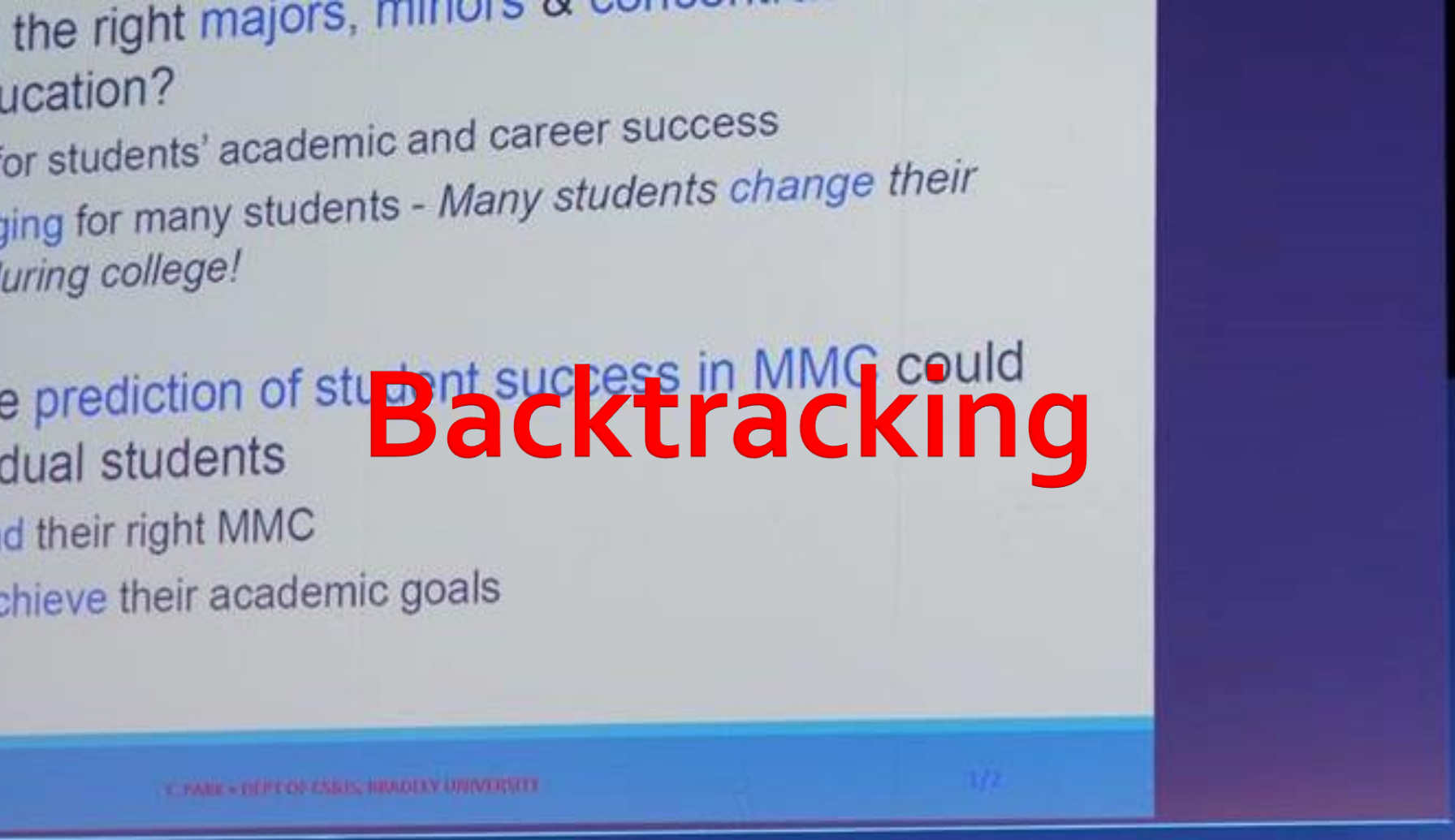
Backtracking



Prof. Young Park

✓ Two Hard Problems So Far

The 0-1 Knapsack Problem

- The **0-1 Knapsack problem** (optimization problem) solved so far.
  - Via Greedy?
    - No
  - Via DP!
    - Yes **O(nW)** or **O(2^n)**
  - Any better?
    - No one has ever found an algorithm whose worst-case time complexity is better than exponential!
    - Yet no one has proven that such an algorithm is not possible!

The Traveling Salesperson Problem

- The **Traveling Salesperson Problem** (optimization problem) solved so far.
  - Via Greedy?
    - No
  - Via DP!
    - Yes **$O(n^2 2^n)$**
  - Any better?
    - No one has ever found an algorithm whose worst-case time complexity is better than exponential!
    - Yet no one has proven that such an algorithm is not possible!

Hard Problems

- Then, what?
- **Still want to improve!**
- How?

  - **Backtracking**
  - **Branch-and-Bound** (for optimization problems)

✓ State Space Tree for A Problem

State Space Tree for A Problem

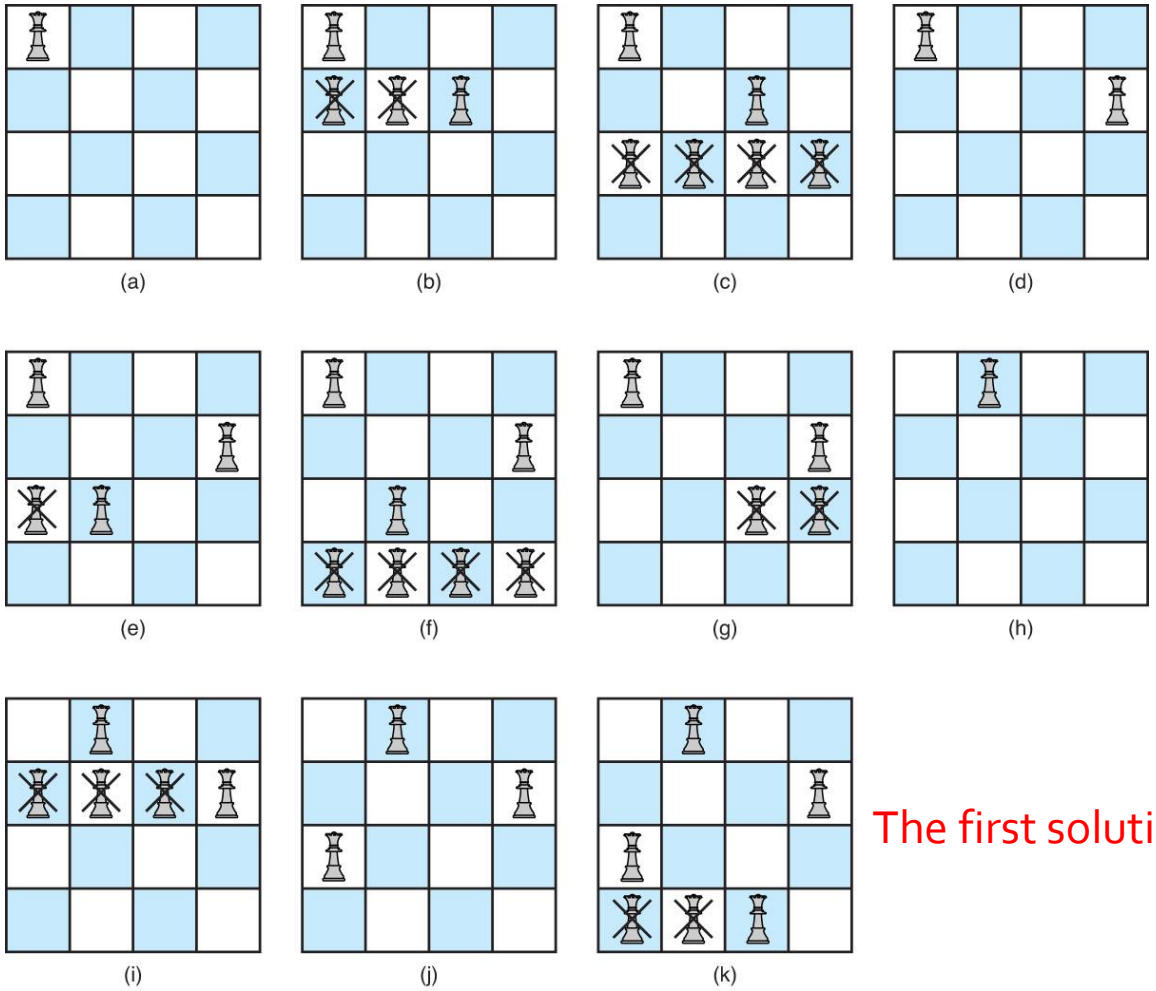
- Given a problem, a **state space** is the set of all possible candidate solutions to the problem.
- We can create the candidate solutions by constructing a virtual, implicit tree.
  - This tree is called a **State Space Tree**.

