# Experiment No: 09

**Aim: To implement N queen problems using backtracking**

**Theory:**

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;

The **eight queens puzzle** is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general ***n*-queens problem** of placing *n* queens on an *n*×*n*chessboard, where solutions exist for all natural numbers *n* with the exception of *n*=2 and*n*=3

A way to place all n queens on the board such that no queens are attacking another queen. Using backtracking, this algorithm prints all possible placements of n queens on an n*n chessboard so that they are non-attacking. Place (k, i) – Returns true if a queen can be placed in $k^{th}$ row and $i^{th}$ column. Otherwise it returns false. X [] is a Global array whose first (k –1) values have been set. Abs(r) returnsthe absolute value of r.

**Algorithm:**

```
Algorithm NQueens( k, n) //Prints all Solution to the n-queens problem
{
    For i := 1 to n do
    {   if Place (k, i) then
              {      x[k] := i;
                       if ( k = n) then write ( x [1 : n] else NQueens(k+1,n);
               }
         }
}
Algorithm Place (k,i)
{
for j := 1 to k-1 do
if (( x[ j ] = // in the same column
or (Abs( x [ j ] - i) =Abs ( j – k ))) // or in the same
diagonalthen return false;
return true;
}
```

**Conclusion:**

- Backtracking provides the hope to solve some problem instances of nontrivial sizes bypruning non-promising branches of the state-spacetree.
- The success of backtracking varies from problem to problem and from instance toinstance.
- Backtracking possibly generates all possible candidates in an exponentially growingstate-spacetree.

# Experiment No: 10

**Aim:** Implement Rabin Karp string matching algorithm

**Theory:**

Rabin Karp string matching algorithm makes use of elementary number-theoretic notion of modulo arithmetic.
Given a pattern P[1..m], let p denote its corresponding decimal value. Similarly, given a text T[1..n],let ts denote the decimal value of the length m substring T[s+1..s+m], for s = 0,1,..,n-m.
Certaintly, ts = p if and only if T[s+1].s+m] = P[1..m]; thus, is a valid shift if and only if ts=p.

**Algorithm**

RABIN-KARP-MATCHER(*T, P, d, q*)

```
1   n   length[T]
2   m   length[P] 3h        d^m -1 mod q
3   p   0
4   t0  0
5   for i   1 to m
6       do p   (dp+ P[i]) mod q
7       t0     (dt0 + T[i]) mod q
8   for s   0 to n -m
9       do if p =ts
10      then if P[1 . . m] = T[s + 1 . . s +m]
11      then "Pattern occurs with shift"s
12      if s <n -m
13      then ts+1  (d(ts- T[s + 1]h) + T[s + m + 1]) modq
```

Running time: Preprocessing time: O(m), matching time: worst case is O((n-m+1)m), while averagewill be lesser.

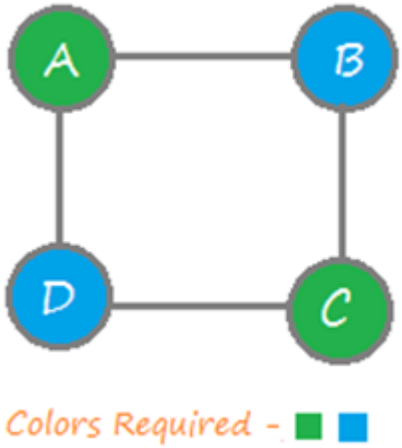
**Conclusion:**
The Rabin karp works on arithmetic operations to match pattern in text and is much faster than Naïvestring matching algorithm.

# Experiment No: 11

**Aim:** Implement Graph Coloring Algorithm
What is Graph Coloring Problem?

**Theory**

For a given graph if it is asked to color all vertices with the 'M' number of given colors, insuch a way that no two adjacent vertices should have the same color.




It it is possible to color all the vertices with the given colors then we have to output the colored result, otherwise output 'no solution possible'.

*The least possible value of 'm' required to color the graph successfully is known asthe* ***chromatic number of the given graph****.*

It it is possible to color all the vertices with the given colors then we have to output the colored result, otherwise output 'no solution possible'.

*The least possible value of 'm' required to color the graph successfully is known asthe* ***chromatic number of the given graph****.*

**Graph Coloring Solution**
**Using Naive Algorithm**

In this approach using the brute force method, we find all permutations of color combinations that can color the graph.

If any of the permutations is valid for the given graph and colors, we output the resultotherwise not. This method is not efficient in terms of time complexity because it finds all colors combinations rather than a single solution.

**Using Backtracking Algorithm**

The backtracking algorithm makes the process efficient by avoiding many bad decisions made in naïve approaches.

In this approach, we color a single vertex and then move to its adjacent (connected) vertex to color it with different color.

After coloring, we again move to another adjacent vertex that is uncolored and repeat the process until all vertices of the given graph are colored.

In case, we find a vertex that has all adjacent vertices colored and no color is left to make it color different, we backtrack and change the color of the last colored vertices and again proceed further.

If by backtracking, we come back to the same vertex from where we started and all colors were tried on it, then it means the given number of colors (i.e. 'm') is insufficient to color the given graph and we require more colors (i.e. a bigger chromatic number).

**Steps To color graph using the Backtracking Algorithm:**

1. Different colors:
   A. Confirm whether it is valid to color the current vertex with the current color (bychecking whether any of its adjacent vertices are colored with the same color).
   B. If yes then color it and otherwise try a different color.
   C. Check if all vertices are colored or not.
   D. If not then move to the next adjacent uncolored vertex.
2. If no other color is available then backtrack (i.e. un-color last colored vertex). *Here **backtracking means to stop further recursive calls** on adjacent vertices by returning false. In this algorithm Step-1.2 (Continue) and Step-2 (backtracking) iscausing the program to try different color option.*
   ***Continue** – try a different color for current vertex.*
   ***Backtrack** – try a different color for last colored vertex.*

**Analysis**:

- **Time Complexity:** O(m^V).
  There are total O(m^V) combination of colors. So time complexity is O(m^V). The upperboundtime complexity remains the same but the average time taken will be less.
- **Space Complexity:** O(V).
  Recursive Stack of graphColoring(…) function will require O(V) space.

**Conclusion:**
Understood the application of Graph Coloring Problem in many practical areas such as pattern matching, designing seating plans, scheduling exam time table, solving sudoku puzzles etc.