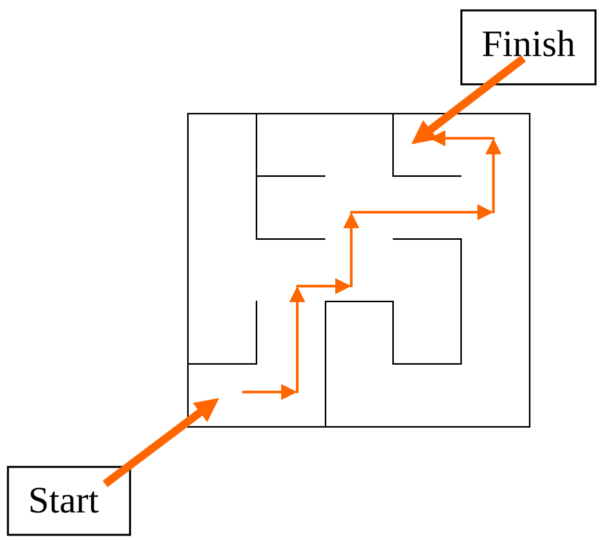
Backtracking

➥ Examples:

- Maze problem



- The bicycle lock problem:

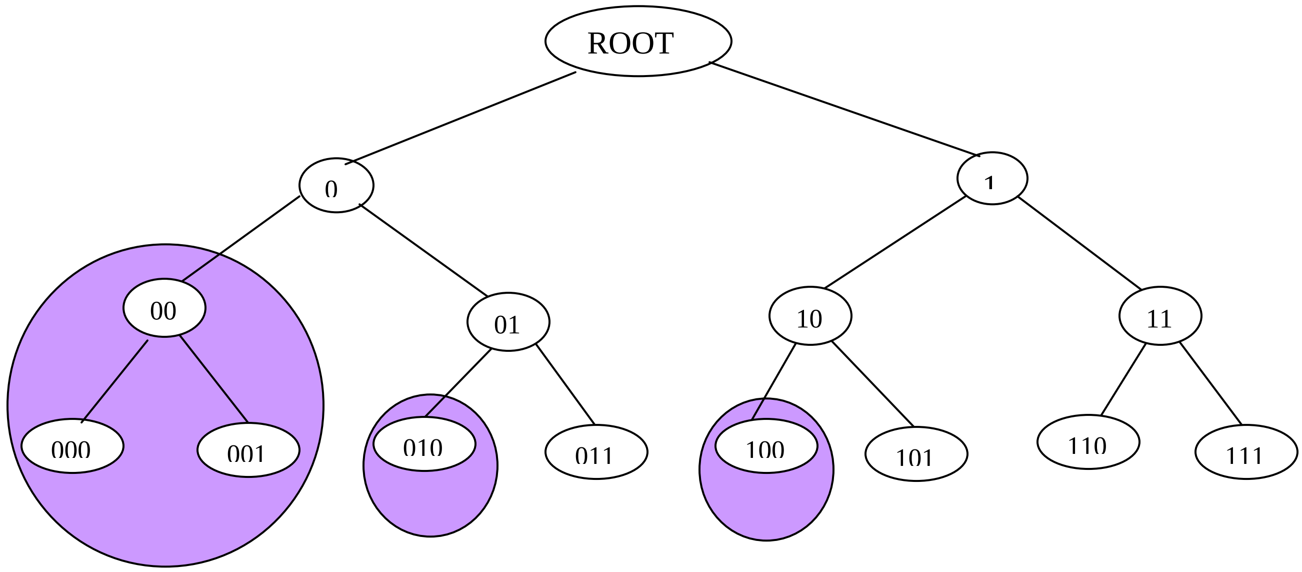
  ✓ Consider a lock with N switches, each of which can be either 0 or 1.

  ✓ We know that the combination that opens the lock should have at least $\left\lceil \frac{N}{2} \right\rceil$ 1's.

