Backtracking Algorithms

SMITA SANKHE
smitasankhe@somaiya.edu

Problem characteristics

- Set of solutions
- Constraint satisfaction problem

Solution

- A N-tuple that satisfies some criterion function
- Backtracking gives all answers
- Builds the solution by adding one component at a time and test against criterion function if the solution vector has any chances of success
- Major advantage:- if a partial vector in no way leads to an optimal solution, then rest of the possible test vector can be ignored entirely.

Conditions and solution space

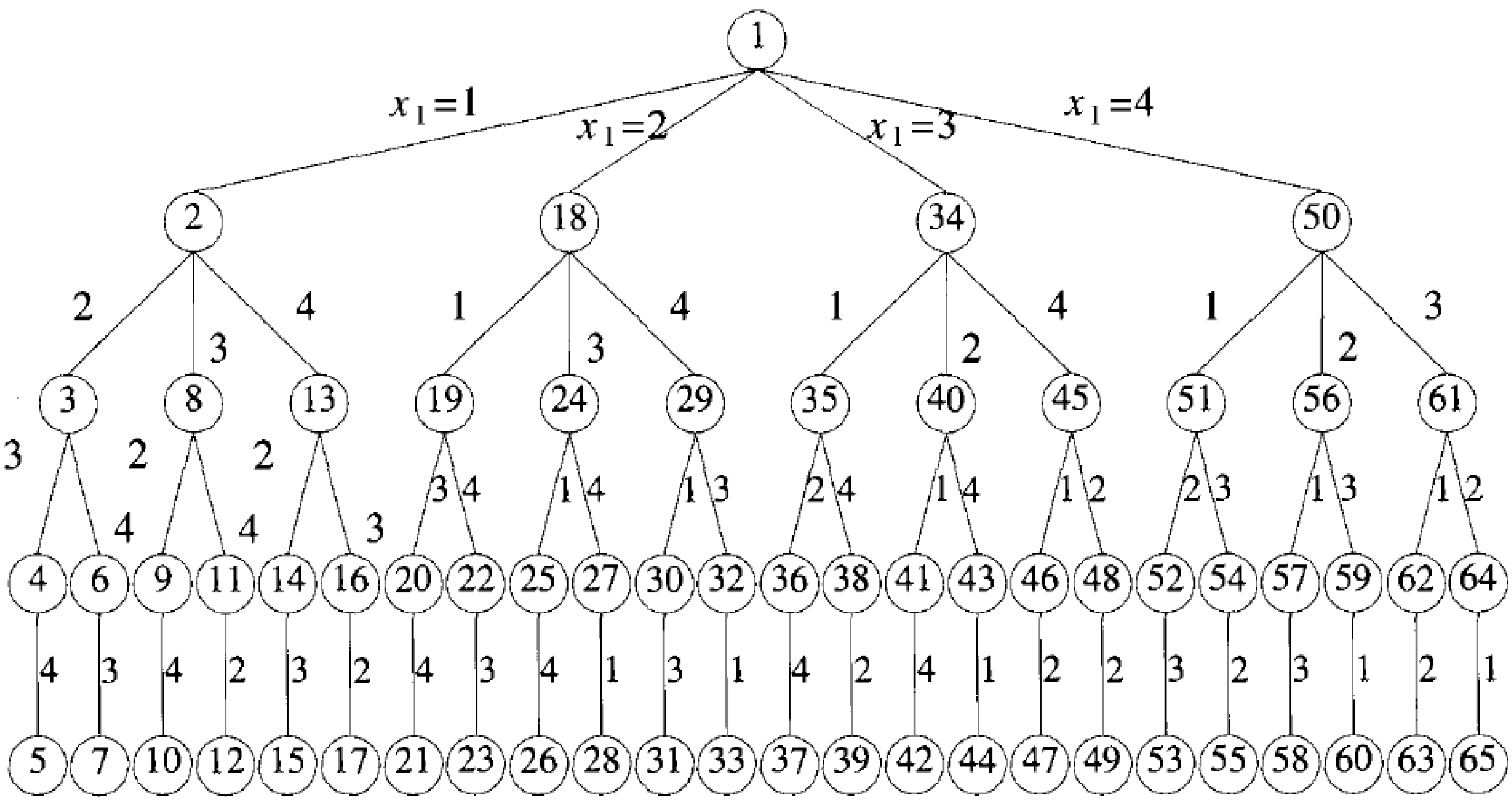
If the solution vector is expressed as a tuple (x1,x2,…..xn) of n problem elements,

- Explicit condition- Rules that restrict each xi to take on values only from a given set.
- E.g. xi= 0 or xi=1 in 0/1 knapsack, xi>0, 1<=i<=n
- Solution space- all tuples those satisfy the explicit constraints define a possible solution space for particular instance of problem being solved.
- Implicit condition- rules that determine which of the tuples in the solution space of problem instance satisfy the criterion function.
- E.g. in N-queen’s problem no two queens can attack in same row, column, diagonal, the knapsack capacity for knapsack problem etc

