1. **Problem : Logic Pyramid**

Identify the logic behind the series   
  
6 28 66 120 190 276....   
  
The numbers in the series should be used to create a Pyramid. The base of the Pyramid will be the widest and will start converging towards the top where there will only be one element. Each successive layer will have one number less than that on the layer below it. The width of the Pyramid is specified by an input parameter N. In other words there will be N numbers on the bottom layer of the pyramid.   
  
The Pyramid construction rules are as follows

1. First number in the series should be at the top of the Pyramid
2. Last N number of the series should be on the bottom-most layer of the Pyramid, with Nth number being the right-most number of this layer.
3. Numbers less than 5-digits must be padded with zeroes to maintain the sanctity of a Pyramid when printed. Have a look at the examples below to get a pictorial understanding of what this rule actually means.

Example   
  
**If input is 2, output will be**   
  
 00006   
00028 00066   
  
**If input is 3, output will be**   
  
  00006   
 00028 00066   
00120 00190 00276   
  
Formal input and output specifications are stated below 

**Input Format:**   
  
First line of input will contain number N that corresponds to the width of the bottom-most layer of the Pyramid

**Output Format:**   
  
The Pyramid constructed out of numbers in the series as per stated construction rules

**Constraints:**

1. **0<N<=14**

**Sample Input and Output**

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 2 | https://www.tcscodevita.com/CodevitaV5/images/pyramid2.png |
| 2 | 3 | https://www.tcscodevita.com/CodevitaV5/images/pyramid1.png |

2. **Problem : Min Product Array**

The task is to find the minimum sum of Products of two arrays of the same size, given that k modifications are allowed on the first array. In each modification, one array element of the first array can either be increased or decreased by 2.   
  
Note- the product sum is Summation (A[i]\*B[i]) for all i from 1 to n where n is the size of both arrays

**Input Format:**

1. First line of the input contains n and k delimited by whitespace
2. Second line contains the Array A (modifiable array) with its values delimited by spaces
3. Third line contains the Array B (non-modifiable array) with its values delimited by spaces

**Output Format:**   
  
Output the minimum sum of products of the two arrays

**Constraints:**

1. **1 ≤ N ≤ 10^5**
2. **0 ≤ |A[i]|, |B[i]| ≤ 10^5**
3. **0 ≤ K ≤ 10^9**

**Sample Input and Output**

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 3 5 1 2 -3 -2 3 -5 | -31 |
| 2 | 5 3 2 3 4 5 4 3 4 2 3 2 | 25 |

**Explanation for sample 1:**   
  
Here total numbers are 3 and total modifications allowed are 5. So we modified A[2], which is -3 and increased it by 10 (as 5 modifications are allowed). Now final sum will be   
(1 \* -2) + (2 \* 3) + (7 \* -5)   
-2 + 6 - 35   
-31   
  
-31 is our final answer.   
  
**Explanation for sample 2:**   
  
Here total numbers are 5 and total modifications allowed are 3. So we modified A[1], which is 3 and decreased it by 6 (as 3 modifications are allowed).   
Now final sum will be   
(2 \* 3) + (-3 \* 4) + (4 \* 2) + (5 \* 3) + (4 \* 2)   
6 - 12 + 8 + 15 + 8   
25   
  
25 is our final answer.

3. **Problem : Consecutive Prime Sum**

Some prime numbers can be expressed as Sum of other consecutive prime numbers.   
For example   
  
5 = 2 + 3   
17 = 2 + 3 + 5 + 7   
41 = 2 + 3 + 5 + 7 + 11 + 13   
  
Your task is to find out how many prime numbers which satisfy this property are present in the range 3 to N subject to a constraint that summation should always start with number 2.   
  
Write code to find out number of prime numbers that satisfy the above mentioned property in a given range.

**Input Format:**   
  
First line contains a number N

**Output Format:**   
  
Print the total number of all such prime numbers which are less than or equal to N.

