ACM International Collegiate Programming Contest

# ARAB COLLEGIATE programming contest

## 2011 Arab Collegiate Programming Contest

Lebanese American University
Beirut, Lebanon
November 29th, 2011

## Instructions

- The contest is composed of ten problems.
- Your work in `/home/`*`username`*`/Desktop/work` will be snapshotted periodically. Please save your work in this directory. Please verify that your IDE or editor saves your work in this directory too.
- The input file name for a problem will be specified in the upper right corner in the first page of each problem, and will be one word, with the extension `in`.
- The output for each problem should be printed out to standard output (console).
- Your Java solution must contain a class that contains the `main` method with the name specified in the problem. This will be the starting point of your program.
- The sample input/output for each problem is in `/home/`*`username`*`/Desktop/work/samples` in the file name the problem specified.
- Please use the Test button in PC^2 before submitting a problem to verify you have followed the instructions correctly.
- To print in Eclipse, please select both the before and after cover pages in the job tab, to be set to "Confidential".

Good Luck!

# [A] Arabic and English

| | |
|---|---|
| Program: | arabic.{cpp\|java} |
| Input: | arabic.in |
| Balloon Color: | Red |

Some computer programs have problems in displaying Arabic text, especially when mixed with English words in the same line, because Arabic is written from right to left and English is written from left to right. In this problem we will try to fix a text with some corrupted lines which consist of a mixture of Arabic and English words. For simplicity, all Arabic letters will be replaced with the letter `'#'`.

Each line will contain at most one English word. For a line containing an English word, the program that will fix the text will swap the words before the English word with the words after the English word. The words before the English word will remain in the same order. The words after the English word will also remain in the same order. For example, if the line is `"# #### ### abc ##"`, it will be fixed to become `"## abc # #### ###"`.

Please note that a line that contains words only of the same language is not corrupt.

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). Next $2T$ lines contain the test cases, each on a pair of lines.

The first line of each case contains a single integer $N$, the number of words in the line to fix ($1 \leq N \leq 100$). The second line contains $N$ words, separated by single spaces, with no leading or trailing spaces, and each word will be at least 1 character and at most 10 characters long.

Each word will be either Arabic or English. Arabic words will consist of one to ten `'#'` letters, and English words will consist of one to ten English lower case letters.

Each line contains at most one English word.

### Output

For each test case, output, on a single line, the fixed line of input text.

### Sample Input/Output

arabic.in

```
3
5
# #### ### abc ##
4
## ### ## #####
4
## ##### # xyz
```

output

```
## abc # #### ###
## ### ## #####
xyz ## ##### #
```

## [B] Between the Mountains

| | |
|---|---|
| Program: | between.{cpp\|java} |
| Input: | between.in |
| Balloon Color: | Green |

Did you try to ride a telepherique (cable car)? It is a lot of fun. Our company is trying to build a new telepherique line between two high mountains and you will be invited for a free ride. The trick to get your free ride is to help the company in building the telepherique line.

The company wants to build two platforms, one on each mountain. The line will extend between these two platforms. The suitable points for holding a platform in each mountain were determined, and the altitudes of these points were reported.

One of the talented engineers suggested that the company can save a lot of energy if the two stations have the same altitude or at least have altitudes which are as close to each other as possible. Your job is to select two points, one at each mountain, from those suitable points, so that the altitude difference between the two points is as little as possible.

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). Follows $2T$ lines containing the test cases, each on a pair of lines.

Each of the two lines in a case describe the altitudes of suitable points to build a platform on one mountain. Each line starts with an integer $N$, the number of reported altitudes ($1 \leq N \leq 1,000$), followed by $N$ integers, which represent the altitudes reported from this mountain. Any altitude value will be between 1 and $1,000,000$, inclusive.

### Output

For each test case, output, on a single line, a single number representing the minimum altitude difference between two suitable platform points, one at each mountain.

