GREEDY METHOD

Dr. KUPPUSAMY P

SCOPE

Content

- **Greedy Method:**
  - General method,
  - Fractional Knapsack problem,
  - Job Scheduling with Deadlines,
  - Minimum cost Spanning Trees,
  - Single-source shortest paths,

Greedy Algorithm

- It follows the problem solving heuristic of making each decision is locally optimal. These locally optimal solutions will finally add up to a globally optimal solution.
- In many problems, a greedy strategy does not produce an optimal solution in general.
- Locally optimal solutions may produce approximate global optimal solution in a reasonable time.
- In general, greedy algorithms have **five** components:
  1. A **candidate set**, from which a solution is created.
  2. A **selection function**, which chooses the best candidate to be added to the solution.
  3. A **feasibility function**, that is used to determine if a candidate can be used to contribute to a solution.
  4. An **objective function**, which assigns a value to a solution, or a partial solution, and
  5. A **solution function**, which will indicate when discovered a complete solution.

Greedy algorithms produce good solutions for some mathematical problems.

Knapsack problem using Greedy Method

- Given a set of items, consider each item with a weight w_i, and a value p_i.
- Determine a subset of items to include in a collection of items.
- So that the total weight is less than **or** equal to a given limit (Capacity), and
- The **total value** is as **large** as possible.

Greedy Knapsack algorithm:

Step 1: Calculate the preciousness ($\frac{p_i}{w_i}$) for each item.

Step 2: Sort preciousness into non-increasing order.

Step 3: Put the objects into the knapsack according to the sorted sequence as possible as we can.

Knapsack problem Algorithm and Notations

- **n** is number of objects,
- Each with a weight $w_i > 0$; profit (cost) is $p_i > 0$
- Capacity of knapsack is C.
- Object Preciousness => $\frac{Cost}{Weight} = \frac{p_i}{w_i}$

  Maximize $\sum_{i=1}^{n} p_i * x_i$

  Subject to $\sum_{i=1}^{n} w_i * x_i \leq C$;

  Here, $0 \leq x_i \leq 1$ & $1 \leq i \leq$ n
- **<u>Two types of Knapsack</u>** :
  1. 0 /1 knapsack, and
  2. Fractional knapsack

0 / 1 Knapsack algorithm Example

- Four objects has been given with weights 2, 3, 4, 5, and its cost value is 3, 7, 2, 9 respectively.
- Solve this using knapsack method with capacity 5 kg.

Solution

n = 4, C = 5, (p_1, p_2, p_3, p_4) = (3, 7, 2, 9) & (w_1, w_2, w_3, w_4) = (2, 3, 4, 5)

Calculate object's Preciousness : $\frac{p_1}{w_1} = \frac{3}{2} = 1.5$; $\frac{p_2}{w_2} = \frac{7}{3} = \mathbf{2.3}$; $\frac{p_3}{w_3} = \frac{2}{4} = 0.5$; $\frac{p_4}{w_4} = \frac{9}{5} = 1.8$

Sort **Preciousness** in descending order **2.3**, 1.8, 1.5, 0.5. Choose maximum value.

1. Hence choose **2.3** with Weight 3 kg and Profit is 7 . Now Remaining Capacity 5 – 3 = 2 kg
2. Choose next most value **1.8** with weight 5 kg. But its capacity is only 2 kg.

   Its 0 / 1 Knapsack. i.e. either choose entire item or look for another item.
3. So, Choose next most value **1.5** with weight 2 kg.

   => Now knapsack is full with weight 5 kg.

Optimal solution: x_1 = 1 (1 item chosen), x_2 = 1 (1 item Chosen), x_3 = 0 (Not chosen) x_4 = 0 (Not chosen)

Total cost (profit) = $\sum_{i=1}^{n} p_i * x_i$ =(3*1) + (7*1) = 3+7 = 10.

Fractional Knapsack algorithm Example

- Three objects has been given with weights 18, 15 10, and its cost value is 25, 24, 15 respectively.
- Solve this using knapsack method with capacity 20 kg.

