CSEN 703 - Analysis and Design of Algorithms

Lecture 6 - Greedy Algorithms

Dr. Nourhan Ehab
nourhan.ehab@guc.edu.eg

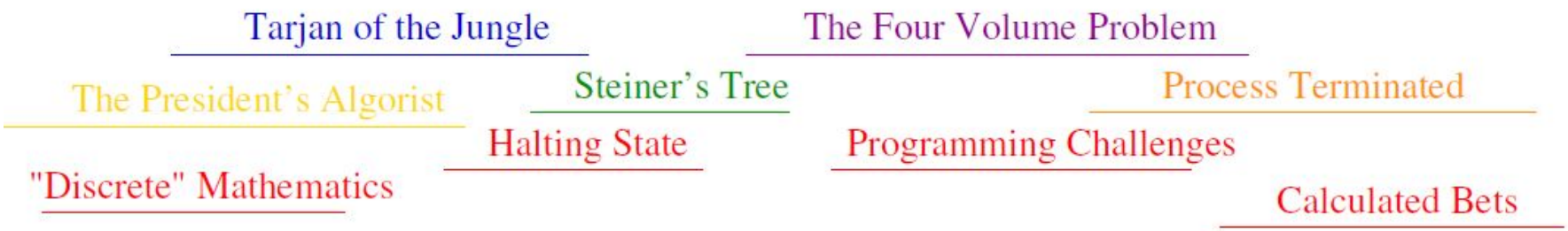
Department of Computer Science and Engineering
Faculty of Media Engineering and Technology

Movie Scheduling Problem

Example

Imagine you are a highly in demand actor, who has been presented with offers to star in n different movie projects under development. Each offer comes specified with the first and last day of filming. Whenever you accept a job, you must commit to being available throughout this entire period. Thus, you cannot accept two jobs whose intervals overlap. For an artist such as yourself, the criterion for job acceptance is clear: you want to make as much money as possible. Because each film pays the same fee, this implies you seek the largest possible set of jobs (intervals) such that no two of them conflict with each other. How can you pick the maximum number of non-overlapping movies?

Movie Scheduling Problem



You can star in maximum 4 movies. Namely Discrete Mathematics, Programming Challenges, Calculated Bets, and one of Halting State or Steiner's Tree.

Possible Solutions

1. Exhaustive Scheduling: List all possible non-overlapping subsets of the available movies and choose the biggest subset
   $\Rightarrow$ But this is in $O(2^n)$
2. Idea! Find a heuristic so that you don't have to consider all 2^n subsets.
   a. Earliest Job First: Work whenever work is available.
   b. Shortest Job First: Choose the shortest movies first.
   c. Earliest Terminating Job First: Choose the movie that terminates the earliest first.

Not All Heuristics Work!

a. Picking the earliest starting movie can make us not take many other movies if it is the longest (like the case of War and Peace)!

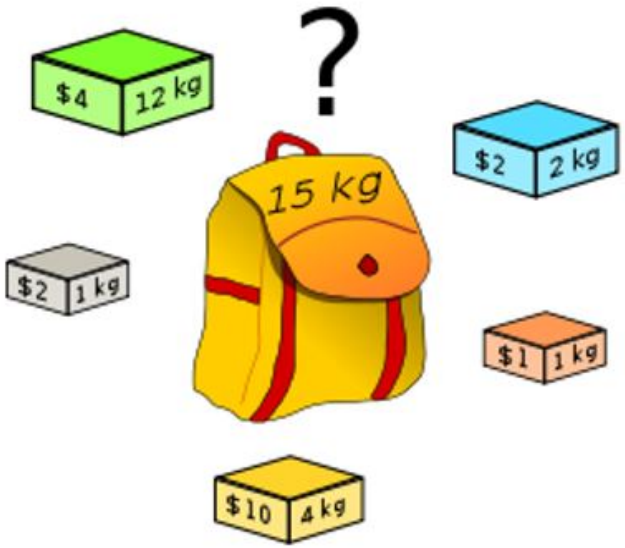
b. Picking the shortest job can block many other overlapping movies (like the right figure)!

c. Picking the earliest terminating job x works! Other jobs may well have started before it, but they must be overlapping. The first of these jobs to terminate is x, so any of the overlapping jobs potentially block out other opportunities to the right of it. x is clearly the best choice.

