Today:

– Greedy Algorithms, Part 1

COSC 581, Algorithms

February 11, 2014

Many of these slides are adapted from several online sources

Reading Assignments

- Today's class:
  - Chapter 16.1-16.2

- Reading assignment for next class:
  - Chapter 16.3, Ch. 21

- Announcement: Exam 1 is on Tues, Feb. 18
  - Will cover everything up through dynamic programming and HW #5

Recall: Optimization problems

- An optimization problem:
  - Given a problem instance, a set of constraints and an objective function
  - Find a feasible solution for the given instance for which the objective function has an optimal value
  - Either maximum or minimum depending on the problem being solved
- A feasible solution satisfies the problem's constraints
- The constraints specify the limitations on the required solutions.
  - For example in the knapsack problem we will require that the items in the knapsack will not exceed a given weight

The Greedy Technique (Method)

- Greedy algorithms make good local choices in the hope that they result in an optimal solution.
  - They result in feasible solutions
  - **Not** necessarily an optimal solution

- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a *local* optimum at each step, you will end up at a *global* optimum

- A proof is needed to show that the algorithm finds an optimal solution.

- A counterexample shows that the greedy algorithm does not provide an optimal solution.

Example: Coin changing problem

- Problem: Return correct change using a minimum number of bills/coins.
- Greedy choice: bill/coin with highest coin value that does not overshoot
  - Example: To make \$6.39, you can choose:
    - a \$5 bill
    - a \$1 bill, to make \$6
    - a 25¢ coin, to make \$6.25
    - A 10¢ coin, to make \$6.35
    - four 1¢ coins, to make \$6.39
- For U.S. money, the greedy algorithm always gives the optimal solution.

Is this currency greedy-optimal?

- In some (fictional) monetary system, “krons” come in 1 kron, 7 kron, and 10 kron coins

Is this currency greedy-optimal?

- In some (fictional) monetary system, “krons” come in 1 kron, 7 kron, and 10 kron coins
- No. Using a greedy algorithm to count out 15 krons, you would get:
  - A 10 kron piece
  - Five 1 kron pieces, for a total of 15 krons
  - This requires 6 coins
- A better solution would be to use two 7 kron pieces and one 1 kron piece
  - This only requires 3 coins
- The greedy algorithm results in a feasible solution, but not in an optimal solution

Elements of the Greedy Strategy

- Sometimes a greedy strategy results in an optimal solution and sometimes it does not.
- No general way to tell if the greedy strategy will result in an optimal solution
- Two ingredients necessary:
  - greedy-choice property
  - optimal substructure

Greedy-Choice Property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
- Unlike dynamic programming, we solve the problem in a top down manner
- Must prove that the greedy choices result in a globally optimal solution

Optimal Substructure

- Like dynamic programming, the optimal solution must contain within it optimal solutions to sub-problems.

- Given a choice between using a greedy algorithm and a dynamic programming algorithm for the same problem, in general which would you choose?

Elements of the Greedy Strategy

Cast problem as one in which we make a greedy choice and are left with one subproblem to solve.

Elements of the Greedy Strategy

To show optimality:

