Backtracking

BACKTRACKING

- It is one of the most general algorithm design techniques.
- Many problems which deal with searching for a set of solutions or for a optimal solution satisfying some constraints can be solved using the backtracking formulation.
- To apply backtracking method, tne desired solution must be expressible as an n-tuple (x1…xn) where xi is chosen from some finite set Si.
- The problem is to find a vector, which maximizes or minimizes a criterion function P(x1….xn).
- The major advantage of this method is, once we know that a partial vector (x1,…xi) will not lead to an optimal solution that (m_{i+1}………..m_n) possible test vectors may be ignored entirely.

- Many problems solved using backtracking require that all the solutions satisfy a complex set of constraints.

- These constraints are classified as:

  i) Explicit constraints.
  ii) Implicit constraints.

1) Explicit constraints:

Explicit constraints are rules that restrict each X_i to take values only from a given set.

Some examples are,

$X_i \geq 0$ or S_i = {all non-negative real nos.}

$X_i = 0$ or 1 or $S_i = \{0,1\}$.

$L_i \leq X_i \leq U_i$ or $S_i = \{a: L_i \leq a \leq U_i\}$

- All tupules that satisfy the explicit constraint define a possible solution space for I.

2) Implicit constraints:

The implicit constraint determines which of the tuples in the solution space I can actually satisfy the criterion functions.

Algorithm:

```
Algorithm IBacktracking (n)
// This schema describes the backtracking procedure .All solutions are generated in X[1:n]
//and printed as soon as they are determined.
 {
   k=1;
   While (k ≠ 0) do
   {
     if (there remains all untried
     X[k] ∈ T (X[1],[2],.....X[k-1]) and Bk (X[1],.....X[k])) is true ) then
     {
      if(X[1],......X[k] )is the path to the answer node)
      Then write(X[1:k]);
      k=k+1;              //consider the next step.
    }
  else k=k-1;                  //consider backtracking to the previous set.
 }
}
```

- All solutions are generated in X[1:n] and printed as soon as they are determined.

N – Queens problem

The problem is to place n queens on an n x n chessboard so that no two “attack” that is no two queens on the same row, column, or diagonal.

Defining the problem:-

- **Assume rows and columns of chessboard are numbered 1 through n.**

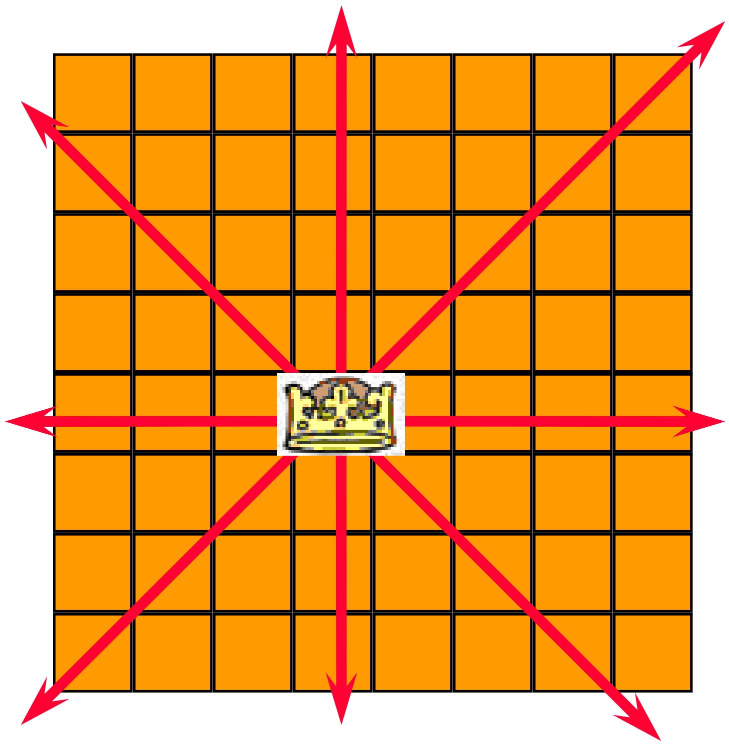
- **Queens also be numbered 1 through n.**

- **Since each queen must be on a different row ,hence assume queen i is to be placed on row i.**

- **Therefore all solutions to the n-queens problem can be represented as n-tuples ($x_1,x_2,\ldots..x_n$), where x_i is the column on which queen i is placed.**

n-Queens Problem

A queen that is placed on an n x n chessboard, may attack any piece placed in the same column, row, or diagonal.