State Space Tree for A Problem

- In State Space Tree:
  - **Nodes:** partial solutions
  - **Edges:** choices in expanding partial solutions

The 4-Queens Problem

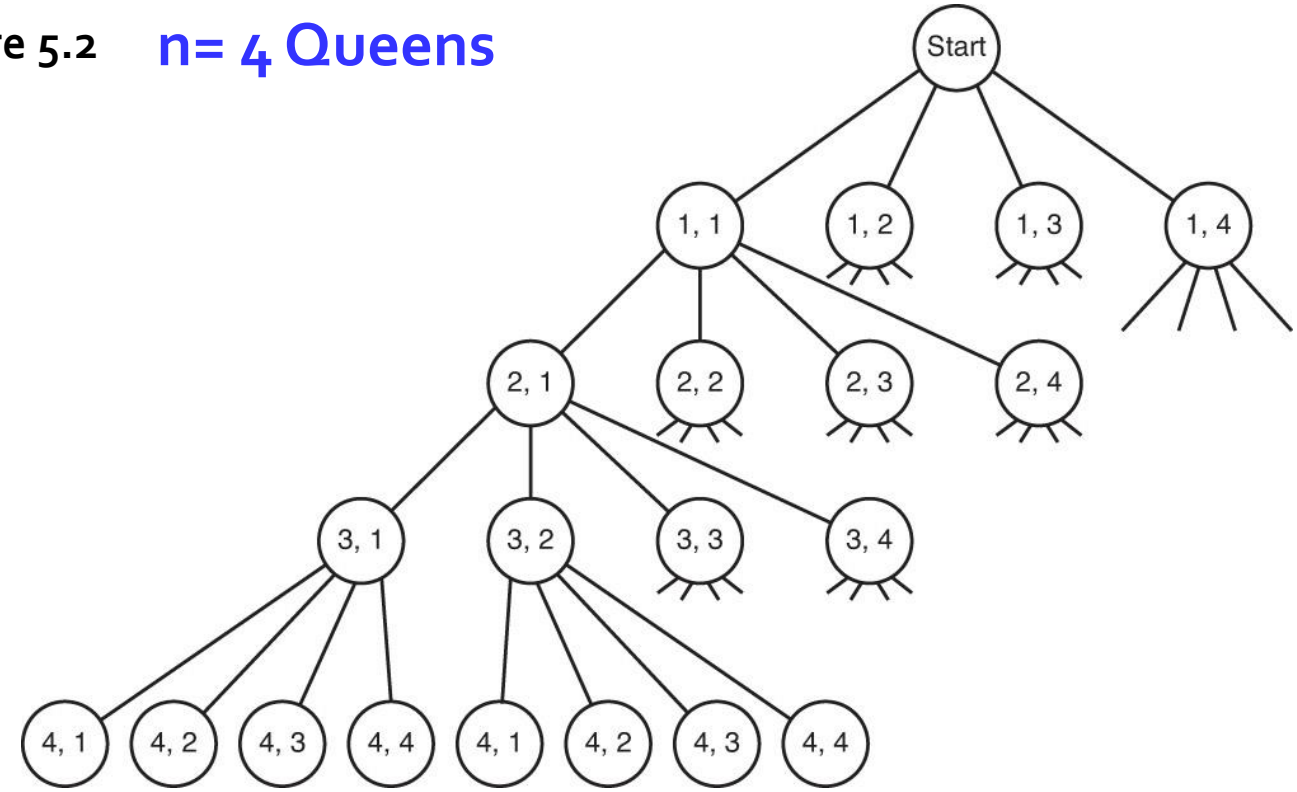
Figure 5.5



The first solution!

The 4-Queens Problem – The State Space Tree

Figure 5.2 **n= 4 Queens**



Total 341 nodes & *256 leaves candidate solutions*

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

Example A.4

▶QUIZ?

- State Space Tree?

✓Brute-Force Approach (BF)

Brute-Force Approach (BF)

- A straightforward approach to solving a problem

Brute-force Approach

- **Brute-Force Approach** examines **all** possible candidate solutions.
- **Exhaustive search** or **Generate-and-test** is simple to implement and will always find a solution if it exists.

Exhaustive Search

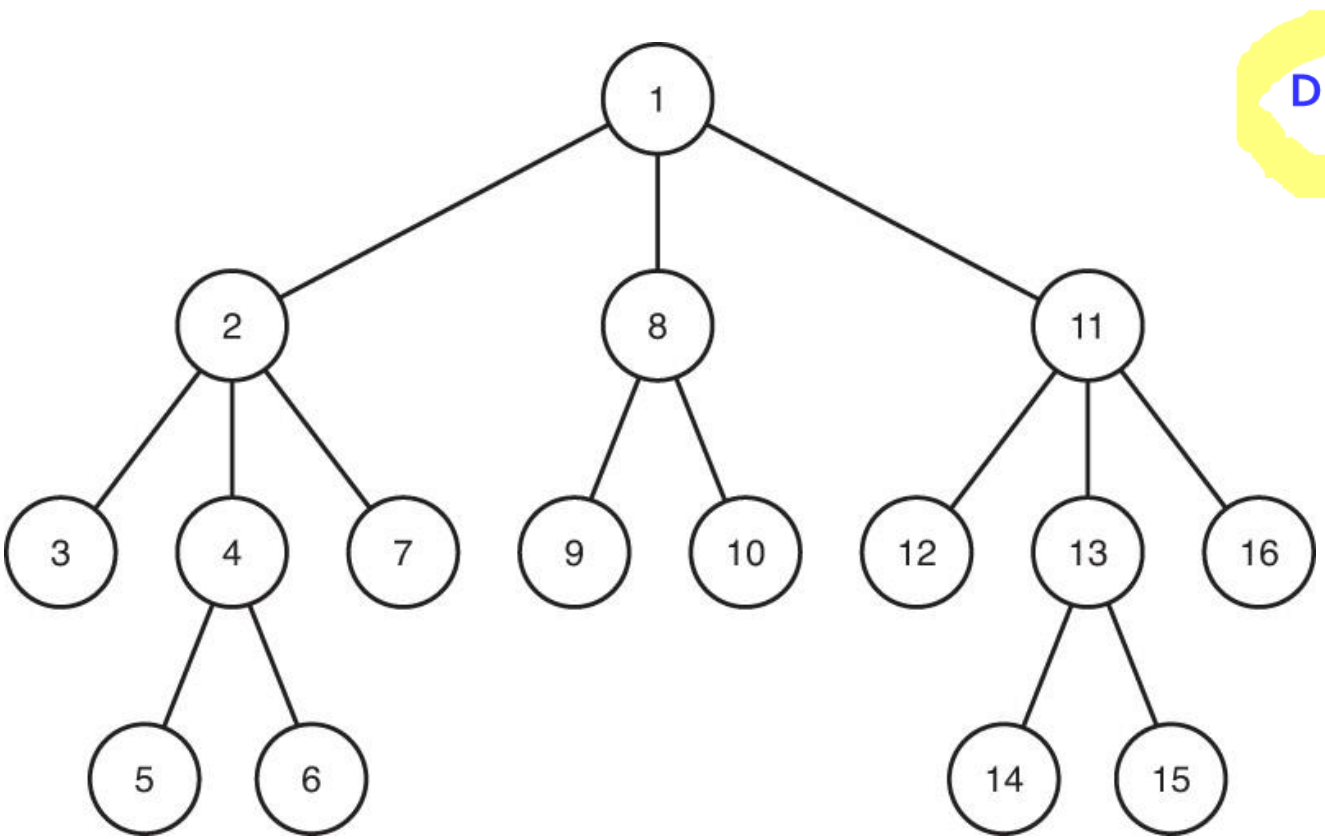
- Searches the **entire** state space tree:
  - **Depth-First Search**
    - Traverse "deeper" whenever possible (pre-order tree traversal)
  - **Breadth-First Search**
    - Traverse "wider" whenever possible (level-by-level tree traversal)

Depth-First Tree Search (DFS)


```
void depth_first_tree_search(node v)
{
        visit v;
        for ( each child u of v)

        depth_first_tree_search(u);
}
```

