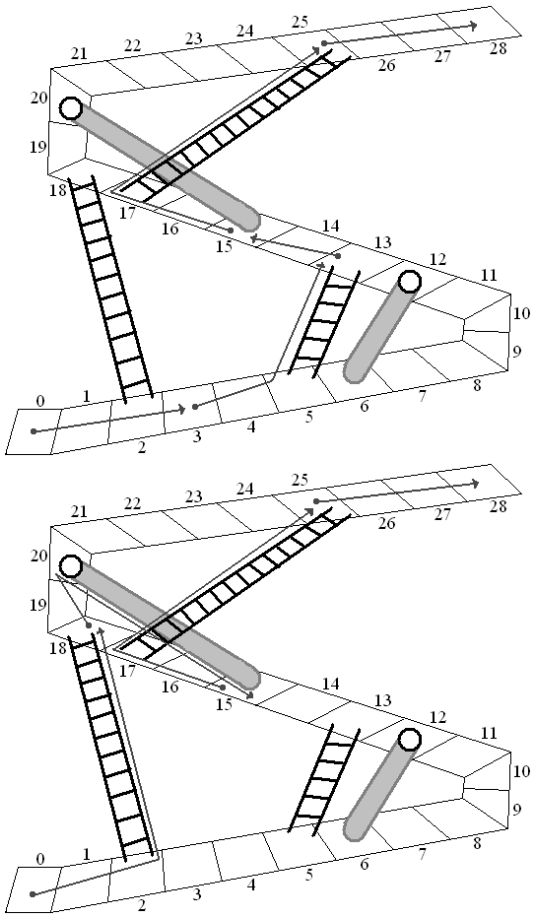
Problem A. Up and Down

Input file: `up.in`
Output file: `up.out`



This problem is based on a children's game, Chutes and Ladders, where players take turns jumping a number of steps along a path. If they land on the base of a ladder, they rise to the top of the ladder in the same turn. If they land at the top of a chute, they slide down to the bottom in the same turn. The idea is to get to the final step on the path. In the children's game the number of steps to move in a turn is determined randomly, so the game requires no decisions. In the version here, Up and Down, players get to choose the number of steps to jump forward in each turn. Both figures above show spaces numbered from 0 to 28, with several chutes and ladders. They show two different possible sequences of moves, assuming jumps of 1, 2 or 3 are allowed. Each jump is illustrated starting at a gray dot and ending at an arrowhead, jumping 1-3 places ahead, sometimes ending there, and sometimes shifting up a ladder or down a chute. The players in Figures 1 and 2 finish in 5 and 4 turns respectively. Figure 2 demonstrates the minimum number of turns for this path configuration and a maximum jump of 3.

It gets harder a figure out the best path if there are more chutes and ladders and many more spaces along the path! Be careful of your algorithm, given the large limit on the number of spaces specified below.

Input

The input will consist of one to twenty data sets, followed by a line containing only 0. The first line of a dataset contains blank separated integers w s p, where w is the number of the winning space, $3 \le w \le 1\,000\,000\,000$, s is the maximum number of spaces to jump in each turn, $2 \le s \le 6$, and p is the total number of chutes and ladders, $1 \le p \le 40$.

The remaining lines of the data set consist of pairs of integers b_i e_i, for $i = 1, 2, ...p$. Each b_i e_i pair is the beginning space and ending space for a chute or ladder, so a turn with a jump to b_i actually ends at e_i. All the integers are positive and less than w, and none of these $2p$ numbers appears more than once. Following the rules of the game, it is possible to eventually reach space w, starting from space 0, for each dataset. The numbers b_i are in increasing order. Numbers in these lines are separated by either a single blank or a newline.

Output

There is one line of output for each data set, containing only the minimum number of turns it takes to start at space 0 and end at space w, with the jump in each turn chosen as a positive integer no larger than s.

The first sample input data set corresponds to the configuration in the Figures.

