LAB-PRAC-12_FUN-STRUC

Point Pairing Party (p1v1d1)

Point Pairing Party [20 marks]

Problem Statement
The first line of the input will give you n, a strictly positive integer. The next n lines will give you n pairs of points on the 2D plane. Each point will have two integer coordinates, separated by a space. The two points in the pair will also be separated by a space.

Naturally, each pair of these points corresponds to a unique line which passes through both the points. In your output you have to print how many pairs of these points correspond to lines that are parallel, that intersect at a single point, and that are identical (i.e. infinite points of intersection). Give your output in the format described below.

HINT: You would find it a bit cumbersome to manage the coordinates for all the points in separate arrays. Instead, you should use structures to make your code easier to read as well as easier to debug. Define a structure for a 2D point which stores two integers, corresponding to the x and y coordinates of the point. Define another structure for a pair of 2D points. Now you can define an array of these pairs and work with them. An example is given below

```
struct point{
int x, y
};

struct pair{
struct point p1, p2;
};
```

Caution

1. The question may require you to compare floating point numbers like slopes etc. Since comparing floating point numbers for equality is dangerous, we will consider two floating point numbers to be the same if their difference in absolute terms is less than 0.0001. Use the fabs() function from math.h to get the absolute value of floating point numbers.
2. Do not make spelling, capitalization or space errors. Look at the output format closely below.
3. Be careful about divide-by-zero errors. Some of the lines may be horizontal or vertical.
4. Count each pair only once. Do not make double counting errors.
5. We assure you that there will be no pair of points were both points are the same since such pairs do not correspond to a unique line.
6. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
```
2
1 1 0 0
1 0 1 4
```

OUTPUT:
PARALLEL: 0
INTERSECT: 1
IDENTICAL: 0

Explanation: The first line passes through the points (1, 1) and (0, 0); the second line passes through the points (1, 0) and (1, 4). Both the lines intersect at (1, 1). Thus, there are no pairs of parallel lines, 1 pair of intersecting lines and no pairs of identical lines.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are three lines in your output. Printing each line correctly, in the correct order, carries 33% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

2
1 0 1 1
0 1 1 1
	PARALLEL: 0
	INTERSECT: 1
	IDENTICAL: 0

5
0 1 1 2
0 2 1 3
0 3 1 4
0 4 1 5
0 5 1 6
	PARALLEL: 10
	INTERSECT: 0
	IDENTICAL: 0

2
1 0 1 1
2 0 2 1
	PARALLEL: 1
	INTERSECT: 0
	IDENTICAL: 0

4
1 0 1 1
0 0 1 1
1 0 1 1
2 0 2 1
	PARALLEL: 2
	INTERSECT: 3
	IDENTICAL: 1

8
1 0 1 1
2 0 2 1
3 0 3 1
4 0 4 1
0 1 1 2
1 2 2 3
2 3 3 4
3 4 4 5
	PARALLEL: 6
	INTERSECT: 16
	IDENTICAL: 6

8
0 2 1 1
2 0 1 1
	PARALLEL: 24
	INTERSECT: 0
	IDENTICAL: 4

```
0 4 2 2
4 0 2 2
0 6 3 3
6 0 3 3
0 8 4 4
8 0 4 4
```

Verify the family tree of Mr C (p1v2d1)

Verify the family tree of Mr C [20 marks]

Problem Statement
Mr C wants to verify if his family tree has any errors or not. In the first line of the input, you will be given a strictly positive integer n, denoting the number of family members. In the next n lines, you will be given the ages of these n family members. All ages will be non-negative numbers, with all ages on a separate line. Store these ages in an array called ages.

ages[0] will be Mr C's own age. Also, if a person's age is at index i in the array, then that person's mother's age will be at index 2*i + 1 and that person's father's age will be at index 2*i + 2. In particular, Mr.Câ€™s motherâ€™s and father's ages are given at indices 1 and 2, respectively.

Your job is to check if the given ages are valid: that for every person, both their parents should be strictly older than the person since it is obviously an error if one's parents are the same age as oneself or are younger than oneself. If the given ages are valid, print the word "VALID" as the output. Otherwise, print the word "INVALID" in the output followed by a space followed by the number of parent child pairs which were in violation of the above age rule.

Hint: Although this question can be solved using loops, there are cute solutions that use recursion. Try a recursive solution to test your recursion prowess.

Caution

1. Array indexing in C starts from 0
2. Be careful not to double count parent-child pairs in case the ages are invalid
3. Count only invalid parent-child pairs, not grandparent-grandchild pairs, etc.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE 1:
INPUT
3
1
2
3

OUTPUT:
VALID

Explanation: Mr.C has age 1, and his parents have ages 2, 3 and are thus strictly older than him.

EXAMPLE 2:
INPUT
3
2
1
2

OUTPUT:
INVALID 2

Explanation: Mr C is older than his mother and is of the same age as his father, both relationships are in violation of the age rule since children must be strictly younger than their parents.

Grading Scheme:
Total marks: **[20 Points]**

There will be no partial grading in this question. An exact match will receive full marks whereas an incomplete match will receive 0 points. Please be careful of missing/extra spaces and missing/lines (take help of visible test cases). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output
3 2 1 3	INVALID 1
7 1 3 3 7 6 5 4	VALID
6 1 1 2 1 1 1	INVALID 4
7 1 2 3 1 4 5 6	INVALID 1
10 1 2	VALID

3 4 5 6 7 8 9 10	
15 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	INVALID 14

Simple Sodoku (p1v3d1)

Simple Sodoku [20 marks]

--

Problem Statement
The standard Sodoku problem concerns a 9x9 grid and the goal is to fill up the grid with integers from 1, 2, ..., 9 so that each row, column and 3x3 block contains the digits 1 - 9 exactly once. In this problem we will consider a simpler version of the problem where the goal will be to fill up an n x n grid with the integers 1, 2, ... n where each row and column must contain each integer from 1, 2, ... n exactly once (i.e. the block constraint is removed in our simpler version of Sodoku). Let us call such an n x n grid a valid SS grid (SS stands for Simple Sodoku).

