Contest Problems
Philadelphia Classic, Spring 2022
University of Pennsylvania

Rules and Information

This document includes 12 problems. Novice teams do problems 1-8; standard teams do problems 5-12.

Any team which submits a correct solution for any of problems 1-4 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-4.**

Problems 1-4 are easier than problems 5-8, which are easier than problems 9-12. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty.

You may use the Internet only for submitting your solutions, reading Javadocs or Python documentation, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

As you may know, you can choose to solve any given problem in **Java, Python, or C++**. We have provided stub files for all questions in all languages that take care of the input parsing for you. **Do not modify any of the parsing or output code**. Just fill out the stub methods in each file and submit it with exactly the same name.

Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long myLong = 1000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. An Angry K-Pop Producer

`KpopProducer.py, KpopProducer.java, KpopProducer.cpp`

A K-Pop producer Ishaan is extremely angry with his lazy backstage dancers while filming the next BlackRed music video! To punish them, he gives them each a letter to hold (which come together to form a string) and then yells out a series of positive integers. For each integer, the dancers must shift right by that amount, but must "wrap" around the right end so that their position in line has changed. (ex. With each call, some rightmost dancers will have moved to the leftmost side.) Can you write a program that determines the final string after he *finally* loses his voice?x

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`danceCommand(s, arr)`

- **s**: a string that represents the initial string
- **arr**: an array of positive integers that represent the "right shifts" of the string

Your function should return a string.

Constraints

$1 \leq |string| \leq 10000$
$1 \leq |arr| \leq 10000$
$0 \leq x \leq 100000$ where x is an element of arr.

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **s**.
The second line contains **n**, the length of the array.
The third line contains **arr** which consists of n integer values, each denoted as x.

Output

For each test case, print a string representing the shifted string.

Sample Input	Parsed Input	Sample Output	Explanation
1 kpop 1 2	`danceCommand(“kpop”, [2])`	opkp	The first two letters are shifted by 2 right, and the last two rightmost letters are wrapped to the left.
1 blackred 1 3	`danceCommand(“blackred”, [3])`	`redblack`	The first 5 letters are shifted by 3 right, and the 3 rightmost letters are wrapped to the left.
1 capital 4 1 3 7 1	`danceCommand(“capital”, [1, 3, 7, 1])`	`pitalca`	After each of the 4 shifts, the string should be: lcapita italcap italcap pitalca

2. Royal Translator

`RoyalTranslator.py, RoyalTranslator.java, RoyalTranslator.cpp`

As Queen Elizabeth II's newly recruited translator, Michelle IV is excited to begin her job at Buckingham Palace! But having just arrived from South Korea, she is extremely jet lagged and needs your help translating. Queen Elizabeth II has given her a string of only letters and spaces, forming a sentence of words (strings), and a "translation guide" mapping from string to string. Note: There is no punctuation in the sentence, and there are spaces separating each string. Help Royal Translator Michelle IV give Queen Elizabeth II her translated string by writing a program that changes each word in the old sentence to its mapped value in the translation guide.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`translate(message, map)`

- **message**: a sentence of strings (made of spaces and characters) to be translated.
- **map**: a map that represents a translation of each possible string to another string.

Your function should return the translated string.

Constraints

$0 \leq words\ (separated\ by\ spaces)\ in\ message \leq 10000$
$0 \leq map \leq 10000$

(cont. on next page)

2. Royal Translator

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **message**
The second line contains **mappingCount**
The next **mappingCount** lines contain two strings each, separated by a space, representing a possible string and its translation

Output

For each test case, print a sentence representing the translated message.

Sample Input	Parsed Input	Sample Output	Explanation
1 tired of translating 3 tired scared of to translating fly	translate(“tired of translating”, {{tired: scared}, {of: to}, {translating: fly}})	scared to fly	tired → scared of → to translating → fly
1 proud red and blue 4 proud gone red with and the blue wind	translate(“proud red and blue”, {{proud: gone}, {red: with}, {and: the}, {blue: wind}})	gone with the wind	proud → gone red→ with and -> the blue -> wind

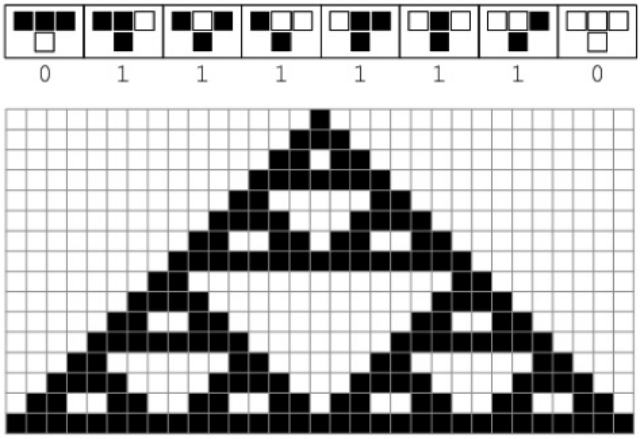
1 and 0	`translate(“and”, {{}})`	and	and → and No mapping, so it’s left alone

3. Egyptian Mosaic

EgyptianMosaic.py, EgyptianMosaic.java, EgyptianMosaic.cpp

Ziwen is a triangle fanatic. All of his friends think he needs professional help but Ziwen refuses. One day during geography class, he sees a picture of Egyptian pyramids and immediately falls in love. The next day he promptly skips class and hops on a flight to Egypt, claiming that this spiritual journey will bring him closer to his beloved shape.

