Implementation and Analysis of N-queens Problem

Tarun Kumar
UE143103
Computer Science Engineering
University Institute of Engineering and Technology
Panjab University
Chandigarh, 160012
Email: tarunk.1996@gmail.com

***Abstract*—This practical illustrates about the implementation and possible solution of a trivial problem i.e. N-Queens problem.**

I. Introduction

The **N-Queen Problem** states that how to place N chess queens on a $N * N$ chess board such that no two queens can attack each other. Keeping in mind that the movement of chess queen can be either **horizontally, vertically or diagonally**. Which means no two queens can be on the same row, same column or at theirs diagonals. The solution of the N-Queen problem is obtained using backtracking the results if the scenario is not feasible until all the possible solutions of the problem are not found.

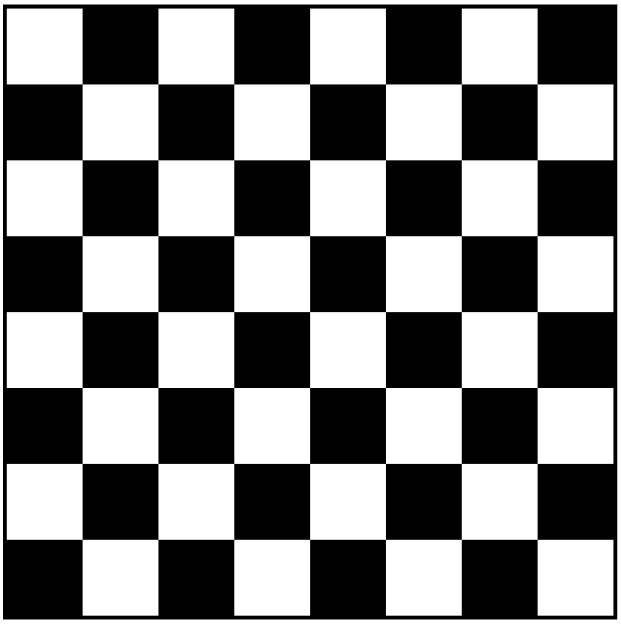

Fig. 1. An sample chess board of 8*8 boxes

The above figure 1 shows us a sample chess board of 8*8 boxes which is generally used in chess games.

II. Algorithms and Implementation

A. Single Source Shortest Path Algorithm

```
int findcol(int a[],int num,int k)
{
   int i;
    for(i=0;i<k;i++)
    {
        if(num==a[i])
            return 1;
    }
    return 0;
}
int finddia(int a[],int num,int k)
{
    int i,j;
    for(i=k-1,j=1;i>=0;i--,j++)
    {
        if(num==a[i]+j||num==a[i]-j)
        {
            return 1;
        }

    }
    return 0;
}
int rec(int p[],int n,int r)
{
   int z,k,m;
   if(r==n)
   {
      return 1;
   }
   for(int i=0;i<n;i++)
   {
      if(findcol(p,i,r)||finddia(p,i,r))
         continue;
      p[r]=i;
      z=rec(p,n,r+1);
      p[r+1]=0;
      if(z)
      {
         int a[100][100];
         for(k=0;k<n;k++)
         {
            for(m=0;m<n;m++)
               a[k][m]=0;
         }
         for(k=0;k<n;k++)
         {
            a[k][p[k]]=1;
         }
         for(k=0;k<n;k++)
         {
```

```
            for(m=0;m<n;m++)
               cout<<a[k][m];
            cout<<endl;
         }
         ct+=1;
         cout<<endl;
      }
   }
    return 0;
}
```

III. Experiment

In this program, algorithm shown above is used for solving the N-Queen problem. In the algorithm 3 variables are passed as arguments i.e. an array, its size (or value of n) and current row which is under consideration. If the current row exceeds from the value of n it will return 1 back to the function else it will go into a loop from i=0 to n. Inside the loop two functions were called findcol() and finddia() which verifies that the current queen must not be placed at same column or at diagonal to the previously inserted queen. If any of the function comes out to be true, the continue statement will be executed which increments the value of i and check for the conditions again until a correct position of the queen is not found or the whole row is completely exhausted (in this step it will simply backtrack one step back and check for other combinations). If a feasible position of the queen is found it will assign the value of i to the array[current row] and recursively call itself with same arguments and incremented value of r(current row) and assign the returned value to a variable z. If the value of z is 1 it means all the n queens are perfectly placed in the chess board and the combination is one of the feasible solution to the n-queen problem. This will print out the position of the queen in the n*n chess board where 1 denotes the position of a queen and 0 means blank box. This process is repeated until all the feasible cases are not checked.

