Design and Analysis of Algorithm

Backtracking
(Hamiltonian Cycle, Graph Coloring, Sum of subset and N-Queen)

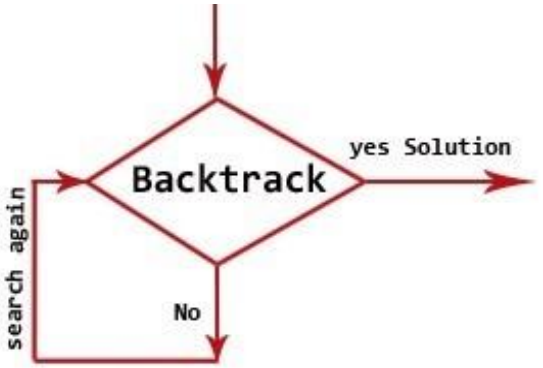
Lecture – 50-53

Overview

- Backtracking is a general algorithmic technique that consider searching every possible combinations in order to solve an optimization problem.
- 'Backtrack' the Word was first introduced by Dr. D.H. Lehmer in 1950s. R.J Walker Was the First man who gave algorithmic description in 1960. Later developed by S. Golamb and L. Baumert.
- Based on DFS search.

Backtracking

- What is Backtracking?
  - Backtracking is nothing but the modified process of the brute force approach. where the technique systematically searches for a solution to a problem among all available options. It does so by assuming that the solutions are represented by vectors (v1, ..., in) of values and by traversing through the domains of the vectors until the solutions is found.



Backtracking

- Algorithmic Approach
  1. Test whether a solution has been found.
  2. If found a solution, then return it
  3. Else for each choice that can be made
     a) Make that choice
     b) Recur
     c) If recurrence returns a solution, return it.
  4. If no choice remain, return failure.

Sometimes called a "Search Tree".

Backtracking

- Problems
  1. Hamiltonian cycle
  2. Graph Coloring
  3. Sum of subset
  4. N-Queen

Backtracking

- Problems
  1. Hamiltonian cycle
  2. Graph Coloring
  3. Sum of subset
  4. N-Queen

Backtracking

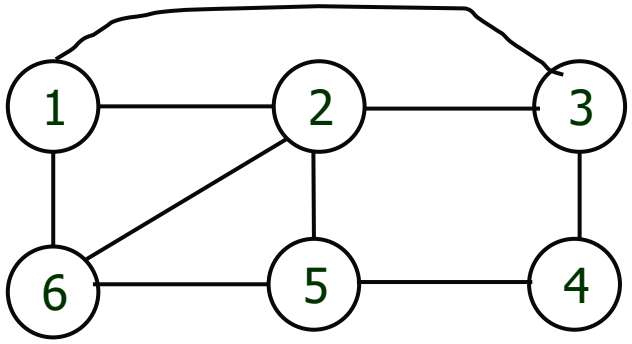
- **Problem 1 : Hamiltonian cycle**
  - Hamiltonian Cycle is a graph theory problem where the graph cycle through a graph can visit each node only once. The puzzle was first devised by Sir William Rowan Hamilton and the Problem is named after him.
  - Let $G = \langle V, E \rangle$ be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once and returns to its starting position.

Backtracking

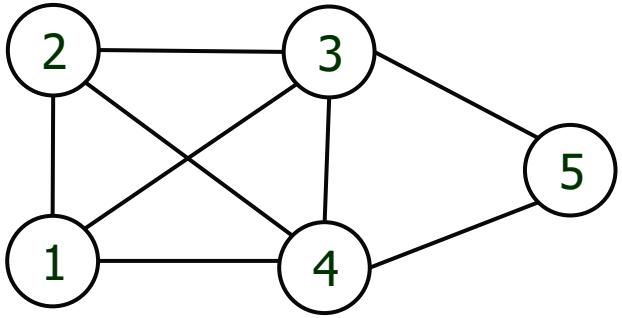
- **Problem 1 : Hamiltonian cycle**
  - In other words a Hamiltonian cycle begins at some vertex $v_1 \in G$ and the vertices of G are visited in the order $v_1, v_2, \ldots\ldots, v_{n+1}$,then the edges (v_1, v_{i+1}) are in $E, 1 \leq i \leq n,$ and the v_i are distinct except for v_1 and v_{n+1}, which are equal.
  - Some Hamiltonian cycle with graph is given in next slide.

Backtracking

- **Problem 1 : Hamiltonian cycle**



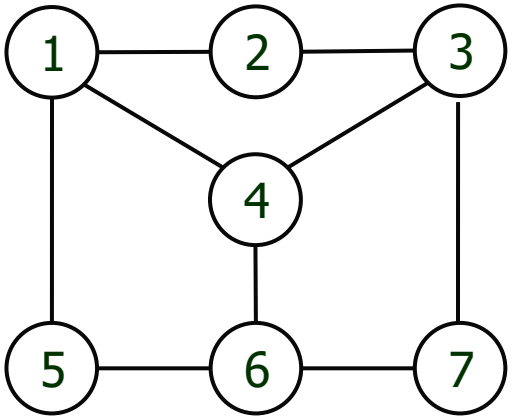
1 2 3 4 5 6 1
1 2 6 5 4 3 1
1 6 2 5 4 3 1



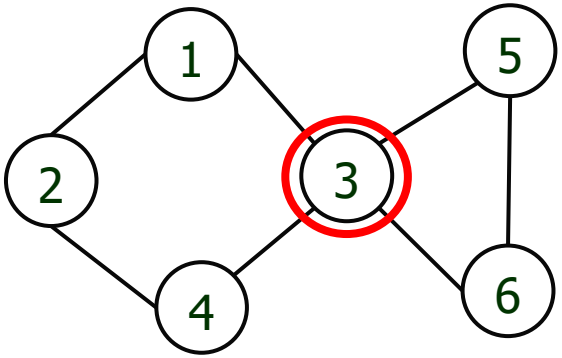
1 2 3 5 4 1
1 2 4 5 3 1

Backtracking

- **Problem 1 : Hamiltonian cycle**



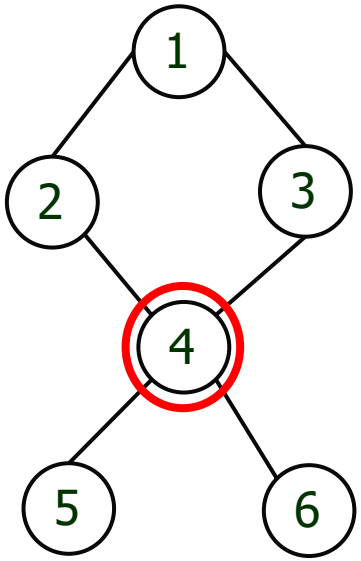
Hamiltonian cycle is not exist in this graph



Hamiltonian cycle is not exist in this graph because of Articulation point vertex.

Backtracking

- **Problem 1 : Hamiltonian cycle**

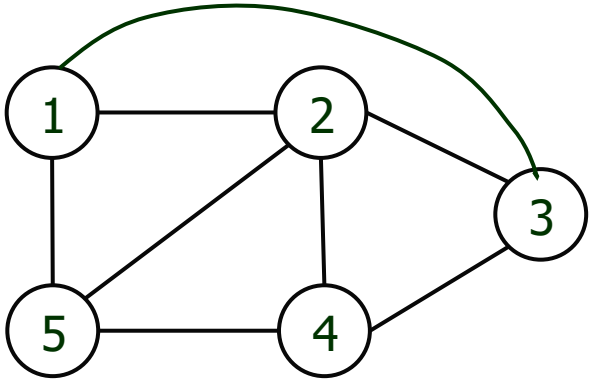


Hamiltonian cycle is not exist in this graph because of Pendent vertex.

Let's find how backtracking is help us for finding the Hamiltonian cycle.

Backtracking

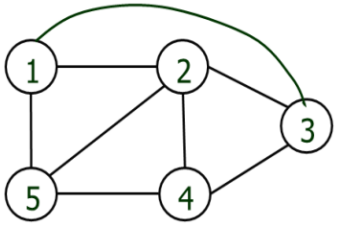
- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



Backtracking

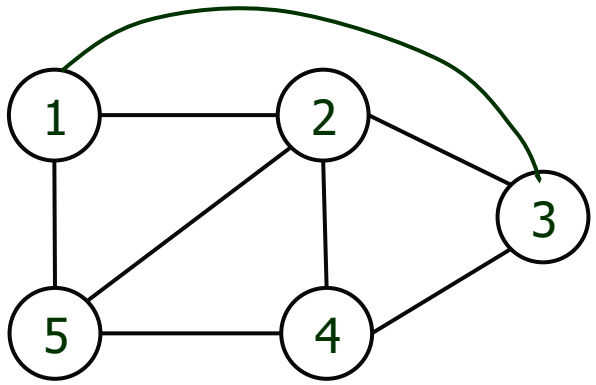
Draw by himself/herself

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking. First find the state space tree.



Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.

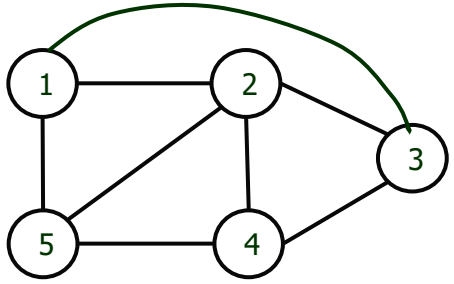


	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

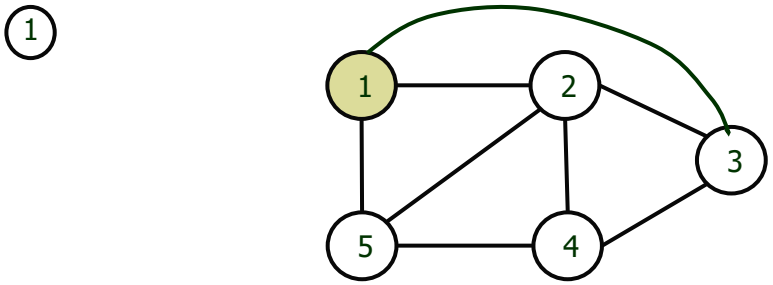
Adjacent Matrix

	1	2	3	4	5
x	0	0	0	0	0

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

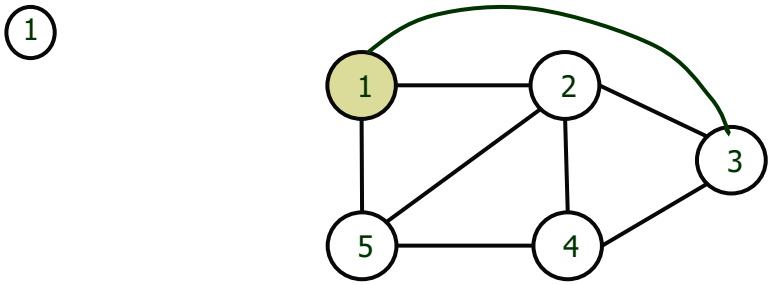
Adjacent Matrix

	1	2	3	4	5
x	1	0	0	0	0

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

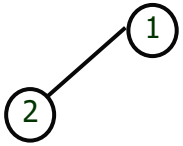
	1	2	3	4	5
x	1	1	0	0	0

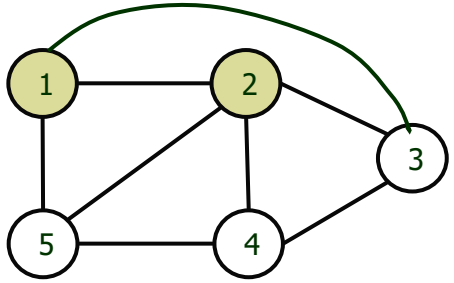
State Space Tree

Node 1 already selected so we can't select it again. So we go for node 2.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

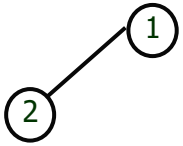
	1	2	3	4	5
x	1	2	0	0	0

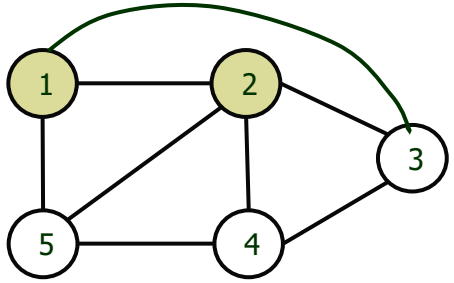
State Space Tree

Node 1 already selected so we can't select it again. So we go for node 2.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

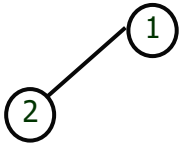
	1	2	3	4	5
x	1	2	0	0	0

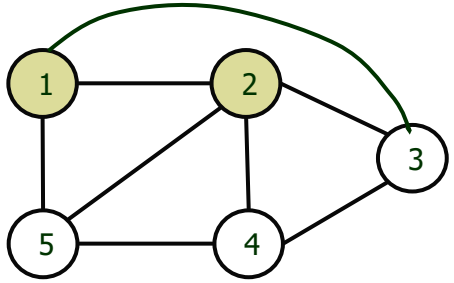
Node 1 already selected so we can't select it again. So we go for node 2.

