Branch-and-Bound (B&B)

The 0-1 Knapsack Problem

- The **0-1 Knapsack problem** (optimization problem) solved so far.
  - Via Greedy?
    - No
  - Via DP!
    - **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!
    - **$O(n^2 2^n)$**
  - 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!

The 0-1 Knapsack Problem & The Traveling Salesperson Problem

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

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

✓Branch-and-Bound (B&B)

Branch-and-Bound (B&B)

- An approach/paradigm to designing algorithms!

Branch-and-Bound (B&B)

- The **branch-and-bound strategy** is very **similar to backtracking** in that a state space tree is used to solve a problem.
  - Does **not** limit us to **any particular way** of traversing the state space tree.
  - **Is used only for optimization problems.**
- Idea?

Branch-and-Bound (B&B) Search

- Does not limit us to any particular way of traversing the state space tree:
  - **Depth-first search**
    - Using Stack
  - **Breadth-first search**
    - Using Queue
  - **Best-first search**
    - Using **Priority Queue**

✓Bound

- A branch-and-bound algorithm computes a number (**bound**) at a node to determine whether the node is promising.
  - The number is a **bound** on **the value of the solution that could be obtained by expanding beyond the node**.
  - If that bound is no better than the value of the best solution found so far, the node is **nonpromising**.
  - Otherwise it is promising.

Breadth-First Branch-and-Bound

```
void beadth_first_B&B (state_space_tree T, number best)
{
        queue_of_node Q;
        node u, v;
        initialize(Q); //initialize queue to be empty
        v = root of T;
        visit v;
        enqueue(Q, v); best = value (v);
        while (!empty(Q))
        {
                dequeue(Q, v);
                for (each child u of v)
                {
                  if ( value(u) is better than best) best=value(u);
                  if ( bound(u) is better than best) "promising"
                        enqueue(Q, u);
                }
        }

}
```

BFS

Branch-and-Bound

- Besides using the bound to determine whether a node is promising,
- We can compare the bounds of promising nodes and visit the children of the one with the best bound.
  - **Best-first search** with branch and bound pruning!
  - Does not limit us to any particular way of traversing the state space tree.

Best-First Branch-and-Bound

Best-First Search

```
void best_first_B&B (state_space_tree T, number best)
{
        priority_queue_of_node PQ;
        node u, v;
        initialize(PQ); //initialize priority_queue to be empty
        v = root of T;
        visit v;
        enqueue(PQ, v); best = value (v);
        while (!empty(PQ))
        {
                dequeue(PQ, v);
                if ( bound(v) is better than best )
                for (each child u of v)
                {
                  if ( value(u) is better than best) best=value(u);
                  if ( bound(u) is better than best) "promising"
                          enqueue(PQ, u);
                }
        }
}
```

Branch-and-Bound

- The branch and bound algorithms can be exponential or worse at **the worst-case** (**like backtracking**).
- The branch and bound algorithms can often arrive at an optimal solution faster than backtracking's depth-first search.
- The branch and bound algorithms are efficient for many large instances.

Branch-and-Bound (B&B)

- An **enhancement** of **backtracking**
  - **Depth-first search**
    - Using Stack
  - **Breadth-first search**
    - Using Queue
  - **Best-first search**
    - Using **Priority Queue**
- Applicable to **optimization problems** only

QUIZ?

- **Bound** is used for?
  - Ruling out certain nodes as "nonpromising" to prune the tree!
    - If a node's bound is not better than the best solution seen so far, the node is non-promising.
  - Guiding the search through state-space!
    - Compare the bounds of promising nodes and visit the one with the best bound.

Branch-and-Bound (B&B)-based Algorithms

- The 0-1 Knapsack Problem via B&B

1. The 0-1 Knapsack problem via B&B

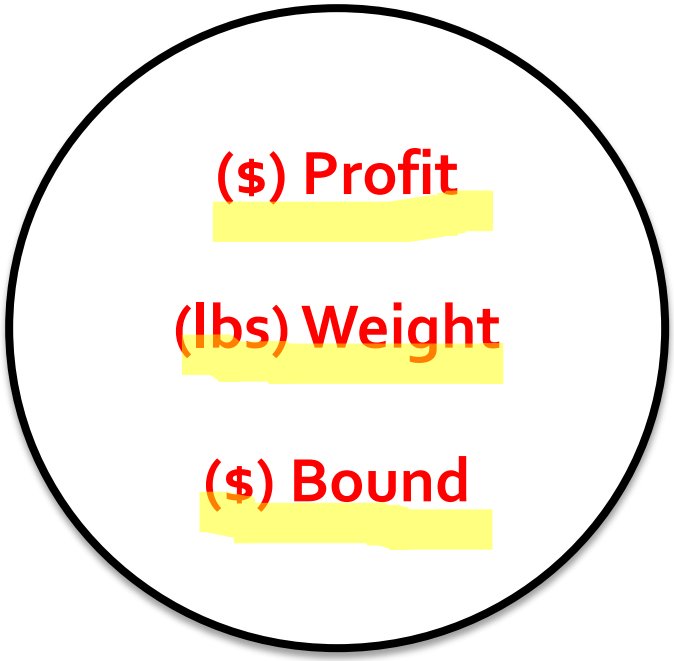
- The branch-and-bound design strategy by applying it to the 0-1 Knapsack problem.
  - First, a simple version called **breadth-first search** with branch-and-bound pruning.
  - Second, an improvement on the simple version called **best-first search** with branch-and-bound pruning.