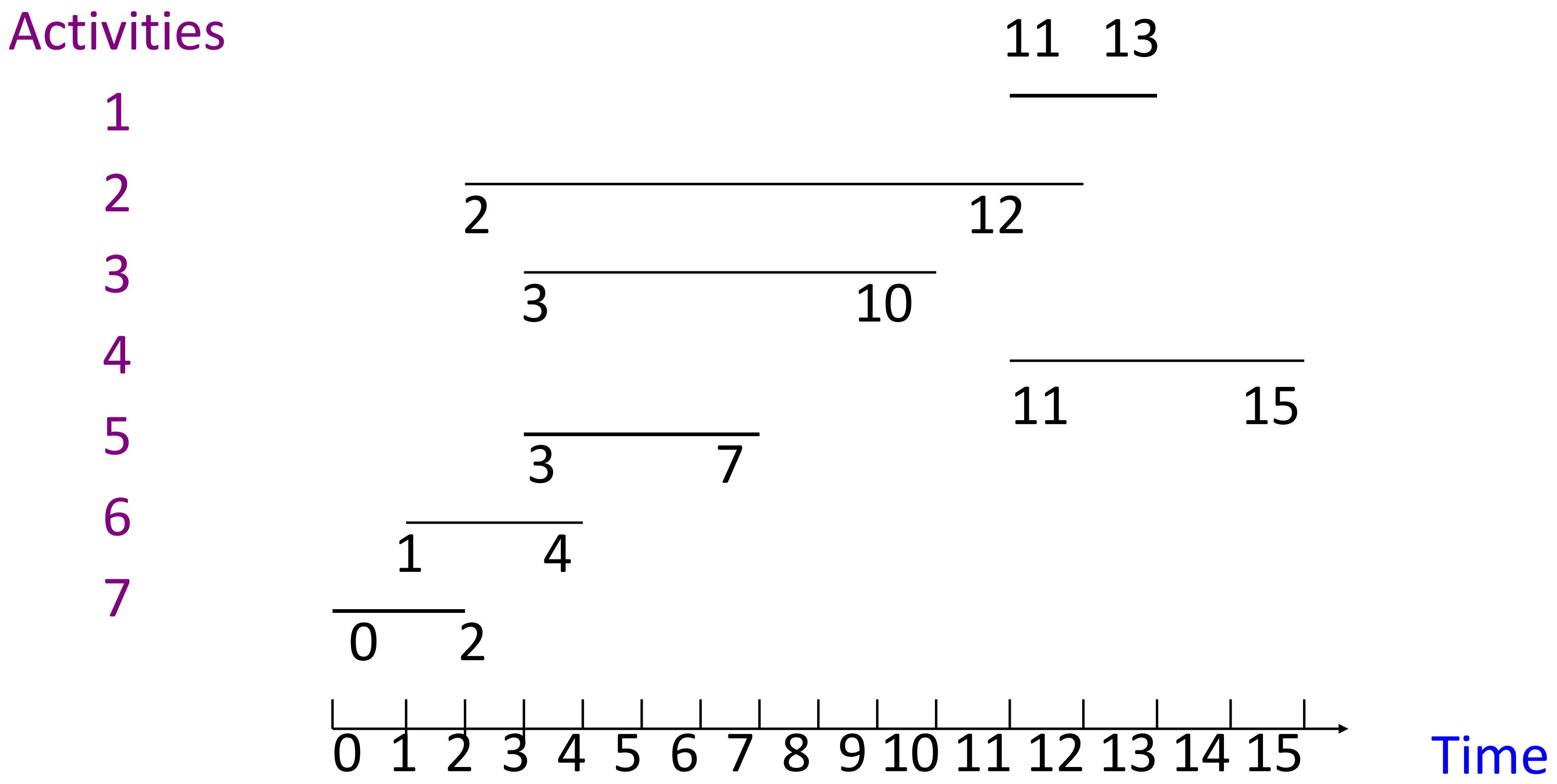
1. Prove there is always an **optimal solution** to original problem that makes the greedy choice.
2. Demonstrate that what remains is a subproblem with property:

   If we combine the optimal solution of the subproblem with the greedy choice we have an optimal solution to original problem.

Activity Selection Problem

- Given a set *S* of *n activities* with *start* time s_i and *finish* time f_i of activity a_i
- Find a maximum size subset *A* of compatible activities (maximum *number* of activities).
- Activities are compatible if they do not overlap
- Can you suggest a greedy choice?