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

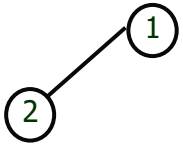
	1	2	3	4	5
x	1	2	1	0	0

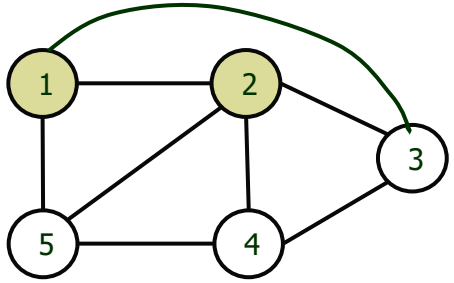
Node 1 already selected so we can't select it again. So we go for node 2.

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

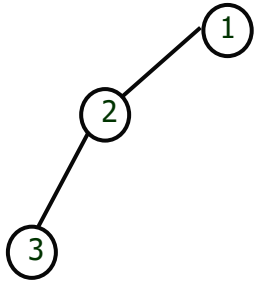
	1	2	3	4	5
x	1	2	~~2~~	0	0

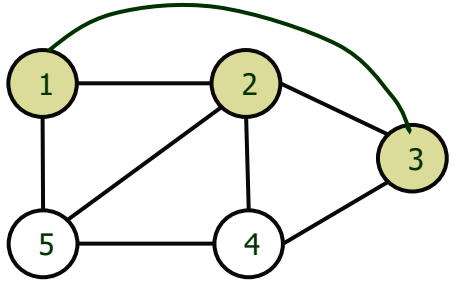
Node 2 already selected so we can't select it again. So we go for node 3.

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

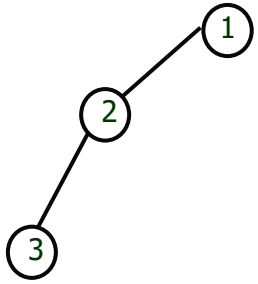
	1	2	3	4	5
x	1	2	3	0	0

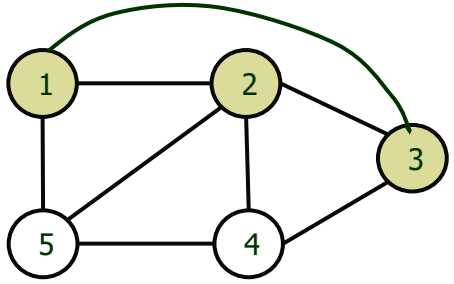
State Space Tree

Node 2 already selected so we can't select it again. So we go for node 3.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

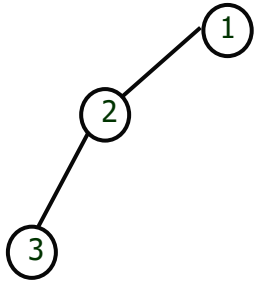
	1	2	3	4	5
x	1	2	3	0	0

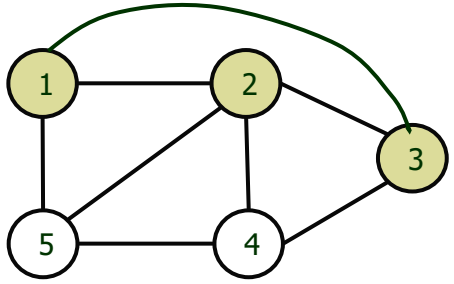
State Space Tree

Node 2 already selected so we can't select it again. So we go for node 3.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

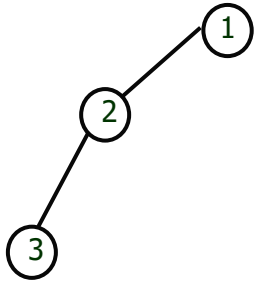
	1	2	3	4	5
x	1	2	3	1	0

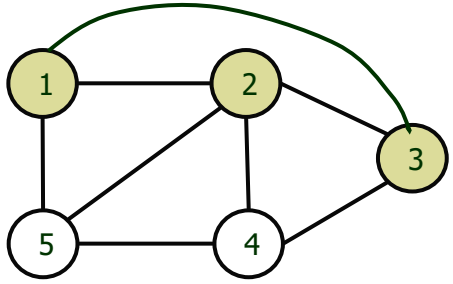
Node 1 already selected so we can't select it again. So we go for node 2.

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

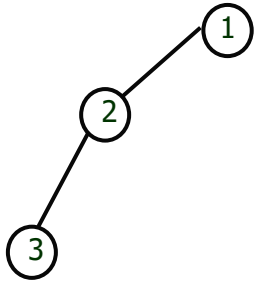
	1	2	3	4	5
x	1	2	3	2	0

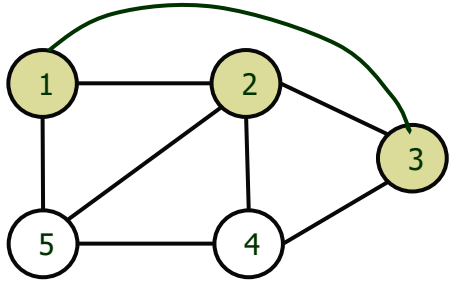
Node 2 already selected so we can't select it again. So we go for node 3.

State Space Tree

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.





	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

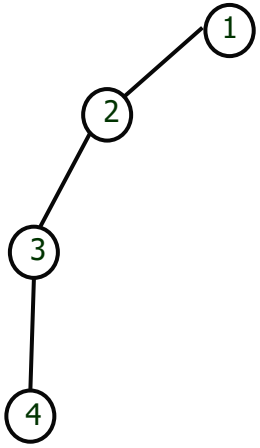
	1	2	3	4	5
x	1	2	3	~~3~~	0

State Space Tree

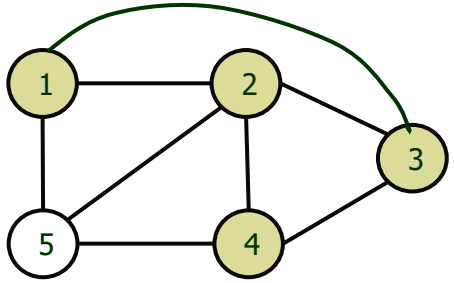
Node 3 already selected so we can't select it again. So we go for node 4.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

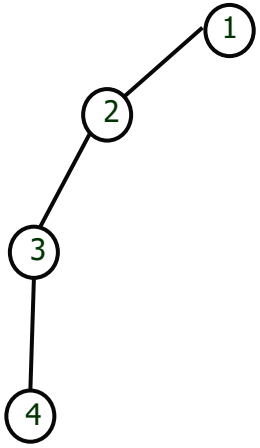
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	0

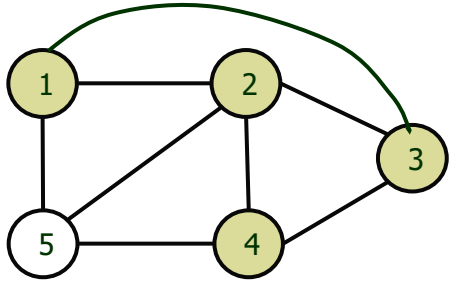
Node 3 already selected so we can't select it again. So we go for node 4.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

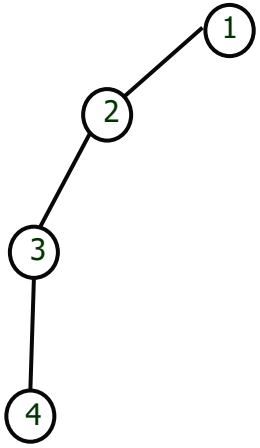
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	1

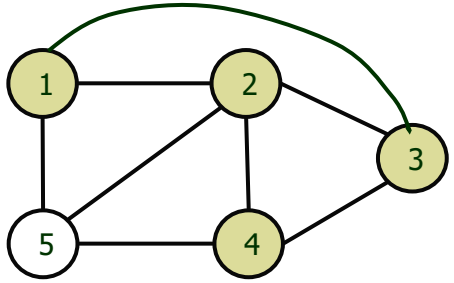
Node 1 already selected so we can't select it again. So we go for node 2.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

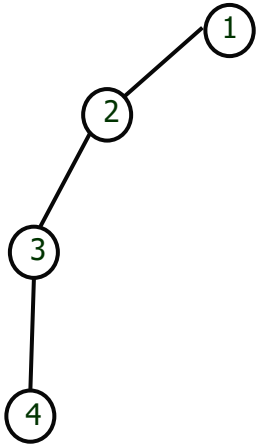
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	2

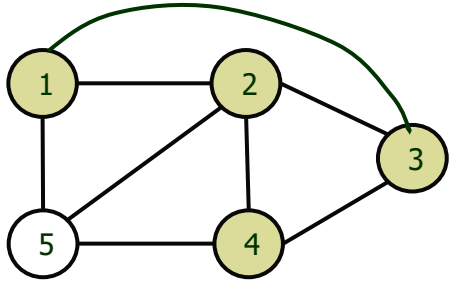
Node 2 already selected so we can't select it again. So we go for node 3.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

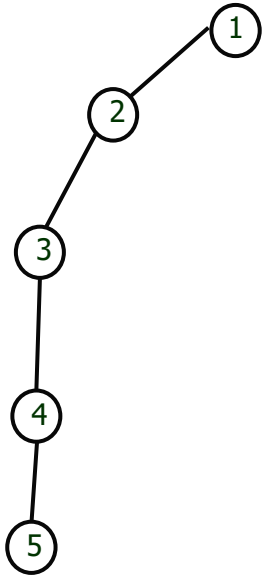
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	3

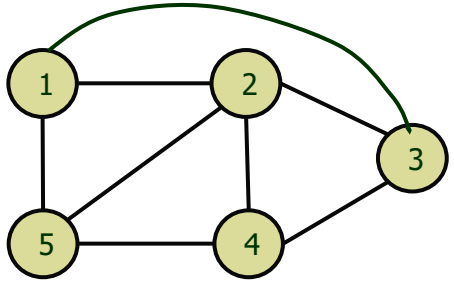
Node 3 already selected so we can't select it again. So we go for node 4.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

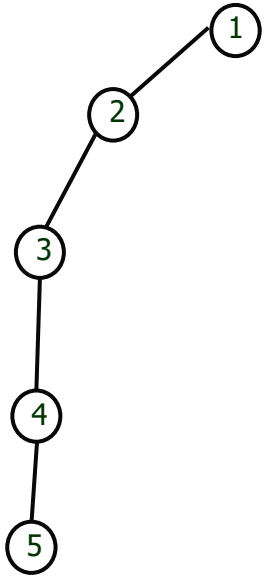
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	5

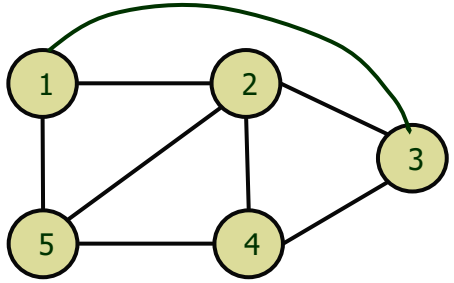
Node 4 already selected so we can't select it again. So we go for node 5.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

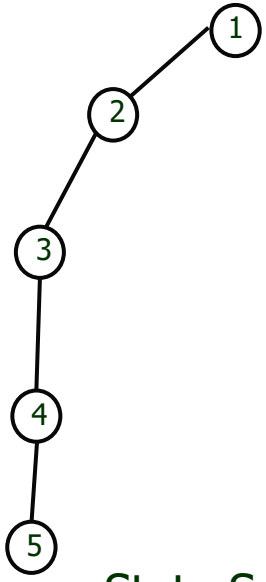
Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	5

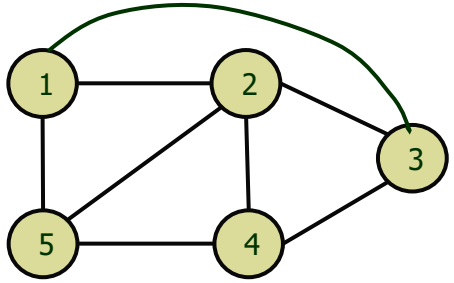
Hamilton cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

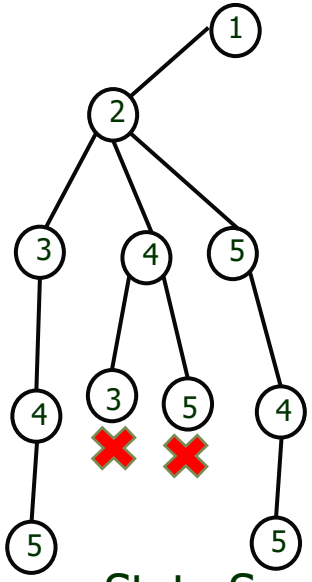
	1	2	3	4	5
x	1	2	3	4	5

Hamilton cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$

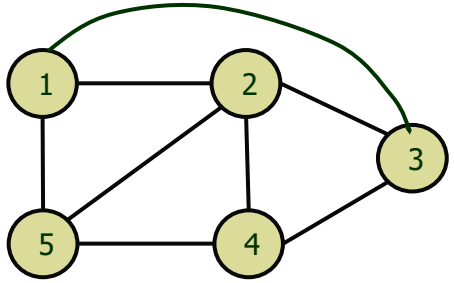
The complete state space tree is shown in next slide.

