L8: BackTracking Practice Questions

1-Tut : N-Queen Problem

Send Feedback

You are given N, and for a given N x N chessboard, find a way to place N queens such that no queen can attack any other queen on the chess board. A queen can be killed when it lies in the same row, or same column, or the same diagonal of any of the other queens. You have to print all such configurations.

Note: Don't print anything if there isn't any valid configuration.

Input Format:

The first line of input contains an integer, that denotes the value of N.

Output Format :

In the output, you have to print every board configuration in a new line. Every configuration will have N*N board elements printed row wise and are separated by space. The cells where queens are safe and placed, are denoted by value 1 and remaining all cells are denoted by value 0. Please refer to sample test cases for more clarity.

Constraints :

1 <= N <= 10

Time Limit: 1 second

Sample Input 1:

4

Sample Output 1 :

0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0

0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0

Explanation:

The first and second configurations are shown in the image below. Here, 1 denotes the placement of a queen and 0 denotes an empty cell. Please note that in both the configurations, no pair of queens can kill each other.

0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0

First Configuration

0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0

Second Configuration

```
#include<bits/stdc++.h>
using namespace std;

void print(int **board,int n){
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cout<<board[i][j]<<" ";
        }
    }
    cout<<endl;
}

bool ispossible(int **board,int n,int i,int j){
    //check col
    for(int row=0;row<n;row++){
        if(board[row][j]==1){
            return false;
        }
    }
    //check row
    for(int col=0;col<n;col++){
        if(board[i][col]==1){
            return false;
        }
    }
    //check diagonal left bend top
    for(int row=i,col=j;row>=0&&col>=0;row--,col--){
        if(board[row][col]==1){
            return false;
        }
    }
    //check diagonal right bend top
    for(int row=i,col=j;row<n&&row>=0&&col>=0&&col<n;row--,col++){
        if(board[row][col]==1){
            return false;
        }
    }
    return true;
}

void find_all_possible(int **board,int n,int i,int j){
    //crossed limits
    if(i==n){
        print(board,n);
    }
    //check for all cols of a particular row
        for(int col=j;col<n;col++){ //col
            if(ispossible(board,n,i,col)){
                board[i][col]=1;
```

```
                find_all_possible(board,n,i+1,0);
                board[i][col]=0;
            }
        }
}

void placeNQueens(int n){
    int **board=new int*[n];
    for(int i=0;i<n;i++){
        board[i]=new int[n];
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            board[i][j]=0;
        }
    }
            find_all_possible(board,n,0,0);
    for(int i=0;i<n;i++){
        delete board[i];
    }
    delete board;
}

int main(){

   int n;
   cin >> n ;
   placeNQueens(n);

}
```

2-Tut : Rat In A Maze Problem

Send Feedback

You are given an integer N and a binary 2D array of size N*N. The given binary matrix represents a maze and a rat starts from the top left cell and has to reach to the bottom right cell of the maze. The rat can move in all the four directions: up, down, left and right.

In the given binary matrix, 0 signifies the cell is a dead end and 1 signifies the cell can be used to move from source to destination cell.

You have to print all the paths, following which rat can move from the top left cell to the bottom right cell of the given binary matrix.

Input Format

The first line of input contains an integer that denotes the value of N.

Each of the following N lines denote rows of the binary matrix and contains either 0 or 1. Each row of the binary matrix contains N elements.

Output Format :

Every possible solution has to be printed on a separate line. Each of the possible solution will have N*N elements, printed row wise and separated by space. The cells that are part of the path should be 1 and remaining all cells should be 0. Please refer to sample test cases for more clarity.

