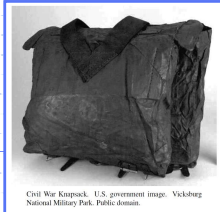


Presentation for use with the textbook, *Algorithm Design and Applications*, by M. T. Goodrich and R. Tamassia, Wiley, 2015

The Greedy Method



Civil War Knapsack. U.S. government image. Vicksburg National Military Park. Public domain.

© 2015 Goodrich and Tamassia

Greedy Method

1

Application: Web Auctions

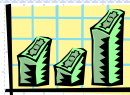
- Suppose you are designing a new **online auction website** that is intended to process bids for multi-lot auctions.
- This website should be able to handle a single auction for 100 units of the same digital camera or 500 units of the same smartphone, where bids are of the form, "**x units for \$y,**" meaning that the bidder wants a quantity of x of the items being sold and is willing to pay \$y for all x of them.
- The challenge for your website is that it must allow for a large number of bidders to place such multi-lot bids and it must decide which bidders to choose as the winners.
- Naturally, one is interested in designing the website so that it always chooses a set of winning bids that maximizes the total amount of money paid for the items being auctioned.
- So how do you decide which bidders to choose as the winners?

© 2015 Goodrich and Tamassia

Greedy Method

2

The Greedy Method



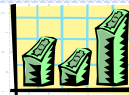
- The greedy method** is a general algorithm design paradigm, built on the following elements:
 - configurations**: different choices, collections, or values to find
 - objective function**: a score assigned to configurations, which we want to either maximize or minimize
- It works best when applied to problems with the **greedy-choice** property:
 - a globally-optimal solution can always be found by a series of local improvements from a starting configuration.

© 2015 Goodrich and Tamassia

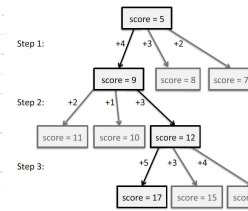
Greedy Method

3

The Greedy Method



- The sequence of choices starts from some well-understood starting configuration, and then iteratively makes the decision that is best from all of those that are currently possible, in terms of improving the objective function.



© 2015 Goodrich and Tamassia

Greedy Method

4

Web Auction Application

- This greedy strategy works for the profit-maximizing online auction problem if you can satisfy a bid to buy x units for \$y by selling $k < x$ units for $\$y/k$.
- In this case, this problem is equivalent to the **fractional knapsack problem**.



© 2015 Goodrich and Tamassia

Greedy Method

5

Web Auctions and the Fractional Knapsack Problem

- In the **knapsack problem**, we are given a set of n items, each having a weight and a benefit, and we are interested in choosing the set of items that maximize our total benefit while not going over the weight capacity of the knapsack.
- In the web auction application, each bid is an item, with its "weight" being the number of units being requested and its benefit being the amount of money being offered.
- In the instance, where bids can be satisfied with a partial fulfillment, then it is an instance of the **fractional** knapsack problem, for which the greedy method works to find an optimal solution.
- Interestingly, for the "0-1" version of the problem, where fractional choices are not allowed, then the greedy method may not work and the problem is potentially very difficult to solve in polynomial time.

© 2015 Goodrich and Tamassia

Greedy Method

6

The Fractional Knapsack Problem



- Given: A set S of n items, with each item i having
 - b_i - a positive benefit
 - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W .
- If we are allowed to take fractional amounts, then this is the **fractional knapsack problem**.
 - In this case, we let x_i denote the amount we take of item i
- Objective: maximize $\sum_{i \in S} b_i(x_i / w_i)$
- Constraint: $\sum_{i \in S} x_i \leq W$

© 2015 Goodrich and Tamassia

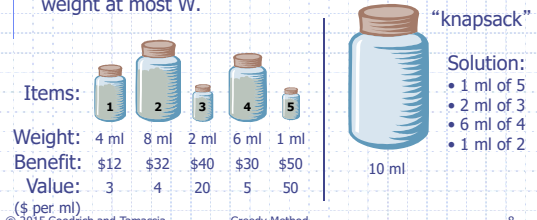
Greedy Method

7

Example



- Given: A set S of n items, with each item i having
 - b_i - a positive benefit
 - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W .



© 2015 Goodrich and Tamassia

Greedy Method

8

The Fractional Knapsack Algorithm



- Greedy choice: Keep taking item with highest **value** (benefit to weight ratio)
 - Since $\sum b_i(x_i / w_i) = \sum (b_i / w_i) x_i$
 - Run time: $O(n \log n)$. Why?
- Correctness: Suppose there is a better solution
 - there is an item i with higher value than a chosen item j , but $x_i < w_i$, $x_j > 0$ and $v_i < v_j$
 - If we substitute some i with j , we get a better solution
 - How much of i : $\min\{w_i - x_i, x_j\}$
 - Thus, there is no better solution than the greedy one

Algorithm fractionalKnapsack(S, W)

Input: set S of items w/ benefit b_i and weight w_i ; max. weight W

Output: amount x_i of each item i to maximize benefit w/ weight at most W

```

for each item  $i$  in  $S$ :
     $x_i \leftarrow 0$ 
     $v_i \leftarrow b_i / w_i$  {value}
     $w \leftarrow 0$  {total weight}
while  $w < W$ :
    remove item  $i$  w/ highest  $v_i$ 
     $x_i \leftarrow \min\{w_i, W - w\}$ 
     $w \leftarrow w + \min\{w_i, W - w\}$ 
  
