Thirty-second Annual
University of Central Florida

High School Programming Tournament

Problems

Problem Name	Filename
Pizza Packing	pizza
Order from Chaos	order
Me Llamo, Jaime	fort
New Island and Radios	radios
Alphabet-Rearrange-Inator 3000	rearrange
Increasing Photo	increasing
Lazy Bob	lazy
Heavy Lifting	heavy
Tic-Tac-Toe Plus	tic
Sum Tone	tone
Number Search	number
Grandmama's Gift Giving	gifts

Call your program file: *filename*.c, *filename*.cpp, *filename*.java, or *filename*.py

For example, if you are solving Pizza Packing:
Call your program file: pizza.c, pizza.cpp, pizza.java, or pizza.py
Call your Java class: pizza

Pizza Packing

Filename: `pizza`

Things have been heating up in Papa Pete's Pizzeria lately. Everyone loves to order Pete's world famous equilateral triangular shaped pizza for weekend get-togethers, but the pizzeria's popularity has caused them to start running out of rectangular boxes in which to deliver the pizza! It seems that some of the staff at the pizzeria simply throw each slice onto the table and haphazardly measure a rectangle box that would contain all the slices. As the boss, Pete thinks that they could optimize the orientation of the pizza slices so that much less box material is used. He has decided that he will hire you, a competent computer programmer, to find the area of the smallest possible rectangle box in which he can fit a pizza order of one or more slices. Pizza slices cannot overlap or else the toppings will get knocked off and crust will get soggy.

The Problem:

Given the number of slices and the side length of each equilateral triangular shaped slice of pizza, find the minimum area of a rectangle in which all slices can fit without overlapping.

The Input:

The first line contains a single, positive integer, *t,* representing the number of pizza delivery orders. Each order will be on a line by itself and will include an integer, n ($1 \leq n \leq 100$), representing the number of slices of pizza and an integer, s ($1 \leq s \leq 100$), representing the side length of all triangular pizza slices, respectively.

The Output:

For each order, output a line containing "`Order #`*x*`:` *y*" where *x* is the delivery order number in the order given in the input (starting from 1) and *y* is the area of the minimum rectangle which could enclose all slices of pizza without any slice overlapping another output to three decimal places and rounded to the nearest thousandth (for example, 3.1474 would round to 3.147 and 3.1475 would round to 3.148).

Sample Input:

```
2
1 5
3 2
```

Sample Output:

```
Order #1: 21.651
Order #2: 6.928
```

Order from Chaos

Filename: `order`

Your math teacher, Mr. Conway, has a bad habit of writing the problem numbers for your class' homework assignments in a random order. Additionally, sometimes he won't realize that he has written down a problem number already and will end up writing the same number multiple times! This makes the task of doing your homework much harder. If you decide to solve them in order, you need to spend several minutes sorting them before you begin. On the other hand, if you work through the problems in the order that Mr. Conway wrote them, you may end up solving the same problem twice.

To deal with this predicament, you decide to write a program that prints out the list of problems the way that a logical math teacher would write them. Since you would like the reorganized list to be as neat as possible, you set the following constraints:

- In the new list, the numbers should be printed in order from lowest to highest.
- There should be no duplicates.
- Any two or more consecutive problem numbers should be condensed into a single segment, represented by a starting number, followed by a hyphen and an ending number. For example, "7, 8, 9" would condense down to "7-9".
- The numbers should be condensed into as few segments as possible.

Different problems or ranges should be separated with a comma and a space. For example, given these rules, the problem numbers "2, 7, 1, 4, 1, 6, 8" should be condensed down to "1-2, 4, 6-8".

The Problem:

Help make your life easier by reorganizing Mr. Conway's homework assignment.

The Input:

The first line will contain a single, positive integer, d, representing the number of days of homework. For each day, there will be two lines. The first line will contain a single integer, p ($1 \leq p \leq 10^5$), representing the number of problems written by Mr. Conway on the board. The next line will contain p integers, n_i ($1 \leq n_i \leq 10^5$), representing the assigned problems.

The Output:

For each day, output a single line, starting with "`Day #s: `" where s is the day number in the input (starting with 1). On the same line, output the condensed version of the problems, following the above constraints. Separate different problems or ranges within each day with a comma and a space.