On the way to the pyramids, he sees a shop offering various Egyptian arts and crafts. Intrigued and hoping to find various triangle shaped trinkets inside, he enters. To Ziwen's surprise, there is a mosaic class going on, where an artist is teaching several people how to make their own mosaic art on a grid of squares. Ziwen cringes at the thought of squares and turns to leave, but the teacher notices him and beckons him over. She hands him a mosaic kit with some starter templates. Ziwen is thrilled to notice a pattern that seems to be made of many triangles.



<< START READING HERE FOR PROBLEM >>

The teacher shows him how to make the mosaic. The first (top) row of the grid is full of white tiles, but the center tile has a black tile. To fill in the next row, Ziwen must follow the rules shown above the template. For each tile he places in the new row, he looks at the three tiles directly northwest, north, and northeast of the current position and matches it with a rule in order to determine which color tile to place down in the current position. A tile outside the area of the mosaic canvas can be treated as a white tile. Ziwen quickly realizes that with this method, he can fill any grid he wants with this epic pattern! Can you write a program that helps Ziwen easily visualize what this pattern will look like on a given grid?

3. Egyptian Mosaic

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`fillMosaic(gridWidth, gridHeight)`

- **gridWidth**: the number of columns, always an odd number, will be a positive integer >= 1
- **gridHeight**: the number of rows, can be odd or even, will be a positive integer >= 1

Your function should return a 2D string array representing the filled out mosaic:

- # - represents a black tile
- . - represents a white tile
- The number of columns will always be an odd number, so the first row should contain all white tiles except for a black tile in the center.

Constraints

$1 \leq gridWidth \leq 10^4$

$1 \leq gridHeight \leq 10^4$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **w** and **h**

Integer **w** representing the grid width (number of columns)
Integer **h** representing the grid height (number of rows)

Output

For each test case, return the completed mosaic in 2D string array form.

(cont. on next page)

3. Egyptian Mosaic

Sample Input	Parsed Input	Sample Output	Printed Output
1 3 3	fillMosaic (3, 3)	[[“.”, “#”, “.”], [“#”, “#”, “#”], [“#”, “.”, “#”]]	.#. ### #.#
1 31 16	fillMosaic (31, 16)	(see printed output →)	...............#............... ....###.............. ....##.##............. ....#######............ ....##.....##........... ....####...####.......... ....##..##.##..##......... ....###############........ ....##.............##....... ......####.........####...... .....##..##.......##..##..... ....########.....########.... ...##......##.....##......##... ..####....####...####....####.. .##..##..##..##.##..##..##..##. ################################
1 15 8	fillMosaic (15, 8)	(see printed output →)	.......#....... ......###...... .....##.##..... ....#######.... ...##.....##... ..#####...#####.. .##..##.##..##. ###############

4. Widest Mountain

`WidestMountain.py, WidestMountain.java, WidestMountain.cpp`

Sasha arrives at her next travel location, Switzerland! However, all her prior traveling has made her tired (she hasn't been going to the gym lately). Since she still has a lot of traveling to do, she wants to get back into shape. She has a map of a trail in the Alps, which includes a list of the elevations at each mile of the trail. To maximize her gains, she wants to find the longest segment of the trail that is a "mountain" and run along it. A mountain is a sequence of elevations that strictly increases, reaches a peak, then strictly decreases. The peak is taller than any point in the mountain. Help Sasha determine the longest portion of the trail she can run.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`findWidestMountain(n, mountainRange)`

- **n**: the length of the trail (in miles)
- **mountainRange**: an array of integers containing the elevations of the trail at each mile

Your function should return an integer representing the length of the widest mountain that Sasha can run across.

Constraints

$1 \leq n \leq 10^6$

$1 \leq |mountainRange| \leq 10^6$

(cont. on next page)

4. Widest Mountain

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **n.**
The second line contains **mountainRange.**

Output

For each test case, output the length of the widest mountain.

