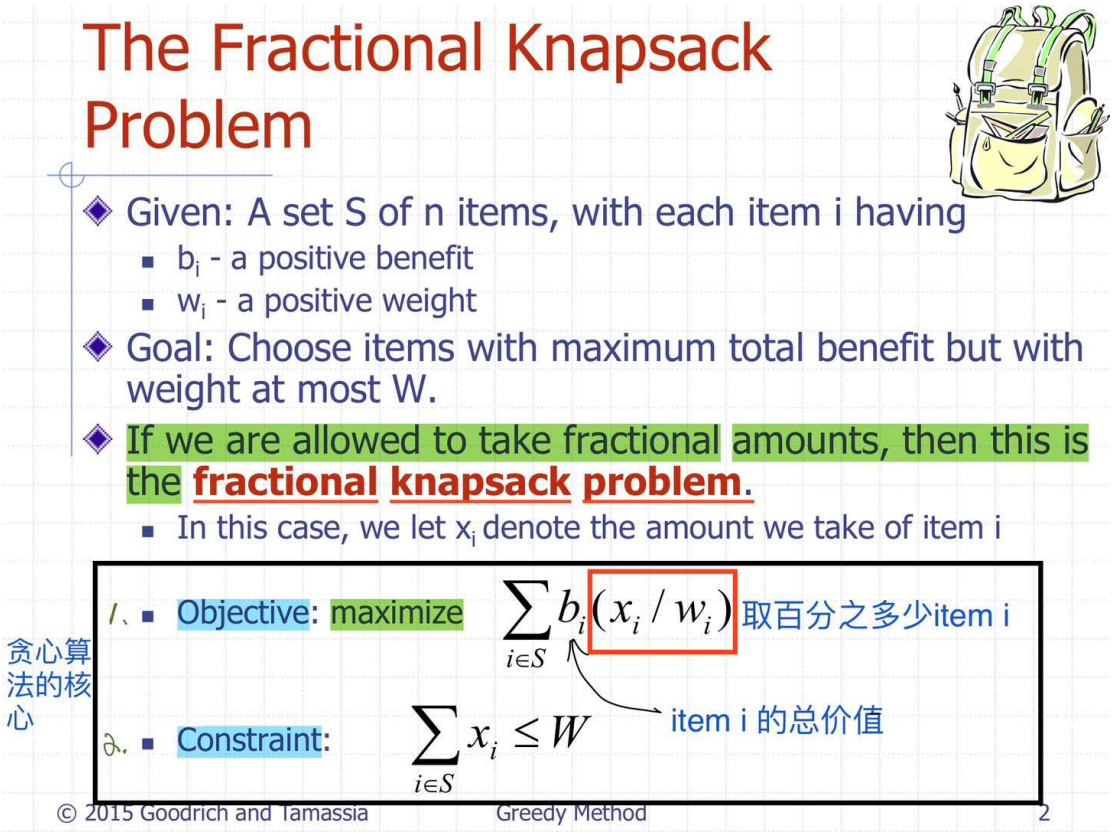
Greedy Algorithm --- Fractional Knapsack