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

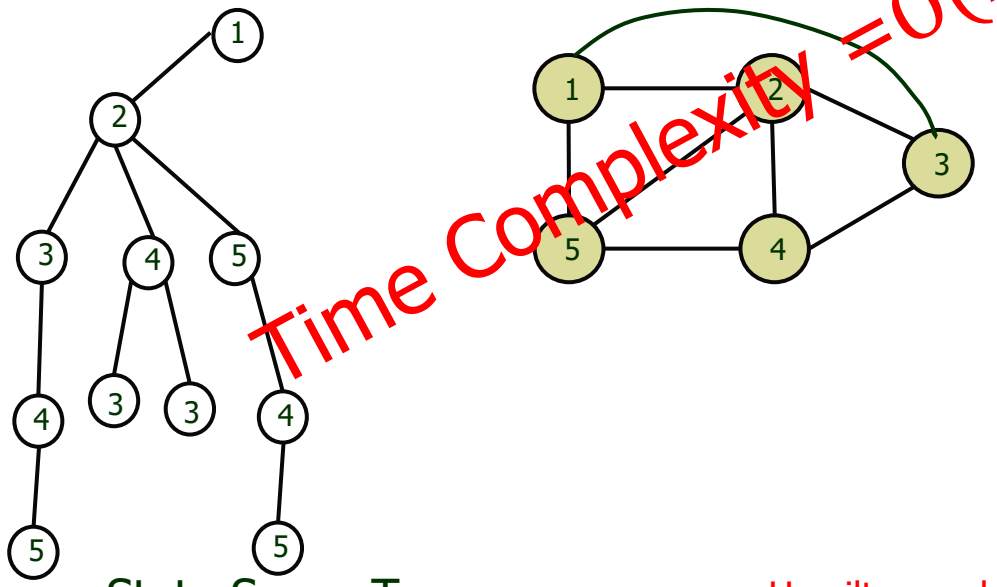
	1	2	3	4	5
x	1	2	3	4	5

Hamilton cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$

Similarly do it for starting node 2.(Self practice)

Backtracking

- **Problem 1 : Hamiltonian cycle**
- **Example 1:** Find the Hamiltonian cycle of the given graph by using backtracking.



State Space Tree

Time Complexity $= O(n!)$

	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacent Matrix

	1	2	3	4	5
x	1	2	3	4	5

Hamilton cycle: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$

Similarly do it for starting node 2.(Self practice)

Backtracking

- **Problem 1 : Hamiltonian cycle**

  Algorithm:

  The main algorithm starts by

  - initializing the adjacency matrix $G[1:n, 1:n]$
  - setting $x[2:n] = 0$
  - setting $x[1] = 1$ (As starting vertex)
  - executing $Hamiltonian(2)$

Note: This algorithm($Hamiltonian(2)$) uses the recursive formulation of backtracking to find all the Hamiltonian cycles of a graph. The graph is stored as an adjacency matrix G[1:n][1:n]. All cycles begin at node 1

Backtracking

- **Problem 1 : Hamiltonian cycle**

Algorithm:

Recursive algorithm that finds all Hamiltonian cycles

```
void Hamiltonian(int k)
  {
     repeat
     { // Generate values for x[k].
       NextValue(k);  // Assign a legal next value to x[k].
       if (x[k]==0) the return;
       if (k == n) then print([1:n])
       else Hamiltonian(k+1);
      } until (false);
   }
```

Backtracking

- The NextValue(int k) searching is execute on following procedure:
  - x[1],...,x[k-1] is a path of k-1 distinct vertices.
  - If x[k]==0, no vertex has yet been assigned to x[k].
  - After execution x[k] is assigned to the lowest numbered vertex that does not already appear in x[1],x[2],...,x[k-1]; and is connected by an edge to [k-1] Otherwise x[k]==0, If k==n, then in addition x[k] is connected to x[1].

Backtracking

- **Problem 1 : Hamiltonian cycle**

Algorithm:

```
void NextValue(int k)
  {
   repeat
   {
       x[k] = (x[k]+1) % (n+1); // Next vertex
       if (x[k]==0) then return;
       if (G[x[k-1]][x[k]]==0)  then
       { // Is there an edge?
          for (j=1 to k-a) do
              if (x[j]==x[k]) then break;   // Check for distinctness.
           if (j==k) then  // If true, then the vertex is distinct.
             if ((k<n) || ((k==n) && G[x[n]][x[1]]!=0))
                 then return;
         }
    } until(false);
  }
```

Backtracking

- Problems
  1. Hamiltonian cycle
  2. Graph Coloring
  3. Sum of subset
  4. N-Queen

Backtracking

- **Problem 2 : Graph Coloring**
  - Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. (Hear coloring of a graph means assignment of colors to all vertices)

Backtracking

- **Problem 2 : Graph Coloring**
  - Input:
  1. A 2D array graph[v][v] where v is the number of vertices in graph and graph[v][v] is adjacency matrix representation of the graph.

$$graph[i][j] = \begin{cases} 1, & if\ there\ is\ a\ direct\ edge\ from\ i\ to\ j \\ 0, & otherwise \end{cases}$$

  2. An integer m which is maximum number of colors that can be used.

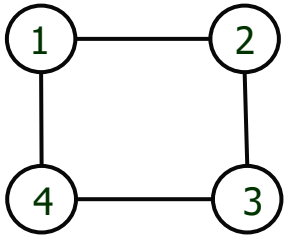
Backtracking

- **Problem 2 : Graph Coloring**
  - Output:

  An array x[v] that should have numbers from 1 to m. x[i] should represent the color assigned to the i^{th} vertex . The code should also return false if the graph cannot be colored with m colors.

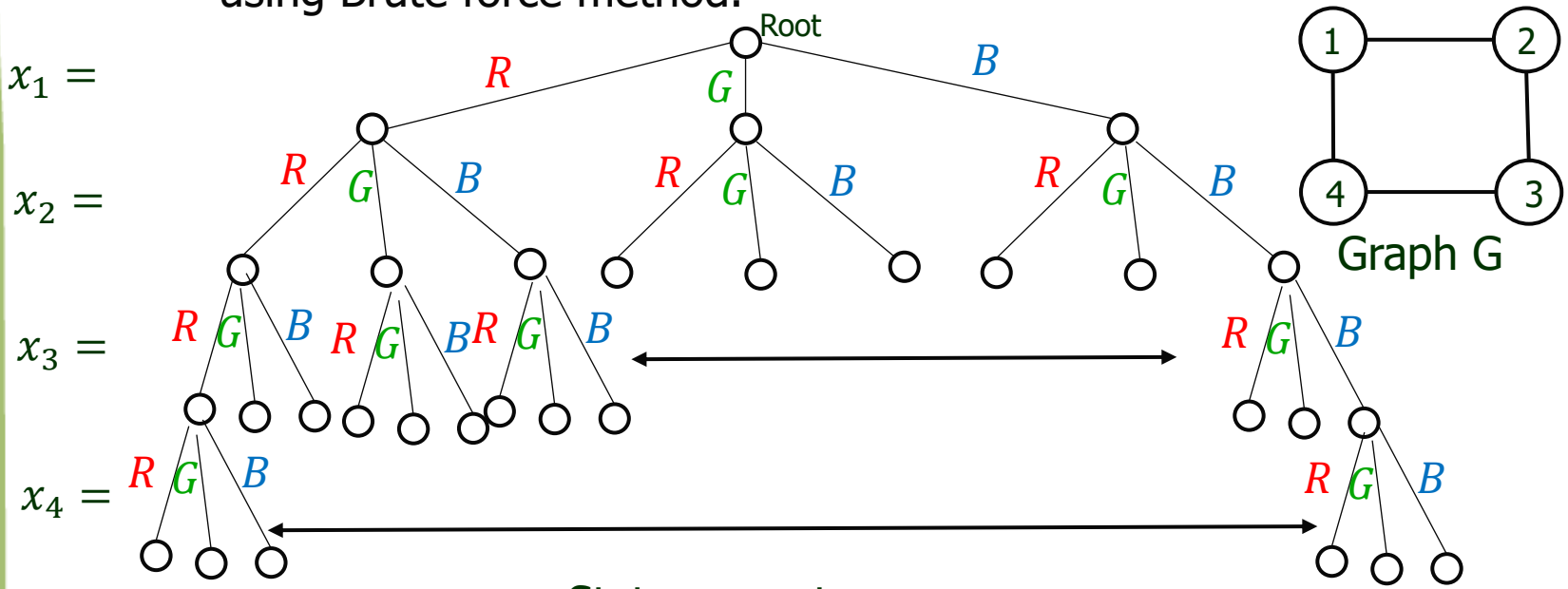
Backtracking

- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Brute force method.



Backtracking

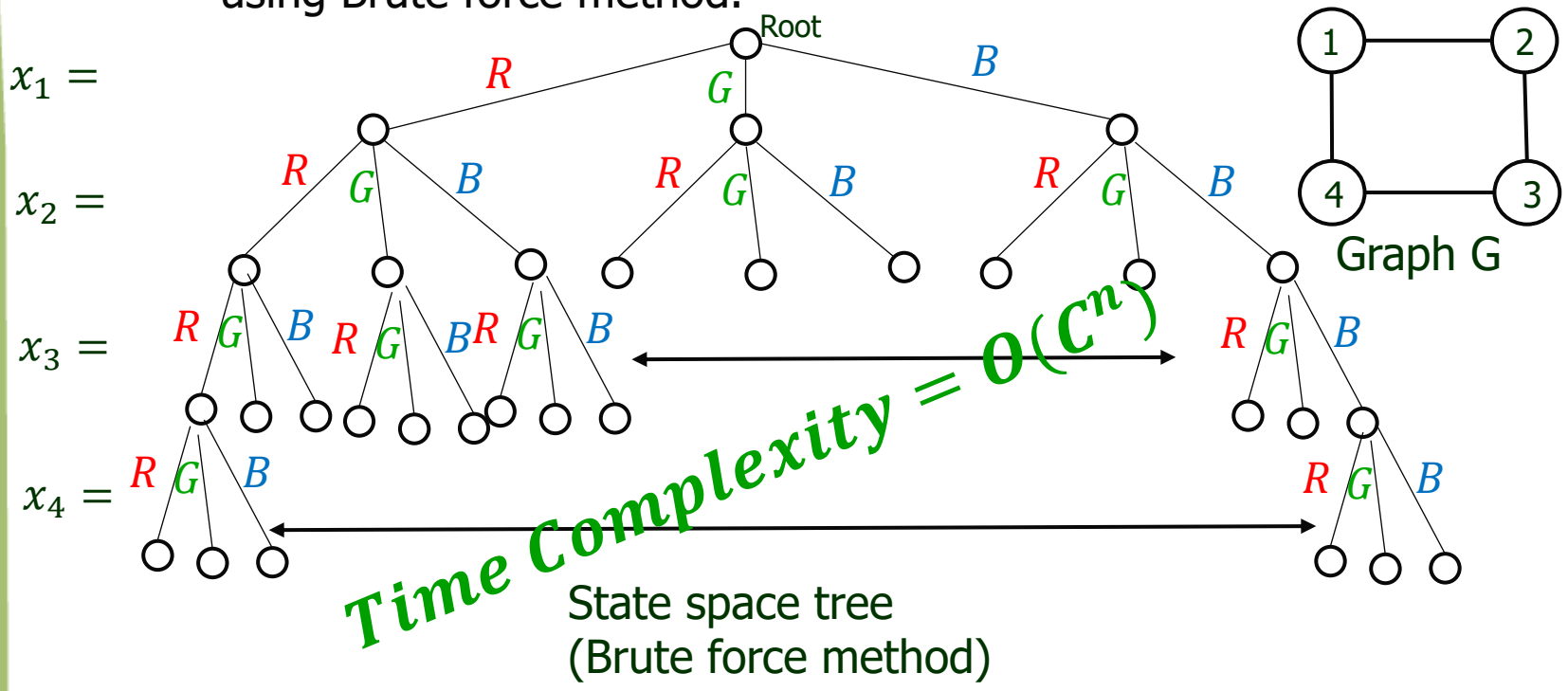
- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Brute force method.



