MERCER UNIVERSITY

Spring Programming Contest

February 23, 2013

An Odd Sum	(prob1)
Best of Seven	(prob2)
Block Party	(prob3)
Canal Navigation	(prob4)
Contest Rules	(prob5)
Morse Enumerations	(prob6)
Namography	(prob7)
Promoting Elves Rights	(prob8)
Proper Subsets	(prob9)
Rock Candy	(prob10)
Searching for a Cure	(prob11)
Walking	(prob12)

The name in parenthesis following each problem is the name you must use as your program's name. You must add the appropriate extension depending upon your choice of programming language (.c .cpp .cs .java .py .rb).

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, $1 \leq n \leq 10^9$ 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]
```

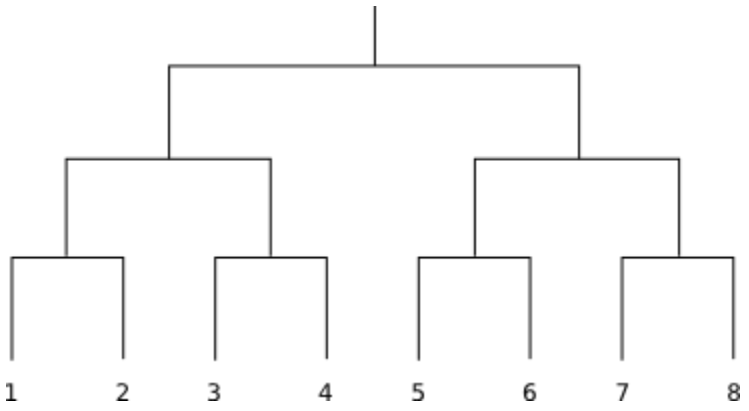
[1]**Adapted from *An Odd Sum* by Ray C. Shiflett and Harris S. Shultz and published in The Mathematics Teacher, Vol. 95, No. 3 (March 2002), pp. 206-209**

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 GRGYBbB OGRGOBY BBGB**RR**B YRY**RR**OR BOYBO**B**O YGBY**BB**G OOBY**B**RG BBYRRBO	YOOGYBY GRGYBbB OGRGOBY BBGB B YRY OR BOYBO O YGBY G OOBY RG BBYRRBO	YOO Y GRGG B OGRY Y BBGG BB YRYBYbR BOYBBBO YGBYOOG OOBYORG BBYRRBO	YOO Y GRGG B OGRY Y BBGG BB YRY**B**YbR BOY**BBB**O YGBYOOG OOBYORG 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
```

Contest Rules (prob5)

Jesse University is holding a programming contest. To register for this contest, schools will need to do three simple steps:

- provide the names of their teams
- pay the registration fee
- submit a problem to the contest

If a school does all three steps by the registration deadline, it is fully registered and its teams are counted in the number of teams in the contest. If a school does two of the three steps, a reminder letter is sent. If a school only does one of the steps, its registration is cancelled. If a school has several teams, they only need to pay once and submit one problem.

Since handling registration is only a small part of running a programming contest, you have been asked to write a program to go through registration information and determine how many teams are registered, which schools need letters, and which registrations will be cancelled.

Input

The first line of the input will be a positive integer with the number of data sets in the input. Each data set will have the information for one contest.

The first line of a data set will be a nonnegative integer *t*, representing the number of team names provided. There will then be *t* lines, each with the name of a school providing a team name (team names are not included in this input). There will be one entry for each team from a school. The next line of the data set will a nonnegative integer *m*, representing the number of schools who have paid their registration fee. There will then be *m* lines, each with the name of a school who paid their fee. Finally, there will be a line with a nonnegative integer *p*, representing the schools that have submitted a problem. There will then be *p* lines, each with the name of a school who submitted a problem. There will be one entry for each problem from a school.

Wherever the input is a school name, blank spaces are not significant at the ends of the school name, but case is significant so "Jesse University" is considered to be a different school from "JESSE University". Both of these school names are different from “Jesse University”.

Output

For each contest, the output should start with a line giving the number of the contest. There should then be a line giving the number of teams that are fully registered (have registered the team, paid and submitted a problem). The next lines should then list the schools that are fully registered, one per line, each followed by the number of teams registered by the school. The school names should be listed in alphabetical order.

There should then be a section of the schools that need a reminder letter (those who have done two of the three steps) that begins with the title shown and lists the school names one per line in alphabetical order. Finally, there should be a section of the schools whose registrations will be cancelled (those who have done one of the three steps) that begins with the title shown and lists the schools one per line in alphabetical order.

If there are any sections (registered/need reminder/cancel) that have no schools, the section header should not be printed.

There should be a blank line after the data for each contest.

