Unit – 5: Backtracking

For detail discussion, students are advised to refer the class discussion

Basic of Backtracking

- In this method solution of problem with “n” inputs belonging to set “Si” is expressed in terms of tuple (x1….xn), where xi is selected from Si. The validity of each tuple is decided on the basis of criterion function.
- Suppose there are “n” inputs in “Si” then there can be “n” possible tuples. The broad approach is to form “n” tuples and evaluate each of these tuple against “P” and save those tuples which are optimum. Backtracking allows similar operations to be carried out in less than “n” trails.

Basics

- **Basic Principle:** In this method instead of building the solution as single entity, backtracking method will build the solution component wise and check it against the partial criterion function.
- For example, if there are “n” elements then first component can be (x1..xi) is checked against (p1..pi) and if partial solution and partial criterion function are not matching then remaining part of solution is simply ignored.

Basics

- Problem solving using backtracking requires that all the solutions must satisfy the complex set of constraints. These constraints can be classified into two classes:
- ***Explicit Constraints***
- ***Implicit Constraints***
- **Explicit Constraints:**
- They are set of rules which allows “xi” to take value only from the given set.
- For example: xi >= 0 or xi = 0 or 1

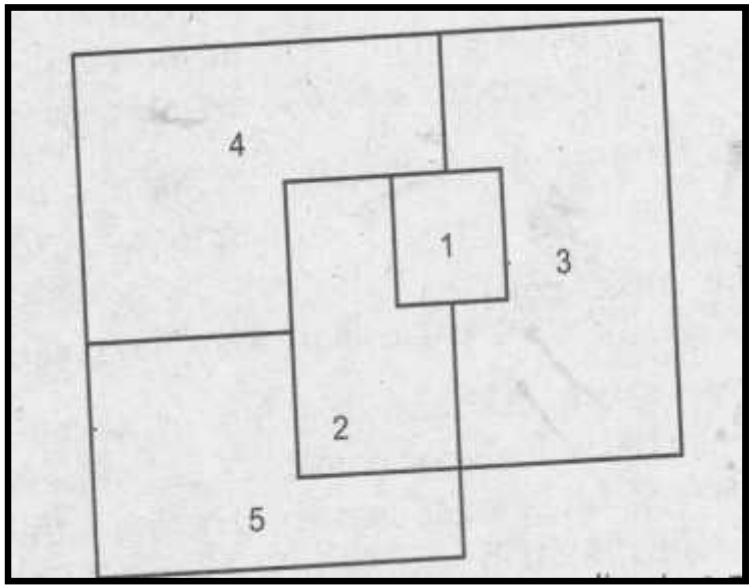
Basics

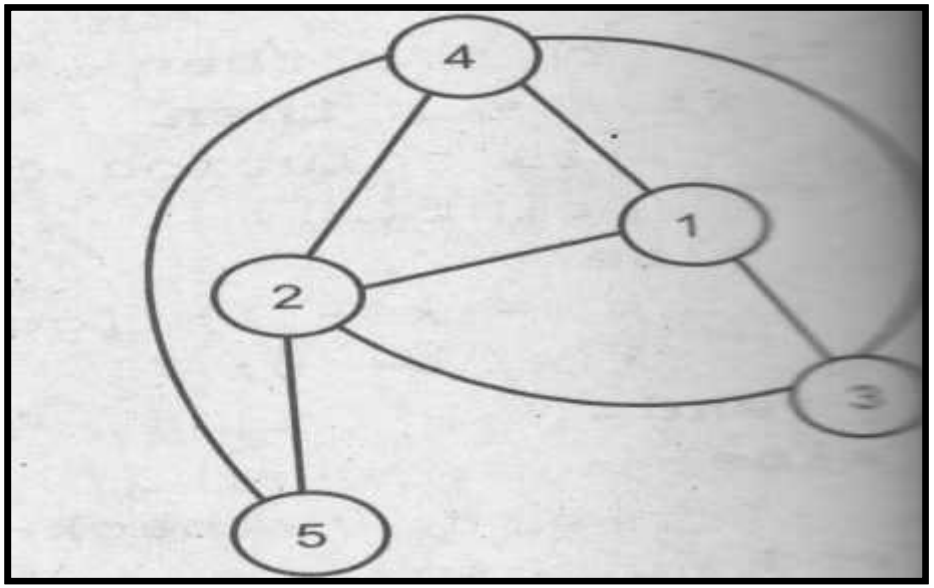
- **Implicit Constraints:**
- These constraints are the rules that decides which of the tuple in the solution space of “i” satisfies criterion function