DFS



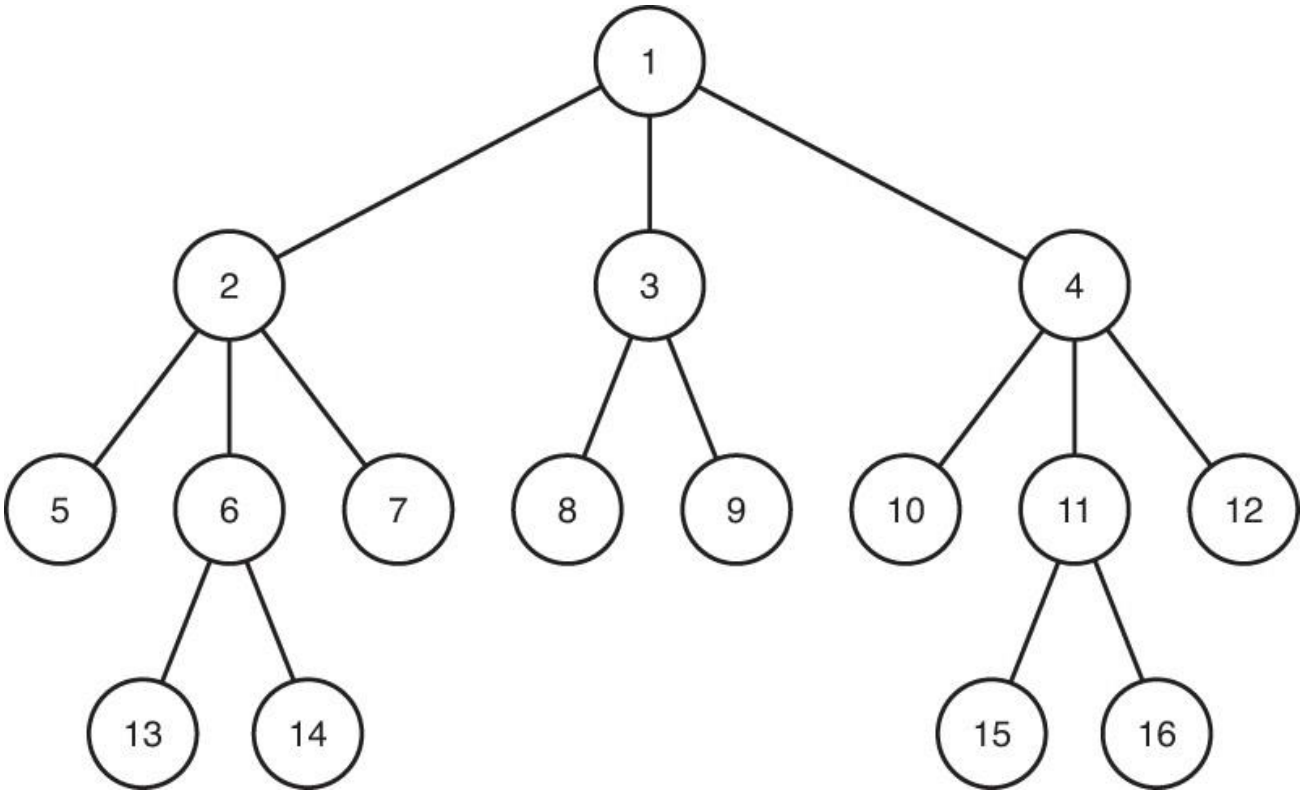
Breadth-First Tree Search (BFS)

BFS

```
void beadth_first_tree_search (tree T)
{
        queue_of_node Q;
        node u, v;
        initialize(Q); //initialize queue to be empty
        v = root of T;
        visit v;
        enqueue(Q, v);
        while (!empty(Q))
        {
                dequeue(Q, v);
                for (each child u of v)
                {
                        visit u;
                        enqueue(Q, u);
                }
        }

}
```

BFS



BFS

Brute-force Approach

- **Brute-force approach** examines all possible candidate solutions.
- Its cost is proportional to **the number of candidate solutions**.
  - Tends to grow very quickly as the size of the problem increases.
  - Inefficient in many practical problems!

▶QUIZ?

- Brute-Force Approach?

✓Backtracking

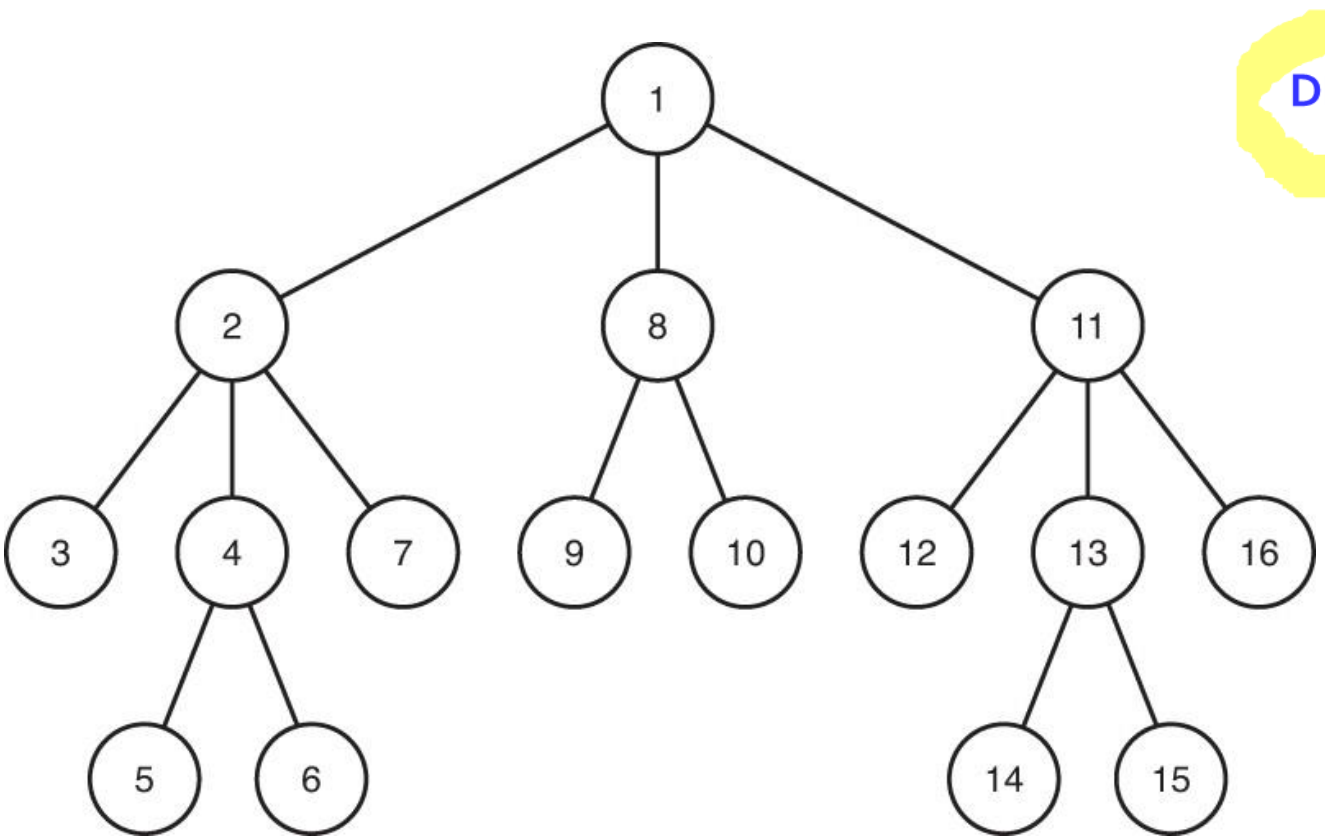
Backtracking

- An approach/paradigm to designing algorithms!

Backtracking

- **Backtracking** is a systematic way to go through a search space by traversing the state space using a **depth-first search with pruning**.
  - Reduces the search by cutting down some branches in the tree!

Backtracking - DFS



Backtracking

- **Backtracking** is a refinement of **the exhaustive brute force approach**.

- Backtracking gives a significant advantage over an exhaustive brute-force search of the state space tree for the average problem.

Backtracking

- **At worst-case**, backtracking tries every path, traversing the entire search space as in an exhaustive search.
- A very useful algorithm approach that can be used in many problems!

- Idea?

The Backtracking Technique

- **Backtracking** consists of
  - **Doing a depth-first search** of a state space tree,
  - **Checking** whether each node is **promising**, and
    - **If there is a potential that a solution might be found**, we call the node **promising**.
  - **Backtracking** **to** the node's parent **if the node is nonpromising**.

Backtracking - Non-Promising

- We call a node **nonpromising** if we can determine that **it cannot possibly lead to a solution**.

Recursive Backtracking

```
void checknode (node v)
{
        visit v;
        if ( promising(v) )
           if (there is a solution at v)
              write the solution;
           else
              for ( each child u of v)
                    checknode(u);
        else
           backtrack;
}
```

Pruned State Space Tree via Backtracking

- This process called **pruning** prunes the sub tree rooted at this nonpromising node.
- A subtree consisting of the visited nodes is called **the pruned state space tree** via backtracking.

Backtracking

- Backtracking is used to solve problems in which a sequence of objects is selected from a specified set so that the sequence satisfies some criterion.
- What Problems?
  - **Find a solution?**
  - **Find the number of all solutions?**
  - **Find all solutions?**
  - **Find the optimal solution?**

▶QUIZ?

- Backtracking?

▶QUIZ?

- Compare **Brute-Force Approach** vs. **Backtracking**?

