Dungeon Guard

Deep beneath the mountain lies the Ancient Dungeon, guarded by a solitary sentinel. The guard cannot see the surface world directly, yet he must always know what is happening outside. Six enchanted crystals are his only eyes and ears. Each crystal continuously emits a categorical signal such as quiet, bright, or fresh, describing what it senses beyond the sealed gate.

For centuries, the dungeon relied on human scouts who would open the gate, confirm the situation, and record both the crystal readings and the actual event outside. In which somehow both observations from the scout and the crystal are always aligned. Those verified logs became the training records that the guard now studies. Recently, however, the dungeon administration decided to reduce operational costs by offering the scouts an early-retirement package and depending solely on the crystals. Although the crystals have replaced the scouts, neither is flawless: sometimes both crystals flicker or misreport, and occasionally the scouts' old notes were inconsistent. As a result, simple hand-crafted rules are rarely sufficient. The guard must learn patterns that can tolerate noise and imperfection.

Your task is to help the guard infer the current situation from these historical records and the new crystal readings.

Problem Statement

You are helping the Dungeon Guard predict what is happening outside whether:

Label	**Meaning**
0	Everything is all clear
1	Visitors Approaching
2	There is Monster Invasion

Your only clues are the 6 crystals, each reporting a work (for example: fresh, quiet, low, etc.)

What you are given:

1. A list of **training examples** — each line tells you the 6 crystal readings and the true situation (the label 0, 1, or 2).
2. A list of **query examples** — each line gives you only the 6 crystal readings, and you must guess the correct situation.

Your program must predict the correct label for each query observation

Input Format

N F
f11 f12 ... f1F y1
f21 f22 ... f2F y2
...
T
fN1 fN2 ... fNF yN
q11 q12 ... q1F
1q21 q22 ... q2F
...
qT1 qT2 ... qTF

- N - number of training samples
- F - number of crystals (features)
- Each of the next N lines: each contains F categorical tokens followed by an integer label y $\in \{0, 1, 2\}$
- T - number of query samples
- Each of the next T lines contains F categorical tokens (without label)

Notes

- Feature values are non-empty ASCII strings without spaces (for example bright, smoke, fresh, low).
- All crystals always emit exactly one value per observation; there are no missing crystals.
- Feature order is consistent between the training and query sections.

- Some query rows may contain combinations of feature values never seen together in training.

Output Format

Output exactly T lines. Line i must contain a single integer and the predicted situation label (0, 1, or 2) for query i.

Example

Input	Output
8 6 fresh quiet bright low smooth stable 0 smoke dim flicker high cracked chaotic 2 rot rumble glare medium rough chaotic 2 fresh metallic dim low smooth stable 1 smoke quiet bright high rough chaotic 2 rot dim shadow low cracked stable 0 fresh rumble dim medium smooth stable 1 smoke metallic bright high rough chaotic 2 4 fresh quiet bright low smooth stable rot rumble dim high cracked chaotic smoke dim flicker high rough chaotic fresh metallic bright medium smooth stable	0 0 2 1

Explanation

In the first line, `8 6` means that there are **8 training samples** and **6 features** (6 crystal readings such as *fresh*, *quiet*, *low*, etc.) for each training sample.

The next **8 lines** each contain:

- **6 feature values** (the readings from each crystal), and
- **1 integer label** at the end, which represents the correct situation:
  - `0` = All Clear
  - `1` = Visitors Approaching
  - `2` = Monster Invasion

After these 8 lines, the number **4** indicates that there are **4 query samples**.
Each query sample contains **only the 6 crystal readings** (no label).

Your task is to **predict the correct label** for each of these 4 query samples and output them in order — one per line.

Constraints

- Execution time limit: ≤ 1 minute
- Memory limit: ≤ 256MB

Scoring

$$Score = 100 \times \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ testcases}$$

The total score is the average across all test files.
Hidden tests are designed to reward generalization and robustness:

- some query readings use value combinations that never occur in the training data;
- a small fraction of training labels may be noisy, reflecting occasional disagreement between scouts and crystals

Story Appendix

The scouts' old notebooks — meticulously cross-checked with crystal logs — became the guard's only window to truth. After the early-retirement program, the scouts are gone and the crystals are now the only observers. Your program must cope with these uncertainties to keep the guard informed — before either error or monster reaches the gate.

Yogi’s Star Constellations

Yogi is an astronomer at the Celestial Mapping Institute. He has observed the stars for many years, recording thousands of bright points scattered across the night sky with his telescope. Each detected star is mapped onto the star chart, which is a 2D plane where the star's position is represented by its coordinates. To make the study of stars more systematic, Yogi defines constellations, which are groups of stars that are “close” to each other in the sky according to a precise set of rules known as the Constellation Formation Rules.

