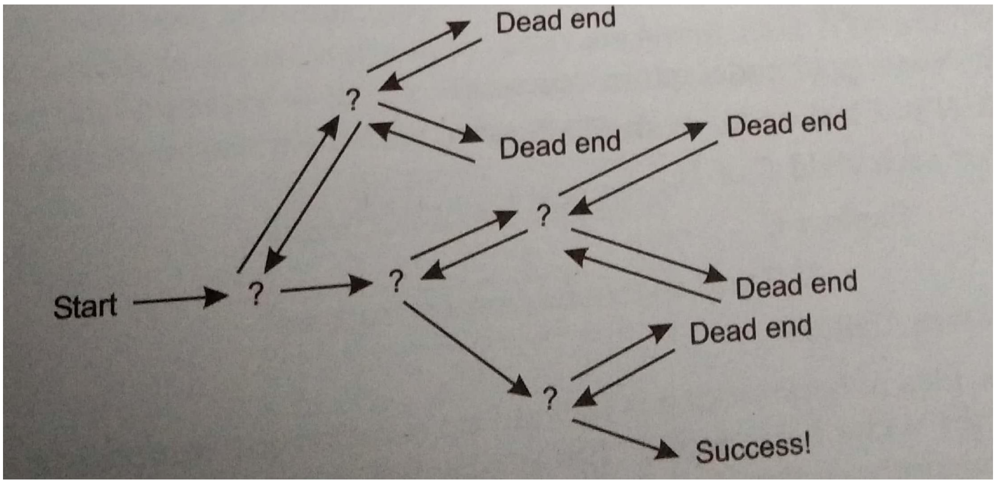
BACK TRACKING

We all seen poor blind people walking in roads.. If they find any obstacles in their way, they would just move backward. Then they will proceed in other direction. A blind person can do this by “intelligence”. similarly, if an algorithm backtracks with intelligence, it is called ***Backtracking algorithm***

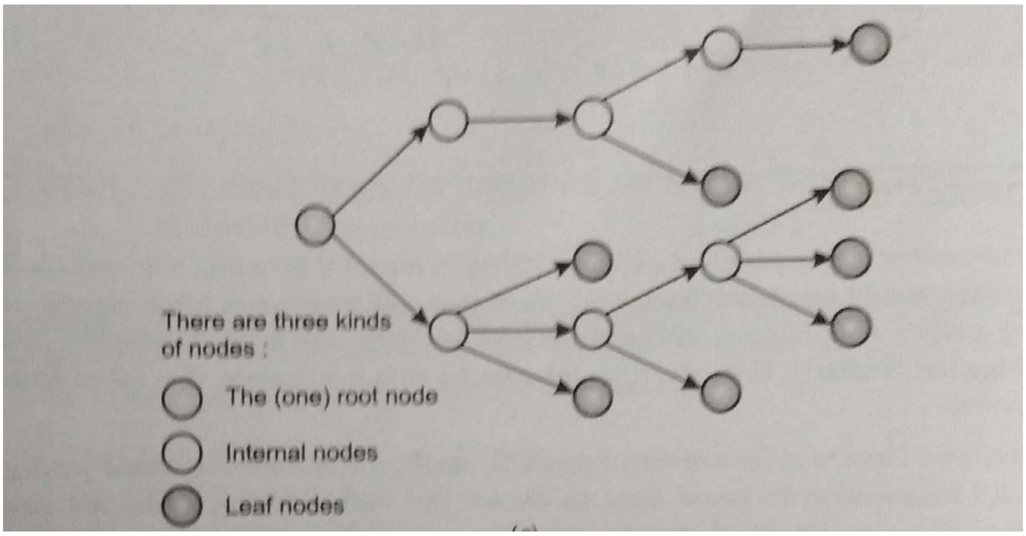
Suppose we have to make a series of decisions, among various choices, where we don’t have enough information to know what to choose and each decision leads to a new set of choices. Some sequence of choices may be a solution to our problem. Backtracking is a methodical way of trying out various sequences of decisions, until we find one that "works".

In the following figure,

- Each non-leaf node in a tree is a parent of one or more other nodes (its children)
- Each node in the tree, other than the root, has exactly one parent



A type of data structure called “tree”,usually composed of nodes. Backtracking can be thought of as searching a tree for a particular “goal” leaf node. Here we are not using the tree data structure. Actually,if we diagram the sequence of choices we make, the diagram looks like a tree, Our backtracking algorithm "sweeps out a tree" in “problem space”.



Backtracking is really quite simple -we "explore" each node, as follows:

To explore node N:

1. If N is a goal node,return "success"
2. If N is a leaf node,return "failure"
3. For each child C of N,
   Explore C
      If C was successful, return "success"
4. Return "failure"

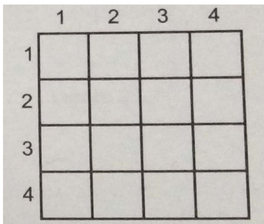
- The basic idea of backtracking is to build up a vector one component at a time and to test whether the vector being formed has any chance of success.
- The major advantage of backtracking algorithm is that if it is realized that the partial vector generated does not lead to an optimal solution then that vector may be ignored.
- Backtracking algorithm determines the solution by systematically searching the space for the given problem.
- Backtracking is a depth first search with some bounding function.
- All solutions using backtracking are required to satisfy a complex set of constraints. The constraints may be **explicit or implicit.**
  - Explicit constraints are rules, which restrict each vector element to be chosen from the given set.
  - Implicit constraints are rules,which determine which of the tuples in the solution space,actually satisfy the criterion function.