### Sample Input/Output

between.in

```
2
8 1 3 5 7 9 7 3 1
8 2 4 6 8 10 8 6 2
8 2 3 5 10 9 3 2 1
7 1 2 6 12 13 3 2
```

output

```
1
0
```

## [C] Circleland

| | |
|---|---|
| `Program:` | `circleland.{cpp|java}` |
| `Input:` | `circleland.in` |
| `Balloon Color:` | `Blue` |

You are visiting circleland, and you have long waited to visit their famous art exhibition. The exhibition has $N$ rooms arranged in a cycle. In every room, there are some artistic pieces. The rooms are named $R_1$, $R_2$, ..., $R_N$. There are also $N$ corridors, named $C_1$, $C_2$, ..., $C_N$, of lengths $L_1$, $L_2$, ..., $L_N$, respectively. Each corridor $C_i$ connects the two rooms $R_i$ and $R_{i+1}$, except $C_N$ which connects $R_N$ and $R_1$. Thus, the whole exhibition forms a cycle. You can walk in both directions in every corridor.

There is a single entrance to the exhibition in room $R_1$, but there are exits in every room. As there is nothing interesting to see in the corridors, you would like to spend the least effort walking through corridors in the exhibition. Compute the minimum total distance you need to walk in corridors such that you enter from the entrance, exit from any exit and visit all rooms.

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). Next $T$ lines contain the test cases, each on a single line.

Each case starts with an integer $N$, the number of rooms in the exhibition ($2 \leq N \leq 100,000$), followed by $N$ numbers, the lengths of the corridors, $L_1$, $L_2$, ..., $L_N$, in this order ($1 \leq L_i \leq 1,000,000$).

The sum of the lengths of all corridors will not exceed $1,000,000,000$.

### Output

For each test case, output, on a single line, a single number representing the minimum total distance you have to walk in corridors such that you visit all rooms.

### Sample Input/Output

circleland.in

```
2
5 1 1 1 1 1
7 100 15 20 42 33 15 24
```

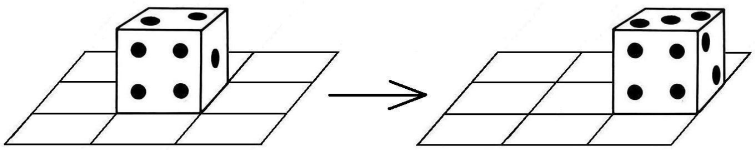
output

```
4
149
```

# [D] Dice on a Board

```
Program:        dice.{cpp|java}
Input:          dice.in
Balloon Color:  Gold
```

You and your friends like to play chess and backgammon every day. But now you are bored of these games, and you would like to play a new game. So you decided to make your own game, which will be played using a backgammon die (singular of dice) on a board similar to the chess board, and it will be a single player game.



The game is played on a board of N rows and M columns. Each cell is either empty or contains a number from 0 to 9, and there is a single die (a die with six faces containing the numbers from 1 to 6) placed in one of the empty cells (the borders of the bottom face is aligned to the axes of the board), and your goal is to move it to a target empty cell.

The initial orientation of the die is defined by a string *S* which is a permutation of the digits from 1 to 6. Each digit represents the number written on a face of the die according to this order: right, left, forward, backward, top, bottom. Moving the die is defined by the following rules:

1. You can move the die from a cell to one of its four adjacent cells by flipping it on the corresponding face. For example, if the current orientation of the die is 136425 and you will move it to the cell on its right, you should flip the die on its right face and it will become the bottom face in the right cell, so the orientation of the die will be 256431. (This is the example in the figure).
2. Your score is initially zero. By moving the die to another cell, if the number on the bottom face is the same as the number in the cell you just moved to, your score will be increased by the sum of these two numbers, otherwise your score will be decreased by the sum of these two numbers. Entering the target cell will not affect your score.
3. You can not leave the board.
4. Once you leave the starting cell, you can not enter it again.
5. Once you enter the target cell, you can not leave it.
6. You can not enter an empty cell, except the target one.

Given the board configuration, the starting cell, the target cell and the die's initial orientation, your task is to move the die from the starting cell to the target cell according the rules above, such that you end up with the maximum possible score. Can you write a program to help you?

**Input**

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 200$). After that follow the specifications of $T$ test cases.