State space tree
(Brute force method)

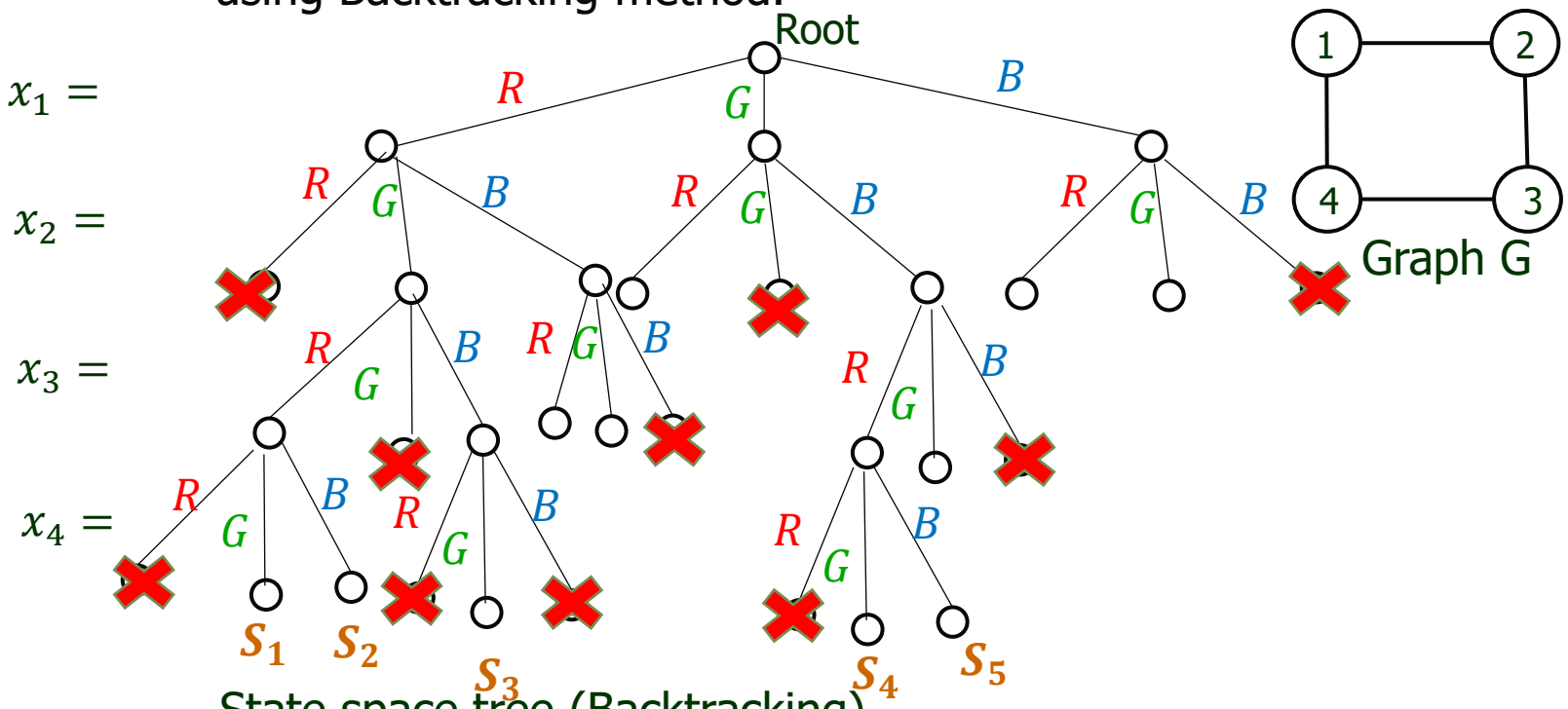
Backtracking

- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Brute force method.



Backtracking

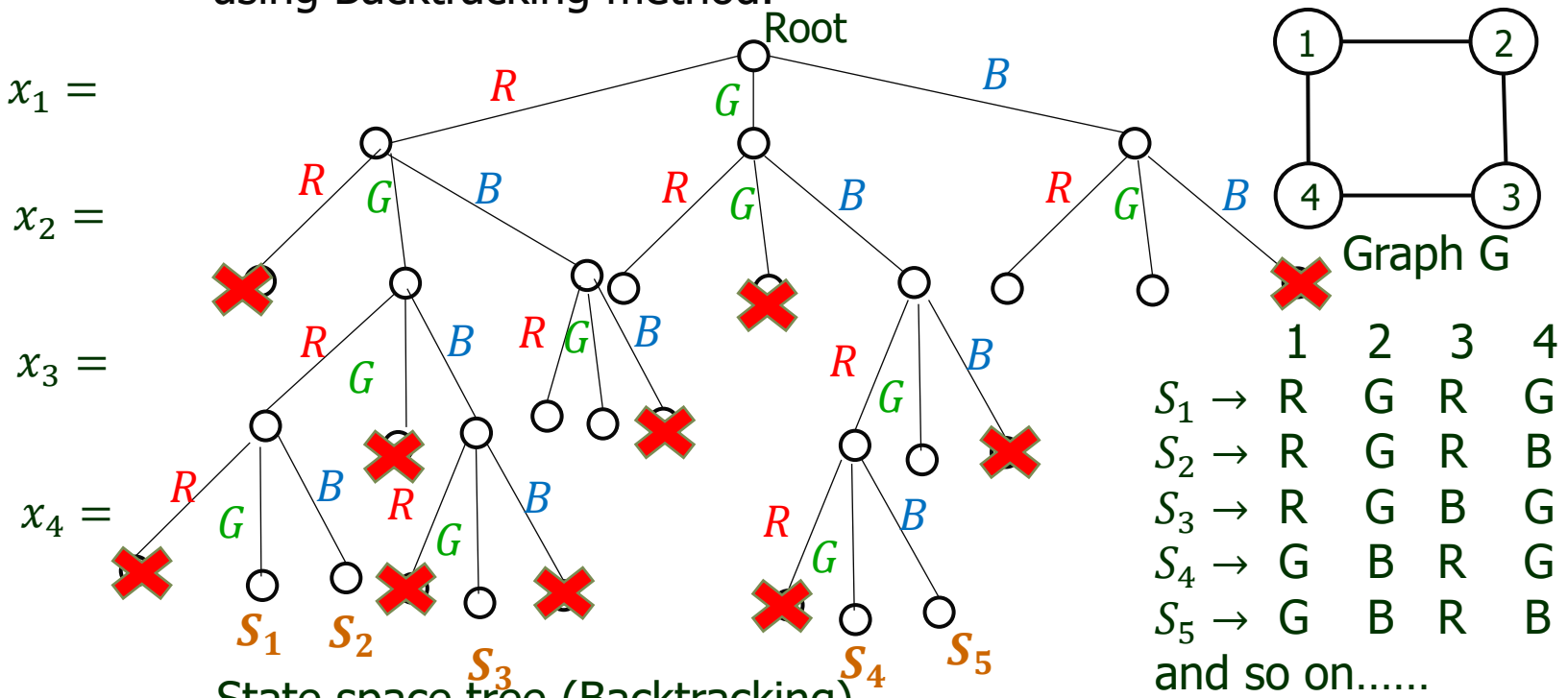
- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Backtracking method.



State space tree (Backtracking)

Backtracking

- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Backtracking method.



State space tree (Backtracking)

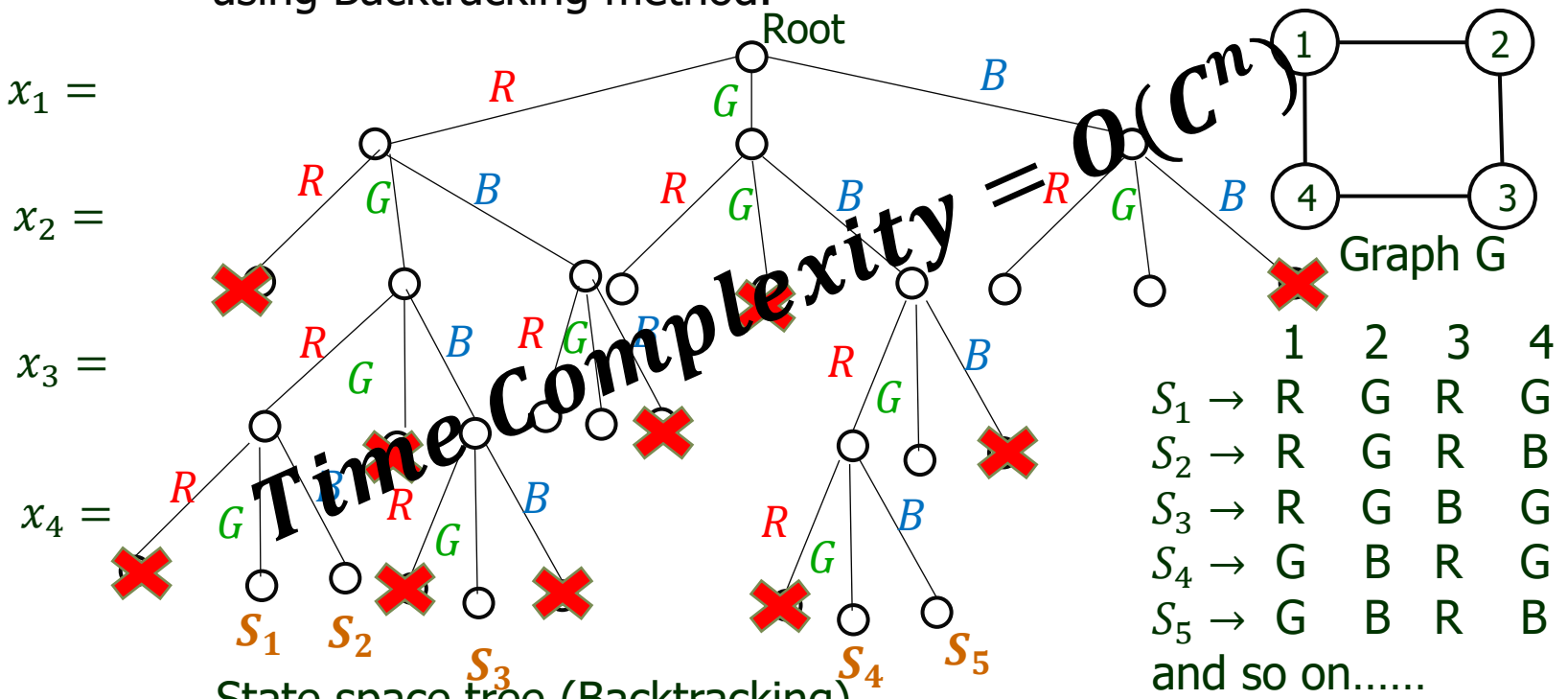
Graph G

	1	2	3	4
$S_1 \rightarrow$	R	G	R	G
$S_2 \rightarrow$	R	G	R	B
$S_3 \rightarrow$	R	G	B	G
$S_4 \rightarrow$	G	B	R	G
$S_5 \rightarrow$	G	B	R	B

and so on.....
(Possible Solutions)

Backtracking

- **Problem 2 : Graph Coloring**
  - Example: Colored the following graph G by using three colour (**R**ed, **G**reen and **B**lue) and draw the state space tree by using Backtracking method.



State space tree (Backtracking)

Backtracking

- **Problem 2 : Graph Coloring**

Algorithm:

Recursive algorithm that finds all Hamiltonian cycles

This program was formed using the recursive backtracking scheme. The graph is represented by its Boolean adjacency matrix G[1:n][1:n]. ssignments of 1,2,….,m

to find the vertices of the graph such that adjacent vertices are assigned distinct integers are printed. Hear k is the next vertex to color.

Backtracking

- **Problem 2 : Graph Coloring**

This algorithm mcoloring (k) was formed using the recursive backtracking schema. The graph is represented by its Boolean adjacency matrix G [1: n, 1: n]. All assignments of1, 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.

Backtracking

- **Problem 2 : Graph Coloring**

```
Algorithm mcoloring (k)
{
  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);
}
```

Backtracking

- **Problem 2 : Graph Coloring**

The Execution procedure of function NextValue(int 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] determined in the range [0,m].
- X[k] is assigned the next highest numbered color while maintain distinctness from the adjacent vertices of vertex k.
- If no such color exits, then x[k]=0.

Backtracking

- **Problem 2 : Graph Coloring**

```
void NextValue(int k)
{repeat
    {
        x[k] = (x[k]+1) % (n+1); // Next vertex
        if (x[k]==0) return;
        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 adjacent vertices have the same color
                then break;
        }
         if (j=n+1) then return;// New color found
     } until(false); // Otherwise try to find another color
   }
```

Backtracking

- Problems
  1. Hamiltonian cycle
  2. Graph Coloring
  3. Sum of subset
  4. N-Queen

Backtracking

- **Problem 3 : Sum of Subset**
  - Given positive numbers $w_i, 1 \leq i \leq n, and\ m$, find all subsets of $\{w_1, w_2, \ldots\ldots, w_n\}$, whose sum is m.

  Example:

  1. If $n = 6, \{w_1, w_2, w_3, w_4, w_5, w_6\} = \{5, 10, 12, 13, 15, 18\}$ and $m = 30$, the desired solution sets are $(5, 10, 15), (5, 12, 13)\ and\ (12, 18)$ and are represented as $(1,1,0,0,1,0), (1,0,1,1,0,0), and\ (0,0,1,0,0,1)$.
  2. If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(7, 11, 13), and\ (7, 24)$ and are represented as $(1,1,1,0), and\ (1,0,0,1)$.

Backtracking

- **Problem 3 : Sum of Subset**
  - Given positive numbers $w_i, 1 \leq i \leq n, and\ m$, find all subsets of $\{w_1, w_2, \ldots\ldots, w_n\}$, whose sum is m.

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(7, 11, 13), and\ (7, 24)$. And are represented as $(1, 1,1,0), and\ (1,0,0,1)$.