Example

up.in	up.out
28 3 5 2 18 5 13 12 6 17 25 20 15 50 6 1 9 45 0	4 3

Problem B. Gnome Sequencing

Input file: `gnome.in`
Output file: `gnome.out`

In the book All Creatures of Mythology, gnomes are kind, bearded creatures, while goblins tend to be bossy and simple-minded. The goblins like to harass the gnomes by making them line up in groups of three, ordered by the length of their beards. The gnomes, being of different physical heights, vary their arrangements to confuse the goblins. Therefore, the goblins must actually measure the beards in centimeters to see if everyone is lined up in order.

Your task is to write a program to assist the goblins in determining whether or not the gnomes are lined up properly, either from shortest to longest beard or from longest to shortest.

Input

The input starts with line containing a single integer N, $0 < N < 30$, which is the number of groups to process. Following this are N lines, each containing three distinct positive integers less than 100.

Output

There is a title line, then one line per set of beard lengths. See the sample output for capitalization and punctuation

Example

gnome.in	gnome.out
3 40 62 77 88 62 77 91 33 18	Gnomes: Ordered Unordered Ordered

Problem C. DuLL

Input file: dull.in
Output file: dull.out

In Windows, a DLL (or dynamic link library) is a file that contains a collection of pre-compiled functions that can be loaded into a program at runtime. The two primary benefits of DLLs are (1) only one copy of a DLL is needed in memory, regardless of how many different programs are using it at the same time, and (2) since they are separate from programs, DLLs can be upgraded independently, without having to recompile the programs that use them. (DLLs have their problems, too, but we'll ignore those for now.) Your job is to calculate the maximum memory usage when running a series of programs together with the DLLs they need.

The DLLs in our system are not very exciting. These dull DLLs (or DuLLs) each require a fixed amount of memory which never changes as long as the DuLL is in memory. Similarly, each program has its own fixed memory requirements which never change as long as the program is executing. Each program also requires certain DuLLs to be in memory the entire time the program is executing. Therefore, the only time the amount of memory required changes is when a new program is executed, or a currently running program exits. When a new program begins execution, all DuLLs required by that program that must be loaded into memory if they are not there already. When a currently running program exits, all DuLLs that are no longer needed by any currently running programs are removed from memory.

Remember, there will never be more than one copy of a specific DuLL in memory at any given time. However, it is possible for multiple instances of the same program to be running at the same time. In this case each instance of the program would require its own memory; however, the instances still share DuLLs in the same way two unrelated programs would.

Input

The input consists of at least one data set, followed by a line containing only 0.

The first line of a data set contains three space separated integers N P S, where N is the number of DuLLs available, $1 \le N \le 20$, P is the number of programs which can be executed, $1 \le P \le 9$, and S is the number of state transitions recorded, $1 \le S \le 32$.

The next line contains exactly N space separated integers representing the sizes in bytes of each of the DuLLs, $1 \le size \le 1\,000$. Each DuLL is implicitly labeled with a letter: 'A', 'B', 'C', ..., possibly extending to 'T'.

Therefore the first integer is the size of 'A', the second integer is the size of 'B', and so on. The next P lines contain information about each of the programs, one program per line. Each line contains a single integer representing the size of the program in bytes, $1 \le size \le 1\,000$, followed by 1 to N characters representing the DuLLs required by that program. There will be a single space between the size of the program and the DuLL labels, but no spaces between the labels themselves. The order of the labels is insignificant and therefore undefined, but they will all be valid DuLL labels, and no label will occur more than once. Each program is implicitly labeled with an integer: 1, 2, 3, ... possibly extending to 9. The final line of the data set will contain S space separated integers. Each integer will either be a positive number q, $1 \le q \le P$, indicating that a new execution of program q has begun, or else it will be a negative number q, $1 \le q \le P$, indicating that a single execution of program q has completed. The transitions are given in the order they occurred. Each is a valid program number; if it is a negative number q then there will always be at least one instance of program q running.

Output

There is one line of output for each data set, containing only the maximum amount of memory required throughout the execution of the data set.