A. Through Graphics

Same procedure has been followed as listed above. The N-Queen Problem using graphics is designed on Java script only. Using canvas feature of Javascript all the necessary objects are created using it. For example coloring of boxes with different color.

*1) Creating a Chess Board of size N*N:*

```
var d=700.0/n;
var can = document.getElementById("canvas1");
var ctx = can.getContext('2d');
for(i=0;i<n;i++)
{
   for(j=0;j<n;j++)
   {
      var ch=parseInt(i+j);
      if(ch%2==0)
      {
         ctx.fillStyle="Black";
      }
      else
      {
         ctx.fillStyle="White";
      }
      ctx.fillRect((j)*d,(i)*d,d,d);
}
```

This code will simply generate a Chess board of size N*N. The overall size of board is fixed i.e. of 700px. $fillStyle$ function accepts the color from user and using $fillRect()$ function it fills the color into it. $fillRect()$ function is accepting 4 arguments i.e. argument 1 and 2 are the x and y coordinates in the canvas where the other two arguments are height and width of the rectangle.

2) Placing N Queens in the Chess Board:

```
var d=700.0/n;
var can = document.getElementById("canvas1");
var ctx = can.getContext('2d');
for(i=0;i<n;i++)
{
   for(j=0;j<n;j++)
   {
      if(p[i]==j)
      {
         ctx.fillStyle="Red";
         ctx.fillRect((j)*d,(i)*d,d,d);
         var c=document.getElementById("canvas1");
         var cx=c.getContext("2d");
         var img=document.getElementById("q");
         cx.drawImage(img,(j)*d,(i)*d,d,d);
       }
}
```

In this section, a feasible solution to the problem is being generated. The position of the queen in the chess board is obtained from the algorithm and while placing a queen in the chess board, the color of that block is highlighted with red color and using $drawImage()$ function the an image of Queen is being inserted on the same highlighted block. $drawImage()$ function takes 5 arguments, first one is the location of the image, the other two are x and y coordinates in the canvas and remaining two are height and width of the image.

IV. Graphs and Tables

Following are results obtained after executing the program:

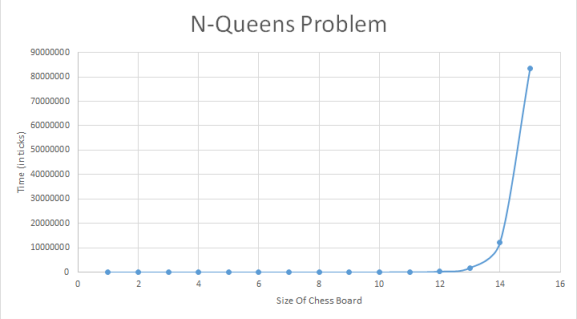

Fig. 2. Graph obtained from results of Table 1

TABLE I
ALL PAIR SHORTEST PATH RESULT

Size Of Board	Time required (in ticks)	Total number of solutions possible
1	1	1
2	1	0
3	1	0
4	3	2
5	8	10
6	30	4
7	116	40
8	538	92
9	2538	352
10	12643	724
11	57865	2680
12	307633	14200
13	1810165	73712
14	12215604	365596
15	83553740	2279184

V. SAMPLE OUTPUTS

```
Enter the value of N: 6
010000
000100
000001
100000
001000
000010

001000
000001
010000
000010
100000
000100

000100
100000
000010
010000
000001
001000

000010
001000
100000
000001
000100
010000

4 Results are Possible
```

Fig. 3. Sample Output of the N-Queen Problem for N=6 without graphics

VI. CONCLUSION

- No solution is feasible for n=2 and 3.
- The growth of the feasible solutions and time to compute the solution increases exponentially.

VII. ATTACHMENTS

The Program code used for solving N-Queen Problem (using graphics) is here : http://pastebin.com/H1B2H4ch