Algorithms

Ok-Ran Jeong

Fall, 2016

Department of Software
Gachon University

8. Backtracking II

Contents

- Sum of Subsets
- Graph Coloring
- Hamiltonian Circuits Problem
- Backtracking Search

- Problem 19: Two-color Graph
- Problem 20: N-Queens Puzzle

Sum-of-Subsets problem

- Recall the thief and the 0-1 Knapsack problem.
- The goal is to maximize the total value of the stolen items while not making the total weight exceed W.
- If we sort the weights in nondecreasing order before doing the search, there is an obvious sign telling us that a node is nonpromising.

Sum-of-Subsets Problem

- Let *total* be the total weight of the remaining weights, a node at the ith level is nonpromising if

  weight + *total* > W

Example

- Say that our weight values are 5, 3, 2, 4, 1
- W is 8
- We could have
  - 5 + 3
  - 5 + 2 + 1
  - 4 + 3 + 1
- We want to find a sequence of values that satisfies the criteria of adding up to W

Tree Space

- Visualize a tree in which the children of the root indicate whether or not value has been picked (left is picked, right is not picked)
- Sort the values in non-decreasing order so the lightest value left is next on list
- Weight is the sum of the weights that have been included at level i
- Let *weight* be the sum of the weights that have been included up to a node at level i. Then, a node at the ith level is nonpromising if

  $weight + w_{i+1} > W$

example: Map coloring

- The Four Color Theorem states that any map on a plane can be colored with no more than four colors, so that no two countries with a common border are the same color
- For most maps, finding a legal coloring is easy
- For some maps, it can be fairly difficult to find a legal coloring

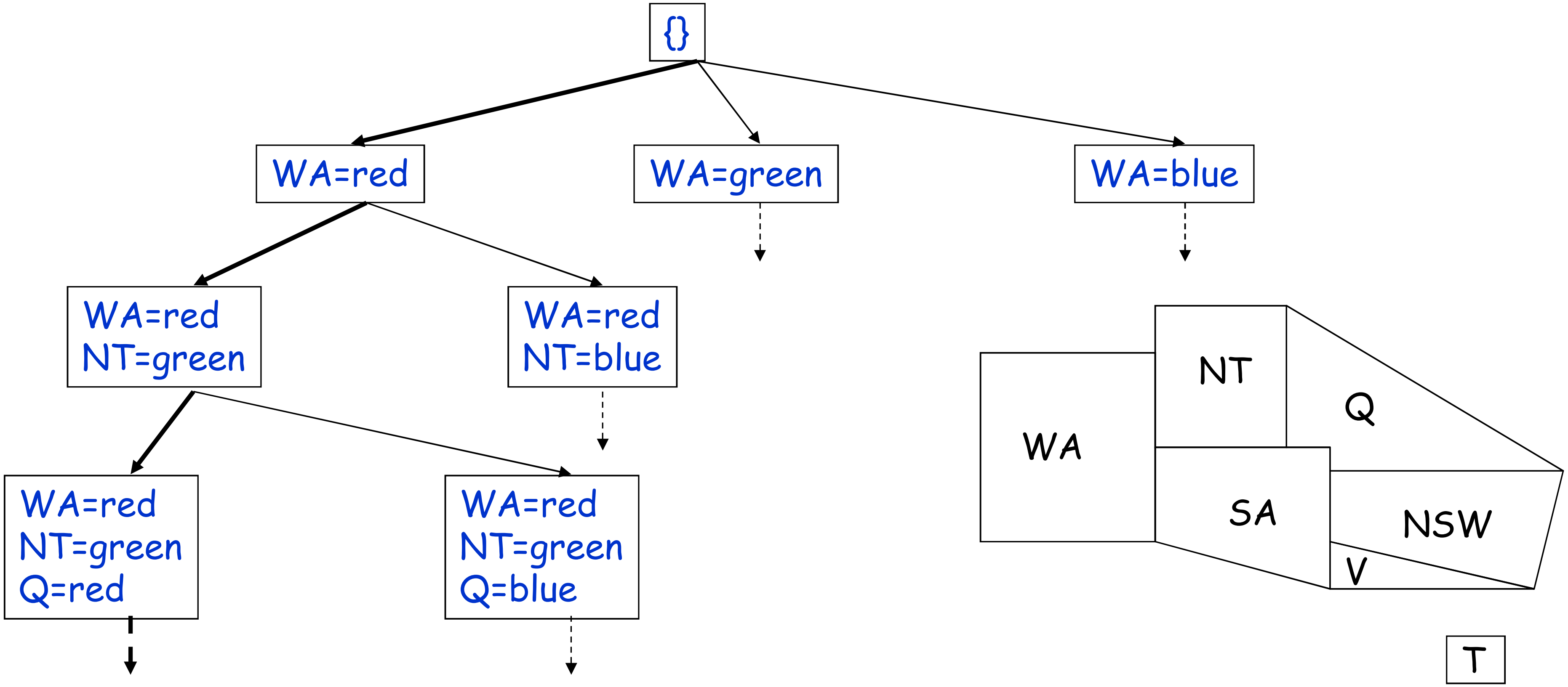
Data Structures

- We need a data structure that is easy to work with, and supports:
  - Setting a color for each country
  - For each country, finding all adjacent countries
- We can do this with two arrays
  - An array of "colors", where countryColor[i] is the color of the i^{th} country
  - A ragged array of adjacent countries, where map[i][j] is the j^{th} country adjacent to country i
    - Example: map[5][3]==8 means the 3^{th} country adjacent to country 5 is country 8

