Backtracking | Set 3 (N Queen Problem)

We have discussed Knight's tour and Rat in a Maze problems in Set 1 and Set 2 respectively. Let us discuss N Queen as another example problem that can be solved using Backtracking.

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.

The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example following is the output matrix for above 4 queen solution.

```
              { 0,  1,  0,  0}
              { 0,  0,  0,  1}
              { 1,  0,  0,  0}
              { 0,  0,  1,  0}
```

Naive Algorithm
Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.

```
while there are untried conflagrations
{
   generate the next configuration
   if queens don't attack in this configuration then
   {
      print this configuration;
   }
}
```

Backtracking Algorithm
The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.  Do following for every tried row.
    a) If the queen can be placed safely in this row then mark this [row,
        column] as part of the solution and recursively check if placing
        queen here leads to a solution.
    b) If placing queen in [row, column] leads to a solution then return
        true.
    c) If placing queen doesn't lead to a solution then umark this [row,
        column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger
    backtracking.
```

Implementation of Backtracking solution

C/C++

```
/* C/C++ program to solve N Queen Problem using
   backtracking */
#define N 4
#include<stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
}

/* A utility function to check if a queen can
   be placed on board[row][col]. Note that this
   function is called when "col" queens are
   already placed in columns from 0 to col -1.
   So we need to check only left side for
   attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i=row, j=col; i>=0 && j>=0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i=row, j=col; j>=0 && i<N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

/* A recursive utility function to solve N
   Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed
      then return true */
    if (col >= N)
        return true;

    /* Consider this column and try placing
       this queen in all rows one by one */
    for (int i = 0; i < N; i++)
    {
        /* Check if queen can be placed on
```

```
          board[i][col] */
        if ( isSafe(board, i, col) )
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            if ( solveNQUtil(board, col + 1) )
                return true;

            /* If placing queen in board[i][col]
               doesn't lead to a solution, then
               remove queen from board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }

     /* If queen can not be place in any row in
        this colum col  then return false */
    return false;
}

/* This function solves the N Queen problem using
   Backtracking. It mainly uses solveNQUtil() to
   solve the problem. It returns false if queens
   cannot be placed, otherwise return true and
   prints placement of queens in the form of 1s.
   Please note that there may be more than one
   solutions, this function prints one  of the
   feasible solutions.*/
bool solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };

    if ( solveNQUtil(board, 0) == false )
    {
      printf("Solution does not exist");
      return false;
    }

    printSolution(board);
    return true;
}

// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

Java

```
/* Java program to solve N Queen Problem using
   backtracking */
```

```
public class NQueenProblem
{
    final int N = 4;

    /* A utility function to print solution */
    void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                System.out.print(" " + board[i][j]
                                 + " ");
            System.out.println();
        }
    }

    /* A utility function to check if a queen can
       be placed on board[row][col]. Note that this
       function is called when "col" queens are already
       placeed in columns from 0 to col -1. So we need
       to check only left side for attacking queens */
    boolean isSafe(int board[][], int row, int col)
    {
        int i, j;

        /* Check this row on left side */
        for (i = 0; i < col; i++)
            if (board[row][i] == 1)
                return false;

        /* Check upper diagonal on left side */
        for (i=row, j=col; i>=0 && j>=0; i--, j--)
            if (board[i][j] == 1)
                return false;

        /* Check lower diagonal on left side */
        for (i=row, j=col; j>=0 && i<N; i++, j--)
            if (board[i][j] == 1)
                return false;

        return true;
    }

    /* A recursive utility function to solve N
       Queen problem */
    boolean solveNQUtil(int board[][], int col)
    {
        /* base case: If all queens are placed
           then return true */
        if (col >= N)
            return true;

        /* Consider this column and try placing
           this queen in all rows one by one */
        for (int i = 0; i < N; i++)
        {
            /* Check if queen can be placed on
               board[i][col] */
            if (isSafe(board, i, col))
            {
                /* Place this queen in board[i][col] */
                board[i][col] = 1;

                /* recur to place rest of the queens */
                if (solveNQUtil(board, col + 1) == true)
                    return true;

                /* If placing queen in board[i][col]
                   doesn't lead to a solution then
                   remove queen from board[i][col] */
                board[i][col] = 0; // BACKTRACK
            }
        }

        /* If queen can not be place in any row in
           this colum col, then return false */
        return false;
    }

    /* This function solves the N Queen problem using
       Backtracking.  It mainly uses  solveNQUtil() to
       solve the problem. It returns false if queens
       cannot be placed, otherwise return true and
       prints placement of queens in the form of 1s.
       Please note that there may be more than one
       solutions, this function prints one of the
       feasible solutions.*/
    boolean solveNQ()
    {
        int board[][] = {{0, 0, 0, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0}
        };

        if (solveNQUtil(board, 0) == false)
        {
            System.out.print("Solution does not exist");
            return false;
        }

        printSolution(board);
        return true;
    }

    // driver program to test above function
    public static void main(String args[])
    {
        NQueenProblem Queen = new NQueenProblem();
        Queen.solveNQ();
    }
}
// This code is contributed by Abhishek Shankhadhar
```