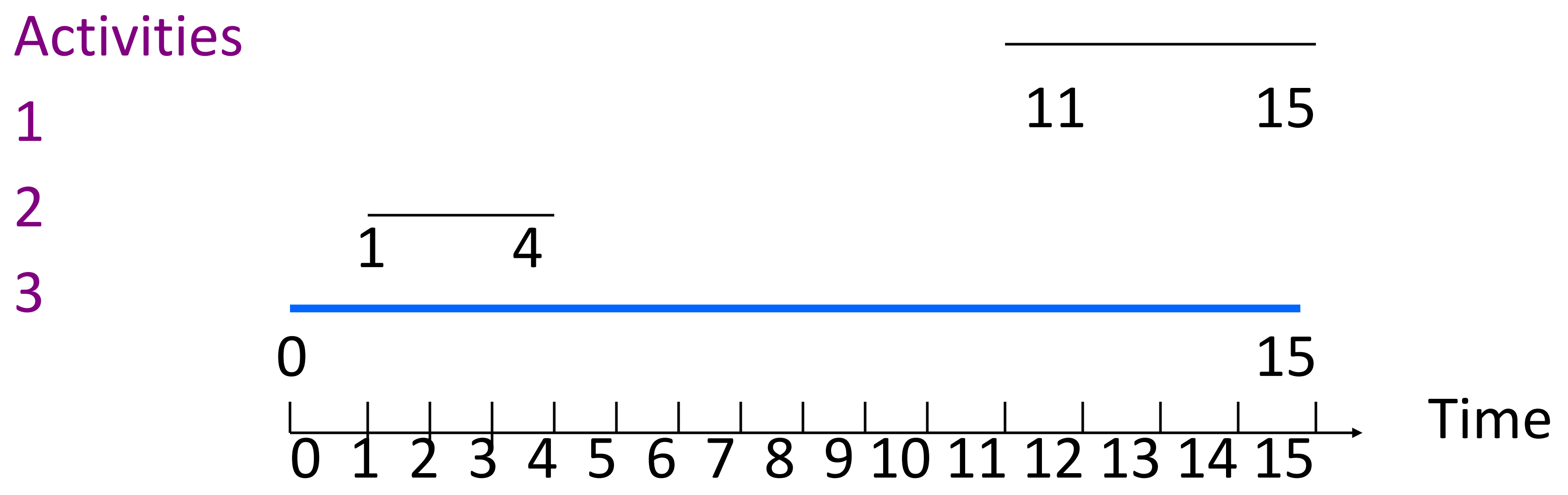
Example



Activities
1
2
3
4
5
6
7
11 13
2 12
3 10
11 15
3 7
1 4
0 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Time

