Unit 6: Greedy Algorithms

Greedy algorithms

- A *greedy algorithm* always makes the choice that looks best at the moment
  - My everyday examples:
    - Driving in Baroda
    - Playing cards
    - Invest on stocks
    - Choose a university
  - The hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works
- greedy algorithms tend to be easier to code

Why it is Greedy?

- Greedy in the sense that it leaves as much opportunity as possible for the remaining activities to be scheduled
- The greedy choice is the one that maximizes the amount of unscheduled time remaining

Characteristic of Greedy Algorithm

- ***Solution function*** checks whether a particular set of candidates provides a solution to problem.(ignoring optimal solution)
- ***Feasible function*** checks whether or not it is possible to complete the set by adding further candidates so as to obtain at least one solution
- ***Selection function*** indicates at any time which of the remaining candidates is most promising
- ***Objective function*** gives value of the solution

Characteristic of Greedy Algorithm

- Greedy(C: Candidate set) S← Φ

```
While C != Φ and not solution(S) x← select(C)
        C= C / {x}
        if(feasible(S U{x}) then S← S U {x}
end
If solution(S) return S
else
      return “No Solution found”
end
```

Elements of Greedy Strategy

- An greedy algorithm makes a sequence of choices, each of the choices that seems best at the moment is chosen
  - NOT always produce an optimal solution
- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy
  - Greedy-choice property
  - Optimal substructure

Greedy-Choice Property

A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.

- Make whatever choice seems best at the moment and then solve the sub-problem arising after the choice is made.
- The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to sub-problems.

Of course, we must prove that a greedy choice at each step yields a globally optimal solution.

Optimal Substructures

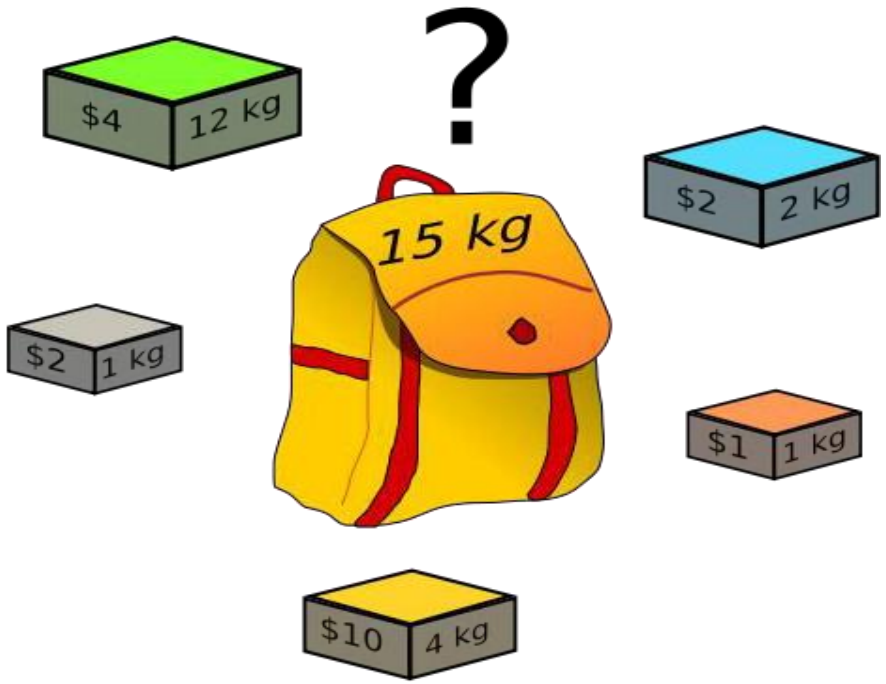
- A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems.

Fractional Knapsack Problem

A thief considers taking W Kg of loot.

The loot is in the form of n items, each with weight w_i and value v_i.

Any amount of an item can be put in the knapsack ,weight limit W is not exceeded.



