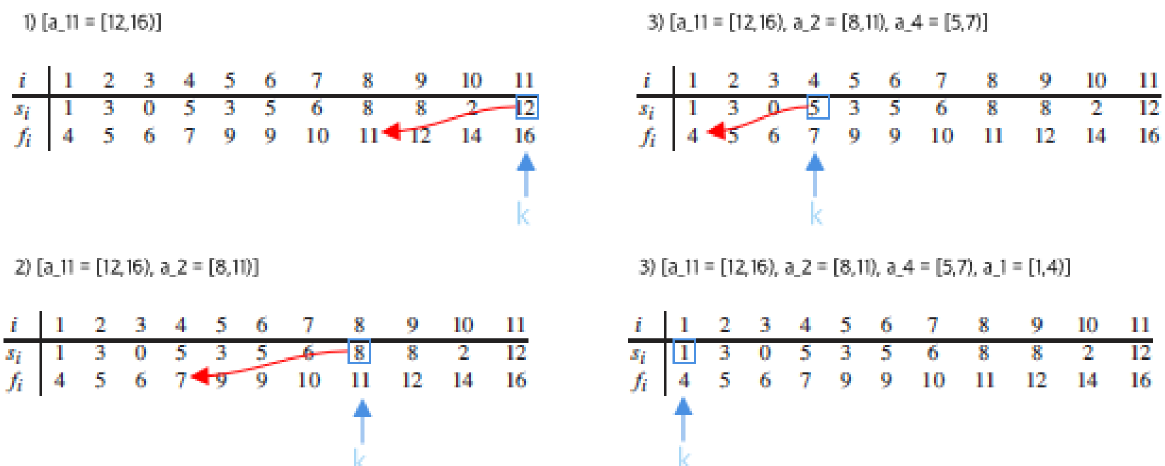
CSC373 Worksheet 2 Solution

July 26, 2020



1.

This approach is a greedy algorithm because algorithm

1) Has the greedy choice: selecting the last activity to start that is compatible with all previously selected activites
2) Has the greedy choice that is always part of optimal solution:

   Claim:

   Consider any nonempty subproblem S_k. Let a_m be an activity in S_k with the last activity to start that is compatible with all previously selected activities. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k

   Proof. Let A_k be a maximum-size subset of mutually compatible activities in S_k, and let a_j be the activity in A_k with the last activity to start that is compatible with all previously selected activities.

   If $a_j = a_m$, we are done, since we have shown that a_m is the maximum-size subset of mutually compatible activities of S_k.

If $a_j \neq a_m$, let the set $A'_k = A_k = \{a_j\} \cup \{a_m\}$ be A_k but subtituting a_m for a_j. The activities in A'_k are disjoint, which follow because the activities in A_k are disjoint, a_j is the first activity in A_k to finish, and $s_j \leq s_m$.

Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m. $\square$

Notes:

- Greedy Algorithm
  - Always makes the choice that looks best at the moment
    * Locally optimal solution leads to globally optimal solution
- Activity-selection Problem (Greedy algorithm using dynamic programming)
  - Goal: Selecting maximum size set of mutually compatible activities

    Example:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

  - Suppose a set exists $S = \{a_1 = [s_1, f_1), a_2 = [s_2, f_2), ..., a_n = [s_n, f_n)\}$
    * a_i represents an i^{th} activity
    * s_i represents starting time
    * f_i represents finishing time
    * $0 \leq s_i < f_i < \infty$
    * $a_1, ..., a_n$ sorted in monotonically increasing order of finish time

      i.e.

      $f_1 \leq f_2 \leq f_3 \leq ... \leq f_{n-1} \leq f_n$

    * a_i and a_j are **compatible**, if intervals $[s_i, f_i)$ and $[s_j, f_j)$ don't overlap

      i.e

      $s_i \geq f_j$ and $s_j \geq f_i$

  - Steps
    1. Think about dynamic programming solution
       * Construct optimal solution using two subproblems

S_{ij}**:** activities that start after activity a_i finishes and before activity a_j starts

i.e.