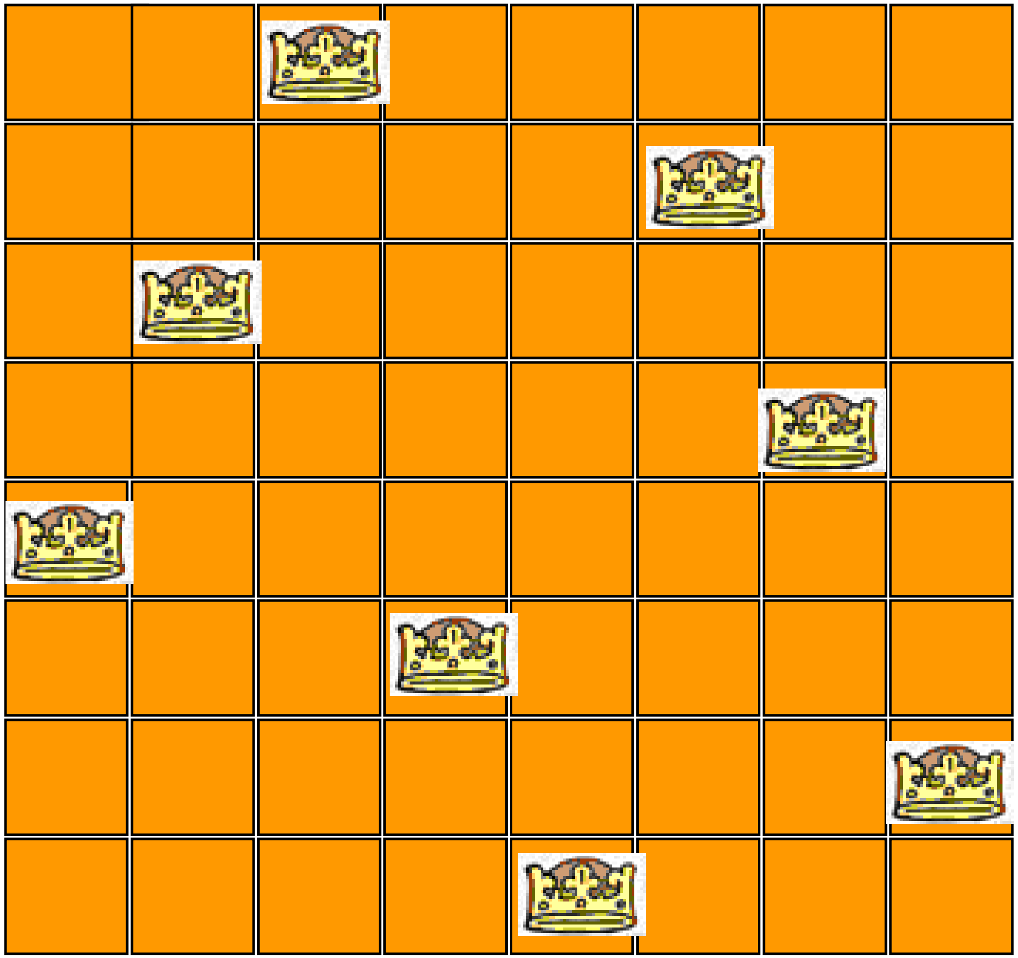
8x8 Chessboard

4-Queens Problem

Can n queens be placed on an n x n chessboard so that no queen may attack another queen?

