Design and Analysis of Algorithms

Lecture # 6

GREEDY ALGORITHMS

Greedy Algorithms

- General Characteristics of greedy algorithms
- Elements of Greedy Strategy
- Examples
  - The Knapsack Problem
  - Job Scheduling Problem

Characteristics of Greedy Algorithms

- Greedy algorithms are characterized by the following features.

1. Greedy approach forms a set or list of candidates $\boldsymbol{C}$.
2. Once a candidate is selected in the solution, it is there forever: once a candidate is excluded from the solution, it is never reconsidered.
3. To construct the solution in an optimal way, Greedy Algorithm maintains two sets.
4. One set contains candidates that have already been considered and chosen, while the other set contains candidates that have been considered but rejected.

Characteristics of Greedy Algorithms

- The greedy algorithm consists of four functions.

  i. **Solution Function**:- A function that checks whether chosen set of items provides a solution.

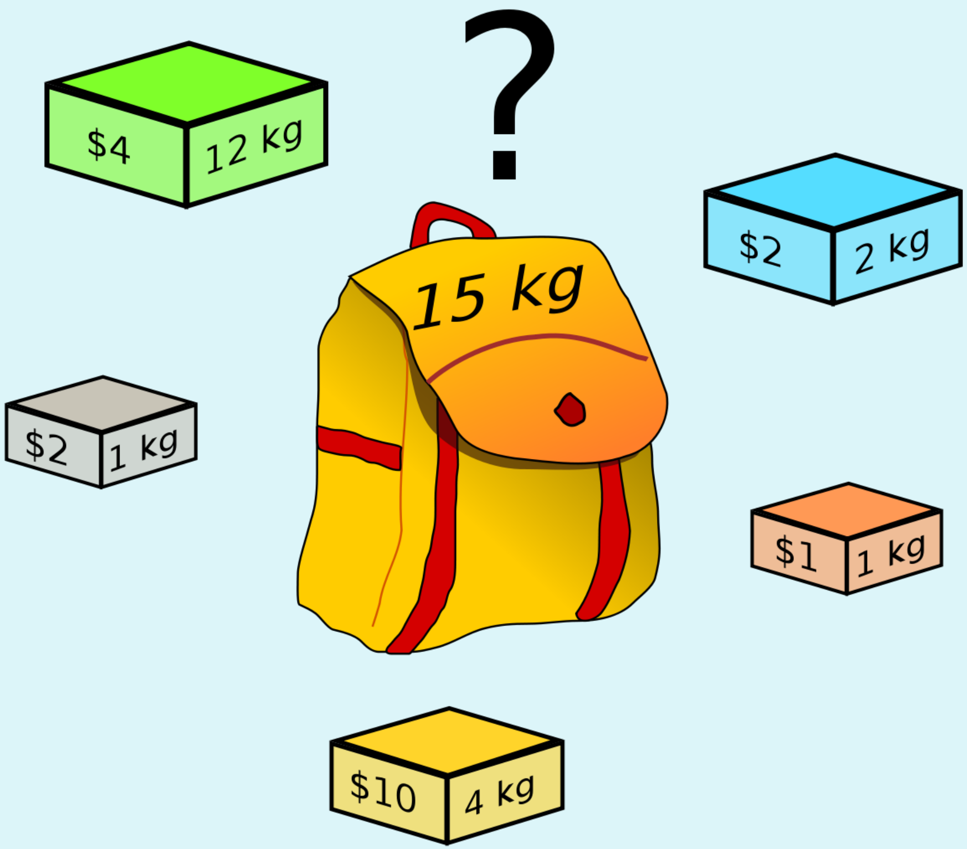
  ii. **Feasible Function**:- A function that checks the feasibility of a set.

  iii. **Selection Function**:- The selection function tells which of the candidates is the most promising.

  iv. **Objective Function**:- An objective function, which does not appear explicitly, but gives the value of a solution.

KNAPSACK PROBLEM

Knapsack Problem



FRACTIONAL KNAPSACK PROBLEM

Introduction

- We are given n objects and a knapsack.
- Object $\boldsymbol{i}$ has a positive weight $\boldsymbol{w_i}$ and a positive value $\boldsymbol{v_i}$ for $\boldsymbol{i} = \mathbf{1}, \mathbf{2} \ldots \boldsymbol{n}$.
- The knapsack can carry a weight not exceeding $\boldsymbol{W}$.
- Our aim is to fill the knapsack in a way that **maximizes** the value of the included objects, while respecting the capacity constraint.
- In a fractional knapsack problem, we assume that the objects **can be broken into smaller pieces**.

Introduction

- So we may decide to carry only a fraction x_i of object i, where $0 \leq x_i \leq 1$.
- In this case, object i contribute $x_i w_i$ to the total weight in the knapsack, and $x_i v_i$ to the value of the load.
- Symbolic Representation of the problem can be given as follows:

maximize $\sum_{i=1}^{n} x_i v_i$ **subject to** $\sum_{i=1}^{n} x_i w_i \leq W$

Where, $v_i > 0$, $w_i > 0$ and $0 \leq x_i \leq 1$ for $1 \leq i \leq n$.