  ✓ Note: The total number of combinations is 2^N

  ✓ The solution space can be modeled by a tree

✓ Example: N=3



➯ Characteristics

- Backtracking technique can be considered as an organized exhaustive search that often avoids searching all possibilities.
- The solution space can be organized as a tree called: search tree
- Use depth-first search technique
- The search tree is pruned using a bounding function.
- Assumptions:
    - ✓ X[1..n] contains the solution of the problem
    - ✓ All possible values of X[i]are elements of a set S_i

- General algorithm:

```
Procedure backtrack(n)
/* X is the solution vector */
Integer k;
Begin
    k =1;
    Compute Sk; /* compute the possible solution values for k=1 */
    While k> 0 do
        While Sk <> φ do
            X[k] = an element of Sk;
            Sk = Sk -{X[k]};
            If B(X[1], …, X[i],…, X[k]) = True
            Then
                Print the solution vector X;
            else begin
                    k = k+1;
                    Compute Sk;
                End;
            End;
        End while;
        k = k-1;
    End while;
End;
```

- Recursive solution:

```
Procedure back_recursive(k)
begin
    For each X[k] in Sk do
        If B(X[1], …, X[i],…, X[k]) = True
        then  Print the solution vector X;
        else begin
                Compute Sk;
                Back_recursive(k+1);
        end if;
    end for;
end;
```

↳Examples:

- n-queen
- sum of subsets
- Hamiltonian cycle
- Graph coloring

➯ n-queen problem:

- Objective: place n queen in an n by n chessboard such that no two queens are not on:

  1) same row
  2) same column
  3) same diagonal
  4)

  1), 2), and 3) form the **bounding function**

- Example: N=4

~~Q~~	~~Q~~	Q	
Q		~~Q~~	~~Q~~
	~~Q~~		Q
	Q		

- Brute force method or exhaustive search: takes $O(\binom{n^2}{n})$

- Using condition 2) ➔ reduce the solution space to n^n

- Using condition 1) ➔ reduce the solution space to n!

- The solution vector is characterized as follows:

  - ✓ X[I] contains the column position of the queen i in the ith row.
  - ✓ $S_k = \{1,2,\ldots, n\}$ S_k represents the number of columns for queen k in the kth row.

- Algorithm:

```
Function bound(k)
Integer i;
Begin
    For i=1 to k-1 do     /* for each row up to k-1 */
        If X[i] = X[k]   /* Are queens on the same row?*/
          or
          |X[i]-X[k]| = |i-k|  /* Are queens are on the same
                                  diagonal */
        then
            return (False);
        endif;
    endfor;
    return(True);
end;
```