Greedy Algorithms

- Heuristic based algorithms used for optimization problems.
- A heuristic directs the choices when making decisions in order to reach the best solution.
- A greedy algorithm does not consider all subproblems. A choice that looks the best at the moment is made to choose a subproblem to solve hoping that solving it will find an optimal solution to the original problem.
- In this way, not all subproblems are considered, and the acquired solutions are suboptimal in general.
- Sometimes, we can prove that a heuristic will always yield optimal solutions for a problem. Such problems are said to possess a greedy choice property.

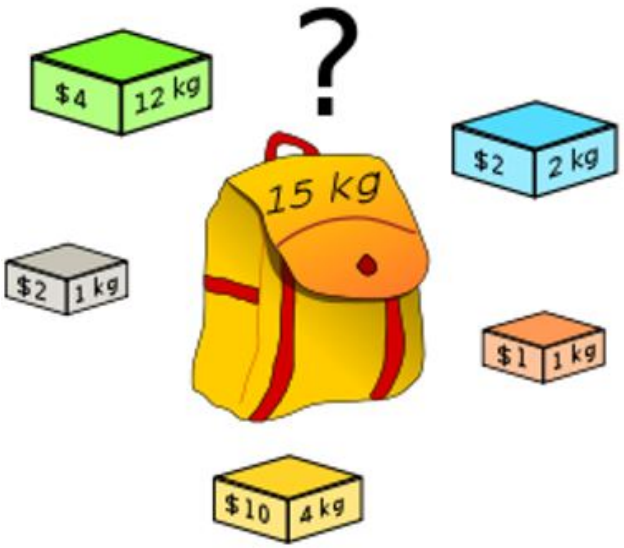
Problem Statement



Example

We are given a set S of n items, such that each item i has a positive value v_i and a positive weight w_i. We wish to find the subset with the maximum total value that does not exceed a given weight W.

Two Variants



1. 0/1 Knapsack: We can only take or leave items.
2. Fractional Knapsack: We can take fractions of items.

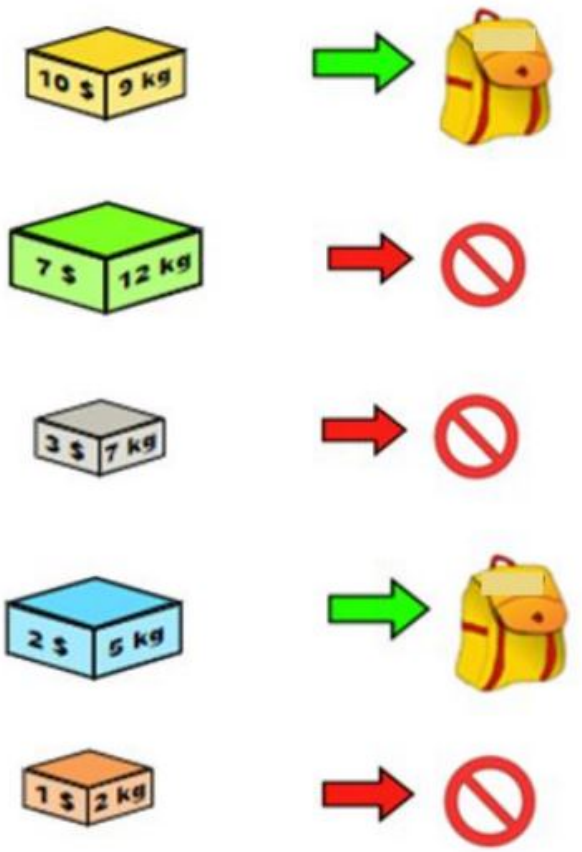
Can we solve both using a greedy algorithm?

0/1 Knapsack

Heuristic 1: Take items with the highest value first.

0/1 Knapsack

Heuristic 1: Take items with the highest value first.



