Design and Analysis of Algorithm
Unit-4

Backtracking

Backtracking

- **Backtracking** is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

- Many problems which deal with searching for a set of solutions or which ask for an optimal solution satisfying some constraints can be solved using the backtracking formulation.

Backtracking

- In the backtrack method, the desired solution is expressible as an n-tuple $(x_1, x_2, \ldots, x_n)$, where the x_i are chosen from some finite set S_i.
- Using backtracking, we find one vector that maximizes(or minimizes or satisfies) a criterion function $P(x_1, x_2, \ldots, x_n)$.
- Sometimes, this method finds all the vectors that satisfies P.
- If m_i is the size of set S_i, then $m = m_1 m_2 \ldots\ldots m_n$ tuples are possible that satisfy the criterion function P.
- The brute force approach would be to form all these n-tuples, evaluate each one with P, and save those which yield the optimum.
- The backtrack algorithm has as its virtue the ability to yield the same answer with far fewer than m trials.

Backtracking

- Its basic idea is to build up the solution vector one component at a time and to use modified criterion functions $P(x_1, x_2, \ldots, x_i)$. (sometimes called bounding functions) to test whether the vector being formed has any chance of success.

- If the partial vector $(x_1, x_2, \ldots, x_i)$ can in no way lead to an optimal solution, then $m_{i+1} \ldots\ldots m_n$ possible test vectors can be ignored entirely.

Backtracking

Explicit constraints

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

Common example of explicit constraints are

(1) $x_i \geq 0$ or S_i = The set of all positive real numbers

(2) $x_i = 0$ or 1 or $S_i = \{0, 1\}$

All tuples that satisfy the explicit constraints define a possible solution space.

Implicit constraints

The implicit constraints are rules that determine which of the tuples in the solution space satisfy the criterion function.

Backtracking

State space tree

When we search the solutions of the problem using backtracking, a tree is formed by the result of search. This tree is said to be sate space tree. Each node in the tree represents a state.

Problem state

Each node in the tree is called problem state.

Solution state

Solution states are those problem states s for which the path from root to s defines a tuple in the solution space.

Answer state

Answer states are those solution states s for which the path from the root to s defines a tuple that is a member of the set of solutions of the problem.

Backtracking

State space

All the paths from the root to other nodes define the state space of the problem.

Live node

A node which has been generated and all of whose children have not yet been generated is called a live node.

E-node

The live node whose children are currently being generated is called the E-node (node being expanded).

Dead node

A dead node is a generated node which is not to be expanded further or all of whose children have been generated.

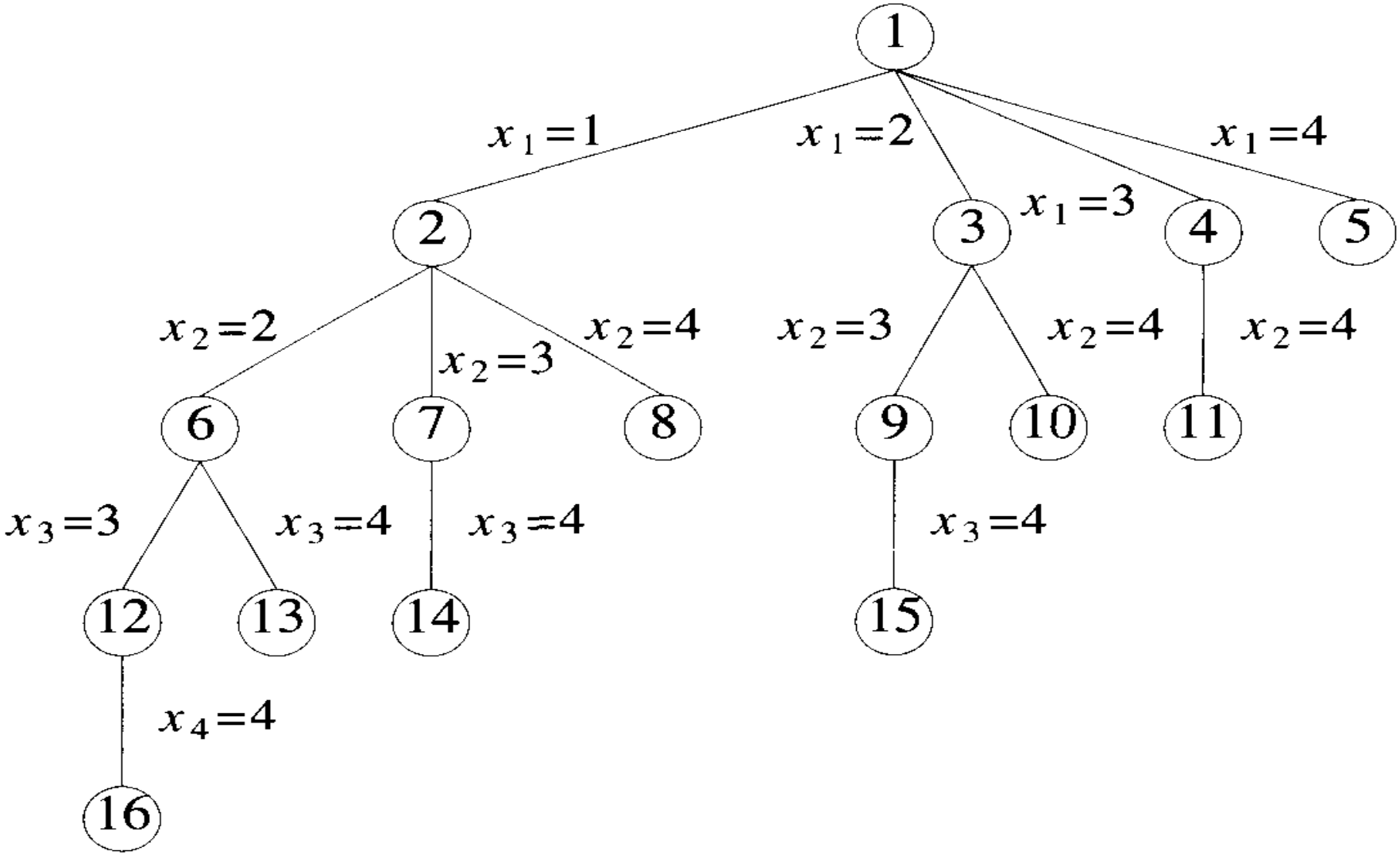
Backtracking

Note: Bounding functions are used to kill live nodes without generating all their children.

Note: Depth first node generation with bounding functions is called backtracking.