**Constraints:**

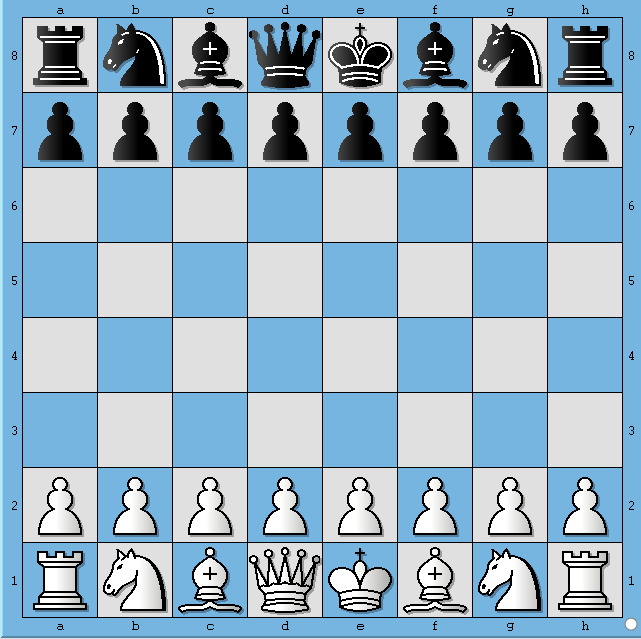
**1. 2<N<=12,000,000,000**

**Sample Input and Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNo.** | **Input** | **Output** | **Comment** |
| 1 | 20 | 2 | (Below 20, there are 2 such numbers: 5 and 17). 5=2+3 17=2+3+5+7 |
| 2 | 15 | 1 |  |

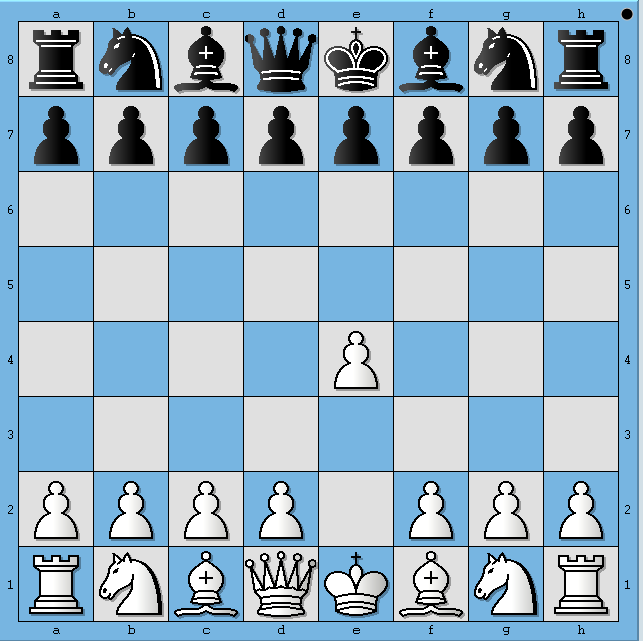
4. **Problem : Bishop Moves**

**Background**  
  
A Chess board position is accurately captured by Forsyth-Edwards notation and is abbreviated as FEN. A FEN "record" defines a particular game position, all in one text line and using only the ASCII character set. A FEN record contains six fields. A complete description of the FEN format to represent Chess positions can be found at [here](https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation)   
  
For the purpose of this problem only consider first of the six fields of FEN. Before we describe the problem, let us look at how FEN maps to a board position. The following 5 images show board positions and its corresponding FEN representation.

