An Odd Sum[1] (prob1)

The Problem

Some numbers such as 40 can be written as the sum of consecutive integers. For example,

$$40 = 6 + 7 + 8 + 9 + 10.$$

Shiflett and Shultz [1] defined *odd-summing* natural numbers as “natural numbers that are the sum of two or more consecutive [positive] odd numbers.” Perfect squares (larger than one) are always odd-summable, while primes will not be. If a number is odd-summable, there may be more than one way to express it in that manner. For example,

$$40 = 19 + 21 = 7 + 9 + 11 + 13$$

For any odd summable number n, you are to enumerate all the sets of consecutive odd positive integers that sum to the given number n.

Input

The first line of input will be the number of problems. The remaining lines will contain one number n, $1 \le n \le 10^9$ per line.

Output

For each problem, output one line for each solution for a number in the input set. On the line, first have the original number, followed by a colon and a space. Then have the set of consecutive odd integers whose sum is n. Output the endpoints of each set as shown in the sample output.

When multiple solutions exist, output them sorted by the first term. If no solution exists for a given n, output a single line with “impossible” instead of the set.

Sample Input

```
3
7
35
400
```

Sample Output (corresponding to sample input)

```
7: impossible
35: [3, 11]
400: [1, 39]
400: [31, 49]
400: [43, 57]
400: [97, 103]
400: [199, 201]
```

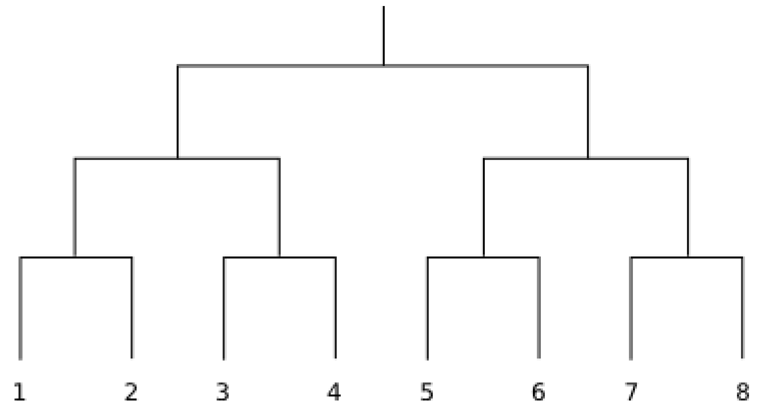
[1]Adapted from *An Odd Sum* by Ray C. Shiflett and Harris S. Shultz and published in The Mathematics Teacher, Vol. 95, No. 3 (March 2002), pp. 206-209

Best of Seven (prob2)

It is almost March Madness! Getting excited for another year of college basketball your college has decided to host its own intramural basketball tournament for students. Another fun aspect of college basketball is trying to predict the outcome of tournament brackets. Your college has decided that it would actually be more interesting to have students try to predict the win/loss records of teams in the tournament.

Unlike normal college basketball (and more like professional basketball) each matchup in the tournament will have a series of games instead of a single game to determine who moves on to the next round. Each team that is paired in a matchup must compete against the other team until it is no longer possible for one team to come back and win. The matchups are best of seven so once a team reaches four wins they move on to the next round and the matchup ends. The rules of the tournament prevent ties. Once a team is eliminated they are kicked out and play no more games. In other words the tournament is single elimination.

The tournament bracket is a full binary tree where each leaf represents a team. The internal nodes of the tree represent a matchup between the winner of the left bracket and the winner of the right.



In order to keep predictions fair the tournament committee needs a program that can take a win loss record prediction for the teams and determine if it is even possible to achieve.

Input

The first line contains a single integer *t* representing the number of tournaments predictions to examine. For each tournament, the first line will be a single integer *n* representing the number of teams in the tournament, where $2 \leq n \leq 512$ and *n* is a power of two. The next *n* lines contains two integers *v and d* representing the number of wins and losses predicted for team *i*, where $0 \leq v, d \leq 50$. The teams are given in order as they would appear left to right on the bottom of a tournament bracket.

Output

For each tournament prediction output the following header, "Tournament #*i*:" where *i* is the current tournament prediction being processed starting from 1. Follow this by a single space and the word "Possible" without quotes if the win/loss record can be achieved in some way and "Impossible" without quotes otherwise. Print a blank line after each tournament prediction.