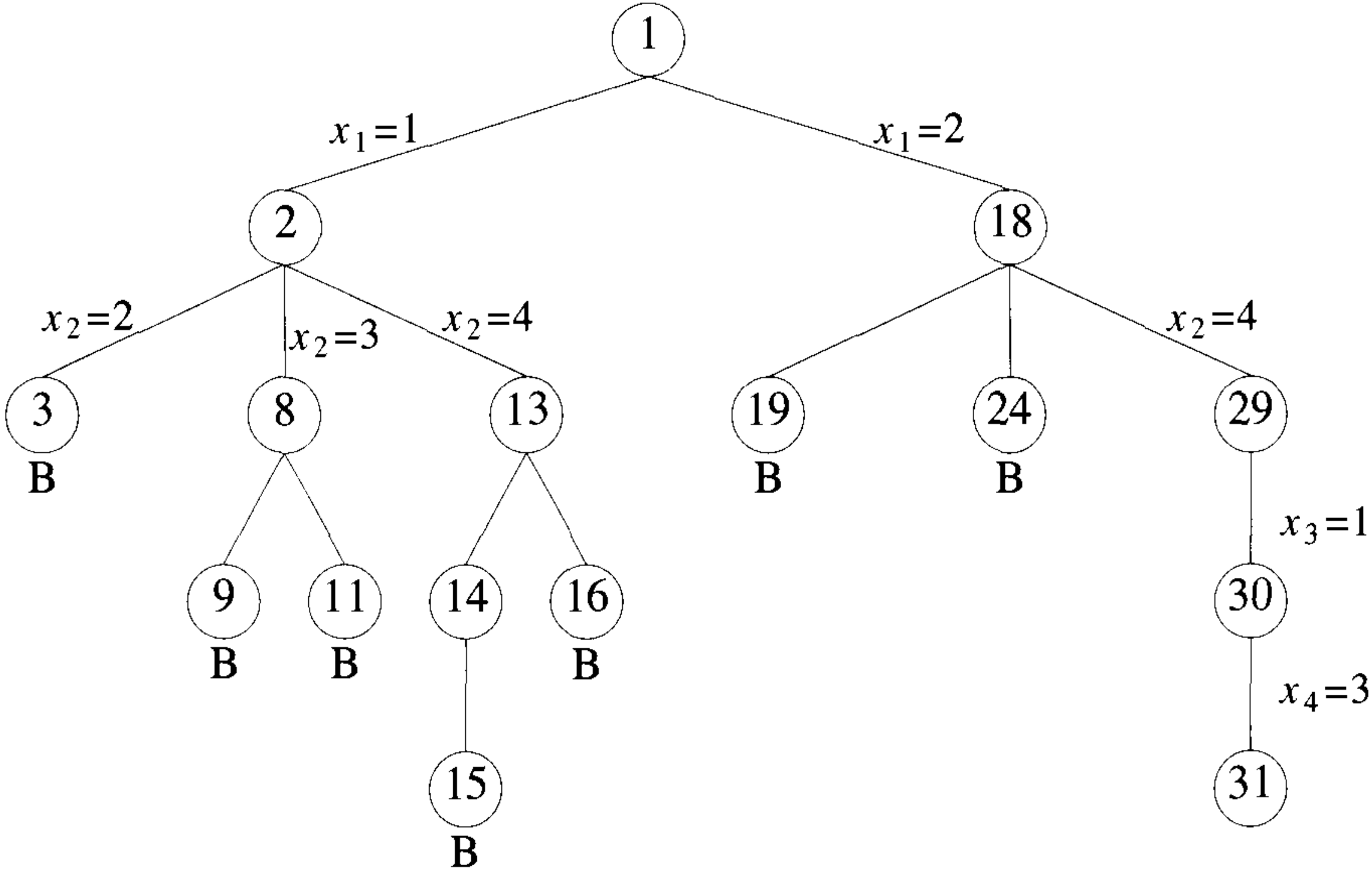
Tree organization of solution space

- Problem state – each node in solution space tree
- Solution states- those problem states s for which path from root to s defines a tuple in solution space (all tuples those satisfy explicit condition)
- Answer states – solution states s for which the path from root to s defines a tuple that is member of the set of solutions. (All tuples those Satisfy the implicit condition)
- State space/solution space tree – set of all legal states in the solution space.

State space/solution space tree for 4-Queen Problem



Partial backtracking tree



Answer states:

For N=4,

- S= {2,4,1,3}
- S={3,1,4,2}

```
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);
        }
    }
}
```

NQueens problem

- **Problem definition:-** The **N queens puzzle** is the problem of placing eight queens on an NxN board such that no two queens attack each other in the same row, column, or diagonal.