   
                    **Figure 1.**

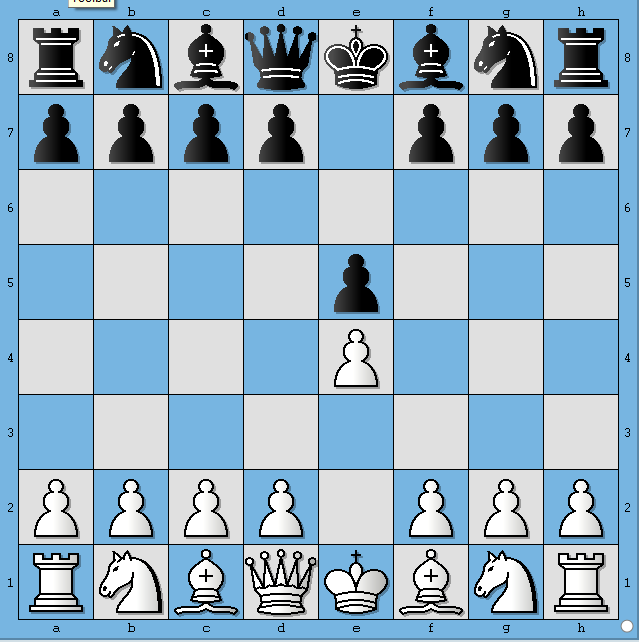
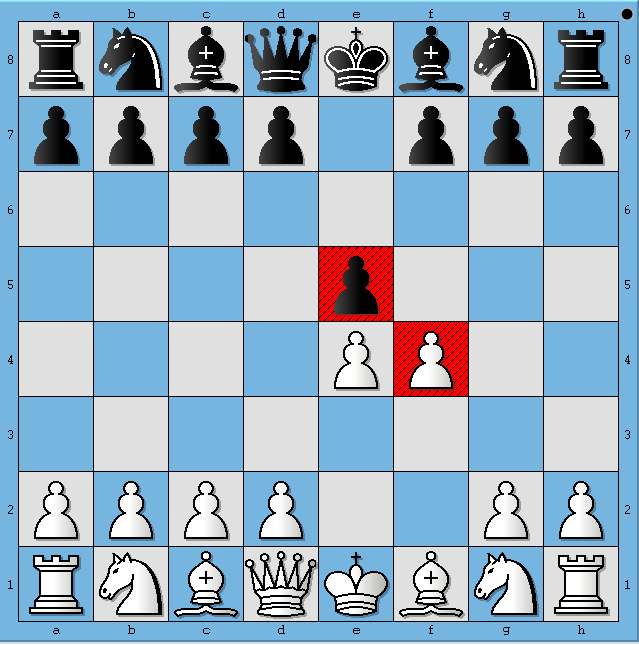
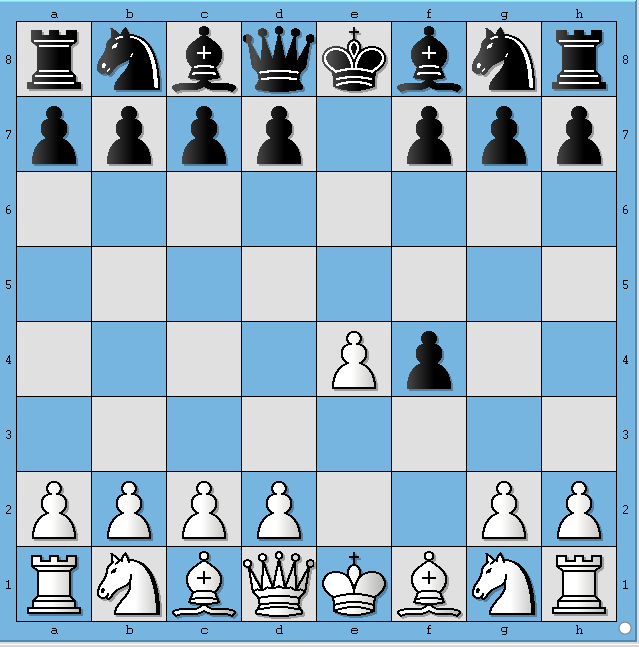
|  |
| --- |
| This board position depicts initial position before any side has made a move. In FEN format this board position is represented as   rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w |

Let's say, White plays e4. Then the board position looks like shown below 

   
                    **Figure 2.**

|  |
| --- |
| This board position depicts the Chess board after White has played e4. In FEN format this board position is represented as    rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b |

Similarly, 3 more half-moves are depicted in following diagrams 

                         **Figure 3.**                                                  **Figure 4.**                                                      **Figure 5.**