N-QUEENS PROBLEM

N-Queens problem is to place n-queens in such a manner on an n x n chessboard that no two queens attack each other by being in the same row,column or diagonal.
It can be seen that for n=l, the problem has a trivial solution, and no solution exists for n=2 and n=3. So first we will consider the 4-queens problem and then generalize it to n-queens problem.

4-**queens problem**

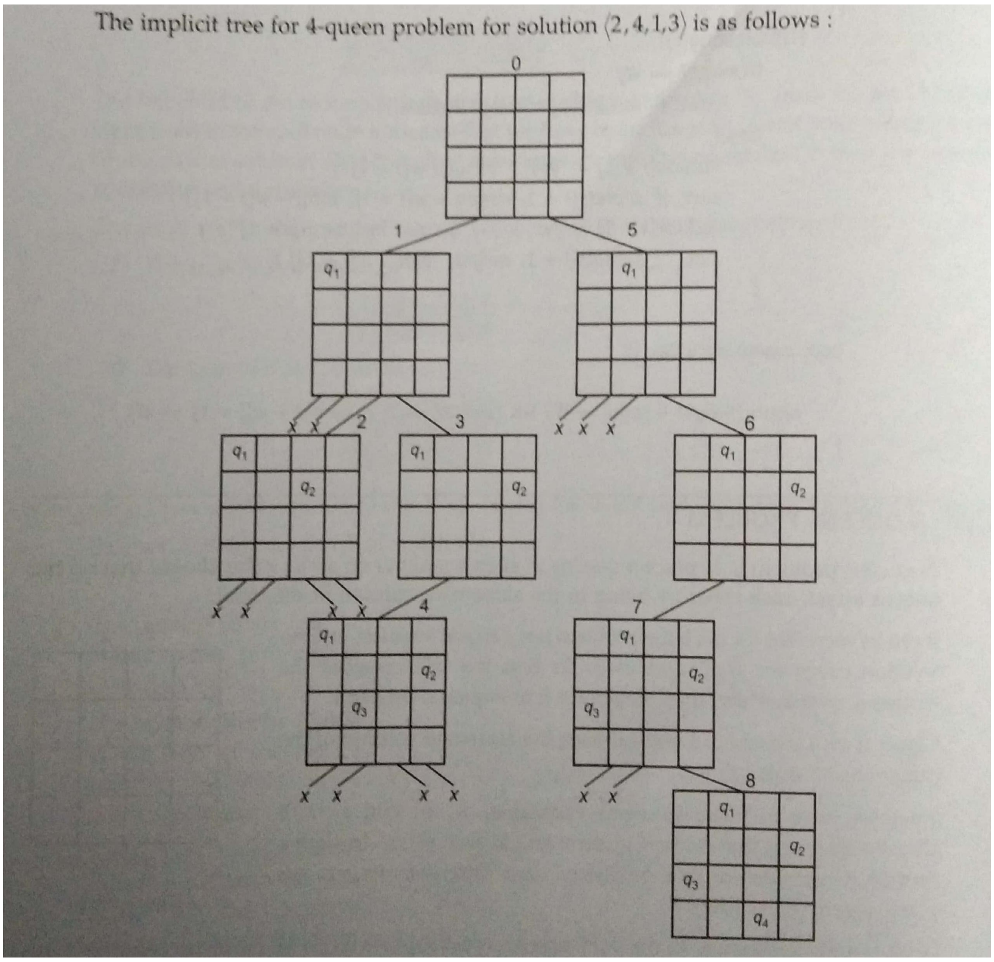
Given a 4x4 chessboard and number the rows and column of the chessboard 1 through Since we have to place 4 queens such as q_1, q_2, q_3 and q_4 on a chessboard,such that no two queens attack each other.



4 × 4 chessboard

In such a condition each queen must be placed on a different row, i.e., place queen "i" on row "i".
we place queen q_1 in the very first acceptable position(1, 1). Next,we place queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2 then the dead end is encountered. Thus the first acceptable position for q_2 is column 3 ie, (2, 3)but then no position is left for placing queen q_3 safely. So we backtrack one step Place the queen q_2 in (2, 4),the next best possible solution. Then we obtain the position for placing q_3 which is (3, 2), But later this position also leads to dead end and no place is

found where q_4 can be placed safely. Then we have to backtrack till q_1 and place it to (1, 2) and then all the other queens are placed safely by moving q_2 to (2, 4), q_3 to(3, 1)and q_4 to (4, 3), That is,we get the solution (2, 4, 1, 3). This is one possible solution for 4-queens problem.For other possible solution the whole method is repeated for all partial solutions. The other solution for 4-queens problem is (3, 1, 4, 2). It can be seen that all the solutions to the 4-queens problem can b represented as 4-tuples (x_1, x_2, x_3, x_4) where "x_i" represents the column on which queen "q_i" is placed.