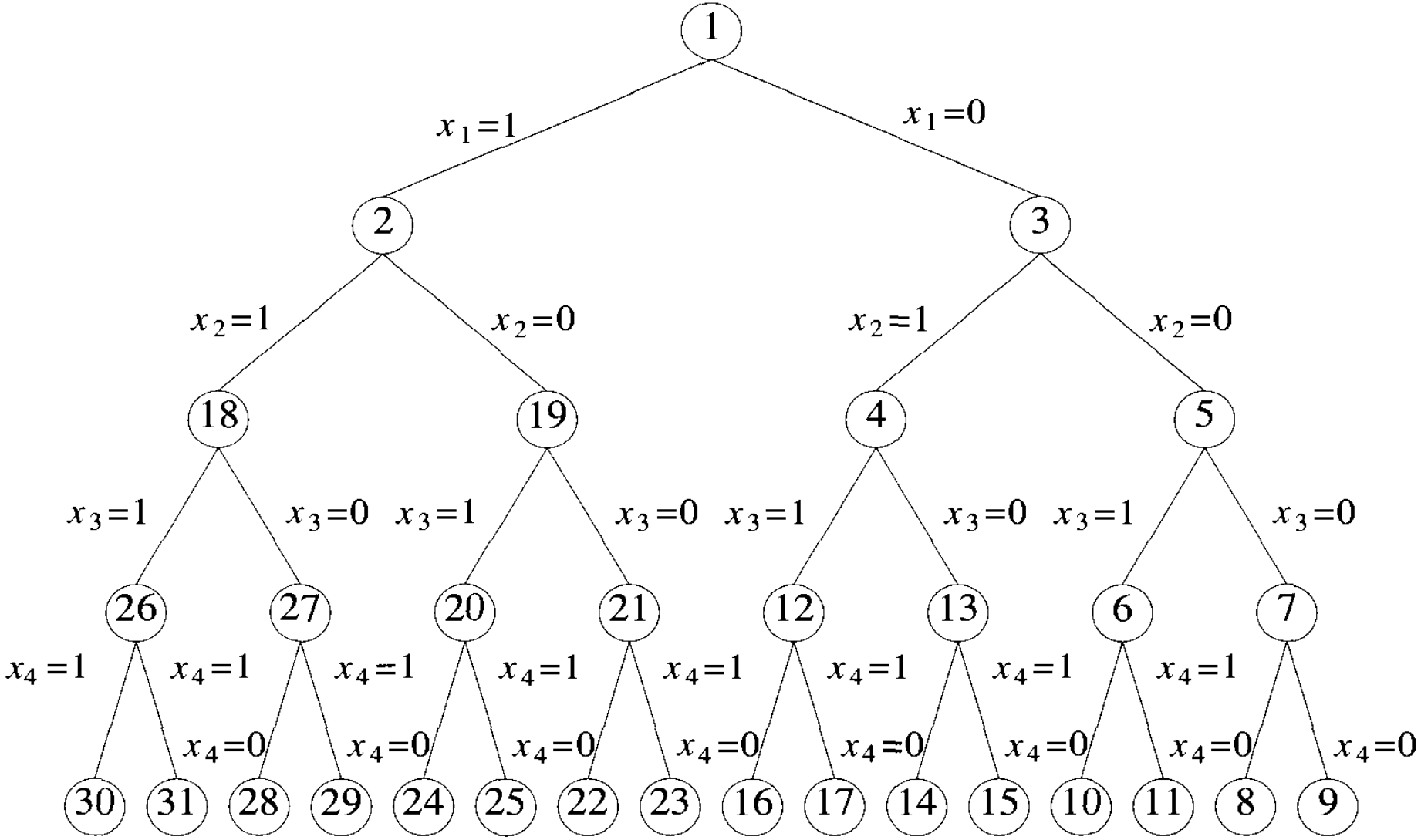
Backtracking

Example:



Backtracking

Example:



N-Queen Problem

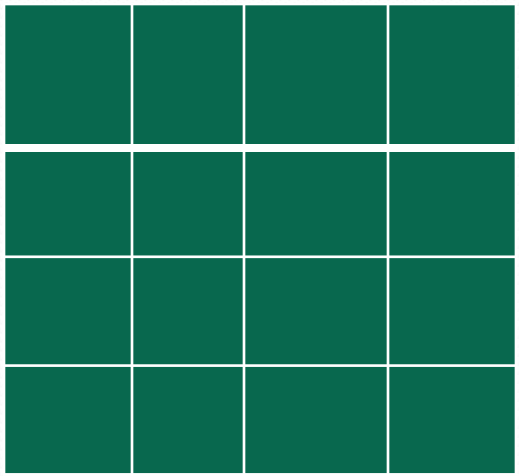
Statement:

In this problem, we have given N queens and a NxN chessboard. The objective of the problem is to place all the N queens on this chessboard in such way no two queens lie on the same row, column or diagonal.

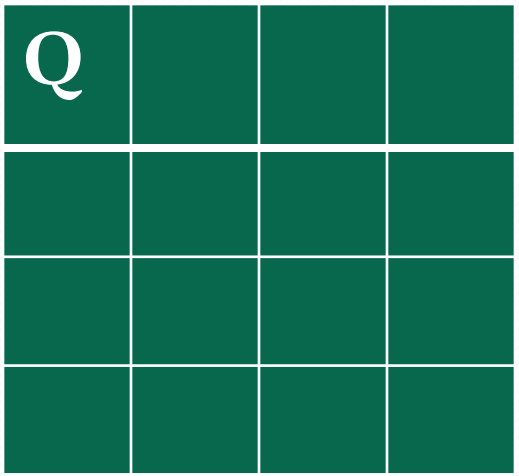
N-Queen Problem

Example: Solve the 4-Queen Problem.

Solution:

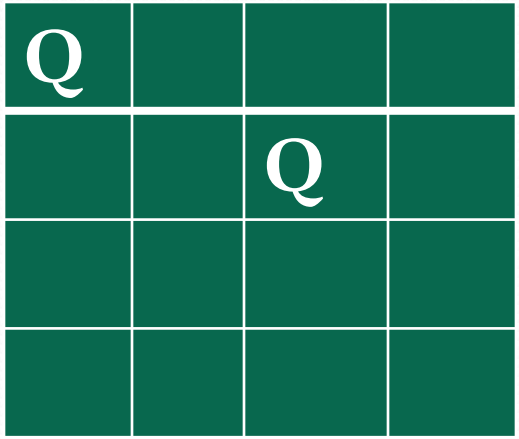


$\Rightarrow$

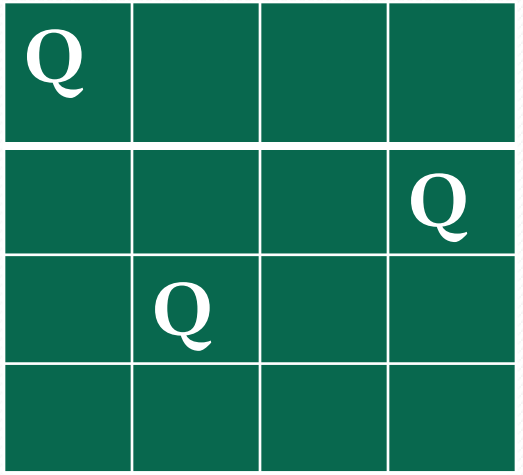


Initial position of chessboard

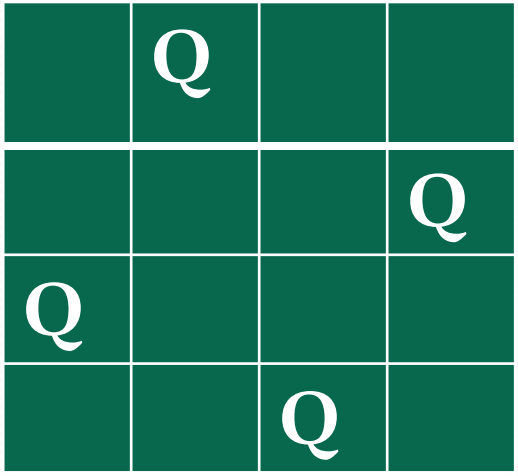
N-Queen Problem



⇒



⇒



Final solution

Backtracking algorithm for N-Queen

- According to this approach, we put 1st queen in 1st row, 2nd queen in 2nd row and so on. We have to only find column number of the queen.
- Therefore, the solution vector will be $(x_1, x_2, ..., x_n)$. Here x_i is the column number of i^{th} queen.
- The explicit constrains for this problem are that no two x_i's can be same and no two queens can be on the same diagonal.
- Using these two constraints, the size of solution space are n!.
- Suppose two queens are placed at positions (i,j) and (k,l). They are on the same diagonal only if

$$i-j = k-l \quad \text{or} \quad i+j = k+l$$

$$\Rightarrow \quad i-k = j-l \quad \text{or} \quad i-k = -(j-l)$$

Therefore, two queens lie on the same diagonal iff

$$|i-k| = |j-l|$$

Backtracking algorithm for N-Queen

An algorithm f or determining the solution of n-queens problem is:-

```
        N-Queen(k, n)
1.      for i ← 1 to n
2.              do if  Place(k,i) = True
3.                      then  x[k] ← I
4.                              if k=n
5.                                      then for  j←1 to n
6.                                                      do print x[j]
7.                              else
8.                                              N-Queen(k+1, n)
```