Solution

n = 3, C = 20, $(p_1, p_2, p_3) = (25, 24, 15)$ & $(w_1, w_2, w_3) = (18, 15, 10)$.

Calculate object's Preciousness : $\frac{p_1}{w_1} = \frac{25}{18} = 1.39$; $\frac{p_2}{w_2} = \frac{24}{15} = \mathbf{1.6}$; $\frac{p_3}{w_3} = \frac{15}{10} = 1.5$;

- Sort **Preciousness** in descending order **1.6,**1.5, 1.39. Choose maximum value.

1. Hence choose **1.6** with Weight 15 kg and Profit is 24 . Now Remaining Capacity 20 – 15 = 5 kg.
2. Choose next most value **1.5** with weight 10 kg. But its capacity is only 5 kg. Choose 5 Kg.

Profit for 5 kg = $(\frac{15}{10}) * 5$

= 7.5 => **Now knapsack is full with weight 20 kg.**

Optimal solution: x_1 = 0 (Not chosen), x_2 = 1 (1 item Chosen), $x_3 = \frac{1}{2} = 0.5$ (5kg out of 10kg chosen)

- **Total cost (profit)** = $\sum_{i=1}^{n} p_i * x_i$ =(24 * 1) + (0.5 * 15) = 24+7.5 = **31.5**

Fractional Knapsack algorithm Example

- Four objects has been given with weights 2, 3, 4, 5, and its cost value is 3, 7, 2, 9 respectively.
- Solve this using knapsack method with capacity 5 kg.

Solution

n = 4, C = 5, (p_1, p_2, p_3 , p_4) = (3, 7, 2, 9) & (w_1, w_2, w_3, w_4) = (2, 3, 4, 5)

Calculate object's Preciousness : $\frac{p_1}{w_1} = \frac{3}{2} = 1.5$; $\frac{p_2}{w_2} = \frac{7}{3} = \mathbf{2.3}$; $\frac{p_3}{w_3} = \frac{2}{4} = 0.5$; $\frac{p_4}{w_4} = \frac{9}{5} = 1.8$

Sort **Preciousness** in descending order **2.3**, 1.8, 1.5, 0.5. Choose maximum value.

1. Hence choose **2.3** with Weight 3 kg and Profit is 7 . Now Remaining Capacity 5 – 3 = 2 kg
2. Choose next most value **1.8** with weight 5 kg. But its capacity is only 2 kg. Choose 2 kg out of 5 kg.

Profit for 2 kg= $(\frac{9}{5}) * 2$

= 3.6 **=> Now knapsack is full with weight 5 kg.**

Optimal solution: x_1 = 0 (Not chosen), x_2 = 1 (1 item Chosen), x_3 = 0 (Not chosen) $x_4 = \frac{2}{5} = 0.4$ (2kg out of 5kg chosen)

Total cost (profit) = $\sum_{i=1}^{n} p_i * x_i$ =(7 *1) + (9 * 0.4) = 7+3.6 = **10.6**

Time complexity of 0/1 & Fractional Knapsack

- Always, greedy algorithm does **not provide** the optimal solution for the 0/1 knapsack problem.
- Because this algorithm always considers the full object. So, It can not maintain (fill) the entire knapsack capacity every time.
- Thus the greedy method does not always guarantee the optimal solution for the **0/1 knapsack** problem.
- So, **greedy approach** is **not solves** the 0/1 knapsack problem.
- But the fraction knapsack problem considers the part (fraction) of the object that fill the entire capacity of the knapsack and maximize the profit.
- Thus the greedy method guarantees the optimal solution for the **fractional problem**.

Time complexity of Fractional Knapsack

- Step 1: Calculate the preciousness ($\frac{p_i}{w_i}$) for each item. - ***C***
- Step 2: Sort preciousness into non-increasing order. - ***n * log n***
- Step 3: Put the objects into the knapsack according to the sorted sequence. - ***C***

Steps:

- Sorting the profits can be done using many algorithms.
- Apply best sorting merge sort **or** heap sort that consumes O(nlogn) for best, average and worst case.
- It is the optimal time among all sorting algorithms.
- Hence, time taken in fractional knapsack is ***O(n log n).***
- If **selection** or **bubble sort** is applied, then the time complexity is $O(n^2)$.