The FENs corresponding to Figure 3, 4 and 5 are represented as   
  
           3. rnbqkbnr/pppp1ppp/8/4P3/4P3/8/PPPP1PPP/RNBQKBNR w  
           4. rnbqkbnr/pppp1ppp/8/4p3/4PP2/8/PPPP2PP/RNBQKBNR b  
           5. rnbqkbnr/pppp1ppp/8/8/4Pp2/8/PPPP2PP/RNBQKBNR w   
  
**Wikipedia describes first field of FEN format as follows**   
  
Each rank is described, starting with rank 8 and ending with rank 1; within each rank, the contents of each square are described from file "a" through file "h". Following the [Standard Algebraic Notation](https://en.wikipedia.org/wiki/Algebraic_chess_notation) (SAN), each piece is identified by a single letter taken from the standard English names (pawn = "P", knight = "N", bishop = "B", rook = "R", queen = "Q" and king = "K").[1] White pieces are designated using upper-case letters ("PNBRQK") while black pieces use lowercase ("pnbrqk"). Empty squares are noted using digits 1 through 8 (the number of empty squares), and "/" separates ranks.   
  
The second field denotes whose move it is now. "w" depicts that it is White's turn to play and "b" indicates that it is Black's turn to play   
  
**CodeVita Problem Statement**   
  
Given a board position in FEN format, your task is to find out all the move(s) that Bishop(s) of the playing side can make. 

**Input Format:**

1. First line contains single FEN record, which corresponds to a particular board position and also indicates whose turn it is.

**Output Format:**

1. The output must be printed as follows
   1. All legal moves that Bishop can make must be in the format "[<Move Format>]"
   2. Where <Move Format> is move represented in format "[fromSquare][toSquare]"
2. See Example section for better understanding of output format
3. Follow Output printing specification to print the output in required format

**Constraints:**

1. Since we focus on only first two parts of the FEN, we are essentially ignoring possibility of Castling. **Hence our test cases don't contain FENs which give rise to such positions.**

**Sample Input and Output**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNo.** | **Input** | **Board Depiction** | **Output** |
| 1 | 3k4/8/8/2P1P3/3B4/2R1R3/8/4K3 w | https://www.tcscodevita.com/CodevitaV5/images/ex1.png | [] |
| 2 | 3k4/8/8/4P3/3B4/2R1R3/8/4K3 w | https://www.tcscodevita.com/CodevitaV5/images/ex2.png | [d4a7, d4b6, d4c5] |
| 3 | 3k4/8/8/8/3B4/8/8/1B2K3 w | https://www.tcscodevita.com/CodevitaV5/images/ex3.png | [d4h8, d4a7, d4g7, d4b6, d4f6, d4c5, d4e5, d4c3, d4e3, d4b2, d4f2, d4a1, d4g1, b1h7, b1g6, b1f5, b1e4, b1d3, b1a2, b1c2] |

**Print Specification:**

1. Should start with "[" and end with "]"
2. If more than one move is possible, moves should be separated by a comma followed by whitespace
3. Moves of a single bishop should be printed in *Move Format*. Scan the board from 8th rank to 1st rank from a-file to h-file. Whichever square gets hit first, that move should be printed first.
4. If more than one bishop exists for side to move, then start scanning for bishop from 8th rank to 1st rank, left to right i.e. from a-file to h-file. Whichever bishop appears first, print all moves for that bishop first.
5. Verify your understanding of how printing should happen against examples shown above

