Greedy Algorithms

Weighted Knapsack

CS 3100 – DSA2

ETHAN BLASER (GRAD TA)

SLIDES ADOPTED FROM PROF. BRUNELLE AND PROF. FLORYAN

Announcements

- Brunelle out of town this week
  - Office hours will be over Discord
  - Time TBA
- PS2 Due 3/16
- PA5 Due 3/21
  - Penalty-free late submission until 3/23
- Q2 Due 3/23
  - Released Friday
  - On Recurrence relations and Divide and Conquer
    - Not purposefully cumulative (but some problems may assume familiarity of past content)
  - We will provide a guide similar to that for Q1
  - Remember to leave sufficient time to upload your paper solutions!
- PA6 Due 3/28
  - Penalty-free late submission until 3/30

Optimization Problems

Greedy algorithms can (sometimes) solve ***optimization problems:***

- Find the best solution among all ***feasible*** solutions

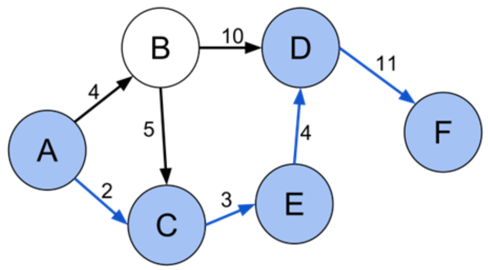
Feasible solutions must meet problem constraints

We can get a score for each feasible solution on some criteria: We call this the ***objective function***

One (or more) feasible solutions that scores highest (by the objective function) is the ***optimal solution(s)***

Examples you've already seen:

- Find the shortest path in a weighted graph G from s to v
- Find the maximum depth (PA4)
- Making Change



Optimization Problem Solving Tree

Optimal Substructure?

Q: Does our optimization problem have optimal substructure? Can we prove it?

Optimal Substructure: If given an optimal solution to the larger problem, it can be seen to be made up of optimal solutions to smaller versions of the same problem

- If A is an optimal solution to a problem, then the components of A are optimal solutions to subproblems (pg. 379 of CLRS)