Each test case is specified in $N + 2$ lines. The first line contains two integers $N$ and $M$ ($1 \leq N, M \leq 10$) representing the number of rows and number of columns of the board, respectively. The second line contains the string $S$ representing the initial orientation of the die in the starting cell as described above. Each line of the remaining $N$ lines contains $M$ characters, the j-th character in the i-th line represents the value of the j-th cell in the i-th row of the board. Each character will be one of the following values:

1. `'.'` means an empty cell.
2. `'S'` means the starting cell (which will appear exactly once in the board).
3. `'T'` means the target cell (which will appear exactly once in the board).
4. A digit from `'0'` to `'9'` means the value written in this cell.

**Output**

For each test case, output, on a single line, one of these values:

1. `"Impossible"` if you can not reach the target cell from the starting cell.
2. `"Infinity"` if there is no limit for your final score, and you can increase it with no limit.
3. Otherwise, output the maximum score which you can get.

**Sample Input/Output**

dice.in

```
3
4 4
123456
S789
0987
789.
09.T
3 6
153462
S16521
.46324
.....T
4 4
623451
S6T.
....
....
....
```

output

```
Impossible
Infinity
12
```

## [E] Error

| | |
|---|---|
| `Program:` | `error.{cpp|java}` |
| `Input:` | `error.in` |
| `Balloon Color:` | `Orange` |

Your boss is very upset. He just discovered that his coming flight to the US was booked wrong. Your boss wanted to do his trip using the least number of transit stops. The secretary have booked flight segments that do not meet this condition. Given that the segments are already booked, your boss decided that the resolution of this issue needs you, the programmer.

You are given the possible flight connections between pairs of cities where each connection can be used in both directions. You are also given the set of faulty segments that were booked by the secretary. Your goal is to book the least number of additional segments, which, when combined with the already booked segments, can enable your boss the flexibility to reach his destination in the least number of stops.

The segments booked by the secretary are all flexible in timing and can be used in any order. Note that you can use some, all or none of the segments already booked, in addition to the segments you will add. However, an already booked segment from $A$ to $B$ can only be used from $A$ to $B$ and not from $B$ to $A$. See examples for further clarification.

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). After that follow the specifications of $T$ test cases.

Each case is specified in $N + 2$ lines. Each case starts with a line containing an integer $N$ ($1 \leq N \leq 100,000$), the number of available direct flight connections. Each of the following $N$ lines contain the names of two different airports between which a direct flight connection exists and can be used in both directions. Every airport name is composed of exactly three capital letters. In each case, there will be at most one available direct flight connection between any pair of airports. The last line in each case contains the faulty ticket booked by the secretary.

The ticket is in the form "$M$ $A_1$ $A_2$ ... $A_{M+1}$", where $M$ is the number of flight segments already booked, and $A_i$ is an airport name (composed of three capital letters). Any two consecutive airport names in this line mean there is a flight segment booked from the earlier airport to the latter airport. The segments booked were so faulty, that there can be cycles and even multiple bookings of the same flight segment. However, the segments are from the available flight connections (which are described in the $N$ lines above).

**Output**

For each test case, output, on a single line, a single number representing the minimum number of flight segments that have to be booked, such that, when combined with the already booked segments, the final ticket can be used to go from airport $A_1$ to airport $A_{M+1}$ in the least number of segments.

**Sample Input/Output**

error.in

```
3
5
LHR JFK
CAI LHR
FRA CAI
FRA JFK
FRA LHR
3 CAI FRA LHR JFK
5
CAI FRA
FRA LHR
LHR CDG
CDG FRA
LHR JFK
7 CAI FRA CDG LHR FRA CDG LHR JFK
3
CAI FRA
FRA JFK
CAI JFK
1 CAI JFK
```

output

```
1
1
0
```

In the first case, adding the segment FRA-JFK or CAI-LHR will enable the boss to go from CAI to JFK in one stop, which is the least number of stops to go from CAI to JFK.

In the second case, the least number of stops can be achieved only using the path CAI-FRA-LHR-JFK. However, the secretary had the segment LHR-FRA booked and not the other way around, so you have to add FRA-LHR to achieve the least number of stops.