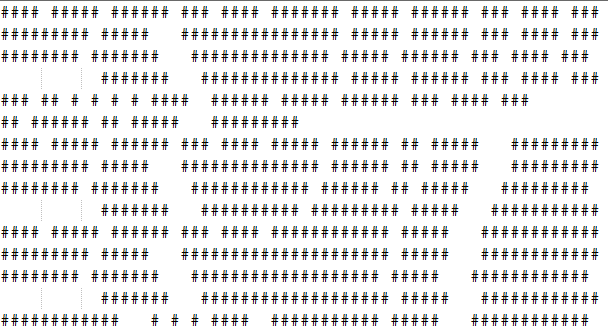
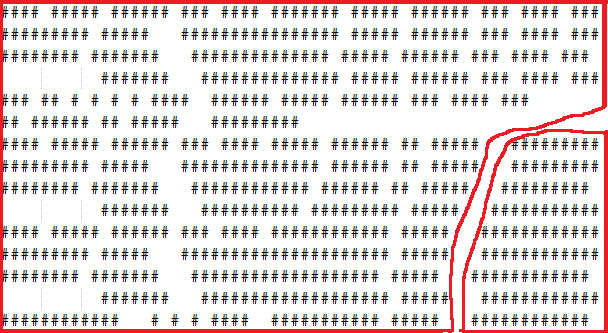
5. **Problem : Continents and Oceans**

There will be several continents on a world map. Idea is to identify the number of continents and the mass in each.   
  
The land is denoted by '#' and ocean by spaces. You have to find the number of continents (connected masses of land) and print the number of # in the continent.   
  
Connected mass of land means that # can be connected in any way i.e. horizontally, vertically or diagonally.

**Input Format:**

1. Path to a file, for example /tmp/T1.txt
2. This file contains 15 rows and 60 columns depicting a map. See Examples section below.

**Output Format:**   
  
Number of # per continent in descending order of size and total number of continents in the format depicted below   
  
Island 1: <Biggest mass> say 100  
Island 2: <Smallest mass> say 50  
Number of continents: 2

**Example 1:**   
  
   
Output:  
Island 1: 593  
Island 2: 98  
Number of continents: 2   
  
   
Output:  
Island 1: 593  
Island 2: 98  
Number of continents: 2

**Example 2:**   
  
   
Output:  
Island 1: 805  
Island 2: 68  
Number of continents: 2

6.

**Problem : Football League Table**

**Statement :**   
  
All major football leagues have big league tables. Whenever a new match is played, the league table is updated to show the current rankings (based on Scores, Goals For (GF), Goals Against (GA)). Given the results of a few matches among teams, write a program to print all the names of the teams in ascending order (Leader at the top and Laggard at the bottom) based on their rankings.   
  
**Rules: :**

1. A win results in 2 points, a draw results in 1 point and a loss is worth 0 points.
2. The team with most goals in a match wins the match.
3. Goal Difference (GD) is calculated as Goals For (GF) - Goals Against (GA)
4. Ranking is decided as follows
   1. Team with maximum points is ranked 1 and minimum points is placed last
   2. Ties are broken as follows
      1. Teams with same points are ranked according to Goal Difference(GD).
      2. If Goal Difference(GD) is same, then team with higher Goals For is ranked ahead
      3. If GF are same, the teams should be at the same rank but they should be printed in case-insensitive alphabetic according of the team names.
5. Teams can play maximum of two matches against each other - Home and Away matches respectively
6. More than 2 matches of same teams, should be considered as Invalid Input
7. A team can't play matches against itself, hence if team names are same for a given match, it should be considered Invalid Input

**Input Format:**   
  
First line of input will contain number of teams (N)  
Second line contains names of the teams (Na) delimited by a whitespace character   
Third line contains number of matches (M) for which results are available  
Next M lines contain a match information tuple {T1 T2 S1 S2}, where tuple is comprised of the following information

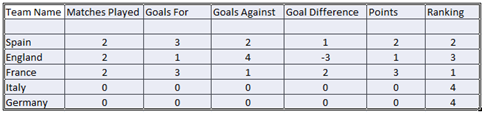
* T1 - Name of the first team
* T2 - Name of the second team
* S1 - Goals scored by the first team
* S2 - Goals scored by the second team

**Output Format:**   
  
Team names in order of their rankings, one team per line  
OR  
Print "Invalid Input" where appropriate.