Ex. Coin Change from prev. lecture

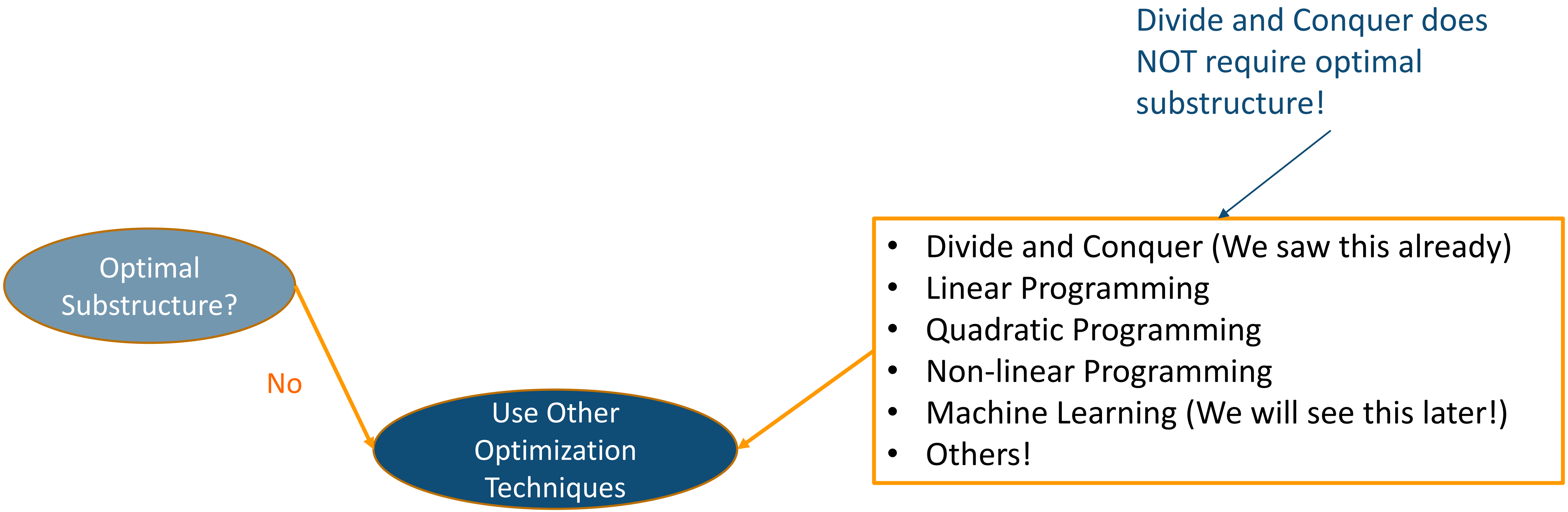
15 cents

=

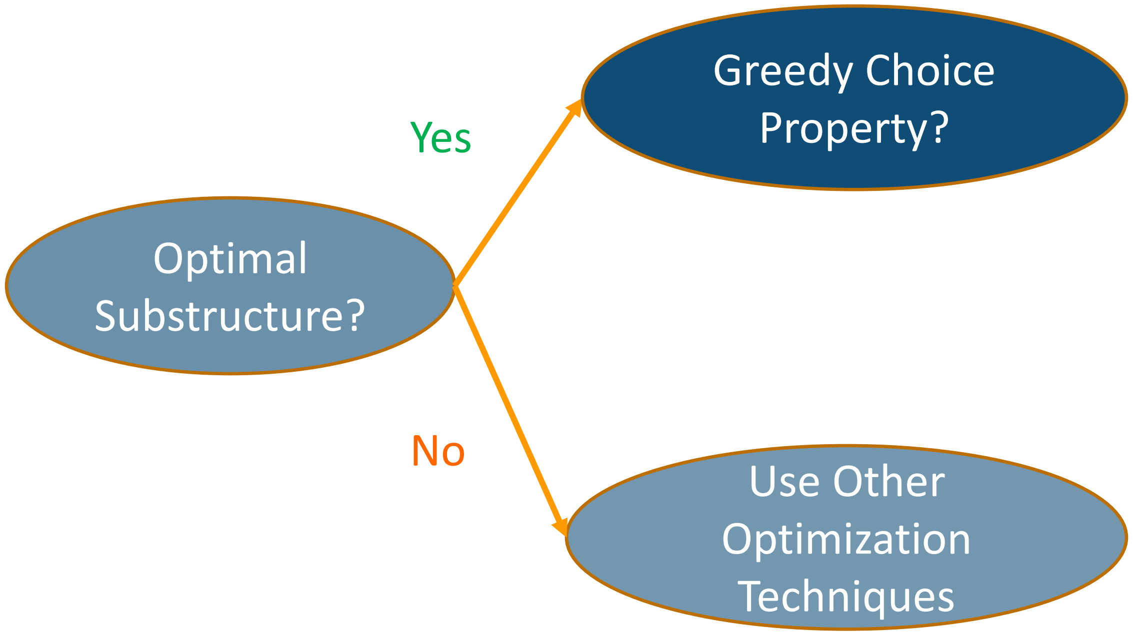
10 cents subproblem

5 cents subproblem

Optimization Problem Solving Tree



Optimization Problem Solving Tree



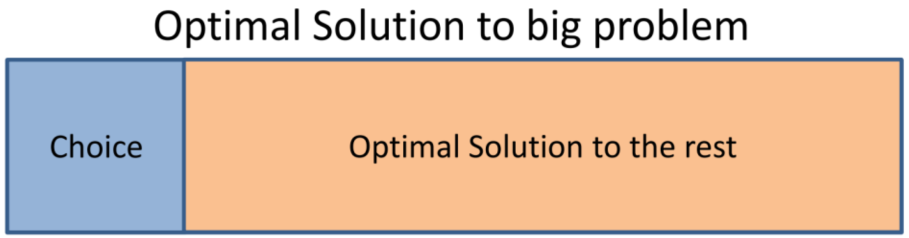
Q: Does this problem exhibit the greedy choice property? If so, what is the greedy choice function?