The following figure shows the complete state space for 4-queens problem.

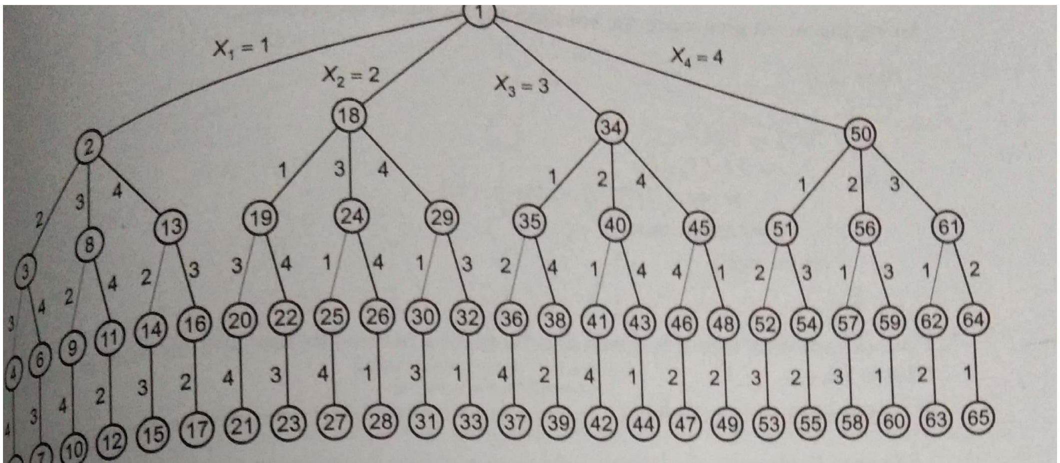
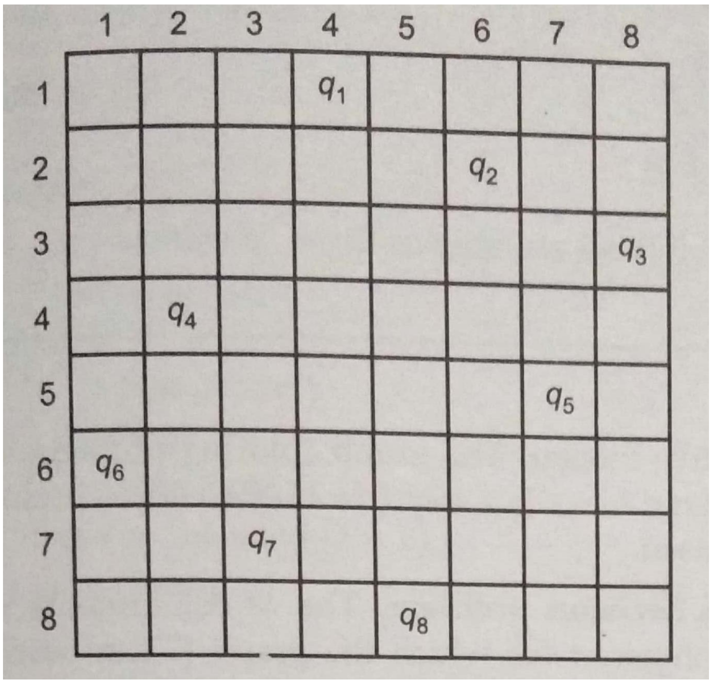


Figure 23.6 4-queens solution space with nodes numbered in DFS

8-queens problem

One possible solution for 8-queens problem is shown below. The solution space of the following solution is (4, 6, 8, 2, 7, 1, 3, 5)



N-queens problem

- If two queens are placed at positions (i, j) and (k, l) then, they are on the same diagonal only if $(i\text{-}j) = k\text{-}I$ or $i+k = k+l$.
  - ✓ The *first* equation implies that $j - l = i - k$
  - ✓ The *second* equation implies that $j - l = k - I$

- Therefore, two queens lie on the same diagonal if and only if $|j - l| = |i - k|$

Using Place() algorithm, we give a precise solution to the n-queens problem.
Place(k, i) returns a Boolean value that is true if the k^{th} queen can be placed in column i. It tests both whether i is distinct from all previous values x_1, x_2, ….., x_{k-1} and whether there is no other queen on the same diagonal.

```
Place (k,i)
    {
        For j ← 1 to k − 1
            do if ( x[j] = i)
                or (Abs (x[j]) − i) = (Abs (j − k))
            then return false ;
        return true ;
    }
```

```
N-Queens (k, n)
{
    for i ← 1 to n
        do if Place (k, i) then
            { x[k] ← i ;
                if (k = n) then
                    write (x[1..n]) ;
                else
                    N-Queens (k + 1, n) ;
            }
}
```