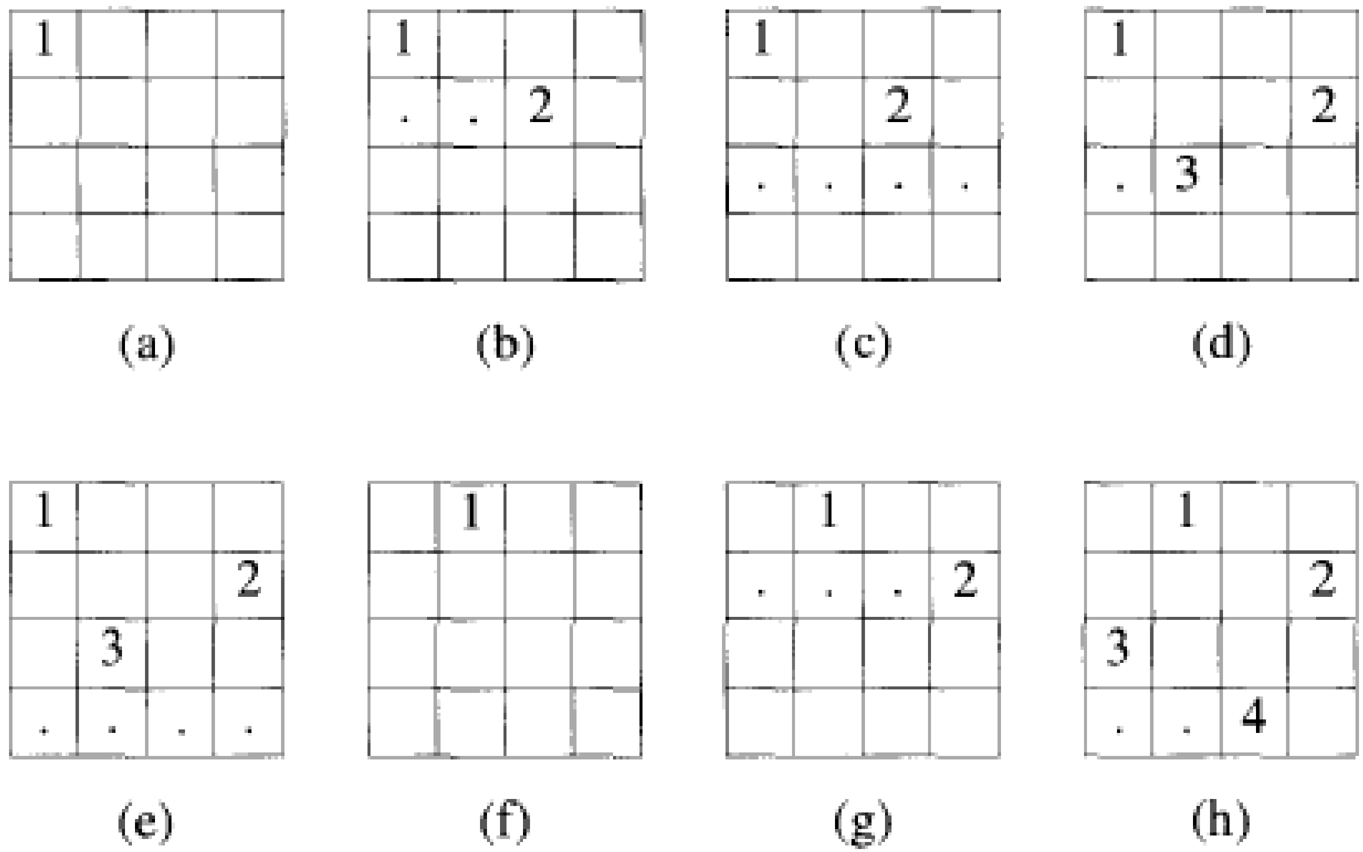
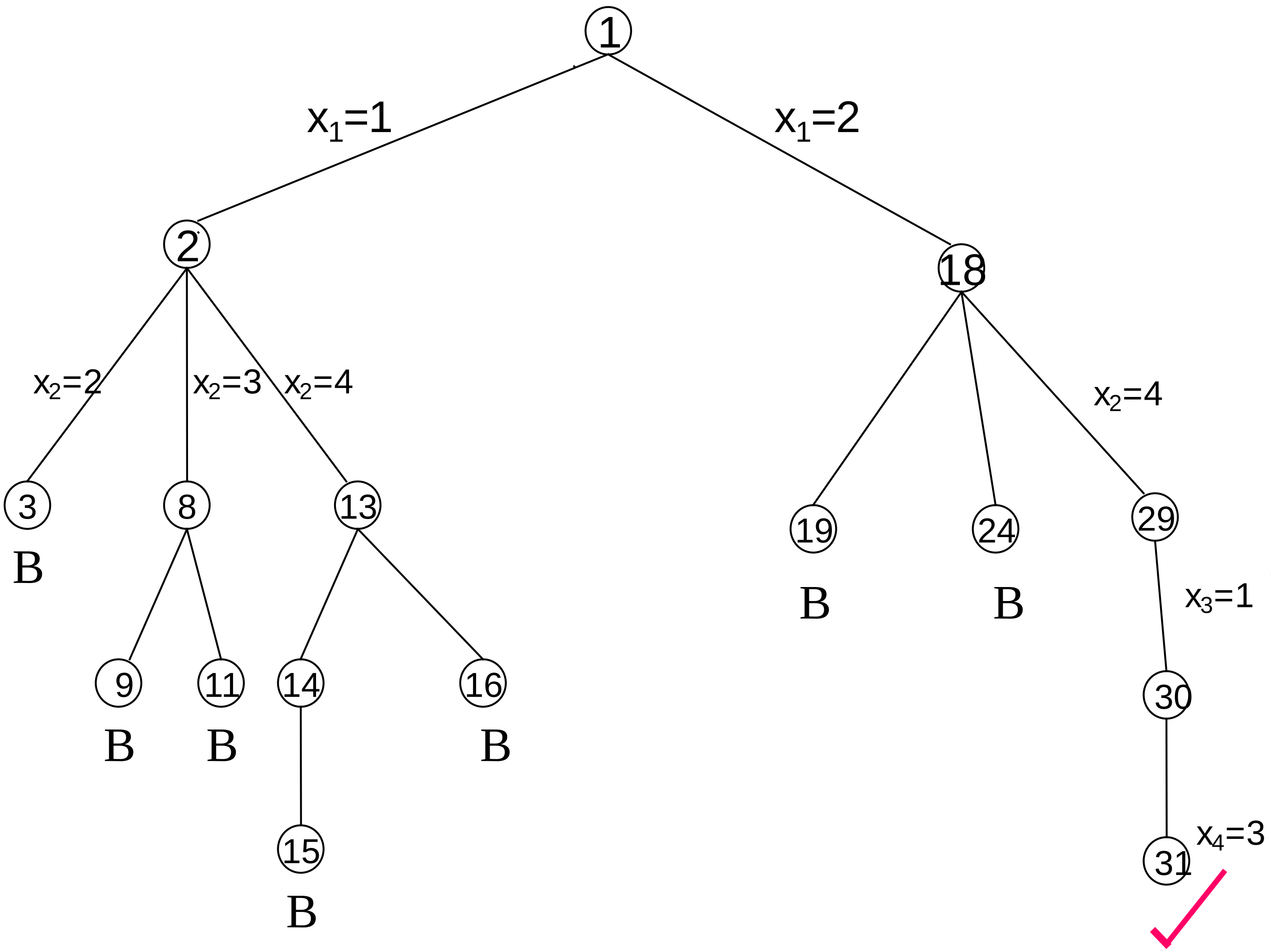
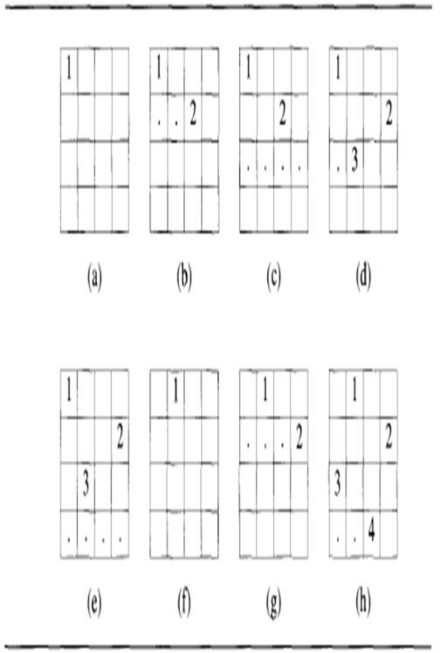
Recursive Algorithm for Backtracking-General Method

```
Algorithm Backtrack(k)
// This schema 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] ∈ T(x[1], ..., x[k - 1]) do
    {
        if (B_k(x[1], x[2], ..., x[k]) ≠ 0) then
        {
            if (x[1], x[2], ..., x[k] is a path to an answer node)
                then write (x[1 : k]);
            if (k < n) then Backtrack(k + 1);
        }
    }
}
```