As input, we will give you a partially filled SS grid. Your job is to fill it in all valid ways possible and show us all the valid SS grids that you get. The first line of the input will give you n, a strictly positive integer. We assure you that n will be a single digit number i.e. n will be strictly less than 10. This will mean that we will work with an n x n SS grid in that test case.

In the next several lines, we will give you some entries of the SS grid that are already filled in. This will be done by giving in each line, three non-negative integers i, j, val. All three integers will be separated by a single space and will indicate that we have already filled in the entry SS[i][j] = val. The value val will be between 1 and n and the values i and j will be between 0 and n-1 since they denote an index in the SS grid. The list of partially filled entries will end when we give you the three numbers -1 -1 -1. This means that there are no more entries of the SS grid that we want to specify.

Your job is to output all possible completions of the SS grid given the already filled-in entries. To output a completion of the SS grid, first output all elements of the first row, then all elements of the second row then all elements of the third row and so on. There should be no spaces between elements of a row nor should there be any spaces or newlines between elements of two different rows i.e. the entire completion must be printed on a single line. Different completions must be printed in different lines.

Suppose there are multiple completions of the SS grid we have given you. In such a case, you have to output completions in lexicographically increasing order. Basically, notice that each completion of the SS grid looks like a giant number (recall that we promised that n will always be strictly positive but strictly less than 10). You have to view the completions as a number and output them in increasing order. In the last line of the output, print how many valid completions were there.

For example, if n = 2, then the following is a valid completed SS grid:
1 2
2 1
But the following is invalid
1 1
2 1
because the first row has the entry 1 repeated.

Caution

1. Warning: we may give you an empty SS grid to being with as well. This will happen if the very first line in the list is -1 -1 -1
2. We promise you that there will be at least one completion of the SS grid given the partially filled-in entries.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You should try to solve this problem recursively. Write a function
int sudokuSolv(int** grid, int* missing, int done, int n, int m)
which takes in a partially filled in grid, a list of missing entries, the number of missing entries that have already been filled, the size of the grid n and m is the number of missing entries in the original board (including those that have already been filled and those that are remaining to be filled in).

The function returns the number of ways this partially filled grid can be completed to form a valid SS grid. The base case can be when done = m i.e. when we have already filled in all the entries. For a recursive case, choose a number to fill at the next unfilled entry and proceed.

To store the location of the missing entries in the original grid, you may either maintain 2 arrays each of size m (one storing row index of the missing entries and the other storing the column index of the missing entries). You should first store locations of all missing entries in row 1 (from left to right), then all missing entries in row 2 (from left to right) and so on. A different way (as used by the function declared above) is to index the 2D grid as a 1D array (as we saw in lectures and on Piazza) in which case you only have to maintain only one index for the missing entries since the entire grid is being represented as a 1D array.

EXAMPLE:
INPUT
```
3
0 0 1
-1 -1 -1
```

OUTPUT:
```
123231312
123312231
132213321
132321213
4
```

Explanation: The incomplete grid, as provided, looks like
1 * *
* * *
* * *
where * indicates an unfilled entry. There are exactly 4 ways of completing this grid with the given constraints. They are:
1 2 3
2 3 1
3 1 2

1 2 3
3 1 2
2 3 1

1 3 2
2 1 3
3 2 1

1 3 2
3 2 1
2 1 3

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

2
1 1 2
-1 -1 -1
	2112
	1

3
0 0 1
-1 -1 -1
	123231312
	123312231
	132213321
	132321213
	4

3
1 1 1
-1 -1 -1
	123312231
	132213321
	231312123
	321213132
	4

2 -1 -1 -1	1221 2112 2
4	1234241331424321

0 0 1
0 1 2
0 2 3
0 3 4
2 2 4
3 3 1
-1 -1 -1
	1234341221434321
	1234431221433421
	3

5
0 0 1
0 1 2
0 2 3
0 3 4
2 2 4
3 3 1
-1 -1 -1
	1234521534354215421343152
	1234521534534214521334152
	1234523154354215421341532
	1234523154514233451245231
	1234523154514324521334521
	1234524153354214351251234
	1234524153354215321441532
	1234524153514323521443521
	1234524153534213521441532
	1234524531314524521353124
	1234524531314525321445123
	1234524531514233521443152
	1234525134314525421343521
	1234525134514233451243251
	1234525134514234351234251
	1234525134534213451241253
	1234531254254314351254123
	1234531254534212451345132
	1234531524234514521354132
	1234531524234515421345132
	1234531524254315421343152
	1234534152254315321441523
	1234534152514232351445231
	1234534152534214521321534
	1234534251514232351445132
	1234534251514234351225134
	1234534251514322351445123
	1234534521214535321445132
	1234534521514324521323154
	1234535124214534351254231
	1234535124234515421341532
	1234535124514322451343251
	1234541253254313451253124
	1234541253354212351454132
	1234541253534213451225134
	1234541523234513521454132
	1234541523254315321434152
	1234541532234513521454123
	1234541532234515421335124
	1234541532354215321424153
	1234541532354215421323154
	1234541532534213521424153
	1234543152254315421331524
	1234543152354212451351234
	1234543152354215421321534
	1234543152514233521424531
	1234543251514233451225134
	1234543251514322451335124
	1234543521214533521454132
	1234543521314525421325134

```
123454352151432352142415 3
```

```
1234543521514323521424153
1234545123234513451251234
1234545123314522351454231
1234545123314525321424531
1234545123514322351434251
1234545132214535321434521
1234545132234515421331524
1234545132514232351434251
1234545132534212451331254
1234545231214533451253124
1234545231314522351454123
1234545231314522451353124
1234545231514232351434152
1234545231514233451223154
1234551234234513451245123
1234551234354212451343152
1234551234354214351224153
1234553124214533451245231
1234553124254313451241253
1234553124314522451345231
1234553124314524521324531
1234554123234513521441532
1234554123254314351231254
1234554123314522351445231
1234554132214533521443521
1234554132234513521441523
1234554132234514521331524
1234554132354212351441253
1234554231214534351235124
1234554231314522351445123
80
```

The Family Tree of Mr C Part Three (p2v1d1)

The Family Tree of Mr C Part Three [20 marks]

Problem Statement
In the first line of the input, we will give you n, a strictly positive number telling you how many members are there in Mr C's family. The family members will have names 1, 2, 3, 4, ..., n. In the next n-1 lines, we will tell you who is whose parent in this family by giving you pairs of numbers separated by a space i.e. in the format P C which tells us that P is the parent of C.