Sample Input	Parsed Input	Sample Output	Explanation
1 5 6 3 5 2 7	`findWidestMountain (5, [6, 3, 5, 2, 7])`	3	The widest mountain is [3, 5, 2]
1 4 2 5 7 13	`findWidestMountain (4, [2, 5, 7, 13])`	0	There is no mountain in this trail as it is strictly increasing and never decreases. Therefore, Sasha cannot run on this trail
1 7 3 8 5 4 1 20 2	`findWidestMountain (7, [3, 8, 5, 4, 1, 5, 2])`	5	The widest mountain is [3, 8, 5, 4, 1]
1 6 2 5 3 7 1 2	`findWidestMountain (6, [2, 5, 3, 7, 1, 2])`	3	The widest mountain is [2, 5, 3] or [3, 7, 1]

5. Shifting Sands

`ShiftingSands.py, ShiftingSands.java, ShiftingSands.cpp`

Brian is feeling adventurous so he flies to Algeria to observe the Sahara Desert. He keeps a journal about what he sees and in that journal he writes that he found a magical 4x4 grid of sand dunes. In this 4x4 grid, gusts of wind blow in any 4 of the cardinal directions and the wind is so strong that it moves the sand dunes. Each sand dune has an associated size given by an integer. Sand dunes can have size 0.

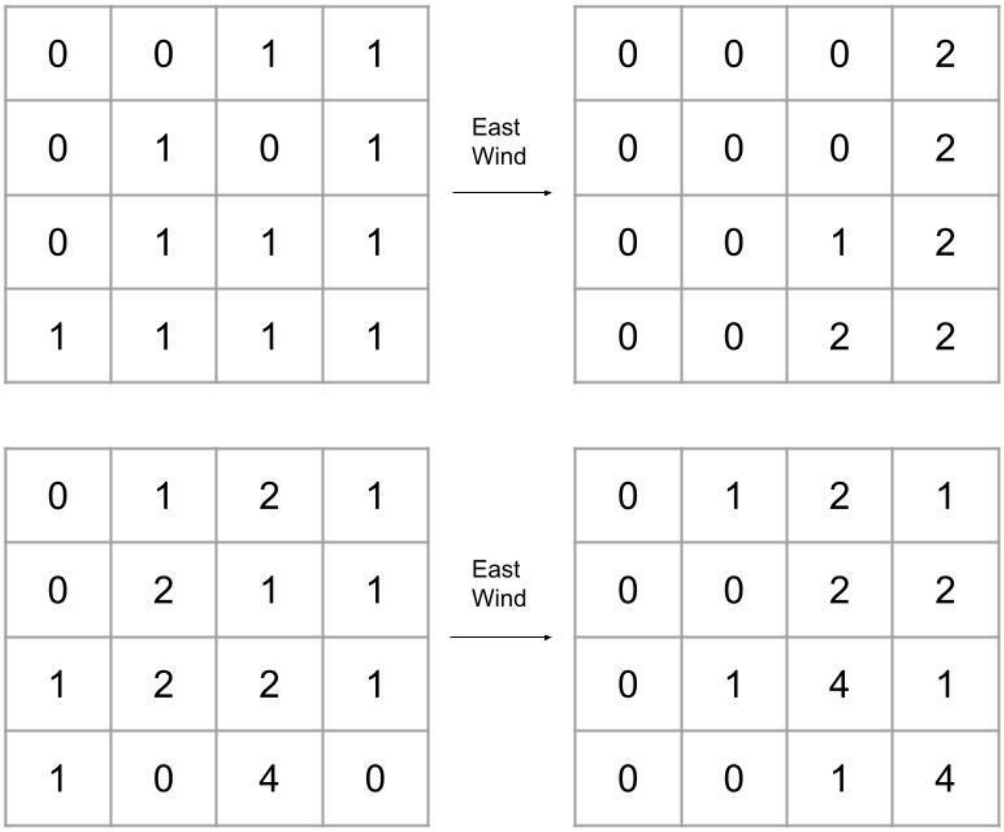
Brian also writes that the sand dunes are moved by the wind according to the following rules:

- Sand dunes cannot move outside of the 4x4 grid
- Sand dunes will blow as far in the direction of the wind as possible.
- When two sand dunes collide from a gust of wind, they can combine into one sand dune if they have the same size. They will combine into a sand dune that has double the size.
- Dunes can only combine once per gust of wind.
- When 3 dunes of the same size collide into each other the two that are further in the direction of the wind will combine. For example, if the wind is blowing to the west the two most western dunes will combine.

(cont. on next page)

5. Shifting Sands

Brian also leaves some drawings of the sand dunes moving:



In his journal, Brian also records the original state that the dunes are in and a list of the directions that wind blows (ex: N, W, W, S, E, N). Can you design an algorithm to find the final state of the sand dunes given the original state and the directions that the wind blows?

(cont. on next page)

5. Shifting Sands

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`shiftingSands(dunes, winds)`

- **dunes**: 2D array of integers representing the height of all of the dunes. **dunes** represent the original height of all of the dunes.
- **winds**: an array of Strings representing the directions the wind blows in

Your function should return a 2D array representing the height of the dunes after the wind has blown.