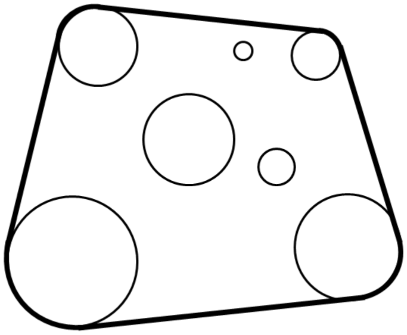
In the third case, the secretary booked an optimal ticket so you do not need to add any segments.

# [F] Fence

```
Program:        fence.{cpp|java}
Input:          fence.in
Balloon Color:  Silver
```

You have a garden with trees and goats. Since you like to protect the trees from the goats, you decided to build a single fence around the trees, and you like this fence to be as short as possible.

In the top view of your garden trees can be modeled by circles in 2D space. In this view, the required fence will be a 2D shape that touch some or all of the trees, such that all the trees are inside this shape. For example, in the figure, the trees are shown as the circles, and the shortest possible fence is shown as the bold surrounding line.



### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 250$). After that follow the specifications of $T$ test cases.

Each case is specified in $N+1$ lines. Each case starts with a line containing an integer $N$, the number of trees in the garden ($1 \leq N \leq 100$), followed by $N$ lines, each in the format "$X$ $Y$ $R$", where $X$ and $Y$ are the 2D coordinates of the center of the tree, and $R$ is the radius of the tree.

$X$, $Y$, and $R$ are all integers between 1 and 1,000, inclusive. No trees in the input touch or intersect each other.

### Output

For each test case, output, on a single line, a single number representing the minimum length of the shortest possible fence that can surround all the trees, rounded to five decimal places.

### Sample Input/Output

fence.in

```
4
5
3 3 2
12 12 2
3 12 2
12 3 2
8 8 1
3
3 3 2
9 3 2
6 7 2
2
5 5 1
10 10 2
1
5 5 10
```

output

```
48.56637
28.56637
23.70857
62.83185
```

# [G] Go

```
Program:        go.{cpp|java}
Input:          go.in
Balloon Color:  Purple
```

Go is an ancient board game for two players that originated in China over 2,000 years ago. The game is notably known for being rich in strategy despite its relatively simple rules.

The game is played by two players who alternately place black and white stones on the vacant intersections of a grid of $N \times N$ orthogonal lines. The objective of the game is to use one's stones to make a larger territory than the opponent.

The horizontal lines are numbered from top to bottom by the numbers from 1 to $N$, and the vertical lines are numbered from left to right by the numbers from 1 to $N$.

Two intersections or stones are adjacent if one of them is adjacent to the other, from right, left, top or bottom. A *"solidly connected group"* is a set of intersections or stones in which every pair is connected by a path of adjacent intersections or stones from the same set.

A solidly connected group of empty intersections or stones is called *surrounded* by a color, if all the adjacent intersections to the group are occupied by stones of that color.

A player's territory consists of all the intersections occupied by the player's stones, in addition to the solidly connected groups of empty intersections surrounded by the player's stone color.

Here is a list of the rules for a simplified version of the game:

1. The board is empty at the beginning of the game.
2. Each player has stones of one color (black or white).
3. Black gets the first turn, then the players alternate turns.
4. A move consists of placing one stone of one's own color on an empty intersection on the board.
5. A player may pass his turn at any time and give the turn to the other player.
6. A solidly connected group of stones of one color is captured and removed from the board once it is surrounded by the opponent.
7. Self-capture happens when your move causes some of your stones to be captured and thus removed.
8. In case of a move causing a group from both players to be captured, capture of the opponent takes precedence over self-capture.

You are given the size of the board and a sequence of moves, and your task is to validate this sequence of moves and print the index (1-based) of the first invalid move (a move is invalid if the player is trying to put a stone in a non-empty intersection). If all moves are valid you should print the sizes of the territories for each player after the last move.

**Input**

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). After that follow the specifications of $T$ test cases.

Each test case is specified on $S + 1$ lines. The first line contains two integers $N$ and $S$ representing the size of the board and the number of moves to validate, respectively ($1 \leq N \leq 20$, $1 \leq S \leq 1,000$). Each line of the remaining $S$ lines will contain either two zeros which means a pass turn, or two integers $R$ and $C$ specifying the horizontal and vertical line numbers of the player's move ($1 \leq R, C \leq N$).