Greedy Choice Property: We can achieve the globally optimal solution by repeatedly making locally optimal choices

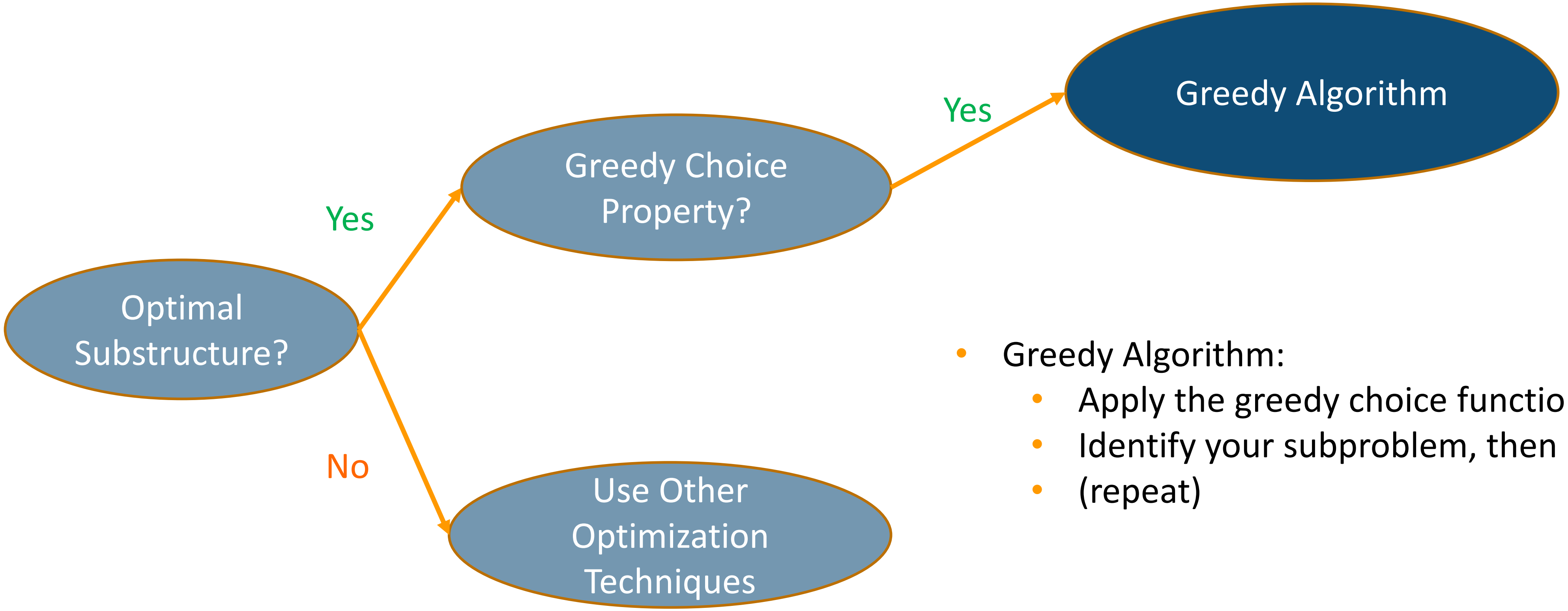
Greedy Choice Function: The rule for how to choose an item guaranteed to be in the optimal solution

- Locally optimal: best choice given what info available now
- Irrevocable: a choice can't be un-done

Must prove optimality for a given problem and greedy choice function!

