Design and Analysis of Algorithm

Lecture-10:
Greedy Algorithm

Algorithm

```
Algorithm JS(d, j, n)
// d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1. The jobs
// are ordered such that p[1] ≥ p[2] ≥ ··· ≥ p[n]. J[i]
// is the ith job in the optimal solution, 1 ≤ i ≤ k.
// Also, at termination d[J[i]] ≤ d[J[i + 1]], 1 ≤ i < k.
{
    d[0] := J[0] := 0; // Initialize.
    J[1] := 1; // Include job 1.
    k := 1;
    for i := 2 to n do
    {
        // Consider jobs in nonincreasing order of p[i]. Find
        // position for i and check feasibility of insertion.
        r := k;
        while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r)) do r := r − 1;
        if ((d[J[r]] ≤ d[i]) and (d[i] > r)) then
        {
            // Insert i into J[ ].
            for q := k to (r + 1) step −1 do J[q + 1] := J[q];
            J[r + 1] := i; k := k + 1;
        }
    }
    return k;
}
```

Given that

$(p_1, p_2, p_3, p_4) = (100,10,15,27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Sort jobs based on profit

$(p_1, p_4, p_3, p_2) = (100,27,15,10)$

$(d_1, d_4, d_3, d_2) = (2, 1, 2, 1)$

Initialize

J_0	J_1	J_2	J_3	J_4
0	1			

Complexity of the algorithm

For the given algorithm there are two parameters in terms of which its complexity can be measured.

1. *Total number of jobs (say n)*
2. *Number of jobs selected (say s)*

There are two loops (1) For loop (2) While loop

For Loop

It will run maximum of n-1 times, So complexity is o(n)

While Loop

In any iteration it will run for k times

Maximum possible value of k is s

The complexity of the given algorithm will thus be O(ns)

Theorems

Theorem Let J be a set of k jobs and $\sigma = i_1, i_2, \ldots, i_k$ a permutation of jobs in J such that $d_{i_1} \le d_{i_2} \le \cdots \le d_{i_k}$. Then J is a feasible solution iff the jobs in J can be processed in the order σ without violating any deadline.

Theorem The greedy method described above always obtains an optimal solution to the job sequencing problem.

Example

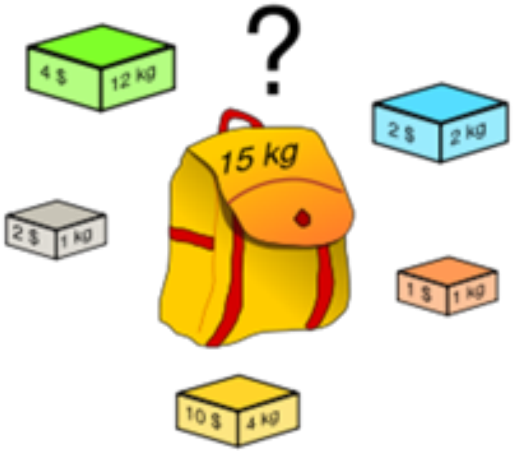
Let n = 5, $(p_1, p_2, p_3, P_4, p_5)$ = (20,15,10,5,1) and (d_1, d_2, d_3, d_4, d_5) = (2,2,1,3,3)

J	Assigned slot	Job considered	Action	Profit
ϕ	none	1	Assign to (1,2)	0
{1}	[1,2]	2	Assign to (0,1)	20
{1,2}	[0,1][1,2]	3	Reject	35
{1,2}	[0,1][1,2]	4	Assign to (2,3)	35
{1,2,4}	[0,1][1,2][2,3]	5	Reject	40

Example

The **knapsack problem** derives its name from the maximization problem of the best choice of essentials that can fit into one bag to be carried on a trip.

Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than a given limit and the total value is as large as possible.



Knapsack Problem

Problem Definition

Want to carry essential items in one bag

Given a set of items, each has

- A cost (i.e., 12kg)
- A value (i.e., 4$)

Goal

To determine the number of each item to include in a collection so that

- The total cost is less than some given cost
- And the total value is as large as possible

The Original Knapsack Problem

Three Types

- 0/1 Knapsack Problem
  Restricts the number of each kind of item to zero or one
- Bounded Knapsack Problem
  Restricts the number of each item to a specific value
- Unbounded Knapsack Problem
  Places no bounds on the number of each item

Fractional Knapsack

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

According to the problem statement,

- There are **n** items in the store
- Weight of i^{th}item $w_i > 0$
- Profit for i^{th} item $pi > 0$ and
- Capacity of the Knapsack is W

In this version of Knapsack problem, items can be broken into smaller pieces. So, one can choose a fraction of item as well.

$\boldsymbol{x_i}\, of\ i^{th}$

The $\boldsymbol{i}^{th}$ item contributes the weight x_i, w_i to the total weight in the knapsack and profit $x_i . p_i$ to the total profit.

Fractional Knapsack Problem

In fractional knapsack problem,

we are given a set S of n items,

each item I has a positive benefit b_i and

a positive weight w_i,

we wish to find the maximum-benefit subset that doesn't exceed a given weight W.