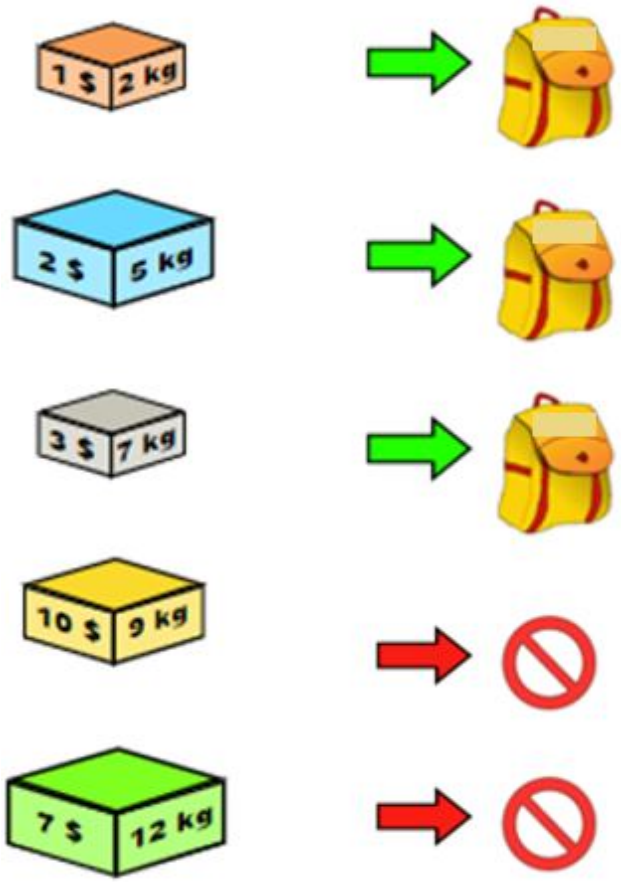
The value of the knapsack is $12\$$. Will this always work?

0/1 Knapsack

Heuristic 2: Take items with the smallest weight first.

0/1 Knapsack

Heuristic 2: Take items with the smallest weight first.



The value of the knapsack is $6\$$. Obviously this is not the best solution.