Applications

- Finding the least wasteful way to cut raw materials

  - Pieces cut from a single sheet of wood, fabric, metal, etc.,

- Portfolio optimization
  - Achieving a higher expected return on investment.
- Cutting stock problems

  - cargo ships or trucks load.

  - containers contents.

  - packing pallets.

- Internet Download Manager
  - The data is broken into chunks.
  - As per the maximum size of data that can be retrieved in one attempt, the server uses this algorithm and packs the chunks so as to utilize the full size limit.

Job sequencing with deadlines using Greedy Method

Problem Statement:

- n jobs, S={1, 2, …, n}, each job *i* has a deadline $d_i \geq 0$ and a profit $p_i \geq 0$.
- User need one unit of time to process each job, and
- User can do at most one job at a time with maximum profit.
- Earn the profit p_i, if job *i* is completed within its deadline.

Algorithm (Maximize the profit):

Step 1: Sort p_i into non-increasing (descending) order. After sorting $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_n$.

Step 2: Add the next job *i* to the solution set, if *i* can be completed within its deadline.

Assign *i* to time slot *[r-1, r]*, where *r* is the largest integer (deadline time period) such that $1 \leq r \leq d_i$, and *[r-1, r]* is free. If *[r-1, r]* is **not free**, reject the job.

Step 3: Stop if all jobs are examined. Otherwise, go to step 2.

Time complexity: $O(n^2)$ => Sorting & Assign into time slot.

Job sequencing with deadlines: **Example**

- Find optimal solution for the given problem below:

Job(*i*)	***1***	***2***	***3***	***4***	***5***
Profit (p_i)	20	15	10	5	1
Deadline (d_i)	2	2	1	3	3

Step 1: Sort the jobs in non increasing order using profit value.

i	p_i	d_i
1	**20**	2
2	**15**	2
3	**10**	1
4	**5**	3
5	**1**	3

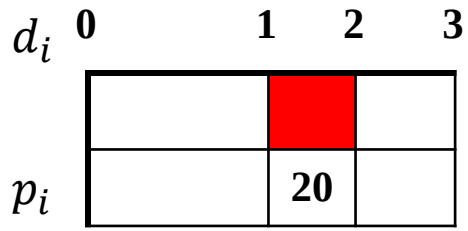
Job sequencing with deadlines: **Example**

Step 2: Add job i to the solution set in time slot [r-1, r], if ***i*** can be completed by its deadline. where *r* is the largest integer such that $1 \leq r \leq d_i$ and *[r-1, r]* is free.

If *[r-1, r]* is **not free**, reject the job.

i	p_i	d_i	Consider	Assign (Max deadline)
1	20	2	[0,1], [1, 2]	[1, 2]
2	15	2		
3	10	1		
4	5	3		
5	1	3		

Solution Set

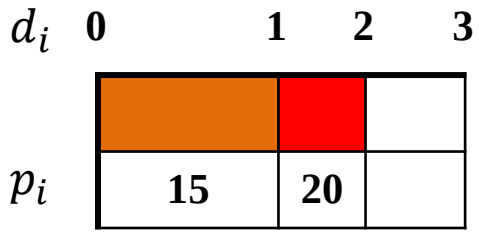


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	20	2	[0,1], [1, 2]	[1, 2]
2	15	2	[0,1], [1, 2]. But [1, 2] already filled.	[0,1]
3	10	1		
4	5	3		
5	1	3		

Solution Set

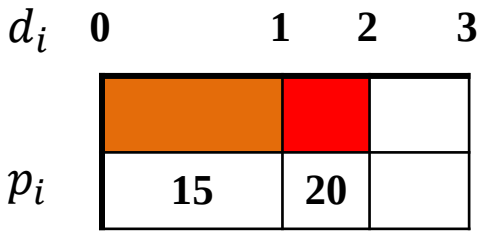


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	20	2	[0,1], [1, 2]	[1, 2]
2	15	2	[0,1], [1, 2]. But [1, 2] already filled.	[0,1]
3	10	1	[0,1] But [0, 1] already filled.	Reject
4	5	3		
5	1	3		