In Mr C's family there is one *head of the family* who has no parent. Everyone else has exactly one parent (it is a weird family where there is only one parent not two). However, people may have multiple children. The head of the family is said to belong to the *first generation* i.e. generation 1. There can be only one head of the family. All children of the head are said to belong to the *second generation* i.e. generation 2. All their children are said to belong to generation 3 and so on.

After giving you information about who is whose parent, we will give you a strictly positive number Q which tells you how many questions we are going to ask you. In the next Q lines we will give you names of Q family members (i.e. numbers between 1 and n, 1 and n included). In your output, you have to print to which generation do these family members belong.

Caution

1. Any number between 1 and n (both included) could be the head. The only distinguishing factor of the head is that the person has no parent
2. The parent child relationships will not be given to you in any particular order.
3. Your output should have Q lines too.
4. Be careful that we wont tell you who is the head's parent since the head has no parent. This is why we will only give you n-1 parent-child relationships.
5. Hint: if we do not give you the parent of a particular family member, that member has to be the head.
6. People may have multiple children but everyone has one and only one parent, except the head of the family who has no parent.
7. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
3
1 2
2 3
2
1
2

OUTPUT:
1
2

Explanation: the family tree looks like
1
\
2
\
3
Since we did not give the parent of 1 in the input, 1 is the head. 1 is the parent of 2 and 2 is the parent of 3. 1 belongs to the first generation and 2 belongs to the second generation.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
3 1 2	1 2

2 3
2
1
2	

6
1 3
3 5
3 6
1 2
1 4
3
2
3
6
	2
	2
	3

1 1 1	1

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

6
1 2
1 3
2 4
2 5
3 6
2
4
5
	3
	3

9
9 2
2 6
3 4
8 5
8 3
5 1
7 8
7 9
9
9
8
7
6
5
4
3
2
1
	2
	2
	1
	4
	3
	4
	3
	3
	4

The Post offices of KRville (p2v2d1)

The Post offices of KRville [20 marks]

Problem Statement
Mr C lives in the country of KRville, named after Kernighan and Ritchie, the developers of the C language. Cities in KRville are arranged on a 2D grid with M rows and N columns. Some of the cities have a post office (PO) in them and some do not. All PO have a 6 digit PIN number associated with them. The PIN is always strictly positive and the first digit of the PIN is never 0.

Two towns are said to be neighbours of each other if they lie on adjacent rows as well as adjacent columns. Thus, if M = 5 and N = 5 the town at index (2,2) has 8 neighbours (1,1), (1,2), (1,3), (2,1), (2, 3), (3,1), (3,2), (3,3). As a visual example, the town numbered 5 in the matrix below has neighbors as the towns numbered 1, 2, 3, 4, 6, 7, 8, 9.

1 2 3
4 5 6
7 8 9

1. If a town T1 has its own PO with pin P1 then T1 is serviced by P1.
2. If a town T2 doesnâ€™t have a post office of its own and the neighbours of T2 do not have a post office either then T2 is not serviced at all.
3. If a town T3 doesnâ€™t have its own PO but one of its neighboring towns has a PO then T3 is serviced by the neighbouring PO. In case multiple neighboring towns have a PO, then the PO with the smallest PIN code will serve T3.

In the first line of the input we will give you M and N which denotes number of rows and columns in the grid. In the second line, you will be given P, which is the number of post offices in the country. In the next P lines, you will be given location of these post-offices and the corresponding PIN numbers as three strictly positive numbers

X Y PIN

X will be the row number of the PO, Y will be the column number of the PO and PIN will be its PIN number. Note that the row numbers and column numbers we give you will start from one i.e. they will not be like C array indices which start from 0.

In the first line of the output, in the format given below. print the number of towns which are not serviced by any PO. In the next line, you have to print the PIN code of the post-office which serves the maximum number of towns. If there are multiple such POs, print the PIN of the post-office with smallest PIN.

Caution

1. Follow the output format closely. Do not make spelling, space, capitalization errors.
2. The row and the column numbers will be given to you in human readable form i.e. the first row will have row number 1. Note that these are not C indices that start from zero. These are human readable location numbers.
3. If the town is a corner town then it has only 3 neighbours. If a town is an edge town, it has 5 neighbours. Otherwise, each town has eight neighbours.

4. We will never give you input where two different PO have the same pin number. Different PO have different pins. All pins are non-negative integers.
5. Do not wrap around the matrix in search for a post office. For example, if M = 3 and N = 5, then the post office at index (1,0) cannot serve the town at index (1,4). Similarly the post office at index (2,2) cannot serve the town at index (0,2). Only postoffices that are physically adjacent to a town can serve a town in case that town does not have a post office of its own.
6. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You may solve this problem using structures (although it is not compulsory). Create a structure
struct town{
int POInTown;
int POServingTown;
int numTownsServedbyPO;
};
where POInTown denotes the PIN code of the post office if one is located in the town itself. If no post office is there in the town, then this variable can be -1. POServingTown tells us which post office serves this town. If there is a post office in town then obviously POServingTown = POInTown. numTownsServedbyPO tells us how many towns does the post office in this town serve. If this town does not have a PO then this number can be -1.

Create a 2D matrix of these structures
struct town[M][N];
and work with it to solve the problem.

