Problem A (Program file: A.CPP or A.PAS)

(Your)((Term)((Project)))

You have typed the report of your term project in your personal computer. There are several one line arithmetic expressions in your report. There is no redundant parentheses in the expressions (omitting a pair of redundant matching parentheses does not change the value of the expression). In your absence, your little brother inserts some redundant matching parentheses in the expressions of your report. Assume that the expressions remain syntactically correct and evaluate to their original value (the value before inserting redundant parentheses). To restore your report to its original form, you are to write a program to omit all redundant parentheses.

To make life easier, consider the following simplifying assumptions:

1. The input file contains a number of expressions, each in one separate line.
2. Variables in the expressions are only single uppercase letters.
3. Operators in the expressions are only binary '+' and binary '-'.

Note that the only transformation allowed is omission of redundant parentheses, and no algebraic simplification is allowed.

Input (File: A.IN)

The input file consists of several test cases. The first line of the file contains a single number M, which is the number of test cases ($1 \le M \le 10$). Each of the following M lines, is exactly one correct expression. There may be arbitrarily space characters in each line. The length of each line (including spaces) is at most 255 characters.

Output (File: A.OUT)

The output for each test case is the same expression without redundant parentheses. Notice that the order of operands in an input expression and its corresponding output should be the same. Each output expression must be on a separate line. Space characters should be omitted in the output expressions.

Sample Input

```
3
(A-B + C) - (A+(B – C)) - (C-(D- E) )
  ((A)-( (B)))
A-(B+C)
```

Sample output

```
A-B+C-(A+B-C)-(C-(D-E))
A-B
A-(B+C)
```

Problem B (Program file: B.CPP or B.PAS)

Painting A Board

The CE digital company has built an Automatic Painting Machine (APM) to paint a flat board fully covered by adjacent non-overlapping rectangles of different sizes each with a predefined color.

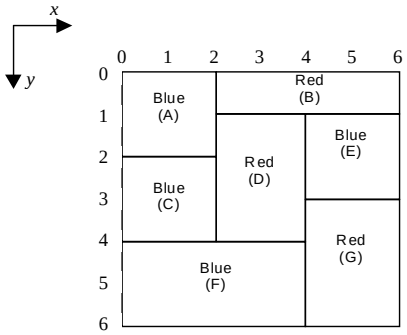


Figure 1

To color the board, the APM has access to a set of brushes. Each brush has a distinct color *C*. The APM picks one brush with color *C* and paints all possible rectangles having predefined color *C* with the following restrictions:

To avoid leaking the paints and mixing colors, a rectangle can only be painted if all rectangles immediately above it have already been painted. For example rectangle labeled F in Figure 1 is painted only after rectangles C and D are painted. Note that each rectangle must be painted at once, i.e. partial painting of one rectangle is not allowed.

You are to write a program for APM to paint a given board so that the number of brush pick-ups is minimum. Notice that if one brush is picked up more than once, all pick-ups are counted.

Input (File: B.IN)

The first line of the input file contains an integer *M* which is the number of test cases to solve ($1 \le M \le 10$). For each test case, the first line contains an integer *N*, the number of rectangles, followed by *N* lines describing the rectangles. Each rectangle *R* is specified by 5 integers in one line: the ***y*** and ***x*** coordinates of the upper left corner of *R*, the ***y*** and ***x*** coordinates of the lower right corner of *R*, followed by the color-code of *R*.

Note that:

1. Color-code is an integer in the range of 1 .. 20.
2. Upper left corner of the board coordinates is always (0,0).
3. Coordinates are in the range of 0 .. 99.
4. *N* is in the range of 1..15.

Output (File: B.OUT)

One line for each test case showing the minimum number of brush pick-ups.