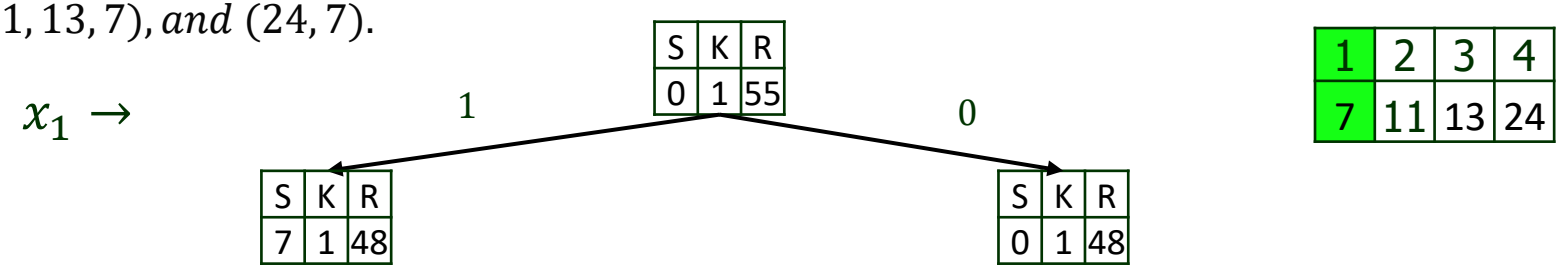
Let's execute the above example with the help of Backtracking and draw the State Space Tree.

Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.



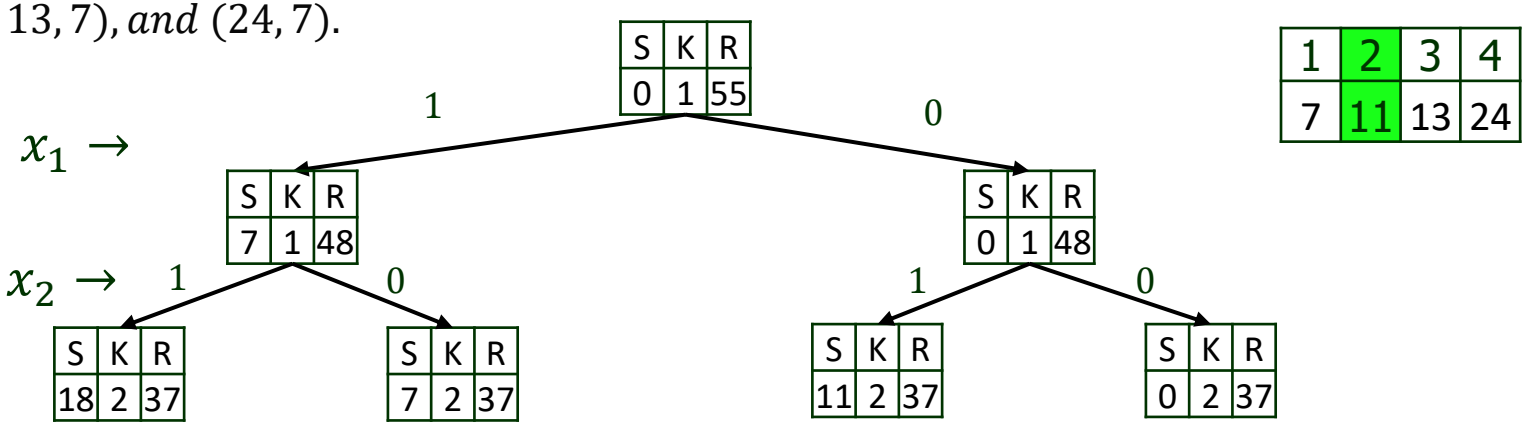
1	2	3	4
7	11	13	24

Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.



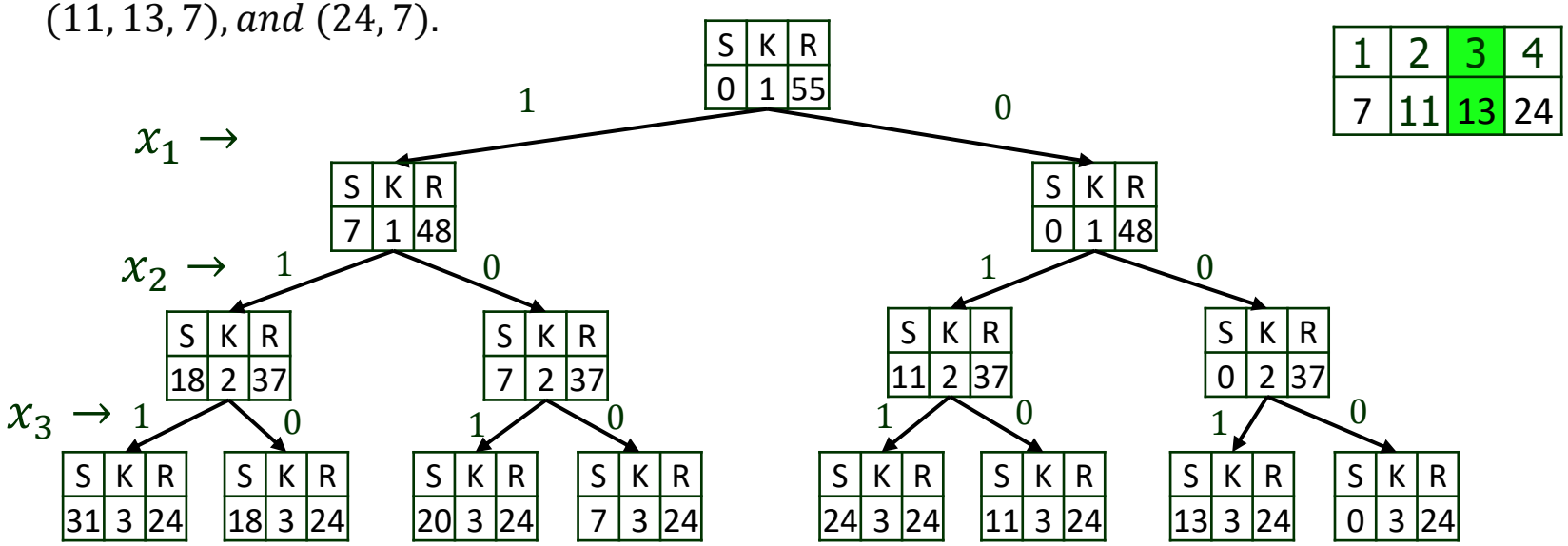
1	2	3	4
7	11	13	24

Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.



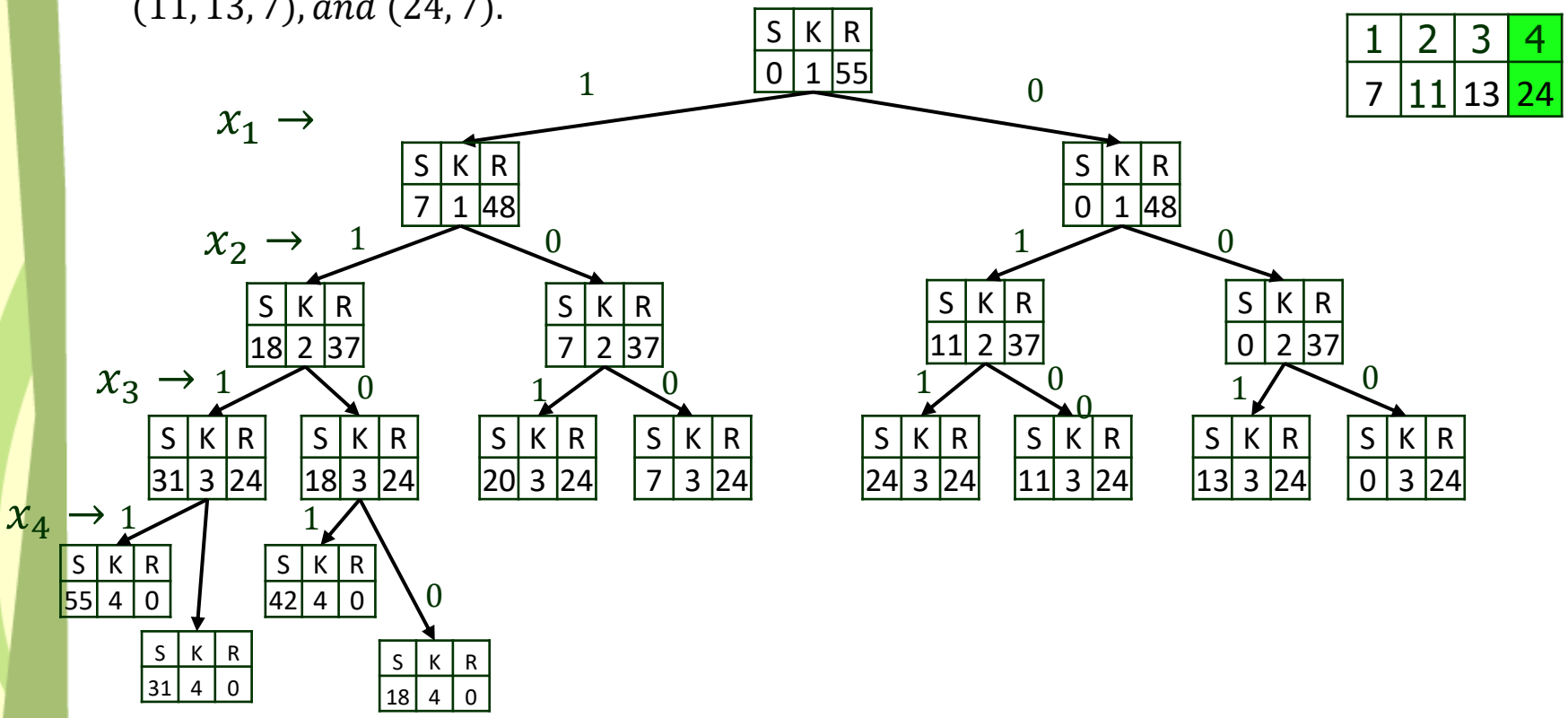
1	2	3	4
7	11	13	24

Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.

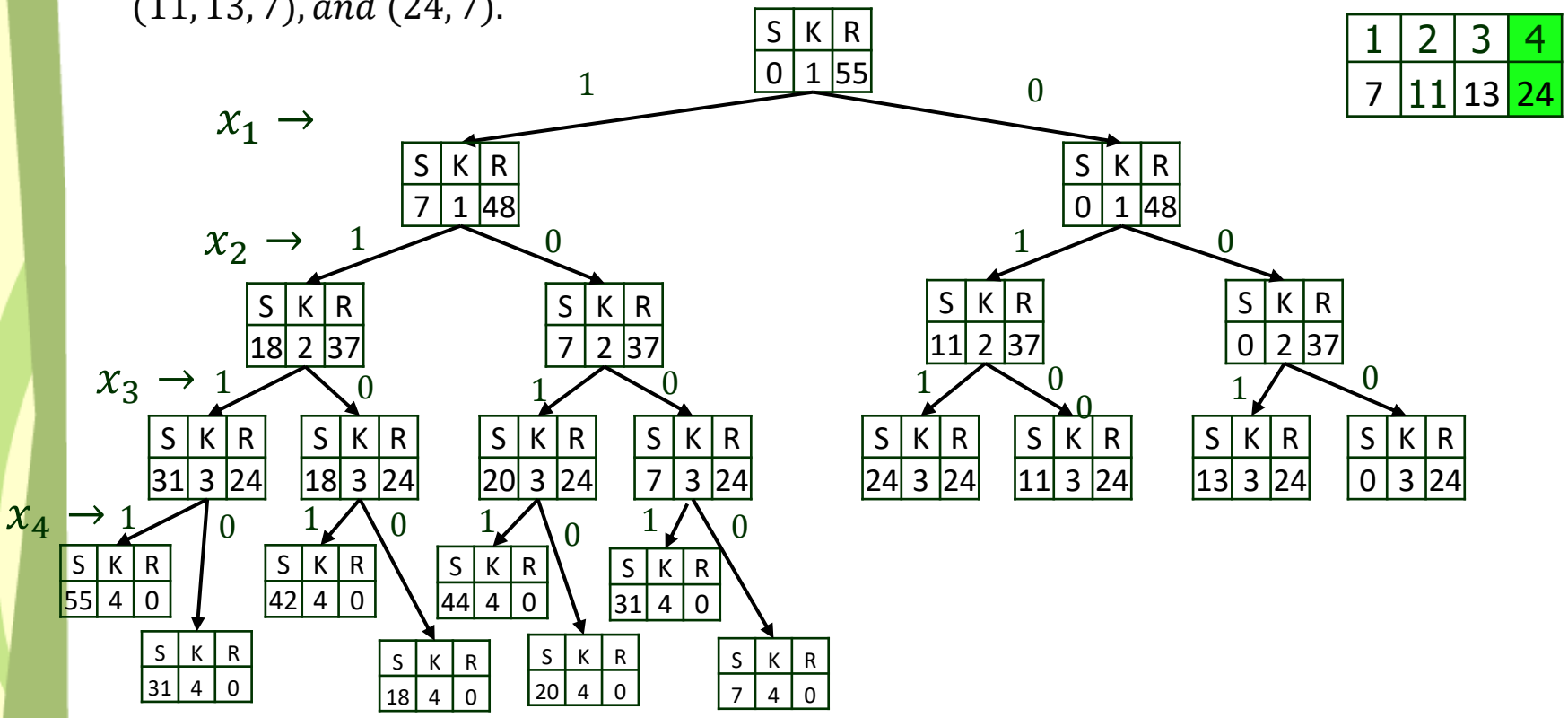


Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.