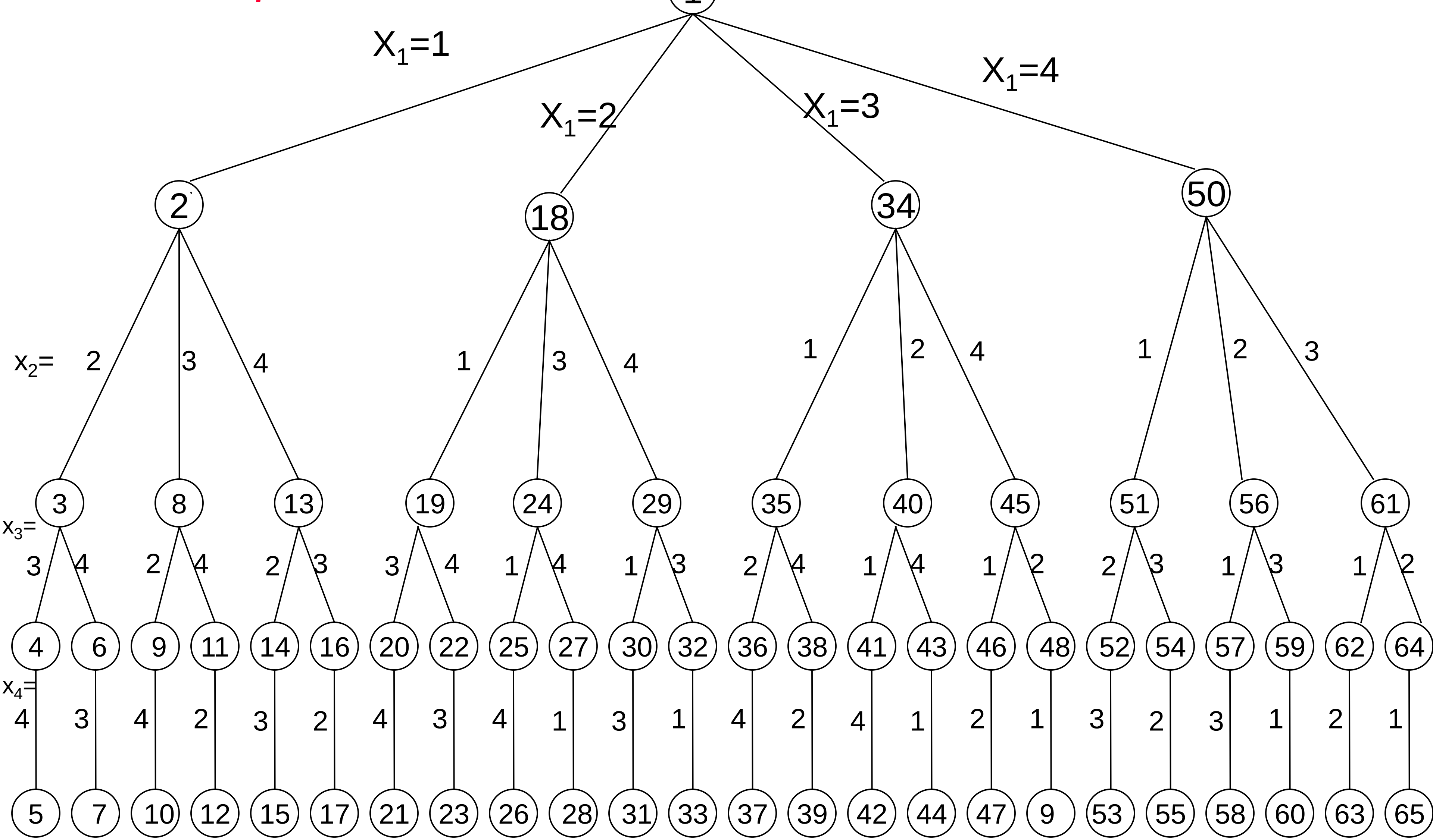
4x4

8-Queens Problem

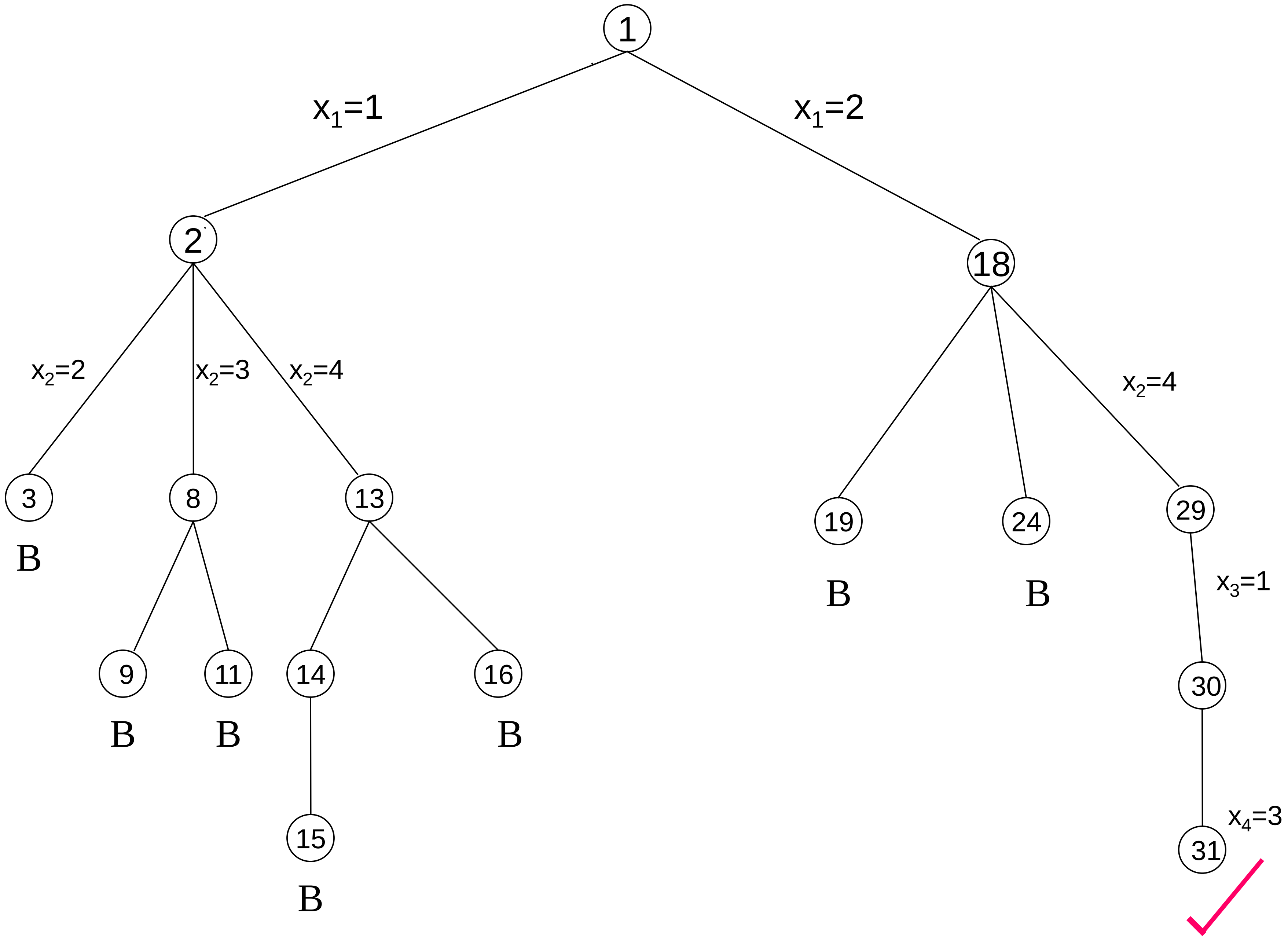


8x8

Tree structure for the case n=4.



Tree organization of the 4-queens solution space. Nodes are numbered as in **depth first search.**



Portion of the tree that is generated during backtracking(n=4).

n - queens problem algorithm

- Every element on the same diagonal that runs from the upper left to the lower right has the same row – column value.
- Similarly, every element on the on the same diagonal that goes from the upper right to the lower left has the same row + column value.

```
Algorithm Nqueen(k, n)
// Using backtracking,  this procedure prints all possible placements
// of n queens on an n×n chessboard so that they are nonattacking.
{
        for i = 1 to n do        // check place of column for queen k
        {
                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)

// It returns true if a queen can be placed in kth row and ith
// column . Otherwise it returns false. x[] is a goal array whose
// first(k-1) values have been set. Abs(r) returns the absolute value of r.

```
{
    for j = 1 to k-1 do
    {       // Two in the same column  or in the same diagonal
        if (  ( x [ j ] = i )   or   ( Abs( x[ j ] - i ) = Abs( j - k ) ) )  then
                return false;
    }
 return true ;
}
```

```
Enter the no. of queens:- 4

The solution is:-
.       Q       .       .

.       .       .       Q

Q       .       .       .

.       .       Q       .


The solution is:-
.       .       Q       .

Q       .       .       .

.       .       .       Q

.       Q       .       .
```