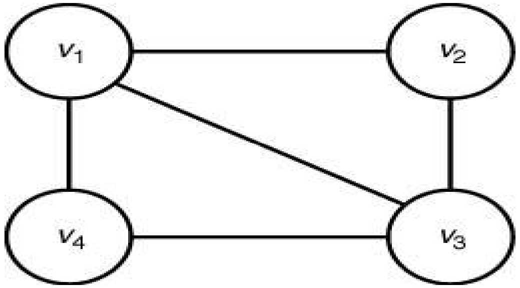
Map coloring

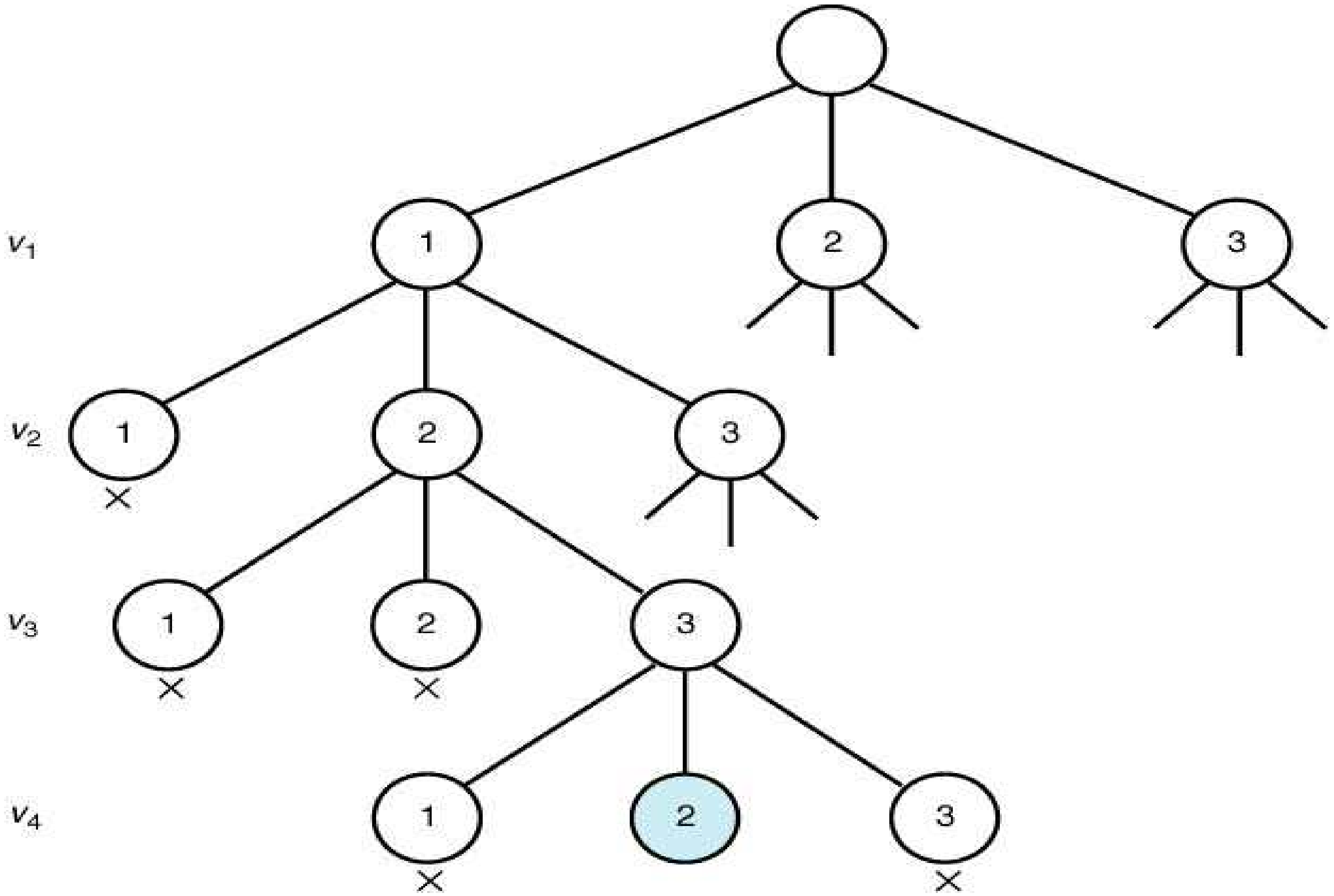
- We went through all the countries recursively, starting with country zero
- At each country we had to decide a color
  - It had to be different from all adjacent countries
  - If we could not find a legal color, we reported failure
  - If we could find a color, we used it and recurred with the next country
  - If we ran out of countries (colored them all), we reported success
- When we returned from the topmost call, we were done

Map coloring



Example: 3-coloring





Graph coloring – algorithm (1/2)

- A Backtracking Algorithm for the Graph Coloring Problem(1/2)