✓ Backtracking-based Algorithms

Backtracking-based Algorithms

1. The N-Queens Problem via Backtracking
2. The Graph m-Coloring Problem via Backtracking

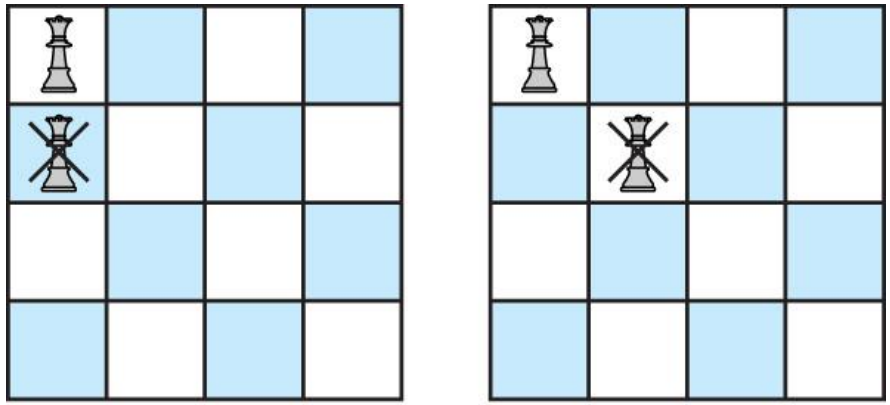
3. The 0-1 Knapsack Problem via Backtracking (Optimization Problem)

1. The n-Queens Problem

- Position *n* queens on an *n* x *n* chessboard so that no two queens threaten each other.
  - The criteria is that **no two queens can be in the same row, column, or diagonal**.

The 4-Queens Problem

- The task is to position four queens on a 4 x 4 chessboard so that no two queens threaten each other.



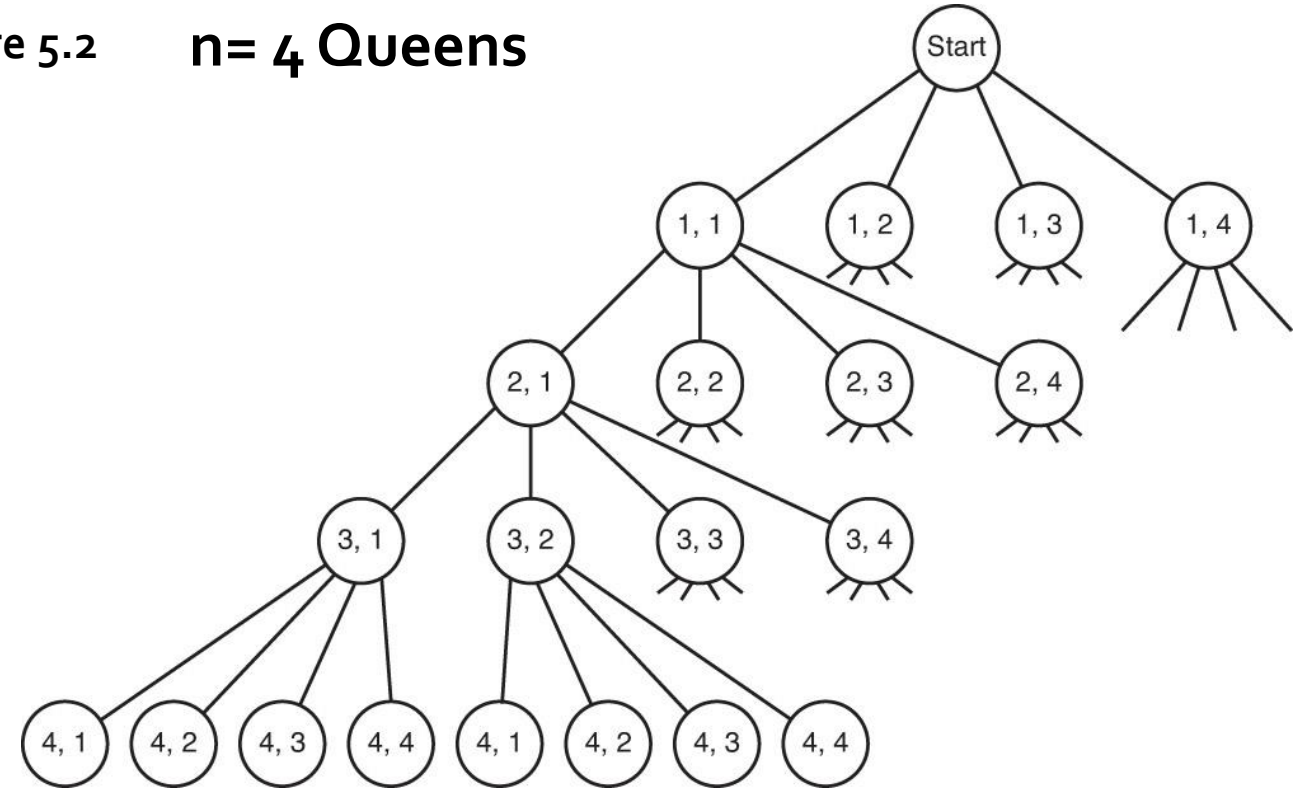
(a) (b)

The n-Queens Problem - The State Space Tree

- We can create the candidate solutions by constructing **a state space tree** in which
  - The column choices for the first queen (in row 1) are stored in level-1 nodes in the tree,
  - The column choices for the second queen are in level-2 nodes,
  - and so on.. .
- A path from the root to the leaf is a candidate solution.

The 4-Queens Problem – The State Space Tree

Figure 5.2 **n= 4 Queens**



Total 341 nodes & *256 leaves candidate solutions*

The 4-Queens Problem via **Brute-force Approach**

- **Brute-Force Approach** examines **341 nodes** and all possible **256 candidate solutions**.
- The number of total nodes in the state space tree of the **n-queen** problem = $n^{n+1} - 1 / n - 1$
- $O(n^n)$

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

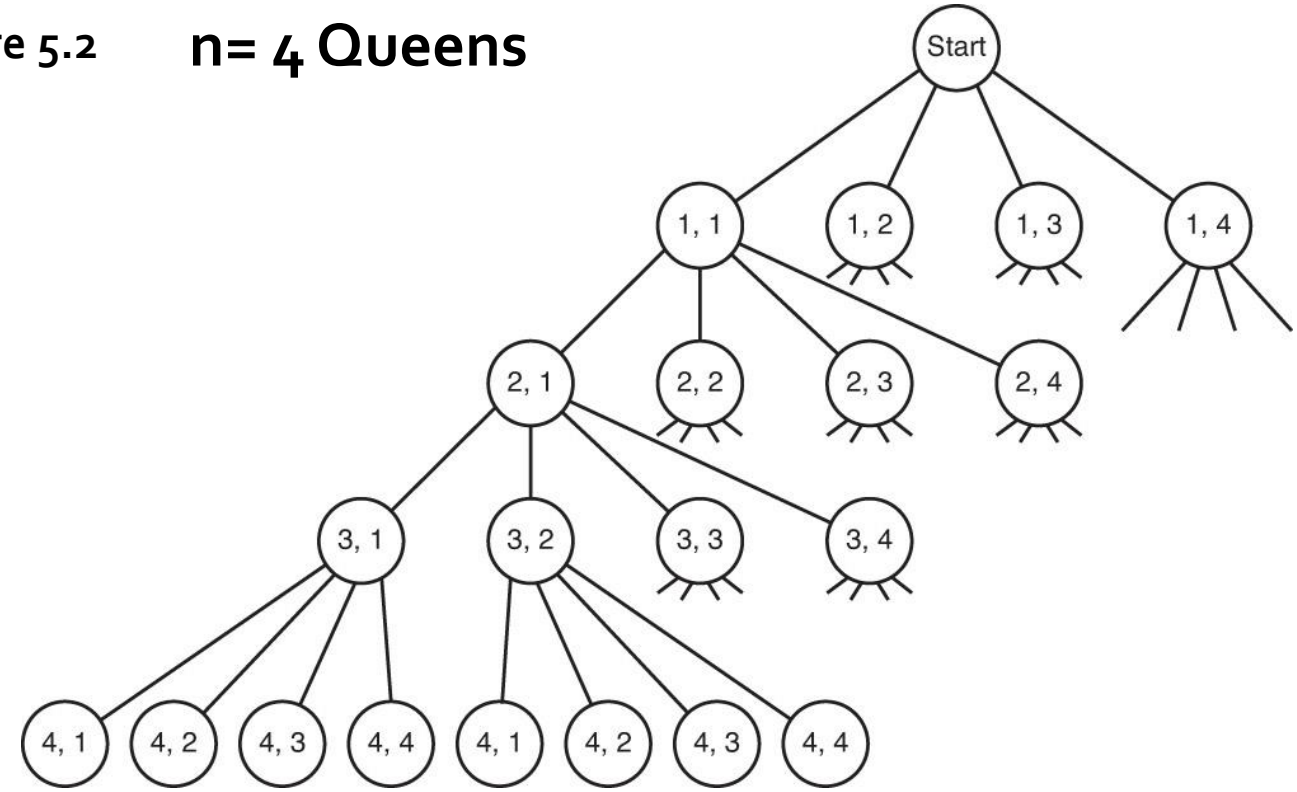
Example A.4

The n-Queens Problem via Backtracking

- *First*, we must visualize a **state space tree** for our solution space.
  - The column choices for the first queen (in row 1) are stored in level-1 nodes in the tree,
  - The column choices for the second queen are in level-2 nodes,
  - and so on.. .