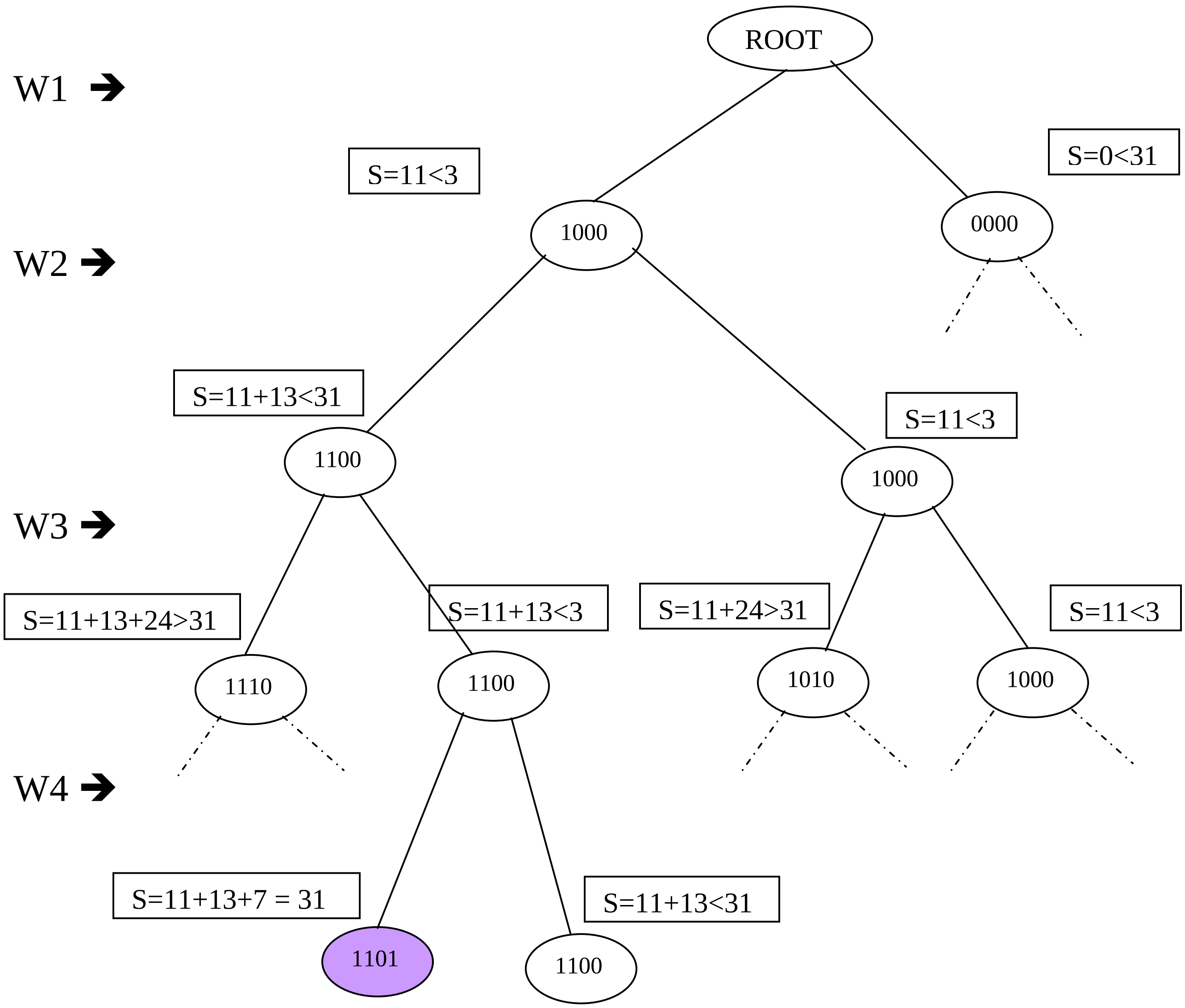
⇘ Sum of subsets

- Input: n distinct positive numbers w_i where 1<=I<=n and integer M
- Output: all combinations whose sum is M

- Solution:
  - ✓ Use static binary tree where level I corresponds to the selection of w_i.
  - ✓ The solution vector X is defined as follows:

    It a bit-map vector where X[i] contains 1 if the w_i is included; otherwise it contains 0;

- Example: n=4
  (w1,w2,w3,w4) = (11,13,24,7)
  M = 31



Bounding function:

Function bound(k)

Begin

If $\left(\sum_{i=1}^{k} X[i]w_i + \sum_{i=k+1}^{n} w_i \ge M\right)$ and $\left(\sum_{i=1}^{k} X[i]w_i \le M\right)$

Then

Return(True);

Else

Return(False);

Endif;

End;

➪ Hamiltonian Cycle

- Definition: A Hamiltonian cycle in an undirected graph G=(V,E) is a simple cycle that passes through every vertex once.

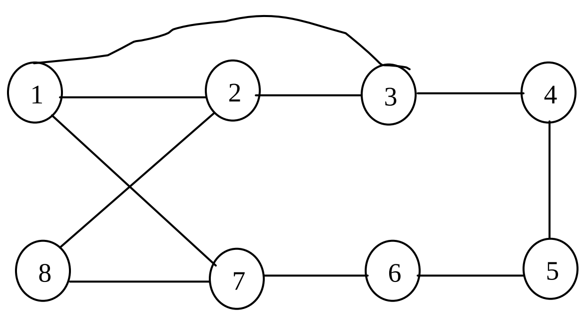
- Input: a graph G with n vertices
- Output: Hamiltonian cycle

- Solution:

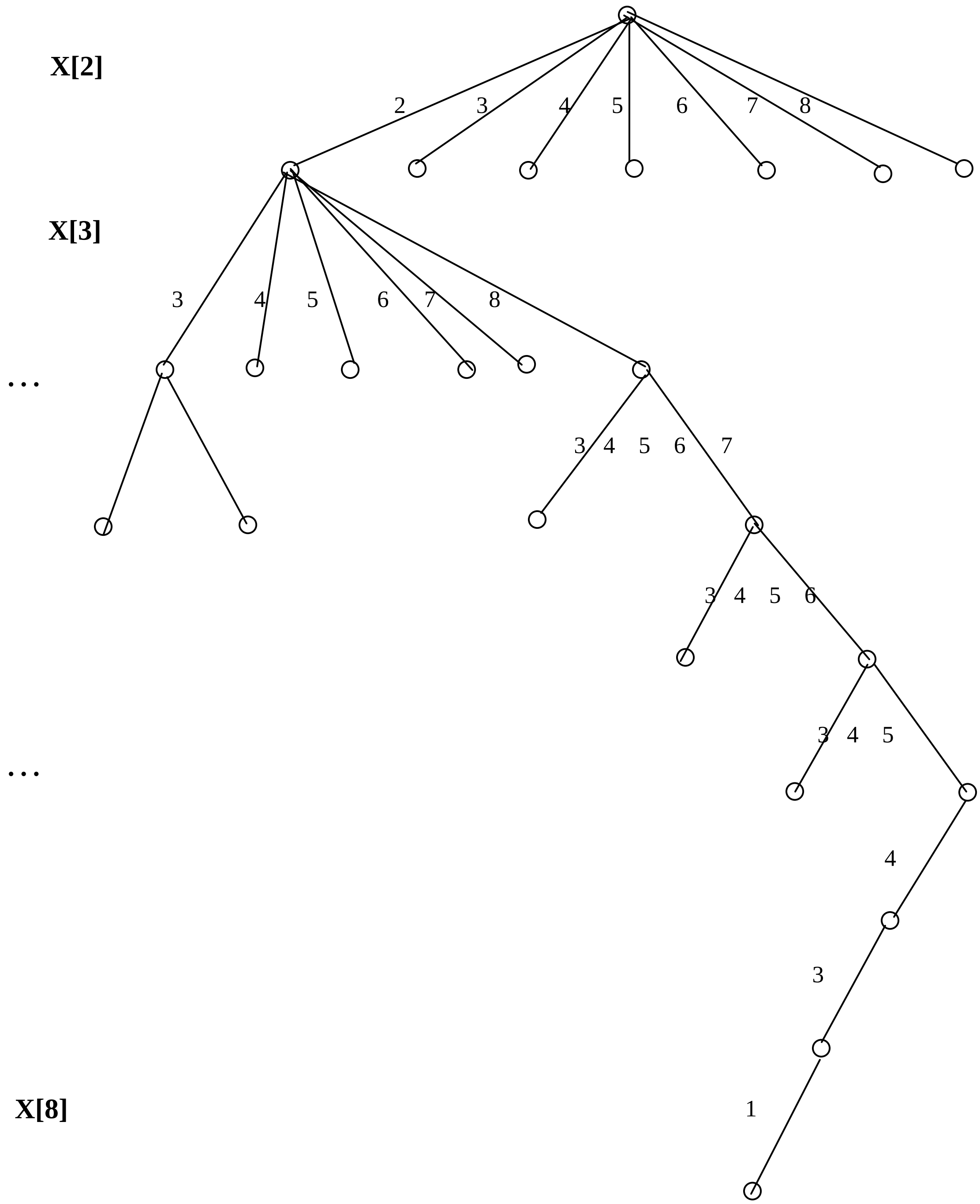
  ✓ Use dynamic tree where level i corresponds to the selection of the ith vertex.

  ✓ The solution vector X[1..n] is such that X[I] is the vertex in the cycle.

- Example:



  ✓ Start at vertex 1


X[2]
2 3 4 5 6 7 8
X[3]
3 4 5 6 7 8
3 4 5 6 7
3 4 5 6
3 4 5
4
3
X[8]
1

- Bounding function:

```
Function bound(k)
integer i;
begin
    for i=1 to k-1 do        /* check for distinctness */
        If x[i]=x[k]
        then
            return(False);
        endif;
    endfor;
    If (X[k],X[k]-1) is an edge in G
    then
        return (True);
    else
        return(False);
    endif;
end;
```

➯ Graph coloring:

- 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.