N Queen's Problem

- Problem:-To place N Queens on NXN board subject to :-
- Conditions
  - Explicit : Tuple of N elements

  X=(x1,x2,…..xn)
  - Implicit
- Backtrack if-(Assupmtion- Q_i is put in Row_i)

$$\textbf{if } ((x[j] = i) \text{ // Two in the same column}$$
$$\textbf{or } (\mathsf{Abs}(x[j] - i) = \mathsf{Abs}(j - k)))$$
$$\text{// or in the same diagonal}$$

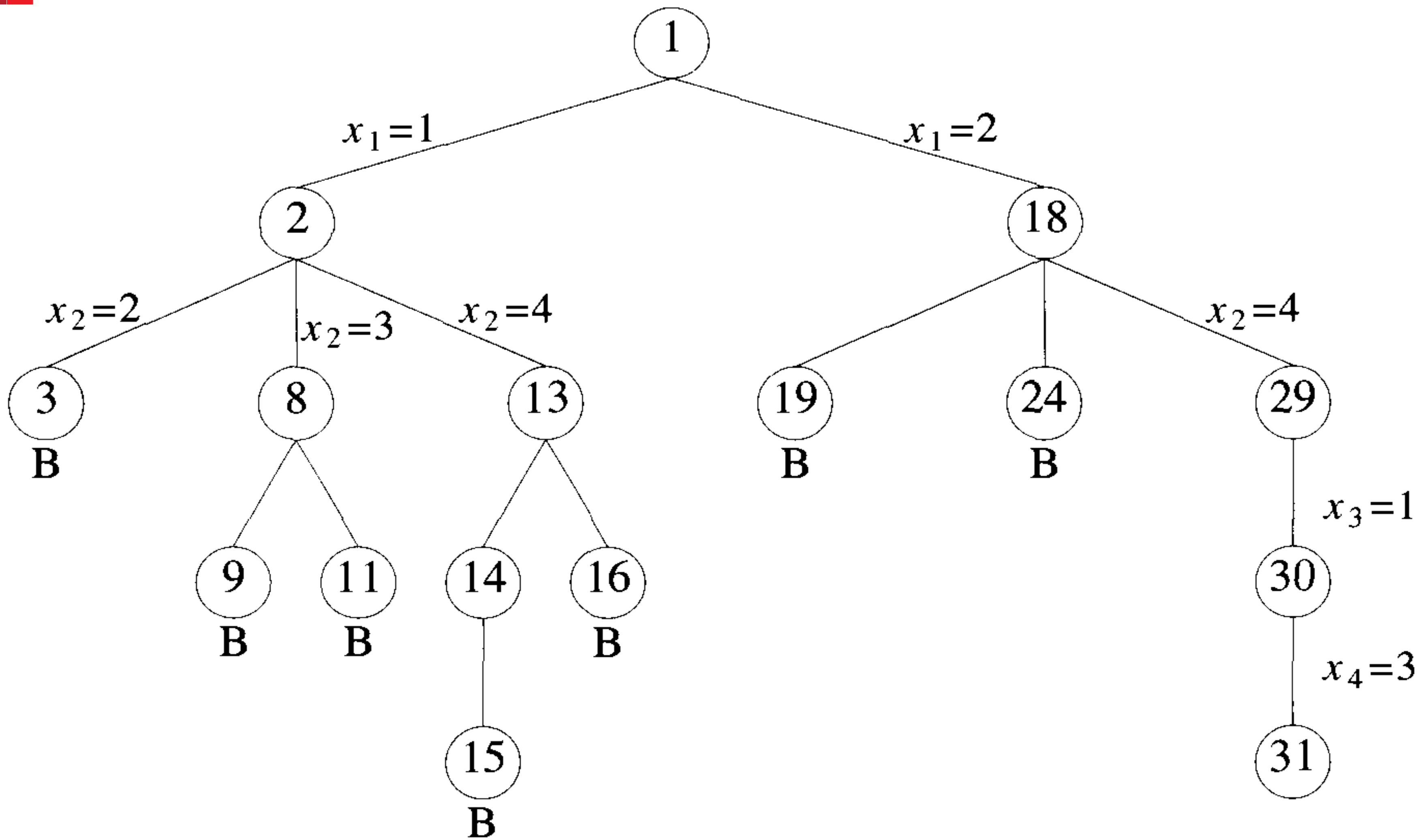
Can a new queen be placed?

```
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;
}
```