**Output**

For each test case, output, on a single line, one of two outputs:

- "Invalid $X$", if there is at least one invalid move, where $X$ is the index of the earliest invalid move. Please note that passing the turn to the other player does not count as a move.
- "$B$ $W$", if there are no invalid moves, where $B$ and $W$ are the sizes of the black and white territories, respectively.

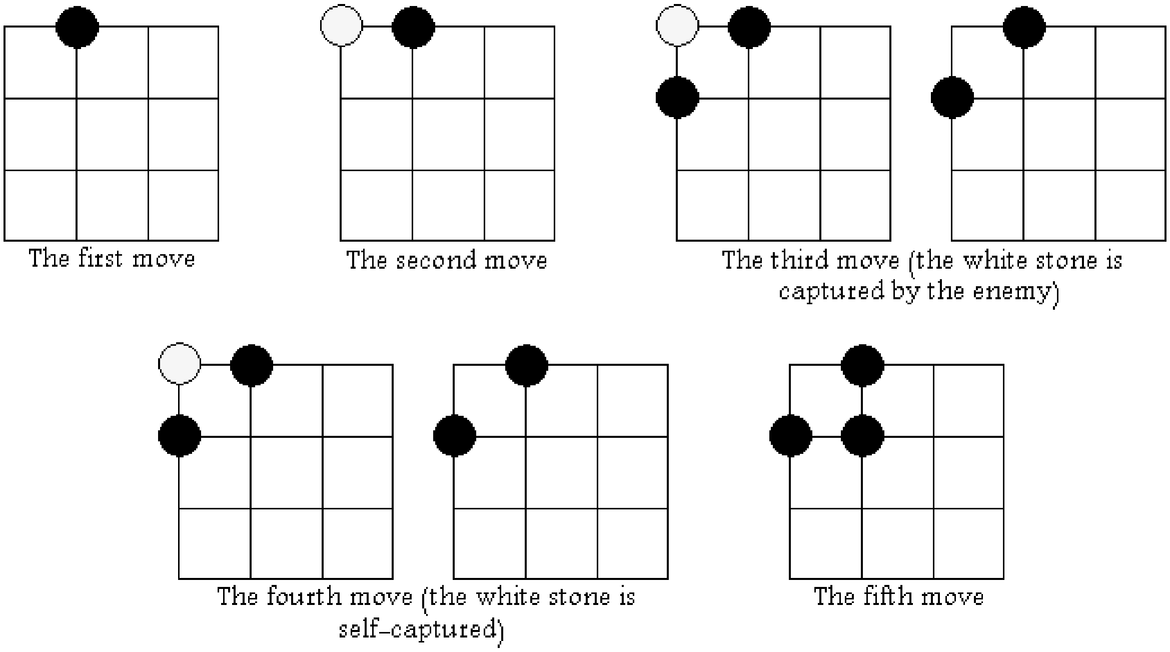
**Sample Input/Output**

go.in

```
3
4 4
1 1
2 2
3 3
4 4
4 5
1 2
1 1
2 1
1 1
2 2
4 8
1 2
1 1
0 0
0 0
2 2
2 1
4 4
1 1
```

output

```
2 2
16 0
Invalid 6
```



The first move

The second move

The third move (the white stone is captured by the enemy)

The fourth move (the white stone is self-captured)

The fifth move

Explanation for the second test case

## [H] Homework

| | |
|---|---|
| `Program:` | `homework.{cpp|java}` |
| `Input:` | `homework.in` |
| `Balloon Color:` | `Pink` |

Now it is time for your math homework. A Tornado operation $T(n)$ is defined as follows:

$$T(n) = \begin{cases} a & \text{, where } n = 0 \\ [T(n-1) + X_n]^{Y_n} & \text{, } \forall n \in \mathbb{Z}^+ \end{cases}$$

$a$ is a given constant, and $\mathbb{Z}^+$ is the set of positive integers. $X_n$ and $Y_n$ are positive integers chosen such that $X_n \leq X_{n+1}$ and $Y_n \leq Y_{n+1}$, for all positive $n$. Also, $minX \leq X_n \leq maxX$ and $minY \leq Y_n \leq maxY$, for all positive $n$.