EXAMPLE:
INPUT
3 3
2
1 1 208016
3 3 200001

OUTPUT:
NO PO: 2
BUSIEST PO: 200001

Explanation: The country looks like this (a * indicates no PO in that town)
208016 * *
* * *
* * 200001

There are two towns without any PO access. 20001 is the busiest PO serving 4 towns. 208016 serves only 3 towns.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

9 9
2
1 1 208016
9 9 200001
	NO PO: 73
	BUSIEST PO: 200001

2 2
1
1 1 100001
	NO PO: 0
	BUSIEST PO: 100001

3 3
8
1 1 100002
1 2 100001
1 3 100003
2 1 100004
2 3 100005
3 1 100009
3 2 100007
3 3 100008
	NO PO: 0
	BUSIEST PO: 100001

1 1
1
1 1 100008
	NO PO: 0
	BUSIEST PO: 100008

2 2
4
1 1 100003
1 2 100002
2 1 100005
2 2 100009
	NO PO: 0
	BUSIEST PO: 100002

3 3
1
2 2 100002
	NO PO: 0
	BUSIEST PO: 100002

Matrix Mandala (p2v3d1)

Matrix Mandala [20 marks]

Problem Statement
In the input we will give you two strictly positive single digit numbers N and C, separated by a space. Thus, both N and C strictly greater than 0 and strictly smaller than 10. The question will work with N X N matrices and we have to color each location in the matrix using one of the colors from 1, 2, ..., C.

Now if there were no other constraints, there would have simply been C^{N^2} such colorings. However, we want beautiful colorings. A coloring is beautiful if no location in the matrix has the same color as one of its neighbors. Every location in the matrix has four neighbors, the location immediately to its right, the location immediately to its left, the location immediately above it and the location immediately below it. If a cell is on the edge of the matrix or at one of the corners of the matrix, then its neighbors are found by wrapping around our search.

For example, consider the following 3 x 3 matrix colored using 9 colors (N = 3, C = 9)
1 2 3
4 5 6
7 8 9
The neighbors of the location with color 5 are the locations with colors 4, 2, 6 and 8. The neighbors of the location with color 2 are the locations with colors 1, 8, 3, and 5, The neighbors of the location with color 7 are the locations with colors 9, 4, 8 and 1. Thus, the search for neighbors simply wraps around the matrix if we reach and edge or a corner.

We want to find and output all beautiful colorings. To output a coloring, simply output all colors of the first row (from left to right), followed by all colors of the second row (from left to right) and so on. Thus, the above coloring would be output as 123456789. Your output must list all beautiful colorings in lexicographically increasing order (i.e. view each coloring as a long number and output the numbers in increasing order). In the last line of the output, print how many beautiful colorings did you find.

Caution

1. Warning: note that there can be no 1 x 1 beautiful coloring since the only location will become its own neighbor from every direction so we cannot give it any color without giving it the same color as one of its neighbors.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You should try to solve this problem recursively. Write a function
int gridColour(int** mat, int rowNext, int colNext, int N, int C);
which takes in a partially filled in matrix, the location (rowNext, colNext) in the matrix where the next color needs to be filled, the size of the matrix and the number of colors allowed. The function prints all possible beautiful colorings possible from this partially filled in matrix and returns the number of ways this partially filled matrix can be completed to form a beautifully colored grid.

The routine should start filling in colors to locations in the first row from left to right, then after the first row is completed, to the second row from left to right and so on. Take care to fill smaller colors first and bigger colors later to preserve the lexicographic ordering. The base case can be when we have filled in all the entries of the matrix i.e. when the next entry to be filled lies in row n+1 i.e. when rowNext = n and colNext = 0. This means that the matrix is completely colored and if beautifully colored, can be printed.

EXAMPLE 1:
INPUT
2 2

OUTPUT:
1221
2112
2

Explanation: There are only two beautifully colored matrices using 2 colors
1 2
2 1

2 1
1 2

Notice that the colorings have been output in increasing order.

EXAMPLE 2:
INPUT
2 1

OUTPUT:
0

Explanation: We cannot beautifully color a 2 x 2 matrix using just one color

EXAMPLE 3:
INPUT
1 5

OUTPUT:
0

Explanation: A 1 x 1 matrix cannot be beautifully colored no matter how many colors are given since the lone element in the matrix ends up being its own neighbor.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 2	1221 2112 2
3 3	123231312 123312231 132213321 132321213 213132321 213321132 231123312 231312123 312123231 312231123 321132213 321213132 12
2 3	1221 1223 1231 1321 1331

	1332 2112 2113 2132 2312 2331 2332 3112 3113 3123 3213 3221 3223 18
3 2	0
4 2	1212212112122121 2121121221211212 2
1 9	0

Mango Mania (p3v1d1)

Mango Mania [20 marks]

Problem Statement
Before IITK was built, large swathes of this campus used to be mango orchards. In fact the Hall-I grounds still contain several mango trees. In the first line of the input, we will give you a strictly positive number n, the number of trees in the orchard. Each tree has been given a number by the gardener. The trees will be numbered 1, 2 ... n. In the next n lines, we will give you details about n trees in the following format - there will be two numbers in each line separated by a space
P M
P will denote the number (from 1 to n) assigned to the tree by the gardener. M will be a non-negative number which will denote the number of mangoes on that tree. Each tree in the orchard has a unique number P i.e. no two trees have the same number.

After this we will give you a non-negative number k which denotes the number of students lined up to eat mangoes. In the next k lines we will give you the roll numbers of the k students. We assure you that k will be less than or equal to n i.e. there will never be more students than mango trees.

The first student in the line will choose the mango tree with the largest number of mangoes (for obvious reasons) and go and occupy that tree. If there is more than one tree with the largest number of mangoes, the student will choose the tree with the smallest tree number P. The second student in the line will similarly go and occupy an unoccupied tree with the largest number of mangoes and if there is more than one of those, choose the one with the smallest tree number P.

In your output, print for every tree in the orchard, what is the roll number of the student occupying that tree in the format given below. If there is no student occupying that tree, print a message as indicated below. Note that you have to print this information about the trees in increasing order of the number assigned to the trees i.e. print the information about tree numbered 1 first then tree number 2 and so on.