Constraints

1 <= N <= 18

0 <= Number of cells with value 1 <= 15

Time Limit: 1 second

Sample Input 1 :

3

1 0 1

1 0 1

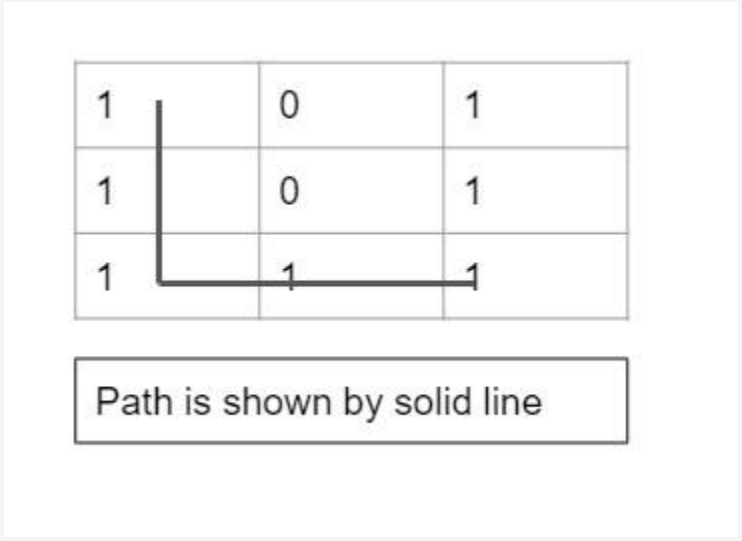
1 1 1

Sample Output 1 :

1 0 0 1 0 0 1 1 1

Explanation :

Only 1 path is possible. The path is shown in the image below.



Sample Input 2 :

3

1 0 1

1 1 1

1 1 1

Sample Output 2 :

1 0 0 1 1 1 1 1 1

1 0 0 1 0 0 1 1 1

1 0 0 1 1 0 0 1 1

1 0 0 1 1 1 0 0 1

Explanation:

As described in the Sample Output 2, four paths are possible.

```
/*
        Note:
        To get all the test cases accepted, make recursive calls in following order: Top, Down, Left, Right.
        This means that if the current cell is (x, y), then order of calls should be: top cell (x-1, y), down cell (x+1, y), left cell (x, y-1) and right cell (x, y+1).
*/
#include<bits/stdc++.h>
using namespace std;


void printSolution(int** solution,int n){
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        cout << solution[i][j] << " ";
                }
        }
        cout<<endl;
}
void mazeHelp(int maze[][20],int n,int** solution,int x,int y){


        if(x == n-1 && y == n-1){
                solution[x][y] =1;
                printSolution(solution,n);
        solution[x][y] =0;
                return;
        }
        if(x>=n || x<0 || y>=n || y<0 || maze[x][y] ==0 || solution[x][y] ==1){
                return;
        }
        solution[x][y] = 1;
        mazeHelp(maze,n,solution,x-1,y);
        mazeHelp(maze,n,solution,x+1,y);
        mazeHelp(maze,n,solution,x,y-1);
        mazeHelp(maze,n,solution,x,y+1);
        solution[x][y] = 0;
}
void ratInAMaze(int maze[][20], int n){

  int** solution = new int*[n];
  for(int i=0;i<n;i++){
        solution[i] = new int[n];
  }
  mazeHelp(maze,n,solution,0,0);


}
#include<bits/stdc++.h>
```

```
using namespace std;
int main() {

        // Write your code here
    int N;cin >> N;
    int maze[20][20];

    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++){
            cin >> maze[i][j];
        }
    }
    ratInAMaze(maze,N);
        return 0;
}

```

Live Problem -1 : https://www.codechef.com/problems/AX06

3-Tut : Crossword Problem

Send Feedback

Coding Ninjas has provided you a crossword of 10*10 grid. The grid contains '+' or '-' as its cell values. Now, you are also provided with a word list that needs to placed accurately in the grid. Cells marked with '-' are to be filled with word list.