Fractional Knapsack Problem - Example

- We are given 5 objects and the weight carrying capacity of knapsack is $\boldsymbol{W = 100}$.
- For each object, weight w_i and value v_i are given in the following table.

Object i	1	2	3	4	5
v_i	20	30	66	40	60
w_i	10	20	30	40	50

- Fill the knapsack with given objects such that the total value of knapsack is **maximized.**

Fractional Knapsack Problem - Greedy Solution

- Three Selection Functions can be defined as,

  1. Sort the items in **descending order of their values** and select the items till weight criteria is satisfied.
  2. Sort the items in **ascending order of their weight** and select the items till weight criteria is satisfied.
  3. To calculate the **ratio value/weight** for each item and sort the item on basis of this ratio. Then take the item with the highest ratio and add it.

Fractional Knapsack Problem - Greedy Solution

Object i	1	2	3	4	5
v_i	20	30	66	40	60
w_i	10	20	30	40	50

Selection	**Objects**					**Value**
	1	**2**	**3**	**4**	**5**	
Max v_i						
Min w_i						
Max v_i/w_i						

Weight Capacity
100

30	50	20	
10	20	30	40
30	10	20	40

Profit = 66 + 20 + 30 + 48 = 164

Fractional Knapsack Problem - Algorithm

```
Algorithm: Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)
for i = 1 to n do
      x[i] ← 0 ; weight ← 0
While weight < W do
i← the best remaining object
      if weight + w[i] ≤ W then
      x[i] ←1
      weight ← weight + w[i]
      else
      x[i] ← (W - weight) / w[i]
      weight ← W
return x
```

W = 100 and Current weight in knapsack= 60; Object weight = 50
The fraction of object to be included will be **(100 – 60) / 50 = 0.8**

JOB SCHEDULING WITH DEADLINES

Introduction

- We have set of n jobs to execute, each of which **takes unit time**.
- At any point of time we can **execute only one job.**
- Job i earns profit $g_i > 0$ if and only if it is executed **no later than** its deadline d_i.
- We have to find an optimal sequence of jobs such that our total **profit is maximized**.
- Feasible jobs: A set of job is feasible if there exits **at least one sequence** that allows all the jobs in the set to be executed no later than their respective deadlines.

Job Scheduling with Deadlines - Example

- Using greedy algorithm find an optimal schedule for following jobs with $\boldsymbol{n = 6}$.
- Profits: $(P_1, P_2, P_3, P_4, P_5, P_6) = (15,20,10,7,5,3)$ &
- Deadline: $(d_1, d_2, d_3, d_4, d_5, d_6) = (1, 3, 1, 3, 1, 3)$

Solution:

Sort the jobs in **decreasing order** of their profit.

Step 1:

Job i	**1**	**2**	**3**	**4**	**5**	**6**
Profit g_i	**20**	**15**	**10**	**7**	**5**	**3**
Deadline d_i.	**3**	**1**	**1**	**3**	**1**	**3**

Job Scheduling with Deadlines - Example

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

Step 2: Find total position $\boxed{P = \min(n, \max(di))}$

Here, $P = \min(6,3) = \mathbf{3}$

P	1	2	3
Job selected	0	0	0

Step 3: $d_1 = 3$: assign job 1 to position 3

P	1	2	3
Job selected	0	0	J1

Job Scheduling with Deadlines - Example

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

Step 4: $d_2 = 1$: assign job 2 to position 1

P	1	2	3
Job selected	J2	0	J1

Step 5: $d_3 = 1$: assign job 3 to position 1

But position 1 is already occupied and two jobs can not be executed in parallel, so reject job 3

Job Scheduling with Deadlines - Example

Job i	1	2	3	4	5	6
Profit g_i	20	15	10	7	5	3
Deadline d_i.	3	1	1	3	1	3

Step 6:

$d_4 = 3$: assign job 4 to position 2 as, position 3 is not free but position 2 is free.

P	1	2	3
Job selected	J2	J4	J1

Now **no more free position** is left so no more jobs can be scheduled.
The final optimal sequence:

Execute the job in order 2, 4, 1 with total profit value 42.

Job Scheduling with Deadlines - Algorithm

Algorithm: Job-Scheduling (P[1..n], D[1..n])

1. Sort all the n jobs in decreasing order of their profit.
2. Let total position P = min(n, max(d_i))
3. Each position 0, 1, 2…, P is in different set and T({i}) = i, for 0 ≤ i ≤ P.
4. Find the set that contains d, let this set be K. if T(K) = 0 reject the job; otherwise:
    1. Assign the new job to position T(K).
    2. Find the set that contains T(K) – 1. Call this set L.
    3. Merge K and L. the value for this new set is the old value of T(L).