Graph Coloring: Algorithm

- Algorithm solution for problem solved using BACKTRACKING are RECURSIVE
- The input to algorithm is vertex number present in the graph
- The algorithm generates the color number assigned to vertex and stores it an array.
- If the constraint are not matched at any point, then remaining part of algorithm is not executed and new cycle is initiated.
- The algorithm terminates with a solution satisfying the constraints.

Chromatic Number



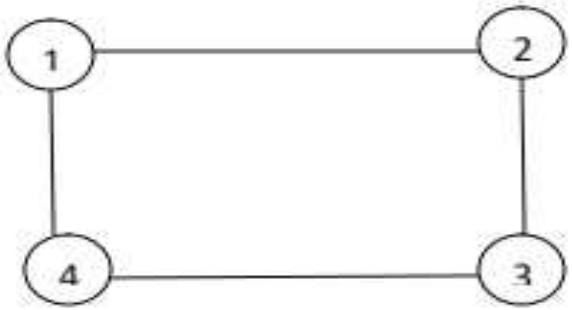


Part – I: mcolor

- This component passes the different values of vertex to “nextvalue” algorithm, and accept the value returned by “nextvalue” algorithm.
- The value is checked and if found suitable then stored in output array.
- The output array x[1..n] is solution for the given problem.

ALGORITHM FOR GRAPH COLORING

```
Algorithm mcolor(k)
{
    The graph is represented in the form of matrix nxn
    "k" is an index of vertex to be colored
    {
        repeat
        {
                Nextvalue(k)
                If (x[k] = 0) then return
                If (k=n) then
                        Write(x[1:n])
                Else
                        Mcolor(k+1)
        } until(false)
    }
}
```



Algorithm: nextvalue

- This algorithm will accept the vertex number from “mcolor” and generates color value for the vertex, with constraint satisfaction.
- To check constraints following conditions are tested:
- Let vertex “k” is in process, then all vertices in the graph are tested for adjacency test.
- If the vertices are adjacent, then, should be different colors
- The color information is stored in array “x”
- Hence G[k,j] != 0 and x[k] = x[j] then color is not suitable otherwise suitable.

```
Algorithm nextvalue(k)
Assume x[1..k-1] are assigned integer in the range of [1,m] such that no
two adjacent vertices are in the same color
x[k] is assigned the next value such that distinctness is maintained
If no such color exists then x[k]=0
{
    repeat
    {
        x[k] = (x[k] + 1) mod m+1
        if x[k] = 0 then return
        for j = 1 to n do
        {
            If (G[k,j]!=0) and (x[k]=x[j]) then
                Break
        }
        If (j=n+1) then
            Return
    }until (false)
}
```

Hamiltonian Cycle

- It is a cycle generated on graph, with same vertex as source and destination.
- The constraints for generating cycle:
  - Each vertex has to be visited once
  - Any edge can be used only once
  - Some of the edges can be skipped, but vertices cannot be skipped.
  - Applications: Travelling salesman problem
  - One of the approach to solve travelling salesman problem algorithmically

Hamiltonian Cycle

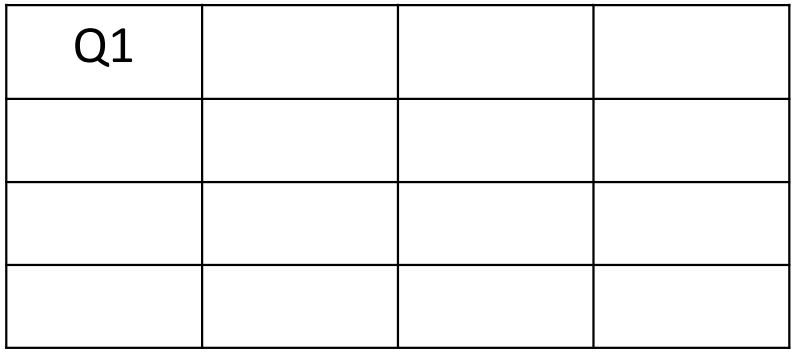
- Example: Refer class discussion and notes.