For example, The following is an example for the input crossword grid and the word list.

```
+-++++++++
+-++-+++++
+-------++
+-++-+++++
+-++-+++++
+-++-+++++
++++-+++++
++++-+++++
++++++++++
----------
```

CALIFORNIA;NIGERIA;CANADA;TELAVIV

Output for the given input should be:

+C++++++++

+A++T+++++

+NIGERIA++

+A++L+++++

+D++A+++++

+A++V+++++

++++I+++++

++++V+++++

++++++++++

CALIFORNIA

Note: We have provided such test cases that there is only one solution for the given input.

Input format:

The first 10 lines of input contain crossword. Each of 10 lines has a character array of size 10. Input characters are either '+' or '-'.

The next line of input contains the word list, in which each word is separated by ';'.

Output format:

Print the crossword grid, after placing the words of word list in '-' cells.

Constraints

The number of words in the word list lie in the range of: [1,6]

The length of words in the word list lie in the range: [1, 10]

Time Limit: 1 second

Sample Input 1:

+-++++++++

+-++-+++++

+-------++

+-++-+++++

+-++-+++++

+-++-+++++

++++-+++++

++++-+++++

++++++++++

CALIFORNIA;NIGERIA;CANADA;TELAVIV

Sample Output 1:

+C++++++++

+A++T+++++

+NIGERIA++

+A++L+++++

+D++A+++++

+A++V+++++

++++I+++++

++++V+++++

++++++++++

CALIFORNIA

```
#include<bits/stdc++.h>
using namespace std;
#include<iostream>
#include<string>
#include<vector>
using namespace std;
#define n 10
void printer(char cross[n][n])
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			cout << cross[i][j];
		}
		cout << endl;
	}
	cout << endl;
}
bool isvalid_vertical(char cross[n][n], int r, int c, string word)
{
	int j = r;
	for (int i = 0;i < word.length();i++)
	{
		if (j > 9)
			return false;
		if (cross[j][c] == '+' || (cross[j][c] != '-' && cross[j][c] != word[i]))
			return false;
		if (cross[j][c] == '-' || cross[j][c] == word[i])
			j++;
	}
	return true;
}
bool isvalid_horizontal(char cross[n][n], int r, int c, string word)
{
	int j = c;
	for (int i = 0;i < word.length();i++) {
		if (j > 9)
			return false;
		if (cross[r][j] == '+' || (cross[r][j] != '-' && cross[r][j] != word[i]))
			return false;
		if (cross[r][j] == '-' || cross[r][j] == word[i])
			j++;
	}
	return true;
}
void set_vertical(char cross[n][n], int r, int c, vector<bool>& v, string word)
{
	int row = r;
```

```
50.             for (int i = 0; i < word.length(); i++)
51.             {
52.                     if (cross[row + i][c] == '-')
53.                     {
54.                             cross[row + i][c] = word[i];
55.                             v.push_back(true);
56.                     }
57.                     else
58.                     {
59.                             v.push_back(false);
60.                     }
61.             }
62. }
63. void reset_vertical(char cross[n][n], int r, int c, vector<bool>v)
64. {
65.             int row = r;
66.             for (int i = 0; i < v.size(); i++)
67.             {
68.                     if (v[i])
69.                     {
70.                             cross[row + i][c] = '-';
71.                     }
72.             }
73. }
74. void set_horizontal(char cross[n][n], int r, int c, vector<bool>& v, string word)
75. {
76.             int col = c;
77.             for (int i = 0; i < word.length(); i++)
78.             {
79.                     if (cross[r][col + i] == '-')
80.                     {
81.                             cross[r][col + i] = word[i];
82.                             v.push_back(true);
83.                     }
84.                     else
85.                     {
86.                             v.push_back(false);
87.                     }
88.
89.             }
90. }
91. void reset_horizontal(char cross[n][n], int r, int c, vector<bool>v)
92. {
93.             int col = c;
94.             for (int i = 0; i < v.size(); i++)
95.             {