Output a blank line after each day.

Sample Input:

```
4
7
2 7 1 4 1 6 8
2
6 8
10
2 7 8 14 2 10 15 1 3 2
5
1 2 3 4 5
```

Sample Output:

```
Day #1: 1-2, 4, 6-8

Day #2: 6, 8

Day #3: 1-3, 7-8, 10, 14-15

Day #4: 1-5
```

Me Llamo, Jaime

Filename: `fort`

Steven, Jamie, and Travis are kids setting up their new fort! This fort is very – how shall we say this? – progressive, and it has top notch security! Their fort has an automated door in which a password (word or phrase) must be spoken. Unfortunately, Steven and Jamie have forgotten the password and are stuck outside the fort! Luckily, Travis has remembered that the password is "hey guys" but Steven and Jamie need your help checking passwords until Travis arrives!

The Problem:

Given a password phrase, determine if it is the correct password "hey guys" or not.

The Input:

The first line will contain a single, positive integer, *n*, representing the number of passwords to be processed. Following this, each of the next *n* lines will contain a single phrase of length at least 1 and most 80. Each phrase will contain only lowercase letters and spaces. Spaces will only appear between words (so not at the beginning or end of a line) and will consist of only single spaces when used (so multiple spaces between words will not appear).

The Output:

For each password, on its own line output either "`hey guys`" if the password is correct, or "`buzz`" if not.

Sample Input:

```
8
triceratops
tapioca
staccato
me llamo jaime
pumpernickel
hey guys
pudding
hey guys
```

Sample Output:

```
buzz
buzz
buzz
buzz
buzz
hey guys
buzz
hey guys
```

New Island and Radios

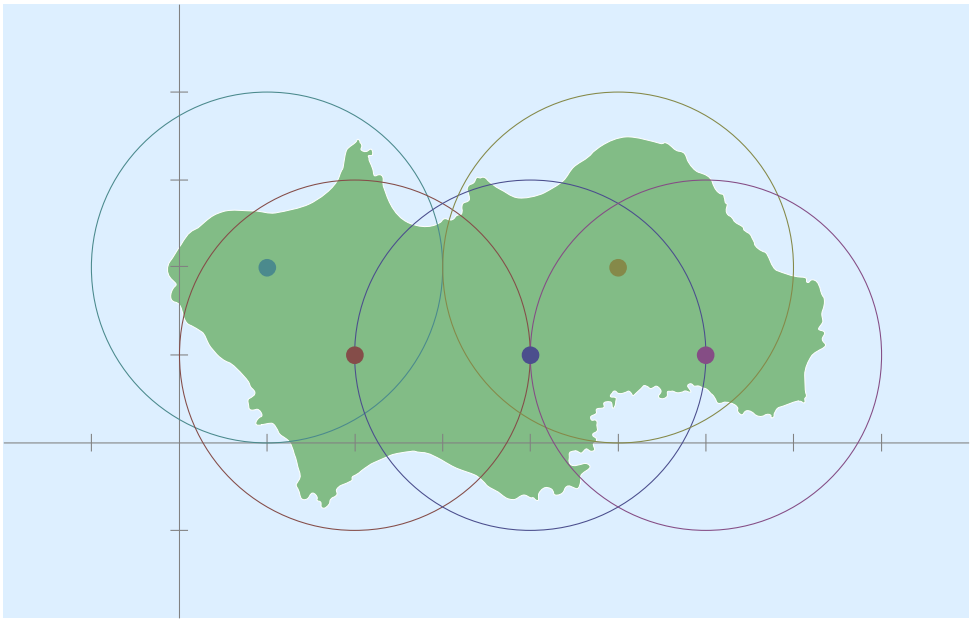
Filename: `radios`

A set of new islands were discovered in the Berland Sea today! In order to populate the islands and expand areas of living, the President of Berland wants all areas of an island to communicate by radios.

Within each island, there is a set of radio towers. Each radio tower can emit a signal that can travel up to *r* miles in radius from the tower, where *r* is an integer. Each tower will use the same *r*. Two radios towers *A* and *B* can communicate with each other if either of the following two conditions is met:

1. Tower *A*'s signals reach Tower *B* (Tower *A*'s signal is said to reach Tower *B* if Tower *B* is within or on the radius *r* from Tower *A*), or
2. Tower *A*'s signals reach the a set of towers whose signals also reach Tower *B*'s signals (for example, if there are three radio towers *A*, *B*, and *C*, and *A* can communicate with *B*, and *B* can communicate with *C*, then *A* can also communicate with *C)*.