All Solutions to N Queen's 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);
        }
    }
}
```

Partial Tree



Graph coloring problem

- **Definition**: A coloring of a graph G=(V,E) is a mapping F:V → C where C is a finite set of colors such that if <v,w> is an element of E then F(v) is different from F(w); in other words, adjacent vertices are not assigned the same color.
- **Conditions**:
  - <u>Explicit Condition</u>: A vector X={x1,x2…Xn} for all n vertices for all possible combinations of colors
  - <u>Implicit Condition:</u> No two adjacent vertices or regions have the same color.
  - <u>Backtracking Condition</u>: If ((k,i) is an edge) and (Color[i]=Color[k]) then Backtrack!

Terms

- Chromatic number- smallest integer m for which G can be colored

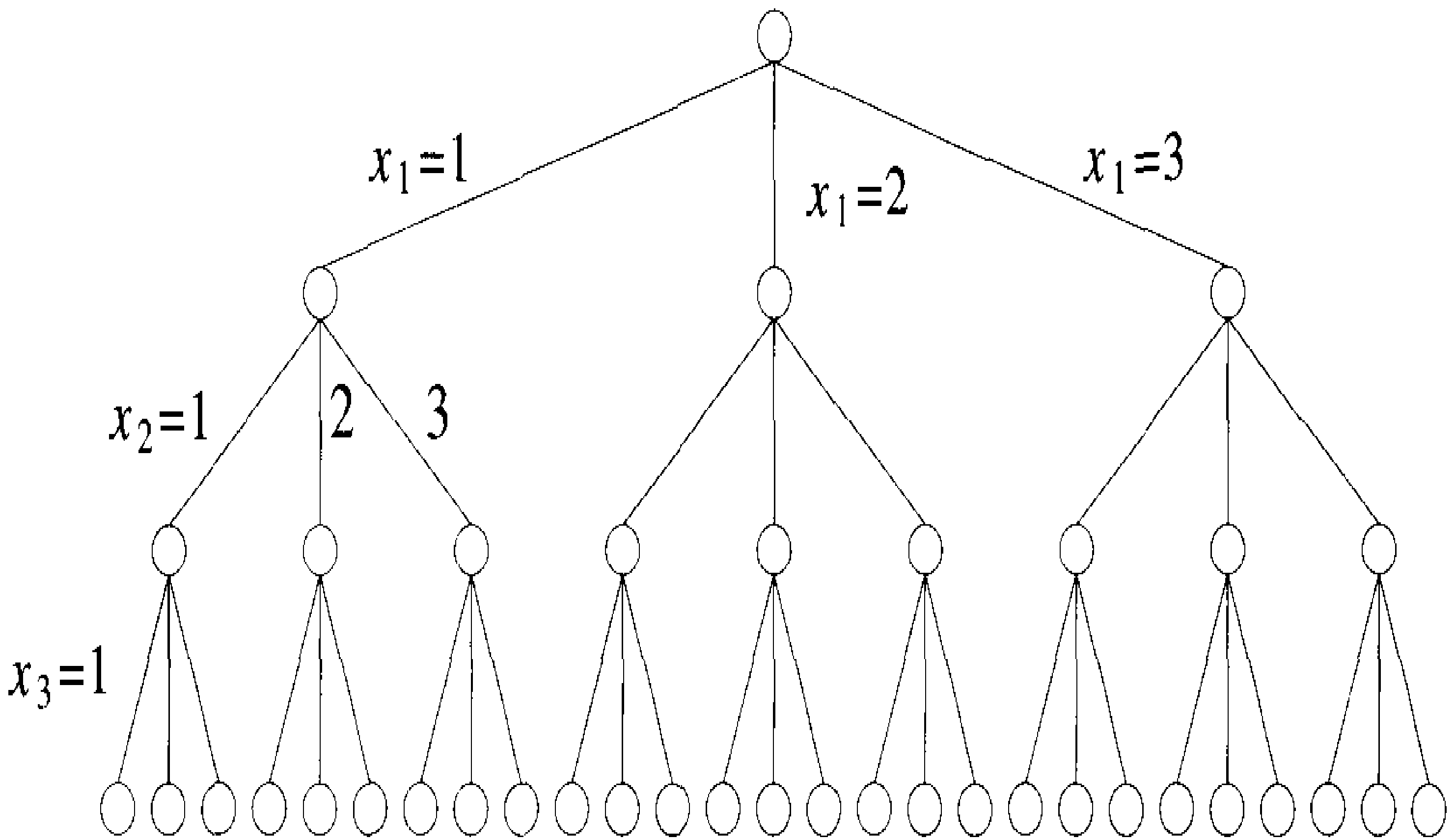
Graph Coloring Problem

