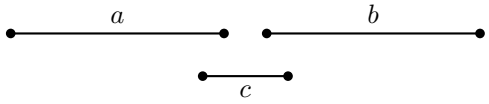
COSC 302: Analysis of Algorithms Lecture — Spring 2018
Prof. Darren Strash
Colgate University

Worksheet 9 — Greedy Algorithms and Dynamic Programming I

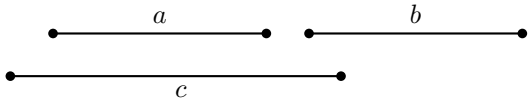
Greedy Algorithms

1. Prove that that the following greedy strategies fail to solve the scheduling problem, by given a counter example:

   (a) Choose the shortest time interval to add to the schedule.

   (b) Choose the time interval with the earliest start time

   (c) Choose the time interval with the fewest overlaps with remaining intervals
   (To be discussed in a future lecture.)

2. Does the following strategy have the greedy choice property? *Repeatedly choose the interval with the latest start time.*

   Solution: Yes, this is the same as selecting the interval with the earliest finishing time. To see this, subtract all the intervals end points from some very high number, then select the intervals with the earliest finishing time. This is the same set of intervals.

3. **(Huffman coding)** Describe how to compute the value w_{ij} in constant time when filling in the table. *(This was mistakenly given for Huffman coding, this will be covered during discussion on Optimal BSTs)*

Dynamic Programming I

4. Define the *0-1 Knapsack Problem* as follows. Suppose you are given a knapsack that has a weight capacity W, and you are given a collection of n items that you may choose to put in your knapsack, where item i has value v_i and weight w_i. Your task is to select a subset of the items with maximum total value, such that the items fit into the weight constraint. That is, compute a set of items satisfying

$$\arg\max_{S\subseteq\{0,1,\ldots,n-1\}} \left\{ \sum_{i\in S} v_i \mid \sum_{i\in S} w_i \leq W \right\}.$$

   (a) Show that the following greedy strategies does not solve the problem

      i. Repeatedly select the remaining item with largest value that fits into the knapsack.

      Solution: Suppose you have a knapsack with capacity 3, and items with value / weight pairs (5,1) (5,1) (5,1) and (10,3). Then the greedy algorithm will pick the item with value 10, when it could have selected 3 items of value 5 for a total value of 15.

      ii. Select the remaining item i that maximizes v_i/w_i that fits into the knapsack.

      Solution: Again suppose you have a knapsack with capacity 3, but this time with value / weight pairs (3,3) and (2,1). The greedy algorithm would select the item of value 2 (which prevents the other item from fitting into the knapsack), but it is possible to get value 3 by choosing the other item.

   (b) Describe how to decompose this problem into subproblems, and give a recurrence that computes the maximum *value* attainable given an instance of the 0-1 Knapsack Problem.

   Let the items be given an id, $1,\ldots,n$. In order to break down the full problem into subproblems, we note that there is some item of highest id (in this case, i is the id of an item) in the knapsack. We therefore break the problem by deciding whether or not to select an item of a given id. We let $K_{i,j}$ denote the maximum value achievable by selecting from items $1,\ldots,i$, when giving a knapsack of weight capacity j. Then we get the following recurrence

$$K_{i,j} = \begin{cases} -\infty & i < 0 \text{ or } j < 0, \\ 0 & i = 0 \text{ and } j \geq 0, \\ 0 & i \geq 0 \text{ and } j = 0, \\ \max\{K_{i-1,j}, K_{i-1,j-w_i} + v_i\} & \text{otherwise.} \end{cases}$$

(c) Show that the dynamic programming paradigm can be applied to this problem, given your choice of problem decomposition. Specifically, show that the 0-1 Knapsack Problem has

i. Optimal substructure

The optimal solution either has item i as the highest id item or it doesn't.

A. Suppose it does, then if we don't maximize the value of the remaining capacity of the knapsack, using the remaining items $1, \ldots, i-1$, then we aren't maximizing the total value.

B. Suppose it doesn't, then if we don't maximize the value with the current capacity of the knapsack, using the remaining items $1, \ldots, i-1$, then we aren't maximizing the total value.

ii. Subproblem overlap

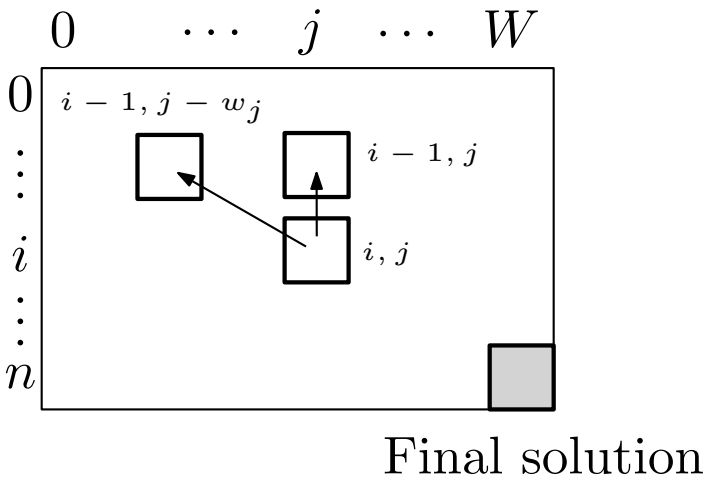
If there are many items with the same weight, or that sum to the same weight, then the same subproblem will be evaluated many times. For instance, consider having multiple items of weight one, two and three. There are many ways to choose items such that their weights sum to three.

iii. Independent subproblems

The two subproblems ($K_{i-1,j}$ and $K_{i-1,j-w_i}$) are independent; this can be seen as follows:

Without loss of generality, to compute the solution $K_{i-1,j}$, the only subproblems that are evaluated have items with id at most $i-2$, and the subproblem $K_{i-1,j-w_i}$ has item id $i-1$.

(d) Draw the array with index bounds; illustrate the dependence on subproblems and highlight the cell with the maximum value for the total problem. Give pseudocode to fill in the array.



Algorithm 1 Maximize value of selecting items (from among n items) that fit into a knapsack of weight capacity W

proc MAXIMUMVALUE(n, W, $w[1..n]$, $v[1..n]$)

```
1:  K[0..n][0..W] = 0
2:  for j ← 1 to W
3:      max_value = 0
4:      for i ← 1 to n
5:          if w[i] ≤ j and max_value < K[i − 1][j − w[i]] + v[i] then
6:              max_value = K[i − 1][j − w[i]] + v[i]   // Select the ith item
7:          else if max_value < K[i − 1][j] then
8:              max_value = K[i − 1][j]   // Don't select the ith item
9:      K[i][j] = max_value
10: return K[n][W]
```