The towers have been set up by the President, but the value for the radius, *r*, is still not set. For each island, help the president of Berland to choose the minimum integer value *r* which allows for each sector to communicate with every other sector. For example, for five towers at (1, 2), (2, 1), (4, 1), (5, 2) and (6, 1), the minimum integer value for *r* should be 2.



Note that islands do not communicate with other islands via radio (other methods are used instead).

The Problem:

For a set of radio towers on a single island, determine the minimum integer radius, *r*, required for each tower to be able to communicate with every other tower on that island.

The Input:

The first line of the input will contain a single, positive integer, *t*, representing the number of islands. Each island's input will be on multiple lines.

For each island, the first line contains a single, positive integer, *n* ($2 \leq n \leq 200$), representing the number of radio towers on that island. The next *n* lines each contain two integers separated by a single space, *x* and *y* ($-10^4 \leq x \leq 10^4$; $-10^4 \leq y \leq 10^4$), representing the x- and y-coordinates for the radio tower, respectively. You are guaranteed that multiple towers within an island will not be located at the same position.

The Output:

For each island, output "`Island #`*`i`*`: `*`r`*" on a new line where *i* is the island number (starting with 1) and *r* is the integer value representing the minimal radius value for the i^{th} island to have all of its radio towers communicate with each other.

Sample Input:

```
2
3
0 0
1 1
2 2
5
-3 -9
3 4
8 28
32 -2
-55 91
```

Sample Output:

```
Island #1: 2
Island #2: 90
```

Alphabet-Rearrange-Inator 3000

Filename: `rearrange`

Thank goodness you're here, agents! We were just about to start the annual agency gift exchange when we learned of Dr. Hunts Mittleshmirtz's latest plot. He has created a new device capable of changing the order of letters in the alphabet! Agent B is already off to go stop him, but in the meantime we'll need your help here. You see, for the gift exchange I had written codes on every present and was going to give my first friend the lexicographically first present, the second the second, and so on.

For example, take presents with the following codes: apple, book, and candle. Normally, I would give the presents in that order: the present with code "apple" first, followed by the present with code "book" and then the present with code "candle" last. However, if Dr. Hunts Mittleshmirtz were to change the order of letters "a" and "c", then it would cause me to give the "candle" present first, then the "book" present, and the "apple" present last!

Yet it gets more complex. If Dr. Hunts Mittleshmirtz were then to change the current ordering of letters "a" and "b", then it would cause me to give the "candle" present first, then the "apple" present, and the "book" present last. With the orders of letters changing all the time, I'll need your help to figure out which present is what number.

Good luck, agents.

The Problem:

Determine the lexicographically k^{th} present, while responding to a changing alphabet ordering.

The Input:

Input begins with a single, positive integer, *t*, representing the number of gift exchanges. Each gift exchange will be described on multiple lines.

The first line of each gift exchange will contain with two integers, *n* and *q* ($1 \le n \le 10^5$; $1 \le q \le 10^5$), representing the number of presents and number of queries, respectively. This will be followed by *n* lines each consisting of a single string, *s*, each representing a single present's label. Each label will consist of only lowercase letters. There will be no more than 10^5 total letters across all *n* strings, and the maximum length of each individual string will be 50 letters. In addition, each label within a single gift exchange will be unique.

This will be followed by *q* lines representing a query. Queries come in two forms, denoted by either a 1 or a 2 as the first number in the line. Type 1 queries represent the Alphabet-Rearrange-Inator 3000 firing, and will be in the form "`1 x y`" where *x* and *y* are distinct (*x* and *y* will not be the same) and represent single lowercase letters that are being swapped in the alphabet.

Type 2 queries are a request for the k^{th} present given the current alphabet, and will come in the form "`2 k`" where $1 \le k \le n$. For example, if *k* is 1, you should return what present should currently be given first (given the ordering at the time of the query); similarly, if *k* is 4, you should return the fourth present. Note these are only queries and the gifts are not actually given!