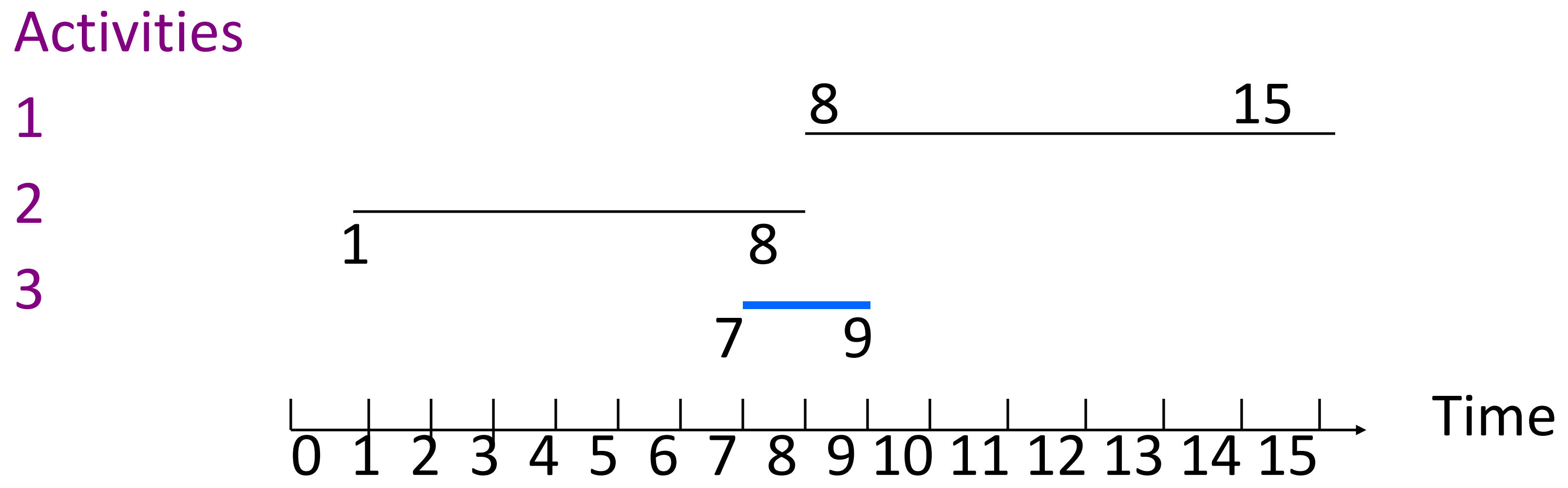
Counterexample 1

- Select by start time

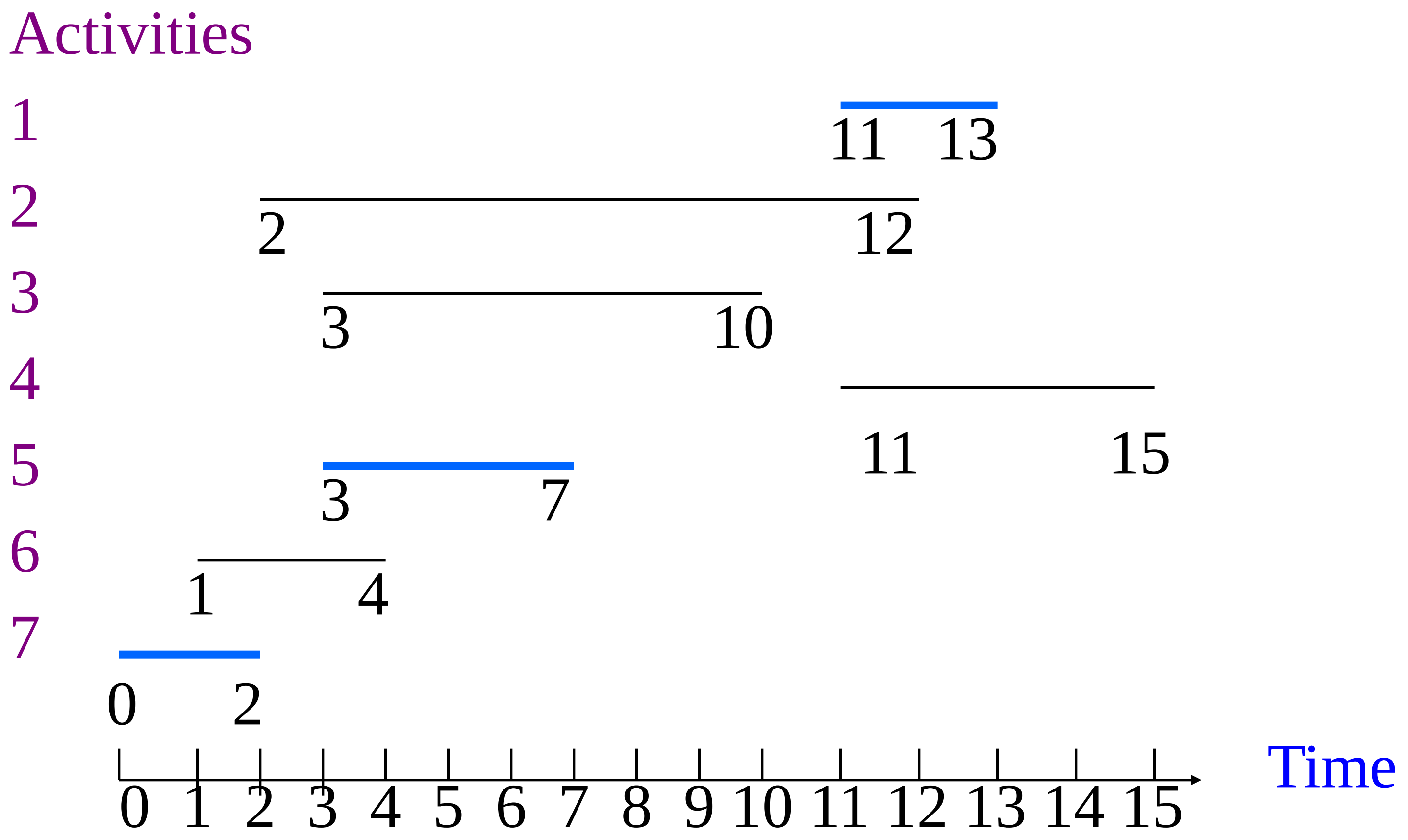


Counterexample 2

- Select by minimum duration



Select by earliest finishing time



Activity Selection

- Assume without loss of generality that we number the intervals in order of *finish time*. So $f_1 < \ldots < f_n$. (Requires a sort)

- Greedy choice: choose activity with minimum finish time

- The following greedy algorithm starts with A={a_1} and then adds all compatible jobs

GREEDY-ACTIVITY-SELECTOR(s,f)

```
$n \leftarrow$ s.length // number of activities
$A \leftarrow \{a_1\}$
$k \leftarrow 1$        //last activity added
for $m \leftarrow 2$ to $n$     //select
    if $s_m \geq f_k$ then      //compatible (feasible)
        $A = A \cup \{a_m\}$
        $k \leftarrow m$      //save new last activity
return $A$
```

GREEDY-ACTIVITY-SELECTOR(s,f)

```
$n \leftarrow$ s.length // number of activities
$A \leftarrow \{a_1\}$
$k \leftarrow 1$        //last activity added
for $m \leftarrow 2$ to $n$     //select                    $(\Theta(n))$
    if $s_m \geq f_k$ then      //compatible (feasible)
        $A = A \cup \{a_m\}$
        $k \leftarrow m$      //save new last activity
return $A$
```

$\Theta(n \lg n)$ when including sort

Analysis of Activity Selection Algorithm

- The activity selected for consideration is always the one with the earliest finish
- Why does this work? Intuitively, it always leaves the maximum time possible to schedule more activities
