**Topic: Backtracking**

**Theory:** In many applications of the backtrack method, the desired solution is expressible as an n-tuple *(x1,...,Xn),* where the x*i* are chosen from some finite set Si. Often the problem to be solved calls for finding one vector that maximizes (or minimizes or satisfies) a *criterion function P(x1,…..* . , *xn). Sometime*s it seeks all vectors that satisfy *P.* For example, sorting the array of integers in. *a[1* : n] is a problem whose solution is expressible by an *n- tuple, w*here x*i* is the index in *a* of the ith smallest element. The criterion function P is the inequality *a[xi]* ≤ *a[xi+1]* for 1 ≤ i < *n.* The set *Si* is finite and includes the integers 1 through *n.* Though sorting is not usually one of the problems solved by backtracking, it is one example of a familiar problem whose solution can be formulated as an n-tuple.

**Control abstraction**:

void Backtrack( int k )

// This is a schema that describes the backtracking process //using recursion. On entering, the first k-1 values x[1], x[2], //…., x[k-1] of the solution vector x[1:n] have been //assigned. x[] and n are global.

{

for (each x[k] such that x[k] Є T(x[1], …, x[k-1])

{

if (Bk (x[1], x[2], …, x[k]))

{

if (x[1], x[2], …, x[k] is a path to an answer node)

output x[1:k];

if (k < n) Backtrack(k+1);

}

}

}

**Batch: B1 Roll No.: 1611077**

**Experiment No. \_\_\_8\_\_\_**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Backtracking Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for 8-Queens problem

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. [**http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**](http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf)
5. [**http://www.slideshare.net/Tech\_MX/8-queens-problem-using-back-tracking**](http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking)
6. [**http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving,8- Queens problem and its applications.

**Algorithm N Queens Problem:-**

void NQueens(int k, int n)

// Using backtracking, this procedure prints all possible placements of n queens on an n X n chessboard so that they are nonattacking.

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

{

if (Place(k, i))

{

x[k] = i;

if (k==n)

for (int j=1;j<=n;j++) Print x[j] ;

else NQueens(k+1, n);

}

}

}

Boolean Place(int k, int i)

// Returns true if a queen can be placed in kth row and ith column. Otherwise it returns false.

// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of r.

{

for (int j=1; j < k; j++)

if ((x[j] == i) // Two in the same column

|| (abs(x[j]-i) == abs(j-k))) // or in the same diagonal

return(false);

return(true);

}

**Example 8-Queens Problem:**

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.

**Solution Using Backtracking Approach:**

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.

**Algorithm:**

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 unmark this [row,

column] (Backtrack) and go to step (a) to try other rows.

4) If all rows have been tried and nothing worked, return false to trigger

backtracking.

**Analysis of Backtracking solution for 8-Queens Problem:**

Space complexity

For this algorithm it is O(N^2). The algorithm uses an auxiliary 2-D array of N rows and N columns to store just positions.

Time complexity

If we add all the loops and recursive calls up and define the run time as T(N), then:

**T(N) = O(N^2) + N\*T(N-1)**

If we draw a recursion tree using this recurrence, the final term will be something like n^3+ n!. By the definition of Big O, this can be reduced to O(n!) running time.

Therefore, Time complexity of 8-Queens Problem is **O(n!).**

**IMPLEMENTATION:**

import java.util.\*;

class Queens

{

static int count=0;

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the no. of queens to be placed: ");

int n=sc.nextInt();

int a[][]=new int[n][n];

int b[][]=new int[n][n];

int i=0,j=-1;

Queens obj=new Queens();

obj.place(i, j, n, a);

System.out.println("\n\nTotal no. of solutions for " + n + " queens is "+count);

System.out.println("\n\nProgram by Mihir Gandhi B1 1611077");

}

public void place(int i, int j, int n, int a[][])

{

int flag=0,flag2=0,final\_flag=0;

int p=0;

int k,l,m;

int u,v;

int x,y;

for(u=0;u<n;u++) // u is the column number

{

p=0;

flag=0;

flag2=0;

final\_flag=0;

for(v=0;v<n;v++)

{

if(a[v][u]!=0)

{

flag2=1;

}

}

if(flag2==0&&u>j)

{

p=u;

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

{

k=i+m;

l=p+m;

if(k>=0&&k<n&&l>=0&&l<n)

{

if(a[k][l]!=0)

{

flag=1;

break;

}

}

k=i-m;

l=p+m;

if(k>=0&&k<n&&l>=0&&l<n)

{

if(a[k][l]!=0)

{

flag=1;

break;

}

}

k=i+m;

l=p-m;

if(k>=0&&k<n&&l>=0&&l<n)

{

if(a[k][l]!=0)

{

flag=1;

break;

}

}

k=i-m;

l=p-m;

if(k>=0&&k<n&&l>=0&&l<n)