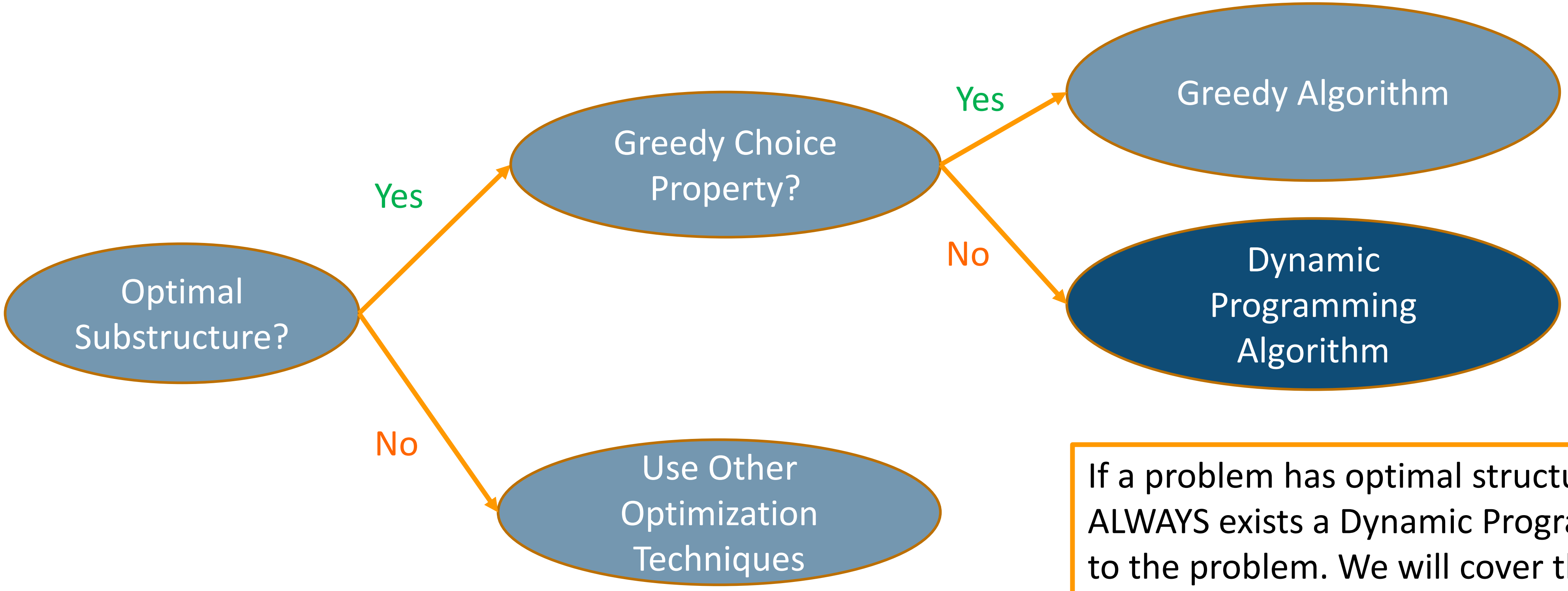


Optimization Problem Solving Tree



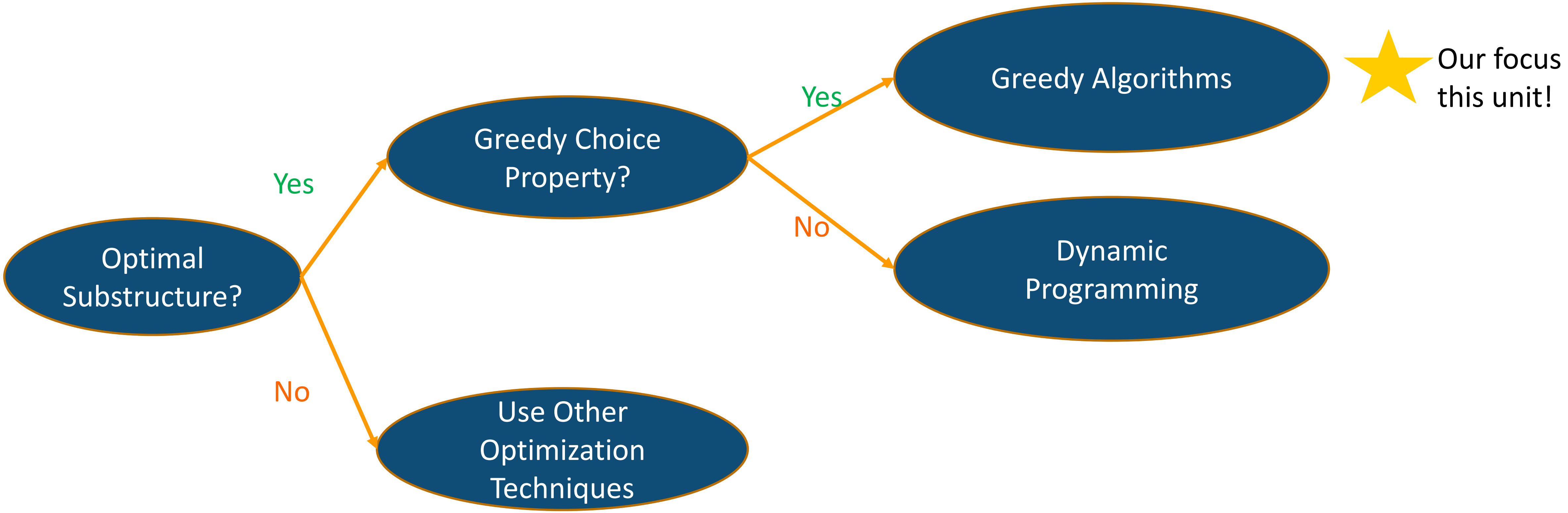
- Greedy Algorithm:
  - Apply the greedy choice function to make a decision
  - Identify your subproblem, then solve it
  - (repeat)