// n: number of nodes, m:number of colors, output: vcolor: assigned color

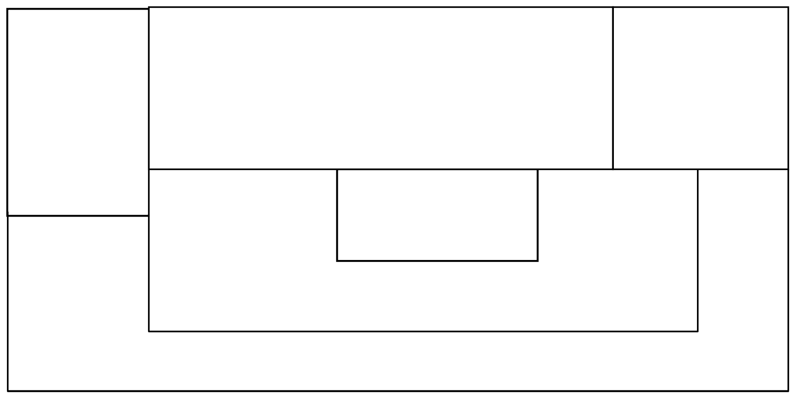
```
void m_coloring(index i)
{
  index color;

  if(promising(i))
   if(i == n)
     cout << vcolor[1] through vcolor[n];
   else
     for(color=1;color <= m;color++) {
        vcolor[i+1] = color;   //assign other color
        m_coloring(i+1);
     }
}
```

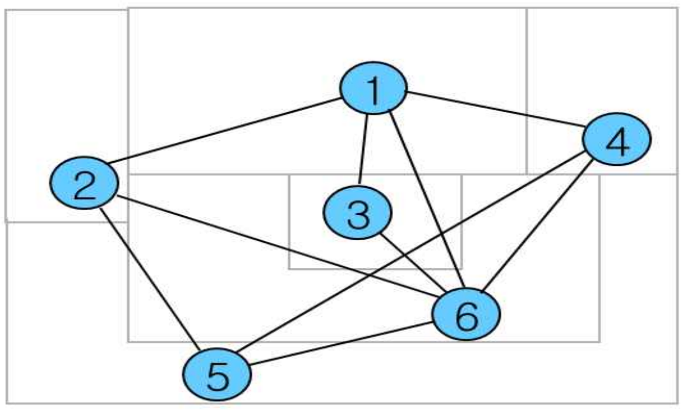
Graph coloring – algorithm (2/2)

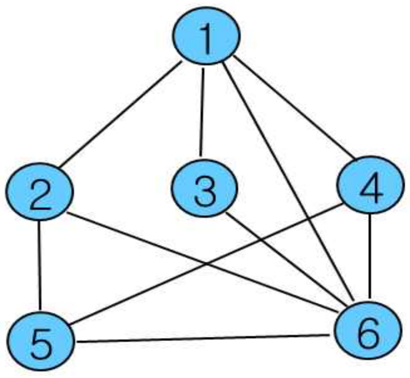
- A Backtracking Algorithm for the Graph Coloring Problem (2/2)

```
bool promising(index i)
{
  index j = 1;
  bool switch = true;
  while(j < i && switch) {
   if(W[i][j] && vcolor[i] == vcolor[j])
     swicth = false;
   j++;
  }
  return switch;
}
```

State-Space Tree (Map coloring)







Graph coloring – algorithm 2 (1/2)

```
kColoring(i , c)
{
        if (valid(i, c)) then {
                color[i] ← c;
                if (i = n) then {return TRUE;}
                else {
                        result ← FALSE;
                        d ← 1;                          ▷ d: color
                        while (result = FALSE and d ≤ k) {
                                result ← kColoring(i+1, d);
                                d++;
                        }
                }
                return result;
        } else {return FALSE;}
}
```

Graph coloring – algorithm 2 (2/2)

```
valid(i, c)
{
        for j ← 1 to i-1 {

                if ((i, j) ∈ E and color[j] = c) then return FALSE;
        }
        return TRUE;
}
```

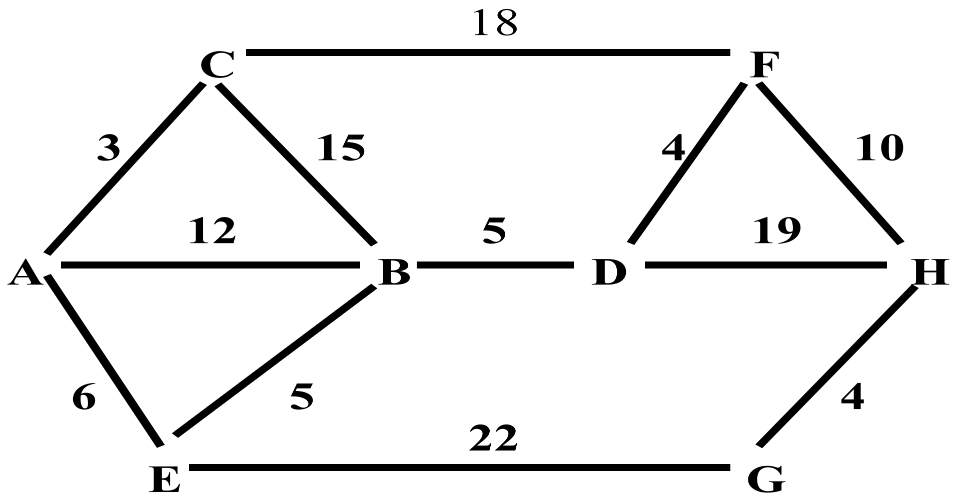
Hamiltonian Circuits Problem

- Hamiltonian circuit (tour) of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.

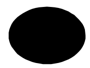
State Space Tree

- Put the starting vertex at level 0 in the tree
- At level 1, create a child node for the root node for each remaining vertex that is adjacent to the first vertex.
- At each node in level 2, create a child node for each of the adjacent vertices that are not in the path from the root to this vertex, and so on.