Sample Input

```
1
7
0 0 2 2 1
0 2 1 6 2
2 0 4 2 1
1 2 4 4 2
1 4 3 6 1
4 0 6 4 1
3 4 6 6 2
```

Sample Output

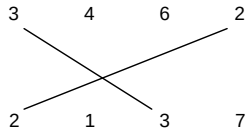
```
3
```

Note : This sample input corresponds to Figure 1.

Problem C (Program file: C.CPP or C.PAS)

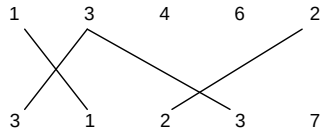
Crossed Matchings

There are two rows of positive integer numbers. We can draw one line segment between any two equal numbers, with values *r*, if one of them is located in the first row and the other one is located in the second row. We call this line segment an *r*-matching segment. The following figure shows a 3-matching and a 2-matching segment.



We want to find the maximum number of matching segments possible to draw for the given input, such that:

1. Each *a*-matching segment should cross exactly one *b*-matching segment, where $a \neq b$.
2. No two matching segments can be drawn from a number. For example, the following matchings are not allowed.



Write a program to compute the maximum number of matching segments for the input data. Note that this number is always even.

Input (File: C.IN)

The first line of the file is the number *M*, which is the number of test cases ($1 \leq M \leq 10$). Each test case has three lines. The first line contains N_1 and N_2, the number of integers on the first and the second row respectively. The next line contains N_1 integers which are the numbers on the first row. The third line contains N_2 integers which are the numbers on the second row. All numbers are positive integers less than 100.

Output (File: C.OUT)

Output file should have one separate line for each test case. The maximum number of matching segments for each test case should be written in one separate line.

Sample Input

```
3
6 6
1 3 1 3 1 3
3 1 3 1 3 1
4 4
1 1 3 3
1 1 3 3
12 11
1 2 3 3 2 4 1 5 1 3 5 10
3 1 2 3 2 4 12 1 5 5 3
```

Sample Output

```
6
0
8
```

Problem D (Program file: D.CPP or D.PAS)

Counting Rectangles

We are given a figure consisting of only horizontal and vertical line segments. Our goal is to count the number of all different rectangles formed by these segments. As an example, the number of rectangles in the Figures 1 and 2 are 5 and 0 respectively.

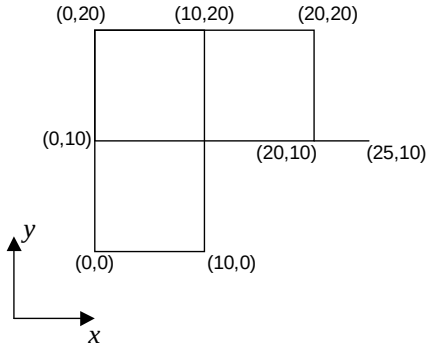


Figure 1

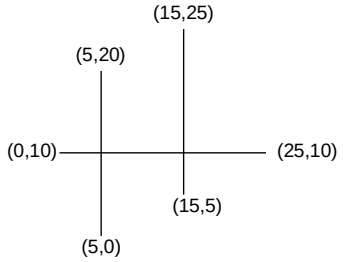


Figure 2

There are many intersection points in the figure. An intersection point is a point shared by at least two segments. The input line segments are such that each intersection point comes from the intersection of exactly one horizontal segment and one vertical segment.

Input (File: D.IN)

The first line of the file contains a single number M, which is the number of test cases in the file ($1 \le M \le 10$), and the rest of the file consists of the data of the test cases. Each test case begins with a line containing s ($1 \le s \le 100$), the number of line segments in the figure. It follows by s lines, each containing x and y coordinates of two end points of a segment respectively. The coordinates are integers in the range of 0 to 1000.

Output (File: D.OUT)

The output for each test case is the number of all different rectangles in the figure described by the test case. The output for each test case must be written on a separate line.

Sample input

```
2
6
0 0 0 20
0 10 25 10
20 10 20 20
0 0 10 0
10 0 10 20
0 20 20 20
3
5 0 5 20
15 5 15 25
0 10 25 10
```

Sample output

```
5
0
```

The above input file contains two test cases corresponding to Figures 1 and 2 respectively.