Optimization Problem Solving Tree

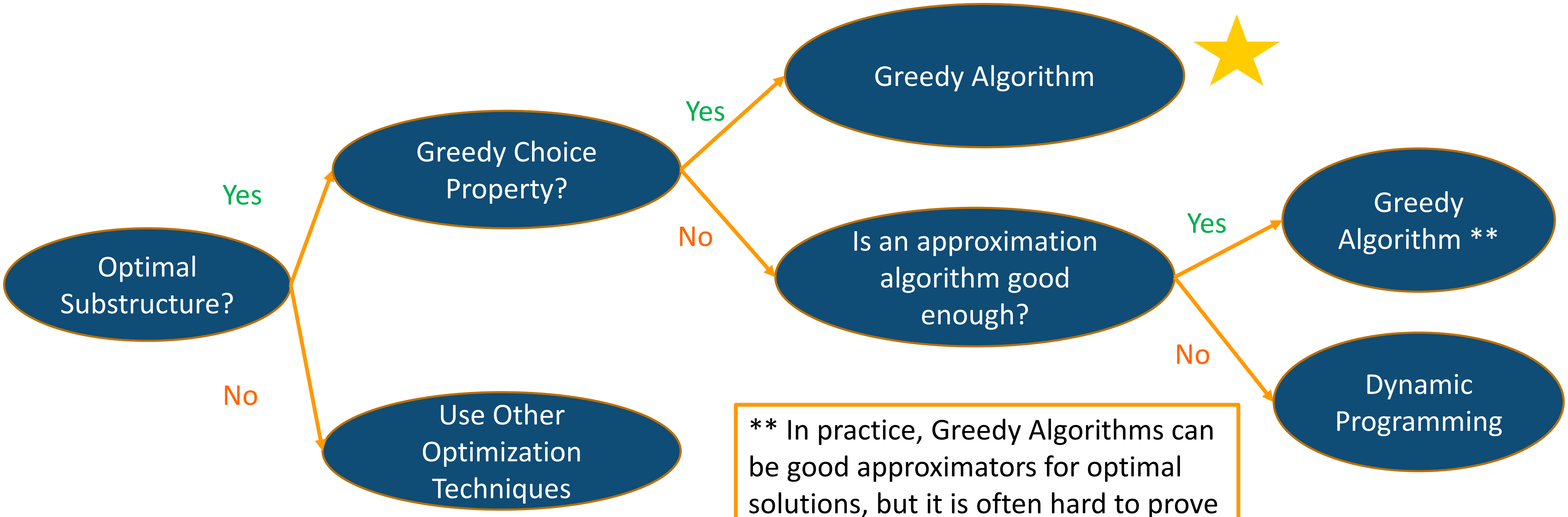


If a problem has optimal structure, then there ALWAYS exists a Dynamic Programming solution to the problem. We will cover this next unit!

Optimization Problem Solving Tree



Optimization Problem Solving Tree (Reality)



** In practice, Greedy Algorithms can be good approximators for optimal solutions, but it is often hard to prove how good

Weighted Knapsack

Knapsack Problems

Description: Thief robbing a store finds n items, each with a profit amount p_i and a weight w_i

- Wants to steal as valuable a load as possible
- But can only carry total weight C in their knapsack
- Which items should they take to maximize profit?