The 4-Queens Problem – The State Space Tree

Figure 5.2 **n= 4 Queens**



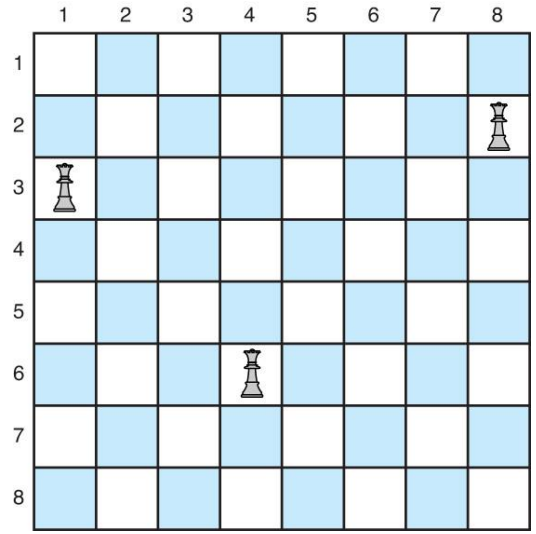
Total 341 nodes & 256 leaves candidate solutions

The n-Queens Problem via Backtracking

- *Second*, we must come up with ways of eliminating branches before we actually traverse them - **nonpromising**.
  - Two queens in the same column.
  - Two queens on the same diagonal.
  - Using these two rules, we can eliminate quite a bit of the solution space!

The n-Queens Problem via Backtracking

- col (i) = the column where the queen in the ith row is located.
  - col (6) - col (3) = 4 - 1 = 3 = 6 - 3
  - col (6) - col (2) = 4 - 8 = -4 = 2 - 6



The n-Queens Problem via Backtracking

- A node is **nonpromising** if
  - The queen in the kth row threatens the queen in the ith row along the **same column**,
    - **col (i) = col (k)**
  - The queen in the kth row threatens the queen in the ith row along one of its **diagonals**,
    - **col (i) - col (k) = i - k**
    - **col (i) - col (k) = k - i**

The n-Queens Problem via Backtracking

- *Third*, we do **backtracking** - When we identify a node as nonpromising, we go back to the parent and try another branch.

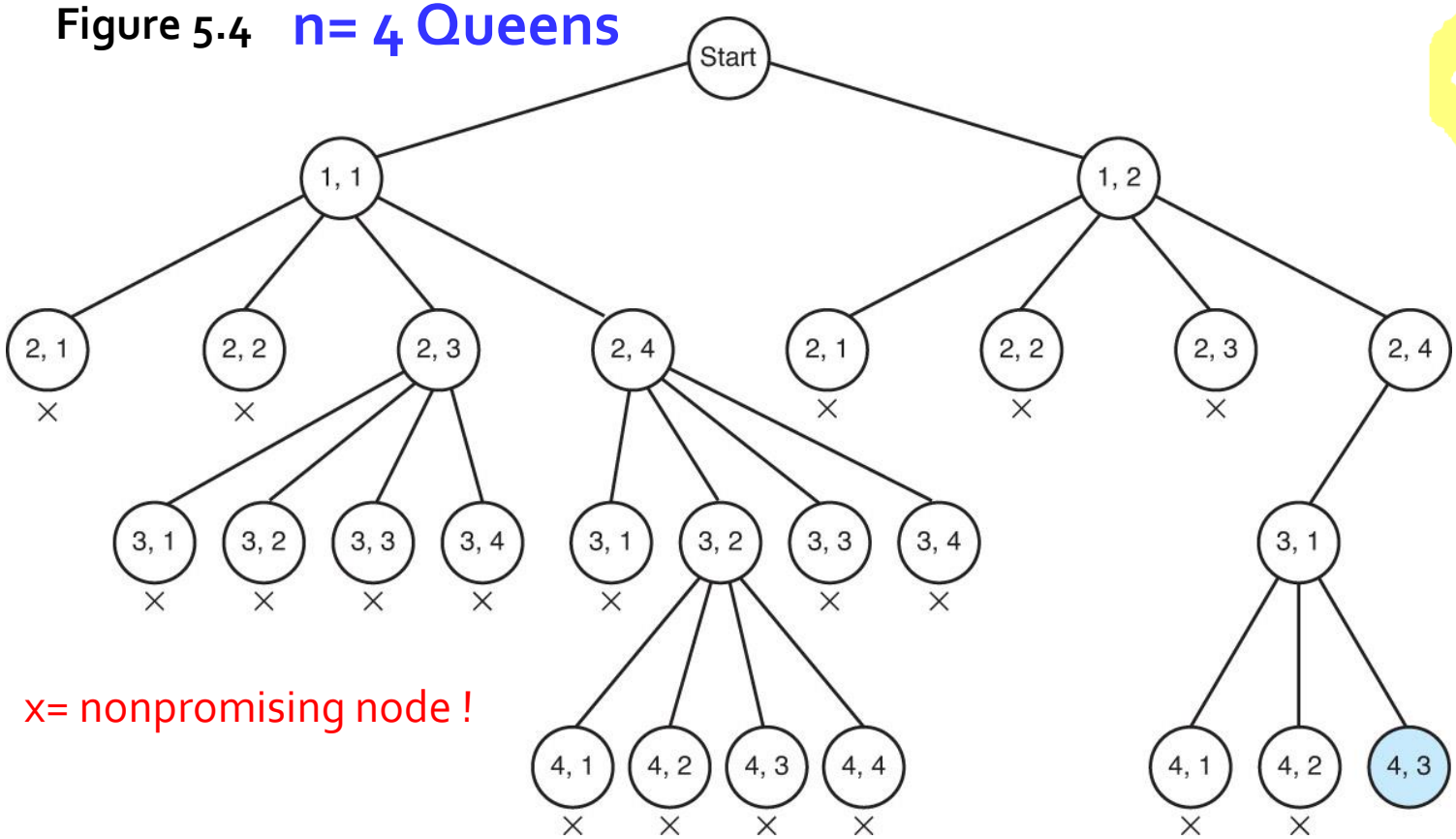
- Eventually we get to any leaf node, we have our solution.

The 4-Queens Problem – The Pruned State Space Tree via Backtracking

Total 341 nodes & 256 leaves candidate solutions

Figure 5.4 **n= 4 Queens**

DFS



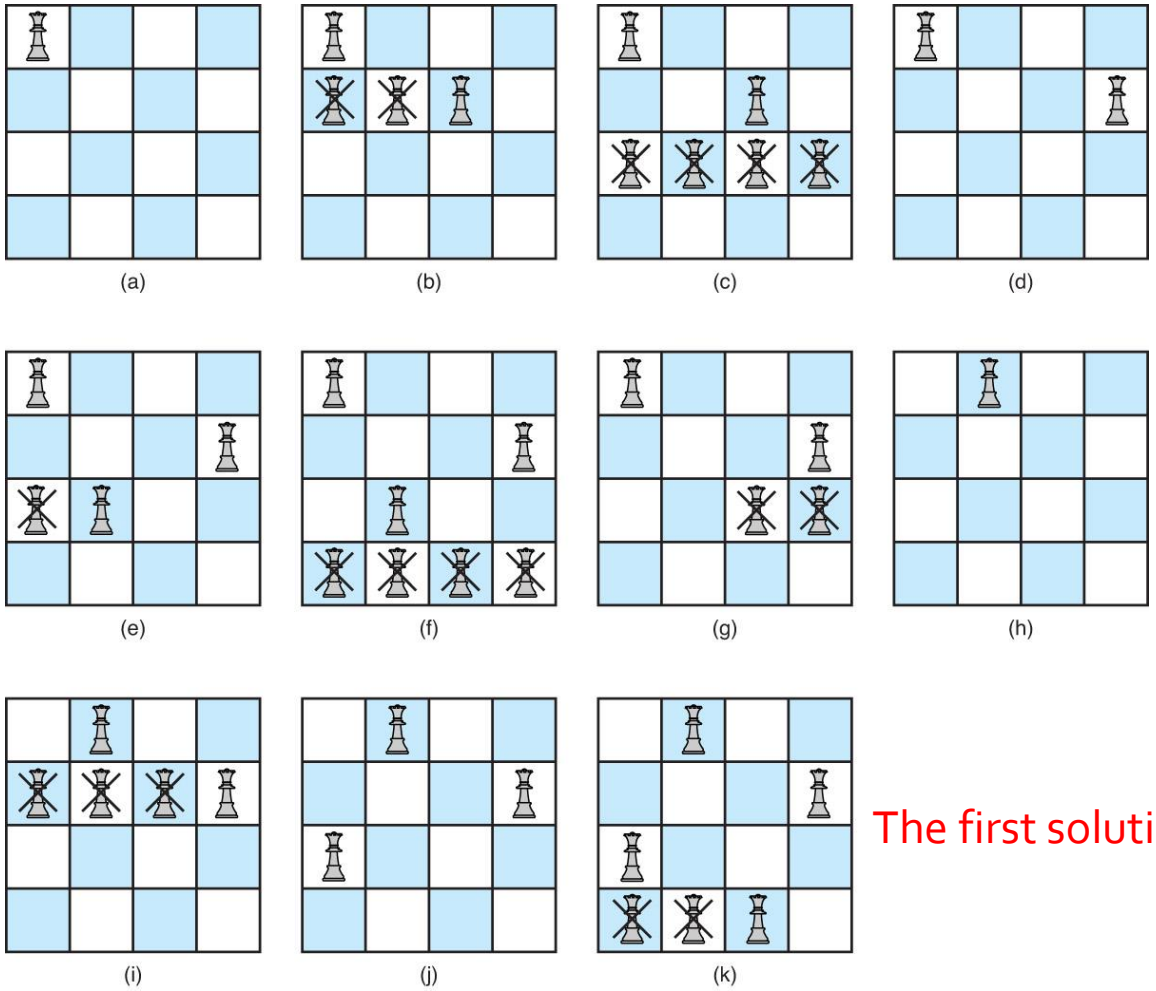
x= nonpromising node !