The Fractional Knapsack Problem: Formal Definition

Given K and a set of n items

weight	w_1	w_2	...	w_n
value	v_1	v_2	...	v_n

Find: $0 \le x_i \le 1, \; i = 1, 2, \ldots, n$ such that

The following is maximized:

$$\sum_{i=1}^{n} x_i v_i$$

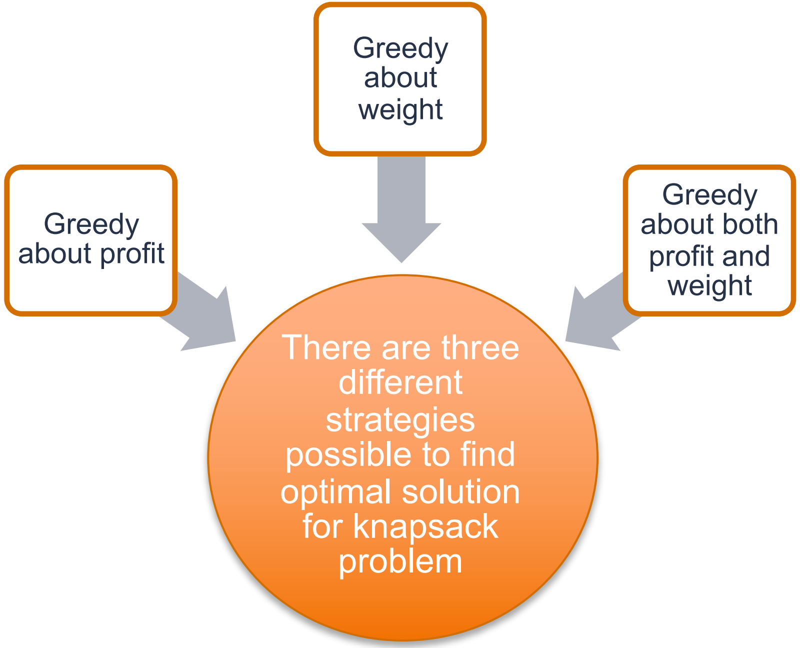
It satisfies the following constraint

$$\sum_{i=1}^{n} x_i w_i \le K$$

Objective Function

Constraints

The Fractional Knapsack Problem: Solution



Greedy about both profit and weight

Item	Weight	Value
1	18	25
2	15	24
3	10	15

Compute value (or profit per unit of weight)

So value of Item 1 per unit weight is $25/18 = 1.3$

Similarly, value of Item 2 per unit weight is $24/15 = 1.6$

and value of Item 3 per unit weight is $15/10 = 1.5$

Calculation

1) Calculate the value-per-kg $\rho i = v_i / W_i \; for \; i = 1,2, \dots, n.$

2) Sort the items by decreasing ρ_i

   Let the sorted item sequence be $1,2, \dots, i, \dots n,$ the corresponding value-per-kg and weight be ρ_i and w_i respectively.

3) Let k be the current weight limit (Initially, k = K). In each iteration, we choose item I from the top of the unselected list.
   1) If $k \geq w_i$, set $x_i = 1$ (we take item i), and reduce $k = k - w_i$, then consider the next unselected item.
   2) If $k < w_i$, set $x_i = k/w_i$ (we take a fraction k/w_i of item i), Then the algorithm terminates.

Fractional Knapsack Algorithm

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

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}

while $w < W$ **do**

remove from S an item i with highest value index {greedy choice}

$a \leftarrow \min\{w_i, W - w\}$ {more than $W - w$ causes a weight overflow}

$x_i \leftarrow a$

$w \leftarrow w + a$

Observation

- Observe that the algorithm may take a fraction of an item.
- This can only be the last selected item.
- We claim that the total value for this set of items is the optimal value.

Feasible and Optimal solution

A feasible solution is any set $(x_1, x_2, x_3, \ldots\ldots\ldots x_n)$ which satisfies the constraint.

An optimal solution is a feasible solution for which the objective function is maximized.

Question: Consider the following instance of the knapsack problem:

n = 3,m= 20,

(p_1, p_2, P_3) = (25,24,15), and

(w_1, w_2, w_3)= (18,15,10).

Four feasible solutions are

	x_1, x_2, x_3	$\sum w_i x_i$	$\sum p_i x_i$
1	1/2,1/3,1/4	16.5	24.25
2	1,2/15,0	20	28.2
3	0,2/3,1	20	31
4	0,1,1/2	20	31.5

Question

<u>Question:</u> Consider the following instance of the knapsack problem. The max capacity of knapsack is 12

	ob_1	Ob_2	Ob_3	Ob_4	Ob_5
p	5	2	2	4	5
w	5	4	6	2	1

Compute profit per unit weight for each object.

	ob_1	Ob_2	Ob_3	Ob_4	Ob_5
p/w	1	0.5	0.33	2	5

$solution \quad 1,1,0,1,1$

Question

Question: Consider the following instance of the knapsack problem:

Item	X1	X2	X3	X4	X5
Profit	15	12	9	16	17
Weight	2	5	3	4	6

The maximum weight of 12 is allowed in the knapsack.

Find the value of maximum profit with the optimal solution of the fractional knapsack problem.

Question

Question: Consider the following instance of the knapsack problem:

	ob_1	Ob_2	Ob_3	Ob_4	Ob_5	Ob_6	Ob_7
p	10	5	15	7	6	18	3
w	2	3	5	7	1	4	1

The maximum weight of 15 is allowed in the knapsack.