Form of the solution: an x_i value for each item, showing if (or how much) of that item is taken

Inputs: C, n, the p_i and w_i values

Two Types of Knapsack Problem

0/1 knapsack problem

- Each item is discrete: must choose all of it or none of it
  - Each x_i is 0 or 1
- Greedy approach does not produce optimal solutions
- But dynamic programming does

Fractional knapsack problem (AKA weighted knapsack)

- Can pick up fractions of each item
  So each x_i is a value between 0 or 1
- A greedy algorithm finds the optimal solution

Fractional Knapsack Problem Statement

Given n objects and a knapsack of capacity C, where object i has total weight w_i and earns profit p_i, find values x_i that maximize the total profit (objective function):

x

x_1	x_2	x_3		x_n

$$\arg\max_{\mathbf{x} \in \mathbb{R}^n} \left\{ \boxed{\sum_{i=0}^{n} x_i p_i} \right\}$$

Objective function

subject to the constraints:

$$\sum_{i=1}^{n} x_i w_i \leq C, \quad 0 \leq x_i \leq 1$$

Optimal Substructure Proof

First, let's show that *<u>fractional knapsack</u>* has the ***<u>optimal substructure property</u>***

Formally: Suppose we have a solution to knapsack $S = \{x_1, x_2, x_3 \ldots\}$ where each x_j is the amount taken of each of the *i* items for a knapsack with capacity C.

Then: It must be the case that $S' = \{x_2, x_3, x_4, \ldots\}$ is optimal for a knapsack of size $C - w_1 x_1$

Optimal Substructure Proof

Formally: Suppose we have a solution to knapsack $S = \{x_1, x_2, x_3 \ldots\}$ where each x_j is the amount taken of each of the *i* items for a knapsack with capacity C.

Then: It must be the case that $S' = \{x_2, x_3, x_4, \ldots\}$ is optimal for a knapsack of size $C - w_1 x_1$

Proof Outline:

1. Let V() be a function that computes the value of an item or of an entire solution
2. Note that $V(S) = V(x_1) + V(S')$ and recall that S is optimal
3. Suppose S' is NOT optimal, then some better solution S'' exists such that $V(S'') > V(S')$ for capacity $\mathrm{C} - w_1 x_1$
4. But now there is a better overall solution: $V(S) = V(x_1) + V(S') < V(x_1) + V(S'')$ so the original S is not actually optimal. Contradiction!!

Greedy Approach for Fractional Knapsack

Build up a partial solutions:

- Determine which of the remaining items to add
- How much can you add (its x_i)
- Repeat until knapsack is full (or no more items)

What's a good **greedy choice?**

Let's try several options on this example:

n = 3, C = 20

Item	Value	Weight
1	25	18
2	24	15
3	15	10

Possible Greedy Choices for Knapsack

Greedy choice #1: by highest profit value

n = 3, C = 20

Item	Value	Weight
1	25	18
2	24	15
3	15	10

Select item 1 first, then item 2, then item 3.
Take as much of each as fits!

1. Item 1 first. Can take all of it, so x_1 is 1. Capacity used is 18 of 20. Profit so far is 25.
2. Item 2 next. Room for only 2 units, so x_2 is 2/15 = 0.133. Capacity used is 20 of 20. Profit so far is 25 + (24 x 0.133) = 28.2.
3. Item 3 would be next, but knapsack full! x_3 is 0. **Total profit is 28.2. x = [1, .133, 0]**

Possible Greedy Choices for Knapsack

Greedy choice #2: by lowest weight

n = 3, C = 20

Item	Value	Weight
1	25	18
2	24	15
3	15	10

Select item 3 first, then item 2, then item 1.
Take as much of each as fits!

1. Item 3 first. Can take all of it, so x_3 is 1. Capacity used is 10 of 20. Profit so far is 15.
2. Item 2 next. Room for only 10 units, so x_2 is 10/15 = 0.667. Capacity used is 20 of 20. Profit so far is 15 + (24 x 0.667) = 31.
3. Item 1 would be next, but knapsack full! x_1 is 0. **Total profit is 31.0.** **x_i = (0, .667, 1)**

Note it's better than previous greedy choice.

Best possible?

Possible Greedy Choices for Knapsack

Greedy choice #3: highest value-to-weight ratio

$n = 3$, $C = 20$

Item	Value	Weight	Ratio
1	25	18	1.4
2	24	15	1.6
3	15	10	1.5

Select item 2 first, then item 3, then item 1.
Take as much of each as fits!