Constraints

$dunes = 4x4$

$0 \leq winds \leq 10^4$

(cont. on next page)

5. Shifting Sands

Input

The first line represents the number of test cases **T**.
For each test case:
The first 4 lines are 4 integers representing the heights of the dunes, **dunes**
The 5th line is a sequence of letters representing the directions the wind blows in, separated by spaces, **winds**

Output

For each test case, output the heights of the dunes.

Sample Input	Parsed Input	Sample Output
1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 E	shiftingSands([[0,0,1,1], [0,1,0,1], [0,1,1,1], [1,1,1,1]], [“E”])	0 0 0 2 0 0 0 2 0 0 1 2 0 0 2 2
1 0 1 2 1 0 2 1 1 1 2 2 1 1 0 4 0 E	shiftingSands([[0,1,2,1], [0,2,1,1], [1,2,2,1], [1,0,4,0]], [“E”])	0 1 2 1 0 0 2 2 0 1 4 1 0 0 1 4
1 0 1 1 2 0 0 2 1 0 2 1 1 4 4 2 1 E W N S	shiftingSands([[0,1,1,2], [0,0,2,1], [0,2,1,1], [4,4,2,1]], [“E”, “W”, “N”, “S”])	4 0 0 0 2 0 0 0 4 1 0 0 8 2 1 0

6. Lantern Festival

`LanternFestival.py, LanternFestival.java, LanternFestival.cpp`

Lantern festivals are a tradition in China. Instead of letting them go Tangled-style, Andrew wants to hang lanterns on his house. He has **n** distinct integer labeled lanterns from 0 to n-1. His house only has room to attach exactly one lantern, but he can hang lanterns under other lanterns. To do so, a lantern can have a horizontal stick attached beneath it with exactly one lantern attached to each side of the stick. The left lantern must be of smaller value than the right one. A lantern can be attached to at most 1 stick.

Andrew is a little picky and has a list of lanterns he wants attached directly under certain lanterns. Given his preferences, can you find if there is a valid way to hang all **n** lanterns? If so, return the order of lanterns from top to bottom, left to right.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`lanternFestival(n, prefs)`

- **`n:`** an integer that represents the total number of lanterns
- **`prefs:`** a 2D array that represent preferences of lanterns
    - The first element contains the top lantern
    - The second element contains the bottom lantern

Your function should return a list that represents the lantern formation or a list containing -1 otherwise.

Constraints

$0 < n \leq 1000$
$0 < |prefs| \leq 1000$

Input

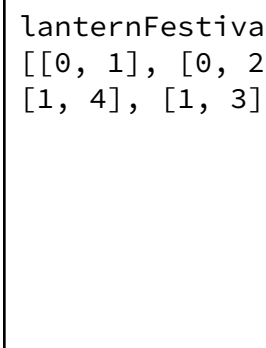
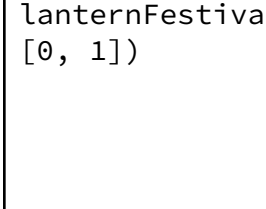
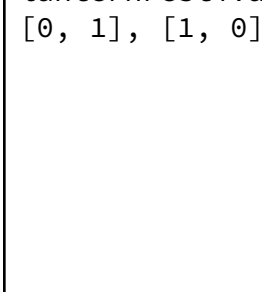
The first line represents the number of test cases **T**.

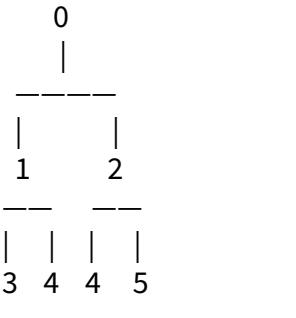
For each of the **T** cases:
The first line represents **n**.
The second line represents the number of preferences.
The next |**prefs**| lines contain two integers separated by a space, where the first element is the top lantern and the second element is the bottom lantern.

Output

For each test case, output the order of the lanterns from top to bottom, left to right, where each lantern is on a new line, or -1 if an arrangement is not possible.
(cont. on next page)

6. Lantern Festival

Sample Input	Parsed Input	Sample Output	Explanation
1 5 4 0 1 0 2 1 4 1 3	lanternFestival(5, [[0, 1], [0, 2], [1, 4], [1, 3]])	0 1 2 3 4	 This is a valid formation because exactly 1 lantern is attached to the root (0) and the other lanterns obey Andrew's preferences.
1 2 1 0 1	lanternFestival(2, [0, 1])	-1	 This is not valid because there needs to be two lanterns attached to the stick below 0
1 2 2 0 1 1 0	lanternFestival(2, [0, 1], [1, 0])	-1	 This isn't physically possible because 0 is

			both above and below 1.
1 6 6 0 1 0 2 1 4 1 3 2 5 2 4	lanternFestival(6, [0, 1], [0, 2], [1, 4], [1, 3], [2, 5], [2, 4]}	-1	 4 is attached to both 1 and 2, which is not allowed.