Sample Input

```
3
2
4 3
3 4
4
10 10
4 8
8 4
3 3
4
8 0
0 4
4 4
0 4
```

Sample Output (corresponding to sample input)

```
Tournament #1: Possible

Tournament #2: Impossible

Tournament #3: Possible
```

Block Party (prob3)

There are a variety of games such as Bejeweled and Shape Shift that are played on a grid of tiles, each having a color and sometimes another image. A move is a swap of two tiles that have the same shape on them. When a swap results in a chain 4 or more of the same color, then those tiles are removed from the board and the tiles above them slide down to fill the gaps. (Two tiles are in a chain if they share a side.) After tiles have slid down, more chains might form. Note: No tiles slide until all chains in the current configuration of the board have been removed.

A board can be represented by a rectangular arrangement of letters. One such arrangement is shown in the figure below. On that board, there are two chains of length 4, one of the character "R" and one of the character "B". Note that to be a chain, the characters must be an exact match; case is significant.

The start of a sequence of reactions is shown in the figure. Reactions continue until there are no more chains. In real games, new tiles replace open spaces in the grid, but we will not be concerned with replacement tiles.

Write a program that processes the sequence of chain reactions in a board.

2 chains identified	Chains removed	Slide completed	Another chain…
YOOGYBY	YOOGYBY	YOO Y	YOO Y
GRGYBbB	GRGYBbB	GRGG B	GRGG B
OGRGOBY	OGRGOBY	OGRY Y	OGRY Y
BBGB**RR**B	BBGB B	BBGG BB	BBGG BB
YRY**RR**OR	YRY OR	YRYBYbR	YRY**B**YbR
BOYBO**B**O	BOYBO O	BOYBBBO	BOY**BBB**O
YGBY**BB**G	YGBY G	YGBYOOG	YGBYOOG
OOBY**B**RG	OOBY RG	OOBYORG	OOBYORG
BBYRRBO	BBYRRBO	BBYRRBO	BBYRRBO

Input

The input is a sequence of test cases, each representing a board. The first line of each test case contains two nonnegative integer values *w* ($0 < w < 256$) and *h* ($0 < h < 256$) separated by a space giving the width and height of the board (in tiles). The line containing the dimensions is followed by *h* lines of *w* non-blank characters each.

The end of input is indicated by a line containing 0 0 for *w* and *h*. This case should not be processed.

Output

The output for each test case is the number of the test case (where the first test case is numbered 1) followed by a colon and a space followed by an integer value that shows the number of tiles remaining after all reactions complete.

Sample input

```
7 9
YOOGYBY
GRGYBbB
OGRGOBY
BBGBRRB
YRYRROR
BOYBOBO
YGBYBBG
OOBYBRG
BBYRRBO
3 2
YBY
BYB
0 0
```

Sample output (corresponding to sample input)

```
1: 51
2: 6
```

Canal Navigation (prob4)

A canal connects two bodies of water. Sometimes the height of the two bodies of water that are connected are different and a series of locks are necessary to construct the canal. In this problem you will have to write a program that allows a boat to travel from one cell of a lock system in a canal to another cell of a lock system in minimal time.

To simplify the problem, assume that the locks create a rectangular arrangement of cells and each cell has the shape of a square with a lock on each of the four sides. Each cell has an elevation of water. We will assume that the boat in question always starts from the northwest corner of the lock system and needs to travel to the south east corner. In each "move", the boat may choose to go south or east, unless the cell in which it's in is southernmost or easternmost. The amount of time in minutes a move between cells takes is simply the difference in elevation in feet of the starting cell before and after the move. After the move, both cells will have the same elevation and then the corresponding lock is put back in place.

Consider the following lock system with six cells:

3	9	5
1	4	5

On our first move, if we move south, we would temporarily open the southern lock of the northwest square. This would make the water levels in both of the westernmost cells equal to 2, taking 1 minute, since the elevation of the first cell changes from 3 to 2 feet. Thus, our ensuing picture is as follows:

2	9	5
2	4	5

From here, if we move east twice, we will arrive at the desired destination in 3 minutes total, since each of the last two moves take one minute as well. Here is a complete look at each move:

3	9	5		2	9	5		2	9	5		2	9	5
1	4	5	→	2	4	5	→	3	3	5	→	3	4	4
					Cost = 1				Cost = 1				Cost = 1	

This sequence of moves leads to the fastest path to move from the northwest corner to the southeast corner. The other two possible paths would take 3.75 minutes and 4 minutes. These are shown below.