1. Item 2 first. Can take all of it, so x_2 is 1. Capacity used is 15 of 20. Profit so far is 24.
2. Item 3 next. Room for only 5 units, so x_1 is 5/10 = 0.5. Capacity used is 20 of 20. Profit so far is 24 + (15 x 0.5) = 31.5.
3. Item 1 would be next, but knapsack full! x_1 is 0. **Total profit is 31.5. x_i = (0, 1, 0.5)**

This greedy choice produces optimal solution! Must prove this.

Greedy Choice Property

<u>Greedy Choice Property</u>: The item with the largest value-to-weight ratio, filled to its max possible amount, must be in some optimal solution.

<u>Terms</u>:

Items are $I = \{i_1, i_2, i_3, \ldots\}$ and each item has a value and weight field (like an object)

Assume ratios of items sorted. $R = \{r_1, r_2, \ldots\}$ and $r_j = \frac{I[j].v}{I[j].w}$ and $r_1 \leq r_2 \leq \cdots \leq r_n$

$C > 0$ is capacity of knapsack

Accounts for when $i_n.w > C$ (when item is bigger than the capacity)

Formally: $x_n = Min\left(\boxed{\frac{C}{i_n.w}}, \boxed{1}\right)$

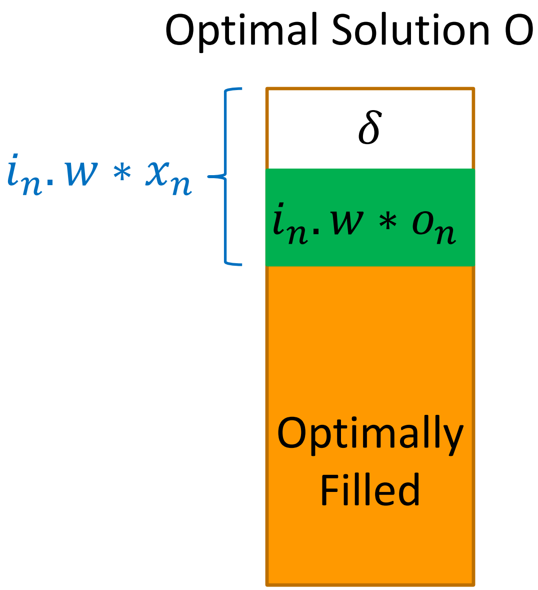
Otherwise, take the whole item

Greedy Choice Property Proof

<u>Greedy Choice Property</u>: The item with the largest value-to-weight ratio, filled to its max possible amount w.r.t. knapsack capacity, must be in some optimal solution.

<u>Proof (Exchange Argument):</u>

- Assume claim is false and the largest value-to-weight ratio item i_n is NOT in optimal solution its maximum amount
  - There exists some other optimal solution $O = \{o_1, o_2, \ldots\}$ where o_n was NOT taken to its maximum amount
- We COULD have taken item i_n some amount $x_n = Min(\frac{C}{i_n.w}, 1)$, but optimal solution O has strictly less than this amount ($o_n < x_n$)
- Let $\delta = i_n.w(x_n - o_n) > 0$ be the extra amount of weight of item i_n that was NOT taken by this optimal solution
  - Note that $0 < \delta < C$ (There must be at least some extra weight AND knapsack is not full)



Greedy Choice Property Proof

Proof (continued):

Note that $0 < \delta < C$ (There must be at least some extra weight AND knapsack is not full)

This extra weight δ must be taken by some other arbitrary item i_j in optimal solution

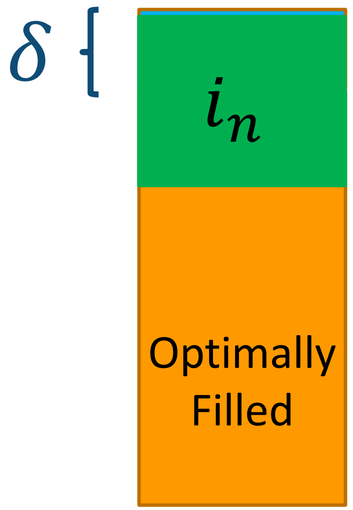
- Note that the ratio of item *j* is the same or worse than item n: $r_j \leq r_n$ *by definition

So, let's swap the i_j that was used to fill δ,with more item i_n. (V is the objective function again) to make a new solution O'

$$V(O') = V(O) - (\delta * r_j) + (\delta * r_n)$$
$$V(O') = V(O) + \delta(r_n - r_j)$$
$$V(O') \geq V(O)$$

Contradiction!!!!