7. Italy Itinerary

`ItalyItinerary.py, ItalyItinerary.java, ItalyItinerary.cpp`

Tien suddenly awakes, his back against a sandy shore, and immediately squints against the bright afternoon sun. Gentle, warm waves are lapping at his sides. An annoying seagull is pecking near his toes relentlessly, but quickly flies off when he waves his foot at it. He can vaguely hear some tourists chattering and playing in the water a few meters away.

Just when Tien decides he never wants to move from this wonderful paradise of a spot, a shadow looms over him and blocks the sun. He opens his eyes with great reluctance to see someone standing over him. “Ciao!” the strange man says with a warm smile. “Welcome to Italy!” Before Tien can wonder how the man even knows he’s a tourist, the stranger hands Tien a piece of paper. “Feel free to use this itinerary while you are here. Addio!” The man walks away with no further explanation.

<< START READING HERE FOR PROBLEM >>

Tien decides he has seen weirder things, shrugs and starts to read the itinerary. It appears to be a list of events to do around Italy, listed in an order such that the travel time would be most time and money efficient if one did the events in the given order.

Tien wants to follow the itinerary and do as many fun events as he can, but doesn’t know if he will have the energy for everything. He takes out a pen and marks next to each event two numbers, one estimating its fun value and one estimating its recovery requirement. The fun value dictates how much enjoyment Tien will get out of attending an event, and the recovery requirement dictates how many consecutive events Tien will have to skip right after attending said event in order to recover his energy (he’s getting old after all).

Of course, even when Tien is on vacation, algorithms never leave his mind, so he immediately wonders if there is a way to write a program to automatically calculate which events he should go to and which ones he will skip. If he chooses to attend an event, he will receive the total fun value but also have to skip the next x events where x is the recovery requirement value. He also has enough money left from his last TA paycheck to splurge on ONE event specifically, so he can double the fun value for any ONE

7. Italy Itinerary

chosen event that he wants to attend (the recovery requirement stays the same). The program should return the maximum collective fun value Tien can obtain from a given itinerary.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`maximizeFun(itinerary)`

- **`itinerary`**: a 2D integer array where `itinerary[i] = [funValue`$_i$`, energyRequirement`$_i$`]`. Each row represents an event, and the events are already ordered as on the itinerary.

Your function should return the MAXIMUM overall fun value this itinerary can provide given your fun and energy preferences/requirements.

Constraints

$1 \leq |itinerary| \leq 600,000$
$1 \leq |funValue_i, energyRequirement_i| \leq 600,000$

Input

The first line represents the number of test cases, **T**.
For each following test case:
The first line contains **n**, or number of events.
The next **n** lines contain two ints each, representing the fun value and energy requirement.

Output

For each test case, output the MAXIMUM overall fun value this itinerary can provide.

(cont. on next page)

Sample Input	Parsed Input	Sample Output	Explanation
1 3 1 2 2 1 3 1	maximizeFun([[1, 2], [2, 1], [3, 1]])	6	In this scenario, you are only able to do 1 event. Thus, you should do the last event. You can double the fun value which will yield 3*2=6.
1 4 3 2 5 3 4 4 4 5	maximizeFun([[3, 2], [5, 3], [4, 4], [4, 5]])	11	If you do the 1st event, you get 3 fun points, then you have to skip the next two events but you can do the 4th event and choose to double its fun to get 8 more fun points, yielding a total of 11. Skipping the 1st event only leaves the option of doing either the 2nd, 3rd, or 4th event individually since your energy won't recover to allow another event, so the maximum you can get (by doubling) on each is 10, 8, or 8 respectively, which are all less than 11.

8. VanuaTube Adventures

VanuaTubeAdventures.py, VanuaTubeAdventures.java, VanuaTubeAdventures.cpp

Meggie and Vatsin are enjoying their summer in the beautiful country of Vanuatu. Vanuatu is composed of **m*n** islands, but a sudden earthquake caused the islands to shift their positions such that they miraculously ended up in an m x n grid of islands! The island that Meggie and Vatsin were chilling at somehow ended up in the bottom left hand corner of the grid. The earthquake also made the bellies of Meggie and Vatsin quake, so they need to get some food for themselves quickly! The only place to get food in the country is located at the island at the top right corner of the grid of islands. The islands are connected by tubes such that each island is connected to the island above, below, to the left, and to the right of it (if such an island exists). Vatsin and Meggie are only allowed to go to the island directly north of them or directly east of them.

Furthermore, some of the islands have tickets which they can trade in for a meal at the top right island. If Vatsin and Meggie encounter a ticket along their path, they **must** collect it. Because Vatsin and Meggie want to both eat the same amount of meals, they want to collect an even number of tickets along the way.

