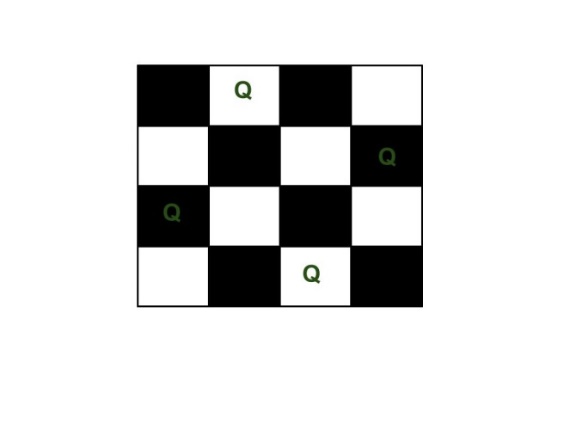
**TASK 11**

**Backtracking - n-Queen’s problem CO4 K3**

Two players are playing a game on a N\*N chessboard. The rules of the game are as follows: no two queens attack each other. For example, the following is a solution for the 4 Queen problem. 

The expected output is in form of a matrix that has ‘Q’s for the blocks where queens are placed and the empty spaces  are represented by ‘.’s . For example, the following is the output matrix for the above 4 queen solution.

*. . Q .   
Q . . .   
. . . Q   
. Q . .*

**Test Case 1:** Implement Rat in a Maze problem by applying the concept of backtracking.

**Test Case 2:** Modify the n-queens problem to handle the value of n in the range from 1 to 3.

**Aim:**

Create to c program to Implement N Queen's problem using Backtracking algorithm

**Algorithm**:

Step 1 : Create and initialize the variable board[] for the required size of chess board.

Step 2 : write a recursive function queen() to check for proper positioning of queen to check conflicts

Step 3:Write a function place() to check conflicts, If no conflict for desired position returns 1 otherwise returns 0.

Step 4 : Loop until all queens are properly placed in the board

**Program:**

#include<stdio.h>

#include<math.h>

#include <stdlib.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

//function for printing the solution

void print(int n)

{

int i,j;

printf("\n\nSolution %d:\n\n",++count);

for(i=1;i<=n;++i)

printf("\t%d",i);

for(i=1;i<=n;++i)

{

printf("\n\n%d",i);

for(j=1;j<=n;++j) //for nxn board

{

if(board[i]==j)

printf("\tQ"); //queen at i,j position

else

printf("\t-"); //empty slot

}

}

}

/\*funtion to check conflicts

If no conflict for desired postion returns 1 otherwise returns 0\*/

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

//checking column and digonal conflicts

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1; //no conflicts

}

//function to check for proper positioning of queen

void queen(int row,int n)

{

int column;

for(column=1;column<=n;++column)

{

if(place(row,column))

{

board[row]=column; //no conflicts so place queen

if(row==n) //dead end

print(n); //printing the board configuration

else //try queen with next position

queen(row+1,n);

}

}

}

**Output:**

- N Queens Problem Using Backtracking -

Enter number of Queens:4

Solution 1:

1 2 3 4

1 - Q - -

2 - - - Q

3 Q - - -

4 - - Q -

Solution 2:

1 2 3 4

1 - - Q -

2 Q - - -

3 - - - Q

4 - Q - -

**Test Case 1:** Implement Rat in a Maze problem by applying the concept of backtracking.

**Aim:**

Create to c program to Implement Implement Rat in a Maze problem by applying the concept of backtracking algorithm

Algorithm:

Step 1: Create a solution matrix, initially filled with 0’s.

Step 2:Create a recursive function, which takes initial matrix, output matrix and position of rat (i, j).

Step 3: if the position is out of the matrix or the position is not valid then return.

Step 4 :Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.

Step 5: Recursively call for position (i+1, j) and (i, j+1).

Step 6:Unmark position (i, j), i.e output[i][j] = 0.

Program:

// C program to solve Rat in a Maze problem using backtracking

#include <stdio.h>

#include <stdbool.h>

// Maze size

#define N 4

bool solveMazeUtil(int maze[N][N], int x, int y,int sol[N][N]);

// A utility function to print solution matrix sol[N][N]

void printSolution(int sol[N][N])

{

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

printf(" %d ", sol[i][j]);

printf("\n");

}

}

// A utility function to check if x, y is valid index for

// N\*N maze

bool isSafe(int maze[N][N], int x, int y)

{

// if (x, y outside maze) return false

if (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)

return true;

return false;

}

bool solveMaze(int maze[N][N])

{

int sol[N][N] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveMazeUtil(maze, 0, 0, sol) == false) {

printf("Solution doesn't exist");

return false;

}

printSolution(sol);

return true;

}

// A recursive utility function to solve Maze problem

bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])

{

// if (x, y is goal) return true

if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {

sol[x][y] = 1;

return true;

}

// Check if maze[x][y] is valid

if (isSafe(maze, x, y) == true) {

// Check if the current block is already part of

// solution path.

if (sol[x][y] == 1)

return false;

// mark x, y as part of solution path

sol[x][y] = 1;

/\* Move forward in x direction \*/

if (solveMazeUtil(maze, x + 1, y, sol) == true)

return true;

// If moving in x direction doesn't give solution

// then Move down in y direction

if (solveMazeUtil(maze, x, y + 1, sol) == true)

return true;

// If none of the above movements work then

// BACKTRACK: unmark x, y as part of solution path

sol[x][y] = 0;

return false;

}

return false;

}

// driver program to test above function

int main()

{

int maze[N][N] = { { 1, 0, 0, 0 },

{ 1, 1, 0, 1 },

{ 0, 1, 0, 0 },

{ 1, 1, 1, 1 } };

solveMaze(maze);

return 0;

}

Output:

1 0 0 0

1 1 0 0

0 1 0 0

0 1 1 1

**Test Case 2:** Modify the n-queens problem to handle the value of n in the range from 1 to 3.

**Aim:**

Create to c program to Implement N Queen's problem using Backtracking algorithm to handle the value of n in the range from 1 to 3

**Algorithm**:

Step 1 : Create and initialize the variable board[] for the required size of chess board.

Step 2 : write a recursive function queen() to check for proper positioning of queen to check conflicts

Step 3:Write a function place() to check conflicts, If no conflict for desired position returns 1 otherwise returns 0.

Step 4 : Loop until all queens are properly placed in the board

**Program:**

#include<stdio.h>

#include<math.h>

#include <stdlib.h>

int a[30],count=0;

int place(int pos) {

int i;

for (i=1;i<pos;i++) {

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void print\_sol(int n) {

int i,j;

count++;

printf("\n\nSolution #%d:\n",count);

for (i=1;i<=n;i++) {

for (j=1;j<=n;j++) {

if(a[i]==j)

printf("Q\t"); else

printf("\*\t");

}

printf("\n");

}}

void queen(int n) {

int k=1;

a[k]=0;

while(k!=0) {

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n) {

if(k==n)

print\_sol(n); else {

k++;

a[k]=0;

}

} else

k--;

}

}

int main() {

int i,n;

printf("Enter the number of Queens\n");

scanf("%d",&n);

queen(n);

printf("\nTotal solutions=%d",count);

}

**Output:**

Enter the number of Queens:1

Solution #1:

Q

Total solutions=1

Enter the number of Queens:2

Total solutions=0

Enter the number of Queens:3

Total solutions=0

**Result:**

Thus the N Queen's problem using Backtracking algorithm was executed successfully.