Example

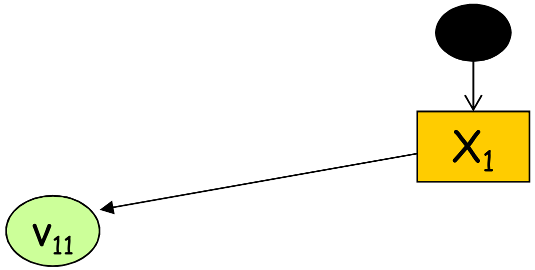


Backtracking Search



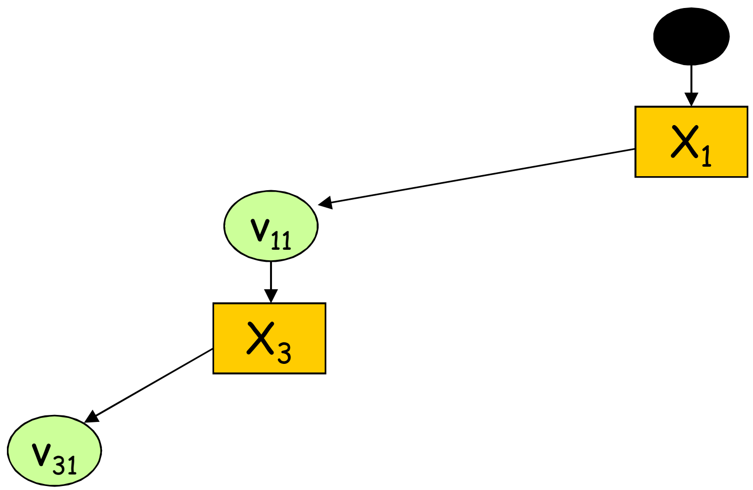
Assignment = {}

Backtracking Search (3 variables)



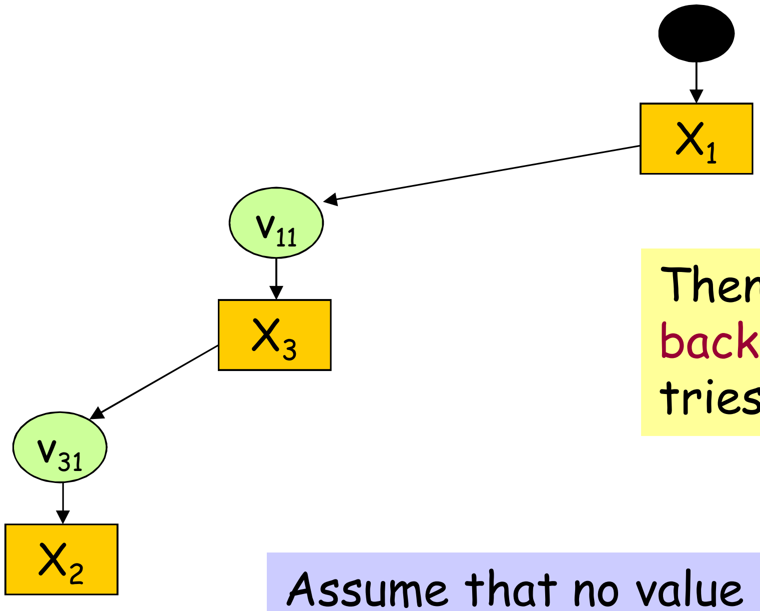
Assignment = $\{(X_1, v_{11})\}$

Backtracking Search (3 variables)



Assignment = $\{(X_1, v_{11}), (X_3, v_{31})\}$

Backtracking Search (3 variables)

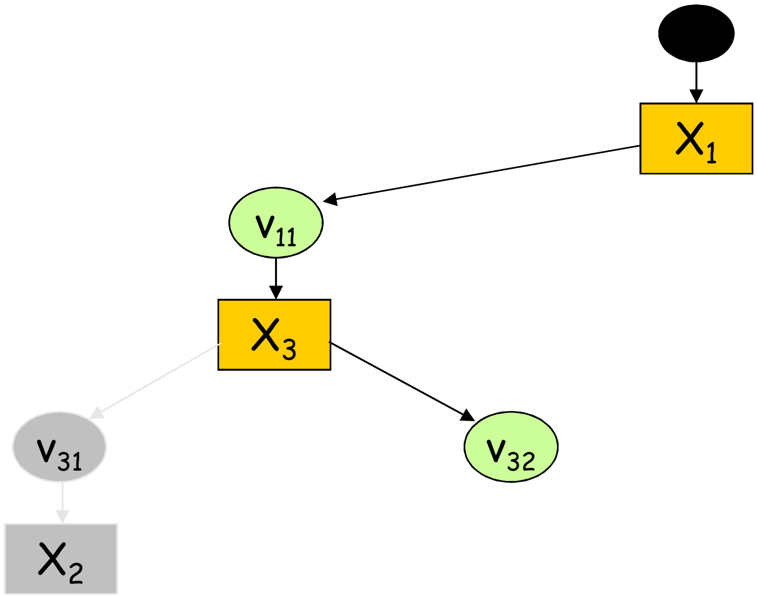


Then, the search algorithm backtracks to the previous variable and tries another value