As the concierge at Vanuatu island, you want to guarantee that no matter what path Vatsin and Meggie take, they end up with an even number of tickets. Moreover, you know that there are **q** plans where the *i*th plan involves adding $\mathbf{d_i}$ tickets to islands such that each island can have at most one ticket. For each of the **q** plans, determine whether it is possible to place the $\mathbf{d_i}$ additional tickets such that no matter what path they take from the bottom left to the top right, Vatsin and Meggie will collect an even number of tickets.

8. VanuaTube Adventures

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`tubeAdventures(n, m, q, islands, queries)`

- **n :** number of rows of islands
- **m:** number of columns of islands
- **q:** number of queries
- **islands**: an array of strings where each string represents a row of islands and a character is a "." if there is no ticket and an "x" if there is a ticket.
- **queries** an array of integers where each element represents the number of tickets to place

Your function should return a list containing YES if it is possible to place the additional tickets, and NO if it is not possible for each query.

Constraints

$0 \leq m \times n \leq 5 \times 10^4$

$1 \leq q \leq 10^5$

Input

The first line contains **n m q**
The next **n** lines contain a string representing a row of islands
The next **q** lines contain an int representing the number of tickets we want to place

Output

Output YES if there exists a valid ticket placement, and NO otherwise per query.

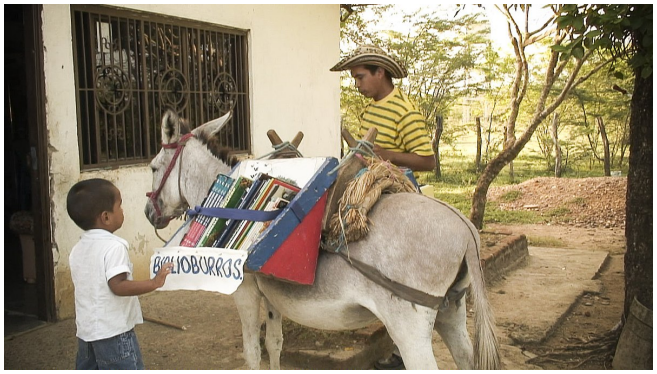
(cont. on next page)

8. VanuaTube Adventures

Sample Input	Parsed Input	Sample Output	Explanation
1 2 3 3 ... ... 1 2 3	tubeAdventures(2, 3, 3, [“...”, “...”], [1,2,3])	NO YES YES	There’s no way to place only one ticket so that Vatsin is always in front of Meggie. For two tickets we could place them as follows: ..x x.. For three tickets: .x. x.x
1 3 3 5 ... ..x ..x 1 2 3 4 5	tubeAdventures(3, 3, 5, [“...”, “..x”, “..x”], [1,2,3,4,5])	NO NO YES NO YES	There’s no way to place only one, two, or four tickets so that Vatsin is always in front of Meggie. For three tickets we could place them as follows: xx. .xx ..x For five tickets: xxx .xx x.x
1 3 6 3 .x.... .x.x.x .....x 8 9 10	tubeAdventures(3, 6, 3, [“.x....”, “.x.x.x”, “.....x”], [8,9,10])	NO NO YES	There’s no way to place only eight or nine tickets so that Vatsin is always in front of Meggie. For ten tickets we could place them as follows: xxxxx. .xxxxx x.xxxx

9. Biblioburro

`Biblioburro.py, Biblioburro.java, Biblioburro.cpp`



Grace and Ani are librarians backpacking in Colombia. They love the nature around them but are deeply booksick, as it has been over 24 hrs since they have been within vicinity of any form of physical literature. When they come across a man with two donkeys, they are thrilled and intrigued to see that strapped to the donkeys' backs are shelves of books! They introduce themselves and learn that the man's name is Luis Sorano. Luis has spent the last ~30 years traveling around rural Colombia with his two "biblioburros" (a mix of the words biblioteca and burro, which are library and donkey in Spanish respectively), providing library access to kids in hard-to-reach, mountainous areas. He laments that even though he loves his job, his donkeys are getting old and he wants to only travel efficiently through the mountains for their sake.

Grace and Ani have had prior programming experience and are eager to help. Luis gives them an ordered list of mountain heights of the mountains that he travels on. When making a trip he will pick a subset of mountains to visit and he will visit them in the increasing order of their positions on the list. A list is considered "efficient" if for ALL POSSIBLE TRIPS he takes through the mountains the absolute difference between consecutive mountains is strictly decreasing as he travels in order from the first mountain to the last mountain. The program should return a boolean True if a list is efficient, and False otherwise.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`efficientTrip(mountainList)`

- **`mountainList`**: an array of integers that represents the height of mountains, in Luis's preferred traveling order

Your program should return a boolean True if a list is efficient, and False otherwise.