Output: The 1 values indicate placements of queens

```
 0  0  1  0
 1  0  0  0
 0  0  0  1
 0  1  0  0
```

Sources:

http://see.stanford.edu/materials/icspacs106b/H19-RecBacktrackExamples.pdf

http://en.literateprograms.org/Eight_queens_puzzle_%28C%29

http://en.wikipedia.org/wiki/Eight_queens_puzzle

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

31 Comments Category: Backtracking Tags: Backtracking

Related Posts:

- Fill two instances of all numbers from 1 to n in a specific way
- Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)
- Print all possible paths from top left to bottom right of a mXn matrix
- Tug of War
- Backtracking | Set 7 (Sudoku)
- Backtracking | Set 5 (m Coloring Problem)
- Backtracking | Set 4 (Subset Sum)
- Backtracking | Set 2 (Rat in a Maze)

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Luna Ram
Given solution is wrong if change the value of N
Plz Add in isSafe function
for(i=row,j=col;i>=0&&j<N;i–,j++)
if(board[i][j])
return false;
for(i=row,j=col;i=0;i++,j–)
if(board[i][j])
return false;

HeyM
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
what is wrong with this solution. And this would come before given one. (I am beginner.)

Darshak Mehta
This is right solution

Guest
How about this code:- http://atiqwhiz.blogspot.in/2014/01/the-n-queens-problem-simple-approach.html
#include

```
#define N 8
using namespace std;
bool nQueens(int solve[],int n)
{
int j,r1,r2,c1,c2;
if(n==N)
return true;
for(int i=0;i<N;i++)
{
r2=n;
c2=i;
for(j=0;j<n;j++)
{ r1=j;
c1=solve[j];
if(r1==r2||c1==c2||abs(r1-r2)==abs(c1-c2))
break;
}
if(j==n)
{
solve[n]=i;
if(nQueens(solve,n+1))
return true;
}
}
return false;
}
int main()
{
int solve[N];
if(nQueens(solve,0))
{ cout<<"Row Columnn";
for(int j=0;j<N;j++)
{
cout<<j<<" "<<solve[j]<<endl;
}
}
else
{
cout<<"nnNo such combination is possible";
}
return 0;
}
```

Guest

How about this code:- http://atiqwhiz.blogspot.in/2014/01/the-n-queens-problem-simple-approach.html

Guest

How about this code:- http://atiqwhiz.blogspot.in/2014/01/the-n-queens-problem-simple-approach.html

```
#define N 8
using namespace std;
bool nQueens(int solve[],int n)
{
int j,r1,r2,c1,c2;
if(n==N)
return true;
for(int i=0;i<N;i++)
{
r2=n;
c2=i;
for(j=0;j<n;j++)
{ r1=j;
c1=solve[j];
if(r1==r2||c1==c2||abs(r1-r2)==abs(c1-c2))
break;
}
if(j==n)
{
solve[n]=i;
if(nQueens(solve,n+1))
return true;
}
}
return false;
}
int main()
{
int solve[N];
if(nQueens(solve,0))
{ cout<<"Row Columnn";
for(int j=0;j<N;j++)
{
cout<<j<<" "<<solve[j]<<endl;
}
}
else
{
cout<<"nnNo such combination is possible";
```