3	9	5		6	6	5		6	5.5	5.5		2	5.5	5.25
1	4	5	→	1	4	5	→	1	4	5	→	3	4	5.25
					Cost = 3				Cost = 0.5				Cost = 0.25	

3	9	5		6	6	5		6	5	5		6	5	5
1	4	5	→	1	4	5	→	1	5	5	→	1	5	5
					Cost = 3				Cost = 1				Cost = 0	

Input

The first line of the input file will contain a single positive integer, *n*, representing the number of lock systems to navigate. Data for each of the *n* lock systems follows.

The first line of data for each lock system will contain two positive integers, *r* ($r \leq 10$) and *c* ($c \leq 10$), representing the number of rows and columns in the grid system, respectively, separated by spaces. (The first row represents the north row of the grid and the first column represents the west column of the grid.) The following *r* lines contain *c* positive integers less than or equal to 1000, each separated by spaces, representing the elevation of each of the cells on the given row, from west to east.

Output

For each lock system, output a single line with the following format

```
Canal k: X
```

where k is the number of the lock system, starting with 1, and X is the number of minutes representing the minimum time to move in the lock system from the northwest corner to the southeast corner with south and east moves only, rounded to three decimal places.

Sample Input

```
2
2 3
3 9 5
1 4 5
2 2
1 11
2 3
```

Sample Output (corresponding to sample input)

```
Canal 1: 3.000
Canal 2: 1.250
```

Prob5: Bookshelf Building

Bookshelf by Free-Photos on Pixabay

Anna just came home from her favourite furniture store *Ikey Yeah!* where she bought a new bookshelf designed by her favourite artist *Bill Lee*. The shelf has a rectangular shape with a width of x and a height of y. Included in the box of the shelf is one board that Anna may put horizontally inside the shelf. For this purpose the frame of the shelf already contains pre-drilled holes at heights $1, 2, \dots, y-1$. Anna may attach the board to the frame at any of these heights. She can also decide not to install it at all.

Anna owns n books, which all have the same depth. The books' depth matches that of the shelf exactly. However, due to differences in the formats and the numbers of pages, her books may have different widths and heights. Anna does not want to flip her books or stack several of them on top of each other. Instead, she wants to store all of them in an upright fashion inside her bookshelf, such that the width and height of each book are aligned with the width and height of the bookshelf. Help Anna to find the perfect position for the board (or tell her not to use it at all) so that she can store all her books in her new bookshelf.

For this problem you may assume that the frame and the board of the shelf are infinitely thin.

Input

The input consists of:

- One line with three integers n, x and y, where
  - n ($1 \leq n \leq 10^4$) is the number of books that Anna owns;
  - x ($1 \leq x \leq 10^4$) is the width of the bookshelf frame;
  - y ($1 \leq y \leq 10^4$) is the height of the bookshelf frame.
- n lines, the ith of which contains two integers w_i and h_i, where
  - w_i ($1 \leq w_i \leq 10^4$) is the width of the ith book;
  - h_i ($1 \leq h_i \leq 10^4$) is the height of the ith book.

Output

If there is no possibility to store Anna's books in the shelf, print out `impossible`. In case Anna decides not to install the board, output -1. Otherwise, output the height at which the board should be installed. If there are multiple possible heights, you may output any one of them.