Caution

1. Observe the output format carefully. Do not make mistakes in spaces, capitalization, or spelling.
2. Two trees may have the same number of mangoes. Some trees may have zero mangoes.
3. No tree will have more than one student occupying it since students always choose among unoccupied trees.
4. There may be no students lined up to eat mangoes, i.e. k may be zero. This may happen, for instance, if the mess is serving mango ice-cream that day.
5. The trees need not be given to you in increasing order of the tree number P. The trees may be given to you in any order. However, your output must be in increasing order of the tree number P.
6. The students are listed in the order they arrive in the orchard. They are not necessarily listed in increasing or decreasing order of their roll numbers.
7. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You may want to store the information about each tree in a structure.

```
struct Tree{
int treeNum;
int numMangoes;
int rollNo;
};
```

and use an array of Tree variables declared as follows

```
struct Tree trees[n];
```

to solve the problem. Using structures is not compulsory though.

EXAMPLE:

INPUT

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

OUTPUT:

```
TREE 1 IS NOT OCCUPIED
STUDENT AT TREE 2 IS 7
TREE 3 IS NOT OCCUPIED
STUDENT AT TREE 4 IS 10
STUDENT AT TREE 5 IS 12
```

Explanation: The first student (roll number 12) heads for tree number 5 with 8 mangoes which is the highest. The second student (roll number 10) goes for tree number 4 since it is the unoccupied tree with largest number of mangoes ((5 mangoes). The third student (roll number 7) heads for tree number 2 with the next largest number of mangoes i.e. 4 mangoes. Tree numbers 1 and 3 are unoccupied.

Note that tree information in output is printed for tree 1 first then tree 2 and so on.

Grading Scheme:

Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

5
1 2
2 4
3 2
4 5
5 8
3
12
10
7
	TREE 1 IS NOT OCCUPIED
	STUDENT AT TREE 2 IS 7
	TREE 3 IS NOT OCCUPIED
	STUDENT AT TREE 4 IS 10
	STUDENT AT TREE 5 IS 12

4
3 0
4 2
1 2
2 5
3
10
1
9
	STUDENT AT TREE 1 IS 1
	STUDENT AT TREE 2 IS 10
	TREE 3 IS NOT OCCUPIED
	STUDENT AT TREE 4 IS 9

5
1 2
2 4
3 2
4 5
5 8
0
	TREE 1 IS NOT OCCUPIED
	TREE 2 IS NOT OCCUPIED
	TREE 3 IS NOT OCCUPIED
	TREE 4 IS NOT OCCUPIED
	TREE 5 IS NOT OCCUPIED

1
1 90
1
32
	STUDENT AT TREE 1 IS 32

10
1 1
3 1
5 1
7 1
9 1
2 1
4 1
6 1
8 1
10 1
6
60
50
40
	STUDENT AT TREE 1 IS 60
	STUDENT AT TREE 2 IS 50
	STUDENT AT TREE 3 IS 40
	STUDENT AT TREE 4 IS 30
	STUDENT AT TREE 5 IS 20
	STUDENT AT TREE 6 IS 10
	TREE 7 IS NOT OCCUPIED
	TREE 8 IS NOT OCCUPIED
	TREE 9 IS NOT OCCUPIED
	TREE 10 IS NOT OCCUPIED

30 20 10	

10
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 3
9 2
10 1
10
1
2
3
4
5
6
7
8
9
10
	STUDENT AT TREE 1 IS 1
	STUDENT AT TREE 2 IS 2
	STUDENT AT TREE 3 IS 3
	STUDENT AT TREE 4 IS 4
	STUDENT AT TREE 5 IS 5
	STUDENT AT TREE 6 IS 6
	STUDENT AT TREE 7 IS 7
	STUDENT AT TREE 8 IS 8
	STUDENT AT TREE 9 IS 9
	STUDENT AT TREE 10 IS 10

Recover the Rectangle (p3v2d1)

Recover the Rectangle [20 marks]

Problem Statement
Mr C had drawn a nice axis-aligned rectangle (i.e. whose sides are parallel either to the x or the y axis) on a piece of paper and decorated his drawing with a few dots. However, one of his mischievous clones came and erased the lines forming the edges of the rectangle leaving only the dots for the corners behind. Help Mr C recover his nice rectangle.

The first line of the input will give you n, a strictly positive number, giving you the number of points on the plane. In the next n lines, we will give you the x and y coordinates of n points on the 2D plane, separated by a space. The coordinates will all be integers. In your output, you have to print the area of the largest axis-aligned rectangle that can be formed out of the n points we have given you. If no axis-aligned rectangle can be formed out of the points we have given you, simply print -1 in the output.

Caution

1. Rest assured that we will give you at least 4 points i.e. n will be greater than or equal to 4.
2. The rectangle we are looking for has non-zero area. Please do not report a single point as a rectangle of area zero. If there is no axis-aligned rectangle of non-zero area, you should print -1 as your output.
3. The rectangle we are looking for must be axis aligned. Do not report a rectangle whose sides are not parallel to the x and y axes.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: An axis-aligned rectangle, as we discussed in class, is always uniquely identified using its lower left corner and its upper right corner. You may also want to use a structure to store the points and use an array of these structure variables to process the points given to you.

```
struct Point{
int x,y;
};
struct Point points[n];
```

EXAMPLE:
INPUT
9
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3

OUTPUT:
4

Explanation: the points (1,1) (1,3) (3,1) (3,3) form a rectangle of area 4.

Grading Scheme:
Total marks: **[20 Points]**

There will be no partial grading in this question. An exact match will receive full marks whereas an incomplete match will receive 0 points. Please be careful of missing/extra spaces and missing/lines (take help of visible test cases). Each visible test case is worth 1 point and each hidden test case is worth 2 points. There are 2 visible and 4 hidden test cases.

All Test Cases (Visible + Hidden)

Input	Output

9
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
	4

4
1 1
1 2
1 3
1 4
	-1

7
1 1
1 2
1 3
2 2
2 3
3 0
3 3
	1

4
1 0
0 1
1 2
2 1
	-1

16
4 3
4 4
1 3
2 2
2 5
3 2
3 5
4 1
4 6
5 3
5 4
6 7
1 1
1 4
1 6
1 7
	15

19
6 4
4 3
2 4
4 3
5 2
1 0
2 1
6 1
7 0
5 4
2 0
3 1
4 2
5 3
3 3
4 2
5 1
3 4
3 2
	12

Crazy for Candy (p3v3d1)

Crazy for Candy [20 marks]

Problem Statement
Mr C just loves candy. Actually he is a bit addicted to candy. The situation got so bad that the doctors had to put Mr C on a rehabilitation program and ordered that he must have exactly n grams of candy every day (not one gram less, not one gram more). This was done to restrict his candy intake so that he can get rid of his addiction.