In the Constellation Formation Rules, the unit of distance in the star chart is light-years, and the coordinate origin is defined such that all positions have non-negative coordinates.

Two stars are considered neighbors if the distance between them does not exceed a given threshold T (in light-years).

A star is called a core star if it has at least M neighboring stars (including itself).

A constellation of a core star p is formed by:

1. including all of p’s neighbors, and
2. for every neighbor q of p that is also a core star, including all of q’s neighbors as well. This process is repeated recursively until no more stars can be added.
3. Constellations are discovered in the order of their core stars’ coordinates. The core star with the lowest x-coordinate is processed first, and if there is a tie, the one with the lowest y-coordinate is chosen. It is guaranteed that no two stars share the same coordinates.

Problem Statement

Yogi’s objective is to form the maximum number of distinct constellations based on these rules. However, he finds drawing constellations by hand infeasible, so he asks you to help him write a program to automatically perform this Constellation Formation Algorithm.

Input	**Output**
N M T x_1 y_1 x_2 y_2 ... x_N y_N	o_1 o_2 ... o_K

Input Format

The first line contains three numbers N, M, and T.

- $1 \le N \le 2{,}000$ — the number of stars
- $2 \le M \le N$ — the minimum number of neighboring stars required for a star to be considered a core star
- $0.0 < T \le 100{,}000.0$ — the neighborhood radius (in light-years).

The next N lines each contain the coordinates of a star (x_i, y_i), where x_i and y_i are integers satisfying $0 \le |x_i|, |y_i| \le 10^6$.

The distance between two stars (x_i, y_i) and $(x□, y□)$ is defined as:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Output Format

The output contains N lines. Each line corresponds to star i (in the same order as the input) and indicates the constellation number it belongs to. If the star does not belong to any constellation, output -1. Constellation numbers start from 1, in the order they are discovered according to the rules above.

Examples

Input	Output
10 3 2.0 1 1 2 1 1 2 8 8 8 9 25 20 9 8 0 0 8 7 2 2	1 1 1 2 2 -1 2 1 2 1

Explanation

In the first line,
`10 3 2.0` means:

- 10 → Number of stars (N)
- 3 → Minimum number of neighboring stars needed to be a core star (M)
- 2.0 → Maximum distance (in light-years) for two stars to be considered neighbors (T)

The next 10 lines give the positions (x, y) of each star.
Each star belongs to a constellation (a group of stars that are close enough to each other).
If a star isn't close enough to form or join any constellation, it is marked with -1.
In the output, each line shows the constellation number for the star at the same position in the input list.
For example:

- Star `(1,1)` belongs to constellation 1.
- Star `(8,8)` belongs to constellation 2.
- Star `(25,20)` isn't near any other stars, so it gets -1.

Scoring

$$Score = 100 \times \frac{Number\ of\ correct\ predictions}{T}$$

The total score is the average across all test files.

The NeuroNet GPU

You are the new lead AI engineer at NeuroNets Inc., a company building a revolutionary autonomous driving system. This system is not a single, monolithic AI model, but a complex pipeline of interconnected machine learning models that work together. For example, a `Perception Model` first identifies objects like cars, pedestrians, and traffic lights from camera footage. Its output is then fed into a `Prediction Model` which predicts the movement of those objects. Finally, a `Path Planning Model` uses this information to decide how to steer the car.

Training these models is computationally expensive and requires powerful, specialized hardware: Graphics Processing Units (GPUs). Your company has a cluster of identical, high-end GPUs. The challenge is that there are dependencies between the models. You can't train the `Path Planning Model` until both the `Perception` and `Prediction` models that feed it data are fully trained.

Your job is to create a schedule for training all the necessary models on the GPU cluster. The entire system can't be deployed for testing until the very last model has finished training. Your goal is to design a schedule that minimizes this total time, getting your self-driving car on the road as quickly as possible!

Problem Statement

You are given a set of `N` machine learning training jobs, a set of `M` identical GPUs, and a list of dependencies between the jobs. Each job `i` has a known processing time `t_i` (the number of hours it takes to train).

A job can only be scheduled to start at time `T` if all of its prerequisite jobs have been completed by or at time `T`. A GPU can only run one job at a time. Once a job starts on a GPU, it runs to completion without interruption (it is non-preemptive).

Your task is to assign each job to a specific GPU and a start time, such that all dependencies are met, and the makespan is minimized. The makespan is defined as the completion time of the job that finishes last.

Input Format

The first line of input contains three space-separated integers: N (the number of jobs), M (the number of GPUs), and P (the number of dependencies).