- The greedy choice maximizes the amount of unscheduled time remaining

Proving the Greedy Algorithm Finds the Optimal Solution

Theorem: Algorithm GREEDY-ACTIVITY-SELECTOR produces solutions of maximum size for the activity selection problem

General form of proofs:

- prove that first greedy choice is correct
- show by induction that all other subsequent greedy choices are correct

Proof:

Let $S_k = \{a_i \in S\colon\ s_i \geq f_k\}$ be the set of activities that start after activity a_k finishes. Activity a_m has the earliest finish time.

Step 1 (Greedy choice property): Show that there is an optimal solution that contains activity a_m.

Suppose A_k is a subset of S_k that is an optimal solution.

Suppose the activity in A_k with the earliest finish time is a_j.

If $a_j = a_m$, then schedule A_k begins with a greedy choice.

If $a_j \neq a_m$, then we need to show that there is another optimal solution A'_k that begins with the greedy choice of a_m.

Let A'_k = A - $\{a_j\} \cup \{a_m\}$

We need to show that A'_k is still optimal and that the activities do not conflict when we replace a_j with a_m.

1. A'_k has the same number of activities as A_k so it is optimal
2. Since $f_m \leq f_j$, activity a_m will finish before the second activity in A'_k begins, so there are no conflicts.

Proof continued:

Step 2 (Optimal substructure): Show that a greedy choice of activity a_1 results in a smaller problem that consists of finding an optimal solution for the activity-selection problem over those activities in S that are compatible with activity a_1.