**Constraints:**

1. **0<N<=10,000**
2. **0<=S1,S2**

**Example:**   
  
Consider 5 teams Spain, England, France, Italy and Germany with the following fixtures:   
  
Match 1: Spain vs. England (3-0)   
(Spain gets 2 points, England gets 0)  
Match 2: England vs. France (1-1)   
(England gets 1 point, France gets 1)   
Match 3: Spain vs. France (0-2)  
(Spain gets 0 points, France gets 2)

   
**Table 1. Points Table after 3 matches**

Since, Italy and Germany are tied for points, goals difference is checked. Both have same, so, Goals For is checked. Since both are same. Germany and Italy share the 4th rank. Since Germany appears alphabetically before Italy, Germany should be printed before Italy.   
  
**Then the final result is:**  
France  
Spain  
England  
Germany  
Italy 

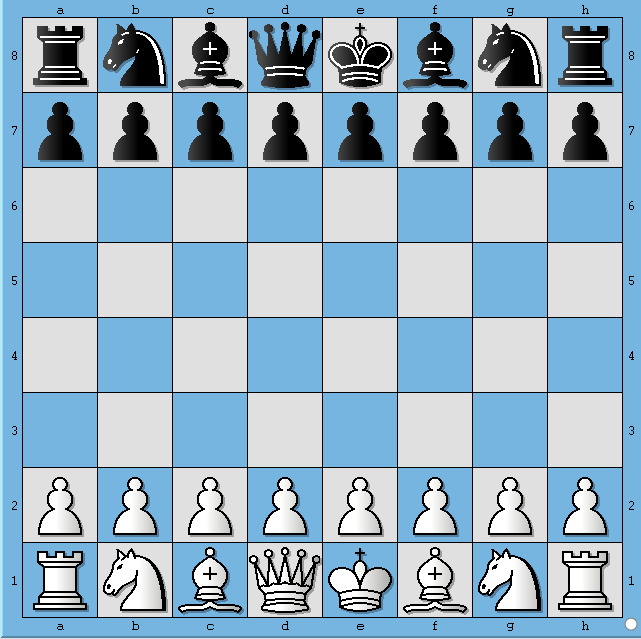
**Sample Input and Output**

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 5 Spain England France Italy Germany 3 Spain England 3 0 England France 1 1 Spain France 0 2 | France Spain England Germany  Italy |
| 2 | 5 Spain England France Italy Germany  3 Spain England 3 0 England France 1 1 Spain Spain 0 2 | Invalid Input |

7.

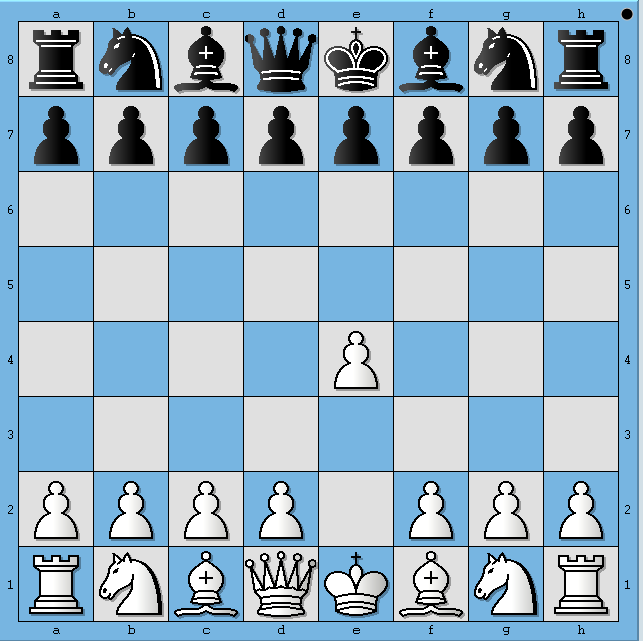
**Problem : Mate In One**

**Background**   
  
A Chess board position is accurately captured by Forsyth-Edwards notation and is abbreviated as FEN. A FEN "record" defines a particular game position, all in one line of text and using only the ASCII character set. A FEN record consists of six fields. A complete description of the FEN format to represent Chess positions can be found [here](https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation)   
  
For the purpose of this problem, only consider first of the six fields of FEN. Before we describe the problem, let us look at how FEN maps to a board position. The following 5 images show board positions and its corresponding FEN representation.