Problem E (Program file: E.CPP or E.PAS)

An Old Stone Game

There is an old stone game, played on an arbitrary general tree *T*. The goal is to put one stone on the root of *T* observing the following rules:

1. At the beginning of the game, the player picks *K* stones and puts them all in one bucket.
2. At each step of the game, the player can pick one stone from the bucket and put it on any empty leaf.
3. When all of the *r* immediate children of a node *p* each has one stone, the player may remove all of these *r* stones, and put one of the stones on *p*. The other *r* - 1 stones are put back into the bucket, and can be used in the later steps of the game.

The player wins the game if by following the above rules, he succeeds to put one stone on the root of the tree.

You are to write a program to determine the least number of stones to be picked at the beginning of the game (*K*), so that the player can win the game on the given input tree.

Input (File: E.IN)

The input file describes several trees. The first line of this file is *M*, the number of trees ($1 \le M \le 10$). Description of these *M* trees comes next in the file. Each tree has $N < 200$ nodes, labeled 1, 2, ... *N*, and each node can have any possible number of children. Root has label 1. Description of each tree starts with *N* in a separate line. The following *N* lines describe the children of all nodes in order of their labels. Each line starts with a number *p* ($1 \le p \le N$, the label of one of the nodes), *r* the number of the immediate children of *p*, and then the labels of these *r* children.

Output (File: E.OUT)

One line for each input tree showing the minimum number of stones to be picked in step 1 above, in order to win the game on that input tree.

Sample Input

```
2
7
1 2 2 3
2 2 5 4
3 2 6 7
4 0
5 0
6 0
7 0
12
1 3 2 3 4
2 0
3 2 5 6
4 3 7 8 9
5 3 10 11 12
6 0
7 0
8 0
9 0
10 0
11 0
12 0
```

Sample Output

```
3
4
```

Problem F (Program file: F.CPP or F.PAS)

Magazine Delivery

The TTT Taxi Service in Tehran is required to deliver some magazines to N locations in Tehran. The locations are labeled L_1 to L_N. TTT assigns 3 cars for this service. At time 0, all the 3 cars and magazines are located at L_1. There are plenty of magazines available in L_1 and the cars can take as many as they want. Copies of the magazine should be delivered to all locations, observing the following rules:

1. For all $i = 2 .. N$, magazines should be delivered at L_i only after magazines are delivered at L_{i-1}.
2. At any time, only one of the three cars is driving, and the other two are resting in other locations.

The time to go from L_i to L_j (or reverse) by any car is a positive integer denoted by $D[i , j]$.

The goal is to organize the delivery schedule for the cars such that the time by which magazines are delivered to all N locations is minimum.

Write a program to compute the minimum delivery time.

Input (File: F.IN)

The input file contains M instances of this problem ($1 \le M \le 10$). The first line of the input file is M. The descriptions of the input data follows one after the other. Each instance starts with N in a single line ($N \le 30$). Each line i of the following $N-1$ lines contains $D[i , j]$, for all $i=1..N-1$, and $j=i+1..N$.

Output (File: F.OUT)

The output file contains M lines, each corresponding the solution to one of the input data. In each line, the minimum time it takes to deliver the magazines to all N locations is written.