We want to show that if A is an optimal solution to the original problem S, then A′= A - { a_1} is an optimal solution to the activity-selection problem
S′ = { $a_i \in$ S: $s_i \geq f_1$}.

Use a proof by contradiction:

Suppose that we could find a solution B′ to S′ with more activities than A′. Then we could add activity a_1 to B′ and have a solution to S with more activities than A. But since we assumed that A was optimal this is not possible and thus we have a contradiction.

Thus, after each greedy choice is made, we are left with an optimization problem of the same form as the original problem. By induction on the number of choices made, making a greedy choice at every step produces an optimal solution.

Greedy versus Dynamic Programming

- Both greedy and dynamic programming exploit the optimal substructure property

- Optimal substructure: a problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to the sub-problems.

Two Knapsack Problems

- 0-1 knapsack problem
  - A thief robbing a store finds n items
  - Item i is worth v_i dollars and weighs w_i pounds (both v_i and w_i integers)
  - Can carry at most W pounds in knapsack
  - Goal: determine the set of items to take that will result in the most valuable load

- Fractional knapsack problem
  - same setup
  - allow thief to take fractions of items

Optimal Substructure Property of Two Knapsack Problems

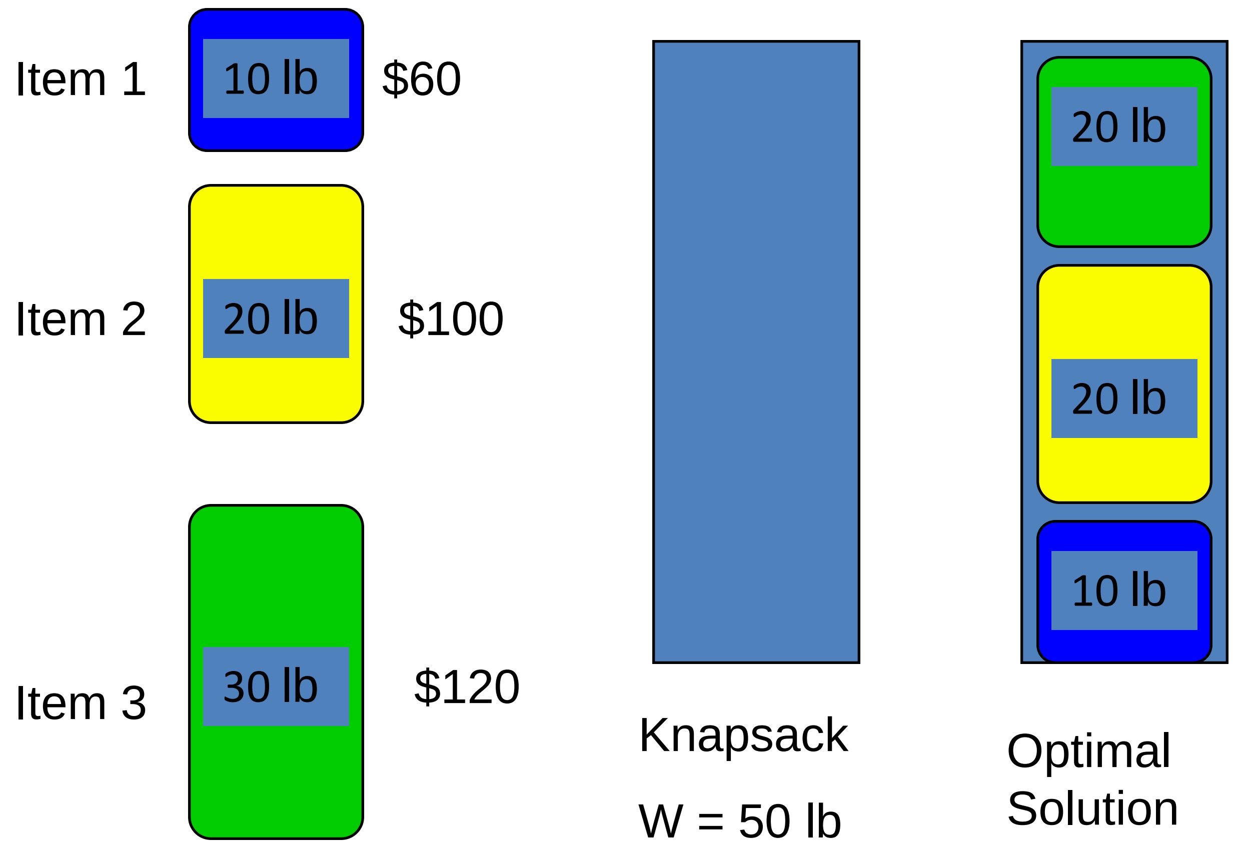
- 0-1 Knapsack:
  - Consider optimal load of weight W
  - If item j is removed from the load, the resulting load is the most valuable load weighing at most $W - w_j$ that can be taken from $n - 1$ original items excluding item j