$S_{19} = \{a_4 = [5, 7), a_6 = [5, 9), a_7 = [6, 10)\}$

A_{ij}**:** maximum set of mutually compatible activities in S_{ij} (including a_k)

- $A_{ik} = A_{ij} \cap S_{ik}$
- $A_{kj} = A_{ij} \cap S_{kj}$
- $A_{ij} = A_{ik} \cup \{a_k\} \cup Akj$
- So, $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$

∗ Verify that optimal solution A_{ij} must include optimal solution to the two subproblems for S_{kj}

Let A'_{kj} be another mutually compatible activities in S_{kj} where $|A'_{kj}| > |A_{kj}|$.

Then we could use A'_{kj} in a solution to subproblem of S_{ij}

Then we have $|A_{ik}| + |A'_{kj}| + 1 > |A_{jk}| + |A_{kj}| + 1 = |A_{ij}|$ mutually compatible activites

This contradicts assumption that A_{ij} is an optimal solution

∗ Verify that optimal solution A_{ij} must include optimal solution to the two subproblems for S_{ik}

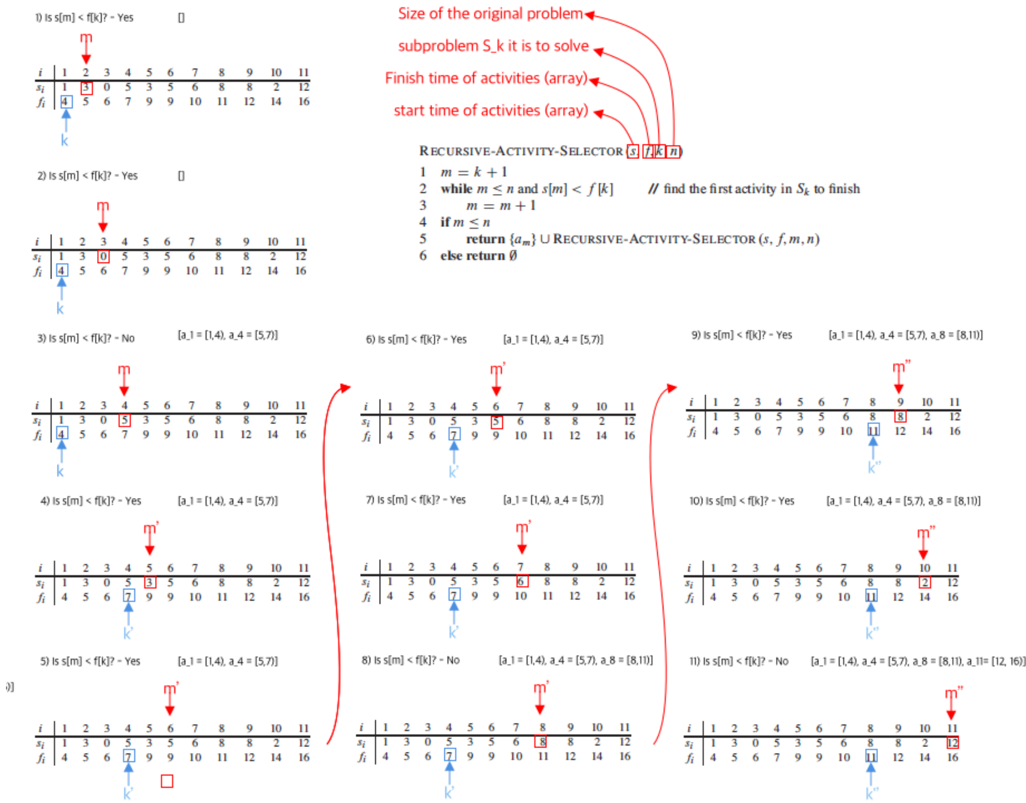
The same applies for activities in S_{ik}

2. Observe that only one choice - greedy choice, and that when we make the greedy choice, only one subproblem remains
   ∗ Steps
     1. Make a greedy choice
        - Choose an activity that makes the most resource possible (intuition)
        - Choose an acitivty that finishes the earliest (intuition)
     2. Solve a subproblem: Find activities that start after a_1 finishes
     3. Verify that making greedy choices always arrive at optimal solution