The Output:

For each gift exchange, output on its own line "`Gift Exchange #`*`i`*`:`" where `i` is the number of the query being answered in the input (starting at 1). Following this, the result of each type 2 query should be output, each on their own lines. Output a blank line after each gift exchange.

Sample Input:

```
2
3 3
a
c
b
2 1
2 2
2 3
4 3
ucf
platypus
knights
chargeon
2 1
1 a k
2 1
```

Sample Output:

```
Gift Exchange #1:
a
b
c

Gift Exchange #2:
chargeon
knights
```

Increasing Photo

Filename: `increasing`

Ali loves taking group photos! He always finds a good opportunity to take a group photo. Ali has two rules for his group photos. Everyone, from left to right, must be in alphabetical order by name, and their heights must be strictly increasing.

Ali also teaches several classes at UCF, and he begins to wonder how many different subsets of students he can take a photo of in each class. Specifically, he wants to know for each class, how many ways he can take a group photo of exactly k students such that from left to right, they are in alphabetical order by their names and their heights are strictly increasing. This is where you come in. Ali gives you a list of the heights of each student in each class. The list has already been sorted alphabetically, so Ali doesn't bother to list the names of each student.

The Problem:

Given the heights of students in each class that Ali teaches, find the number of ways that Ali can take a group photo of exactly k students, where their names are in alphabetical order, and their heights are strictly increasing.

The Input:

The input begins with a single, positive integer, c, representing the number of classes Ali is teaching. For each class, the following information is given:

First is a line containing two integers, n and k ($1 \leq n \leq 10^5$; $1 \leq k \leq 10$), representing the number of students in that class and the number of students that he wants in each group photo, respectively.

This is followed by a line containing n integers, h_i ($1 \leq h_i \leq 10^9$), where the i^{th} number on this line is the height of the i^{th} student. The heights have been listed in such a way that the names of each student are already sorted alphabetically. All heights are unique, meaning no two students will have the same height.

The Output:

For each class, output a line containing "`Class #`*`i`*`: `*`x`*" where i is the class number in the order they are given in the input (starting from 1) and x is the number of ways to take a photo of k students such that it follows Ali's rule. Since this quantity can get quite large, output x modulo 1,000,000,007 instead. Modulo can be defined as the remainder after the division of two numbers, and the operator symbol used for modulo is "%" (example: 7 % 3 = 1).

Sample Input:

```
2
5 3
1 2 3 4 5
10 2
10 5 9 2 4 6 8 3 1 7
```

Sample Output:

```
Class #1: 10
Class #2: 16
```

Lazy Bob

Filename: `lazy`

Lazy Bob wants to travel from point 1 to point *n*. These points are connected by some roads, so there are many paths that he can take. A path is a series of roads that Bob can take from one point to another. Two paths are different if the roads are different or taken in a different order. The length of a path is the sum of lengths of all the roads which comprise it.

Lazy Bob has a particularly lazy way of choosing what path to travel. For any pair of paths, if a path is *k* times as short as the other, he is *k* times as likely to take that path instead of the other. Therefore it can be seen that he is always most likely to take the shortest path. If there are multiple paths of the same length, he is equally likely to take any of them. Bob does not consider any paths that do not start at point 1 (his current position), or do not end at point *n*. Additionally, Bob never includes the same road twice in a path.

The city council wants to make it easier to get lazy people out of their homes, so they are planning to build a new express highway from point 1 to point *n* of length *m*. In order to avoid bad traffic, the only people allowed to take this new highway are people who have not yet left point 1 (the council determined that this is a fool-proof way to prevent the non-lazy people from taking the highway). Recall that due to beauraucratic oversight, the express highway may be longer than the existing shortest path from point 1 to point *n*. The city council is interested to know the probability that Lazy Bob will use their highway. Keep in mind that when Bob reaches point *n*, he stops.

The Problem:

Given the current roads in a city, what is the probability Lazy Bob will use the new path?

The Input:

Lazy Bob will travel to multiple cities, each building such a highway. Therefore, the first line will contain a single, positive integer, *t*, representing the number of cities through which Lazy Bob will travel. Each city will be described on multiple lines.

The first line for each city will contain three integers, *n* ($2 \le n \le 10$), *m* ($1 \le m \le 1{,}000{,}000$), and *r* ($1 \le r \le 10$), representing the number of points in the city, the length of the new highway in the city, and the current number of roads in the city, respectively.