Assume that no value of X_2 leads to a valid assignment

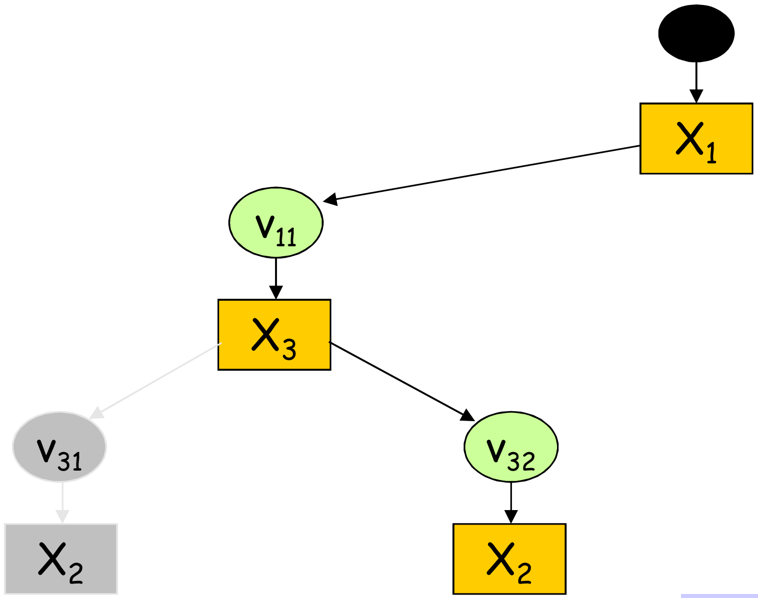
Assignment = $\{(X_1, v_{11}), (X_3, v_{31})\}$

Backtracking Search (3 variables)



Assignment = $\{(X_1,v_{11}), (X_3,v_{32})\}$

Backtracking Search (3 variables)

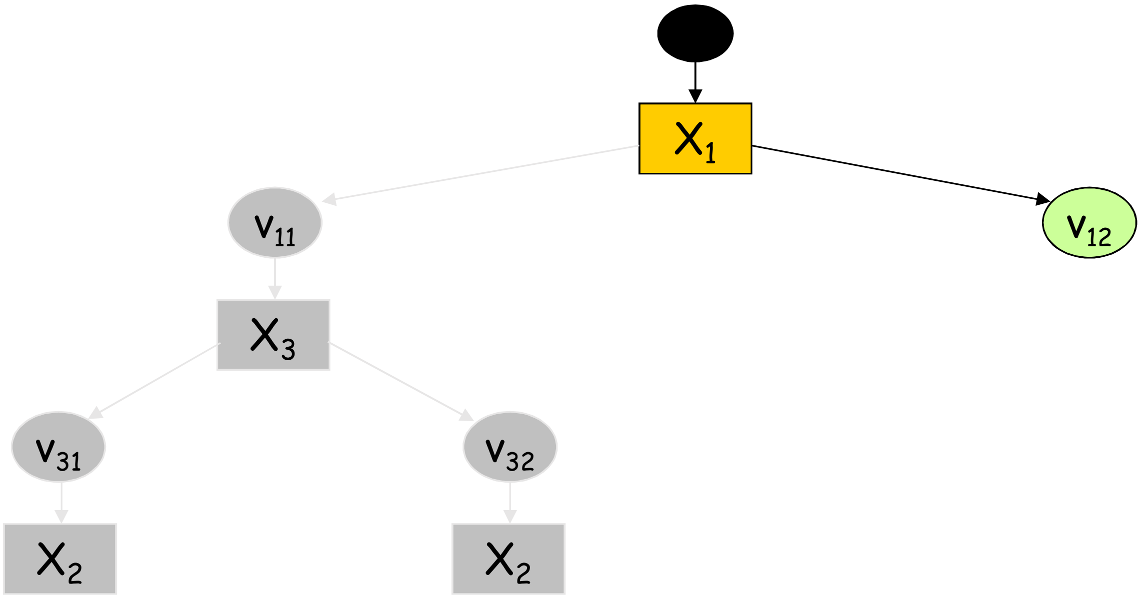


The search algorithm backtracks to the previous variable (X_3) and tries another value. But assume that X_3 has only two possible values. The algorithm backtracks to X_1

Assume again that no value of X_2 leads to a valid assignment

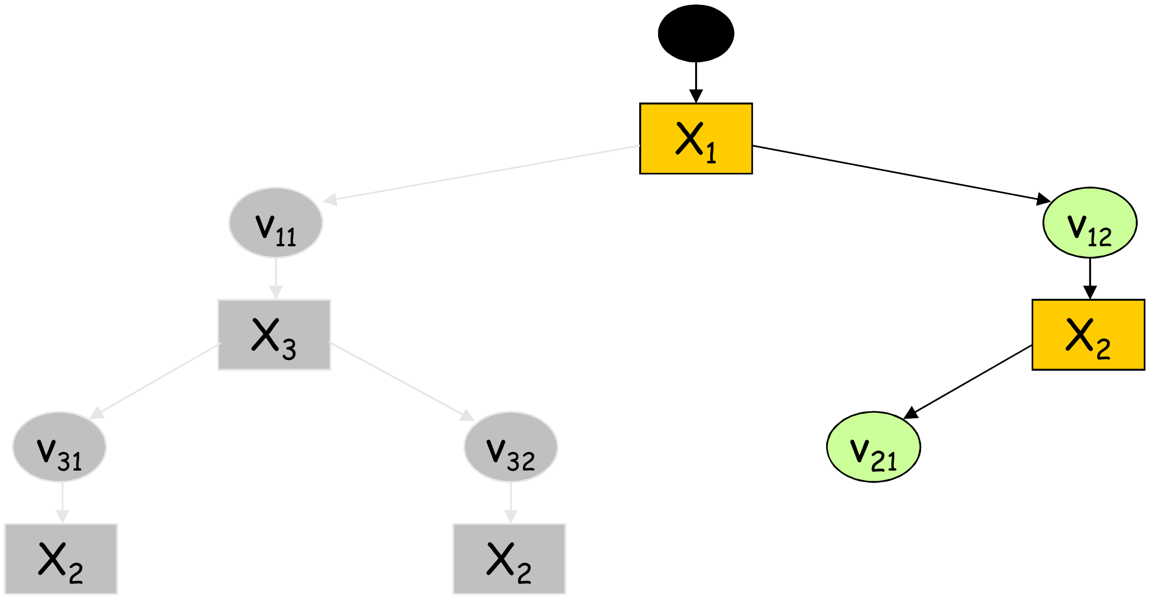
Assignment = $\{(X_1,v_{11}), (X_3,v_{32})\}$