The 0-1 Knapsack Problem via B&B

- **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 B&B

- Each node consists of



The 0-1 Knapsack problem via B&B

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

The 0-1 Knapsack problem via B&B

- **First**, a simple version called **breadth-first search** with branch-and-bound pruning.

The 0-1 Knapsack problem via B&B (**Breadth-First Search** with Branch-and-Bound Pruning)

- Example 6.1
  - 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

The 0-1 Knapsack problem via B&B

(**Breadth-First Search** with Branch-and-Bound Pruning)

Figure 6.2

W=16

Maxprofit = ?

Total 31 nodes

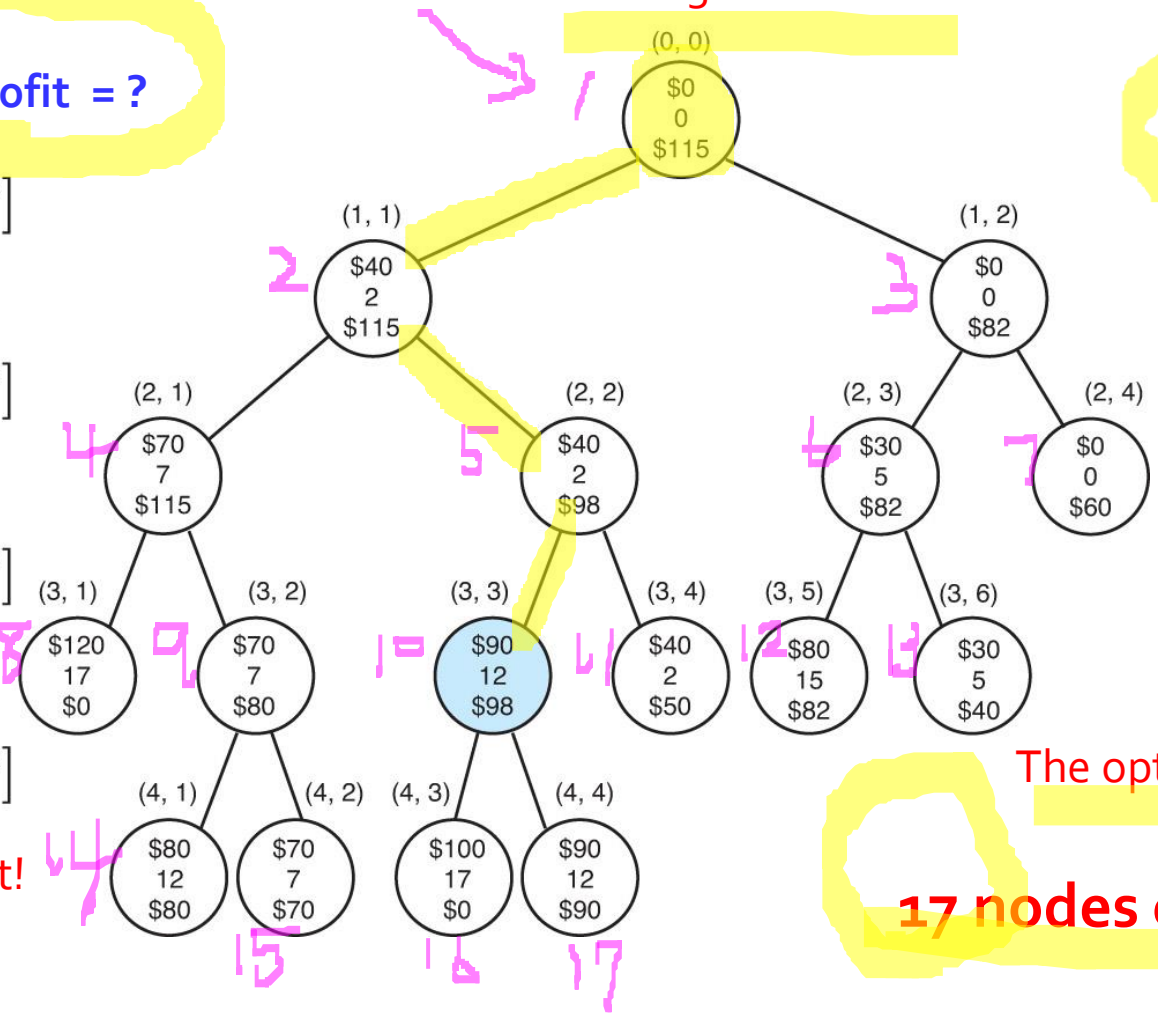
BFS

Item 1 [$40, 2]

Item 2 [$30, 5]

Item 3 [$50, 10]

Item 4 [$10, 5]



Profit per unit weight!