```
Algorithm mColoring(k)
// This algorithm was formed using the recursive backtracking
// schema. The graph is represented by its boolean adjacency
// matrix G[1 : n, 1 : n]. All assignments of 1, 2, ..., m to the
// vertices of the graph such that adjacent vertices are
// assigned distinct integers are printed. k is the index
// of the next vertex to color.
{
    repeat
    {// Generate all legal assignments for x[k].
        NextValue(k); // Assign to x[k] a legal color.
        if (x[k] = 0) then return; // No new color possible
        if (k = n) then      // At most m colors have been
                             // used to color the n vertices.
            write (x[1 : n]);
        else mColoring(k + 1);
    } until (false);
}
```

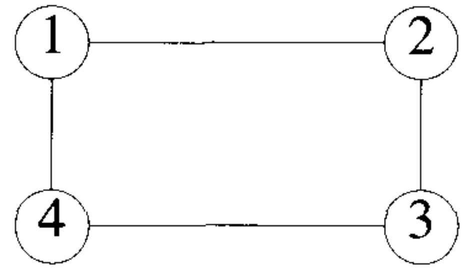
Graph Coloring Problem

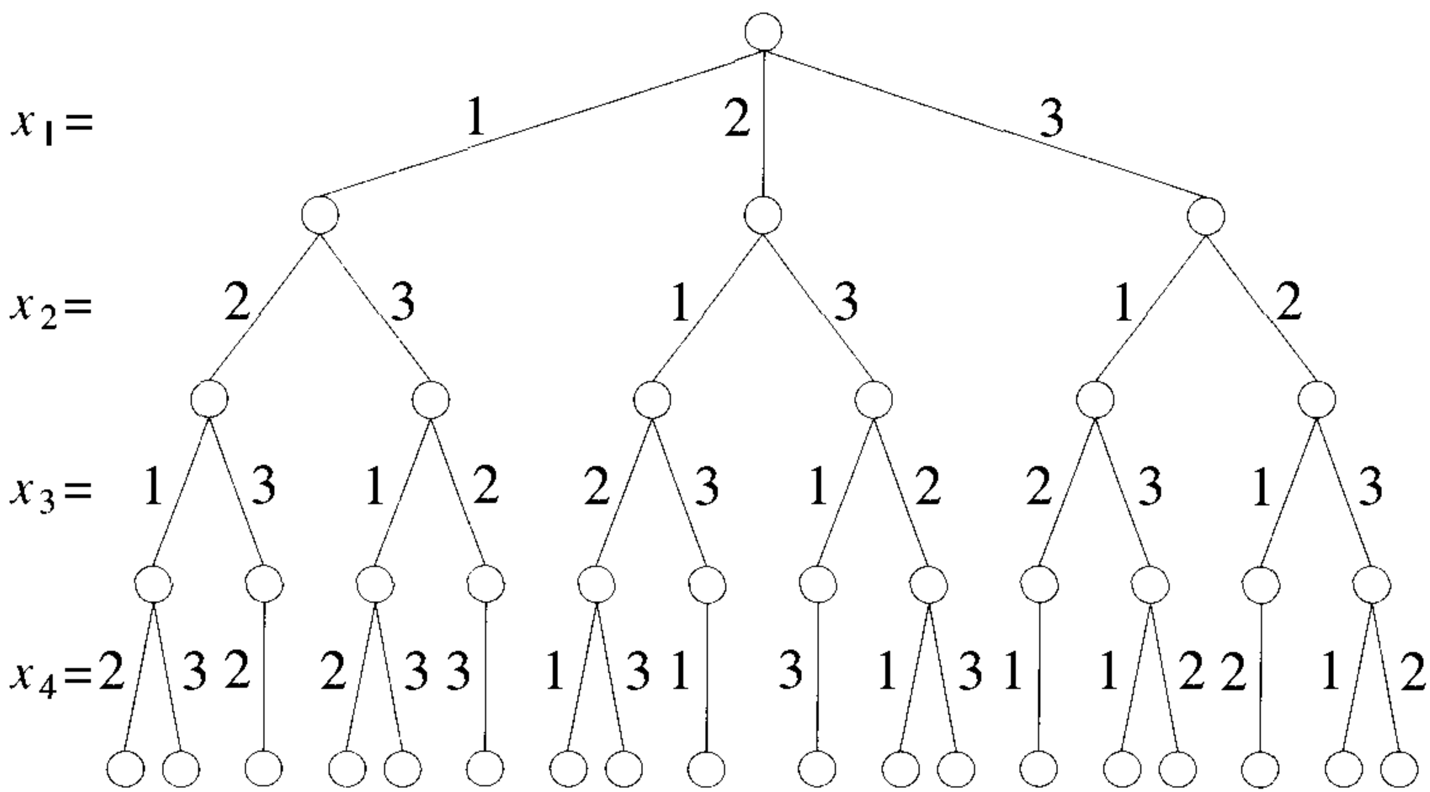
```
Algorithm NextValue(k)
// x[1], ..., x[k-1] have been assigned integer values in
// the range [1, m] such that adjacent vertices have distinct