Sample Input 1	Sample Output 1

4 4 4
3 1
1 1
1 3
2 2
	3

Sample Input 2

```
2 4 3
2 3
1 1
```

Sample Output 2

```
-1
```

Sample Input 3

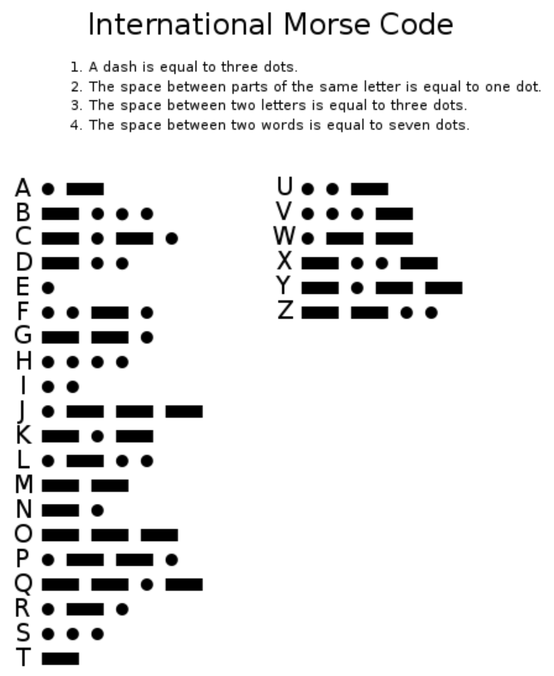
```
3 3 3
2 2
2 1
2 1
```

Sample Output 3

```
impossible
```

Morse Enumeration (prob6)

Morse code was a very widespread communication format before the use of telephones and computers. In fact, Morse code is still in widespread use simply because of its simplicity and clarity on noisy signals. However, one problem can arise with this method of communication: exact meaning. Consider the table below (retrieved from `http://en.wikipedia.org/wiki/Morse_code`):



International Morse Code

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬		
L	● ▬ ● ●		
M	▬ ▬		
N	▬ ●		
O	▬ ▬ ▬		
P	● ▬ ▬ ●		
Q	▬ ▬ ● ▬		
R	● ▬ ●		
S	● ● ●		
T	▬		

Notice that Morse code specifies spacing between letters, and a different spacing between words. If those spaces were lost, it would be impossible to determine the correct meaning of the message without examining all possible transmitted messages. For instance, consider the string "`.-`". The two characters together could encode the single letter "`A`" or the string "`ET`". As strings grow longer, the number of potential meanings grows as well. For instance, the string "`.-...`" has 15 possible interpretations: "`AEEE`", "`AEI`", "`AIE`", "`AS`", "`EB`", "`EDE`", "`ENEE`", "`ENI`", "`ETEEE`", "`ETEI`", "`ETIE`", "`ETS`", "`LE`", "`REE`", and "`RI`".

You job is, given a message in International Morse Code, determine the number of strings of letters it could represent.

Input

The input will consist of a number of lines with length no greater than 30 characters. The end of input will consist of line with a "#" at the beginning.

Output

For each test case, print the number of possible messages that can encoded with the given series of dashes and dots on a separate line.

Sample Input

```
.-
....
.-...
#
```

Sample Output (corresponding to sample input)

```
2
8
15
```

Prob7: Flip Flow

Hourglass by nile on Pixabay

For over 600 years, the hourglass has been a well-known timekeeping instrument. An hourglass consists of two glass flasks arranged one above the other which are connected by a narrow channel. Inside the hourglass there is sand which slowly flows from the upper to the lower flask. Hourglasses are typically symmetrical so that the sand always takes the same amount of time to run through, regardless of orientation. For the purposes of this problem, we also assume that the flow rate of the sand is a known constant and does not depend on the amount or distribution of sand in the upper half.

Your friend Helen was bored and has been playing around with her hourglass. At time 0, all the sand was in the lower half. Helen flipped the hourglass over several times and recorded all the moments at which she did so. How many seconds does she need to wait from the current time until all the sand is back in the lower half?

Input

The input consists of:

- One line with three integers t, s and n, where
  - t ($1 \leq t \leq 10^6$) is the current time;
  - s ($1 \leq s \leq 10^6$) is the amount of sand in the hourglass, in grams;
  - n ($1 \leq n \leq 1\,000$) is the number of times the hourglass was flipped.
- One line with n integers $a_1, \ldots, a_n$ ($0 < a_1 < \cdots < a_n < t$), the times at which the hourglass was flipped.

All times are given in seconds. You may assume that the sand flows from the top to the bottom at a constant rate of 1 gram per second.

Output

Output the time in seconds needed for the hourglass to run out starting from time t.

Sample Input 1

```
10 7 2
4 9
```

Sample Output 1

```
4
```

Sample Input 2

```
2000 333 3
1000 1250 1500
```

Sample Output 2

```
0
```

Sample Input 3

```
100 10 5
15 20 93 96 97
```

Sample Output 3

```
5
```

This page is intentionally left (almost) blank.

Promoting Elves' Rights (prob8)

You'd think that elves get some time off after Christmas, but in fact, they stay busy in the spring dying Easter eggs and filling baskets. Hermione wants to help the elves with their work in order for them to enjoy more free time and the rights they deserve. Thus she has founded S.P.E.W. (Society for the Promotion of Elfish Welfare). Besides magic, she needs analytical thinking to organize the kitchen work the best way possible. The good news for her is that the Muggles have developed computers with programming capabilities.