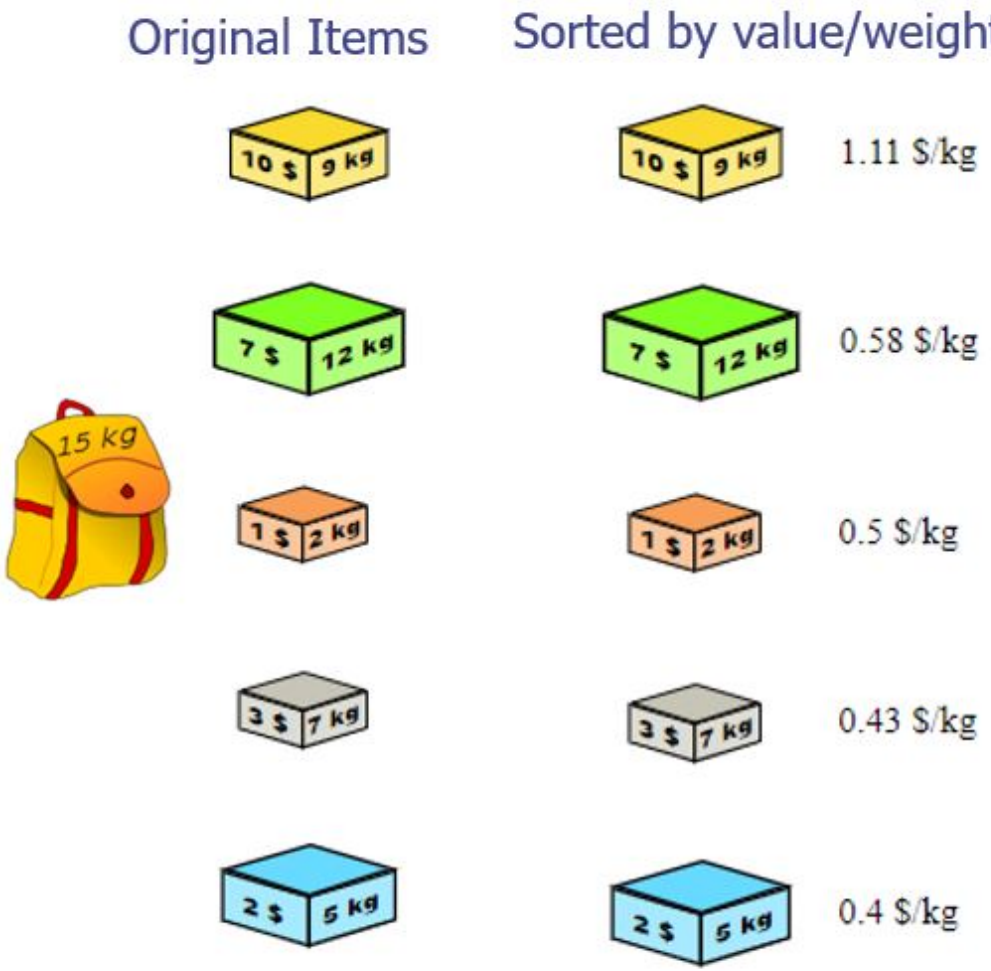
0/1 Knapsack

Key Takeaway

Greedy heuristics do not work for the 0/1 Knapsack problem.

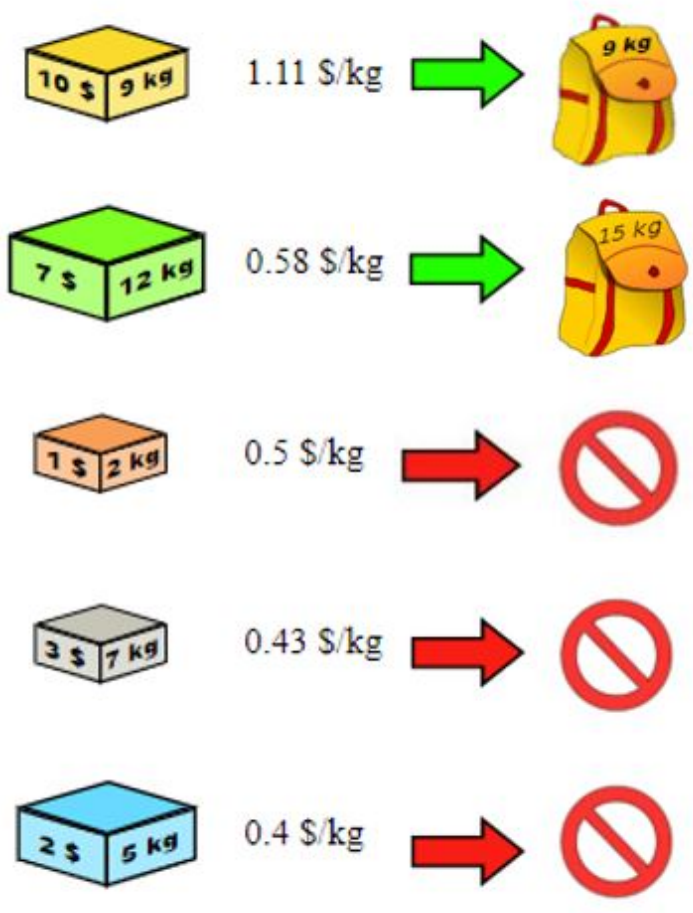
Fractional Knapsack

Heuristic: Take the item with the highest value to weight ratio.



Fractional Knapsack

Heuristic: Take the item with the highest value to weight ratio.



The total value is $13.48\$$. Why doesn't this work for 0/1?

Fractional Knapsack Greedy Algorithm

```
1  FractionalKnapSack(S, w, v, W)
2  r, x = empty array of the same size as w;
3  for each item i in S do
4  |   x_i = 0;
5  |   r_i = v_i / w_i;
6  end
7  currentW = 0;
8  while currentW < W do
9  |   Remove item i with the highest r_i;
10 |   x_i = min{w_i, W − currentW};
11 |   currentW = currentW + x_i;
12 end
13 return x
```

Complexity is $O(n\ log(n))$ where $n = length(S)$.