Backtracking algorithm for N-Queen

An algorithm f or determining the solution of n-queens problem is:-

Place(k, i)

1. for j ← 1 to k-1
2. do if (x[j]=i) or (abs(x[j]-i) = abs(j-k))
3. then return (False)
4. return(True)

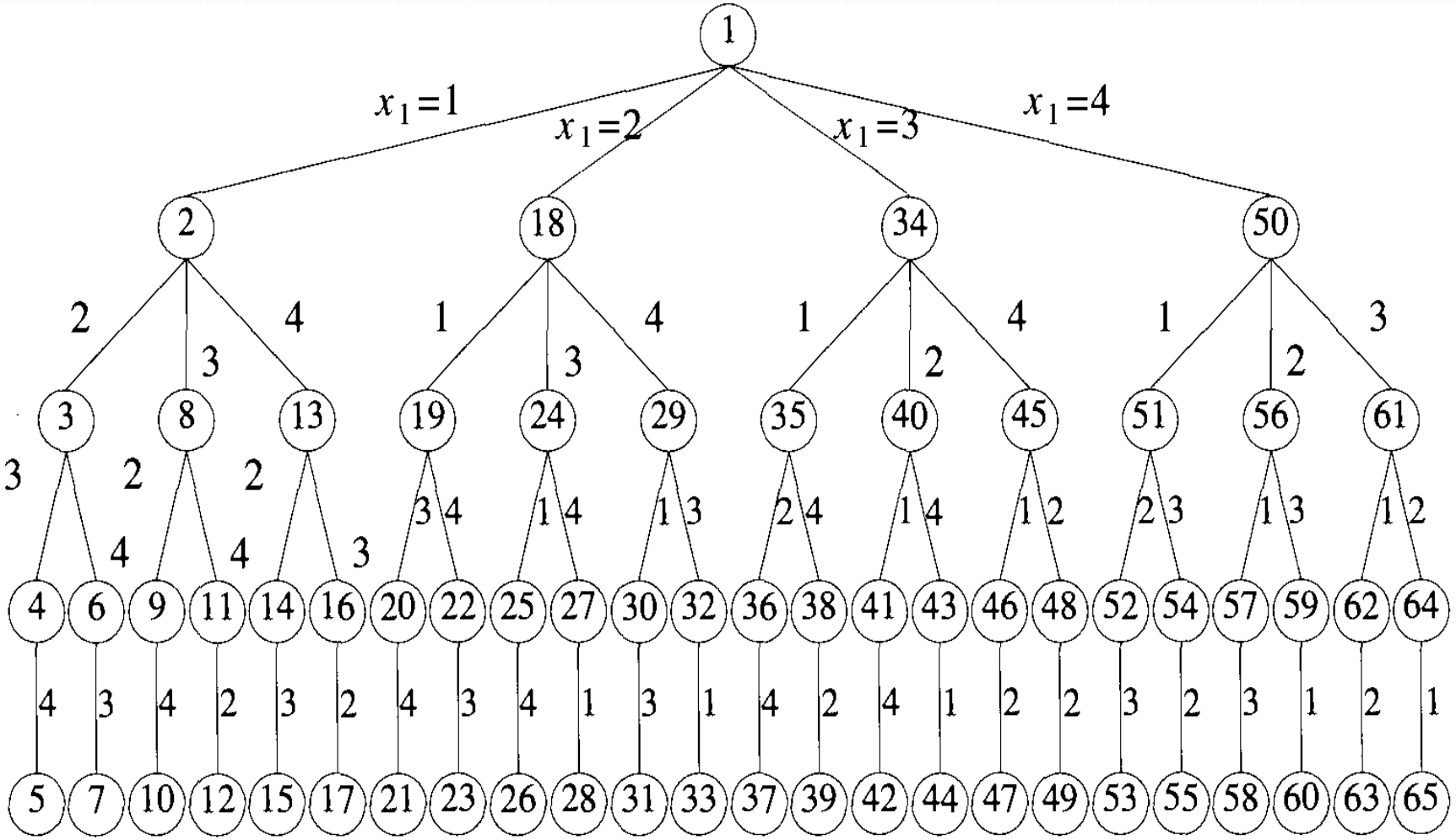
The initial call is N-Queen(1, n).

Running time of this algorithm is O(n!).

Backtracking algorithm for N-Queen

Example: Draw the state space tree for 4-queen problem.

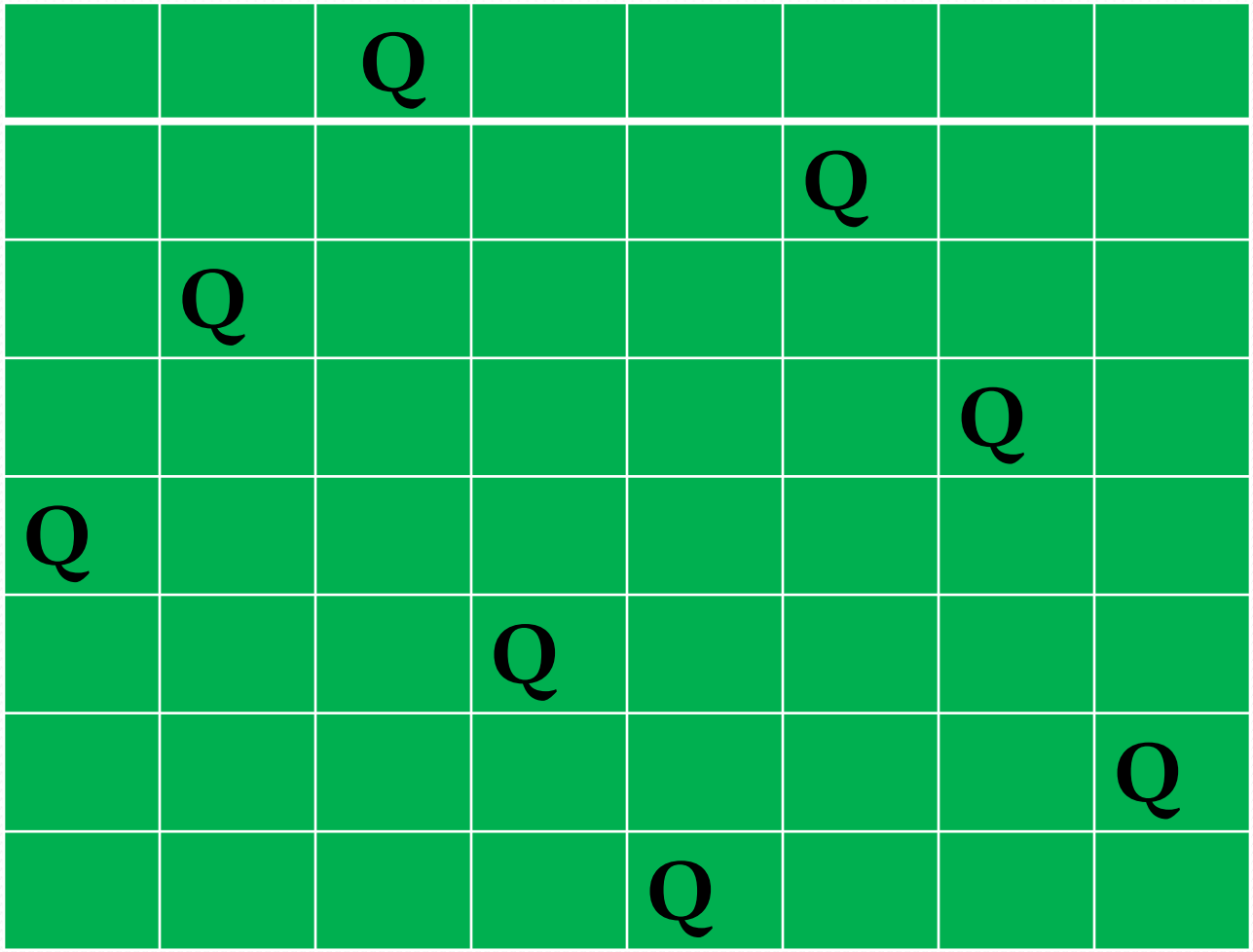
Solution: State space tree for 4-queen problem is the following:-