Example

dull.in	dull.out
2 2 3 500 600 100 A 200 B 2 1 2 5 4 8 100 400 200 500 300 250 AC 360 ACE 120 AB 40 DE 2 3 4 -3 1 2 -2 1 0	1600 2110

Problem D. Black Vienna

Input file: vienna.in
Output file: vienna.out

This problem is based on the game of Black Vienna. In this version there are three players and 18 cards labeled A-R. Three of the cards are set aside (hidden) and form the "Black Vienna" gang. The remaining cards are shuffled and dealt to the players so that each player has 5 cards. Players never reveal their cards to each other. There is a separate deck of "interrogation cards" which contain three distinct letters in ascending order, like ACG or BHR. Turns rotate through players 1, 2, and 3. On each player's turn, that player selects an interrogation card, puts it face up in front of another player, and that other player must indicate the total number of these cards being held, without saying which ones. All players see the result of the "interrogation". The play continues until a player deduces the three cards in the "gang".

For example, suppose the cards are distributed as follows,

and the game then proceeds:

Player 1: DGJLP; Player 2: EFOQR; Player 3: ACHMN; Gang: BIK

Turn 1: Player 1 interrogates player 2 with BJK; answer 0
Turn 2: Player 2 interrogates player 3 with ABK; answer 1
Turn 3: Player 3 interrogates player 2 with DEF; answer 2
Turn 4: Player 1 interrogates player 2 with EIL; answer 1
Turn 5: Player 2 interrogates player 3 with FIP; answer 0
Turn 6: Player 3 interrogates player 1 with GMO; answer 1
Turn 7: Player 1 interrogates player 2 with OQR; answer 3
Turn 8: Player 2 interrogates player 3 with ADQ; answer 1
Turn 9: Player 3 interrogates player 1 with EGJ; answer 2

In fact, the game does not need to get to turn 9. With enough thought, player 1 can deduce after turn 8 that the gang is BIK. It is your job to analyse records of games and deduce the earliest time that the gang could be determined for sure.

Input

The input will consist of one to twelve data sets, followed by a line containing only 0.

The first line of a dataset contains the number, t, of turns reported, $2 \leq t \leq 15$.

The next line contains four blank separated strings for the hands of players 1, 2, and 3, followed by the cards for the gang.

The remaining t lines of the data set contain the data for each turn in order. Each line contains three blank separated tokens: the number of the player interrogated, the string of interrogation letters, and the answer provided.

All letter strings will contain only capital letters from A to R, in strictly increasing alphabetical order. The same interrogation string may appear in more than one turn of a game.

Output

There is one line of output for each data set. The line contains the single character "?" if no player can be sure of the gang after all the turns listed. If a player can determine the gang, the line contains the earliest turn after which one or more players can be sure of the answer.

Example

vienna.in	vienna.out
9 DGJLP EFOQR ACHMN BIK 2 BJK 0 3 ABK 1 2 DEF 2 2 EIL 1 3 FIP 0 1 GMO 1 2 OQR 3 3 ADQ 1 1 EGJ 2	8 ?

vienna.in	vienna.out
3 ABCDE FGHIJ KLMNO PQR 3 BKQ 1 1 ADE 3 2 CHJ 2 0	

Problem E. Duplicate Removal

Input file: dup.in
Output file: dup.out

The company Al's Chocolate Mangos has a web site where visitors can guess how many chocolate covered mangos are in a virtual jar. Visitors type in a guess between 1 and 99 and then click on a "Submit" button. Unfortunately, the response time from the server is often long, and visitors get impatient and click "Submit" several times in a row. This generates many duplicate requests.

Your task is to write a program to assist the staff at ACM in filtering out these duplicate requests.

Input

The input consists of a series of lines, one for each web session. The first integer on a line is N, $0 < N \leq 25$, which is the number of guesses on this line. These guesses are all between 1 and 99, inclusive. The value $N = 0$ indicates the end of all the input.

Output

For each input data set, output a single line with the guesses in the original order, but with consecutive duplicates removed. Conclude each output line with the dollar sign character '$'. Note that there is a single space between the last integer and the dollar sign.

Example

dup.in	dup.out
5 1 22 22 22 3 4 98 76 20 76 6 19 19 35 86 86 86 1 7 0	1 22 3 $ 98 76 20 76 $ 19 35 86 $ 7 $

Problem F. Rock, Paper, Scissors

Input file: rps.in
Output file: rps.out

Rock, Paper, Scissors is a classic hand game for two people. Each participant holds out either a fist (rock), open hand (paper), or two-finger V (scissors). If both players show the same gesture, they try again. They continue until there are two different gestures. The winner is then determined according to the table below:

Rock beats Scissors

Paper beats Rock

Scissors beats Paper

Your task is to take a list of symbols representing the gestures of two players and determine how many games each player wins.