The work she has to schedule takes a specific amount of time measured in minutes for each job that needs to be completed. The elves need to finish the maximum amount of chores within a certain deadline and of course, they want to take as little time as possible in doing so. For example they may have 20 chicken eggs that take 3 minutes an egg to dye, 10 quail eggs that take 1 minute an egg to dye, and 14 ostrich eggs, that take 10 minutes an egg to dye. If they have 20 minutes to dye eggs, they'll dye 10 quail eggs and 3 chicken eggs, leaving a minute break. Hermione will always choose the schedule that gets the most jobs done in the least amount of time.

Input

The input will contain one or more input sets. The end of input will be indicated by a line with just 0 on the line.

Each line of an input set will begin with an integer *d*, $0 < d < 500$, representing the maximum amount of time the elves have to work. There will then be an integer *j*, $0 < j < 11$, representing the number of types of jobs available to the elves. The line will then have *j* types of jobs the elves have to complete within the deadline, formatted as a pair of numbers (*n*, *t*). The first number, *n*, indicates the number of this type of job, where $0 < n < 1{,}000$. The second number, *t*, indicates the time it takes to complete each of this type of job in minutes, where $0 < t < 100$.

Output

The output for each input set should be the maximum number of jobs the elves can complete within the deadline and the time needed to complete these jobs. If there is more than one collection of jobs with the maximum count, the one with the shortest total time should be used.

Sample Input

```
20 3 (10, 1) (5, 5) (8, 2)
50 2 (12, 4) (42, 3)
12 4 (1, 4) (3, 5) (1, 3) (1, 1)
47 4 (1, 10) (3, 5) (2, 7) (3, 3)
33 2 (100, 5) (200, 2)
0
```

Sample Output

```
15 20
16 48
3 8
8 38
16 32
```

Proper Subsets (prob9)

A set is a collection of elements, or members. Order and count do not matter, so the set {1, 2, 3} is the same as the sets {3, 2, 1} and {1, 2, 1, 3, 3, 1, 2}.

If A and B are sets and every element of A is also an element of B, then A is a subset of (or is included in) B, denoted by $A \subseteq B$. For example $\{1, 2, 3\} \subseteq \{4, 3, 2, 1\}$ and $\{1, 2, 3\} \subseteq \{3, 2, 1\}$.

If A is a subset of B, but A is not equal to B (i.e. there exists least one element of B not contained in A), then A is also a proper subset of B, denoted by $A \subset B$. For example
$\{1, 2, 3\} \subset \{4, 3, 2, 1\}$ but $\{1, 2, 3\} \not\subset \{3, 2, 1\}$.

Write a program to determine whether one set of identifiers is a proper subset of another.

Input

The input will begin with a positive integer that represents the number of test cases. Each test case will consist of two sets, one per line. The sets will begin with a curly brace and have 0 to 20 identifiers in the set. Identifiers will be 1 to 15 characters. The characters in the identifiers will be just letters and digits. There will be one comma between identifiers and a closed curly brace at the end of the list of all identifiers. There may be blanks between the braces and the identifier and between the identifiers and the commas. Case is not significant in the identifiers, so "BEAR" and "bear" should be considered to be the same identifier. There may be duplicates in the lists.

Output

The output for each case will be the case number, followed by a colon, a space and the word YES indicating if the second set is a proper subset of the first set and NO otherwise.

Sample input

```
4
{Apple,Grape,Orange,Banana,Pear}
{ORANGE,GRAPE}
{Apple, Grape, Pear, Orange, GrapeFruit, Apple}
{ORANGE, BANANA, GRAPE, Grape, Grape, Grape, grape}
{ Grapefruit }
{Grape}
{Door,Shelf,Knob,Window}
{Door,Shelf,Knob,Window}
```

Output (corresponding to sample input)

```
1: YES
2: NO
3: NO
4: NO
```

Prob10: Knightly Knowledge

Ancient cultures had an enormous and mostly unexpected knowledge of the earth and geometry. When they were building monuments, they did not place them at arbitrary locations. Instead, each building spot was carefully selected and calculated. Nowadays we can observe this in the form of so-called *ley lines*. A ley line is a horizontal or vertical straight line of infinite length that runs through at least two ancient monuments. Ley lines are expected to be a source of some kind of magic energy and every church built along such a line is a *mighty church*.

