Lecture: Greedy Algorithms

- Warm-up: A variant of 0-1-Knapsack
- Greedy Choice Property: Prove that it holds
- Fractional Knapsack
- Elements of Greedy Strategy: Recipe for designing greedy algorithms
- The Big Picture: Divide-and-Conquer vs. Dynamic Programming vs. Greedy algorithms

These notes are hand-written, unedited and sketchy. They are primarily used for, and based on my lectures.

If you find any bug, impreciseness, or a rare poor-/mis-interpretation of facts, please let me know. I will be grateful for any additional comments you have that are intended to make the quality of the notes better.

Please note that I will provide my hand-written lecture notes only for a subset of my lectures, not for all lectures. Therefore, it is your responsibility to attend all the lectures, take notes regularly, and ask me and/or the TAs if you have any questions.

Thank you!
--Chittu

Written using an Intuos 5 Touch Pen Tablet and Microsoft Office OneNote
A Variant of 0-1-Knapsack:

Given: Item(2): 1 2 ... n

Weight: \( w_1 < w_2 < \cdots < w_n \)

Value: \( v_1 > v_2 > \cdots > v_n \)

And a knapsack of capacity (by weight) \( W \).

Goal: Determine a subset of items from \( I \), that can fit in the knapsack and maximizes the total value of items in \( I \).

\[ \text{Maximize: } \sum_{i=1}^{n} x_i v_i \quad \text{Subject to: } \sum_{i=1}^{n} x_i w_i \leq W, \]

where, \( x_i = 1 \) if item \( i \) is picked; otherwise \( x_i = 0 \).

Idea: Use dynamic programming. Sure, we can solve using DP.

Observation: We've more information available to us:

\[ \text{i.e. } w_1 < w_2 < \cdots < w_n, \text{ and } v_1 > v_2 > \cdots > v_n. \]

Perpetual Question: Can we do better?

Greedy Strategy: "Pick items in the decreasing order of \( \frac{v_i}{w_i} \)."

\[ \text{i.e. same as increasing order of their weights.} \]

MUST prove that our greedy choice property works!

Claim:

(i) If \( W > w_1 \), an optimal solution contains item 1.

(ii) If \( W < w_1 \), then the knapsack is empty.

Proof:

(i) Let \( S \) be an optimal solution. Assume \( W > w_1 \).

- Clearly, \( S \) must be nonempty, assuming \( v_1 > 0 \).
- If \( S \) contains item 1, then we are done.
• Otherwise, let item $i$, ($i \neq 1$), be any other item in $S$. 
• Now, let's do cut-and-paste:
  $$S' = (S - \{\text{item } i^2\}) \cup \{\text{item 1}\}.$$ 
• Since $w_i > w_1$ and $v_i < v_1$, $S'$ has greater value than that of $S$, and $S'$ has weight $\leq$ that of $S$.
• $\therefore S'$ must also be optimal.
• Since $S'$ contains item 1, we have the claim.

(ii) If $W < w_1$, then item 1 does not fit to the knapsack, which means no item fits into the knapsack. $\square$

**GREEDY-0-1-KNAPSACK-VARIANT**($v$, $w$, $n$, $W$)
1. Sort the items in the increasing order of their weights. $O(n \log n)$
2. $i \leftarrow 1$, $S = \emptyset$
3. While $i \leq n$ and $\text{remainingWeight} > 0$ $\Theta(n)$
4. If $w_i < \text{remainingWeight}$
5. $S \leftarrow S \cup \{\text{item } i^2\}$
6. $\text{remainingWeight} \leftarrow \text{remainingWeight} - w_i$
7. Else break
8. $i \leftarrow i + 1$
9. Return $S$

**Complexity Analysis:**

• Time: Clearly, the running time is $O(n \log n)$.

• Space: In the worst case $S$ holds all the items, so space is $O(n)$. 
Fractional Knapsack: a.k.a. Continuous Knapsack

Given:
- Item(I): 1 2 \cdots n
- Weight(w_i): w_1, w_2 \cdots w_n \quad w_i > 0 \text{ for } 0 \leq i \leq n
- Value: v_1, v_2 \cdots v_n \quad v_i > 0 \text{ for } 0 \leq i \leq n

Knapsack of total capacity (by weight) W.

Goal: Determine a subset I' \subseteq I of items that can fit in the knapsack and maximizes the total value of items in I'.

Additionally, we have the flexibility to pick a fraction x_i, 0 \leq x_i \leq 1, of the item i.

Formally, maximize \sum_{i=1}^{n} x_i v_i \quad \text{subject to } \sum_{i=1}^{n} x_i w_i \leq W,

where v_i \geq 0, w_i > 0, and 0 \leq x_i \leq 1, for 1 \leq i \leq n

Greedy Strategy: "Pick (fraction of) items in the decreasing order of \frac{v_i}{w_i}.''

Exercise: Design an algorithm for Fractional KNAPSACK and analyze its performance.
What we know so far:

1. Optimal Substructure: An optimal solution to a problem instance contains within it optimal solutions to subproblems.

2. Overlapping Subproblems: A recursive solution contains a "small" number of "distinct" subproblems "repeated" many times.

3. Greedy Choice Property: Locally optimal choices lead to globally optimal solution.

- Only $\Rightarrow$ Divide and Conquer
- $1 \& 2$ only $\Rightarrow$ Dynamic Programming
- $1, 2 \& 3$ $\Rightarrow$ Greedy Algorithms

Must prove the greedy choice property so that you can use greedy algorithms.

If this can be shown, we have just one problem to solve!
Note: Greedy choice Property: “Locally opt. choices give global opt. solution.”

Greedy Strategy:
1. Determine the optimal substructure.
2. Give a recursive formulation of the problem.
3. Show that if we make a greedy choice, we are left with only one subproblem to solve.
4. Prove that it is always safe to make the greedy choice.
5. Design a recursive greedy algorithm.
6. Convert it into an iterative algorithm.