In the following example:

Turn :	1 2 3 4 5
Player 1 :	R R S R S
Player 2 :	S R S P S

Player 1 wins at Turn 1 (Rock beats Scissors), Player 2 wins at Turn 4 (Paper beats Rock), and all the other turns are ties.

Input

The input contains between 1 and 20 pairs of lines, the first for Player 1 and the second for Player 2. Both player lines contain the same number of symbols from the set {'R', 'P', 'S'}. The number of symbols per line is between 1 and 75, inclusive. A pair of lines each containing the single character 'E' signifies the end of the input.

Output

For each pair of input lines, output a pair of output lines as shown in the sample output, indicating the number of games won by each player.

Example

rps.in	rps.out
RRSRS	P1: 1
SRSPS	P2: 1
PPP	P1: 0
SSS	P2: 3
SPPSRR	P1: 2
PSPSRS	P2: 1
E	
E	

Problem G. A to Z Numerals

Input file:	numeral.in
Output file:	numeral.out

Roman numerals use symbols I, V, X, L, C, D, and M with values 1, 5, 10, 50, 100, 500, and 1000 respectively. There is an easy evaluation rule for them:

Rule: Add together the values for each symbol that is either the rightmost or has a symbol of no greater value directly to its right. Subtract the values of all the other symbols. For example: MMCDLXIX $= 1000 + 1000 - 100 + 500 + 50 + 10 - 1 + 10 = 2469$.

Further rules are needed to uniquely specify a Roman numeral corresponding to a positive integer less than 4000:

1. The numeral has as few characters as possible. (IV not IIII)
2. All the symbols that make positive contributions form a non-increasing subsequence. (XIV, not VIX)
3. All subtracted symbols appear as far to the right as possible. (MMCDLXIX not MCMDLIXX)
4. Subtracted symbols are always for a power of 10, and always appear directly to the left of a symbol 5 or 10 times as large that is added. No subtracted symbol can appear more than once in a numeral.

Rule 4 can be removed to allow shorter numerals, and still use the same evaluation rule: IM $= -1 + 1000 = 999$, ILIL $= -1 + 100 + -1 + 100 = 198$, IVL $= -1 - 5 + 100 = 94$. This would not make the numerals unique, however. Two choices for 297 would be CCVCII and ICICIC. To eliminate the second choice in this example, Rule 4 can be replaced by

4'. With a choice of numeral representations of the same length, use one with the fewest subtracted symbols.

Finally, replace the Roman numeral symbols to make a system that is more regular and allows larger numbers: Assign the English letter symbols a, A, b, B, c, C, ..., y, Y, z, and Z to values 1, 5, 10, 5×10, 10^2, 5×10^2, ..., 10^{24}, 5×10^{24}, 10^{25}, and 5×10^{25} respectively. Though using the whole alphabet makes logical sense, your problem will use only symbols a-R for easier machine calculations. ($R = 5 \times 10^{17}$)

With the new symbols a-Z, the original formation rules 1-3, the alternate rule 4', and the evaluation rule, numerals can be created, called A to Z numerals. Examples: ad $= -1 + 1000 = 999$; aAc $= -1 - 5 + 100 = 94$.

Input

The input starts with a sequence of one or more positive integers less than 7×10^{17}, one per line. The end of the input is indicated by a line containing only 0.

Output

For each positive integer in the input, output a line containing only an A to Z numeral representing the integer.