- Fractional Knapsack:
  - Consider optimal load of weight W
  - If we remove weight w of item j , the remaining load is the optimal load weighing $W - w$ that the thief can take from the original $n - 1$ original items excluding $w_j - w$ pounds of item j

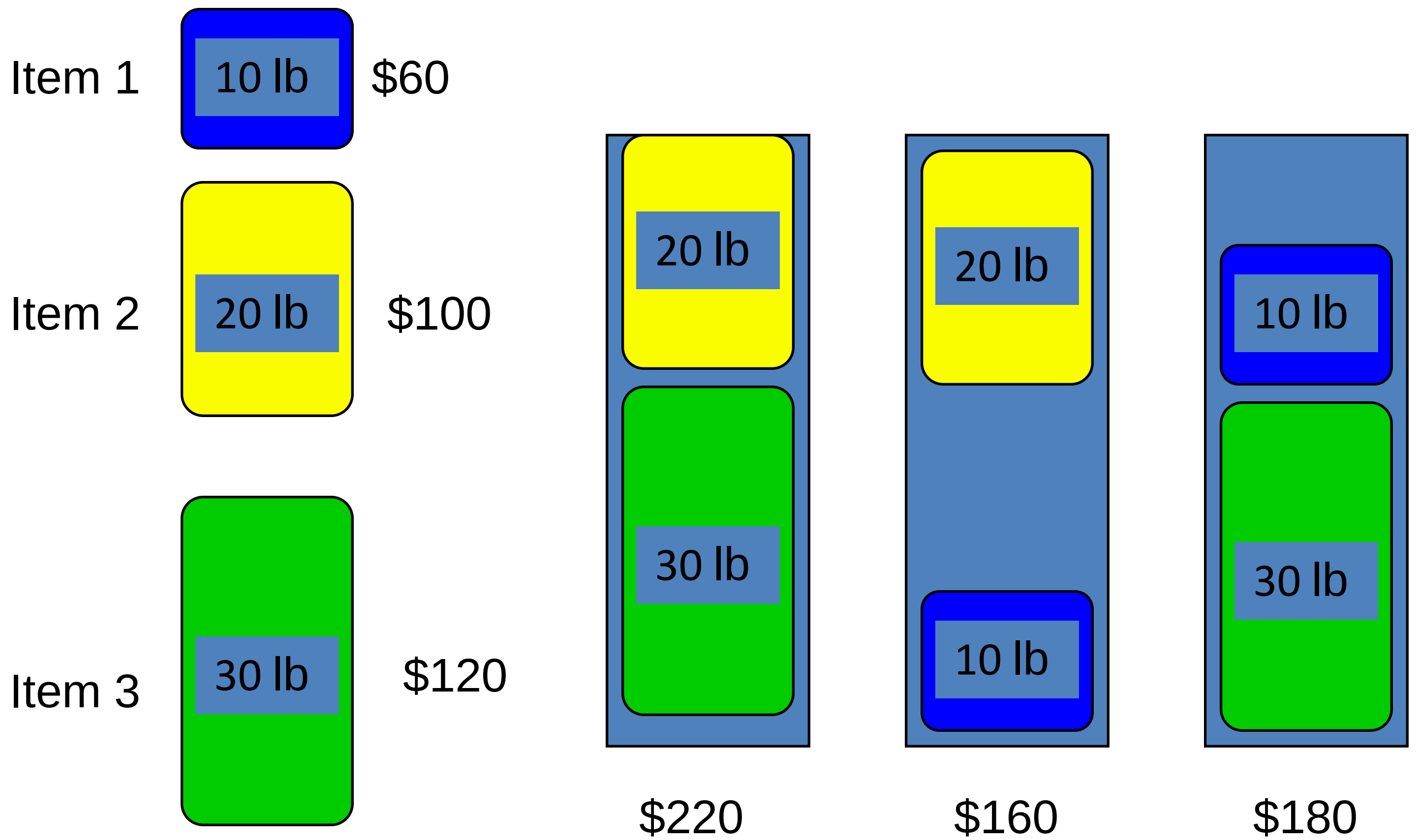
Fractional Knapsack can be Solved Using Greedy Algorithm