Backtracking algorithm for N-Queen

Example: Solve the 8-queens problem.

Solution:



Sum of Subsets problem

Statement: In this problem, we are given n distinct positive numbers (called weights) and one another number m. We have to find all combinations of these numbers whose sums are m.

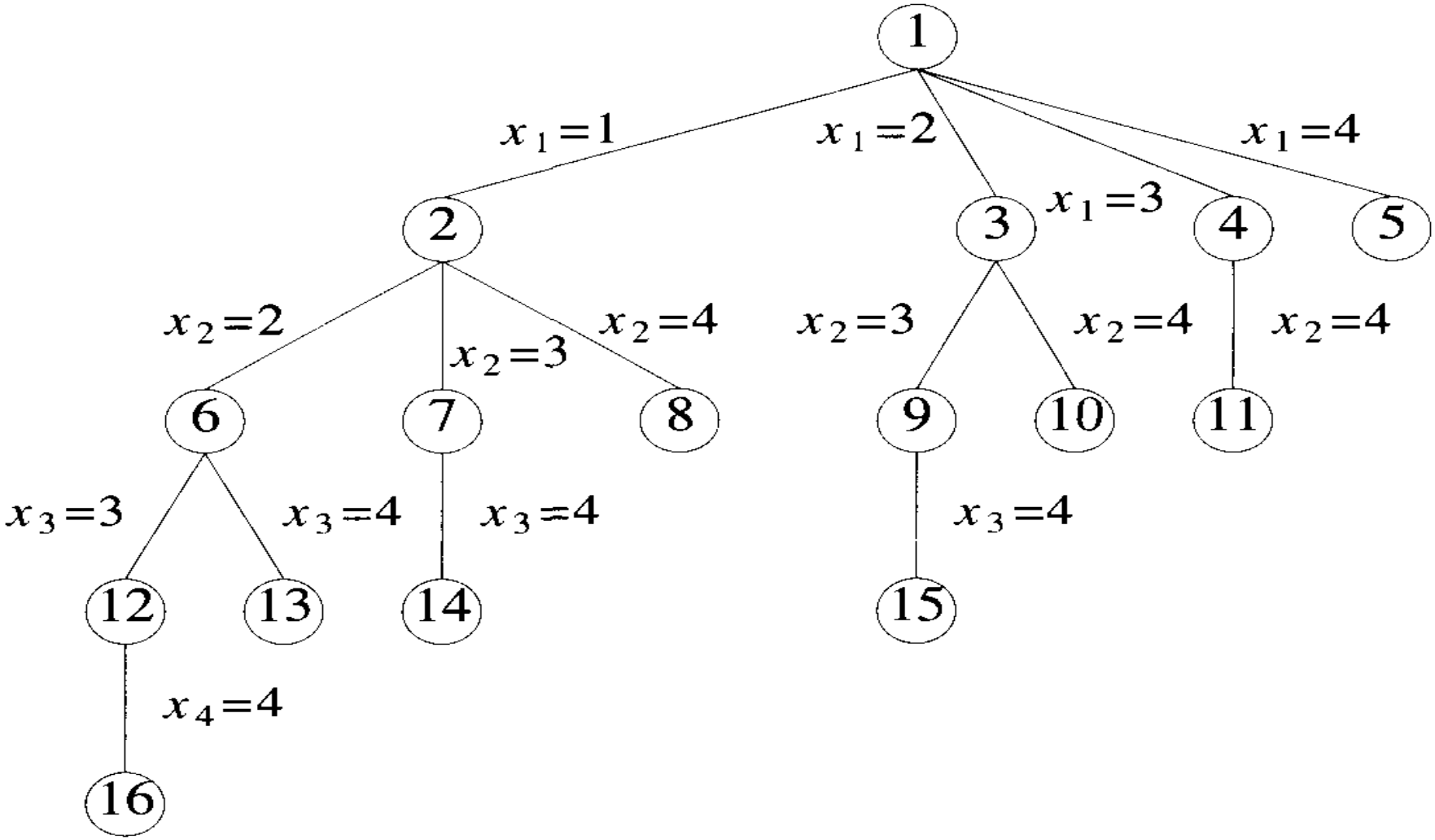
This problem can be formulated using either fixed or variable sized tuples.

Example: Consider n=4 , (w1, w2, w3, w4) = (11, 13, 24, 7) and m=31.

- The desired subsets are (11, 13, 7) and (24, 7).
- The solution in variable-size tuple will be (1, 2, 4) and (3,4).
- The solution in fixed-size tuple will be (1, 1, 0, 1) and (0, 0, 1, 1).