Example

numeral.in
999
198
98
297
94
666666666666666666
0

numeral.out
ad
acac
Acaaa
ccAcaa
aAc
RrQqPpOoNnMmLlKkJjIiHhGgFfEeDdCcBbAa

Problem H. Cell Towers

Input file:	cell.in
Output file:	cell.out

Cell phones generally provide service by connecting to a nearby cellular tower. At any given time there may be several towers in range. The cell phone, however, will only connect to the one with the best signal strength. The purpose of

this problem is to track a traveler carrying a cell phone. At each mile marker along a road, note which tower the cell phone is using, and report when it is different from the previous marker. The position of towers will be specified as X-Y coordinates in miles relative to some arbitrary origin. The traveler will travel down a road composed of straight line segments laid end to end. The road will not intersect itself. Assume that there are markers occurring every mile along the road, with mile marker zero being at the starting point. If the road ends at least 0.5 miles past the last mile marker, the end of the road is labeled with the next mile. For instance if the road is 8.6 miles long, the endpoint is labeled as mile 9. If the road is 8.2 miles long, regular mile marker 8 is the last.

If d is the distance to a tower with power p, the signal strength for the tower will be calculated as p/d^2, rounded to the nearest integer. A tower will never be placed at the position of a mile marker. Consider the examples shown in the Figures below. Each shows a road and labeled mile markers and cell towers.

In Figure 1, where the segments of the road all follow the background grid, mile markers come at intersections in the grid. The cell towers A and B are at $(1, 4)$ and $(5, 4)$. Both have power 1 000. At mile markers 0 and 1 the strength of A is greater. At mile markers 2 – 4 the strength of A and B are equal. In such cases you are to note the cell tower with the label closer to the beginning of the alphabet, so it continues to be A. At mile markers 5 – 9 B is stronger. At mile 10 the towers are of equal strength, but the one with first letter is reported, A. Tower A remains strongest to the end. The long crossbars show the mile markers where the reported strongest tower is different than at the previous marker.

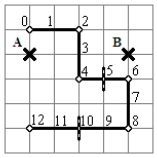


Figure 1

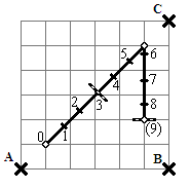


Figure 2

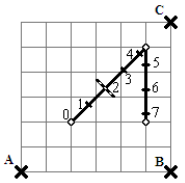


Figure 3

The example in Figure 2 has three towers: A at $(0, 0)$ and C at $(6, 6)$, both with power 1 000, while tower B at $(6, 0)$ has a power of 600. The mile markers are at the tick marks. They show the traveler's progression along the road. The mile markers are no longer at grid intersections because of the angled road. Initially tower A is the strongest. Tower C is strongest at mile markers 3 – 8. The end of the road at $(5, 2)$ is more than a half mile from mile 8, so it is labeled as mile 9. At the endpoint, B is strongest, unlike at mile 8, so a report is made for the endpoint.

Figure 3 is similar to Figure 2, except it shifts the beginning of the road and changes the power of cell tower B to 300. Tower A starts as the strongest, and then at mile markers 2 – 7 tower C is strongest. The endpoint at $(5, 2)$ is not considered, since it is less than .5 miles from mile marker 7, even though tower B is strongest at this endpoint.

Input

The input consists of at least one data set, followed by a line containing only 0.

The first line of a data set contains two space separated integers T R, where T is the number of towers, $1 \le T \le 10$, R is the number of line segments which comprise the road, $1 \le R \le 10$.

The next T lines each contain three space separated integers representing the X-coordinate, Y-coordinate, and power of one tower, respectively. The towers are implicitly labeled 'A', 'B', 'C', and so on.

The next line contains $2(R + 1)$ integers which are the coordinates for the $R + 1$ points that define the road. The road starts on the first point and moves in straight line segments through all R remaining points. All coordinates will be integers between 0 and 100, inclusive. No two coordinate pairs are equal. The power for a tower will be an integer between 1 and 1 000 000, inclusive.

Output

There is one line of output for each road in the data set. The line consists of ordered pairs separated by a single space. The first element of a pair is a number representing a mile marker. The second element of the pair is a letter corresponding to the tower with the strongest signal. There are entries for mile 0 and every mile marker where the strongest tower recorded is different than at the previous mile marker. The ordered pairs are surrounded by parentheses, and the elements are separated by a comma, with no whitespace inside the ordered pair.