Sample Input

```
1
5
10 20 3 4
5 10 20
8 18
19
```

Sample Output

```
22
```

Problem G (Program file: G.CPP or G.PAS)

Space Ant

The most exciting space discovery occurred at the end of the 20th century. In 1999, scientists traced down an ant-like creature in the planet Y1999 and called it M11. It has only one eye on the left side of its head and just three feet all on the right side of its body and suffers from three walking limitations:

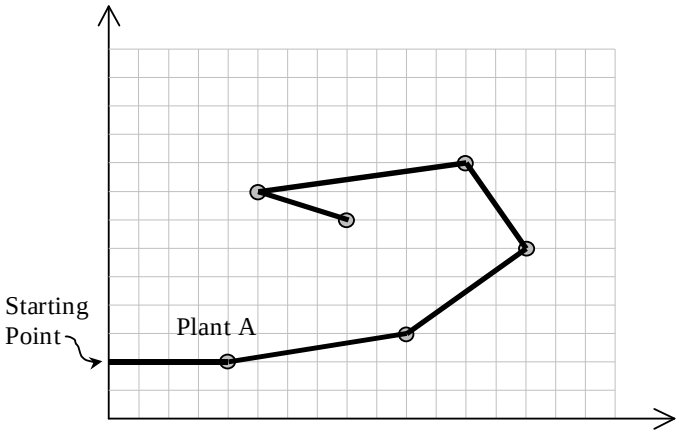
1. It can not turn right due to its special body structure.
2. It leaves a red path while walking.
3. It hates to pass over a previously red colored path, and never does that.

The pictures transmitted by the Discovery space ship depicts that plants in the Y1999 grow in special points on the planet. Analysis of several thousands of the pictures have resulted in discovering a magic coordinate system governing the grow points of the plants. In this coordinate system with x and y axes, **no two plants share the same x or y**.

An M11 needs to eat exactly one plant in each day to stay alive. When it eats one plant, it remains there for the rest of the day with no move. Next day, it looks for another plant to go there and eat it. If it can not reach any other plant it dies by the end of the day. Notice that it can reach a plant in any distance.

The problem is to find a path for an M11 to let it live longest.

Input is a set of (x, y) coordinates of plants. Suppose A with the coordinates (x_A, y_A) is the plant with the least y-coordinate. M11 starts from point $(0,y_A)$ heading towards plant A. Notice that the solution path should not cross itself and all of the turns should be counter-clockwise. Also note that the solution may visit more than two plants located on a same straight line.



Input (File: G.IN)

The first line of the file is *M*, the number of test cases to be solved ($1 \le M \le 10$). For each test case, the first line is *N*, the number of plants in that test case ($1 \le N \le 50$), followed by *N* lines for each plant data. Each plant data consists of three integers: the first number is the unique plant index (*1..N*), followed by two positive integers *x* and *y* representing the coordinates of the plant. Plants are sorted by the increasing order on their indices in the input file. Suppose that the values of coordinates are at most 100.

Output (File: G.OUT)

Output file should have one separate line for the solution of each test case. A solution is the number of plants on the solution path, followed by the indices of visiting plants in the path in the order of their visits.

Sample Input

```
2
10
1 4 5
2 9 8
3 5 9
4 1 7
5 3 2
6 6 3
7 10 10
8 8 1
9 2 4
10 7 6
14
1 6 11
2 11 9
3 8 7
4 12 8
5 9 20
6 3 2
7 1 6
8 2 13
9 15 1
10 14 17
11 13 19
12 5 18
13 7 3
14 10 16
```

Sample Output

```
10 8 7 3 4 9 5 6 2 1 10
14 9 10 11 5 12 8 7 6 13 4 14 1 3 2
```

Problem H (Program file: H.CPP or H.PAS)

The Erythea Campaign

O' mighty warrior,

Thy mission is to slay the foul king of Erythea.
Thou shall find him in his realm in the south.

God bless you,
King of Isladia.

After reading the order, you know you have a long, dangerous way down to the south, to find the king of Erythea in his realm and kill him. The Erythea realm is a rectangular region, with a number of horrible strongholds in it. Impenetrable walls enclose the region, so the only way for you to enter the realm is to fly by your Pegasus (flying horse) and land in some point in the region. The hiding place of the king is known so you just need to find your way to that location. As the area is extensive and covered by different terrain types, you have to travel on the grid-like roads in the region. The problem is, there are many guards on the towers of strongholds who can see you, and once seen, you have no chance to see your family again! The closer you travel to the strongholds, the greater the chances to be seen by the guards. The problem is to find the safest way from the point your Pegasus lands to the point where the king is.