96.                     if (v[i])
97.                     {
98.                             cross[r][col + i] = '-';
```

```
                }
            }
        }

        bool crossword(char cross[n][n], vector<string>words, int index)
        {
            //base case is if index become == v.size()----> then print the crossword;
            if (index >= words.size())
            {
                printer(cross);
                return true;
            }
            for (int r = 0; r < n; r++)
            {
                for (int c = 0; c < n; c++)
                {
                    if (cross[r][c] == '-' || cross[r][c] == words[index][0])
                    {
                        if (isvalid_vertical(cross, r, c, words[index]))
                        {
                            //maintain a helper array
                            vector<bool>helper;
                            set_vertical(cross, r, c, helper, words[index]);
                            //pass this helper array in the function set_vertical where this helper array is a boolean array.
                            //set vertical(helper, ...., ....)
                            if (crossword(cross, words, index + 1))
                            {
                                return true;
                            }
                            //setvertical call karne k baad crossword call karna hai for next index words.
                            // crossword(cross, words, index+1)
                            // ye jo crossword function hai wo boolean hai. if it returns true, just return.
                            // or if it return false, call the reset vertical function---> reset_vertical(helper, ....)
                            reset_vertical(cross, r, c, helper);
                            // agar is valid vertical false return karta hai to hum isvalid_horizontal mei check kar lenge.
                            // isme bhi vhi hoga helper, set_horizontal, crossword, reset_horizontal.
                        }
                        if (isvalid_horizontal(cross, r, c, words[index]))
                        {
                            vector<bool>helper;
                            set_horizontal(cross, r, c, helper, words[index]);
                            if (crossword(cross, words, index + 1))
                            {
```

```
                                                return true;
                                            }
                                            reset_horizontal(cross, r, c, helper);
                                        }
                                    }
                                }
                            }
                        }
                        return false;
                    }
                    int main()
                    {
                        char arr[n][n];
                        for (int i = 0; i < n; i++)
                        {
                            string s;
                            cin >> s;
                            for (int j = 0; j < n; j++)
                            {
                                arr[i][j] = s[j];
                            }
                        }
                        string s;
                        cin >> s;
                        vector<string>words;
                        for (int i = 0; i < s.length(); i++)
                        {
                            int j = i;
                            while (s[j] != ';' && j < s.length())
                            {
                                j++;
                            }
                            words.push_back(s.substr(i, j - i));
                            i = j;
                            j++;
                        }
                        bool x = crossword(arr, words, 0);
                    }
```