For example if $a = 1$, $X_1 = 2$, $X_2 = 4$, and $Y_1 = Y_2 = 3$, then:
$T(2) = [T(1) + X_2]^{Y_2} = [(T(1) + X_1)^{Y_1} + X_2]^{Y_2} = [(1+2)^3 + 4]^3 = 29,791$.

Given $a$, $minX$, $maxX$, $minY$, $maxY$ and two positive integers $P$ and $C$, your homework is to find the minimum value of $n$ such that $T(n) + C$ is divisible by $10^P$, by choosing appropriate values for $X_1$, ..., $X_n$ and $Y_1$, ..., $Y_n$.

### Input

Your program will be tested on one or more test cases. The first line of the input will contain a single integer $T$, the number of test cases ($1 \leq T \leq 200$). Next $T$ lines contain the test cases, each on a single line.

Each of those lines will contain 7 integers, $a$, $minX$, $maxX$, $minY$, $maxY$, $P$ and $C$, separated by single spaces, given in this order ($1 \leq minX, maxX, minY, maxY \leq 100$, $1 \leq P \leq 3$, $1 \leq a, C \leq 1,000,000$).

### Output

For each test case, output, on a single line, a single integer representing the minimum value for $n$ such that $T(n) + C$ is divisible by $10^P$. If there is no such value, output -1 instead.

### Sample Input/Output

homework.in

```
4
4 1 1 1 2 1 5
4 1 100 1 100 1 6
3 1 1 2 2 2 11
1 2 2 1 1 3 2
```

output

```
1
0
2
-1
```

## [I] Identify the Number

| | |
|---|---|
| `Program:` | `identify.{cpp|java}` |
| `Input:` | `identify.in` |
| `Balloon Color:` | `Black` |

You are given three numbers $N$, $Q$ and $R$. You want to find $M$, such that:

- $M$ is positive.
- The string decimal representation of $M$ is a subsequence of the string decimal representation of $N$, i.e. $M$ can be obtained from the removal of zero or more digits from the decimal representation of $N$.
- $M$ gives a remainder of $R$ when divided by $Q$.
- $M$ is the maximum possible.

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 200$). Next $T$ lines contain the test cases, each on a single line.

Each case contains three integers, separated by single spaces, $N$, $R$, $Q$ as described in the problem, in this order ($1 \leq N < 10^{1,000}$, $0 \leq R < Q \leq 1,000$). All numbers in the input will not contain leading zeros.

### Output

For each test case, output, on a single line, the number $M$ as described in the problem, with no leading zeros. If no such $M$ can be found, output on a single line `"Not found"` instead.

### Sample Input/Output

identify.in

```
3
840 0 8
901 3 8
123456789 10 100
```

output

```
840
91
Not found
```

In the first case, 840 is divisible by 8, thus it is the largest possible value of $M$.

In the second case, the subsequences of 901 are $\{9, 0, 1, 90, 01, 91, 901\}$. Out of these there is 0 which is not positive and 01 which has a leading zero. Only 91 has a remainder of 3 when divided by 8.

In the third case, there is no subsequence of 123456789 that gives a remainder of 10 when divided by 100.

## [J] Joy of the Cylinder Game

```
Program:        joy.{cpp|java}
Input:          joy.in
Balloon Color:  White
```

The cylinder game is new and not published anywhere yet. The inventor of this game is interested in writing a solver for the game before publishing it, and he needs your help to write this solver.

The game is played on a cylinder. This cylinder is divided into $N$ rows numbered from 1 to $N$ (the first row is the top-most one), and each row is divided into $M$ cells numbered from 1 to $M$, and each cell contains a number. The cell number 1 is adjacent to the cell number $M$ in the same row (check the image for more clarification).

Each cell will be defined by two numbers $X$ and $Y$, where $X$ is the row number and $Y$ is the cell number in this row. The distance between two cells $(X_1, Y_1)$ and $(X_2, Y_2)$ is the absolute difference between $X_1$ and $X_2$ added to the minimum of (the absolute difference between $Y_1$ and $Y_2$) and ($M$ - the absolute difference between $Y_1$ and $Y_2$).