}
return 0;
}
Byanjati
for the higher queen problem , we could use 2-Swap Operator for the best Complexity , and it implement the backtracking method too
virat
this will only print only one feasible solution….what about other combinations
Guduru Siva Reddy
public class Nqueens {
public static void nqueens(int k, int n, int[] a) {
for (int i = 1; i <= n; i++) {
if (place(k, i, a) == true) {
a[k] = i;
if (k == n) {
System.out.println();
for (int f = 1; f < a.length; f++) {
System.out.println(a[f]);
}
} else {
nqueens(k + 1, n, a);
}
}
}
}
public static boolean place(int k, int i, int[] a) {// helper function to
// check the cell is
// valid or not
for (int j = 1; j <= k – 1; j++) {
if (a[j] == i)// checking whether in same column
return false;
if (Math.abs(j – k) == Math.abs(a[j] – i))// for checking diagonally
return false;
}
return true;
}
public static void main(String[] args) {
int[] a = new int[4 + 1];// We can replace 4 with n, and 5 with n+1 for
// nqueens
nqueens(1, 4, a);// array a is for solution space
}
}
Guduru Siva Reddy
This solution prints all possible solutions.
hh
What is complexity of your soln?
Nikunj Bhartia

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *,int*);
int checklist(int *,int );
void display(int *,int );
int count=0,ans=1;

permute(int list[],int i,int n)
{   int j;
    if(i==n){
        if(checklist(list,n))
          {printf("\n soltion no. %d\n\n",count);display(list,n);}
    }
    for(j=i ; j<=n ; j++){
        swap(&list[i],&list[j]);
        permute(list,i+1,n);
        swap(&list[i],&list[j]);
      }
}

inline void swap(int *a,int*b){
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}

int checklist(int list[],int n){
    int j,k,rd,ld;
    for(j=n ; j>=0 ; j--){
        rd=list[j]+1;
        ld=list[j]-1;
      for(k=j-1 ; k>=0 ; k--){
            if(list[k]==list[j] || list[k]==rd || list[k]==ld)
              return 0;
            else{
              rd++;
              ld--;
            }
        }
    }
    count++;
    return 1;
}

void display(int list[],int n){
    int i,j;
    for(i=0;i<=n;i++){
        for(j=0;j<=n;j++){
            if(list[j]==i) printf("%3d",i+1);
            else printf("  .");
        }
        printf("\n\n");
    }
   if(ans==1){
    printf("\n\n  Display next solution ? 1 : 0  <::>  Display all soltions ? 2 : 0  ---> ");
    scanf("%d",&ans);
    }
    else if(ans==0) exit(0);
   printf("\n\n\n");
}

void main(){
    int n,*list,i;
  while(n){
    printf("\n \n Enter the value of n : ");
    scanf("%d",&n);
    printf("\n");
    list=(int *)malloc(n*sizeof(int));
    for(i=0 ; i<n ; i++)
      list[i]=i;
    permute(list,0,n-1);
    printf(" Total number of possible solutions (displayed above) = %d\n\n",count);
    count=0;
  }
}
```

Kavish Dwivedi
Here is my solution for 16 queen problem

```
#include&lt;stdio.h&gt;
#define N 16
int sol[N][N];
int check(int row,int col)
{
        int i,j;
        for(i=0;i&lt;col;i++)
                if(sol[row][i]==1)
                {
                        //printf(&quot;False
&quot;);
                        return 0;
                }
        for(i=row,j=col ; i&gt;= 0 &amp;&amp; j&gt;=0 ; i--,j--)
        {
                if(sol[i][j]==1)
                        return 0;
        }
        for(i=row,j=col;i&lt;N&amp;&amp;j&gt;=0;i++,j--)
        {
                if(sol[i][j]==1)
                        return 0;
        }
        return 1;
}
int solve(int col)
{
        int i;
        if(col==N)
        return 1;
        else
        {
                for(i=0;i&lt;N;i++)
                {
                        if(check(i,col)==1)
                        {
                                sol[i][col]=1;
                                if(solve(col+1)==1)
                                {
                                        //printf(&quot;i=%d
&quot;,i);
                                        return 1;
                                }
                                else
                                        sol[i][col]=0;
                        }
                }
        }
        return 0;
}
int main()
{
        int i,j;
        for(i=0;i&lt;N;i++)
        for(j=0;j&lt;N;j++)
        sol[i][j]=0;
        if(solve(0)==1)
        {
                for(i=0;i&lt;N;i++)
                {
                        for(j=0;j&lt;N;j++)
                        printf(&quot;%d&quot;,sol[i][j]);
                        printf(&quot;
&quot;);
                }
        }
        else
        {
                printf(&quot;Not possible
&quot;);
        }
        return 0;
}
```

Kavish Dwivedi
Sorry , some typo error came unnoticed.