// integers. A value for x[k] is determined in the range
// [0, m]. x[k] is assigned the next highest numbered color
// while maintaining distinctness from the adjacent vertices
// of vertex k. If no such color exists, then x[k] is 0.
{
    repeat
    {
        x[k] := (x[k] + 1) mod (m + 1); // Next highest color.
        if (x[k] = 0) then return; // All colors have been used.
        for j := 1 to n do
        {   // Check if this color is
            // distinct from adjacent colors.
            if ((G[k, j] ≠ 0) and (x[k] = x[j]))
            // If (k, j) is and edge and if adj.
            // vertices have the same color.
                then break;
        }
        if (j = n + 1) then return; // New color found
    } until (false); // Otherwise try to find another color.
}
```

State space tree for mColoring when n =3 and m = 3

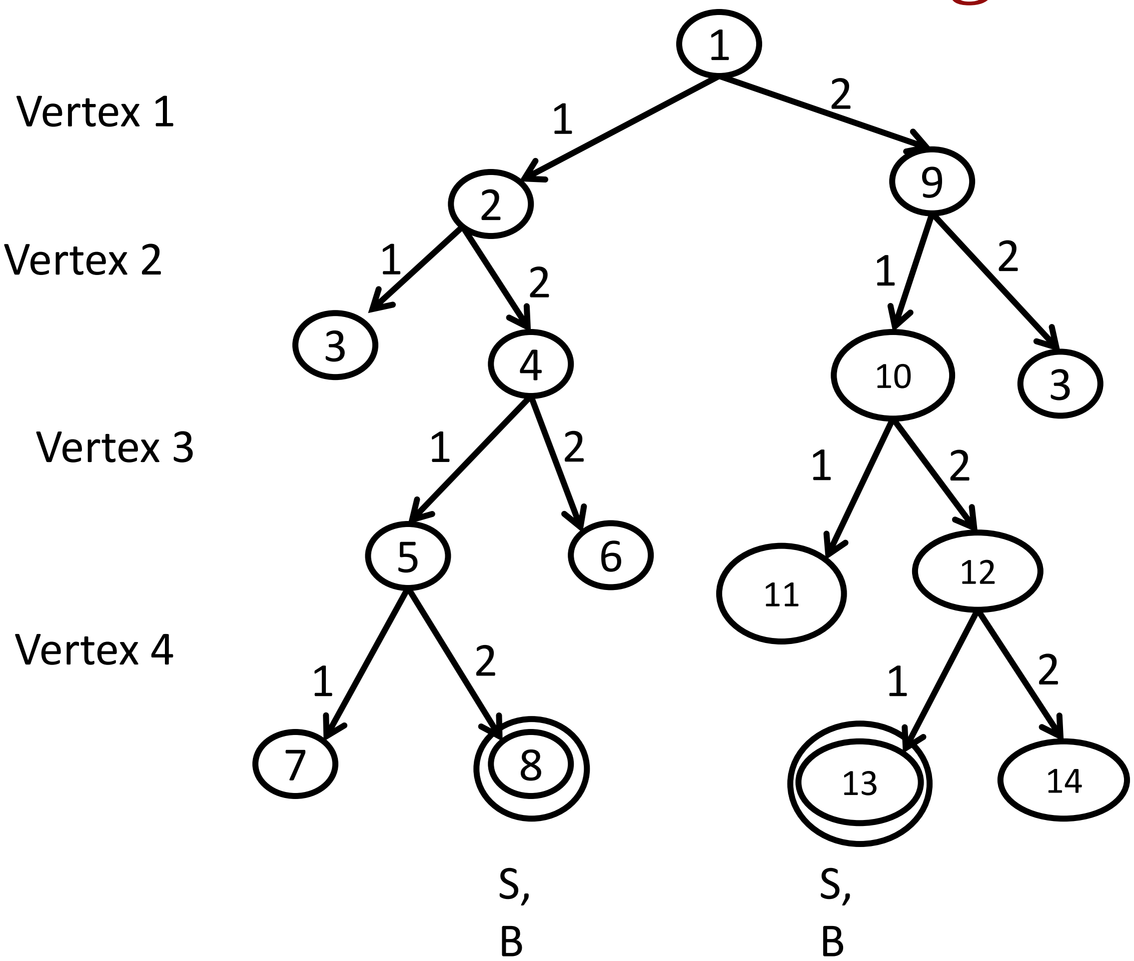


Solution space tree
for n=4, m=3



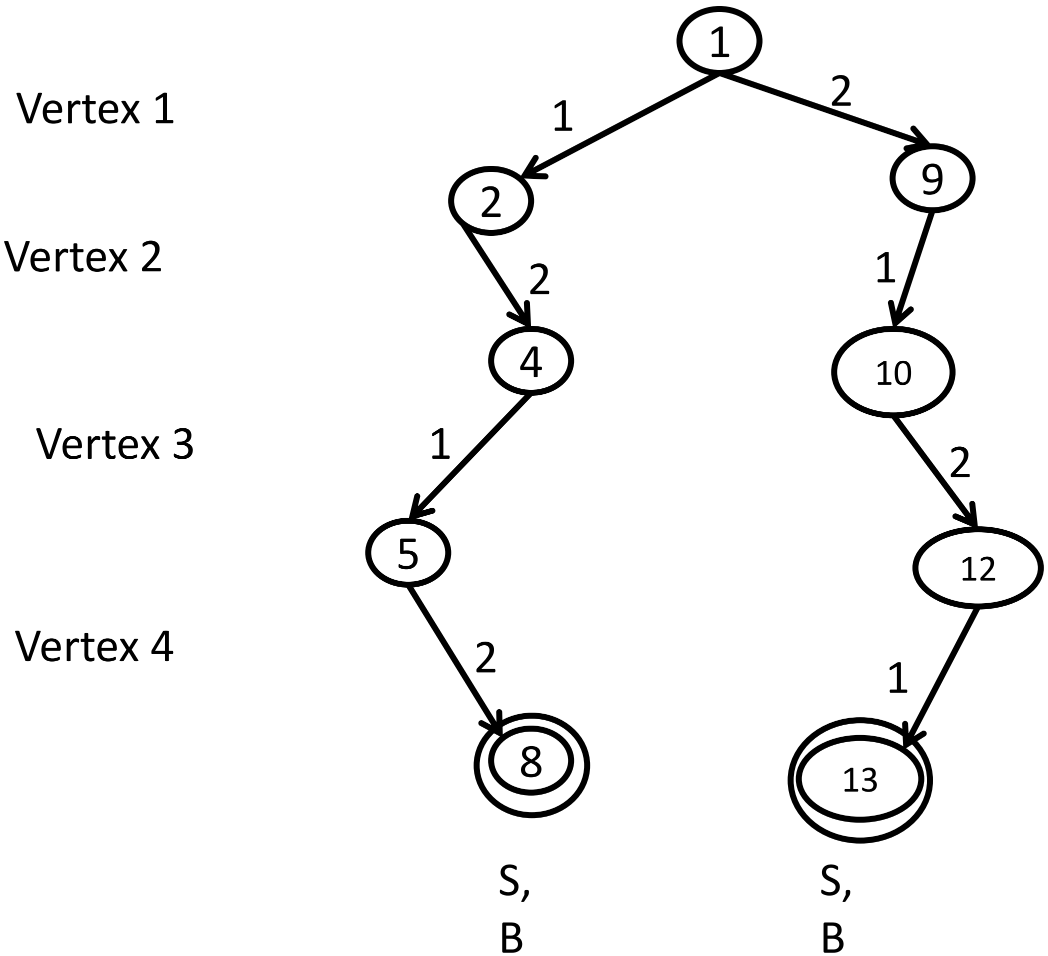


Backtracking tree



for n=4, m=2

Solution tree



Q.3

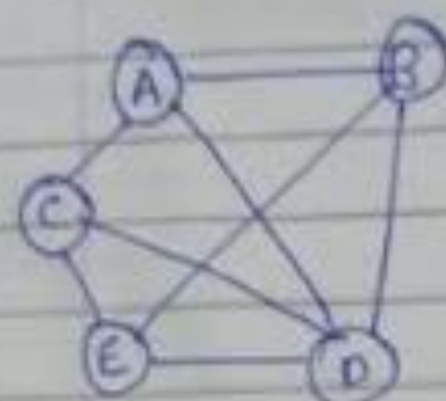