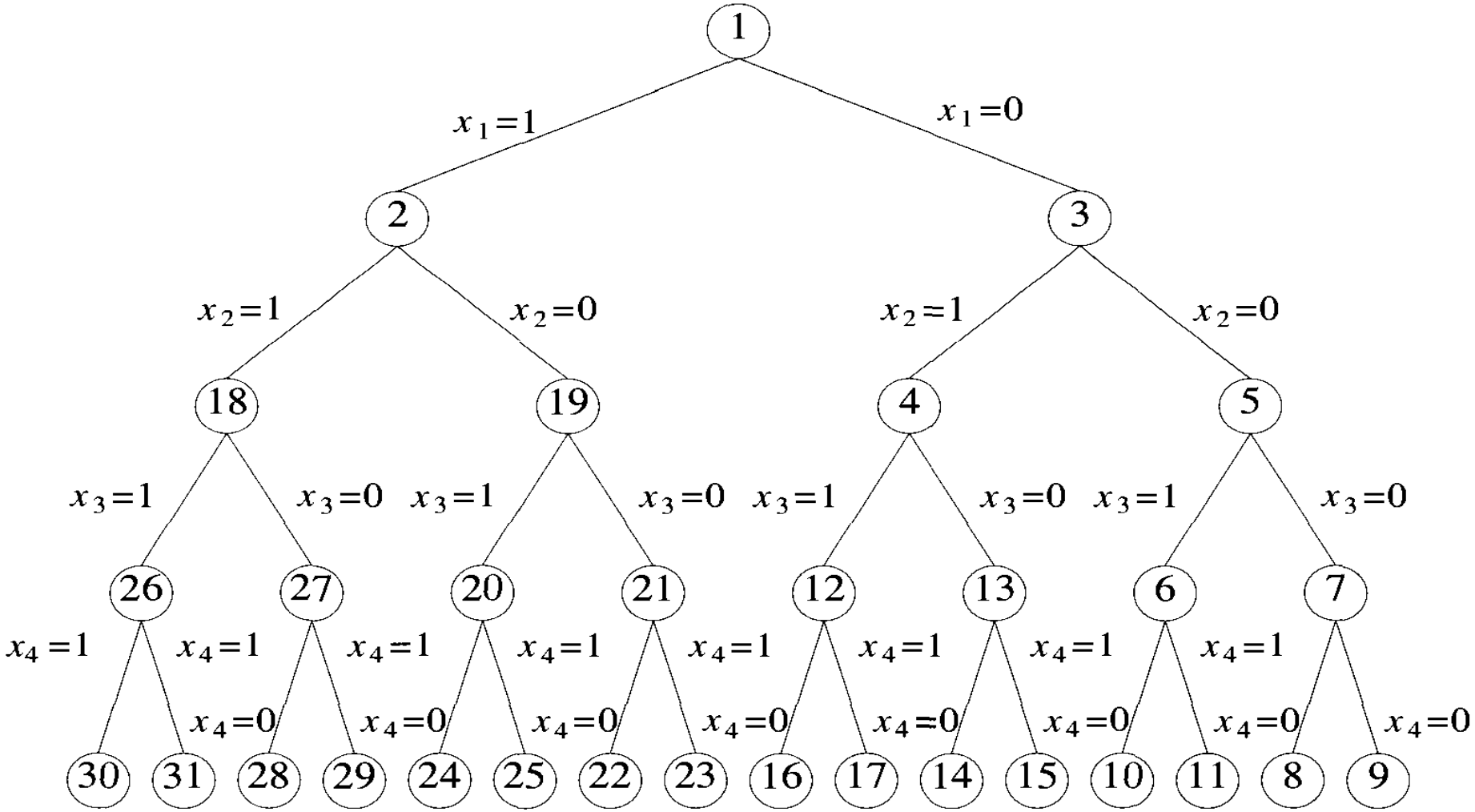
Sum of Subsets problem

The state space tree for this question in case of variable-size tuple is the following:-



Sum of Subsets problem

The state space tree for this question in case of fixed-size tuple is the following:-



Backtracking approach for Sum of Subsets problem

Backtracking solution for this problem uses fixed-sized tuple strategy.

In this case, the value of x_i in the solution vector will be either 1 or 0 depending on whether weight w_i is included or not.

The bounding function is $B_k(x_1, x_2, \ldots\ldots, x_k)$ = true iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

Clearly $(x_1, x_2, \ldots\ldots, x_k)$ can not lead to an answer node if this condition is not satisfied.

If all the weights are initially in ascending order then $(x_1, x_2, \ldots\ldots, x_k)$ can not lead to an answer node if

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} > m$$

Backtracking approach for Sum of Subsets problem

Therefore, we use the following bounding function

The bounding function is $B_k(x_1, x_2, \ldots\ldots, x_k)$ = true iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

and

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m$$

Note: Here, we have assume all the weights are in ascending order.

Backtracking approach for Sum of Subsets problem

```
SumOfSub(s,k,r)
1.    x[k] ← 1
2.    if (s+w[k] = m)
3.          for j←1 to k
4.                print x[j]
5.    else if (s+w[k]+w[k+1] ≤ m)
6.                SumOfSub(s+w[k], k+1, r-w[k])
7.    if (( s+r-w[k] ≥ m) and (s+w[k+1] ≤ m))
8.          x[k] ← 0
9.           SumOfSub(s, k+1, r-w[k])
```

The initial call is SumOfSub(0, 1, $\sum_{i=1}^{n} w_i$)

Backtracking approach for Sum of Subsets problem

Time complexity of this algorithm is $O(2^n)$.

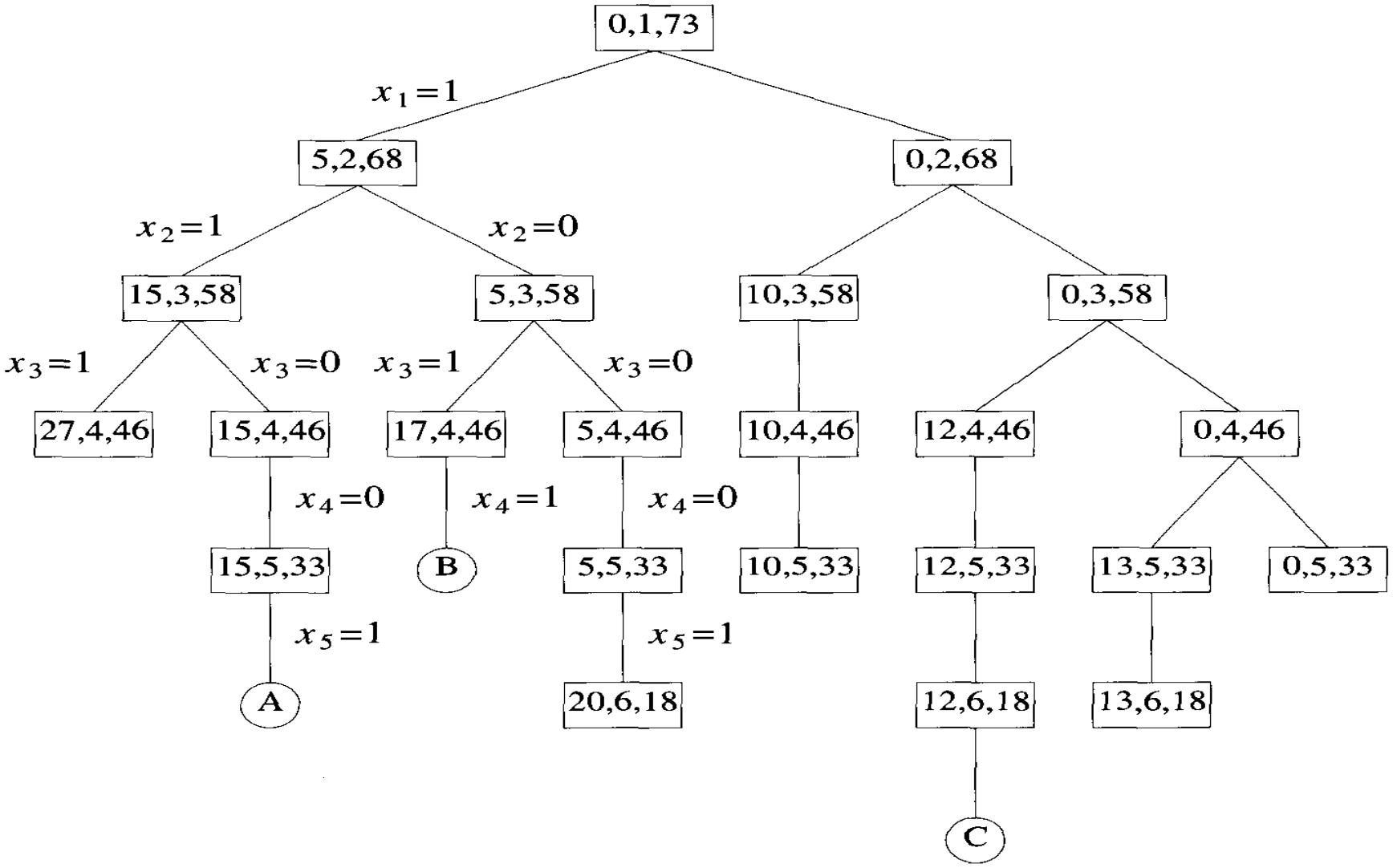
Because

Total number of nodes = $1+2+2^2+2^3+.....+2^n$

$= 2^{n+1}-1$

Example: Let w = {5, 10, 12, 13, 15, 18} and m = 30. Find all possible subsets of w that sum to m. Do this using SumOfSub. Draw the portion of the state space tree that is generated.