Fractional Knapsack Problem

Item	Value	Weight
1	4	12
2	2	1
3	10	4
4	1	1
5	2	2

Let's select items which gives me highest value

Item 3 Profit =10 Remaining weight is 15-4 = 11

Item 1 Profit =10+4*(11/12) = 10 + 3.66 = 13.66
Remaining weight is 11 -11 =0

Fractional Knapsack Problem

Item	Value	Weight
1	4	12
2	2	1
3	10	4
4	1	1
5	2	2

Let's select items with lowest weight that keeps knapsack empty for long period

Item 4 Profit =1 Remaining weight is 15-1 = 14

Item 2 Profit =1+2 Remaining weight is 14-1 = 13

Item 5 Profit =1+2+2 Remaining weight is 13-2 = 11

Item 3 Profit =1+2+2+10 =15
Remaining weight is 11-4 = 7

Item 1 Profit =15+4*(7/12) = 15 + 2.33 = 17.33
Remaining weight is 7 -7 =0

Fractional Knapsack Problem

Item	Value	Weight	Value/weight
1	4	12	4/12= 0.33
2	2	1	2/1 = 2
3	10	4	10/4 = 2.5
4	1	1	1/1= 1
5	2	2	2/2 = 1

Let's select items with highest profit per unit quantity

Item 3 Profit =10 Remaining weight is 15-4= 11

Item 2 Profit =10+2 Remaining weight is 11-1 = 10

Item 5 Profit =10+2+2 Remaining weight is 10-2 = 8

Item 4 Profit =10+2+2+1 =15
Remaining weight is 8-1 = 7

Item 1 Profit =15+4*(7/12) = 15 + 2.33 = 17.33
Remaining weight is 7 -7 =0

Example: Given $n = 5$ objects and a knapsack capacity $W = 100$ as in Table I. Three solutions???? .

w	10	20	30	40	50
v	20	30	66	40	60
v/w	2.0	1.5	2.2	1.0	1.2

Table I

select	x_i					value
Max v_i	0	0	1	0.5	1	146
Min w_i	1	1	1	1	0	156
Max v_i/w_i	1	1	1	0	0.8	164

Table II

There seem to be 3 obvious greedy strategies:

(Max value) Sort the objects from the highest value to the lowest, then pick them in that order.

(Min weight) Sort the objects from the lowest weight to the highest, then pick them in that order.

(Max value/weight ratio) Sort the objects based on the value to weight ratios, from the highest to the lowest, then select.

Greedyknapsack(v[1..n], w[1..n], W)

```
For i= 1 to n
    x[i]=0
    ratio[i]= v[i]/w[i]
end
Weight=0
Profit=0
While (weight <W)
      i= select item from highest ratio to lowest  ratio in order
      if (weight+w[i] <= W )then
                  x[i]=1
                   profit=profit +x[i]*v[i]
                  weight= weight + w[i]
     else
                   x[i]=   (W-weight)/w[i]
                   profit=profit  +x[i]*v[i]
                   weight= W.
     end
end
Return profit.
```

Complexity

Greedy-fractional-knapsack (w, v, W)

- **Input:** an integer *n*, positive values *wi* and *vi* , for $1 \le i \le n$, n_n and another positive value *W*.
- **Output:** *n* values *xi* such that $0 \le xi \le 1$ and $\sum_{i=1}^{n} x_i w_i \le W$ and $\sum_{i=1}^{n} x_i v_i$ is maximized.
- x[i] =0

  weight = 0

  Sort the *n* objects from large to small based on the ratios *vi*/*wi* . The arrays *w*[1..*n*] and *v*[1..*n*] store the respective weights and values after sorting. → $\theta(n \log n)$

  while ($i \le n$ and weight < *W*) → $\theta(n)$

    do i = best remaining item

    IF weight + w[i] $\le$ W

    then x[i] = 1

    weight = weight + w[i]

    i++

    else

    x[i] = (w - weight) / w[i]

    weight = W

    i++

  return x

$$\rightarrow T(n) = \theta(n \log n)$$

- **Time Complexity-**
-

- The main time taking step is the sorting of all items in decreasing order of their value / weight ratio.
- If the items are already arranged in the required order, then while loop takes O(n) time.
- The average time complexity of **Quick Sort** is O(nlogn).
- Therefore, total time taken including the sort is O(nlogn).

Greedy Algorithm for Fractional Knapsack problem

Fractional knapsack can be solvable by the greedy strategy

- Compute the value per unit weight v_i/w_i for each item
- Obeying a greedy strategy, take as much as possible of the item with the greatest value per unit weight.
- If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per unit weight, and so forth until there is no more room
- $O(n \lg n)$ (we need to sort the items by value per unit)

PRACTICE PROBLEM (FRACTIONAL KNAPSACK PROBLEM)

- Find the optimal solution for the fractional knapsack problem making use of greedy approach. Consider-
- n = 5
- w = 60 kg
- (w1, w2, w3, w4, w5) = (5, 10, 15, 22, 25)
- (b1, b2, b3, b4, b5) = (30, 40, 45, 77, 90)

Job Scheduling Problem

Job Sequencing With Deadlines

The sequencing of jobs on a single processor with deadline constraints is called as Job Sequencing with Deadlines.