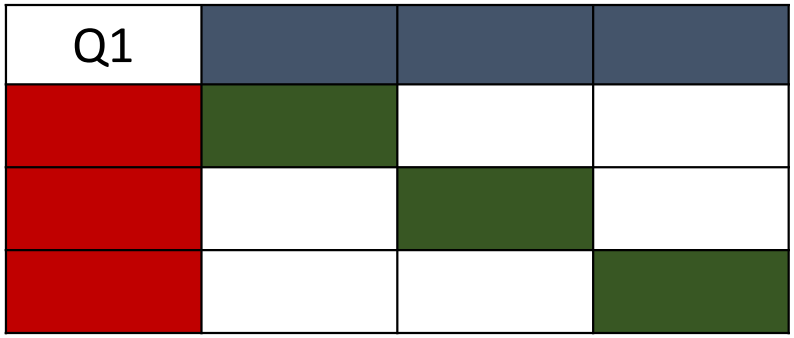
N-Queen Problem

- Basic: Given a chess board of size nxn, it is required to place “n” queen on the chess board with following constraints:
- 1. No two queens should be in same row
- 2. No two queens should be in same column
- 3. No two queens should be diagonally opposite

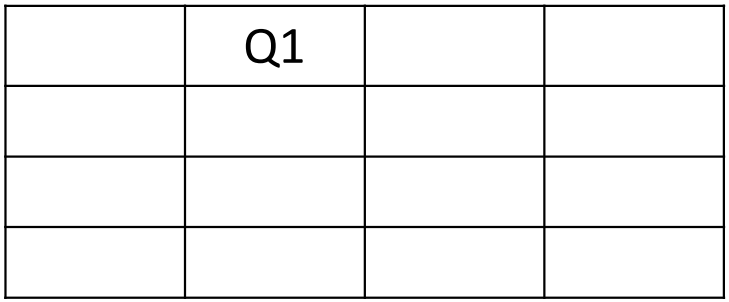
Why backtracking: if any of above constraint does not satisfies at “k” queen, then it is required to adjust 1..k-1 queens.