There are already several monuments and churches. An ancient culture is planning to construct a new monument, but the location is yet to be determined. They are looking for the spot that will turn the most ordinary churches into mighty churches. The monument can be co-located with a church or an existing monument – in that case, the new monument will simply be built around the church or monument.

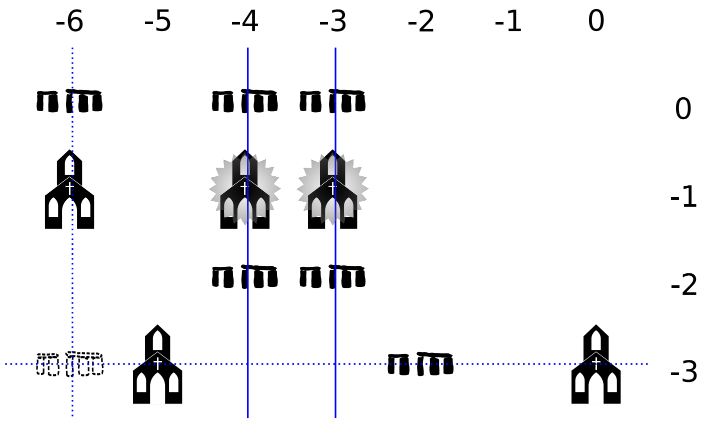


Figure K.1: Illustration of Sample 2 with 2 ley lines (|), 6 monuments (), 2 mighty churches ( ) and 3 ordinary churches ( ) that are turned into mighty churches when building the dashed monument.

Input

The input consists of:

- One line with two integers m and c ($0 \leq m, c \leq 1\,000$), the number of already built monuments and churches.
- m lines with the coordinates of the monuments.
- c lines with the coordinates of the churches.

All coordinates are given as two integers x and y ($-10^6 \leq x, y \leq 10^6$). None of the given coordinate pairs coincide with one another, but any may coincide with the location of the new monument.

Output

Output three integers: the coordinates for where to build the new monument and the number of ordinary churches that will be turned into mighty churches with this new monument. The coordinates should be in the range ($-10^6 \leq x, y \leq 10^6$). If there are multiple optimal solutions, you may output any one of them.

Sample Input 1

```
2 3
0 5
5 0
0 1
0 3
3 0
```

Sample Output 1

```
0 0
3
```

Sample Input 2

```
6 5
-6 0
-4 0
-3 0
-4 -2
-3 -2
-2 -3
-6 -1
-4 -1
-3 -1
-5 -3
0 -3
```

Sample Output 2

```
-6 -3
3
```

Sample Input 3

```
1 2
0 0
1 0
0 1
```

Sample Output 3

```
0 0
2
```

Prob11: Mixtape Management

Mary has created a mixtape with her favourite reggae tracks. The mixtape consists of a list of MP3 files on her computer that she wants to share with her friends Wendy and Larry. However she knows that her friends have different musical tastes and will therefore also have different preferences for the order in which the tracks are played.

Mary knows that Wendy is a Windows user and Larry is a Linux user and realised that she can use this to her advantage. This is because Windows and Linux use different methods to sort files within a directory in case their names contain numerical data. In Windows, numbers in file names are read as actual numbers, causing the files to be sorted by increasing values of these numbers. In Linux, there is no special handling for numbers, so file names are sorted lexicographically. See Figure M.1 for an example of file sorting on the two operating systems.

Linux	Windows
`337.mp3`	`7.mp3`
`34.mp3`	`34.mp3`
`3401.mp3`	`79.mp3`
`7.mp3`	`337.mp3`
`780.mp3`	`780.mp3`
`7803.mp3`	`3401.mp3`
`79.mp3`	`7803.mp3`

Figure M.1: Illustration of the first sample case. Note that the file extensions `.mp3` do not influence the ordering and are purely for illustration.