Backtracking Search (3 variables)



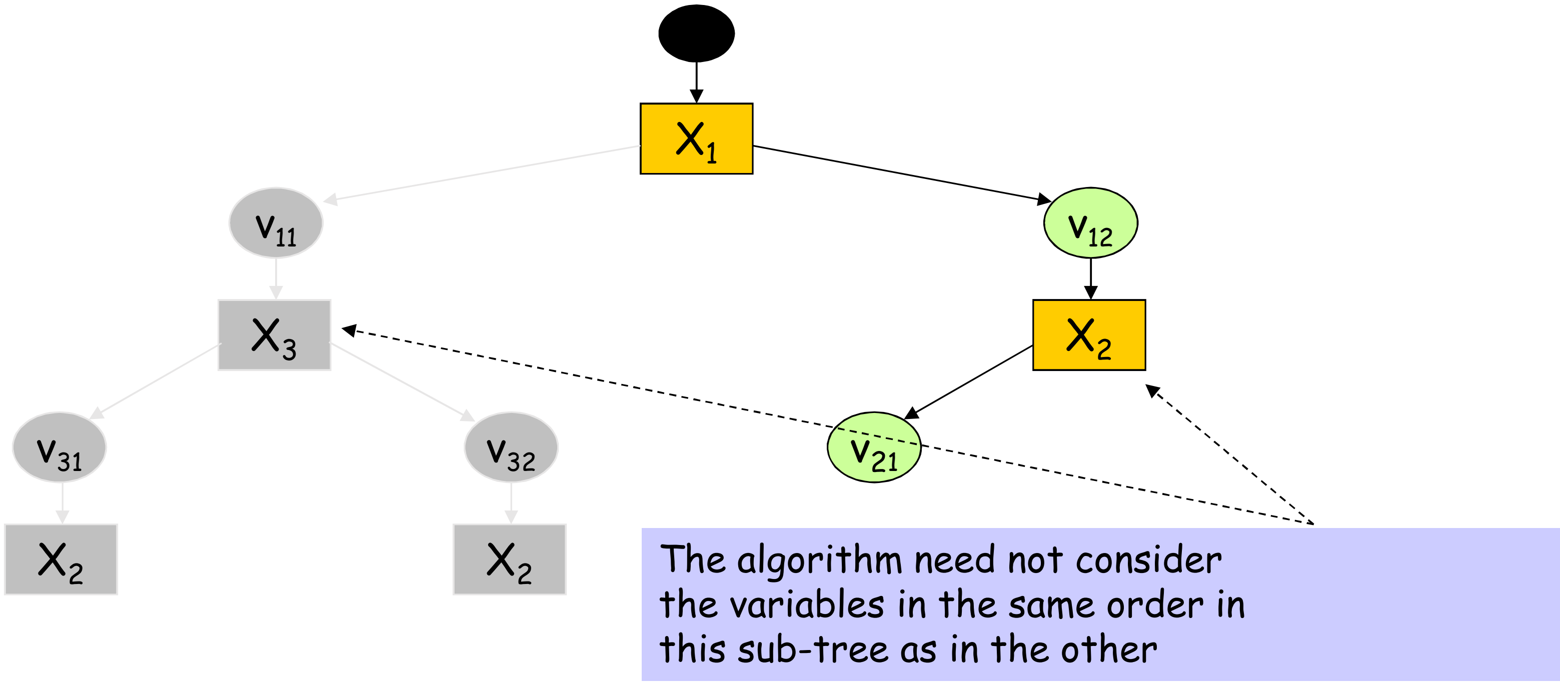
Assignment = $\{(X_1, v_{12})\}$

Backtracking Search (3 variables)