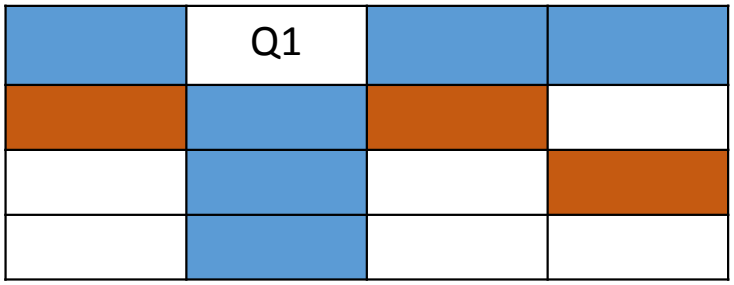
Sub-Problem: 4 Queen problem

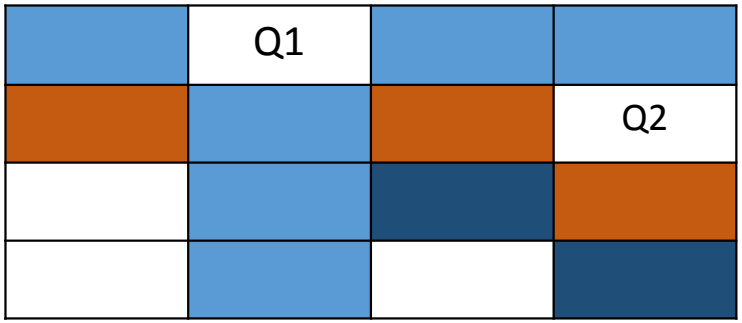


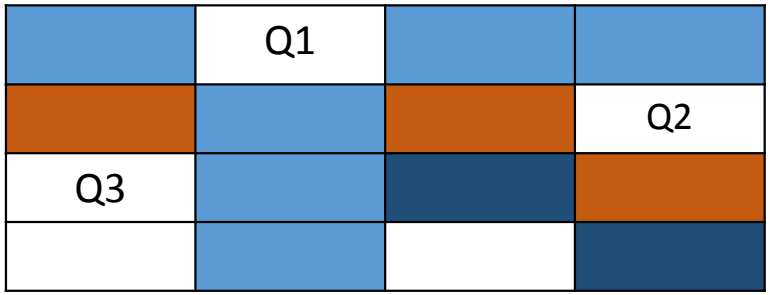


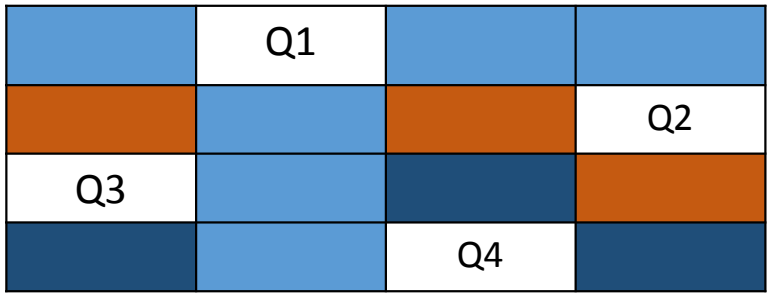
Sub-Problem: 4 Queen problem



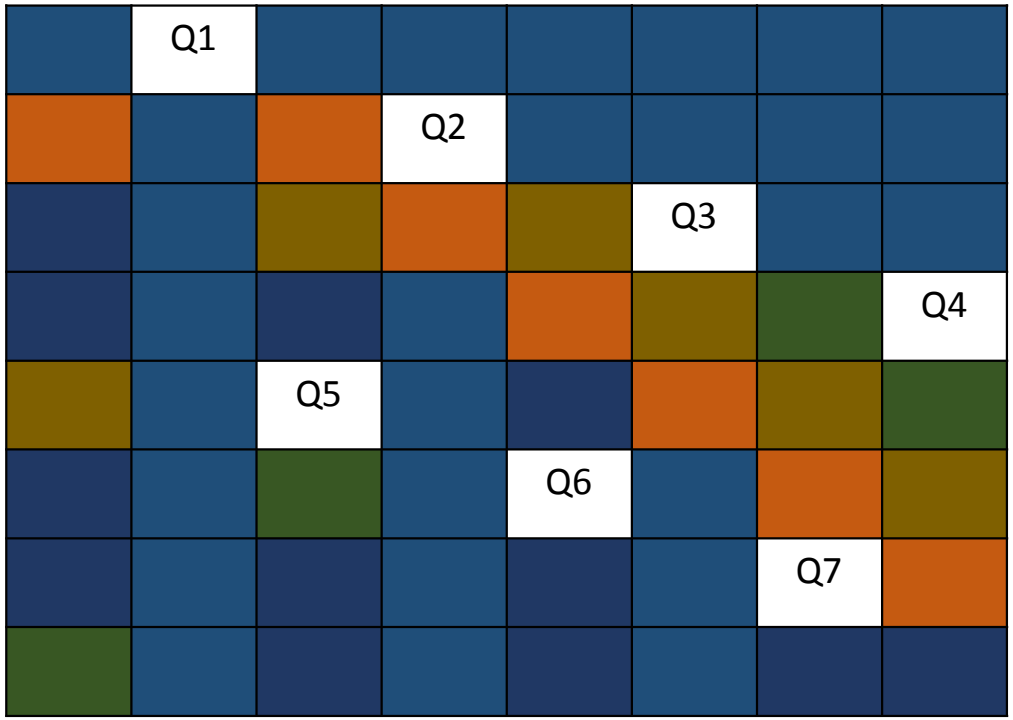




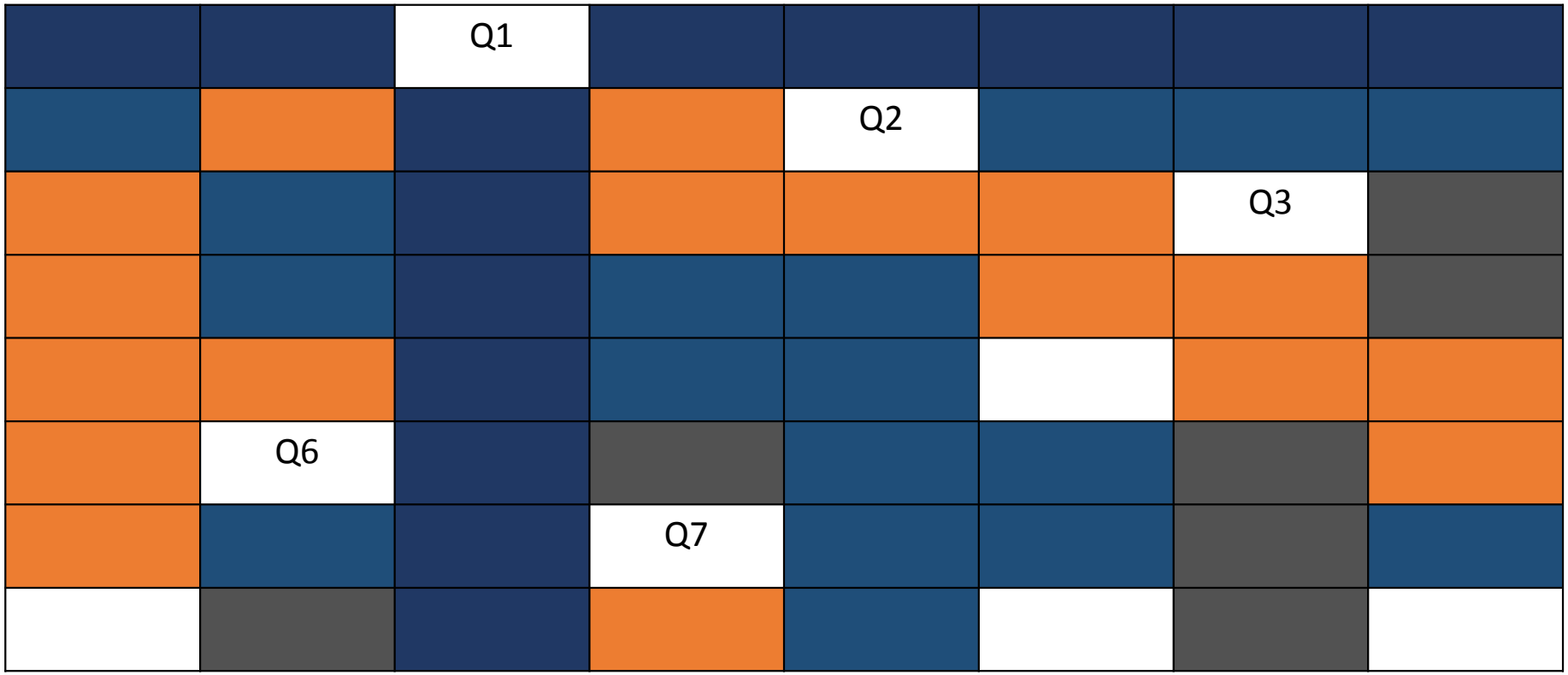




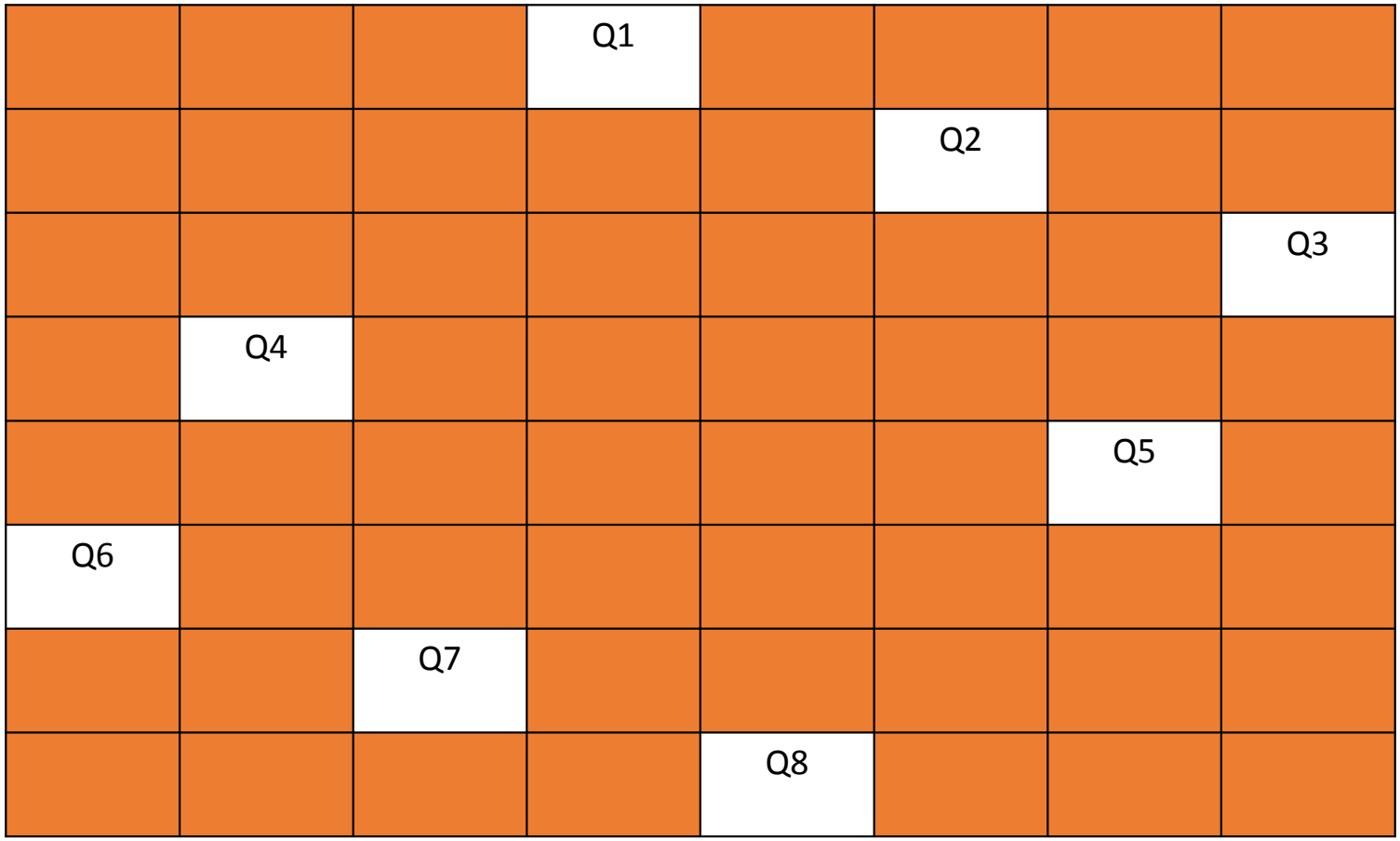
Solution to 8 Queen Problem



Second Attempt



Solution to 8 Queen Problem



Solution to 8 Queen Problem: Constraint logic

	1	2	3	4	5	6	7	8
1								
2								
3								
4					*			
5								
6								
7								
8								

Algorithm: Part - I