The first solution!

27 nodes checked!

The 4-Queens Problem via Backtracking

Figure 5.5



The first solution!

The n-Queens Problem via Backtracking

- **Algorithm 5.1** The Backtracking Algorithm for the n-Queens Problem
- The number of total nodes in the state space tree of the **n-queen** problem = $\mathbf{n^{n+1} - 1 / n - 1}$

- **$O(n^n)$** at worst-case!

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

Example A.4

The n-Queens Problem

- n=1: 1-Queen
  - 1
- n=2: 2-Queens
  - 0
- n=3: 3-Queens
  - 0
- n=4: 4-Queens
  - 1 unique & 2 distinct

The n-Queens Problem

- n=5: 5-Queens
  - 2 & 10
- n=6: 6-Queens
  - 1 & 4
- n=7: 7-Queens
- n=8: 8-Queens (the instance using a standard chessboard)
  - 1 2 & 92
- …

The n-Queens Problem via Backtracking

- Analysis of Algorithm 5.1 The Backtracking Algorithm for the n-Queens Problem
  - **Table 5.1**

▶QUIZ? The 3-Queens Problem via Backtracking

- n=3: **3-Queens** using Backtracking? Find the **first** solution?

▶QUIZ? The 4-Queens Problem via Backtracking

- n=4: **4-Queens** using Backtracking? Find the **second** solution?

▶QUIZ? The 5-Queens Problem via Backtracking

- Show the **first two solutions** to the **5-Queens** problem using the Backtracking algorithm.

The n-Queens Problem via Backtracking Visualization

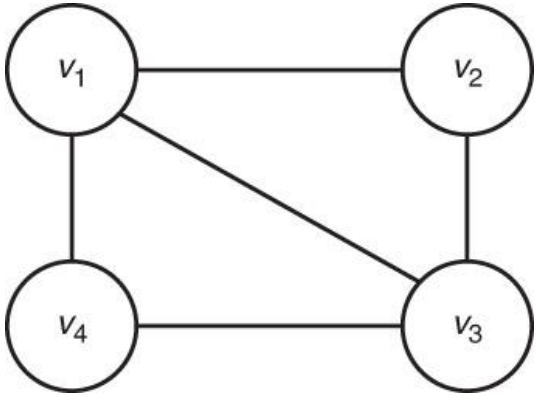
- ***The n-Queens Problem via Backtracking Visualization***

2. The m-Coloring Problem for Graphs

- Find all ways to color an undirected graph using at most **m different colors**, so that **no two adjacent vertices are the same color**.

The m-Coloring Problem for Graphs

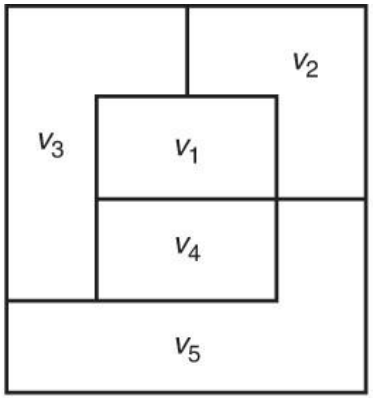
- Example 5.5
  - The m=2-coloring problem?
    - 0
  - The m=3-coloring problem?
    - 6

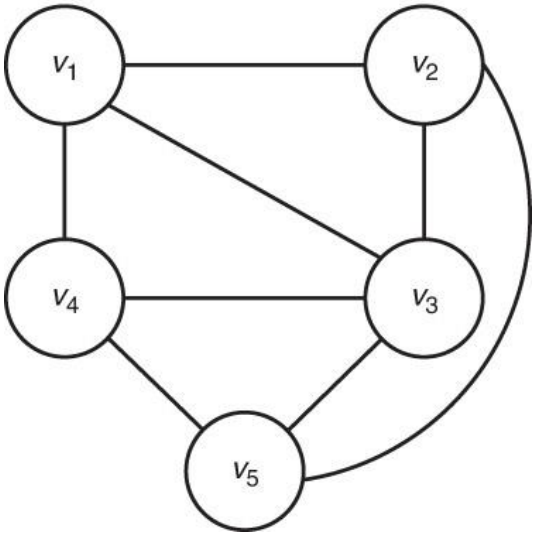


The m-Coloring Problem for Map - Planar Graph

- An important application of graph coloring is the coloring of maps.
- A graph is called planar if it can be drawn in a plane in such a way that no two edges cross each other.
- To every map there corresponds a planar graph.
  - Each region of the map is represented by a vertex.
  - If one region is adjacent to another region, we join their corresponding vertices by an edge.

Map and Planar Graph





The m-Coloring Problem for Planar Graph

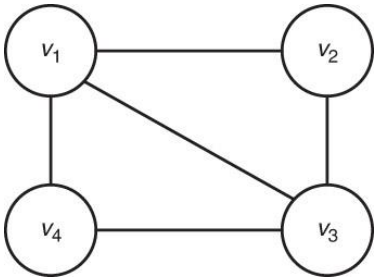
- The m-Coloring problem for planar graphs is to determine how many ways the map can be colored, so that **no two adjacent regions are the same color**.

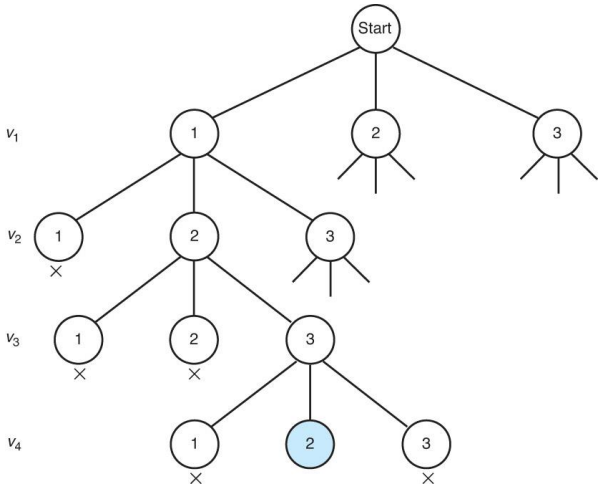
The m-Coloring Problem - The State Space Tree

- We can create the candidate solutions by constructing **a state space tree** for the m-coloring problem for planar graphs.
  - **G=(V,E)**
  - **m**

The 3-Coloring Problem - The State Space Tree

- The **state space tree**?





m=3 colors
n= 4 vertices

Total 121 nodes & 81 leaves candidate solutions

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

Example A.4

The 3-Coloring Problem via **Brute-force Approach**

- **Brute-Force Approach** examines 121 nodes and all possible 81 candidate solutions.
- The number of total nodes in the state space tree for **n** vertices using **m** colors = $\mathbf{m^{n+1} - 1 / m - 1}$
- $\mathbf{O(m^n)}$

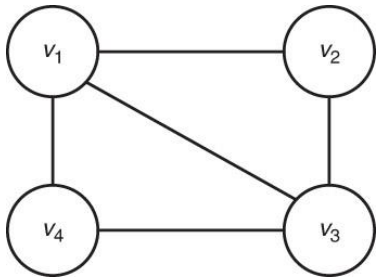
$$\sum_{i=0,n} m^i = m^{(n+1)} - 1 / m - 1$$

Example A.4

The m-Coloring Problem via Backtracking

- A node is **nonpromising** if
  - A vertex that is adjacent to the vertex being colored at the node has already been colored the same color that is being used at that node.