```
#include<stdio.h>
#define N 8
int sol[N][N];
int check(int row,int col)
{
int i,j;
for(i=0;i<col;i++)
if(sol[row][i]==1)
{
//printf("False\n");
return 0;
}
for(i=row,j=col ; i>= 0 && j>=0 ; i–,j–)
{
if(sol[i][j]==1)
return 0;
}
for(i=row,j=col;i<N&&j>=0;i++,j–)
{
if(sol[i][j]==1)
return 0;
}
return 1;
}
int solve(int col)
{
int i;
if(col==N)
return 1;
else
{
for(i=0;i<N;i++)
{
if(check(i,col)==1)
{
sol[i][col]=1;
if(solve(col+1)==1)
{
//printf("i=%d\n",i);
return 1;
}
else
sol[i][col]=0;
}
}
}
return 0;
}
int main()
{
int i,j;
for(i=0;i<N;i++)
for(j=0;j<N;j++)
sol[i][j]=0;
if(solve(0)==1)
{
for(i=0;i<N;i++)
{
for(j=0;j<N;j++)
printf("%d",sol[i][j]);
printf("\n");
}
}
else
{
printf("Not possible\n");
}
return 0;
}
```

Zeenat Islam
this solution is getting hanged when N is 16… what changes to make to get this code to work for large values of N??
GeeksforGeeks
Could you please post the code that you tried?
Ajinkya
What is the time complexity of this approach? and of backtracking in general… someone pointed out that it is exponential… cna someone work out the time complexity in detail?
Thanks

```
/* Paste your code here (You may delete these lines if not writing code) */
```

sk007
Here is another solution for 8×8 board using the backtracking principle:

```
int column[8];

void NQueen(int row){
if(row==8){
printBoard();
return;
}
for(i=0;i<8;i++){
   column[row]=i;
   if(check(row))
       NQueen(row+1);
}

}


int check(int row){
//There will be attacking queens if columns are same for two rows or difference in columns are same as the rows

for(i=0;i<row;i++){
    if((column[i]==column[j])||(column[i] - column[j])== abs(i-j))
        return 0;

}

return 1;

}

}
```

Anand
http://anandtechblog.blogspot.com/
Anand
http://anandtechblog.blogspot.com/2011/08/8-queen-problem.html
Doom
heres the code to solve sudoku using same technique
http://ideone.com/vQ7Ej
Nitish Garg
What update will be required to print all the possible ways to place N queens on an N X N chessboard, like for 8 queens, we have 92 different solutions?
kartik
See the following modified code.

```
 #define N 4
#include<stdio.h>

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            printf(" %d ", board[i][j]);
        printf("\n");
    }
    printf("\n");
}

/* A utility function to check if a queen can be placed on board[row][col]
   Note that this function is called when "col" queens are already placeed
   in columns from 0 to col -1. So we need to check only left side for
   attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
    {
        if (board[row][i])
            return false;
    }

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
    {
        if (board[i][j])
            return false;
    }

    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
    {
        if (board[i][j])
            return false;
    }

    return true;
}

/* A recursive utility function to solve N Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed then return true */
    if (col >= N)
    {
        printSolution(board);
        return true;
    }


    bool res = false;

    /* Consider this column and try placing this queen in all rows
       one by one */
    for (int i = 0; i < N; i++)
    {
        /* Check if queen can be placed on board[i][col] */
        if ( isSafe(board, i, col) )
        {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            if ( solveNQUtil(board, col + 1) == true )
            {
                res = true;
            }

            /* If placing queen in board[i][col] doesn't lead to a solution
               then remove queen from board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }

     /* If queen can not be place in any row in this colum col
        then return false */
    return res;
}

/* This function solves the Maze problem using Backtracking.  It mainly uses
solveNQUtil() to solve the problem. It returns false if queens cannot be placed,
otherwise return true and prints placement of queens in the form of 1s. Please
note that there may be more than one solutions, this function prints one of the
feasible solutions.*/
bool solveNQ()
{
    int board[N][N] = { {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    };

    if ( solveNQUtil(board, 0) == false )
    {
      printf("Solution does not exist");
      return false;
    }

    return true;
}

// driver program to test above function
int main()
{
    solveNQ();

    getchar();
    return 0;
}
```

reema
@GeeksForGeeks Please Don’t Forgot to Analyze and mention Time,Space Complexity
kartik
@rahul: Time complexity is exponential in worst case. Same is the case with all other Backtracking algos like Rat in a Mzae, Knight Tour, Subseet Sum.. etc
hh
how come exponential for this ..I thought it should be O(nxn)