{

if(a[k][l]!=0)

{

flag=1;

break;

}

}

}

}

else

{

continue;

}

if(flag==0)

{

a[i][p]=1;

i++;

if(i==n)

{

count++;

System.out.print("\nSolution No. "+ count+"\n");

for(x=0;x<n;x++)

{

for(y=0;y<n;y++)

{

System.out.print(a[x][y]+" ");

}

System.out.print("\n");

}

}

final\_flag=1;

break;

}

}

if(final\_flag==1)

{

j=-1;

place(i, j, n, a);

}

else

{

if(i>0)

{

int s=0;

i--;

while(a[i][s]!=1)

{

s++;

}

a[i][s]=0;

j=s;

place(i, j, n, a);

}

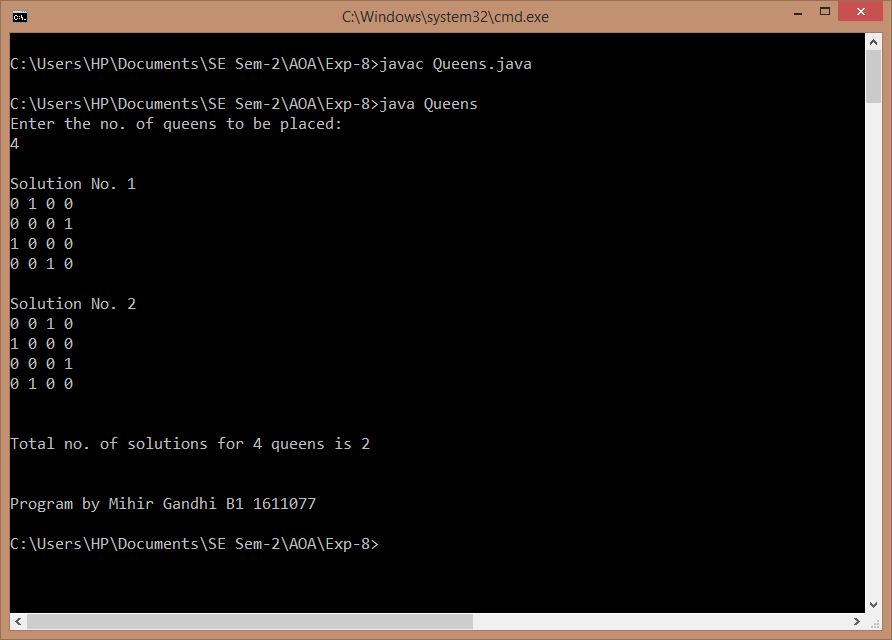
}

}

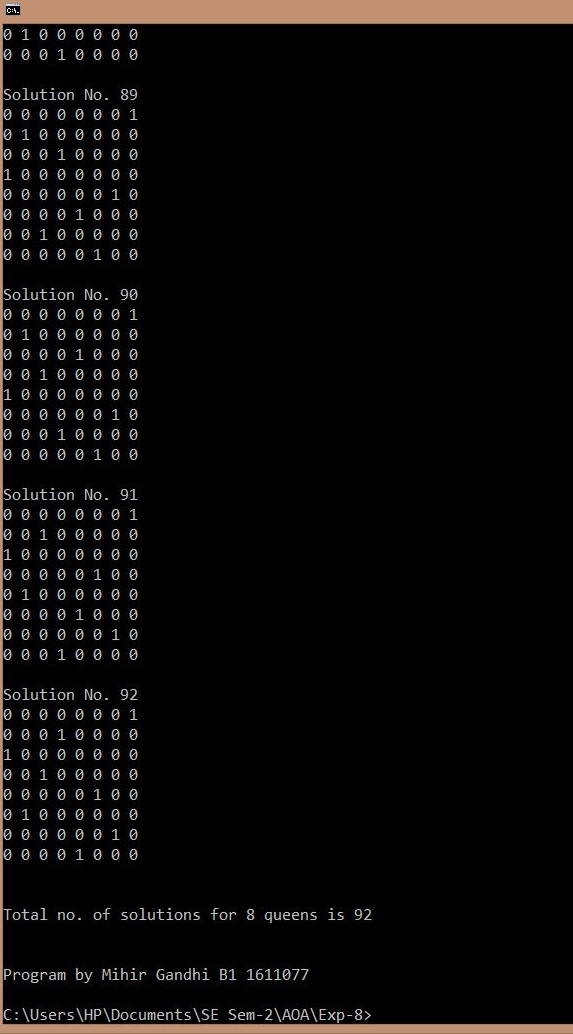
}

**OUTPUT:**

4 Queens

****

8 Queens

****

**CONCLUSION:**

Thus, we have learnt the Backtracking strategy of problem solving for 8-Queens problem and implemented the same successfully. The actual outcome matched with the expected outcome.