```
2<- 4  The length = 3
Step by Step shortest path is...
3<-1<-2<- 4  The length = 7
Step by Step shortest path is...
4  The length = 0
```

Review Questions

1. *Design and explain Dijkstra's shortest path algorithm.* **GTU : June-11, Marks 3**
2. *Explain Dijkstra's algorithm to find minimum distance of all nodes from a given node. (Greedy algorithm)* **GTU : Summer-12, Marks 4**

5.9 Knapsack Problem

GTU : June-11, Summer-17,18, Winter-19, Marks 7

The Knapsack problem can be stated as follows.

Suppose there are n objects from i = 1, 2,...n. Each object i has some positive weight w_i and some profit value is associated with each object which is denoted as p_i. The Knapsack carry at the most weight W.

While solving above mentioned Knapsack problem we have the capacity constraint. When we try to solve this problem using Greedy approach our goal is,

1. Choose only those objects that give maximum profit.
2. The total weight of selected objects should be ≤ W.

And then we can obtain the set of feasible solutions. In other words,

$$\text{maximized} \sum_{n}^{1} p_i x_i \text{ subject to } \sum_{n}^{1} w_i x_i \leq W$$

Where the Knapsack can carry the fraction x_i of an object i such that $0 \leq x_i \leq 1$ and $1 \leq i \leq n$.

Example 5.9.1 *Consider knapsack capacity W = 50, W = (10, 20, 40) and V = (60, 80, 100). Find maximum profit using greedy approach.* **GTU : Summer-17, Marks 7**

Solution : We will first find the feasible solutions. From these feasible solutions we choose the solution that gives us maximum profit. For obtaining feasible solution we use 3 approaches, these are -

1) Select object with maximum profit
2) Select object with minimum weight
3) Select object with maximum P/W or V/W ratio

Method 1 : Select object with maximum profit.

Object	1	2	3
V	60	80	100
W	10	20	40

Selected object	Profit	Weight	Remaining Weight
x_3	100	40	50 – 40 = 10
x_1	60	10	10 – 10 = 0

Hence **maximum profit = 160** with solution (x_3, x_1).

Method 2 : Select object with minimum weight.

Selected object	Profit	Weight	Remaining Weight
x_1	60	10	50 – 10 = 40
x_2	80	20	40 – 20 = 20
x_3	$100 * \frac{1}{2} = 50$	$40 * \frac{1}{2} = 20$	20 – 20 = 0

Hence solution is $\left(x_1, x_2, \frac{x_3}{2}\right)$ with total profit = 190.

Method 3 : Select object with maximum V/W ratio.

V / W	x_1	x_2	x_3
	6	4	2.5

Selected object	Profit	Weight	V/W	Remaining Weight
x_1	60	10	6	50 – 10 = 40
x_2	80	20	4	40 – 20 = 20
x_3	$100 * \frac{1}{2} = 50$	$40 * \frac{1}{2} = 20$	2.5	20 – 20 = 0

Hence solution is $\left(x_1, x_2, \frac{x_3}{2}\right)$ with total profit = **190.**

Example 5.9.2 *Solve the following knapsack problem using greedy method. Number of items = 5. Capacity W = 100. Weight vector {50, 40, 30, 20, 10} and Profit vector = (1,2,3,4,5).*

GTU : Summer-18, Marks 7

Solution : Let, the given data be -

Item	x_1	x_2	x_3	x_4	x_5
Profit (p)	1	2	3	4	5
Weight (w)	50	40	30	20	10
p/w	0.02	0.05	0.1	0.2	0.5

Method 1 : Maximum profit object.

Item selected	Profit	Weight	Remaining Weight
x_5	5	10	100 – 10 = 90
x_4	4	20	90 – 20 = 70
x_3	3	30	70 – 30 = 40
x_2	2	40	40 – 40 = 0

Total profit = 14

Total weight = 100

Method 2 : Minimum weight object.