However, candy is only available in the shops in certain weights. We need to help Mr C find all possible ways in which he can take store-bought candy and eat them to fulfill his daily intake. The first line of the input will give you n, the number of possible weights of candy available in the stores. The next line will give you these n weights in increasing order. All the weights will be distinct i.e. no weight will ever repeat. The weights will all be in grams and will all be strictly positive integers, separated by a space. The last line of the input will give you k, the total weight of candy Mr C must consume, in grams.

In your output you have to print all possible ways in which Mr C can consume various numbers of candy of weights given in the input, so that his total candy intake is exactly k grams. Assume that there is unlimited amount of candy of each given weight available in the stores so Mr C can consume as many number of candy of a certain weight as he wants.

If there is no way store bought candy can be consumed in different amounts to get exactly k grams of candy, then print "MR C IS DOOMED" (without quotes) in the output. Otherwise, if one or more combinations are possible, then on every line of the output, you have to print a possible combination of candy that total k grams by printing how many number of candy of each weight Mr C consumes in that combination.

For example, if the stores have two types of candy (i.e. n = 2) of weight 2 grams and 3 grams, and Mr C's daily intake is 6 grams, then he can fulfill his intake in two ways

1. Take 3 candy of 2 grams each. This is represented as "30" (without quotes - no space between the two numbers, no space at the end) indicating that 3 candy of first kind and 0 of second kind are to be taken.
2. Take 2 candy of 3 grams each. This is represented as "02" (without quotes - no space between the two numbers, no space at the end) indicating that 0 candy of first kind and 2 candy of second kind are to be taken.

Your output must print both these combinations on different lines. However, if there are multiple possible combinations, as in the case above, then you must output combinations in a lexicographically decreasing order as described below. We assure you that in none of the combinations Mr C will take, will he every require more than 9 number of any type of candy. Thus, every combination can be written as an n digit number (one digit for every type of candy). You have to output combinations so that these numbers appear in decreasing order. Thus, in the above example, the output should be
30
02
since 30 > 2

Caution

1. Candy weights will all be strictly positive and no two of the n weights we give you will be the same. The n candy weights will be given in increasing order.
2. Be careful to output the combinations in exactly the same order as described in the question. If you output the correct combinations but in the wrong order, then the autograder may give you zero marks since it counts partial marks only when the correct combination is given at the correct location.
3. You have to print the leading zeros in your output as well to indicate that zero number of candy is to be taken. Do not omit the leading zeros. Your output must contain exactly n digits. See the above

example, the second combination is printed as 02 not just 2.
4. We will not penalize you for extra new lines at the end of your output but do not have extra spaces at the end of any line of your output.

HINTS: This problem can be solved using recursion. You can write a function of the following kind
void printCombinations(int *weights, int *num, int done, int n, int rem, int *found)

1. The first argument is simply the array of weights of the various types of candy
2. The second argument is a partially filled array to store how many number of candy of each type Mr C should eat
3. The third argument tells us how many types of candy have we already accounted for and how many are left (basically till what point is the array num filled up and from which point onward we should start filling num hereon)
4. The fourth argument tells us how much weight of candy remains to be accounted for. Initially this number is k since we have to account for k grams of candy but as we fill up the num array, this weight will go down.
5. The fifth argument is simply a flag to signal whether we found a single combination or not. Set *found = 1 the moment you find even one valid combination. If found remains zero throughout, you have to print "MR C IS DOOMED" (without quotes) in the output.

You can start off the recursion process, for example, as follows
int found = 0;
int num[n];
printCombinations(weights, num, 0, n, k, &found);

EXAMPLE 1:
INPUT
2
2 3
6

OUTPUT:
30
02

Explanation: See above example.

EXAMPLE 2:
INPUT
3
2 4 6
9

OUTPUT:
MR C IS DOOMED

Explanation: There is no way to express 9, an odd number, as a sum of a bunch of even numbers. All weights are even numbers so no matter how many of each type of candy Mr C takes, the resultant sum will always be even.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

2
2 3
6
	30
	02

3
2 4 6
9
	MR C IS DOOMED

5
1 2 3 4 5
5
	50000
	31000
	20100
	12000
	10010
	01100
	00001

4
2 3 5 7
7
	2100
	1010
	0001

5
2 4 6 7 8
19
	60010
	41010
	30110
	22010
	20011
	11110
	03010
	01011
	00210

14
2 3 4 5 6 7 8 9 10 11 12 13 14 15
15
	61000000000000
	50010000000000
	41100000000000
	40000100000000
	33000000000000
	31001000000000
	30110000000000
	30000001000000
	22010000000000
	21200000000000
	21000010000000
	20100100000000
	20011000000000
	20000000010000
	13100000000000

	12000100000000 11101000000000 11020000000000 11000000100000 10210000000000 10100001000000 10010010000000 10001100000000 10000000000100 05000000000000 03001000000000 02110000000000 02000001000000 01300000000000 01100010000000 01010100000000 01002000000000 01000000001000 00200100000000 00111000000000 00100000010000 00030000000000 00010000100000 00001001000000 00000110000000 00000000000001

A Brutal Cipher Called Brutus (p4v1d1)

A Brutal Cipher Called Brutus [20 marks]

Problem Statement
Mr C needs to convey a secret message to one of his clones without the other clones being able to overhear and know the secret. The two agree on two single digits (i.e. 0-9). These will be given to you in the first line of the input as a non-negative two digit number. Let the first digit in the number be called P and the second digit be called Q. For example, let P = 2 and Q = 3 (i.e. the number given to you is 23).

The first thing Mr C does is he rotates the letters at odd locations in the alphabet (e.g. A, C, E, G etc) right by P locations among themselves. In this example, since P = 2, A will shift to E's position in the alphabet, C will shift to G's position in the alphabet and so on. The process wraps around i.e. W shifts to A's position and Y shifts to C's position.