Here-

- You are given a set of jobs.
- Each job has a defined deadline and some profit associated with it.
- The profit of a job is given only when that job is completed within its deadline.
- Only one processor is available for processing all the jobs.
- Processor takes one unit of time to complete a job.

The problem states-

“How can the total profit be maximized if only one job can be completed at a time?”

Approach to Solution

- ✓ A feasible solution would be a subset of jobs where each job of the subset gets completed within its deadline.
- ✓ Value of the feasible solution would be the sum of profit of all the jobs contained in the subset.
- ✓ An optimal solution of the problem would be a feasible solution which gives the maximum profit.

Greedy Algorithm

Greedy Algorithm is adopted to determine how the next job is selected for an optimal solution.
The greedy algorithm described below always gives an optimal solution to the job sequencing problem-

Step-01:

Sort all the given jobs in decreasing order of their profit.

Step-02:

- Check the value of maximum deadline.
- Draw a Gantt chart where maximum time on Gantt chart is the value of maximum deadline.

Step-03:

- Pick up the jobs one by one.
- Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before its deadline.

Greedy Algorithm

Problem-

Given the jobs, their deadlines and associated profits as shown-

Jobs	J1	J2	J3	J4	J5	J6
Deadlines	5	3	3	2	4	2
Profits	200	180	190	300	120	100

Answer the following questions-

1.Write the optimal schedule that gives maximum profit.

2.Are all the jobs completed in the optimal schedule?

3.What is the maximum earned profit?

Greedy Algorithm

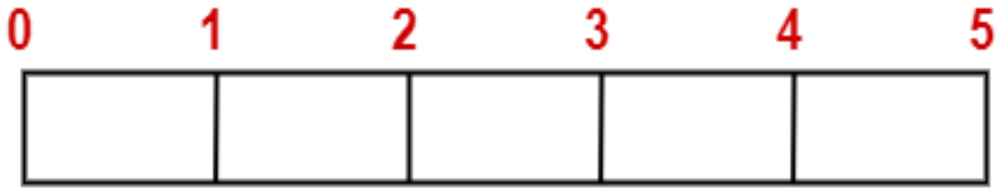
Solution-

Step-01: Sort all the given jobs in decreasing order of their profit-

Jobs	J4	J1	J3	J2	J5	J6
Deadlines	2	5	3	3	4	2
Profits	300	200	190	180	120	100

Step-02:

Value of maximum deadline = 5.
So, draw a Gantt chart with maximum time on Gantt chart = 5 units as shown-



Gantt Chart

- We take each job one by one in the order they appear in Step-01.
- We place the job on Gantt chart as far as possible from 0.

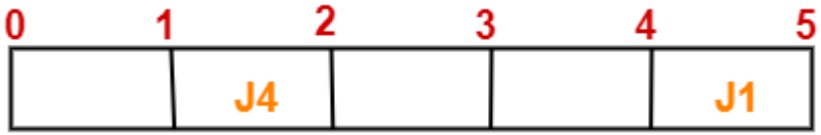
Greedy Algorithm

Step-03:

•We take job J4.
•Since its deadline is 2, so we place it in the first empty cell before deadline 2 as-

Step-04:

•We take job J1.
•Since its deadline is 5, so we place it in the first empty cell before deadline 5 as-



Step-05:

•We take job J3.
•Since its deadline is 3, so we place it in the first empty cell before deadline 3 as-

Greedy Algorithm

Step-06:

•We take job J2.
•Since its deadline is 3, so we place it in the first empty cell before deadline 3.
•Since the second and third cells are already filled, so we place job J2 in the first cell as-

Step-07:

•Now, we take job J5.
•Since its deadline is 4, so we place it in the first empty cell before deadline 4 as-

Now,
•The only job left is job J6 whose deadline is 2.
•All the slots before deadline 2 are already occupied.
•Thus, job J6 can not be completed.

Solution – Job Scheduling Problem

Part-01:

The optimal schedule is:- **J2 , J4 , J3 , J5 , J1**
This is the required order in which the jobs must be completed in order to obtain the maximum profit.

Part-02:

•All the jobs are not completed in optimal schedule.
•This is because job J6 could not be completed within its deadline.

Part-03:

Maximum earned profit
= Sum of profit of all the jobs in optimal schedule
= Profit of job J2 + Profit of job J4 + Profit of job J3 + Profit of job J5 + Profit of job J1
= 180 + 300 + 190 + 120 + 200
= 990 units