The first step in the game is to select any cell from the first row to start from, then you will move from the current cell to any cell in the next row, without exceeding a distance of $K$. You will keep doing this move until you reach the last row. Your final score will be the sum of the numbers inside the cells you selected in your steps. The goal of the game is to maximize this score.

Can you help by writing a solver for this game?

### Input

Your program will be tested on one or more test cases. The first line of the input will be a single integer $T$, the number of test cases ($1 \leq T \leq 100$). After that follow the specifications of $T$ test cases.

Each case is specified on $N+1$ lines. The first line of a case contains three integer $N$, $M$ and $K$, representing the number of rows, number of cells in each row and the maximum allowed distance in every move, respectively ($1 \leq N, M, K \leq 1,000$).

Each line of the remaining $N$ lines contains $M$ numbers. The $j$-th number in the $i$-th line is the number in the cell $(i, j)$. The absolute value of all the given numbers is at most $1,000$.

### Output

For each test case, you must print one line of output that starts by the maximum score you can get, followed by a single space, followed by $N$ numbers separated by single spaces, where the $i$-th number is the index of the selected cell in the row number $i$.

If there are multiple solutions that reach the same maximum score, you should print the lexicographically smallest one. (A solution $X$ is defined as lexicographically smaller than a solution $Y$ if $X$ has a smaller number than $Y$ at the first position where they differ).

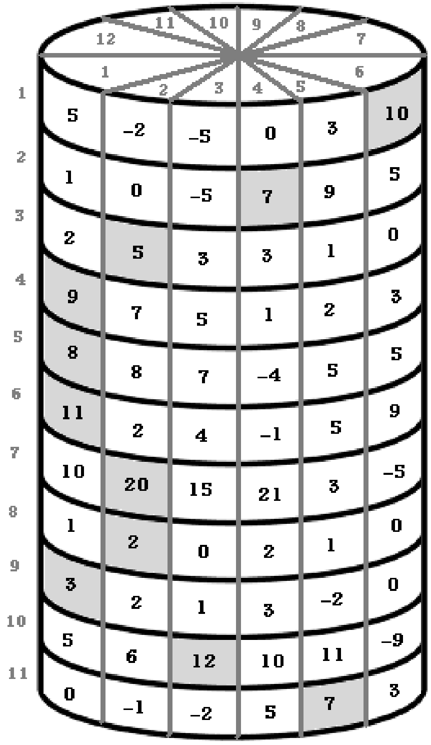
### Sample Input/Output

joy.in

```
1
11 12 3
5 -2 -5 0 3 10 0 0 0 0 0 0
1 0 -5 7 9 5 0 0 0 0 0 0
2 5 3 3 1 0 0 0 0 0 0 0
9 7 5 1 2 3 0 0 0 0 0 0
8 8 7 -4 5 5 0 0 0 0 0 0
11 2 4 -1 5 9 0 0 0 0 0 0
10 20 15 21 3 -5 0 0 0 0 0 0
1 2 0 2 1 0 0 0 0 0 0 0
3 2 1 3 -2 0 0 0 0 0 0 0
5 6 12 10 11 -9 0 0 0 0 0 0
0 -1 -2 5 7 3 0 0 0 0 0 0
```

output

```
94 6 4 2 1 1 1 2 2 1 3 5
```



|  | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 5 | -2 | -5 | 0 | 3 | 10 |
| 2 | 1 | 0 | -5 | 7 | 9 | 5 |
| 3 | 2 | 5 | 3 | 3 | 1 | 0 |
| 4 | 9 | 7 | 5 | 1 | 2 | 3 |
| 5 | 8 | 8 | 7 | -4 | 5 | 5 |
| 6 | 11 | 2 | 4 | -1 | 5 | 9 |
| 7 | 10 | 20 | 15 | 21 | 3 | -5 |
| 8 | 1 | 2 | 0 | 2 | 1 | 0 |
| 9 | 3 | 2 | 1 | 3 | -2 | 0 |
| 10 | 5 | 6 | 12 | 10 | 11 | -9 |
| 11 | 0 | -1 | -2 | 5 | 7 | 3 |

This is the sample test case. The unviewable cells (behind) are all zeros, and the best solution is highlighted