Optimal Solution O'



Proof Summary

Greedy Choice Property: The item with the largest value-to-weight ratio, filled to its max possible amount w.r.t. knapsack capacity, must be in some optimal solution.

1. Toward Contradiction: Suppose there is some optimal solution O that does NOT have the largest value-to-weight ratio item i_n filled to its max possible amount
2. Then there must be some space in the knapsack where some other item i_j is used where i_n could have been used instead
3. We can exchange item i_j with more item i_n which has a higher value-to-weight ratio giving a new solution O’ to the problem
4. We showed that this new solution O’ has a larger overall profit than the original “optimal solution” O.
5. Contradiction! Therefore, the largest weight value item must be in the optimal solution

Fractional Knapsack Algorithm

```
FRACTIONAL_KNAPSACK(a, C)
n = a.last
for i = 1 to n
    ratio[i] = a[i].p / a[i].w
sort(a, ratio)
weight = 0
i = n
while (i >= 0 and weight < C)
    if (weight + a[i].w ≤ C)
        println "select all of object " + a[i].id
        weight = weight + a[i].w
    else
        r = (C – weight) / a[i].w
        println "select " + r + " of object " + a[i].id
        weight = C
    i = i-1
```

Sorting is θ(nlogn)

For loop and while loop take θ(n) time

Greedy Algorithm: **θ(nlogn)**
Brute Force: $O(2^n)$

Another Knapsack Example to Try

Assume for this problem that: $\sum_{i=1}^{n} w_i \le C$

Ratios of profit to weight:

$p_1/w_1 = 5/120 = .0417$
$p_2/w_2 = 5/150 = .0333$
$p_3/w_3 = 4/200 = .0200$
$p_4/w_4 = 8/150 = .0533$
$p_5/w_5 = 3/140 = .0214$

What order do we examine items?

What are the x_i values that result?

What's the total profit?

0/1 Knapsack

Let's try this same greedy solution with the 0/1 version

- New example inputs →

n = 3, C = 4

Item	Value	Weight	Ratio
1	3	1	3
2	5	2	2.5
3	6	3	2

1. Item 1 first. So x_1 is 1.
   Capacity used is 1 of 4. Profit so far is 3.
2. Item 2 next. There's room for it! So x_2 is 1. Capacity used is 3 of 4.
   Profit so far is 3 + 5 = 8.
3. Item 3 would be next, but its weight is 3 and knapsack only has 1 unit left!
   So x_3 is 0. **Total profit is 8. x_i = (1, 1, 0)**

But picking items 1 and 3 will fit in knapsack, with total value of 9

- Thus, the greedy solution does not produce an optimal solution to the 0/1 knapsack algorithm
- Greedy choice left unused room, but we can't take a fraction of an item
- The 0/1 knapsack problem doesn't have the *greedy choice property*

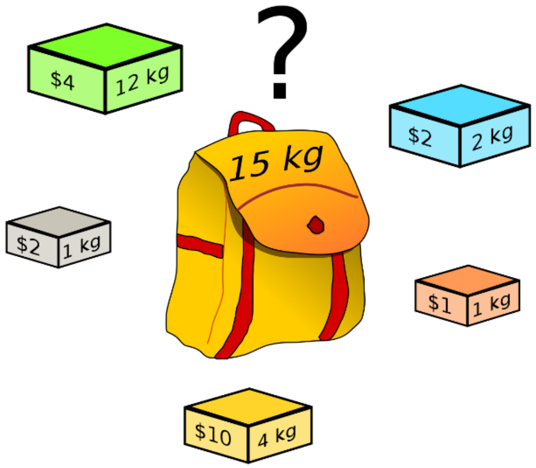
Recap

Optimization problems can sometimes be solved by Greedy Algorithms

- Discussed “Optimization Problem Solving Tree” to determine which algorithmic approach best suits your problem

Greedy Algorithms require the problem to have:

- Optimal Substructure
- Greedy Choice Property



Weighted Knapsack Problem

- Proved optimal substructure
- Formulated the Greedy Choice function
- Proved the greedy choice function is optimal via Exchange Argument