`
`lao mcgill`
`harris mcgill`
`---`
`natasha`
`mcgill`
`turov`
`---`
	`-`
	`carlos cooper`
	`-`
		`natasha – nobody in the input is a tech, so no suggestions`
		`mcgill – only lao, carlos and cooper are mercenaries, but lao is a direct friend. For the other two there is a path from each of them to mcgill, so they are valid suggestions`
		`turov – the only other officer is graves, however there is no path from graves to turov, so no suggestions.`
		 

Example Input	Expected Output	Explanation
`kenov trainer` `doncho trainer`	`kenov niki` `joro`	`All users here are trainers, and are in the same "group" (everyone has a path to everyone else).`

joro trainer niki trainer --- doncho niki joro doncho kenov doncho kenov niki --- joro kenov ---		joro has a direct connection to doncho, so only niki and kenov can be suggestions kenov has a direct connection to everyone except joro, so only joro can be a suggestion