The 3-Coloring Problem The State Space Tree Pruned via Backtracking



m=3 colors
n= 4 vertices

Total 121 nodes & 81 leaves candidate solutions

DFS

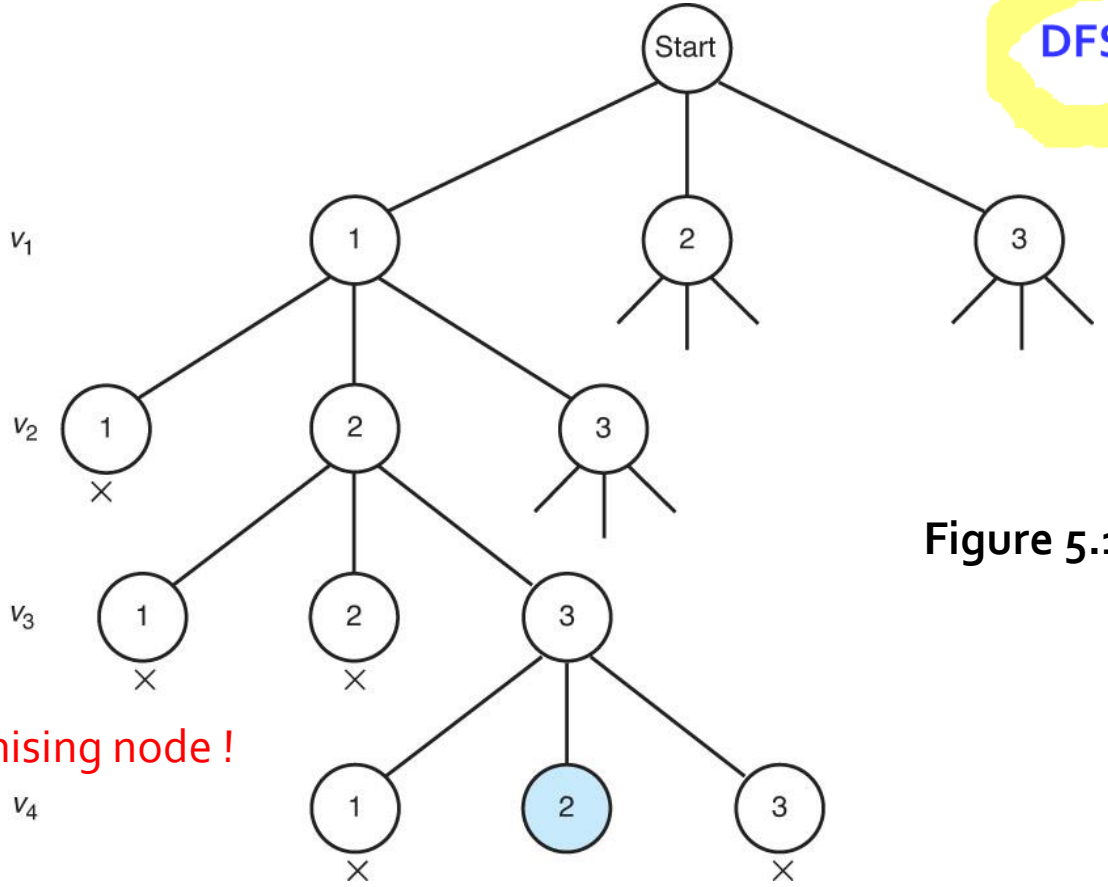


Figure 5.12

x= nonpromising node !

The first solution!

9 nodes checked!

The m-Coloring Problem via Backtracking

- **Algorithm 5.5** The Backtracking Algorithm for the m-Coloring Problem
- The number of total nodes in the state space tree for **n** vertices using **m** colors =

  $\mathbf{m^{n+1} - 1 / m - 1}$

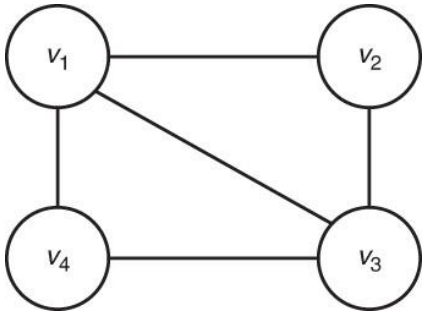
- $\mathbf{O(m^n)}$ at worst-case!

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

Example A.4

▶QUIZ? The 3-Coloring Problem via Backtracking

- **3-coloring** using Backtracking? Find the **second** solution?



3. The 0-1 Knapsack Problem

- We solved **the 0-1 Knapsack Problem** using Dynamic Programming.
  - **$O(2^n)$** at worst-case

The 0-1 Knapsack problem

- The 0-1 Knapsack problem is an optimization problem.
- No polynomial time algorithm is known for the 0-1 Knapsack problem.
- Nobody has shown that a polynomial time algorithm is not possible for the 0-1 Knapsack problem.

Backtracking to Optimization Problem

- We can apply Backtracking to the 0-1 Knapsack problem!
  - **Applying Backtracking to Optimization Problem!**

▶QUIZ? The 0-1 Knapsack Problem

- **n= 4 & W = 16 lb**
- Item1= $40 & 2 lb
- Item2= $30 & 5 lb
- Item3= $50 & 10 lb
- Item4= $10 & 5 lb

- Solution?
  - **Item 1 & item 3 (Max profit= $90)**

The 0-1 Knapsack Problem via Backtracking

- We build a **state space tree** for this problem.
  - Sort items by profit/weight.
  - Choosing the left child of a node means that you include an item, and
  - Choosing the right child means that you do exclude an item.
  - We then go to node at level 1 continue same way.

▶QUIZ? The 0-1 Knapsack Problem via Backtracking

- **n= 4 & W = 16 lb**
- Item1= $40 & 2 lb
- Item2= $30 & 5 lb
- Item3= $50 & 10 lb
- Item4= $10 & 5 lb

Sort by profit/weight

- **The state space tree?**

$\sum_{i=0,n} 2^i = 2^{(n+1)} - 1$

Example A.3

Total 31 nodes

The 0-1 Knapsack Problem via Backtracking

- This problem is an **optimization problem**.
  - This means that we do not know if a node contains a solution until the search is over.
  - We have to find **all** solutions, but we also have to **keep track of which one is best**.

- Idea?
  - **Backtracking for Optimization Problems**

Backtracking for Optimization Problems

Backtracking for Optimization Problems

```
void checknode (node v)
{
        visit v;
        if ( value(v) is better than best) best= value(v);
        if ( promising(v) )
            for ( each child u of v)
                checknode(u);
        else
            backtrack;
}
```

- best has the value of the best solution found so far.
- value (v) is the value of the solution at the node v.

The 0-1 Knapsack Problem via Backtracking

- best?
  - **Maxprofit** is the value of **the profit** in the **best solution** found **so far**!
- value(v)?
  - **Profit (v)** is the value of **the profit** in the **node v**.

The 0-1 Knapsack Problem via Backtracking

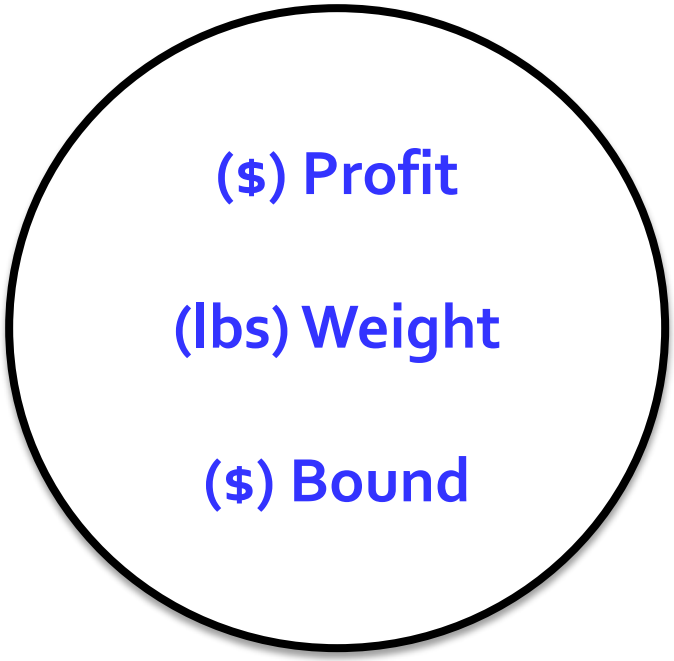
- **Weight (v)** is the **total weight** of the items in the **node v**.
- **Bound(v)?**

The 0-1 Knapsack Problem via Backtracking

- **Bound(v) (Potential profit upper bound)** is an **upper bound** on the **profit** we could achieve by expanding **beyond the node v**!
- **Pretending the Fractional Knapsack!**
- We will use bound(v) to determine whether v is **non-promising**!

The 0-1 Knapsack Problem via Backtracking

- Each node consists of



The 0-1 Knapsack Problem via Backtracking

- A node is **nonpromising** if
  1. The **total weight** thus far is greater than or equal to W.
  2. The **bound** (potential profit upper bond) is **smaller than or equal to** the value of **maxprofit**.