Constraints

$1 \leq testcases \leq 10^5$

$1 \leq |mountainList| \leq 100$

9. Biblioburro

$1 \leq mountainHeight \leq 10^{18}$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains the number of mountains
The second line contains the list of mountain heights, separated by spaces

Output

Output 1 if a list is efficient and 0 if it isn't

Sample Input	Parsed Input	Sample Output	Explanation
1 5 1 8 4 6 5	efficientTrip(5, [1, 8, 4, 6, 5])	1	If he chooses just a single mountain or two mountains to travel to, then it is obviously efficient. It can be checked that if any 3 or 4 mountains are chosen the gaps are decreasing. For example, if he chooses the 1st, 2nd, and 4th mountains (i.e. the subarray [1, 8, 6]), the first gap is |1 - 8| = 7 and the second gap is |8 - 6| = 2, so the gaps are decreasing.
1 3 5 8 6	efficientTrip([5, 8, 6])	1	Note that |5 - 8| = 3 and |8 - 6| 2, so the gaps are decreasing.
1 4 1 2 5 6	efficientTrip([1, 2, 5, 6])	0	If we chose the 1st, 2nd, and 3rd mountains, the gaps are |1 - 2| = 1 and |2 - 5| = 3, so the gaps between consecutive mountains are not strictly decreasing.

10. Ethan Chee Superhero

`Superhero.py, Superhero.java, Superhero.cpp`

PClassic's resident superhero Ethan Chee is busy at work helping ordinary civilians. Today, he has traveled to Universal Studios Singapore to help **n** children be brave thrill seekers for the day.

The **n** children are in line for a roller coaster with a circular ride vehicle. Passengers sit around the circumference of the circular vehicle. Because Universal Studios is so rich, they have managed to engineer a ride vehicle which can fit an unlimited number of passengers.

However, the children are too scared to ride without their best friend. The children are numbered from **0** to **n - 1**. Each child has a best friend and will ride only if they can sit next to their best friend. You may assume that no child is their own best friend.

Given a 0-indexed integer array **bff**, where **bff[i]** denotes the i^{th} child's best friend, help Superhero Ethan Chee determine the maximum number of children he can help face their fears on the roller coaster.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`superhero(n, bff):`

- **`n:`** an integer that represents the total number of children.
- **`bff:`** A 0-indexed array where **bff[i]** denotes the i^{th} child's best friend

Your function should return an integer denoting the maximum number of children Ethan Chee can get to ride the roller coaster.

Constraints

$2 \leq n \leq 10^4$
$|bff| = n$

Input

The first line represents the number of test cases **T**.
For each test case:
The first line contains **n** which represents the total number of children.
The next **n** lines each contain one integer, representing the current child's best friend.

Output

Output an integer representing the maximum number of children who can ride the roller coaster.

Sample Input	Parsed Input	Sample Output	Explanation
1 3 1 2 0	`superhero(3, [1, 2, 0])`	3	The children can sit in the following circle: 0-1-2 and their preferences will be satisfied.
1 6 1 2 0 0 0 0	`superhero(3, [1, 2, 0, 0, 0, 0])`	3	4 people want to sit next to child 0, but unfortunately the ride cannot take all of them. Children 0, 1 and 2 can form a circle which is the maximum number for this scenario.

11. Christian's Tropical Vacation

vacation.py, vacation.java, vacation.cpp

To escape his responsibilities, Christian has decided to travel to the Amazon Rainforest. Being an avid botanist, he decides that the beauty of a tree rooted at t (represented as $t!$) is given by:

$$t! = |s| \prod_{i=1}^{n} t_i!$$

Where t is the root of the tree, $|s|$ is the number of leaves in t and $t_i!$ is the beauty of the ith subtree. The beauty of a single vertex is 1.

While exploring the Amazon, Christian stumbles upon a magical device that allows him to combine trees. Given complete trees t and g, the device creates the tree $h = t * g$ such that h is the tree where each leaf in t is replaced by g. (We define a single vertex to be a leaf)

Furthermore, Christian notices that his surroundings is composed of n complete trees where the ith tree t_i of height h_i is defined by an array $[a_1, a_2, \ldots, a_{h_i}]$ such that $a_1 = 1$ and each vertex in the jth level of the tree ($1 \le j < h_i$) has exactly a_{j+1} children. The root of the tree has level 1. Given q queries (l, r), help Christian find the beauty of the tree created by $t_l * (t_{l+1} * (t_{l+2} * \ldots (\ldots * t_r)))$ mod 1e9 + 7.

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`vacation(n, q, trees, queries):`

- **n:** the number of trees
- **q:** the number of queries
- **trees:** a 2d array describing the trees. The ith array contains the height of the ith tree, followed by the number of children for each vertex in each level.
- **queries:** an array describing the q queries. The ith query contains $[l_i, r_i]$.

Your function should return an integer denoting the beauty of the resulting tree formed by $t_l * (t_{l+1} * (t_{l+2} * \ldots (\ldots * t_r)))$ mod $10^9 + 7$

Constraints

$1 \leq n, q \leq 100000$

$1 \leq h_i \leq 200000, 1 \leq a_j \leq 10^9$

$1 \leq l_i \leq r_i \leq n$

We guarantee that the sum over all h_i does not exceed 2×10^5

Input

The first line represents the number of trees *n* and the number of queries *q*.