```

© 2015 Goodrich and Tamassia

Greedy Method

9

Analysis of Greedy Algorithm for Fractional Knapsack Problem

- We can sort the items by their benefit-to-weight values, and then process them in this order.
- This would require $O(n \log n)$ time to sort the items and then $O(n)$ time to process them in the while-loop.
- To see that our algorithm is correct, suppose, for the sake of contradiction, that there is an optimal solution better than the one chosen by this greedy algorithm.
- Then there must be two items i and j such that

$$x_i < w_i, x_j > 0, \text{ and } v_i > v_j.$$
- Let $y = \min\{w_i - x_i, x_j\}$.
- But then we could replace an amount y of item j with an equal amount of item i , thus increasing the total benefit without changing the total weight, which contradicts the assumption that this non-greedy solution is optimal.

© 2015 Goodrich and Tamassia

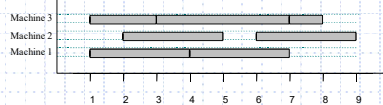
Greedy Method

10

Task Scheduling



- Given: a set T of n tasks, each having:
 - A start time, s_i
 - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



© 2015 Goodrich and Tamassia

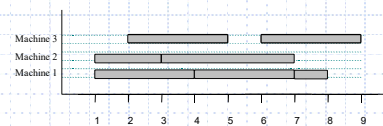
Greedy Method

11

Example



- Given: a set T of n tasks, each having:
 - A start time, s_i
 - A finish time, f_i (where $s_i < f_i$)
- [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines



© 2015 Goodrich and Tamassia

Greedy Method

12

Task Scheduling Algorithm



- ◆ Greedy choice: consider tasks by their start time and use as few machines as possible with this order.
 - Run time: $O(n \log n)$. Why?
- ◆ Correctness: Suppose there is a better schedule.
 - We can use $k-1$ machines
 - The algorithm uses k
 - Let i be first task scheduled on machine k
 - Task i must conflict with $k-1$ other tasks
 - But that means there is no non-conflicting schedule using $k-1$ machines

Algorithm *taskSchedule(T)*

Input: set T of tasks w/ start time s_i and finish time f_i
Output: non-conflicting schedule with minimum number of machines
 $m \leftarrow 0$ {no. of machines}
while T is not empty
 remove task i w/ smallest s_i
 if there's a machine j for i then
 schedule i on machine j
 else
 $m \leftarrow m + 1$
 schedule i on machine m

© 2015 Goodrich and Tamassia

Greedy Method

13

Text Compression

- ◆ Given a string X , efficiently encode X into a smaller string Y
 - Saves memory and/or bandwidth
- ◆ A good approach: **Huffman encoding**
 - Compute frequency $f(c)$ for each character c .
 - Encode high-frequency characters with short code words
 - No code word is a prefix for another code
 - Use an optimal encoding tree to determine the code words

© 2015 Goodrich and Tamassia

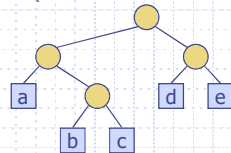
Greedy Method

14

Encoding Tree Example

- ◆ A **code** is a mapping of each character of an alphabet to a binary code-word
- ◆ A **prefix code** is a binary code such that no code-word is the prefix of another code-word
- ◆ An **encoding tree** represents a prefix code
 - Each external node stores a character
 - The code word of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child)

00	010	011	10	11
a	b	c	d	e



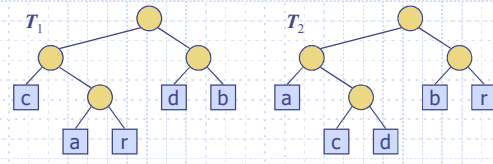
© 2015 Goodrich and Tamassia

Greedy Method

15

Encoding Tree Optimization

- ◆ Given a text string X , we want to find a prefix code for the characters of X that yields a small encoding for X
 - Frequent characters should have long code-words
 - Rare characters should have short code-words
- ◆ Example
 - $X = \text{abracadabra}$
 - T_1 encodes X into 29 bits
 - T_2 encodes X into 24 bits



© 2015 Goodrich and Tamassia

Greedy Method

16

Huffman's Algorithm

- ◆ Given a string X , Huffman's algorithm constructs a prefix code that minimizes the size of the encoding of X
- ◆ It runs in time $O(n + d \log d)$, where n is the size of X and d is the number of distinct characters of X
- ◆ A heap-based priority queue is used as an auxiliary structure

© 2015 Goodrich and Tamassia

Greedy Method

17

Huffman's Algorithm

Algorithm *Huffman(X)*:

Input: String X of length n with d distinct characters

Output: Coding tree for X

Compute the frequency $f(c)$ of each character c of X .

Initialize a priority queue Q .

for each character c in X do

 Create a single-node binary tree T storing c .

 Insert T into Q with key $f(c)$.

while $\text{len}(Q) > 1$ **do**

$(f_1, T_1) = Q.\text{remove_min}()$

$(f_2, T_2) = Q.\text{remove_min}()$

 Create a new binary tree T with left subtree T_1 and right subtree T_2 .

 Insert T into Q with key $f_1 + f_2$.

$(f, T) = Q.\text{remove_min}()$

return tree T

© 2015 Goodrich and Tamassia

Greedy Method

18