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

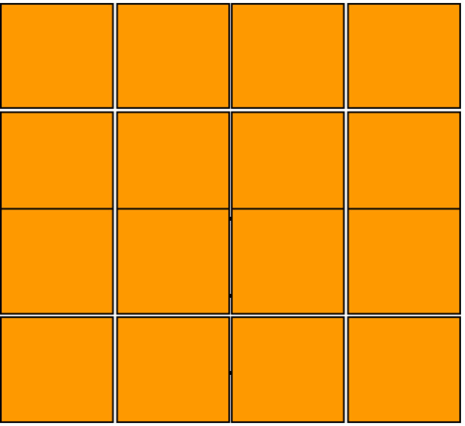
State Space Tree for 4-queens problem





N-Queens Problem

- **Condition to ensure - No two queens on same Row**:
  - Each queen must be on a different row ,hence 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.
- **Condition to ensure - No two queens on same Column**:
  - Select a distinct value for each x_i
- **Condition to ensure - No two queens on same Diagonal**



(i,j)	(k,l) are diagonal
(2,3)	(1,2), (3,4)
	(1,4), (3,2), (4,1)
(1,2)	(2,3), (3,4)
	(2,1)

N-Queens Problem

- If two queens are placed at positions (i, j) and (k, l). They are on the same diagonal then following conditions hold good.

1) Every element on the same diagonal that runs from the upper left to the lower right has the same (row – column) value.

$$i - j = k - l \text{ -----(1)}$$

2) 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.

$$i + j = k + l \text{ -----(2)}$$

- First equation implies : $j - l = i - k$

  Second equation implies : $j - l = k - i$

- Therefore, two queens lie on the same diagonal if and only if

$$\boxed{|j - l| = |i - k|}$$

Algorithm for N-Queens Problem

```
Algorithm NQueens(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
    {
        if Place(k, i) then
        {
            x[k] := i;
            if (k = n) then write (x[1 : n]);
            else NQueens(k + 1, n);
        }
    }
}
```

Algorithm for N-Queens Problem

```
Algorithm Place(k, 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 the absolute value of r.
{
    for j := 1 to k − 1 do
        if ((x[j] = i) // Two in the same column
            or (Abs(x[j] − i) = Abs(j − k)))
                // or in the same diagonal
            then return false;
    return true;
}
```