The second line contains N space-separated integers: t1, t2, ..., tN, where ti is the processing time for job i. Jobs are numbered 1 to N.

The next P lines each contain two space-separated integers u and v, signifying that job u is a prerequisite for job v (job u must be completed before job v can begin).

Output Format

Your output must consist of N lines. Each line describes the schedule for one job and must contain three space-separated integers: job_id, gpu_id, and start_time.

- job_id: The ID of the job (from 1 to N).
- gpu_id: The ID of the GPU you assign this job to (from 1 to M).
- start_time: The non-negative integer time at which the job begins training.

You can print the lines in any order of job_id. The schedule you output must be valid. A schedule is valid if:

1. For every dependency (u, v), the start time of v is greater than or equal to the completion time of u. (i.e., start_time[v] ≥ start_time[u] + tu).
2. For any single GPU, the time intervals for the jobs assigned to it do not overlap.

Constraints

- $1 \leq N \leq 200$
- $1 \leq M <= 50$
- $0 \leq P \leq N (N - 1) / 2$
- $1 \leq ti \leq 1000$
- The dependency graph is guaranteed to be a Directed Acyclic Graph (DAG), meaning there are no circular dependencies.

In 25% of the test cases, we have $N \leq 15$.

Scoring

For each test case, your submission will be checked for validity

- If your output describes an invalid schedule, you will receive 0 points for that test case.
- If your schedule is valid, your score is based on its makespan. Let YourMakespan be the makespan of your schedule. Let BestMakespan be the makespan of the best-known solution for that test case. Your score for the test case will be calculated as:

$$Score = 5 \times \frac{BestMakeSpan}{YourMakeSpan}$$

- Your total score for the problem is the sum of your scores over all 20 test cases. The maximum score for the problem is 100.

Examples

Sample Input

Input	Output
5 2 3 3 5 2 4 6 1 3 2 4 2 5	1 1 0 2 2 0 3 1 3 4 2 5 5 1 5

Explanation of Sample Input:
The first line contains 5 Jobs, 2 GPUs and 3 Dependencies
The next line is the time it takes for each job to complete
This means

- Job 1 = 3 hours
- Job 2 = 5 hours
- …

The next 3 lines are dependencies, meaning in this case

Job 3 can only after Job 1 finishes
Job 4 and 5 can start only after Job 2 finishes

Explanation of Sample Output
Each line of the output means:
`[Job Number] [Which GPU the Job runs on] [When it start]`
For example, 3 1 3 means Job 3 starts on GPU 1 at time = 3

Step - By - Step Explanation

- At time = 0, jobs 1 and 2 are ready. We start Job 1 on GPU 1 and Job 2 on GPU 2.
    - Job 1 (duration 3) runs on GPU 1 from time = 0 to time = 3.
    - Job 2 (duration 5) runs on GPU 2 from time = 0 to time = 5.
- At time = 3, Job 1 finishes. GPU 1 is now free. Job 3 (which needs Job 1) is now Available.
    - Job 3 (duration 2) starts on GPU 1 from time = 3 to time = 5.
- At time = 5, Job 2 and Job 3 both finish. GPU 1 and GPU 2 are both free. Jobs 4 and 5 (which need Job 2) are now available.
    - Job 4 (duration 4) starts on GPU 2 from time = 5 to time = 9.
    - Job 5 (duration 6) starts on GPU 1 from time = 5 to time = 11.

The last job to finish is Job 5 at time = 11. The **makespan is 11.**

Validity Check:

- Dependencies: All dependencies are met. (e.g., Job 3 starts at time = 3, which is when its prerequisite Job 1 finishes).
- GPU Overlaps: There are no overlaps. GPU 1 runs jobs in [0,3], [3,5], [5,11]. GPU 2 runs jobs in [0,5], [5,9].

Score Calculation:
Assuming the best possible makespan for this input is 11, the score would be:

$$5 \times \frac{11}{11} = 5\ Points$$

Example of an Invalid Output:

1 1 0
2 2 0

3 2 2
4 2 5
5 1 5

Explanation of Invalidity:
This schedule is invalid. Let's analyze the assignment for Job 3. The output contains `3 2 2`. This means that Job 3 should start on GPU 2 at time = 2. This is invalid for two reasons:

1. Dependency Violation: Job 3 requires Job 1 to be finished. Job 1 is scheduled to finish at time = 3. Starting Job 3 at time = 2 violates this prerequisite.
2. Resource Conflict (GPU Overlap): Job 2 is already scheduled to run on GPU 2 during the time interval [0, 5]. Attempting to start Job 3 on the same GPU at time = 2 would mean both jobs are running on GPU 2 simultaneously, which is not allowed.

Score Calculation:
Because the schedule is invalid, the score for this output is **0 points.**