→ Define the problem (01 M)

→ Define the explicit, Implicit and Backtracking conditions (01 M)

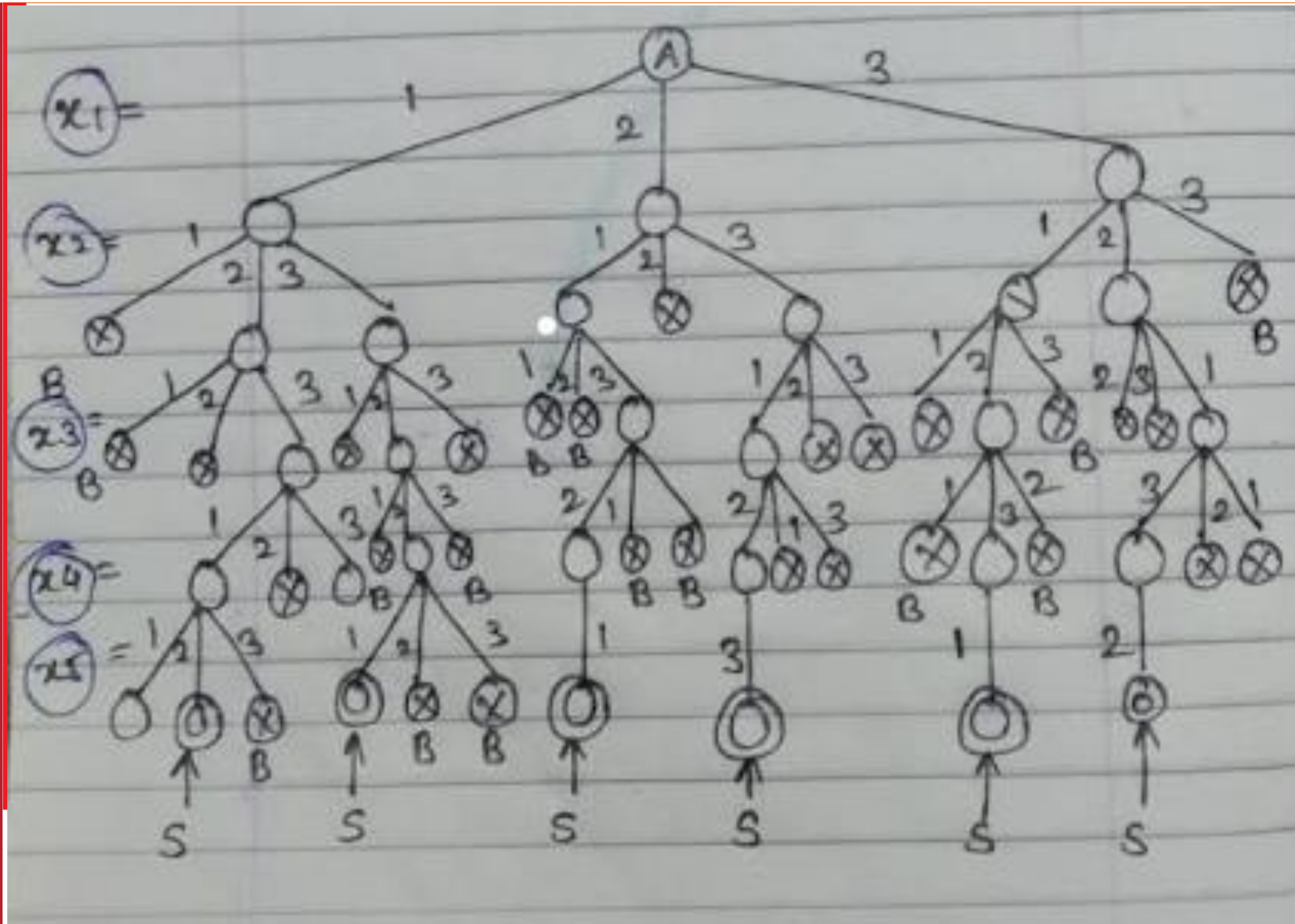
→ Draw the Backtracking tree (04 M)

→ Draw the solution tree (02 M)

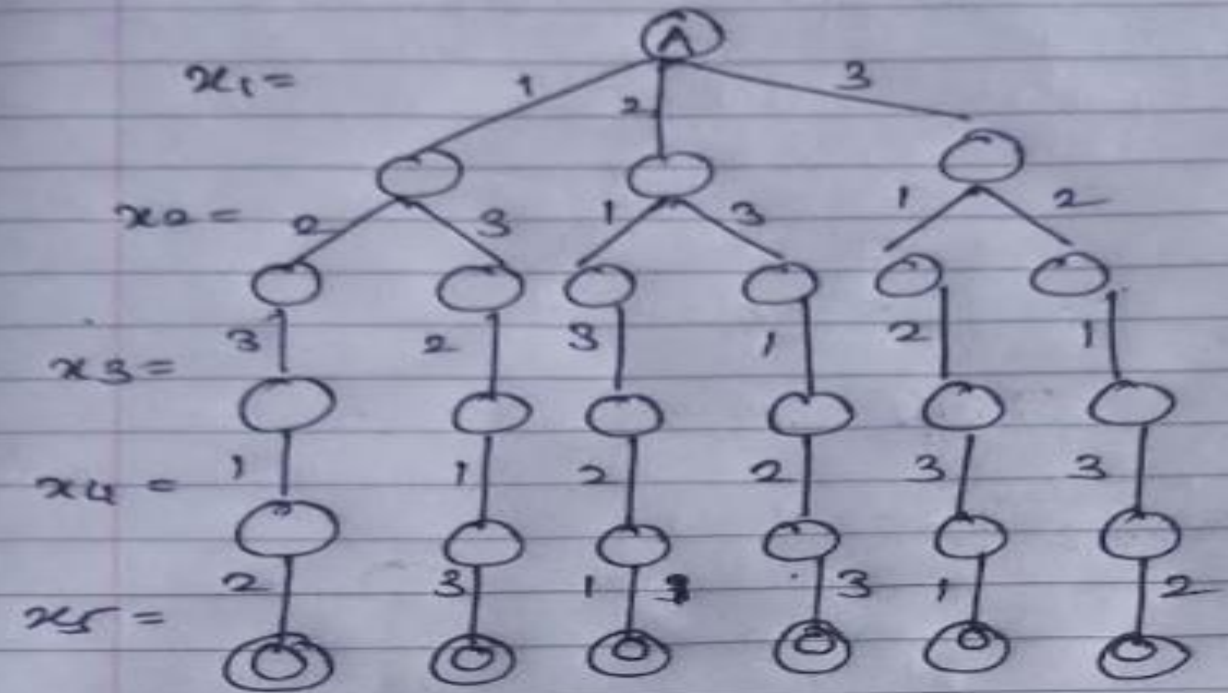
→ List all possible solutions (02 M)



(A, B, C, D, E) → chromatic Number = 03


A
x1 =
x2 =
x3 =
x4 =
x5 =
B
S

Solution tree :-



$(A, B, C, D, E) \rightarrow (x_1, x_2, x_3, x_4, x_5)$

All possible soln :-

$\{ (1,2,3,1,2), (1,3,2,1,3)$
$(2,1,3,2,1), (2,3,1,2,3)$
$(3,1,2,3,1), (3,2,1,3,2) \}$

Sum of subsets problem

Problem definition:

- Find a subset of a given set A = {a1, . . . , an} of n positive integers whose sum is equal to a given positive integer d.

For example, for A = {1, 2, 5, 6, 8} and d = 9, there are two solutions: {1, 2, 6} and {1, 8}. Of course, some instances of this problem may have no solutions.

Explicit and Implicit Constraints

- **Explicit constraints:**
  - **I) xi >= 0;**
  - **II) xi ∈ {j | j is an integer and $1 \le j \le n$}**
- **Implicit constraint:** determines which of the tuples in the solution space I can actually satisfy the criterion functions.
  - **I) No two xi can be the same**
  - **II) $\Sigma w_{xi} = m$**
  - **III) $x_i < x_{i+1}$, $1 \le i < k$ (total order in indices) · Helps in avoiding the generation of multiple instances of same subset; (1, 2, 4) and (1, 4, 2) are the same subset.** By **sorting the initial array**, **we need not to consider rest of the array, once the sum so far is greater than target number.** We can backtrack and check other possibilities.

Bounding Functions/ Backtracking Condition:

I)

$\sum_{i=1}^{k} W_i x_i + W_{k+1} \leq m$ ←*Total value by adding next item in list should be less than m*

II) $\sum_{i=1}^{k} W_i x_i + \sum_{i=k+1}^{m} W_i \geq m$ ←***Sum of items present in list and items left out should be more than m***

Ex:- n=6, m=30, w [1:6]= { 5,10,12,13,15,18 }
Portion of state space tree generated by SumOfSub.
circular nodes indicate subsets with sums equal to m.

-