Solution:



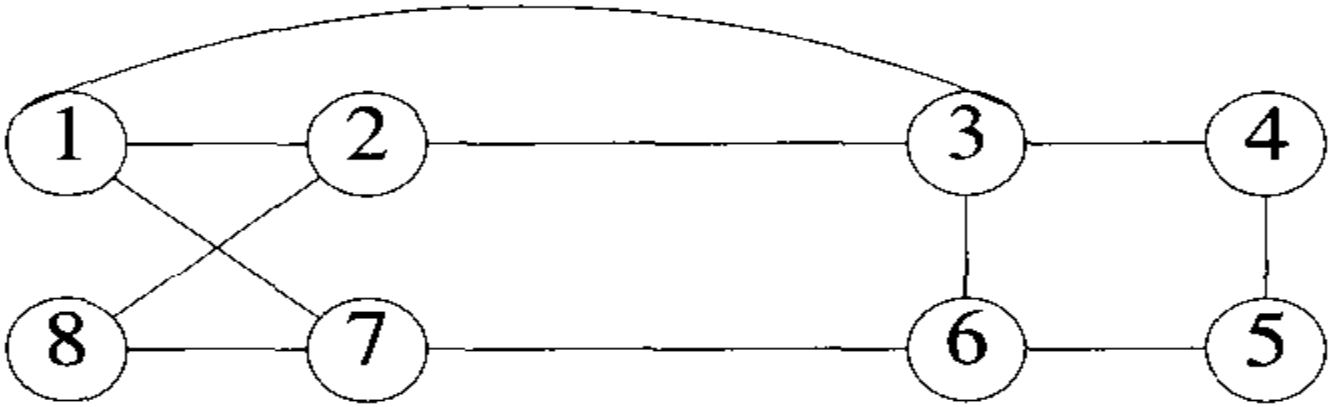
Hamiltonian Cycle Problem

Hamiltonian cycle: A cycle in a graph is said to be Hamiltonian cycle if it contains all the vertices of the graph and no vertex is repeated.

Statement: In Hamiltonian cycle problem, we have to find a Hamiltonian cycle present in the given graph.

Hamiltonian Cycle Problem

Example: Consider the following graph.



Find a Hamiltonian cycle in this graph.

Solution: Hamiltonian cycles are

(1, 3, 4, 5, 6, 7, 8, 2, 1)

(1, 2, 8, 7, 6, 5, 4, 3, 1)

Backtracking algorithm for Hamiltonian Cycle Problem

- The backtracking solution vector $(x_1, x_2, ..., x_n)$ is defined so that x_i represents the i^{th} visited vertex of the proposed cycle.

- First we choose, x_1, can be any vertex of the n vertices.

- We will take $x_1 = 1$. To avoid printing same cycle n-times, we require that $x_1 = 1$.

Backtracking algorithm for Hamiltonian Cycle Problem

- If $1 < k < n$, then x_k can be any vertex v that is distinct from $x_1, x_2, \ldots\ldots, x_{k-1}$ and v is connected by an edge to x_{k+1}. The vertex x_n can only be the one remaining vertex and it must be connected to both x_{n-1} and x_1.

- This algorithm is started by first initializing the adjacency matrix G[1:n][1:n], then x[2:n] to 0 and x[1] to 1 and then executing algorithm Hamiltonian(2).

- Time complexity of this algorithm = **$O(2^n n^2)$.**

Backtracking algorithm for Hamiltonian Cycle Problem

```
1   Algorithm Hamiltonian(k)
2   // This algorithm uses the recursive formulation of
3   // backtracking to find all the Hamiltonian cycles
4   // of a graph. The graph is stored as an adjacency
5   // matrix G[1 : n, 1 : n]. All cycles begin at node 1.
6   {
7       repeat
8       { // Generate values for x[k].
9           NextValue(k); // Assign a legal next value to x[k].
10          if (x[k] = 0) then return;
11          if (k = n) then write (x[1 : n]);
12          else Hamiltonian(k + 1);
13      } until (false);
14  }
```

Backtracking algorithm for Hamiltonian Cycle Problem

```
1   Algorithm NextValue(k)
2   // x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
3   // no vertex has as yet been assigned to x[k]. After execution,
4   // x[k] is assigned to the next highest numbered vertex which
5   // does not already appear in x[1 : k − 1] and is connected by
6   // an edge to x[k − 1]. Otherwise x[k] = 0. If k = n, then
7   // in addition x[k] is connected to x[1].
8   {
9       repeat
10      {
11          x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
12          if (x[k] = 0) then return;
13          if (G[x[k − 1], x[k]] ≠ 0) then
14          { // Is there an edge?
15              for j := 1 to k − 1 do if (x[j] = x[k]) then break;
16                                  // Check for distinctness.
17              if (j = k) then // If true, then the vertex is distinct.
18                  if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
19                      then  return;
20          }
21      } until (false);
22  }
```