Sample input

```
2
7
Rockefeller University
Willingham College
Gamble College
Wolfson College
Rockefeller University
Willingham College
Benjamin University
3
Benjamin University
Willingham College
Wolfson College
4
Willingham College
Gamble College
Benjamin University
Mary Blount College
3
Bob John University
BOB John University
Bob John University
1
Bob John University
1
Bob John University
```

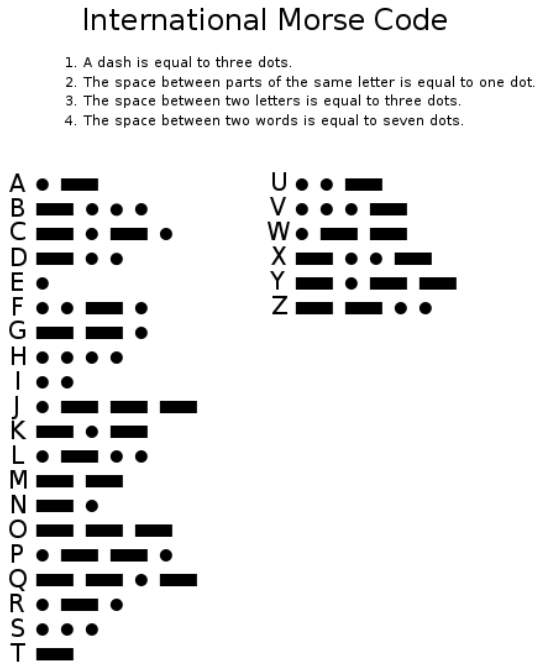
Sample output (corresponding to sample input)

```
Contest number 1
---3 Registered Teams:
Benjamin University (1)
Willingham College (2)
---Send reminders to:
Gamble College
Wolfson College
---Cancel registration for:
Mary Blount College
Rockefeller University

Contest number 2
---2 Registered Teams:
Bob John University (2)
---Cancel registration for:
BOB John University
```

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`):



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
```

Namography (prob7)

Given an unidentified chemical compound, scientists often use *chromatography* to separate and identify the constituent chemical elements. Unfortunately, analysis of one or two characteristics proves insufficient to separate out the constituent elements for certain compounds. To illustrate this problem, we will use a substitute technique for chromatography called *namography* on a collection of U.S. presidential candidates. In this namography example, each candidate is identified by a point <*x*, *y*, *z*> where *x* and *y* represent the number of letters in the candidate's first name and last name respectively and *z* represents the candidate's height in centimeters. If a mixture of candidates has two or more candidates that share the same values for *x* and *y*, namography will combine their points into a single point <*x*, *y*, *sum of the z's*>.

Write a program that determines the points produced by this namography technique from a mixture of presidential candidates and whether all the candidates are separated.

Input

The first line of input contains a single integer *m*, ($1 \le m \le 100$), which is the number of candidate mixtures to analyze. Each mixture consists of a single line containing a single integer *c*, ($1 \le c \le 200$), which is the number of presidential candidates in the mixture followed by *c* lines containing the first name, a space, the last name, a space, and an integer *h*, ($1 \le h \le 200$), which is the height in centimeters of each candidate. You may assume that no candidate appears more than once per mixture.

Output

For each candidate mixture, you should generate one line of output with the following values: the mixture number as a decimal integer (start counting at one), a space, the word yes or no indicating whether all the candidates are separated by this namography technique, a space, and the points generated by the namography technique in the form <x,y,z>, sorted by x then y then z and separated by single spaces.

Sample Input

```
2
2
John McCain 175
Barack Obama 185
3
John McCain 175
Barack Obama 185
Mitt Romney 188
```

Output (corresponding to sample input)

```
1 yes <4,6,175> <6,5,185>
2 no <4,6,363> <6,5,185>
```

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
```

Rock Candy (prob10)

Toby loves rock candy. In fact it is his favorite type of food! Lucky for him he lives very close to a rock candy mine. Toby is also very lazy. That is why he would like to do the minimum work possible to collect his rock candy.

The bottom of the rock candy mine is a rectangular platform. Around the outside of the platform he has placed collections devices. If Toby can push the rock candy into the collections devices he will have more rock candy for his candy stockpile.

Pushing candy is simple. Toby just has to maneuver himself to the far side of the piece of candy and push. The candy will move away from Toby and will hopefully end up in a collection bin. The diagram below shows Toby pushing a piece of rock candy off the platform into a collection bin. Notice it only takes one push to move the rock candy off the platform. Also notice that once the candy leaves the square that location is now walkable.

```
..........        ..........

..TR------>   =>  ..T........R

..........        ..........
```

Despite its name, rock candy is actually very fragile. If a piece of rock candy is pushed into another piece of rock candy then it will break apart and he will be forced to sweep it up to collect it. Toby has decided that this is too much work. Similarly if the rock candy collides with an immoveable slab of rock then the candy will break apart. This also would cause a huge mess that he would rather not clean up. It is also worth noting that Toby cannot move through rock candy or rock slabs.