The next *n* lines represent the trees where the *i*th line contains the height of the tree h_i followed $a_1, a_2, \ldots, a_{hi}$.

The next *q* lines represent the queries where the *i*th line contains the query (l_i, r_i)

Output

For each of the *q* queries, output the beauty of the resulting tree formed by $t_l * (t_{l+1} * (t_{l+2} * \ldots (\ldots * t_r)))$ mod $10^9 + 7$

Sample Input	Parsed Input	Sample Output	Explanation
3 2 1 1 3 1 6 3 2 1 4 1 1 1 3	vacation(3, 2 [[1, 1], [3, 1, 6, 3], [2, 1, 4]], [[1, 1], [1, 3]])	1 170502810	The first query just results in a single vertex of beauty 1. The second query results in a complete tree of height 4 where the root has 6 children, each vertex in the second level has 3 children, and each vertex in the third level has 4 children. The resulting value of the beauty is 6 * 3 * 4 * ((12 *4^3)^6) mod (10^9 + 7) = 170502810
6 4 2 1 3 1 1	vacation(6, 4 [[2, 1, 3], [1, 1], [6, 1, 7,	3 86135074 463522752	The first query results in a complete tree of height 2 where the root has 3

6 1 7 5 3 2 2 5 1 6 1 9 3 3 1 2 2 3 1 1 1 1 2 2 6 1 5 3 4	5, 3, 2, 2], [5, 1, 9, 1, 9, 3], [3, 1, 2, 2], [3, 1, 1, 1]], [[1, 2], [2, 6], [1, 5], [3, 4]])	818120065	children. The beauty of the tree is 3 * (1^3) = 3. The next 3 queries are left as an exercise.

12. Aaron's Terrifying Tetrahedron

`terrifyingTetra.py, terrifyingTetra.java, terrifyingTetra.cpp`

Aaron is not like most people. While venturing in France, he decides to avoid the typical attractions such as the Eiffel Tower, but rather spends his time exploring classrooms of dead French Mathematicians.

One day, he enters the classroom of Professor Gonem Sardgap. On the blackboard of the classroom is a geometric sketch which Aaron wants to learn more about.

The blackboard depicts a tetrahedron (not-necessarily regular) with six planes slicing through it. The planes are such that each plane passes through the midpoint of a distinct edge of the tetrahedron, but is also perpendicular to the opposite edge.

For example, if the tetrahedron is labeled ABCD, there is one plane passing through the midpoint of edge AB which is also perpendicular to the edge CD. Aaron sees that Professor Sardgap has written a theorem on the board that says that all these planes will coincide at one point.

Interested in this theorem, Aaron wants your help to write a program that will find the point of concurrence of these six planes given the four coordinates of the vertices of the tetrahedron.
You may assume the following:

- *No three vertices are collinear.*
- *The four vertices are not coplanar.*
- *All four vertices are distinct.*

Implementation Details

Implement the following function. It will be called by the grader once for each test case.

`terrifyingTetra(pts)`

- **`pts`**: a list of lists of the form [x, y, z], where each list corresponds to one vertex, and the first element of the list is the x coordinate, the second is the y-coordinate, and the third is the z-coordinate. We also have |pts| = 4.

Your function should return the x-, y-, and z-coordinate of the point of concurrence, rounded to the nearest tenth.

Constraints

$-100 \leq x, y, z \leq 100$
$|pts| = 4$

Input

The first line represents the number of test cases **T**.
Each test case consists of four lines:
The ith line contains three decimals **x y z** corresponding to the coordinates of the ith vertex

Output

For each test case, print three decimals rounded to the nearest tenth representing the point of concurrence of the planes.

Sample Input	Parsed Input	Sample Output	Explanation

`1`
`1 1 1`
`-1 1 1`
`1 -1 1`
`1 1 -1`
	`terrifyingTetra([[1, 1, 1], [-1, 1, 1], [1, -1, 1], [1, 1, -1]])`	`1.0 1.0 1.0`	The planes coincide at the point (1.0, 1.0, 1.0)

`1`
`1 0 -0.707416`
`-1 0 -0.707416`
`0 1 0.707416`
`0 1 -0.707416`
	`terrifyingTetra([[1, 0, -0.707416], [-1, 0, -0.707416], [0, 1, 0.707416], [0, 1, -0.707416]])`	`0.0 1.0 -0.7`	The planes coincide at the point (0.0, 1.0, -0.7)

`1`
`1 1 1`
`-1 0 1`
`1 -1 1`
`1 0 -1`
	`terrifyingTetra([[1, 1, 1], [-1, 0, 1], [1, -1, 1], [1, 0, -1]])`	`0.8 0.0 0.8`	The planes coincide at the point (0.8, 0, 0.8)