Next, Mr C rotates the letters at even locations in the alphabet (e.g. B, D, F, H, J etc) right by Q locations among themselves. In this example, since Q = 3, B will shift to H's position in the alphabet, D will shift to J's position in the alphabet and so on. The process wraps around i.e. Z shifts to F's position.

The original alphabet is
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
After doing these two rotations, the alphabet looks like
W V Y X A Z C B E D G F I H K J M L O N Q P S R U T

After this, the final step is to reverse the alphabet to get
T U R S P Q N O L M J K H I F G D E B C Z A X Y V W

Thus, A maps to T, B maps to U and so on. In the second line of the input, we will give you a message containing only uppercase English characters or spaces. You have to tell us what is the encrypted message (spaces remain spaces in the encrypted message).

In the first line of the output, print the alphabet as obtained after the two rotation steps (print the 26 characters in a single line - no spaces between two characters, no spaces at the end). In the second line of the output, print the alphabet after further performing the reversal step. In the third line of the output, print the encrypted string.

Caution

1. The original message will contain no more than 99 characters and will be presented to you in a single line.
2. Be careful, one or both the digits in the key can be zero, i.e. the first line can give you a number like 01 or even 00.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
23
HOW DO YOU DO

OUTPUT:
WVYXAZCBEDGFIHKJMLONQPSRUT
TURSPQNOLMJKHIFGDEBCZAXYVW
OFX SF VFZ SF

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are three lines in your output. Printing each line correctly, in the correct order, carries 33% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
23 HOW DO YOU DO	WVYXAZCBEDGFIHKJMLONQPSRUT TURSPQNOLMJKHIFGDEBCZAXYVW OFX SF VFZ SF
60 WEEK TWELVE LAB QUESTION	OBQDSFUHWJYLANCPERGTIVKXMZ ZMXKVITGREPCNALYJWHUFSDQBO DVVP UDVCSV CZM JFVHURLA
02 THIS IS A WEIRD ALGORITHM	AXCZEBGDIFKHMJOLQNSPURWTYV VYTWRUPSNQLOJMHKFIDGBEZCXA

	GSND ND V ZRNIW VOPHINGSJ
00 BRUTUS CIPHER	ABCDEFGHIJKLMNOPQRSTUVWXYZ ZYXWVUTSRQPONMLKJIHGFEDCBA YIFGFH XRKSVI
15 A	YRATCVEXGZIBKDMFOHQJSLUNWP PWNULSJQHOFMDKBIZGXEVCTARY P
99 I AM TRAPPED	IJKLMNOPQRSTUVWXYZABCDEFGH HGFEDCBAZYXWVUTSRQPONMLKJI Z HV OQHSSDE

Triangle Tangle (p4v2d1)

Triangle Tangle [20 marks]

Problem Statement
In the first line of the input we will give you n, a strictly positive integer. In the next n lines, we will give you the three corners of n triangles on the 2D plane. Each corner will be described using its x and y coordinates. All coordinates will be integers. To specify a triangle, we will give you 6 integer coordinates, separated by a space as described below
x1 y1 x2 y2 x3 y3
This will describe a triangle. In the last line of the input, we will give you two points on the 2D plane which will uniquely define a line (as well as a line segment). Both points will be similarly give to you by specifying their x and y coordinates, separated by a space. All coordinates will be integers.
x1 y1 x2 y2

In your output you have to print n lines, telling us which all of the triangles intersect the line. If a certain triangle does intersect with the line, print "YES" (without quotes) else print "NO" (without quotes). An intersection is counted whenever a triangle and a line share more than one point in common. Thus, if only one vertex of the triangle lies on the line, it is not counted as an intersection. However, if an entire edge of the triangle is present on the line, it is counted as an intersection.

Caution

1. The question may require you to compare floating point numbers like slopes etc. Since comparing floating point numbers for equality is dangerous, we will consider two floating point numbers to be the same if their difference in absolute terms is less than 0.0001. Use the fabs() function from math.h to get the absolute value of floating point numbers.
2. Be careful that coordinates can be negative too. Also be careful that the line we give you may be a vertical or a horizontal line. Also be careful that the line we give you may actually be along one of the edges of one of the triangles.
3. We will never give you line segments with zero length i.e. the two points in the last line will never be the same.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You may want to use structures (although it is not compulsory to use structures) to make your code neater and easier to debug. As we saw in class, create a structure
struct Point{
int x,y;

```
};
using which you can create line structures
struct Line{
struct Point p1, p2;
};
and triangle structures
struct Triangle{
struct Point p1, p2, p3;
};
```

EXAMPLE:
INPUT

```
2
0 0 2 0 0 2
5 6 7 8 9 11
0 0 1 1
```

OUTPUT:

```
YES
NO
```

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output

2
0 0 2 0 0 2
5 6 7 8 9 11
0 0 1 1
	YES
	NO

3
0 0 2 0 0 2
0 0 1 0 0 1
0 0 3 0 0 3
0 2 1 2
	NO
	NO
	YES

3
0 0 2 0 0 2
0 0 1 0 0 1
0 0 3 0 0 3
2 0 2 1
	NO
	NO
	YES

3
0 0 1 1 2 0
0 0 2 0 0 2
	YES
	YES
	YES

0 0 -2 0 0 -2 0 0 1 1	

3
0 0 1 1 2 3
1 2 3 4 5 5
1 1 2 2 2 1
-5 0 0 -5
	NO
	NO
	NO

6
0 0 1 1 2 3
1 2 3 4 5 5
1 1 2 2 2 1
-2 1 2 1 0 -5
-3 2 -4 3 -5 0
3 -2 4 -3 5 0
0 1 1 1
	YES
	NO
	YES
	YES
	YES
	NO

Basic Balanced Bracketing (p4v3d1)

Basic Balanced Bracketing [20 marks]