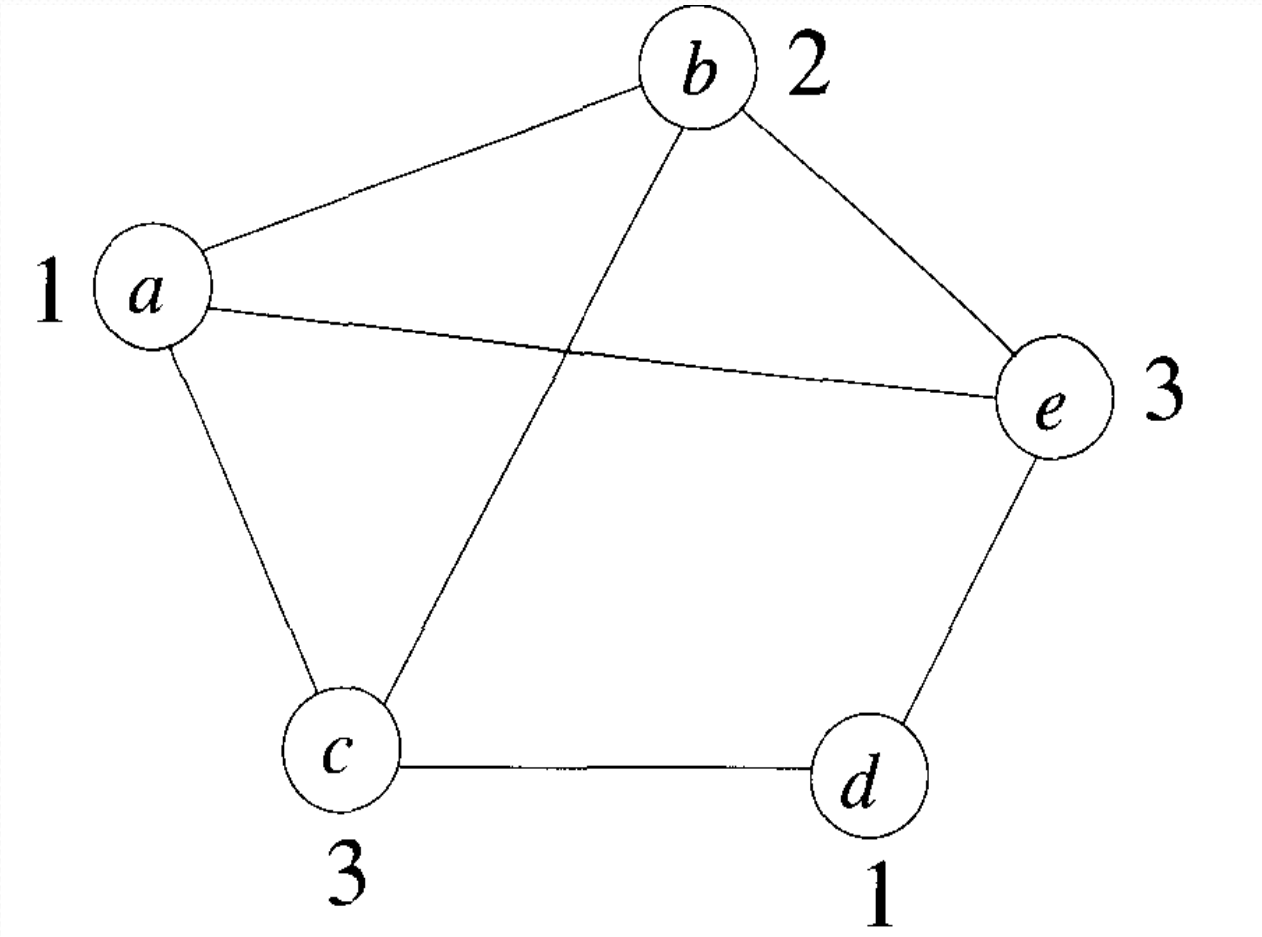
Graph Coloring Problem

- Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used. This is termed the **m-colorability decision problem.**

- The **m-colorability optimization problem** asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the **chromatic number** of the graph.

Graph Coloring Problem

Example: Consider the following graph



Find the chromatic number of this graph.

Backtracking algorithm for Graph Coloring Problem

- Following algorithm determines all the different ways in which a given graph can be colored using at most m colors.

- In this algorithm, a graph is represented by its adjacency matrix G[1:n, 1:n]. The colors are represented by the integers 1, 2, ……., m and the solutions are represented by the n-tuple (x_1, x_2, ……., x_n), where x_i is the color of node i.

- Function **mColoring** is begun by first assigning the graph to its adjacency matrix, setting the array x[] to 0, and then invoking the statement **mColoring(1).**

Backtracking algorithm for Graph Coloring Problem

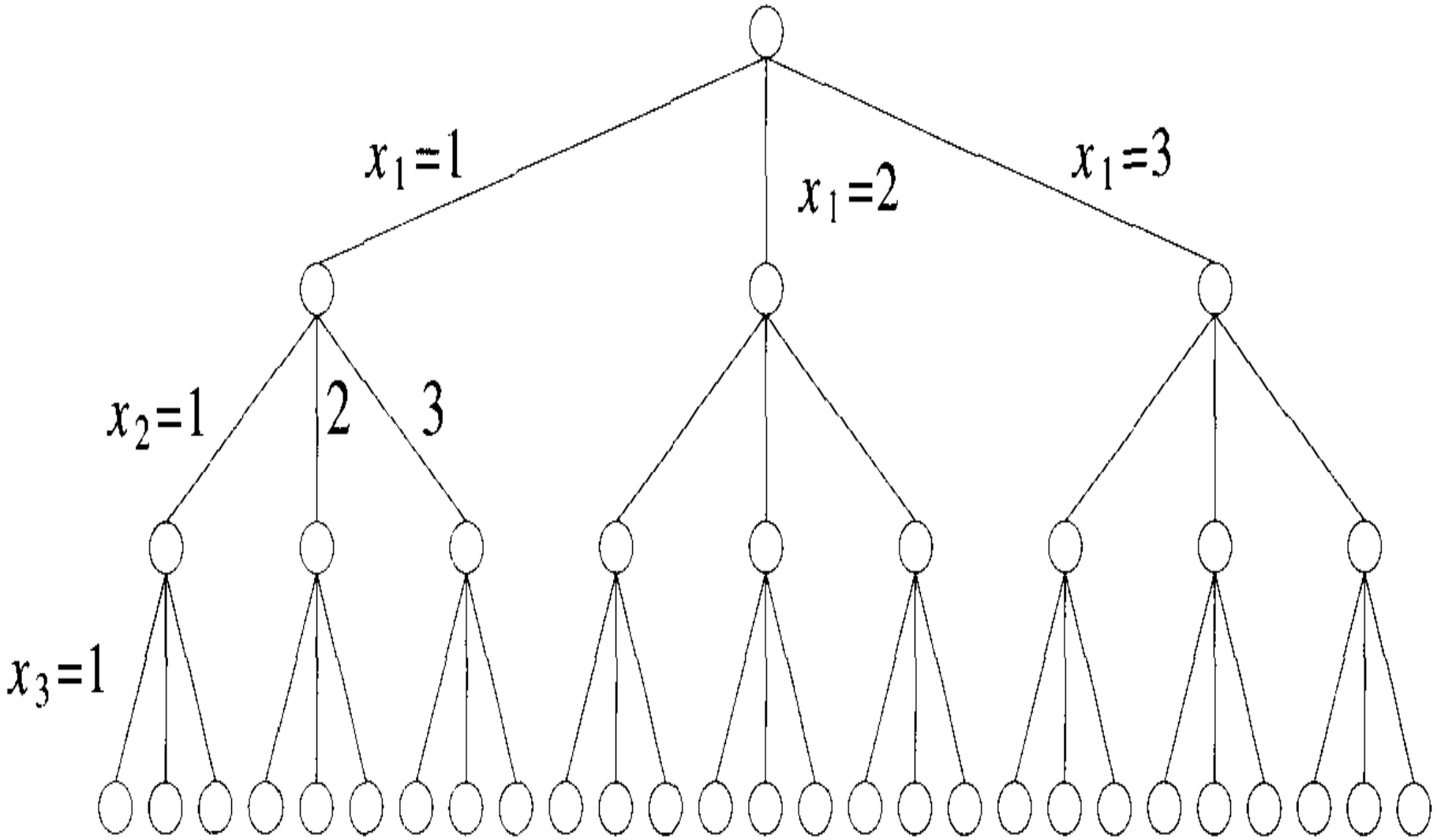
```
mColoring(k)
{
        while(1)
        {
                NextValue(k)
                if (x[k] = 0)
                        return
                if(k=n)
                        print x[1:n]
                else
                        mColoring(k+1)
        }
}
```

Backtracking algorithm for Graph Coloring Problem

```
NextValue(k)
{
        while(1)
        {
                x[k] ← (x[k] + 1) mod (m+1)
                if (x[k] = 0)
                        return
                for ( j← 1  to  n )
                        if(G[k,j] ≠ 0 and (x[k] = x[j]))
                                break
                if (j=n+1)
                        return
        }
}
```

Backtracking algorithm for Graph Coloring Problem

State space tree when n = 3 and m = 3 is



Backtracking algorithm for Graph Coloring Problem

- An upper bound on the computing time of mColoring can be arrived at by the number of internal nodes in the state space tree is $\sum_{i=0}^{n-1} m^i$.