4-Ass : **Sudoku Solver**

Send Feedback

Coding Ninjas has provided you a 9*9 sudoku board. The board contains non zero and zero values. Non zero values lie in the range: [1, 9]. Cells with zero value indicate that the particular cell is empty and can be replaced by non zero values.

You need to find out whether the sudoku board can be solved. If the sudoku board can be solved, then print true, otherwise print false.

Input Format:

There are nine lines in input. Each of the nine lines contain nine space separated integers. These nine lines represent the sudoku board.

Output Format:

The first and only line of output contains boolean value, either true or false, as described in problem statement.

Constraints:

The cell values lie in the range: [0, 9]

Time Limit: 1 second

Note:

Input are given in a way that backtracking solution will work in the provided time limit.

Sample Input 1:

9 0 0 0 2 0 7 5 0

6 0 0 0 5 0 0 4 0

0 2 0 4 0 0 0 1 0

2 0 8 0 0 0 0 0 0

0 7 0 5 0 9 0 6 0

0 0 0 0 0 0 4 0 1

0 1 0 0 0 5 0 8 0

0 9 0 0 7 0 0 0 4

0 8 2 0 4 0 0 0 6

Sample Output 1:

true

```
1.  #include<iostream>
2.  using namespace std;
3.
4.  bool find_empty(int board[][9],int &row,int &col){
```

```
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            if(board[i][j]==0){
                row=i;
                col=j;
                return true;
            }
        }
    }
    return false;
}

bool check_row(int board[][9],int i,int j,int num){
    for(int row=0;row<9;row++){
        if(board[row][j]==num){
            return false;
        }
    }
    return true;
}

bool check_col(int board[][9],int i,int j,int num){
    for(int col=0;col<9;col++){
        if(board[i][col]==num){
            return false;
        }
    }
    return true;
}

bool check_block(int board[][9],int i,int j,int num){
    int checki=i>=0&&i<3?0:i>=3&&i<6?3:6;
    int checkj=j>=0&&j<3?0:j>=3&&j<6?3:6;
    for(int row=checki;row<checki+3;row++){
        for(int col=checkj;col<checkj+3;col++){
            if(board[row][col]==num){
                return false;
            }
        }
    }
    return true;
}

bool solve_sudoku(int board[][9]){
    int row,col;
    bool found=find_empty(board,row,col);
    if(found){
        // go from 1 to 9 and
        for(int i=0;i<=9;i++){
```

```
            bool rowcheck=check_row(board,row,col,i);
            bool colcheck=check_col(board,row,col,i);
            bool blockcheck=check_block(board,row,col,i);
            if(rowcheck&&colcheck&&blockcheck){
                board[row][col]=i;
                bool findifvalid=solve_sudoku(board);
                if(findifvalid){
                    return true;
                }
                board[row][col]=0;
            }
        }
        // check row
        // check col
        // check block
        // fill that row,col and call solve_sudoku again
        // if returned true - return true
        // if returned false - try for another num
    }else{
        // no empty means sudoku is filled or solved
        return true;
    }
    return false;
}

bool sudokuSolver(int board[][9]){
    return solve_sudoku(board);

}

int main(){

   int n = 9;
   int board[9][9];
   for(int i = 0; i < n ;i++){
            for(int j = 0; j < n; j++){
            cin >> board[i][j];
        }
   }
   if(sudokuSolver(board)){
            cout << "true";
   }else{
            cout << "false";
   }
}

```

5-Ass : Subset Sum

Send Feedback

You are given an array of integers and a target K. You have to find the count of subsets of the given array that sum up to K.

Note:

1. Subset can have duplicate values.

2. Empty subset is a valid subset and has sum equal to zero.

Input Format:

The first line of input will contain an integer T, that denotes the value of number of test cases.

Each test case contains two lines. The first line of each test case will contain two space-separated integers N and K, where N is the size of the given array and K is the target value.

The second line of each test case will contain N space separated integers denoting the elements of the input array.

Output Format ;

For each test case, print the number of subsets that sum upto K in a separate line.

Constraints

1 <= T <= 50

2 <= N <= 15

0 <= array[i] <= 10^9

0 <= K <= 10^9

Time Limit: 1 second

Sample Input 1:

1

5 6

2 4 4 3 1

Sample Output 1:

3

Explanation:

Following are the three subsets, that sum upto K=6:

1. [2, 4], Element 4 in this subset is the one at index 1

2. [2, 4], Element 4 in this subset is the one at index 2

3. [2, 3, 1]

Sample Input 2:

2

8 8

1 4 1 3 1 1 2 3

8 2

4 1 4 4 2 4 1 1

Sample Output 2:

30

4

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

void countSubsetSum(ll* a, ll n, int index, ll currSum, ll target, ll &count){
    if(index>n) return ;
    if(index==n){
        if(currSum==target) count++;
        return;
    }
    countSubsetSum(a,n,index+1,currSum+a[index],target,count);
    countSubsetSum(a,n,index+1,currSum,target,count);
}
int main(){

    int t;
```

```
    cin>>t;
    while(t--){
        ll n,k;
        cin>>n>>k;
        ll* a = new ll[n];
        for(int i = 0; i<n; i++) cin>>a[i];
        ll ans = 0;
        countSubsetSum(a,n,0,0,k,ans);
        cout << ans << endl;

    }
}
```