Fractional Knapsack

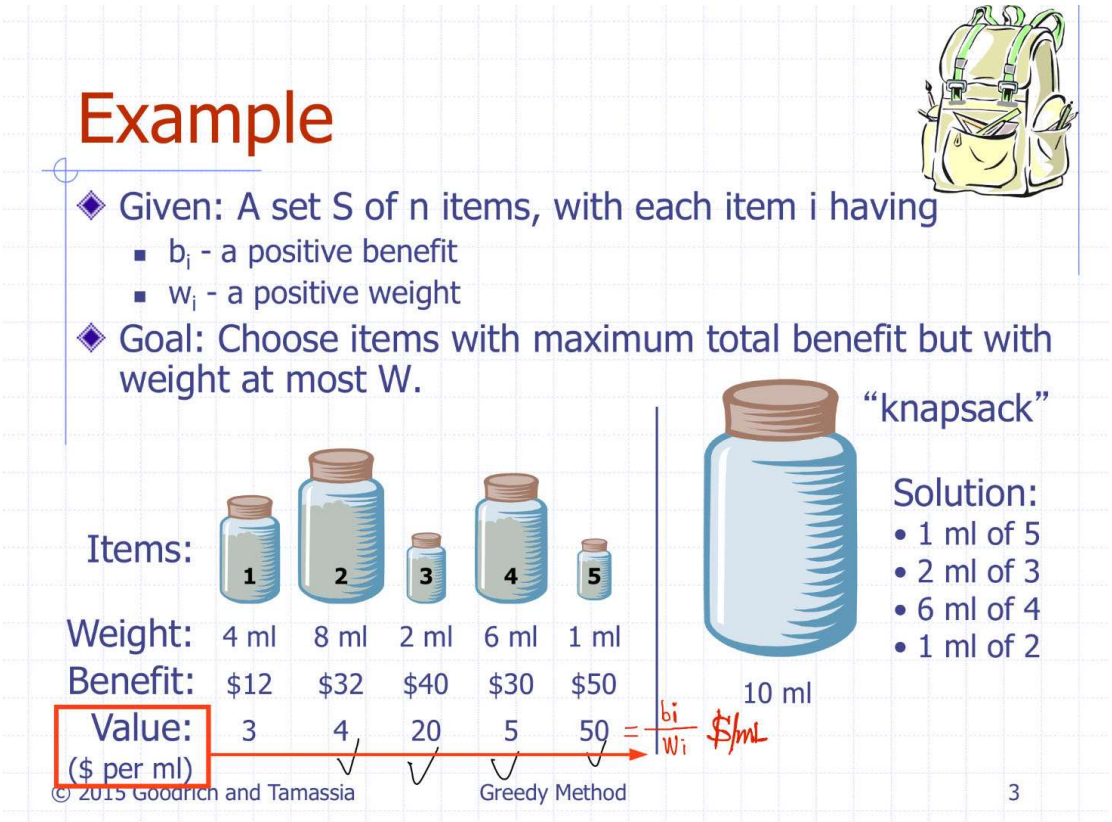
Task Schedule

Explain the problem



注意绿色部分和贪心算法的核心

Give an example or given an instance of the problem, solve it



Explain the algorithm and how are choices made and why, any important properties involved in making such a choice

Algorithm FractionalKnapsack(S, W):
Input: Set S of items, such that each item $i \in S$ has a positive benefit b_i and a positive weight w_i; positive maximum total weight W
Output: Amount x_i of each item $i \in S$ that maximizes the total benefit while not exceeding the maximum total weight W

O(n)
for each item $i \in S$ **do**
$x_i \leftarrow 0$
$v_i \leftarrow b_i/w_i$ // ***value index*** of item i
$w \leftarrow 0$ // total weight

O(n log n)
while $w < W$ and $S \neq \emptyset$ **do**
remove from S an item i with highest value index (v_i) // greedy choice
$a \leftarrow \min\{w_i, W - w\}$ // more than $W - w$ causes a weight overflow
$x_i \leftarrow a$
$w \leftarrow w + a$
return x

Algorithm 10.3: A greedy algorithm for the fractional knapsack problem.

Analyzing the Greedy Algorithm for the Fractional Knapsack Problem

Runtime analysis:

The above FractionalKnapsack algorithm can be implemented in $O(n \log n)$ time, where n is the number of items in S. Specifically, we can use a heap-based priority queue (Section 5.3) to store the items of S, where the key of each item is its value index. With this data structure, each greedy choice, which removes the item with greatest value index, takes $O(\log n)$ time.

Alternatively, we could even 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 of Algorithm 10.3. Either way, we have that the greedy algorithm for solving the fractional knapsack problem can be implemented in $O(n \log n)$ time. The following theorem summarizes this fact and also shows that this algorithm is correct.

Is it greedy or dynamic or some other type of algorithm, explain why?

For fractional Knapsack problem, we do not think about the subproblem overlap. And Sum(local optimal) is the global optimal. And Greedy algorithm is faster than Dynamic programming for this question.

Compare/contrast the problems and algorithms

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

1、 Objective: maximize $\sum_{i \in S} b_i (x_i / w_i)$ 取百分之多少item i

item i 的总价值

贪心算法的核心

2、 Constraint: $\sum_{i \in S} x_i \le W$

Given a scenario would you use 0-1 Knapsack or Fractional Knapsack, or

You give a scenario of when using 0-1 knapsack is appropriate and when Fractional is appropriate

Given Fractional Knapsack algorithm prove it is correct, that it returns an optimal choice of weights of item to be included in the knapsack

Theorem 10.1: *Given a collection S of n items, such that each item i has a benefit b_i and weight w_i, we can construct a maximum-benefit subset of S, allowing for fractional amounts, that has a total weight W in $O(n \log n)$ time.*

Proof: To see that our algorithm (10.3) for solving the fractional knapsack problem 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

$$x_i < w_i, \quad x_j > 0, \quad \text{and} \quad v_i > v_j.$$

因为vi > vj, 所以我们优先装i, 所以我们先装一部分i，即只先装xi的重量，不把wi全部装进去。然后，我们剩y = min { wi – xi , xj }，的重量，全装j，如果将y这些空间交给i item去装，会产生更大的利润，产生矛盾，贪心最好。

Let

$$y = \min\{w_i - x_i, x_j\}.$$

But then we could then 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. Therefore, we can correctly compute optimal amounts for the items by greedily choosing items by increasing benefit-to-weight values. ■

The proof of this theorem uses an ***exchange argument*** to show that the greedy method works to solve this problem optimally. The general structure of such an argument is a proof by contradiction, where we assume, for the sake of reaching a contradiction, that there is a better solution than one found by the greedy algorithm. We then argue that there is an exchange that we could make among the components of this solution that would lead to a better solution. In this case, this approach shows

理论思想：Exchange arguement

为什么Greedy Algorithm is best rather than non-greedy solution?

我们回答这个考点时一定要说：

Assume there is a non-greedy solution which is better than the greedy algorithim.

xi < wi, xj > 0, and vi > vj.

Let y = min{wi - xi, xj}.

We **replace** an amout y of item j with an equal amount of item i, thus increasing the total benefit without changing the total weight. It is contradicts the assumption that this non-greedy solution sis optimal. So, Greedy Algo is better!

replace 这个词就是exchange arguement的核心！！！