After deciding on the order in which she wants Wendy and Larry to listen to the tracks, Mary has already sorted the files according to Larry's taste. Now she wants to rename the files such that the filenames are distinct positive integers without leading zeroes, they are sorted in increasing lexicographic order, and when sorting them by value the new order will match Wendy's taste. For this purpose, she has come up with a permutation $p_1, \ldots, p_n$ that describes how to rearrange Larry's list into Wendy's list: for every i, the ith number in lexicographic order needs to be the p_ith smallest by value. Help Mary find a suitable sequence of filenames.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 100$), the number of tracks.
- One line with n distinct integers $p_1, \ldots, p_n$ ($1 \leq p_i \leq n$ for each i), the given permutation.

Output

Output n distinct integers in lexicographically increasing order, your sequence of filenames. All numbers must be positive integers less than 10^{1000} and may not contain leading zeroes. Any valid sequence of filenames will be accepted.

Sample Input 1

```
7
4 2 6 1 5 7 3
```

Sample Output 1

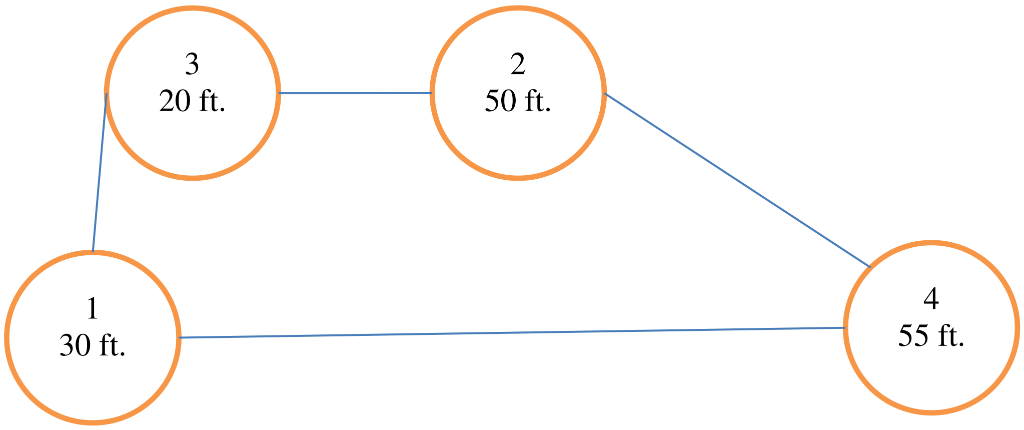
```
337 34 3401 7 780 7803 79
```

Sample Input 2	Sample Output 2
4 4 1 3 2	234 6 87 9

Walking (prob12)

April and Chip, now married for quite some time, have realized that their routine walks are pretty boring. So, April, being the smarter of the two, devises a way to make their walks more interesting: Given two locations that they want to visit, minimize the work required to get from one to another. Since April and Chip don't really like climbing hills, they decided to allow work to be the change in elevation. So, if they walked up a hill with a 10 foot elevation change, and then back down, the amount of work they would have done would be 20.

Consider the world below:



To go from 1 to 2, Chip and April could go to 3 first, giving a total work of 40 (10 from 1 to 3, 30 from 3 to 2) or go to 4 first for a total work of 30 (25 from 1 to 4, 5 from 4 to 2). They'll pick the path from 1 to 4 to 2, since that's the smaller value.

Your task is to determine the path that requires the minimum amount of work, given a set of nodes with elevation and the paths between them.

Input

The input will consist of a number of lines, the first having a single integer C, telling how many cases to follow. The next C cases will be formatted as follows. There will first be a line with a number N, telling how many locations are in the neighborhood, where the first location is location 1, the second location 2, and so on until the last, which is location N. The next N lines will contain a single integer each, where the *i*th of these lines represents the integer elevation for location *i*. Following this is an integer M, telling how many paths there are between locations. There will then be M lines with pairs of integers representing locations in parentheses indicating there is a path between the locations. A comma followed by a space separates the integers. Paths go both ways, so a path from 1 to 2 is also a path from 2 to 1. After those lines follows another integer K, indicating how many locations will be visited. There will then be K lines, each containing the locations that Chip and April want to visit in the order that they should be visited. You should assume that they start on the first location in this set, and stop on the last location in this set.

Output

For each case, you should print: "The least amount of work on trip n is: A", where n is the trip number (starting at 1) and A is the least cost. You may assume that all locations that are on April and Chip's walk are in fact reachable.

Sample Input

```
1
4
30
45
20
55
4
(1, 3)
(2, 4)
(4, 1)
(3, 2)
4
1
2
3
4
```

Sample Output

```
The least amount of work on trip 1 is: 95
```