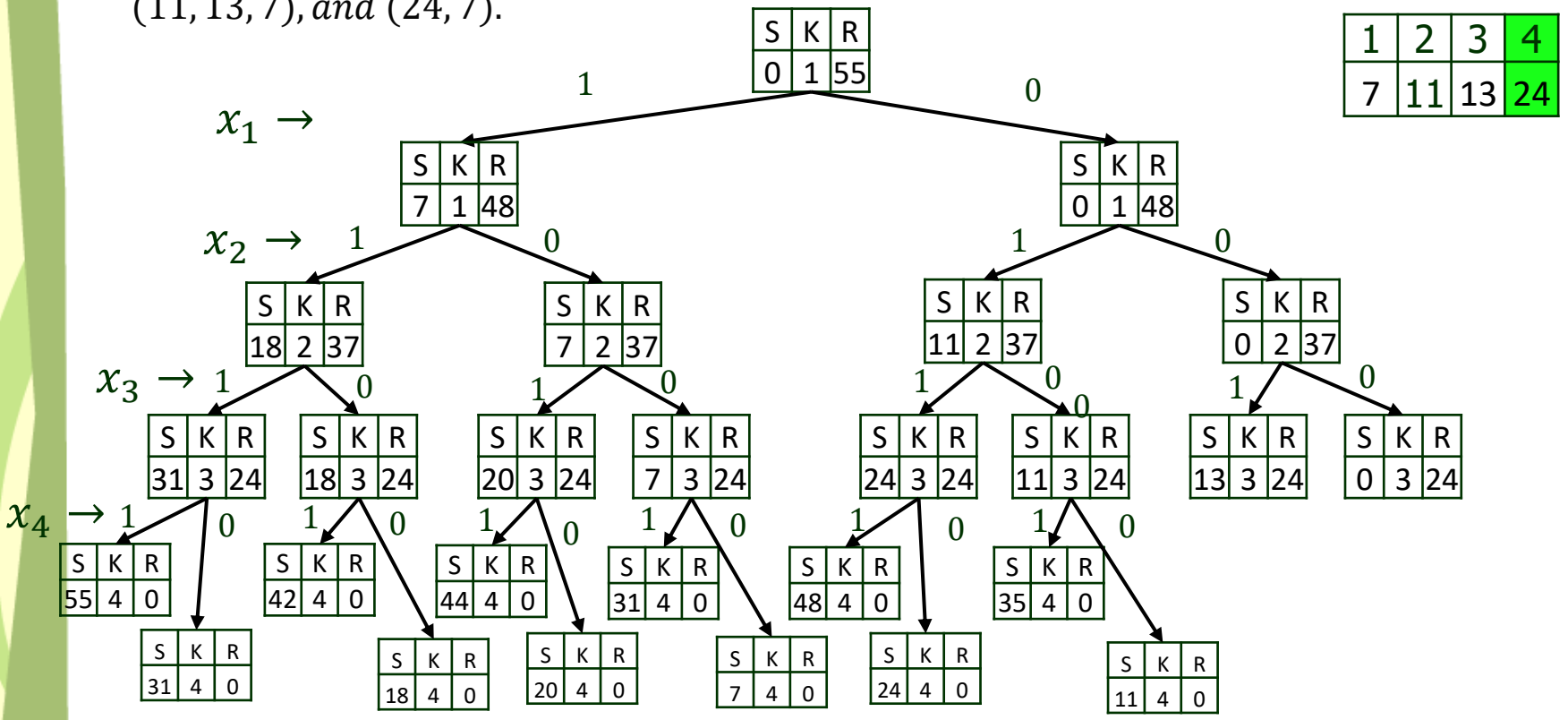


Backtracking

- **Problem 3 : Sum of Subset**

Example:

If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.

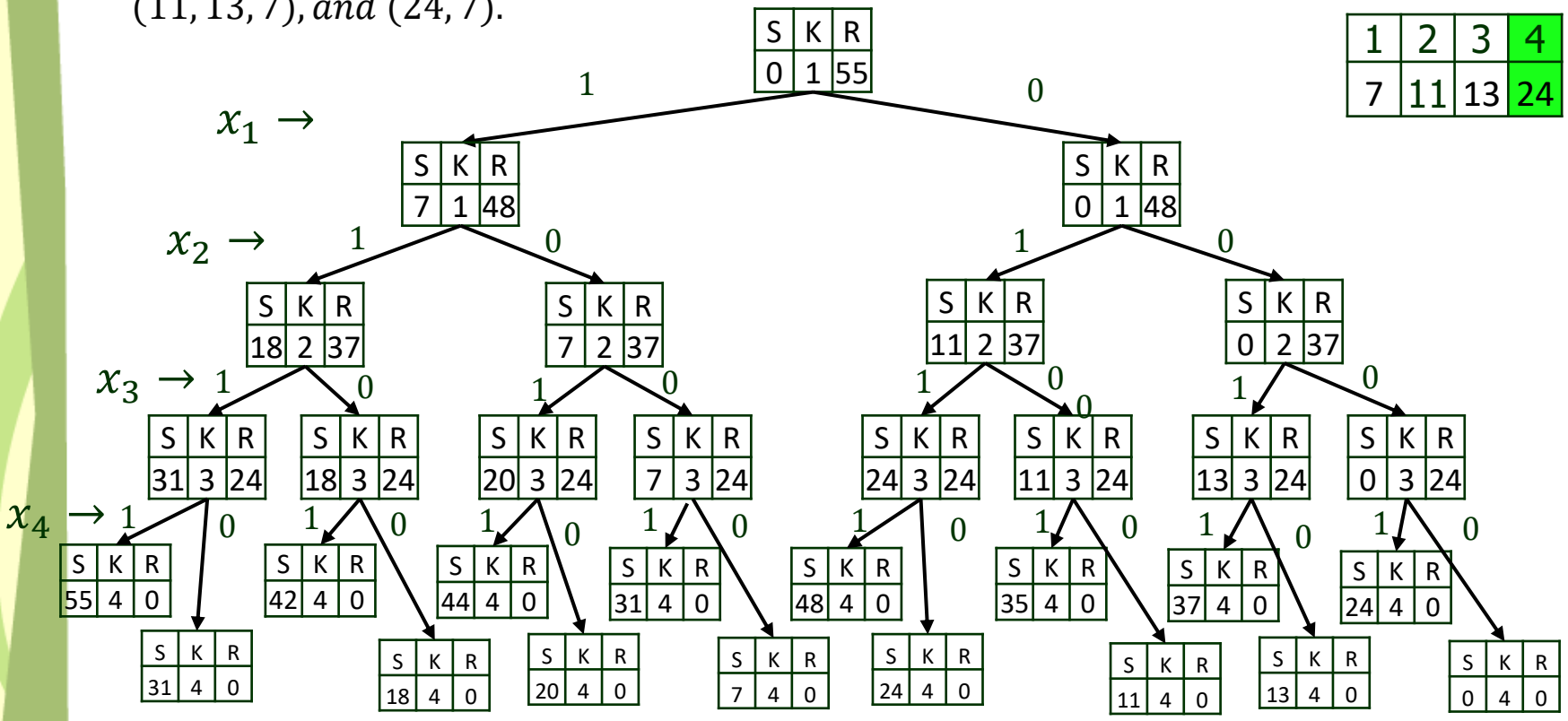


Backtracking

- **Problem 3 : Sum of Subset**

  Example:

  If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.

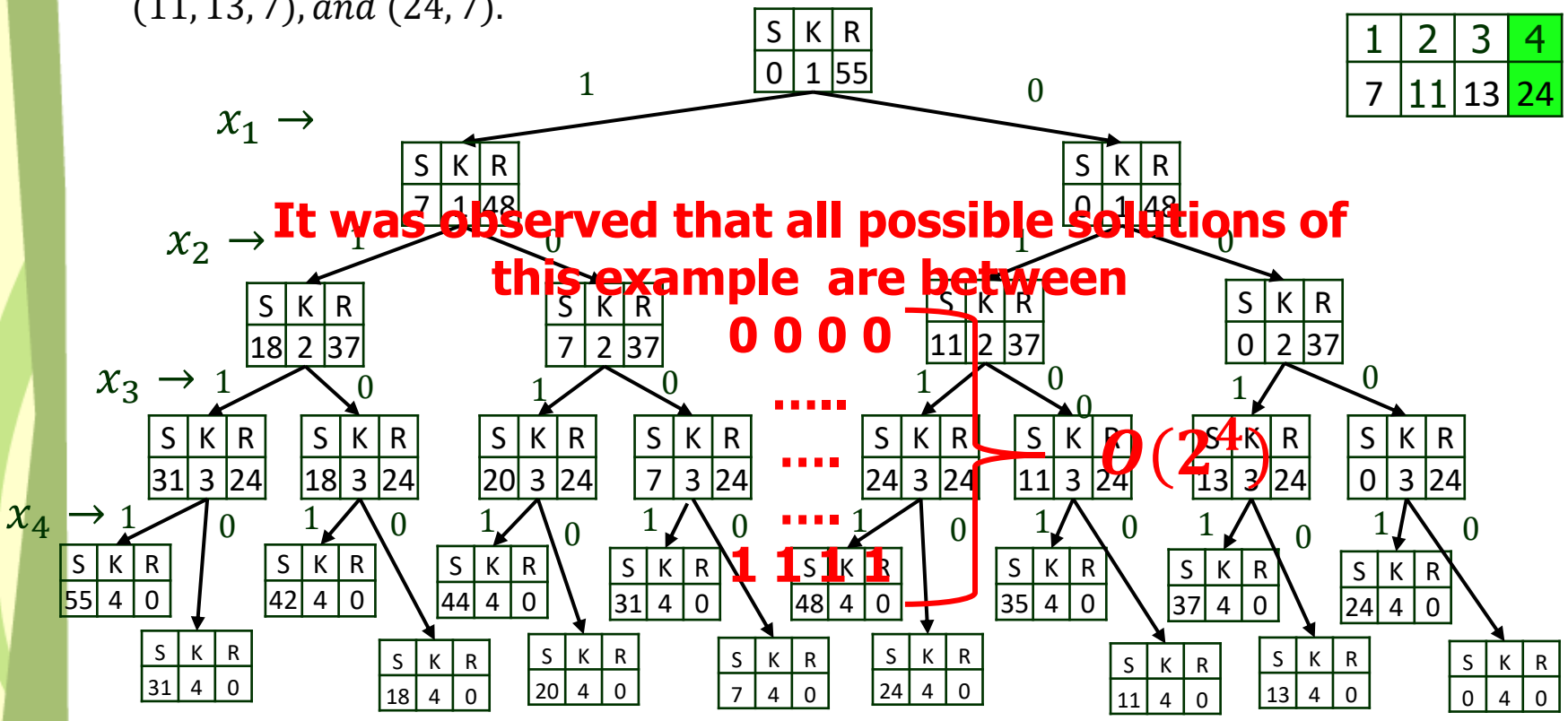


Backtracking

- **Problem 3 : Sum of Subset**

Example:

If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.

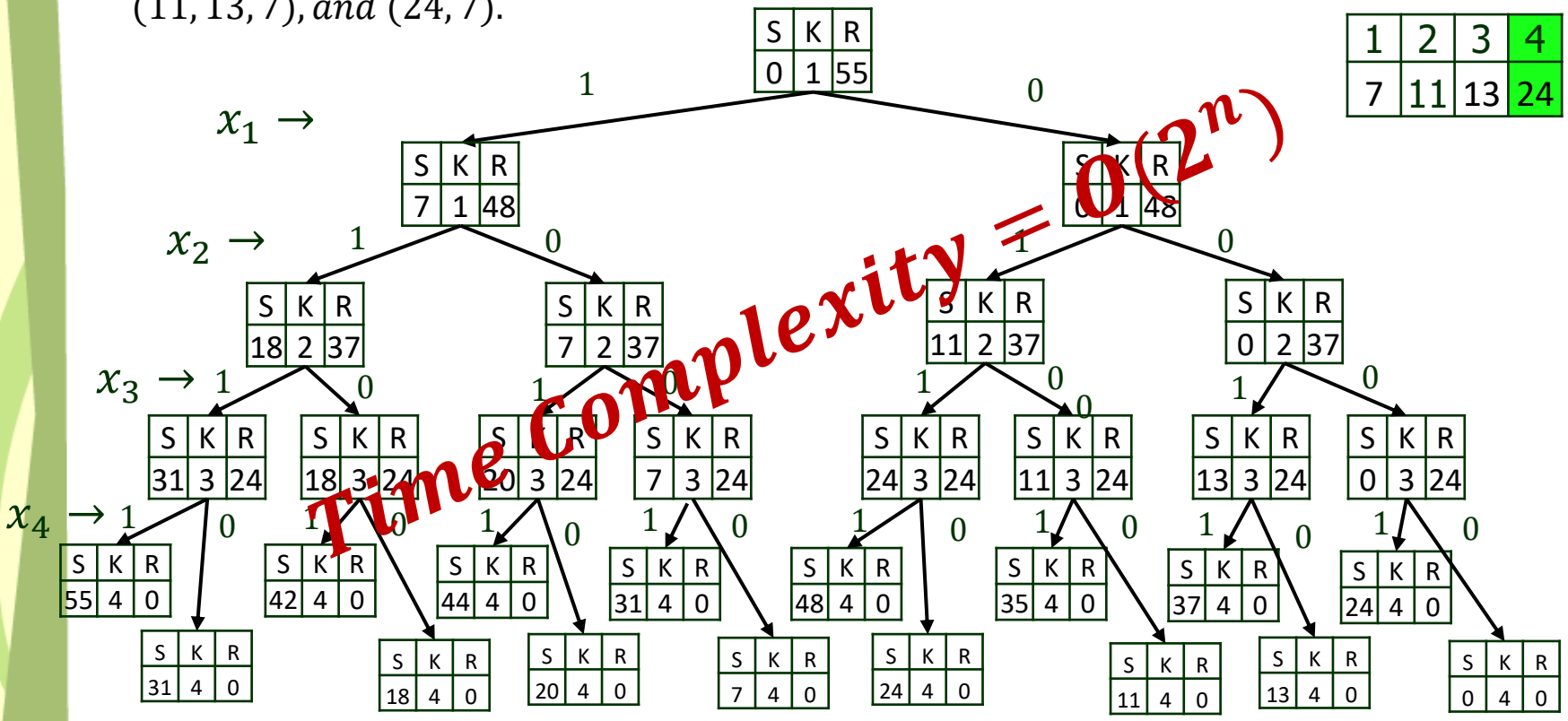


Backtracking

- **Problem 3 : Sum of Subset**

Example:

If $n = 4, \{w_1, w_2, w_3, w_4\} = \{7, 11, 13, 24\}$ and $m = 31$, the desired solution sets are $(11, 13, 7), and\ (24, 7)$.