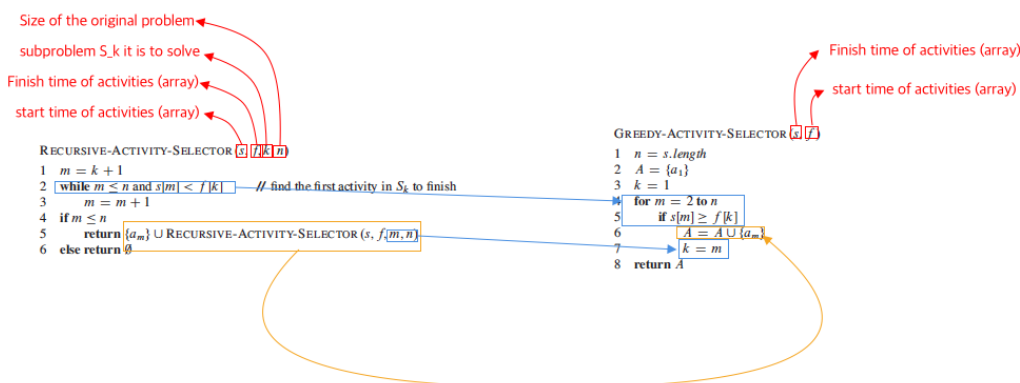
Theorem 16.1 (Page 418):

Consider any non-empty subproble S_k, and let a_m be an activity in S_k with the earliest finish time. Then a_m is included in some maximum-size subset of mutually compatible activities of S_k

3. Develop recursive greedy solution



4. Convert the recursive algorithm into iterative one



2. • Greedy Choice

   – Choose x_i that is greater than the current maximum as the upper bound of unit length closed interval

   – Choose x_i that is smaller than the current minimum as the lower bound of unit length closed interval

Example:

$\{0, 1, 2, 3, 4, 5\} \rightarrow [0, 5]$

$\{0, -1, 3, 5, 2\} \rightarrow [-1, 5]$

- Optimal Substructure

  Let I be the following instance of the problem: Let n be the number of items, and let x_i be the i^{th} point in the set.

  Let $A = [x_{\min}, x_{\max}]$ be the solution. The greedy algorithm works by assigning $x_{\min} = min(x_{\min}, x_n)$ and $x_{\max} = max(x_{\max}, x_n)$, and then continuing by solving the subproblem

$$I' = (n - 1, \{x_1, ..., x_{n-1}\}) \tag{1}$$

  until $n = 0$.

  We need to show that the strategy gives optimal solution.

Notes:

- Stopped because it's taking too much time.
- I struggled on this problem.
- I am having difficulty arguing why the algorith is correct
  - How can i generate a claim?
- Unit length
  - $[1, 25, 2.25]$ includes all x_i such that $1.25 \leq x_i \leq 2.25$.
- Greedy-choice property and optimal substructure to problem are the two key ingredients
- Summary of Steps for Greedy Algorithm
  1. Determine the optimal structure of the problem
  2. Develop a recursive solution.
  3. Show that if we make the greedy choice, then only one subproblem remains
  4. Prove that it is always safe to make the greedy choice
  5. Develop a reursive algorithm that implements the greedy strategy