This is why Toby would like you to devise a strategy to collect as much rock candy as possible. Of the ways to collect the most rock candy he would like you to determine the minimum amount of effort possible. Toby defines effort as taking a single step or pushing a single piece of rock candy. To keep the directions simple for Toby, he is only allowed to take steps in the four cardinal directions on the platform.

Input

The first line contains a single integer *m* representing the number of mines that need a strategy. For each mine, the first line will be two integers *r and c* representing the number of rows and columns in the mine description, where $2 \leq r,c \leq 30$. The next *r* lines contains *c* characters, representing the mine description.

Each character of the mine will be one of the following:

'.' representing empty walkable space.

'T' representing where Toby is initially standing. (walkable)

'R' representing a piece of rock candy.

'#' representing an immovable piece of rock slab.

There will be exactly one 'T' character and at most 10 pieces of rock candy.

Output

For each mine output the following header, "Mine #*i*:" where *i* is the current mine being processed starting from 1. Follow this by a single space and two space separated integers. The first is the maximum number of rock candy pieces that can be found. The second is the minimum amount of effort required to obtain those pieces. Print a blank line after output line.

Sample Input	**Sample Output (corresponding to sample input)**
3 3 3 ..T RR. RR. 4 6 ###..T .R.#.. ##..R. .R.... 4 5 .#T#. RR.RR #.... R#.#.	Mine #1: 0 0 Mine #2: 3 11 Mine #3: 2 9

Searching for a Cure (prob11)

Stiff person syndrome (SPS) is an extremely rare neurologic disorder that is characterized by muscle rigidity and spasms in the trunk and limbs and a heightened sensitivity to stimuli such as noise, touch, and emotional distress. While it may only affect one in a million people, one of the former judges of this contest is currently suffering from it, so we're interested in finding out more about it. Unfortunately, while there is much written about medical issues, it's difficult to find information about such a rare condition. For this problem, you will write a program that will search a stream of text and identify lines which may be about SPS.

In order for a line to be considered to be about SPS, it needs to have at least one of the following groups of strings.

- at least one of the strings "stiff person syndrome" or "Moersch-Woltman Condition" in any case (upper, lower, or mixed). Spacing is significant so “stiff person syndrome” does not match.
- at least two of the strings "SPS", "GAD", "GABA", or "GLRA1" in upper case only

Notice that you only need to search for strings, not words. If you have the line "GADSDEN PURCHASE", it does contain the string "GAD".

Input

The input to the program will consist of one or more lines of text. End of input will be indicated by end file.

Output

The output should be a list of lines which meet the conditions above. You should print the line number (where the first line in the input is line number 1), a colon, a space, and then the first 40 characters in the line (or the entire line if it is less than 40 characters long). Lines that match should be printed only once, in the order they appear in the input.

Sample input

```
Sadly, Dr. Plaut, the Maryville coach will not be at the contest.
She is suffering from Stiff Person Syndrome (SPS).
Perhaps it should be known as Stiff Woman's Syndrome since 2/3 of the people
with SPS are female.  But SPS affecting just 1 in 1,000,000, it's hard to know.
GAD antibodies have been linked to SPS.
But gad antibodies only appear in 60% of SPS cases.
```

Sample output (corresponding to sample input)

```
2: She is suffering from Stiff Person Syndr
5: GAD antibodies have been linked to SPS.
```

References

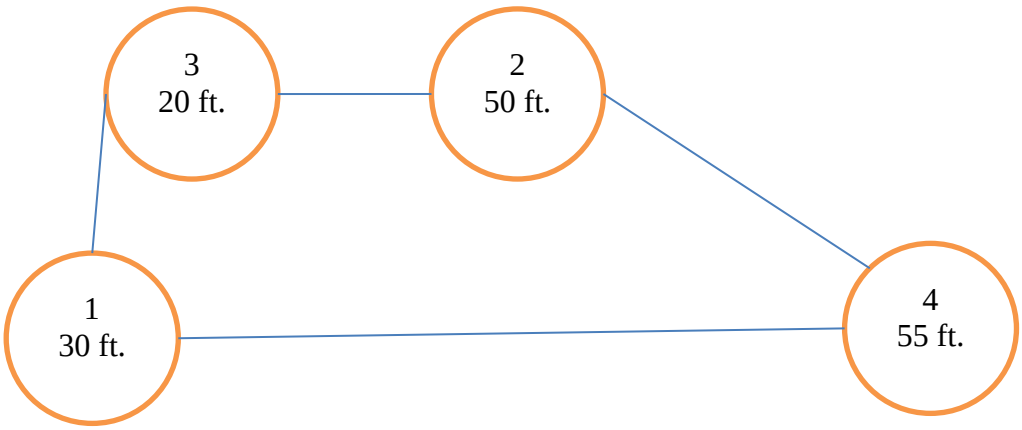
http://www.ninds.nih.gov/disorders/stiffperson/stiffperson.htm

http://en.wikipedia.org/wiki/Stiff_person_syndrome

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
```