   
                    **Figure 1.**

|  |
| --- |
| This board position depicts initial position before any side has made a move. In FEN format this board position is represented as    rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR |

Let's say, White plays e4. Then the board position looks like shown below 

   
                    **Figure 2.**

|  |
| --- |
| This board position depicts the Chess board after White has played e4. In FEN format this board position is represented as    rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR |

Similarly, 3 more half-moves are depicted in following diagrams 

|  |  |  |
| --- | --- | --- |
| https://www.tcscodevita.com/CodevitaV5/images/cv15R2_R1P2_fig3.png | https://www.tcscodevita.com/CodevitaV5/images/cv15R2_R1P2_fig4.png | https://www.tcscodevita.com/CodevitaV5/images/cv15R2_R1P2_fig5.png |
| **Figure 3.** | **Figure 4.** | **Figure 5.** |

The FENs corresponding to Figure 3, 4 and 5 are represented as   
  
           3. rnbqkbnr/pppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR  
           4. rnbqkbnr/pppp1ppp/8/4p3/4PP2/8/PPPP2PP/RNBQKBNR  
           5. rnbqkbnr/pppp1ppp/8/8/4Pp2/8/PPPP2PP/RNBQKBNR   
  
**Wikipedia describes first field of FEN format as follows**   
  
Piece placement (from white's perspective). Each rank is described, starting with rank 8 and ending with rank 1; within each rank, the contents of each square are described from file "a" through file "h". Following the [Standard Algebraic Notation](https://en.wikipedia.org/wiki/Algebraic_chess_notation) (SAN), each piece is identified by a single letter taken from the standard English names (pawn = "P", knight = "N", bishop = "B", rook = "R", queen = "Q" and king = "K").[1] White pieces are designated using upper-case letters ("PNBRQK") while black pieces use lowercase ("pnbrqk"). Empty squares are noted using digits 1 through 8 (the number of empty squares), and "/" separates ranks   
  
**Statement**   
  
Given a board position in FEN format, your task is to find out all move(s) that lead to a forced mate. White to play and win in 1 move. 

**Input Format:**

1. First line contains single FEN record, which corresponds to a particular board position
2. Second line contains -1 which indicates the end of input

**Output Format:**

1. The output must be printed as follows
   1. A string "Mating moves " followed by "[<move format>]"
   2. Where <move format> is move represented in format "[Piece][fromSquare]-[Piece][toSquare]"
   3. If a mating move involves a capture then represent it as "[Piece][fromSquare]-[Piece]x[toSquare]"
   4. If the Piece inflicting a mate is a pawn represent it only as [fromSquare]-[toSquare]
   5. If there is more than one mating move, then follow the rules given below
      1. Moves must be sorted alphabetically according to their ascii values
      2. Two moves must be separated by a comma followed by a white space characters
      3. Between the last move and terminating "]" character there should be no space or comma
2. See Example section for better understanding of output format

**Constraints:**

1. The board position will always be White to move and mate in 1
2. Since we focus on only first part of the FEN, we are essentially ignoring possibility of Castling being a mating move. **Hence our test cases don't contain FENs which give rise to such positions.**
3. There is no need to handle [*En Passant*](https://en.wikipedia.org/wiki/En_passant) positions. There are no test cases involving *En Passant* moves.
4. No need to implement [*pawn promotion*](https://en.wikipedia.org/wiki/Promotion_(chess)) rules. Our test cases do not contain positions which will lead to a pawn getting promoted and inflicting a mate.
5. There are no test cases in which capture by a white pawn leads to a mate.