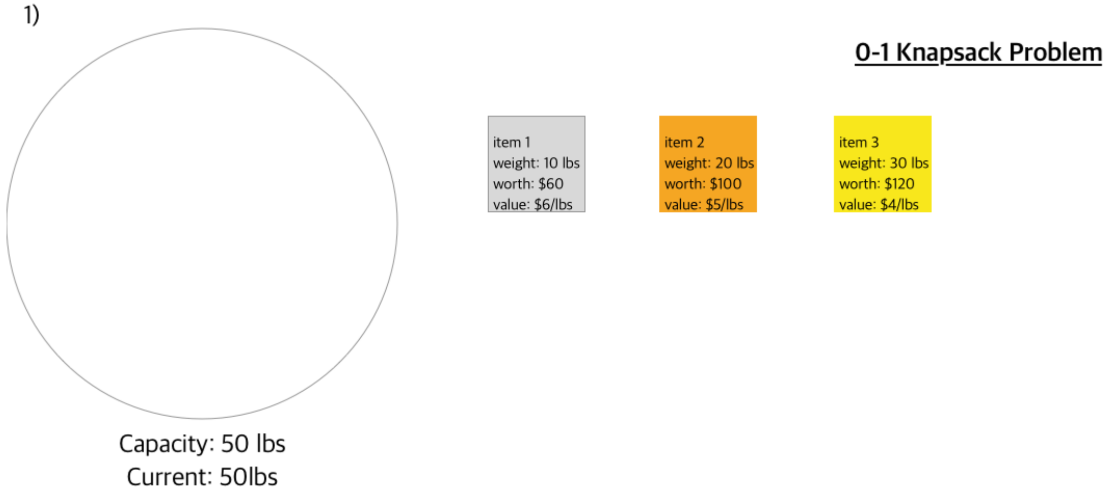
6. Convert the recursive algorithm to an iterative algorithm

- Criteria for Greedy Algorithm

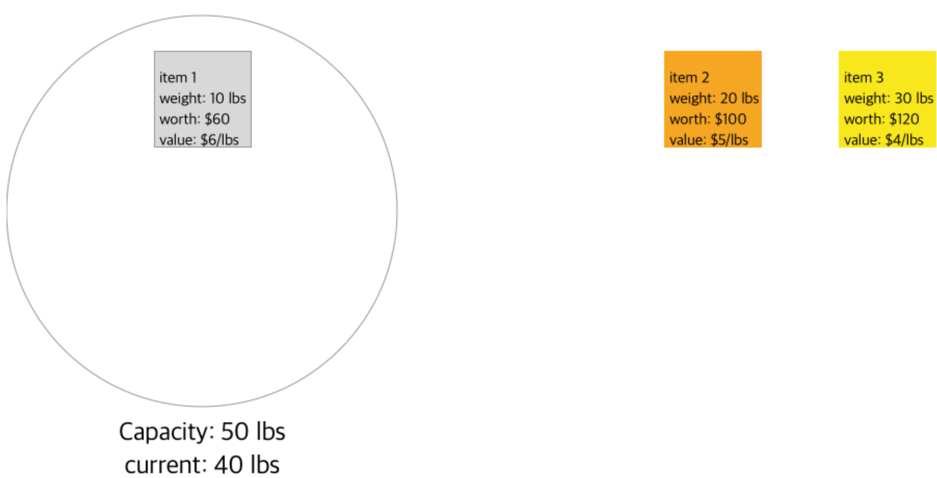
1. Greedy-choice property
   - Exists if we can assemble a globally optimal solution by making a locally optimal (greedy) choices
2. Optimal Substructure
   - Exists if an optimal solution to the problem contains within it optimal solutions to subproblems.

- Greedy vs Dynamic Programming
  - 0-1 Knapsack Problem

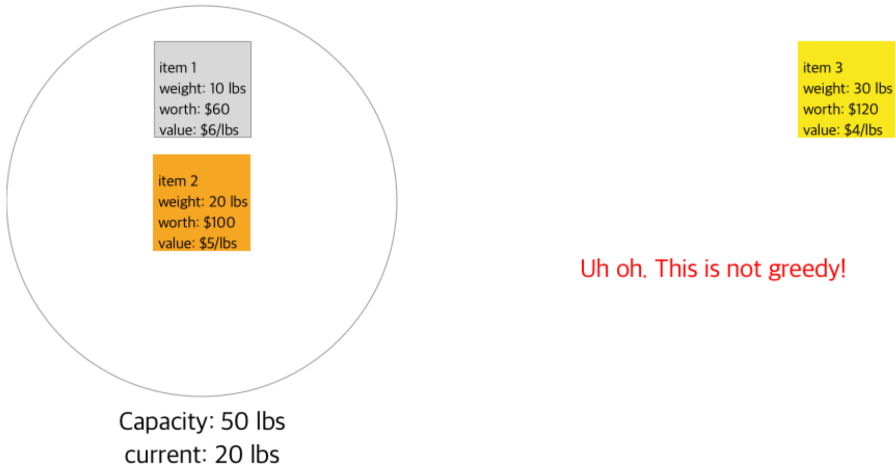
1)