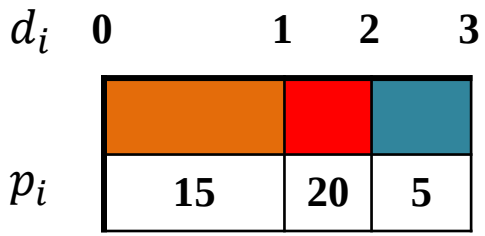
Solution Set



Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	20	**2**	[0,1], [1, 2]	[1, 2]
2	15	**2**	[0,1], [1, 2]. But [1, 2] already filled.	[0,1]
3	10	**1**	[0,1] But [0, 1] already filled.	Reject
4	5	**3**	[0,1], [1, 2], [2,3] But [0, 1], [1, 2] already filled.	[2,3]
5	1	**3**		

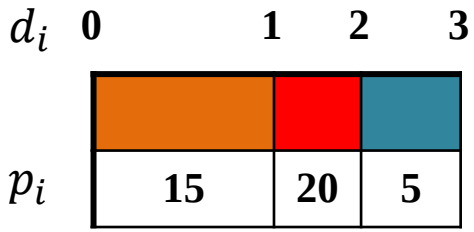
Solution Set



Step 3: Repeat until all jobs are examined.

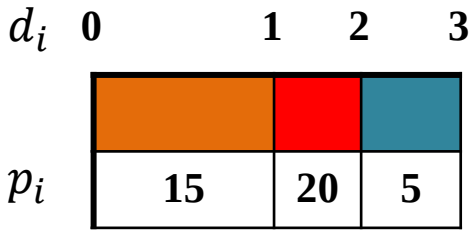
i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	20	**2**	[0,1], [1, 2]	[1, 2]
2	15	**2**	[0,1], [1, 2]. But [1, 2] already filled.	[0,1]
3	10	**1**	[0,1] But [0, 1] already filled.	Reject
4	5	**3**	[0,1], [1, 2], [2,3] But [0, 1], [1, 2] already filled.	[2,3]
5	1	**3**	[0,1], [1, 2], [2,3] But [0, 1], [1, 2] , [2,3] already filled.	Reject

Solution Set



All jobs are examined. Calculate Optimal Solution (Profit).

Solution Set



- All jobs are examined. Calculate Optimal Solution (Profit).
- The optimal solution = {1, 2, 4}.
- The total profit = 20 + 15 + 5 = 40.

Applications:

- Job scheduling to processor
- Flight landing or take off in airport
- Car repair workshop

Job sequencing with deadlines: **Example 2**

- Find optimal solution for the given problem below:

Job(*i*)	1	2	3	4	5
Profit (p_i)	30	12	70	25	20
Deadline (d_i)	2	3	2	1	1

Step 1: Sort the jobs in non increasing order using profit value.

i	p_i	d_i
1	70	2
2	30	2
3	25	1
4	20	1
5	12	3

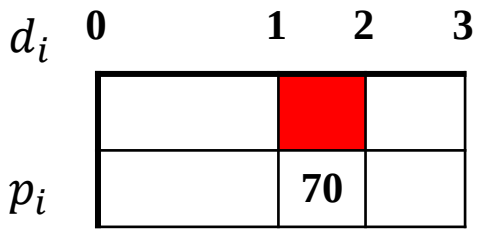
Job sequencing with deadlines: **Example**

Step 2: Add job i to the solution set in time slot [r-1, r], if i can be completed by its deadline. where r is the largest integer such that $1 \leq r \leq d_i$ and [r-1, r] is free.