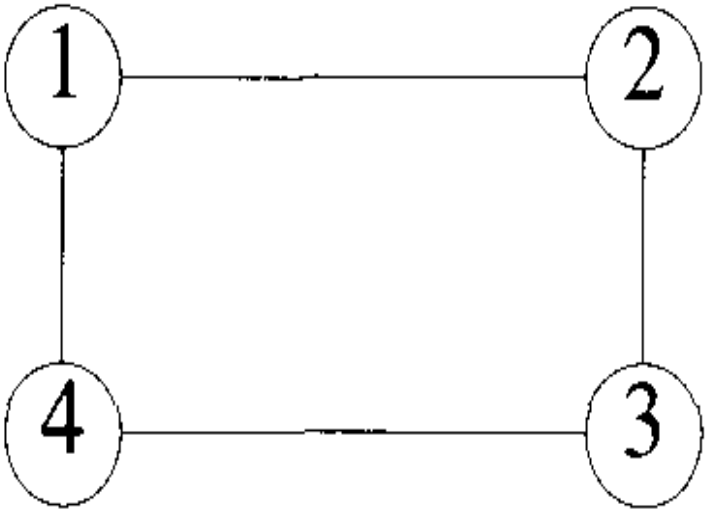
- At each internal node, O(mn) time is spent by NextValue to determine the children corresponding to legal colorings. Hence the total time is bounded by

$$\sum_{i=0}^{n-1} m^{i+1}\, n = \sum_{i=1}^{n} m^i\, n = n(m^{n+1} - m)/(m-1)$$

$$= O(nm^n)$$

Backtracking algorithm for Graph Coloring Problem

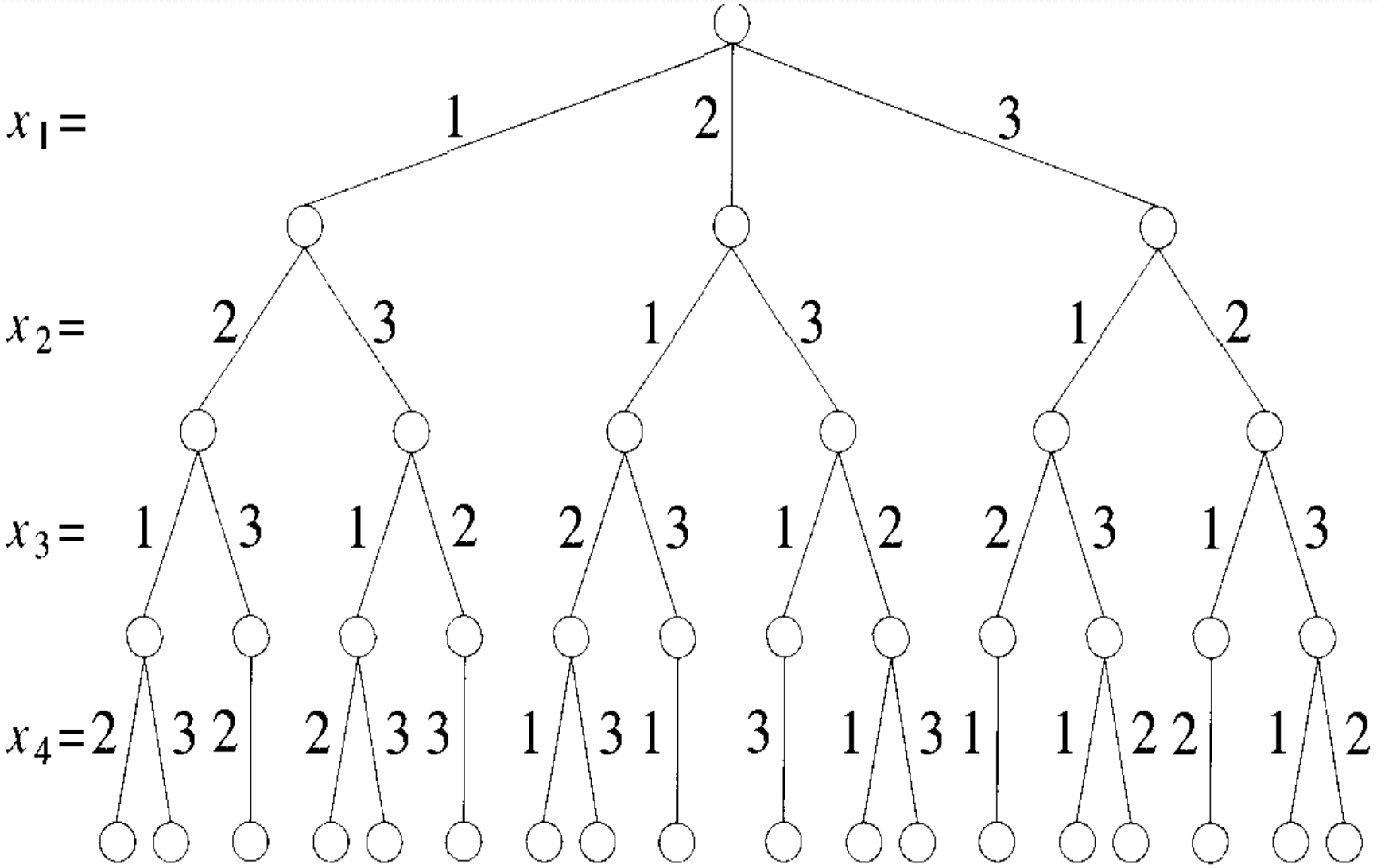
Example: Consider the following graph:-



- Here m = 3.
- Draw state space tree for this graph using mColoring.

Backtracking algorithm for Graph Coloring Problem

Solution: State space tree for the graph will be



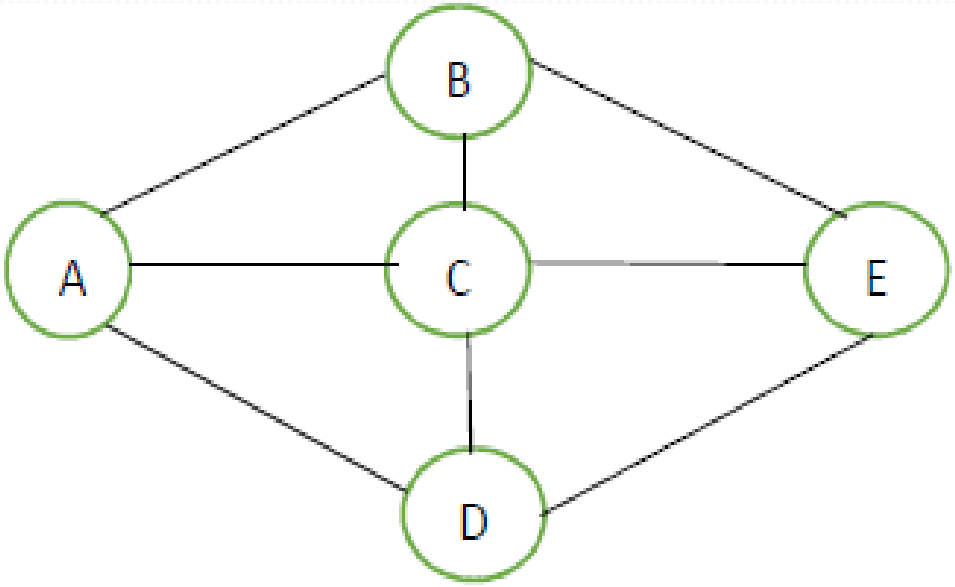
AKTU Examination Questions

1. Explain application of graph coloring problem.
2. Solve the Subset sum problem using Backtracking, where

   n=4, m=18, w[4] = {5, 10, 8, 13}
3. Define Graph Coloring.
4. What is backtracking? Discuss sum of subset problem with the help of an example.
5. Explain Implicit and Explicit constraints of N-queen Problem.

AKTU Examination Questions

6. What is the difference between Backtracking and Branch & Bound? Write Pseudo code for Subset Sum Problem using Backtracking. Give example for the same.
7. Consider a graph G=(V,E). We have to find a Hamiltonian cycle using backtracking method.



Thank you.