- The algorithm will generate array x[1..n] to store the column address of the queens and index of array “x” represents row address.
- For example if at location x[3]=2, then one of the location in matrix is [2,3] for storing third queen.
- The algorithm is divided into two parts:
  - Part 1: to pass the values of Q’s
  - Part 2: to generate the locations.

Algorithm: Part - I

```
Algorithm Nqueen(k,n)
{
   //Initial value of k=n and n=8
   for i = 1 to n do
   {
        if place(k,i) then
            x[k] = i
        if (k=n) then write x[1..n]
        else
            Nqueen(k+1, n)
    }
}
```

Algorithm: Part 2

- The place(k,i) accepts different values of “i“ for “k” queen and check the constraints, if satisfied then returns true value. (“i“ represent column)
- Condition checked:
- For the “k” queen, in “x” array from [1..k-1] the location should not be used.
- Similarly diagonal condition is checked.

Algorithm: Part 2

```
Algorithm place(k,i)
{
  //Assume that k-1 locations are already computed in array
"x".  Absolute function is used in the algorithm
  for j = 1 to k-1 {
    if(x[j]=i)  or
        (abs(x[j]-i = abs(j-k)) then
          return false }
    return true;
}
```

Applications:
Game designing
Security
Generating obstacles in space
Construction planning