If [r-1, r] is **not free**, reject the job.

i	p_i	d_i	Consider	Assign (Max deadline)
1	70	2	[0,1], [1, 2]	[1, 2]
2	30	2		
3	25	1		
4	20	1		
5	12	3		

Solution Set

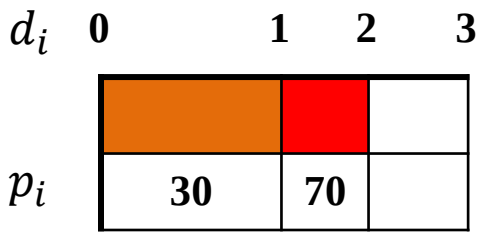


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	70	2	[0,1], [1, 2]	[1, 2]
2	30	2	[0,1], [1, 2] But [1, 2] already filled.	[0,1]
3	25	1		
4	20	1		
5	12	3		

Solution Set

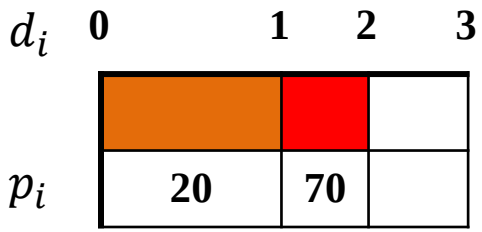


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	70	2	[0,1], [1, 2]	[1, 2]
2	30	2	[0,1]	[0,1]
3	25	1	[0,1], [1, 2] But [0, 1], [1, 2] already filled.	Reject
4	20	1		
5	12	3		

Solution Set

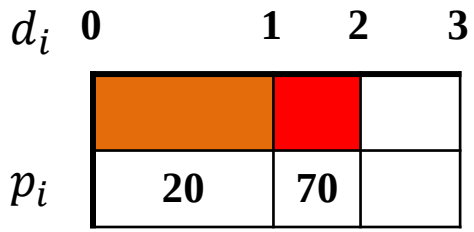


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	70	2	[0,1], [1, 2]	[1, 2]
2	30	2	[0,1]	[0,1]
3	25	1	[0,1], [1, 2] But [0, 1], [1, 2] already filled.	Reject
4	20	1	[0,1]. But [0, 1] already filled.	Reject
5	12	3		

Solution Set

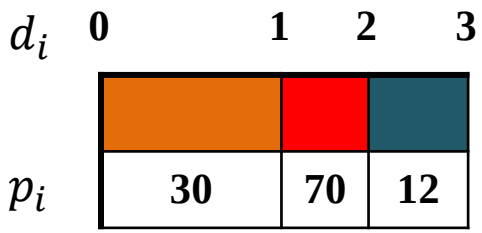


Job sequencing with deadlines: **Example**

Step 3: Repeat until all jobs are examined.

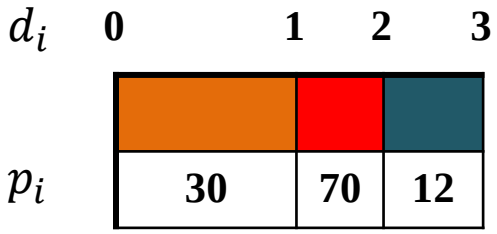
i	p_i	d_i	**Consider**	**Assign (Max deadline)**
1	70	2	[0,1], [1, 2]	[1, 2]
2	30	2	[0,1]	[0,1]
3	25	1	[0,1], [1, 2] But [0, 1], [1, 2] already filled.	Reject
4	20	1	[0,1]. But [0, 1] already filled.	Reject
5	12	3	[0,1], [1, 2], [2,3]	[2,3]

Solution Set



All jobs are examined. Calculate Optimal Solution (Profit).

Solution Set



- All jobs are examined. Calculate Optimal Solution (Profit).
- The optimal solution = {1, 2, 5}.
- The total profit = 30 + 70 + 12 = 112.

References

1. Anany Levitin, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, Third Edition, 2017.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, MIT Press/PHI Learning Private Limited, Third Edition, 2012.
3. E. Horowiz, S. Sahni, S. Rajasekaran, “Fundamentals of Computer Algorithms”, Universities Press, 2nd Ed, 2008.