**Sample Input and Output**

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 3k4/8/3p4/7B/3Q4/2R1R3/8/5K2 -1 | Mating moves [Qd4-Qxd6] |
| 2 | 8/R7/3k4/8/2Q1P1B1/8/8/2K1R3 -1 | Mating moves [Qc4-Qc7, Qc4-Qd5, e4-e5] |

**Explanation:**

|  |  |
| --- | --- |
| **Board position for sample input 1:**  https://www.tcscodevita.com/CodevitaV5/images/cv15ph2r1_R1P7_fig6.jpg                      **Figure 6.** | **Board position for sample input 2:**  https://www.tcscodevita.com/CodevitaV5/images/cv15ph2r1_R1P7_fig7.jpg                      **Figure 7.** |

8.

**Problem : Where's my car!**

The year is 2050. The population surge has taken over our metropolitan cities. High rise buildings are on a rise. And as usual, with the increase in population, the problem of parking in the city has increased manifold.   
  
To reduce the problem of parking, the government has built multi-storey parking lots all over the city. Imagine the city as an X-Y grid. And there are roads connecting all the neighbouring grid points except diagonals. And there is a pre-defined intersection interval 'I' for parking lots such that at every I th intersection, there is a parking lot, starting from (0,0). For example, for a city of grid size 4x7 and I=3, you'll have 6 parking lots at (0,0), (0,3), (0,6), (3,0), (3,3) and (3,6).   
  
Now all the cars have been fitted with self-driving mechanism. So whenever you get out of a car at any point in the grid, it will choose the nearest parking lot and automatically drive to it. If two parking lots are at equal distance from where you left, it will choose the parking lot with the lowest X-coordinate first, and if X-coordinates are same, the lowest Y-coordinate.   
  
At the parking lot, the cars will start getting parked from the ground floor and in the first available slot. As each floor gets filled up, newer cars will start parking on floors above them. Assume all the parking lots in the city have unlimited number of floors and a common maximum capacity of each floor 'C'.   
  
Now whenever the owner wants to know where his car is parked or wants to retrieve it, he'll open the app 'Where's my car!' and insert his car number and the app will tell him the coordinates of the parking lot, the floor number and the slot number.

**Input Format:**   
  
The first line will contain a positive integer T determining the number of test cases.   
For each test case, the first line will contain four positive integers X, Y, I and C, delimited by space  
where,

* X,Y denote the grid size of the city,
* I denotes the intersection interval for parking lots
* C denotes the maximum capacity of a floor in all the parking lots in the city.

The next line will contain a positive integer N denoting the number of events following.

* An event can be parking of a car(P) or retrieval of a car(R ).
* The next N lines will begin with either a character 'P' or 'R'.
* If the line begins with P, it will also contain two positive integers, x and y denoting the current coordinate of the car where it is left off and has to drive itself to the nearest parking lot, and the car number S which is a unique 10-digit number for each car and can contain a mix of numbers and alphabet.
* If the line begins with 'R', it will contain the car number to retrieve.

**Output Format:**   
  
For each test case, output the following.  
First output the total number of parking lots in the city.   
Then, for every retrieval event in the input list, output 4 space separated integers, p,q,r,s where

* p,q is the coordinate of the parking lot where the car is parked in,
* r is the floor number and s is the slot number.
* For brevity, output each number p,q,r,s, modulo 10.

**Constraints:**

1. **1<=T<=10**
2. **1<= X,Y < 2,000,000,000**
3. **1<= C,N < 100,000**
4. **1<= I <= X,Y**
5. **0<= p <= X**
6. **0<= q <= Y**
7. **Max no. of parking lots < 1,000,000**

**Sample Input and Output**

|  |  |  |
| --- | --- | --- |
| **SNo.** | **Input** | **Output** |
| 1 | 1 4 7 2 2 5 P 1 1 MH04CZ2123 P 3 2 BR11HY2863 R MH04CZ2123 P 3 3 KA14NU7644 R KA14NU7644 | 12 0 0 0 1 2 2 0 2 |
| 2 | 1 4 7 2 2 5 P 2 1 MH04CZ2123 P 3 5 BR11HY2863 R MH04CZ2123 P 3 4 KA14NU7644 R KA14NU7644 | 12 2 0 0 1 2 4 0 2 |