Backtracking

- **Problem 3 : Sum of Subset**

  What to do before executing SumOfSubset():

  - Find all subsets of $w[1:n]$ that sum to m.
  - The values of $x[j], 1 \leq j \leq k,$ have already been determined.
  - $s = \sum_{j=1}^{k-1} w[j] \times x[j] \ \ and \ r = \sum_{j=k}^{n} w[j]$
  - The $w[j]'s$ are in nondecreasing order.
  - It is assumed that $w[1] \leq m \ \ and \ \ \sum_{i=1}^{n} w[i] \geq m$.

Let's execute the example with the help of Backtracking and draw the State Space Tree.

Backtracking

- **Problem 3 : Sum of Subset**

Algorithm:

```
SumOfSubset(s, k, r)
  {
    // Generate left child. Note : s + w[k] ≤ m.
       x[k]=1;
       if (x[k]==0) return;
       if (s + w[k]=m) then print (x[1:k]);  // Subset found
                //There is no recursive call here as w[j]>0, 1≤ j ≤n.
       else if (s + w[k] +w[k+1] ≤ m)
                then SumOfSubset(s + w[k], k + 1, r - w[k]);
       // Generate right child.
       if ((s + r - w[k] ≥ m) and (s+w[k+1] ≤ m)) then
       {
          x[k]=0;
          SumOfSubset(s, k + 1, r - w[k]);
        }
  }
```

Backtracking

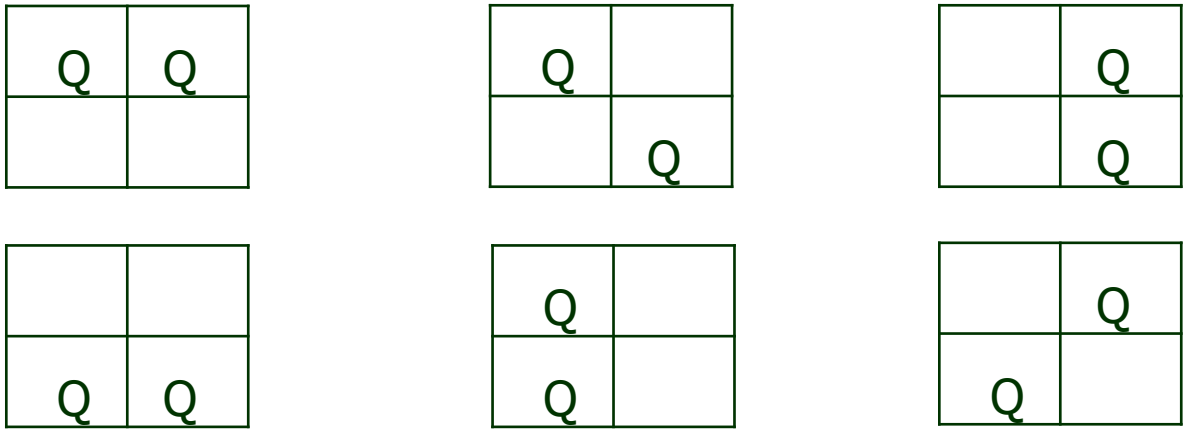
- Problems
  1. Hamiltonian cycle
  2. Graph Coloring
  3. Sum of subset
  4. N-Queen

Backtracking

- **Problem 4 : N Queen**
  - N-Queen problem is based on chess games.
  - The problem is based on arranging the queens on chessboard in such a way that no two queens can attack each other.
  - The N-Queen problem states as consider a n x n chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

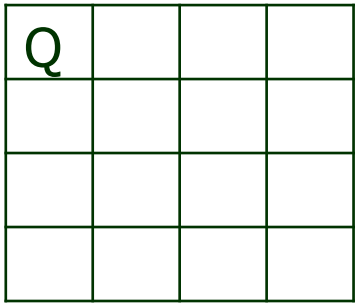
Backtracking

- **Problem 4 : N Queen**
  - 2 – Queen's problem is not solvable because 2 – Queens can be placed on 2 x 2 chess board as shown in below.



Backtracking

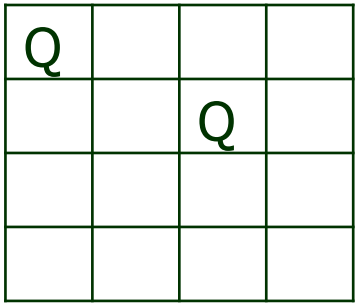
- **Problem 4 : N Queen**
- How to solve N-Queen Problem.
  1. Let us take the example of 4 – Queens and 4 x 4 chessboard.
  2. Start with an empty chessboard.
  3. Place queen 1 in the first possible position of its row i.e. on 1st row and 1st column



Backtracking

- **Problem 4 : N Queen**
- How to solve N-Queen Problem.

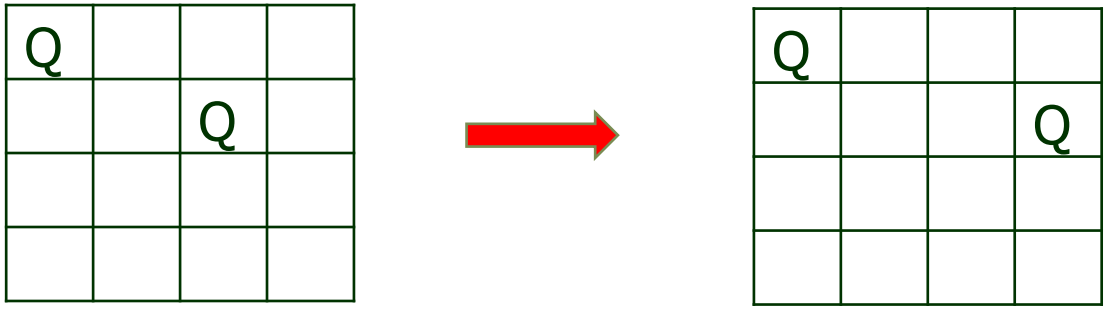
4. Then place queen 2 after trying unsuccessful place (1, 2), (2, 1), (2, 2) at (2, 3) i.e. 2nd row and 3rd column.



Backtracking

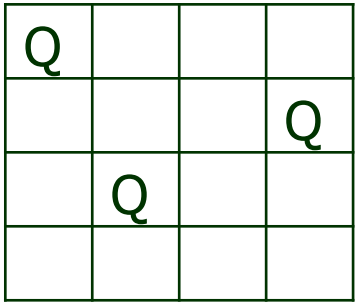
- **Problem 4 : N Queen**
- How to solve N-Queen Problem.

5. This is a dead end because a 3rd queen cannot be placed in next column, as there is no acceptable position for queen 3. Hence algorithm backtracks and places the 2nd queen at (2, 4) position..



Backtracking

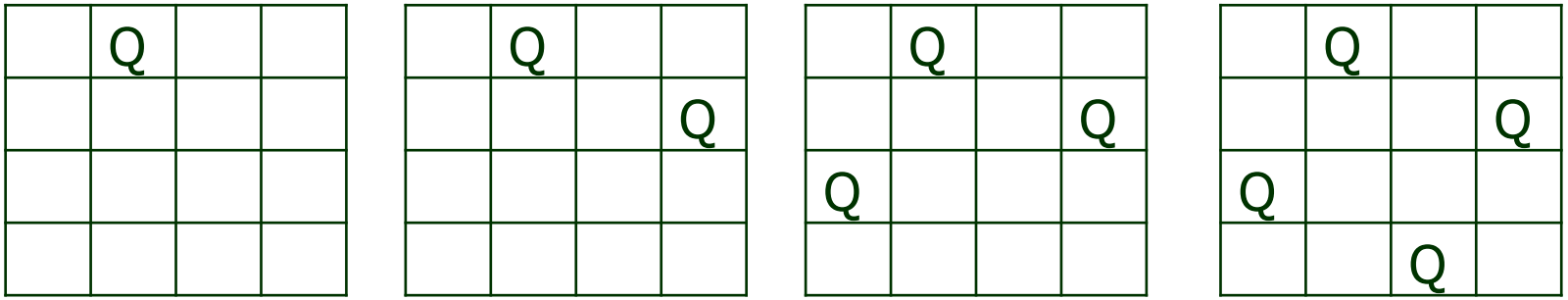
- **Problem 4 : N Queen**
- How to solve N-Queen Problem.
  6. Then place 3rd queen at (3,2) but it again another dead lock end as next queen (4th queen) cannot be placed at permissible position.



Backtracking

- **Problem 4 : N Queen**
- How to solve N-Queen Problem.

7. Hence we need to backtrack all the way up to queen 1 and move it to (1, 2). viii. Place queen 1 at (1, 2), queen 2 at (2, 4), queen 3 at (3, 1) and queen 4 at (4, 3).



8. Thus the solution is obtained (2, 4, 1, 3) in row wise manner in x array.

Backtracking

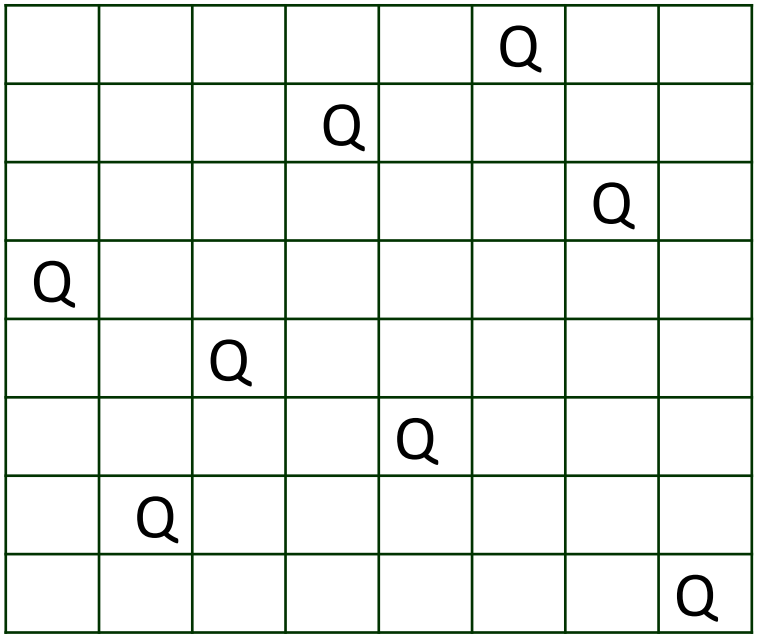
- **Problem 4 : N Queen**
- How to solve N-Queen Problem.

9. A classic problem in combinatorics is to place 8-Queens on an 8 by 8 chessboard so that no two can "attack" each other (along a row, column, or diagonal).
10. Since each queen (1-8) must be on a different row, we can assume queen i is on row i.
11. All solutions to the 8-queens problem can be represented as an 8-tuple $(x_1, x_2, \ldots\ldots, x_n)$ where queen i is on column x_i .
12. The one of the solution to 8-Queen Problem is shown in next slide.