More abstractly, you have an $m \times n$ grid with squares of the same size, denoting the realm and the roads in it. You can travel along the lines in the grid (roads), and at each intersection (road crossing) you may turn into another road. Assume each stronghold comprises a set of adjacent squares of the grid (Figure 1). As you cannot enter a stronghold, your path never intersects the interior of a stronghold, yet you can travel on a road which is on stronghold boundaries (Figure 2). Suppose you can land your Pegasus exactly on a road crossing (the source point – S in Figure 1) and the hiding place of the king is on another road crossing (the destination point – D in Figure 1). Neither of these points lie inside a stronghold but may be placed on a stronghold boundary (as D does in Figure 1). Each road crossing is assigned a risk level which depends on the shortest road distance from the crossing to a point of the grid which is on the boundary of a stronghold. For a road crossing with the shortest road distance d to a boundary of strongholds, the risk level equals to $m + n - d$ (Figure 3). It is assumed that **there is at least one stronghold in the region**, so that the definition of risk level is well-defined. The problem is, given the region's map and the source and destination points, find the path from the source to the destination which lies on the grid lines, so that sum of the risk levels of the points on the path (including source and destination) be minimum. As stated before, the path cannot intersect the interior of a stronghold.

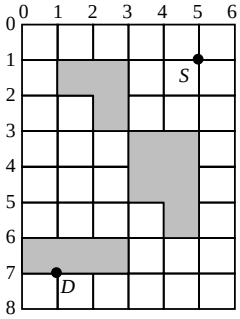


Figure 1. *An* 8×6 *region. You can travel along the grid lines (including boundary lines). The shaded squares are strongholds.*

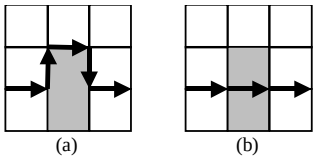


Figure 2. (a) *Traveling on the boundary of a stronghold is allowed.*
(b) *Crossing a stronghold is not allowed.*

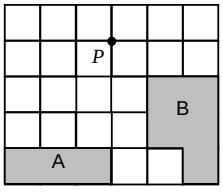


Figure 3. *Point P in a* 5×6 *grid has distance* 3 *from stronghold A and distance* 2 *from stronghold B. Thus, risk level of P is* (5+6)-2 = 9.

Input (File: H.IN)

The input file consists of several test cases. The first line of the file contains a single number M, which is the number of test cases in the file ($1 \le M \le 10$). The rest of the file consists of the data of the test cases. Each test case data begins with a line containing the number of rows and the number of columns in the grid, which are in the range 1 to 80. The second line of a test case contains two pairs of integers, which are ***y* and *x*** coordinates of

the source point (where the Pegasus landed) and the **y** and **x** coordinates of the destination point (where you may find the king). The horizontal and vertical lines in the grid are indexed from left to right and top to bottom from 0, so the coordinates can be expressed using these indices.

Following the first two lines, there are lines that describe the map of the region. Each line consists of a string of 0's and 1's, describing squares of the corresponding row. A 1 in the string tells you that the corresponding square in the grid belongs to a stronghold. The width of the region is the length of all strings, and its height in the number of strings.

Output (File: H.OUT)

The output for each test case is the total risk of the minimum risk path from the landing point to the destination. Recall that the total risk of a path is sum of the risk levels of the points in the path (including source and destination). If no path exists between source and destination, the output should be '`no solution`'. The output for each test case must be written on a separate line.

Sample input

```
2
8 6
1 5 7 1
000000
011000
001000
000110
000110
000010
111000
000000
5 5
4 0 1 5
10000
10000
11111
00011
00001
```

Sample output

```
149
101
```

The above input file contains two test cases. The first test case is the one shown in Figure 1.