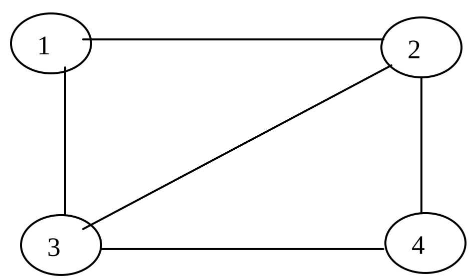
- Input: Graph G of n vertices and a set of m colors in C={1,2,…,m}

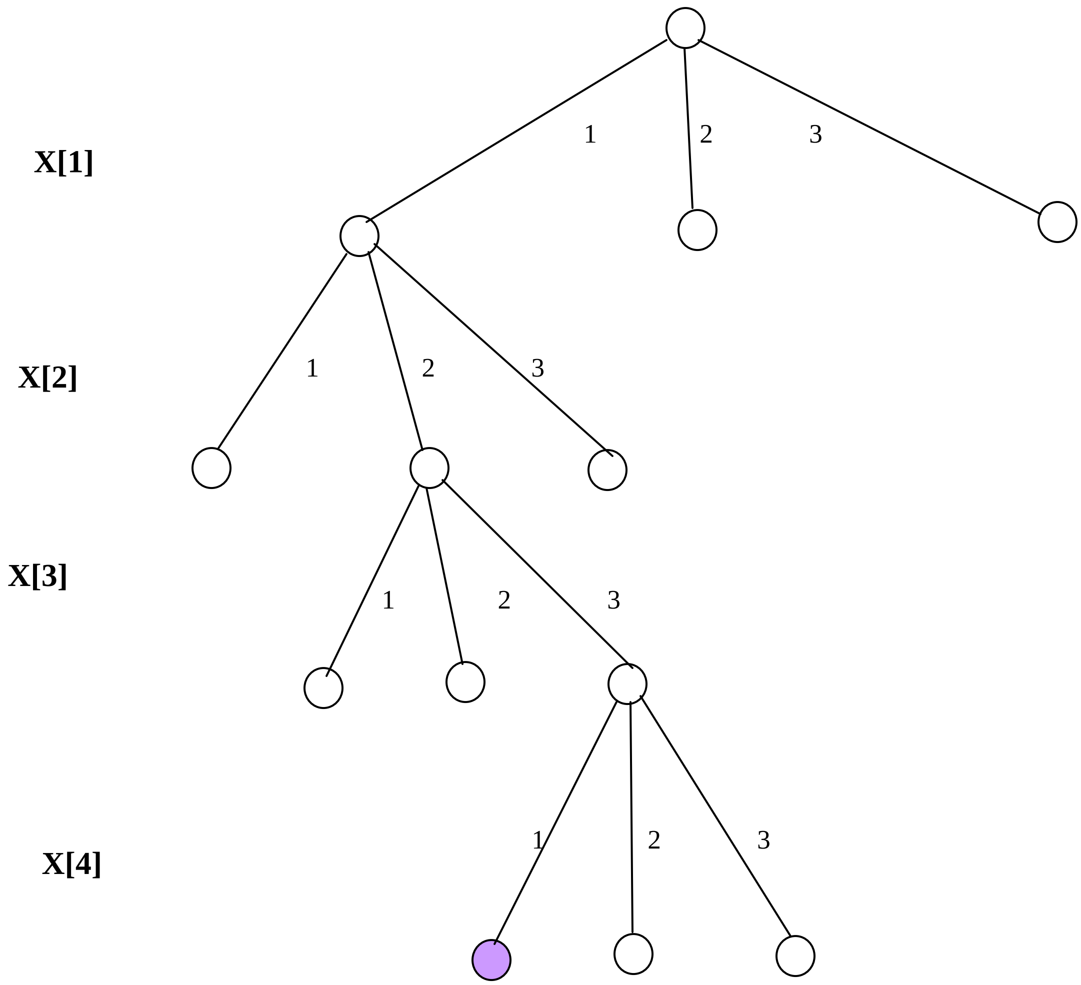
- Output: - color the vertices of G such that no two adjacent vertices have the same color

- Solution:

  - ✓ Use a static tree
  - ✓ The solution vector X[1..n] is such X[i has the color of the ith node.

- Example: n=4 and m=3





- Bounding function:

```
Function next_color(k)
Integer i;
begin
        for i=1 to k-1 do
                If ((k,i) is an edge) and (X[i]=X[k])
                then
                        return (False);
                endif;
        endfor;
end;
```