Backtracking

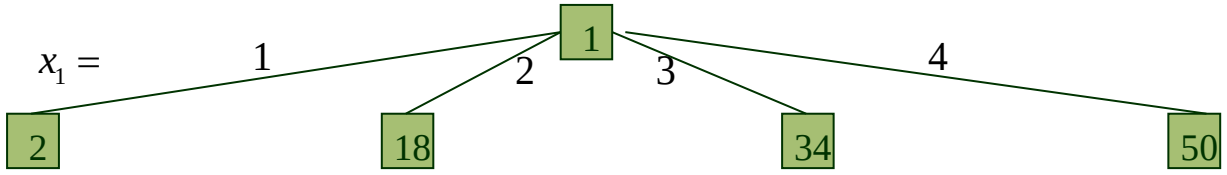
- **Problem 4 : N Queen**

  One of the solution to 8-Queen Problem



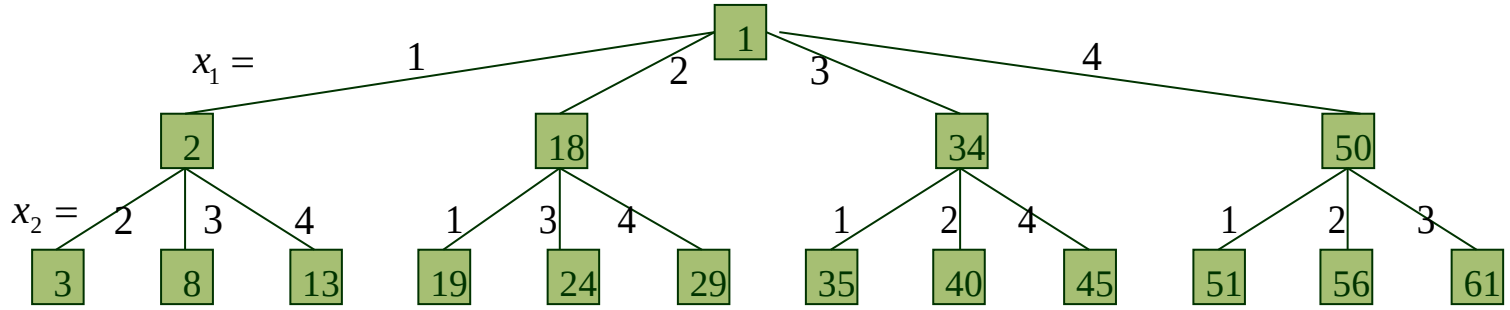
Backtracking

- **Problem 4 : N Queen**
  - N Queens are placed on n by n chessboard so that no two attack (no two queens are on the same row, column, or diagonal).
  - A tree bellow shows the state space tree for n=4.



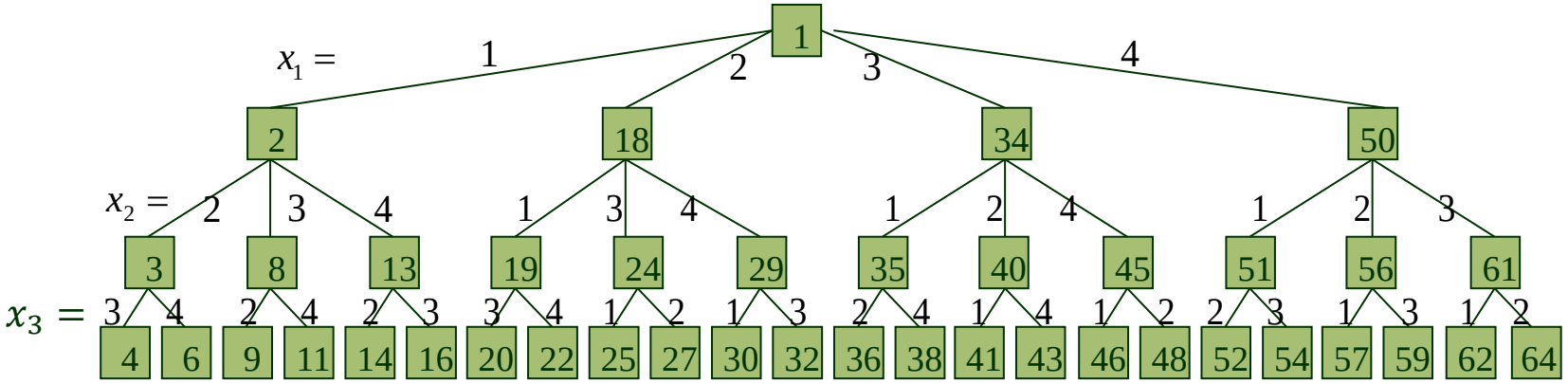
Backtracking

- **Problem 4 : N Queen**
  - N Queens are placed on n by n chessboard so that no two attack (no two queens are on the same row, column, or diagonal).
  - A tree bellow shows the state space tree for n=4.



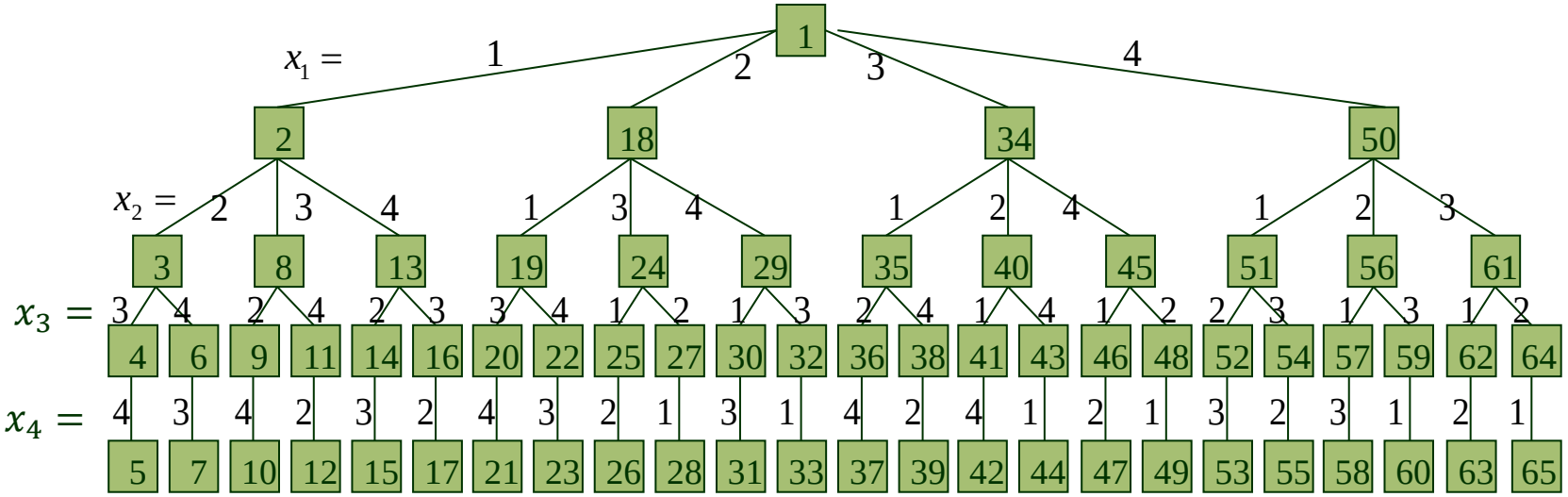
Backtracking

- **Problem 4 : N Queen**
  - N Queens are placed on n by n chessboard so that no two attack (no two queens are on the same row, column, or diagonal).
  - A tree bellow shows the state space tree for n=4.



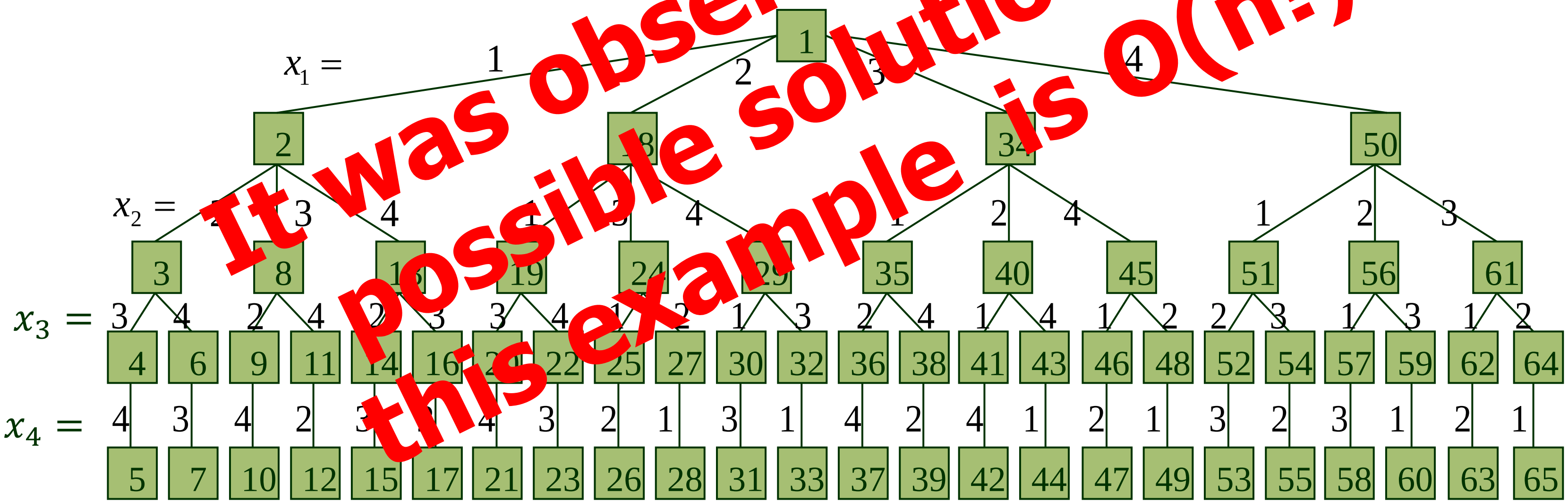
Backtracking

- **Problem 4 : N Queen**
  - N Queens are placed on n by n chessboard so that no two attack (no two queens are on the same row, column, or diagonal).
  - A tree bellow shows the state space tree for n=4.



Backtracking

- **Problem 4 : N Queen**
  - N Queens are placed on n by n chessboard so that no two attack (no two queens are on the same row, column, or diagonal).
  - A tree bellow shows the state space tree for n=4.



It was observed that all possible solutions of this example is O(n!)

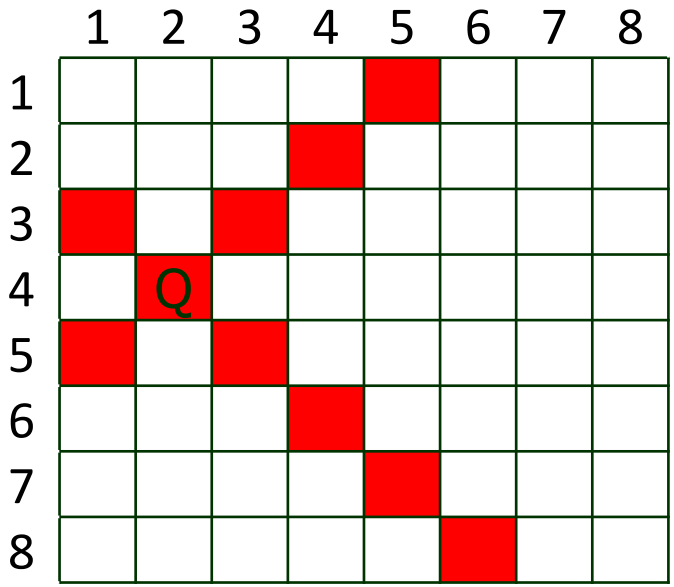
Backtracking

- **Problem 4 : N Queen**
- Logic behind the place() of N-Queen Problem:
  1. Let $(x_1, x_2, \ldots\ldots, x_n)$ represent where the i^{th} queen is placed ($in\ row\ i\ and\ column\ x$) on an $n \times n$ chessboard.
  2. It was observed that two queens on the same diagonal that runs from "upper left" to "lower right" have the same "row-column" value.
  3. Also, two queens on the same diagonal from "upper-right" to "lower left" have the same "row+column" value. (illustrated with a 8 x 8 chessboard)

Backtracking

- **Problem 4 : N Queen**

  Logic behind the place() of N-Queen Problem



4. The Queen is available in [i,j] =[4,2]
5. The squares that are diagonal to that queen from upper left to lower right are [3,1], [5,3], [6,4], [7,5], and [8,6]. (Note : All these squares have a "row-column" value of 2)
6. Similarly, the squares that are diagonal to that queen from upper right to lower left are [1,5], [2,4], [3,3], and [5,1]. (Note : All these squares have a "row+column" value of 6)

Backtracking

- **Problem 4 : N Queen**

  Logic behind the place() of N-Queen Problem:

  7. Then two queens at (i, j) and (k, l) are on the same diagonal if and only if :

     $\Longrightarrow i - j = k - l \; or \; i + j = k + l$

     $\Longrightarrow i - k = j - l \; or \; j - l = k - i$

     $\Longrightarrow \quad |j - l| = |i - k|$

  8. Algorithm PLACE(k,i) returns true if the kth queen can be placed in column i and runs in $O(k)$ time.
  9. Using PLACE, the recursive version of the general backtracking method can be used to give a precise solution to the n-queens problem
  10. Array x[ ] is global. Algorithm invoked by NQueens(1,n).

Backtracking

- **Problem 2 : N Queen (Algorithm)**

Recursive algorithm for N-Queen

```
void NQueens(int k, int n)
// Using backtracking, this procedure prints all possible placements of n queens
// on an n x n  chessboard so that they are nonattacking.
{
 for i=1 to n do
 {
    if (Place(k, i))
    {
       x[k] = i;
       if (k==n)
            print (x[1:n])
       else
            NQueens(k+1, n);
     }
  }
}
```

Backtracking

- **Problem 2 : N Queen (Algorithm)**

Recursive algorithm for N-Queen

```
bool Place(int k, int 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)  or         // Two in the same column
                        || (abs(x[j]-i) == abs(j-k)))    // or in the same diagonal
          return(false);
 return(true);
}
```


Thank U