The following *r* lines each will contain three integers, *u* ($1 \le u \le n$), *v* ($1 \le v \le n$) and *w* ($1 \le w \le 1{,}000{,}000$), representing a road connecting point *u* to point *v* with distance *w*, respectively. Each road is one-directional, meaning Bob can take it from *u* to *v* but not from *v* to *u*.

The Output:

For each city, output "`City #i: p%`" where *i* is the number of the city in the input (starting with 1) and *p* is the probability that Bob takes the new path output to three decimal places and rounded to the nearest thousandth. For example, 0.0034 will round to 0.003, and 0.0035 will round to 0.004.

Sample Input:

```
3
5 50 4
1 2 25
2 3 25
3 4 25
4 5 25
3 10 4
1 2 3
2 1 3
1 2 6
2 1 6
7 50 8
1 2 40
1 3 30
1 4 40
2 3 40
3 7 20
4 5 10
5 6 10
5 7 100
```

Sample Output:

```
City #1: 66.667%
City #2: 100.000%
City #3: 35.294%
```

Heavy Lifting

Filename: `heavy`

In the middle of Alabama Bones' expedition into the cursed temple of the Forbidden One, he was confused by a particularly tricky puzzle. As he entered the main crypt, he realized all the treasure was placed on an elevated platform hanging from the ceiling h units high. Since he forgot his ladder, he had to improvise with what little materials he found in the room.

He is presented with a large 8 by 8 square grid of stone slabs on the floor of the crypt. The cell in the i-th row and j-th column contained a stack of slabs, $a_{(i,j)}$, each a unit tall. A cell x is adjacent to another cell y if x shares an edge or corner with cell y. Unfortunately, Alabama's sidekick can't climb so he can only move onto piles that are 1 unit higher than his current elevation. Since Alabama is quite nimble and can climb up any tower of slabs, he needs to remove slabs from the platform in such a way that there exists a staircase path of adjacent slabs through the grid that increases in height one unit per cell until finally reaching the treasure platform, at a height of h. Alabama would like to build this staircase of stone slabs all at once so that when his sidekick starts walking on them, he never has to move a slab again.

Doing all this moving of large stone slabs can be very draining to Mr. Bones' stamina, so he would like to know the minimum amount of stamina he must expend for the duo to reach the top. Since the slabs are quite heavy, Alabama can only pick up a stone slab from the top of a pile and throw it off the grid and for a cost of 1 stamina. When he throws a stone, it shatters on the ground making it impossible to use again. The crypt makers were fair enough to ensure that there will always be a way for Alabama to create a staircase path to the top.

The Problem:

Given the heights of the stacks of stones slabs in an 8 by 8 grid, determine the minimum stamina required to create an increasing height path which goes from 0 to h inclusive.

The Input:

The first line contains a single positive integer, t, representing the number of crypt puzzles to solve. The next line of each puzzle is an integer, h ($0 \leq h \leq 10$), representing the height of the tomb from the problem described above. The next 8 lines contain 8 non-negative integers, the j-th integer of the i-th line representing $a_{i,j}$ ($0 \leq a_{i,j} \leq 100$). It is guaranteed that there exists some way to orient the slabs in such a way that there is path to the top.

The Output:

For each crypt, output a line containing "`Crypt #`x`:` y" where x is the crypt puzzle in the order given in the input (starting from 1), and y is the minimum amount of stamina required by Alabama Bones to move the slabs into an appropriate path for his sidekick.

Sample Input:

```
2
5
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 3 0 0 0 0
0 0 0 0 3 0 0 0
0 0 0 9 4 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
10
0 0 0 0 10 0 0 0
0 0 0 0 9 0 0 0
0 0 2 1 8 0 0 0
0 0 3 0 7 0 0 0
0 0 4 5 6 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Sample Output:

```
Crypt #1: 5
Crypt #2: 0
```

Tic-Tac-Toe Plus

Filename: `tic`

Tired of the monotony of the simple game of Tic-Tac-Toe, Alex and Barb have come up with a slightly more challenging variation of the game, which they call Tic-Tac-Toe Plus. The rules of the game are as follows:

- Games are played on a 4x4 grid.
- Each cell has an integer point value, which is determined before the game.
- Players take turns selecting an empty cell and drawing an X in it (there are no O's).
- Each time a player places an X in a cell, he/she earns the number of points associated with that cell. The number of points can be negative, meaning that player loses points.
- A bonus of p points is given to the first player to make 3-in-a-row. This means that before the player made a move, the board did not contain 3 adjacent X's (horizontally, vertically, or diagonally), but after the player made a move, the board *did* contain 3 adjacent X's.
- Similarly, a bonus of q points is given to the first player to make 4-in-a-row.
- It is possible by this definition to make the first 3-in-a-row and 4-in-a-row in the same move. In this case, both bonuses are applied at once.
- The game ends when the grid is completely full (all games are 16 moves long).
- The winner is the player with the most points. There can be a draw if the players end with the same score.

Every time they play their game, Alex insists on going first, since his name comes first alphabetically. Thinking this is unfair, Barb decides to write a program to determine if Alex has an unfair advantage.

The Problem:

Given the point values for each cell, as well as p and q, determine who will win the game if both players play optimally (meaning that if a player can force a win, he/she will) starting with an empty board. Alex will always go first.

The Input:

The first line will contain a single, positive integer, t, representing the number of Tic-Tac-Toe Plus games to consider.

For each game, there will be five lines. The first line will contain two integers, p ($0 \le p \le 100$) and q ($0 \le q \le 100$), representing the number of bonus points given to the first player to make 3-in-a-row and 4-in-a-row, respectively. The second through fifth lines each contain four integers, $v_{i,j}$ ($-100 \le v_{i,j} \le 100$). These 16 numbers represent the point values associated with each cell in the grid.

The Output:

For each game, output a single line, starting with the prefix "`Game #`*g*`: `" where *g* is the current game number (starting with 1). If one player will beat the other player when they both play optimally, output the winner's name (either "`Alex`" or "`Barb`") followed by "` wins.`" Otherwise, output "`It's a draw.`"

Sample Input:

```
4
0 0
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
2 3
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0
1 1 2 2
3 3 4 4
5 5 6 6
7 7 8 8
2 15
-2 -1 -2 -4
3 1 -3 1
1 0 0 4
-1 2 4 -3
```

Sample Output:

```
Game #1: Alex wins.
Game #2: Barb wins.
Game #3: It's a draw.
Game #4: Barb wins.
```

Sum Tones

Filename: `tone`

Haley is a musician and composer who has become increasingly interested in creating mathematically intriguing music. Since music is simply sound waves played at different frequencies, she has translated all her sheet music into numbers representing each note's pitch. Pitch is a measure of the rate of vibrations caused by sound in hertz (Hz). She was recently introduced to the concept of a *combination tone*, which is a term describing the phenomenon of an additional tone that is artificially perceived when two real tones are played concurrently. Haley has been hard at work creating songs that sound particularly beautiful with a specific type of combination tone known as a *sum tone*, where two tones' pitches are added together to create an artificial third tone. An example of this could be heard if a note with frequency 3 Hz is played concurrently with a note of 5 Hz to make a *sum tone* of 8 Hz. Her songs consist of n notes represented as integers in a list where each adjacent pair of notes is played together to form a new sum tone.

To make her new songs more mathematically interesting, Haley has decided to take one of her old pieces and remove some of the notes so that the pitch of all sum tones are powers of two. She would like some way to do this and keep the song as long as possible. Consider the song represented by the notes [2, 3, 2, 5]. She could remove the 3 and 5 and have the song [2, 2] where the adjacent notes sum to $4 = 2^2$. Alternatively, she could instead remove both 2s and be left with the song [3, 5] where the adjacent notes sum to $8 = 2^3$. In either case she can only make a song of at most length 2 while maintaining that all sum tones are powers of two. Since Haley is a much better musician than she is a problem solver, she's asked you to solve the problem for her!

The Problem:

Given a song as a list of notes represented by integer frequencies, find the length of the longest non-empty subsequence such that all adjacent pairs of frequencies sum to powers of two.

The Input:

The first line contains a single, positive integer, t, representing the number of songs. The first line for each song will be an integer, n ($1 \leq n \leq 10^5$), representing the number of notes in the original song. The next line will contain n integers, $a_1, a_2, \ldots a_n$ ($1 \leq a_i \leq 10^{18}$), representing the frequencies of the notes.

The Output:

For each song, output a line containing "`Song #`x`:` y" where x is the song number in the order given in the input (starting from 1), and y is the length of the maximum nonempty subsequence that satisfies Haley's requirements.

Sample Input:

```
3
4
2 3 2 5
5
5 5 5 5 5
7
9 23 9 7 1 3 125
```

Sample Output:

```
Song #1: 2
Song #2: 1
Song #3: 7
```

Number Search

Filename: `number`

Lately, Sally has been interested in a puzzle called a number search. A number search is similar to a word search, except that in a number search, a number can also be found by applying the addition (+), subtraction (-), multiplication (*), or division (/) operation. In order for a number to be found in a number search, it must be in one of the following forms:

- **n**, where n is a positive integer.
- **a+b**, where a and b are positive integers.
- **a-b**, where a and b are positive integers.
- **a*b**, where a and b are positive integers.
- **a/b**, where a and b are positive integers and a/b is also a positive integer (a is evenly divisible by b).

Only zero or one operations may be used to calculate a number. A number can be found in any of the 8 directions (horizontally from left to right or right to left, either direction vertically, or any direction diagonally).

```
85-71
7/2-1
552*5
```

For example, consider the number search shown above. Some numbers that can be found include 85 (top left, using no operations), 78 (`85-7`), 10 (`2*5`), 4 (`8/2`, diagonally), and 578 (vertically on the left side). Note that -7 cannot be found because -7 is not a positive integer. However, -2 can be found from 5-7. For simplicity, number search puzzles do not contain any 0's.

On several puzzles, Sally has had trouble finding certain numbers in the number search, and she has begun to doubt that they can be found at all. Sally needs your help!

The Problem:

Help Sally out by telling her whether certain numbers can be found in the given puzzles.

The Input:

The first line will contain a single, positive integer, *p*, representing the number of puzzles.

The input for each puzzle will consist of several lines. The first line contains two integers, *r* ($1 \le r \le 15$) and *c* ($1 \le c \le 15$), representing the number of rows and columns in the puzzle, respectively. The following *r* lines each contain *c* characters, each of which will either be a digit between 1 and 9, or one of the operators: '+', '-', '*', or '/'.

The next line contains a single integer, *q* ($1 \le q \le 10^3$), representing the number of queries Sally would like to answer. Each of the next *q* lines contains an integer, *m* ($-10^{15} \le m \le 10^{15}$), representing a number to check for in the puzzle.

The Output:

For each puzzle, output the following line: "`Number Search #x:`" where *x* is the puzzle number in the input (starting from 1). Then, for each query, output on a separate line either "`Yes`" if the number can be found, or "`No`" if it cannot. Output a blank line after each puzzle.

Sample Input:

```
2
3 5
85-71
7/2-1
552*5
7
85
78
10
4
578
-7
-2
3 6
1*89-2
+5-371
88*2-2
5
87
40
972
23
-9
```

Sample Output:

```
Number Search #1:
Yes
Yes
Yes
Yes
Yes
No
Yes

Number Search #2:
Yes
No
Yes
Yes
No
```

Grandmama's Gift Giving

Filename: `gifts`

Tyler is excited to visit his grandmother (who he calls Grandmama) this Friday for a family get-together! You see, he and the other grandchild (his cousin) each get a gift from Grandmama when they have a family visit. However, Grandmama doesn't want to give the same gift to both grandchildren, and also does not want to be unfair. Therefore, she wants the difference in price between the two gifts to be as small as possible.

The Problem:

Help Grandmama out by determining the pair of gifts (from a set of three) that provide the smallest difference.

The Input:

The first line will contain a single, positive integer, *v*, representing the number of visits with Grandmama. The input for each visit will consist of a single line containing three integers, *a* ($1 \leq a \leq 50$), *b* ($1 \leq b \leq 50$) and *c* ($1 \leq c \leq 50$), representing the price of the three potential gifts (in dollars), respectively.

The Output:

For each visit, output on a new line the smallest price difference of choosing two gifts from the potential three gifts.

Sample Input:

```
3
7 3 6
1 2 3
10 15 22
```

Sample Output:

```
1
1
5
```