Fractional Knapsack Algorithm Correctness

- Let item i be the item with the maximum value to weight ratio. We want to show that the optimal solution contains as much of item i as possible. We prove this by contradiction.
- Suppose that there is an optimal solution where we did not take as much of i as possible and we also assume that our knapsack is full.
- Since i is the item with the highest ratio, there must be an item j in the knapsack with a lower ratio $\frac{v_j}{w_j} < \frac{v_i}{w_i}$.
- Suppose we replace item j of weight x by item i of the same weight x. This means that we get a higher value for the knapsack (as the ratio for i is higher).
- But this contradicts our original assumption that the solution was optimal!
- This means that the optimal solution always contains as much of i as possible.

Egyptian Fractions

- The ancient egyptians did not write fractions with a numerator greater than 1.
- Instead they wrote fractions as a sum of different unit fractions.
  $\frac{2}{3} = \frac{1}{3} + \frac{1}{4} + \frac{1}{12}$
  or $\frac{2}{3} = \frac{1}{3} + \frac{1}{5} + \frac{1}{20} + \frac{1}{12}$
  or $\frac{2}{3} = \frac{1}{3} + \frac{1}{6} + \frac{1}{30} + \frac{1}{20} + \frac{1}{12}$
  and so on ...

Example

Given a fraction $\frac{n}{d}$ with $n < d$, generate the shortest equivalent sequence of egyptian fractions.

Egyptian Fractions

- Egyptian Fractions are used to divide things equally on a number of items.
- Suppose we want to divide 3 pizzas equally on 4 people. Since $\frac{3}{4} = \frac{1}{2} + \frac{1}{4}$, this basically means that each of the 4 people gets half a pizza, then each gets quarter a pizza.

Egyptian Fractions Greedy Solution

- Heuristic: Choose the largest possible unit fraction X less than $\frac{n}{d}$.
- But how to get this X? $X = \frac{1}{\lceil \frac{d}{n} \rceil}$. Why?
  - $\frac{d}{n} < \lceil \frac{d}{n} \rceil$.
  - Then, it must be that $\frac{n}{d} > \frac{1}{\lceil \frac{d}{n} \rceil}$.
  - Example: $\frac{11}{12}$.
    $\frac{12}{11} = 1.09$.
    Then, the smallest possible whole number greater than $\frac{12}{11}$ is 2.
    Therefore, $\frac{11}{12} < \frac{1}{2}$.
- Repeat by doing the same thing on $\frac{n}{d} - X$.

Egyptian Fractions Greedy Solution

1 EgyptianFractions($\frac{n}{d}$)
2 **if** $n == 1$ **then**
3 | **return** "$\frac{n}{d}$" ;
4 **end**
5 $newd = \lceil \frac{d}{n} \rceil$;
6 **return** "$\frac{1}{newd}$" + EgyptianFractions($\frac{n}{d} - \frac{1}{newd}$)

Egyptian Fractions Greedy Solution

Example

- EgyptianFractions($\frac{11}{12}$) = $\frac{1}{2}$ - EgyptianFractions($\frac{5}{12}$).
  EgyptianFractions($\frac{5}{12}$) = $\frac{1}{3}$ - EgyptianFractions($\frac{1}{12}$)
  $\frac{11}{12} = \frac{1}{2} + \frac{1}{3} + \frac{1}{12}$
- EgyptianFractions($\frac{2}{3}$) = $\frac{1}{2}$ - EgyptianFractions($\frac{1}{6}$).
  $\frac{2}{3} = \frac{1}{2} + \frac{1}{6}$
- EgyptianFractions($\frac{6}{14}$) = $\frac{1}{3}$ - EgyptianFractions($\frac{2}{21}$).
  EgyptianFractions($\frac{2}{21}$) = $\frac{1}{11}$ - EgyptianFractions($\frac{1}{231}$).
  $\frac{6}{14} = \frac{1}{3} + \frac{1}{11} + \frac{1}{231}$

Applications

- The knapsack problem has lots of applications in financial modelling, inventory management, network traffic control, scheduling, cargo loading, and processing allocation in distributed systems (among many others).
- Egyptian fractions are used in electronic circuit design. It is used for assessing the resistance of electric circuits.
- A lot of graph algorithms are greedy algorithms as we will see later on in the course.