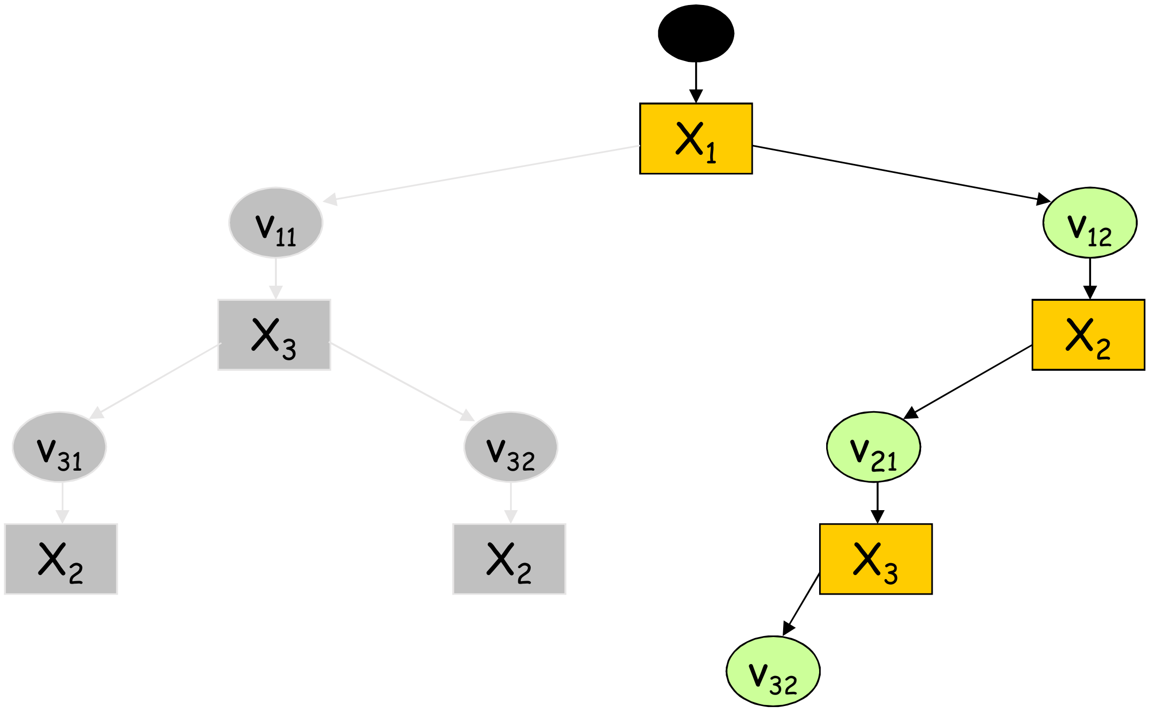
Assignment = $\{(X_1,v_{12}), (X_2,v_{21})\}$

Backtracking Search (3 variables)



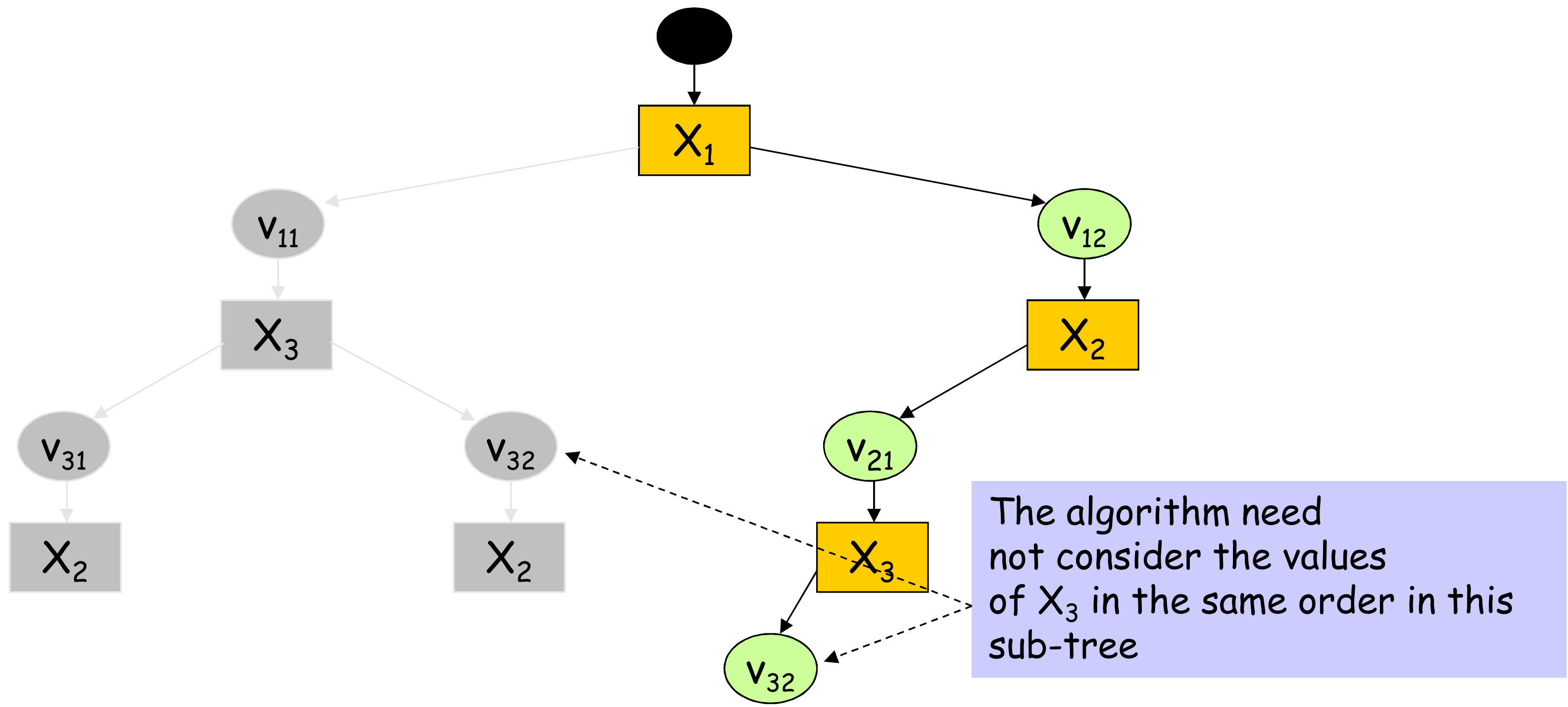
Assignment = $\{(X_1,v_{12}), (X_2,v_{21})\}$