- What is the greedy selection criterion?

Fractional Knapsack Example



Greedy Does Not Work for 0-1 Knapsack



Other Possible Greedy Strategies

- Pick the heaviest item first?
- Pick the lightest item first?
- Need dynamic programming: For each item, consider an optimal solution that does and does not include the item.
- Moral:
  - Greedy algorithm sometimes gives the optimal solution, sometimes not, depending on the problem.
  - Dynamic programming, when applicable, will typically give optimal solutions, but are usually trickier to come up with and sometimes trickier to implement.

0-1 Knapsack Solution

- The dynamic programming solution to this problem is similar to the LCS problem. At each step, consider including or not including each item in a solution

- Let x_i be *0* if item i is not included and 1 if it is included

- Our goal is to maximize the value of the pack while keeping the weight $\leq$ W

Dynamic Programming vs. Greedy Algorithms

- Dynamic programming
  - We make a choice at each step
  - The choice depends on solutions to subproblems
  - Bottom up solution, from smaller to larger subproblems

- Greedy algorithm
  - Make the greedy choice and THEN
  - Solve the subproblem arising after the choice is made
  - The choice we make may depend on previous choices, but not on solutions to subproblems
  - Top down solution, problems decrease in size

By the way… Dijkstra’s alg. is greedy

- Dijkstra’s algorithm finds the shortest paths from a given node to all other nodes in a graph
  - Initially,
    - Mark the given node as *known* (path length is zero)
    - For each out-edge, set the distance in each neighboring node equal to the *cost* (length) of the out-edge, and set its *predecessor* to the initially given node
  - Repeatedly (until all nodes are known),
    - Find an unknown node containing the smallest distance
    - Mark the new node as known
    - For each node adjacent to the new node, examine its neighbors to see whether their estimated distance can be reduced (distance to known node plus cost of out-edge)
      - If so, also reset the predecessor of the new node

In-class Exercise

You are given a set $\{x_1, x_2, \ldots x_n\}$ of points that all lie on the same real line. Let us define a “unit-length closed interval” to be a line segment of length 1 that lies on the same real line as the points in the given set. This line segment “contains” all the points that lie along the segment, including the end points.

Describe a greedy algorithm that determines the smallest set of unit-length closed intervals that contains all of the given points.

In-class Exercise (con't.)

Prove this alg. is correct.

Reading Assignments

- Reading assignment for next class:
  - Chapter 16.3, Ch. 21

- Announcement: Exam 1 is on Tues, Feb. 18
  - Will cover everything up through dynamic programming and HW #5