Example

cell.in
2 5 1 4 1000 5 4 1000 1 5 3 5 3 3 5 3 5 1 1 1 3 2 0 0 1000 6 0 600 6 6 1000 1 1 5 5 5 2 3 2 0 0 1000 6 0 300 6 6 1000 2 2 5 5 5 2 0
cell.out
(0,A) (5,B) (10,A) (0,A) (3,C) (9,B) (0,A) (2,C)

Problem I. RIPOFF

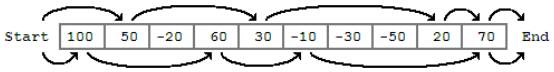
Input file: `ripoff.in`
Output file: `ripoff.out`

Business has been slow at Gleamin Lemon Used Auto Sales. In an effort to bring in new customers, management has created the Rebate Incentive Program Of Fabulous Fun (or RIPOFF). This is a simple game which allows customers

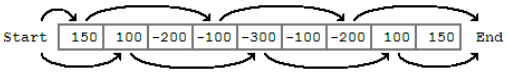
to try and win a rebate on an automobile purchase. The RIPOFF game is a board game where each square is labeled with a rebate amount. The customer advances through the board by spinning a spinner. Each square he lands on adds to his total rebate amount. When he reaches the end of the board he is rewarded with the total rebate amount.

Of course, given the company involved, it should come as no surprise that there are a couple of catches written in the fine print. The first is that there is a limit to the number of turns the customer has to finish the game; if he doesnt reach the end within the allotted number of turns then he loses his rebate. The second is that some of the squares actually have a negative amount which subtract from the rebate instead of adding to it. A particularly unlucky customer might even come out of the game with a negative rebate.

Even with these catches, the management of Gleamin Lemon is concerned that someone might win a particularly large rebatesomething they would like to avoid at all costs. Your job is to take a particular configuration for the RIPOFF game and decide the maximum rebate a customer could possibly obtain. Consider, for example, the game board below. Assume we have 5 turns to finish the game, and each turn we can move between 1 and 4 spaces depending on what we spin. Notice that we must start just before the board begins, so spinning a 1 causes us to land on the first square. Also notice we must end by landing past the end of the last square. It does not have to be exact; any number that gets us off of the board will work.



The illustration shows two different possible ways the game might go. Following the arrows on the top, if we spin a 2, 3, 4, 1, and 1 respectively, we will win a total rebate of 50 + 30 + 20 + 70 = $170. However, the best possible rebate we could win would be $220. We would win this amount if we spun a 1, 3, 2, 4, and 1 respectively, as shown by the lower path. Notice that we did not land on every square with a positive number; if we had we wouldnt have been able to make it to the end of the board before the 5 turns was up.



The illustration in Figure 2 shows a game where we have 4 turns to finish the game, and can move up to 3 spaces each turn. Again, two different paths are shown, the one on top earning a rebate of -$150, and the one on bottom earning a rebate of -$100. In fact, -$100 is the highest possible rebate we could earn for this game (a fact that would no doubt please the management of Gleamin Lemon). Of course, there also might be a sequence of moves in which we do not reach the end before the turn limite.g. spinning a 1 every time. Although not finishing would actually be preferable to finishing with a negative rebate, in this problem we are only going to consider sequences of moves which allow us to reach the end before the turn limit.

Input

The input consists of one to twenty data sets, followed by a line containing only 0.

The first line of a data set contains three space separated integers N S T, where

N is the total number of squares on the board, $2 \leq N \leq 200$.

S is the maximum number of spaces you may advance in each turn, $2 \leq S \leq 10$.

T is the maximum number of turns allowed, where $N + 1 \leq ST$ and $T \leq N + 1$.

The data set ends with one or more lines containing a total of N integers, the numbers on the board. Each number has magnitude less than 10000.

Output

The output for each data set is one line containing only the maximum possible rebate that can be earned by completing the game.

To complete the game you must advance a total of $N + 1$ spaces in at most T turns, each turn advancing from 1 to S spaces inclusive. It will always be possible to complete a game. However, there may be a very large number of different turn sequences that will finish, so you will need to be careful in choosing your algorithm.

The sample input data corresponds to the games in the Figures.

Example

ripoff.in	ripoff.out
10 4 5 100 50 -20 60 30 -10 -30 -50 20 70 9 3 4 150 100 -200 -100 -300 -100 -200 100 150 0	220 -100