The optimal solution!

17 nodes checked!

▶QUIZ? The 0-1 Knapsack problem via B&B (Breadth-First Search)

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

The 0-1 Knapsack problem via B&B

- **Second**, an improvement called **best-first search** with branch-and-bound pruning.
  - Compare the bounds of promising nodes and **visit the children of the one with the best (largest) bound**.
  - We often arrive at an optimal solution more quickly than if we simply proceeded blindly in a predetermined order.

The 0-1 Knapsack problem via B&B
(Best-First Search with Branch-and-Bound Pruning)

- Example 6.2
  - 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

The 0-1 Knapsack problem via B&B

(Best-First Search with Branch-and-Bound Pruning)

Figure 6.3

W=16

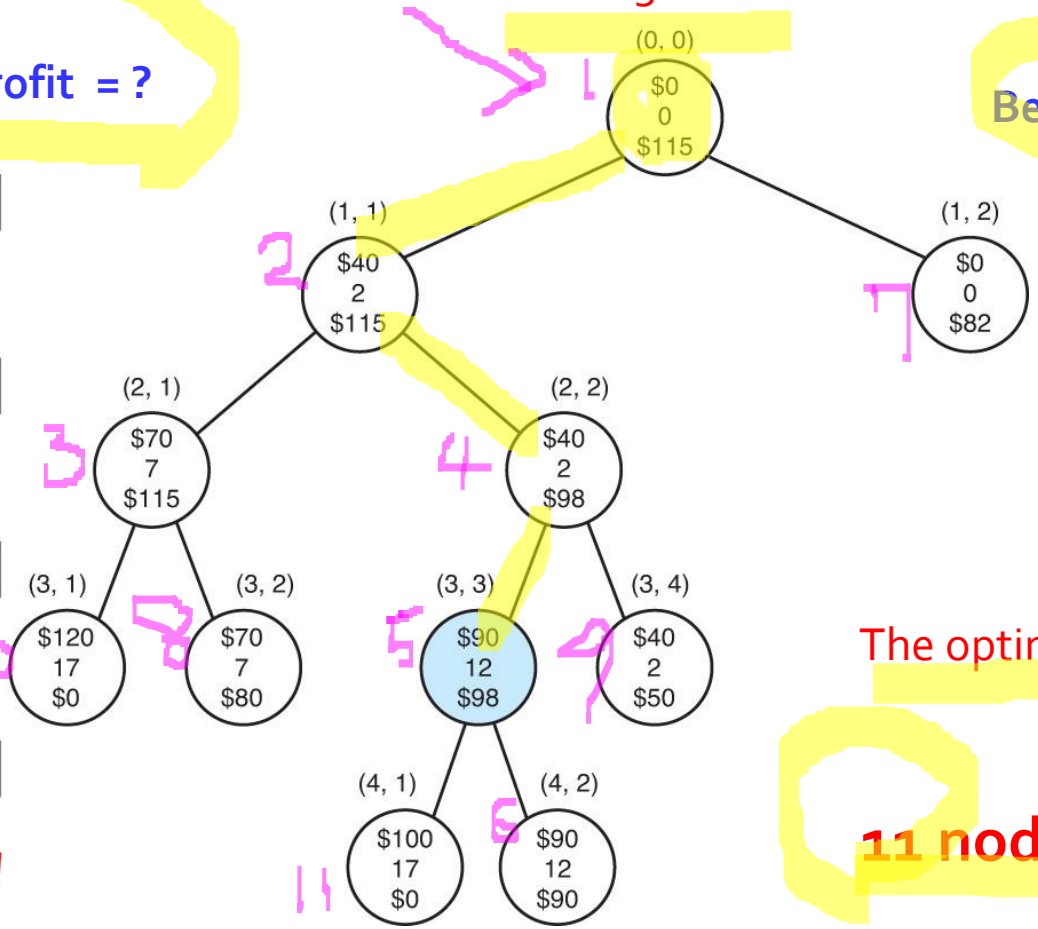
Maxprofit = ?

Item 1 [$40, 2]

Item 2 [$30, 5]

Item 3 [$50, 10]

Item 4 [$10, 5]



Total 31 nodes

Best-First Search

The optimal solution!