2)

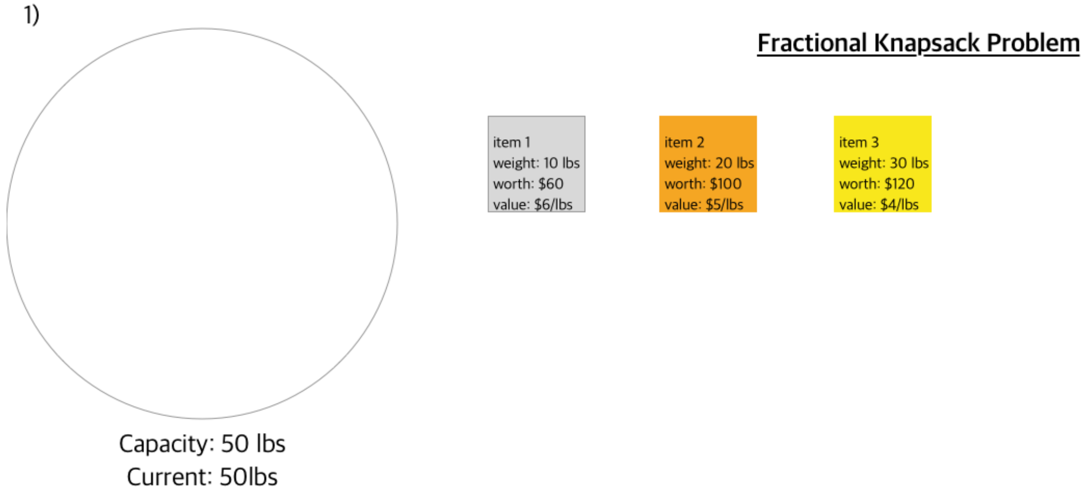


3)



– Fractional Knapsack Problem

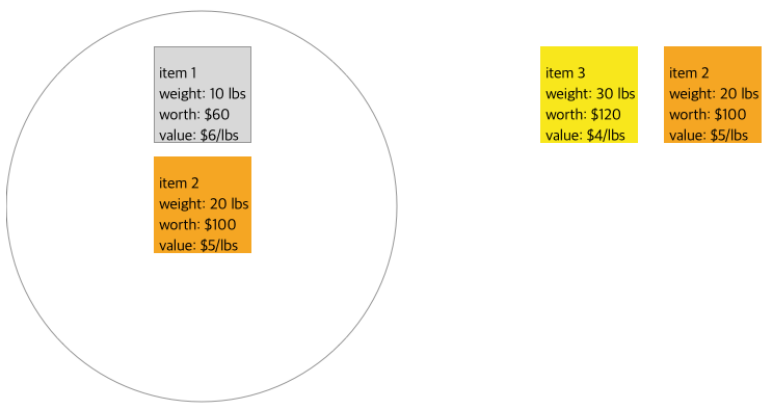
1)



2)

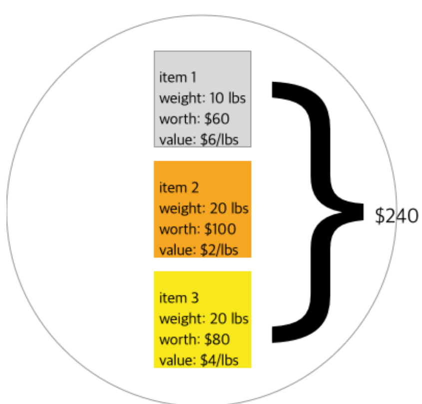
Capacity: 50 lbs
current: 40 lbs

3)



Capacity: 50 lbs
current: 20 lbs

4)



Capacity: 50 lbs
current: 20 lbs