Problem Statement
Brackets are essential to coding. We use brackets to indicate blocks of statements, for instance, in loops, conditional statements, functions etc. However it is very important to have *balanced bracketing*. This means that whenever an opening bracket (is introduced, its sister bracket) must also be introduced sometime later. Thus, the following are examples of illegal bracketing (there are more open brackets than closing brackets)
(
(()
((())
However, the reverse is also true. Every closing bracket) must have her sister bracket specified somewhere. Thus, the following are also examples of illegal bracketing (there are more closing brackets than opening brackets)
)
())
(()))
Moreover, a closing bracket can come only after her sister opening bracket has already come. Thus, in no point in the string, when read from left to right, should we have more closing brackets than opening brackets. This has to hold even if the total number of opening and closing brackets in the overall string is the same. Thus, the following are also examples of illegal bracketing (even though total number of opening brackets is the same as the total number of closing brackets, some of the closing brackets come before their sister opening brackets have come)
)(
)()(
()))(
Any bracketing that does not violate the above rules i.e. has the same number of opening and closing bracket, and makes sure that at no point have we closed more brackets than opened brackets, is a valid bracketing. Examples include
((()))
()()()
(())()
()(())

In the input you will be given a number n. You have to print all possibly ways of performing valid bracketing using n opening brackets and n closing brackets. Print each valid way in a separate line. You have to print the valid bracketings in lexicographically increasing order. For this, we will consider that the opening bracket (has a "smaller" value than the closing bracket). Say we assign the opening bracket a value of 1 and the closing bracket a value of 2. Then the valid bracketings mentioned above would look like

((())) 111222
()()() 121212
(())() 112212
()(()) 121122

However, it is clear that these have not been specified in increasing order. The correct order would have been

((())) 111222
(())() 112212
()(()) 121122
()()() 121212

Caution

1. You only have to output the bracketings - do not output the number equivalents.
2. We will not penalize you for extra newlines at the end of your output. However, do not have extra spaces at the end of any line of the output.
3. Make sure that you output the bracketings in correct order. If you output all possible bracketings but in incorrect order then the autograder may give you zero marks since partial marks are given only when the correct line is output at the correct location.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS: You may want to solve this problem using recursion. Define a function of the form
void generateBalanced(char *str, int n, int pos, int open, int close)

1. str: a string in which we will write the bracketing
2. n: tells us how many opening and closing brackets are needed in total
3. pos: tells us which is the next position in the string where characters have to be written
4. open: tells us how many opening brackets have we written so far
5. close: tells us how many closing brackets have we written so far

Make use of the variables open and close to make sure that you never have more closing brackets than open brackets at any point in the string. The recursion can be started off by calling the function as
generateBalanced(str, n, 0, 0, 0);

EXAMPLE:
INPUT
3

OUTPUT:
((()))
(()())
(())()
()(())
()()()

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2	(())
	()()
3	((()))
	(()())
	(())()
	()(())
	()()()
4	(((())))
	((()()))
	((())())
	((()))()
	(()(()))
	(()()())
	(()())()
	(())(())
	(())()()
	()((()))
	()(()())
	()(())()
	()()(())
	()()()()
1	()
5	((((()))))
	(((()())))
	(((())()))
	(((()))())
	(((())))()
	((()(())))
	((()()()))
	((()())())
	((()()))()
	((())(()))
	((())()())
	((())())()
	((()))(())
	((()))()()
	(()((())))
	(()(()()))
	(()(())())
	(()(()))()
	(()()(()))
	(()()()())
	(()()())()

	(()())(()) (()())()() (())((())) (())(()()) (())(())() (())()(()) (())()()() ()(((()))) ()((()())) ()((())()) ()((()))() ()(()(())) ()(()()()) ()(()())() ()(())(()) ()(())()() ()()((())) ()()(()()) ()()(())() ()()()(()) ()()()()()
6	(((((()))))) ((((()())))) ((((())()))) ((((()))())) ((((())))()) ((((()))))() (((()(())))) (((()()()))) (((()())())) (((()()))()) (((()())))() (((())(()))) (((())()())) (((())())()) (((())()))() (((()))(())) (((()))()()) (((()))())() (((())))(()) (((())))()() ((()((())))) ((()(()()))) ((()(())())) ((()(()))()) ((()(())))() ((()()(()))) ((()()()())) ((()()())()) ((()()()))() ((()())(())) ((()())()()) ((()())())() ((()()))(()) ((()()))()() ((())((()))) ((())(()()))

```
((())(())())
((())(()))()
((())()(()))
((())()()())
((())()())()
((())())(())
((())())()()
((()))((()))
((()))(()())
((()))(())()
((()))()(())
((()))()()()
(()(((()))))
(()((()())))
(()((())()))
(()((()))())
(()((())))()
(()(()(())))
(()(()()()))
(()(()())())
(()(()()))()
(()(())(()))
(()(())()())
(()(())())()
(()(()))(())
(()(()))()()
(()()((())))
(()()(()()))
(()()(())())
(()()(()))()
(()()()(()))
(()()()()())
(()()()())()
(()()())(())
(()()())()()
(()())((()))
(()())(()())
(()())(())()
(()())()(())
(()())()()()
(())(((())))
(())((()()))
(())((())())
(())((()))()
(())(()(()))
(())(()()())
(())(()())()
(())(())(())
(())(())()()
(())()((()))
(())()(()())
(())()(())()
(())()()(())
(())()()()()
()((((()))))
()(((()())))
()(((())()))
()(((()))())
```

	()((()))()
	()((()(())))
	()((()()()))
	()((()())())
	()((()()))()
	()((())(()))
	()((())()())
	()((())())()
	()((()))(())
	()((()))()()
	()(()((())))
	()(()(()()))
	()(()(())())
	()(()(()))()
	()(()()(()))
	()(()()()())
	()(()()())()
	()(()())(())
	()(()())()()
	()(())((()))
	()(())(()())
	()(())(())()
	()(())()(())
	()(())()()()
	()()(((())))
	()()((()()))
	()()((())())
	()()((()))()
	()()(()(()))
	()()(()()())
	()()(()())()
	()()(())(())
	()()(())()()
	()()()((()))
	()()()(()())
	()()()(())()
	()()()()(())
	()()()()()()