Backtracking Search (3 variables)



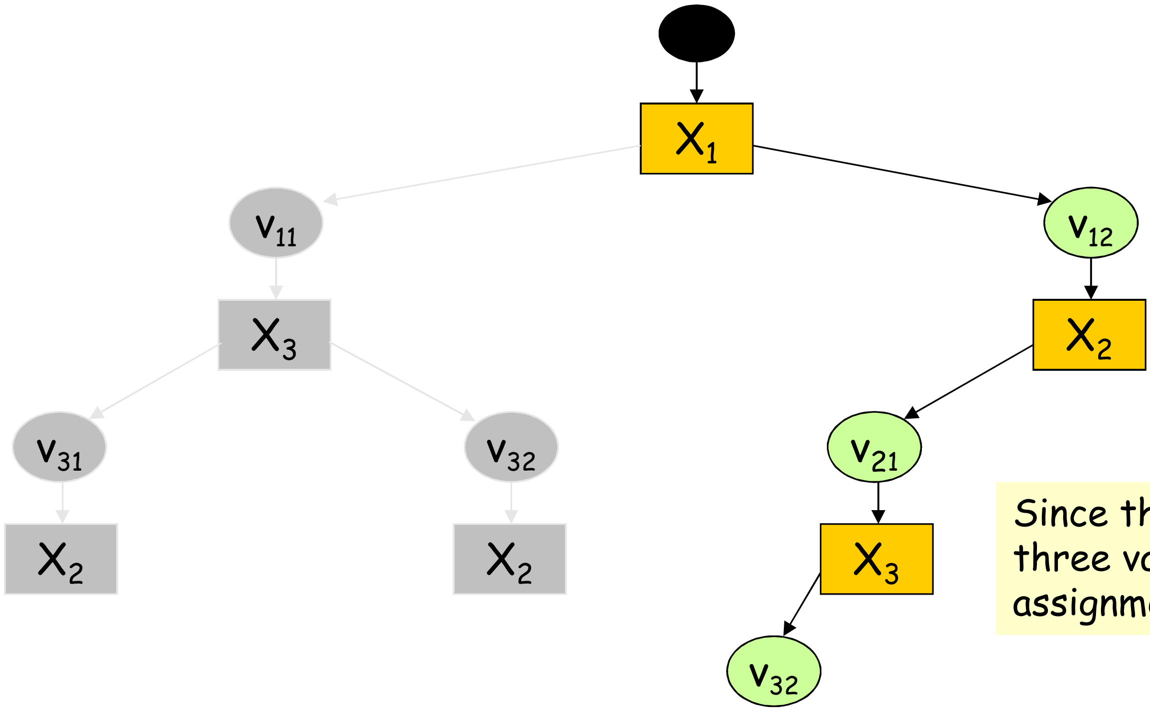
Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Backtracking Search (3 variables)



Assignment = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

Backtracking Search (3 variables)



Since there are only three variables, the assignment is complete

Assignment = $\{(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})\}$

Problem 19: two-color graph

- The *four-color theorem* states that every planar map can be colored using only four colors in such a way that no region is colored using the same color as a neighbor.
  - After being open for over 100 years, the theorem was proven in 1976 with the assistance of a computer. Here you are asked to solve a simpler problem.
  - Decide whether a given connected graph can be two-color graph, i.e., can the vertices be painted red and black such that no two adjacent vertices have the same color.
- To simplify the problem, you can assume the graph will be connected, undirected, and not contain self-loops (i.e., edges from a vertex to itself).

Problem 19: two-color graph

- Input
  - The first line contains the number of vertices . (1<n<30)
  - Each case starts with a line containing the number of vertices n, where 1<n<30.
  - Each vertex in labeled by a number from 0 to n-1.
  - After this, lines follow, each containing two vertex numbers specifying an edge. An input with $n = 0$ marks the end of the input and is not to be processed.
- Output
  - Decide whether the input graph can be 2-colored (bicolorable), and print the result as shown below.

Problem 19: two-color graph

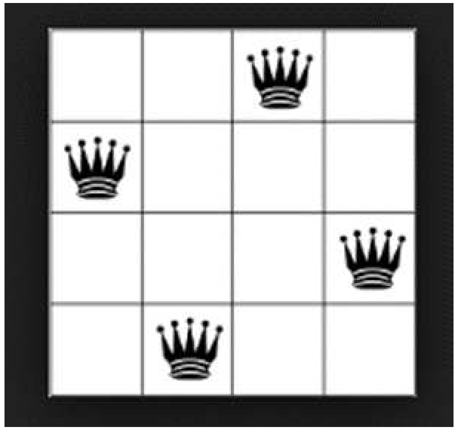
- Sample input
  - 3
  - 3
  - 0 1
  - 1 2
  - 2 0

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

- Sample output
  - ✓ not two-color
  - ✓ two-color

Problem 20: N-Queens Puzzle

- The N-Queens Puzzle is the problem of placing four chess queens on an n×n chessboard so that no two queens attack each other.
- The queen is the most powerful piece in the game of chess, able to move any number of squares vertically, horizontally, or diagonally.
- Thus, a solution requires that no two queens share the same row, column, or diagonal.
- The n queens' problem asks how many distinct ways there are to place n mutually non-attacking queens on an n×n chessboard.
- **Write a program to compute the total number of ways one can put the four queens on a chessboard so that no two of them are in attacking position.**



THANK YOU