The 0-1 Knapsack Problem via Backtracking

- Example 5.6
  - **n= 4 & W = 16 lb**
  - Item1= $40 & 2 lb
  - Item2= $30 & 5 lb
  - Item3= $50 & 10 lb
  - Item4= $10 & 5 lb

Sort by profit/weight

  - **Item 1 & item 3 (Max profit= $90)**

The 0-1 Knapsack Problem State Space Tree

- **n= 4 & W = 16 lb**
- Item1= $40 & 2 lb
- Item2= $30 & 5 lb
- Item3= $50 & 10 lb
- Item4= $10 & 5 lb

Sort by profit/weight

Total 31 nodes

The 0-1 Knapsack Problem –
The State Space Tree Pruned via Backtracking

Figure 5.14

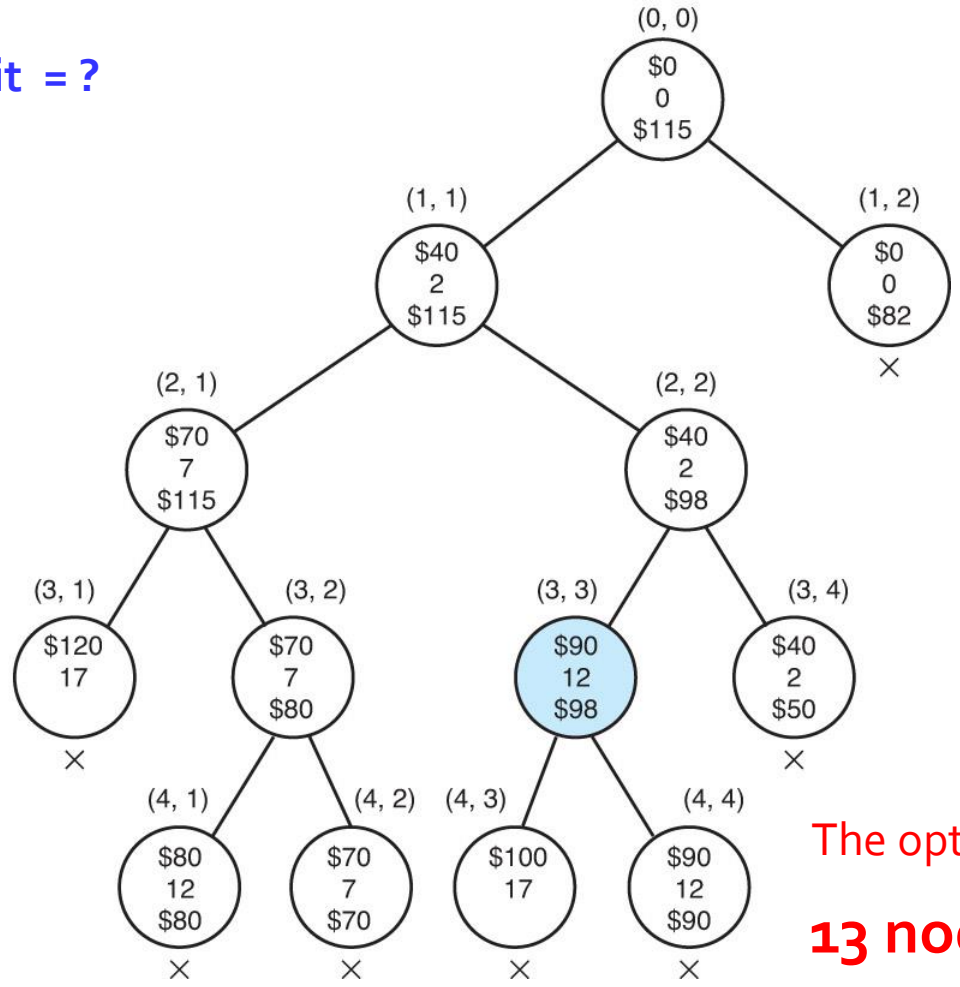
W=16

Maxprofit = ?

Total 31 nodes

DFS

Item 1 [$40, 2]
Item 2 [$30, 5]
Item 3 [$50, 10]
Item 4 [$10, 5]



Profit per unit weight!

The optimal solution!

13 nodes checked!

x= nonpromising node !

The 0-1 Knapsack Problem –
The State Space Tree Pruned via Backtracking

Figure 5.14

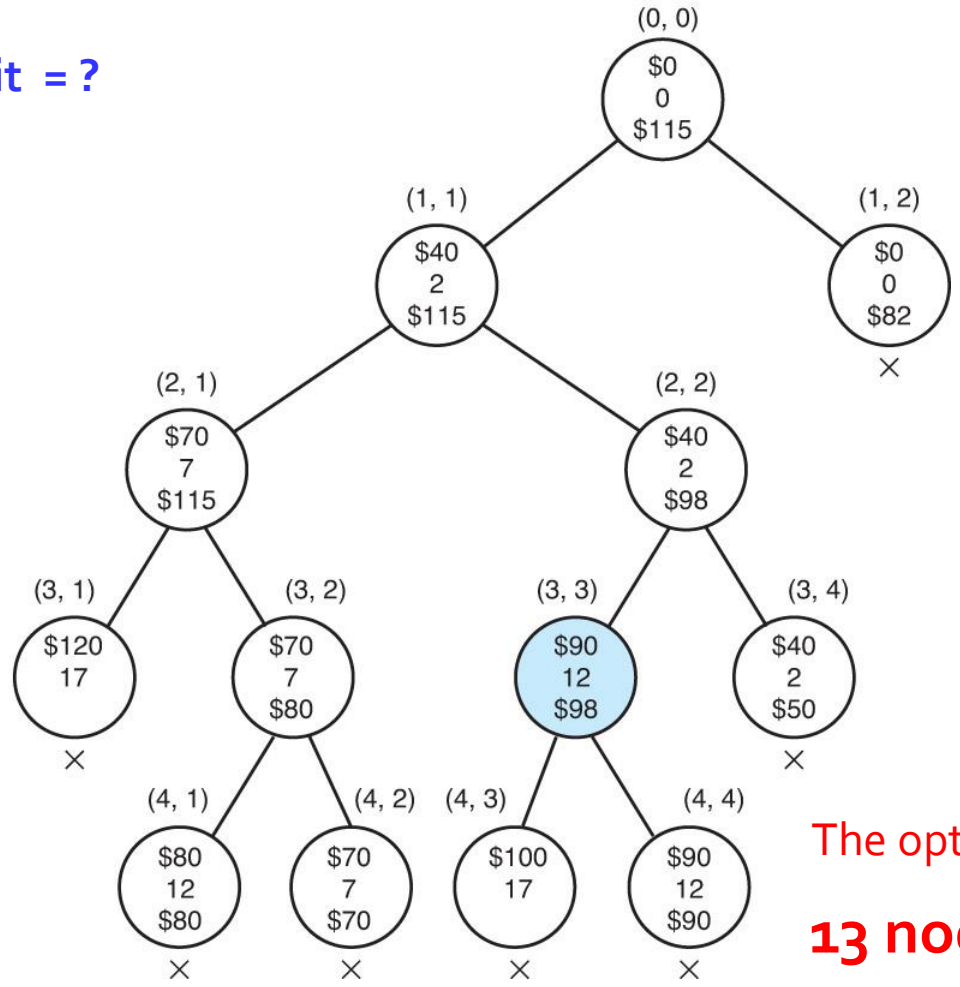
W=16

Maxprofit = ?

Total 31 nodes

DFS

Item 1 [$40, 2]
Item 2 [$30, 5]
Item 3 [$50, 10]
Item 4 [$10, 5]



Profit per unit weight!

The optimal solution!

13 nodes checked!

x= nonpromising node !

The 0-1 Knapsack Problem via Backtracking

- **Algorithm 5.7** The Backtracking Algorithm for the 0-1 Knapsack Problem
  - The number of total nodes in the state space tree for **n** items = $\mathbf{2^{n+1} - 1}$
  - $\mathbf{O(2^n)}$ at worst-case!

$$\sum_{i=0,n} 2^i = 2^{(n+1)} - 1$$

Example A.3

  - Backtracking algorithms for problems such as the 0-1 Knapsack problem are **still exponential-time in the worst case**!
  - **They are useful because they are efficient for many large instances!**

The 0-1 Knapsack Problem via Backtracking

- Backtracking algorithms depend on the data they are given!
  - This is because the decision to eliminate branches depends on the data you are processing.
    - With one set of data, the 0-1 Knapsack Problem might have to complete a depth-first search of the tree without eliminating any branches.
    - With another set of data, the 0-1 Knapsack Problem might eliminate most of the branches.

The 0-1 Knapsack Problem via Backtracking

- This is actually called
- **Branch-and-Bound** using **Depth-First Search!**

▶QUIZ? The 0-1 Knapsack Problem via Backtracking

- W = 6 lb
- Item1= $10 & 1 lb
- Item2= $18 & 2 lb
- Item3= $32 & 4 lb
- Item4= $14 & 2 lb

✓ Backtracking-based Algorithms Summary

1. The N-Queens Problem via Backtracking
2. The Graph m-Coloring Problem via Backtracking

3. The 0-1 Knapsack Problem via Backtracking (Optimization Problem)

Homework Assignment

Homework Assignment?

- **Chapter 5: Exercise #2 (The 5-Queens Problem) and #18** (No need to show all the actions step by step. Draw the pruned state space tree via Backtracking up to the point where the first solution is found and show the first solution.)

✓ Textbook Readings

- Chapter 5:
  - 5.1
  - 5.2
  - 5.5
  - 5.7

the right majors, minors &

ucation?

or students' academic and career success

ging for many students - *Many students change their*
uring college!

e prediction of student success in MMC could
dual students
d their right MMC
chieve their academic goals

END

Prof. Young Park