11 nodes checked!

Profit per unit weight!

▶QUIZ? The 0-1 Knapsack problem via B&B (Best-First Search)

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

The 0-1 Knapsack Problem

- Recall:
  - Section 5.7
  - Using **Backtracking for the 0-1 Knapsack Problem**!

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

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

Figure 5.14

W=16

Maxprofit = ?

Item 1 [$40, 2]

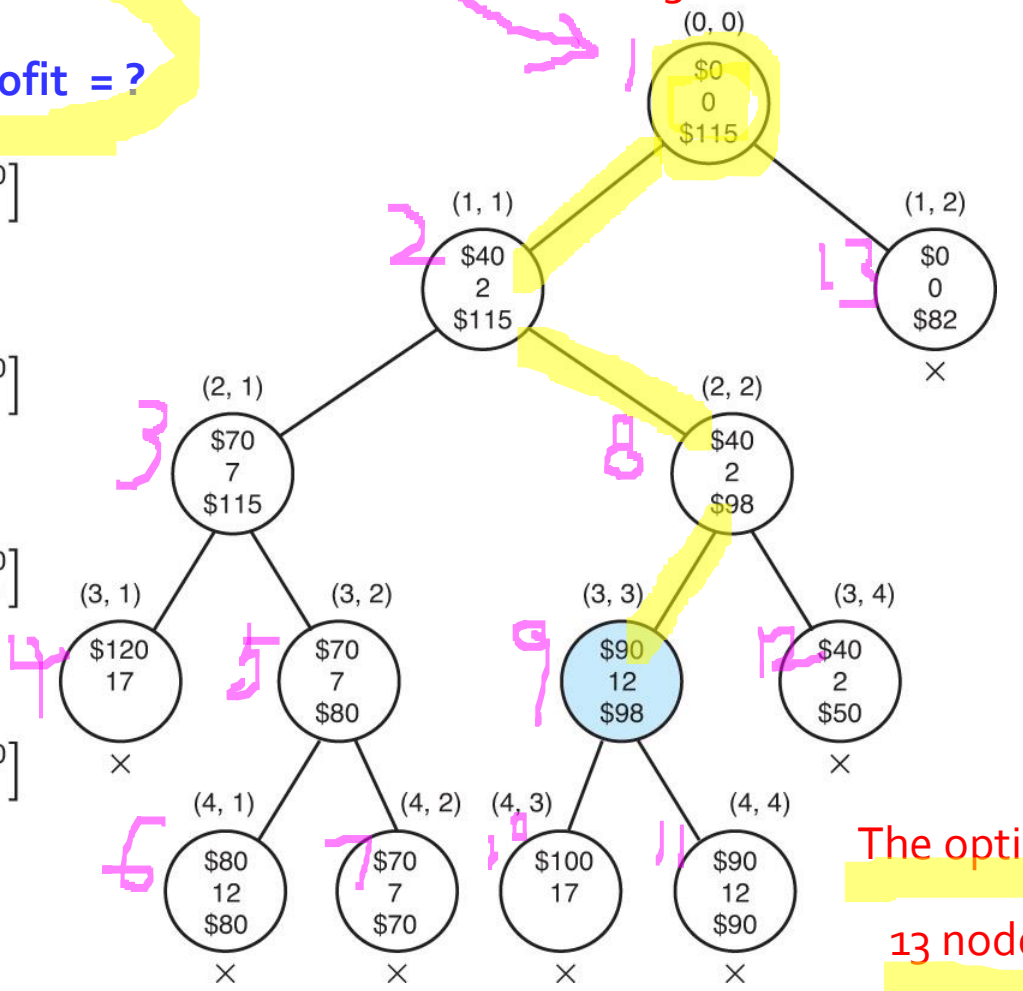
Item 2 [$30, 5]

Item 3 [$50, 10]

Item 4 [$10, 5]

Total 31 nodes

DFS



Profit per unit weight!

The optimal solution!

13 nodes checked!

x= nonpromising node !

The 0-1 Knapsack Problem

- Actually,
  - Using **Backtracking** for the 0-1 Knapsack Problem!
  - This is a **depth-first search** with **branch-and-bound pruning**.

▶QUIZ?

- Compare **Brute-Force Approach** vs. **Branch-and-Bound**?

▶QUIZ?

- Compare **Backtracking** vs. **Branch-and-Bound**?

▶QUIZ?

- Compare **Brute-Force Approach** vs. **Backtracking** vs. **Branch-and-Bound**?

Branch-and-Bound (B&B)-based Algorithms Summary

- The 0-1 Knapsack Problem via B&B

Homework Assignment

Homework Assignment?

- **Chapter 6: Exercise #1 and #4** (Draw the pruned state space tree.)

Textbook Readings

- Chapter 6:
  - 6.1