Item selected	Profit	Weight	Remaining Weight
x_5	5	10	100 – 10 = 90
x_4	4	20	90 – 20 = 70
x_3	3	30	70 – 30 = 40
x_2	2	40	40 – 40 = 0

Total profit = 14

Total weight = 100

Method 3 : Maximum profit / weight ration.

Item selected	Profit	Weight	Remaining Weight
x_5	5	10	100 – 10 = 90
x_4	4	20	90 – 20 = 70
x_3	3	30	70 – 30 = 40
x_2	2	40	40 – 40 = 0

$\therefore$ Total profit = 14

and Total weight = 100

Thus the solution to this problem is (x_5, x_4, x_3, x_2) or (5,4,3,2).

Example 5.9.3 *Consider the instance of the 0/1 (binary) knapsack problem as below with P depicting the value and W depicting the weight of each item whereas M denotes the total weight carrying capacity of knapsack. Find the optimal answer using greedy design technique. Also write the time complexity of greedy approach for solving knapsack problem. P = [40, 10, 50, 30, 60], W = [80, 10, 40, 20, 90], M = 110.*

GTU : Winter-19, Marks 4

Solution : Let n = 5 and M = 110 we will first find out P/W ratio select the objects in decreasing order of P/W ratio.

Objects	1	2	3	4	5
Profit	40	10	50	30	60
Weight	80	10	40	20	90
P/W	$\frac{1}{2} = 0.5$	1	1.25	1.5	0.66

Now each time we will select max $\left(\frac{P}{W}\right)$ value, from remaining list.

Object selected	Profit	Weight	Remaining weight
4	30	20	110 – 20 = 90
3	50	40	90 – 40 = 50
2	10	10	50 – 10 = 40
5 Not selected	60	Can not select as W > remaining capacity	
Total profit = 90. Solution (2, 3, 4)			

From above table, note that we can not select object 5 and object 4 as it is a (0/1) knapsack problem and we can not put fractional weight in knapsack.

Thus now, total weight = 70

total profit = 90

The solution is (x_4, x_3, x_2).

5.9.1 Algorithm

The algorithm for solving Knapsack problem with Greedy approach is as given below.

```
Algorithm Knapsack_Greedy(W,n)
{
//p[i] contains the profits of i items such that 1 ≤ i ≤ n
//w[i] contains weights of I items.
//x[i] is the solution vector.
//W is the total size of Knapsack.
      for i : =1 to n do
{
if[w[i]<W) then//capacity of knapsack is a constraint
{
 x[i] : =1.0;
 W=W – w[i];
}
}
if(i<=n) then x[i] : =W/w[i];
}
```

This is the basic operation of selecting x[i] lies within for loop.
∴ Time complexity = Θ(n).

Review Question

1. *What is a fractional knapsack problem ? Design and analyze greedy algorithm to solve it.*

GTU : June-11, Marks 5

5.10 Job Scheduling Problem

GTU : Winter-11,14,15, Marks 7

Consider that there are n jobs that are to be executed. At any time t = 1,2,3,.. only exactly one job is to be executed. The profits pi are given. These profits are gained by corresponding jobs. For obtaining feasible solution we should take care that the jobs get completed within their given deadlines.

Let n = 4

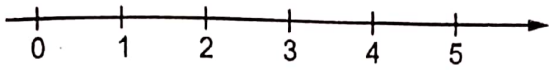
n	pi	di
1	70	2
2	12	1
3	18	2
4	35	1

We will follow following rules to obtain the feasible solution

- Each job takes one unit of time.

- If job starts **before or at its deadline**, profit is obtained, otherwise no profit.
- Goal is to schedule jobs to maximize the total profit.
- Consider all possible schedules and compute the minimum total time in the system.

Consider the Time line as



The feasible solutions are obtained by various permutations and combinations of jobs.

n	pi
1	70
2	12
3	18
4	35
1,3	88
2,1	82
2,3	30
3,1	88
4,1	105
4,3	53

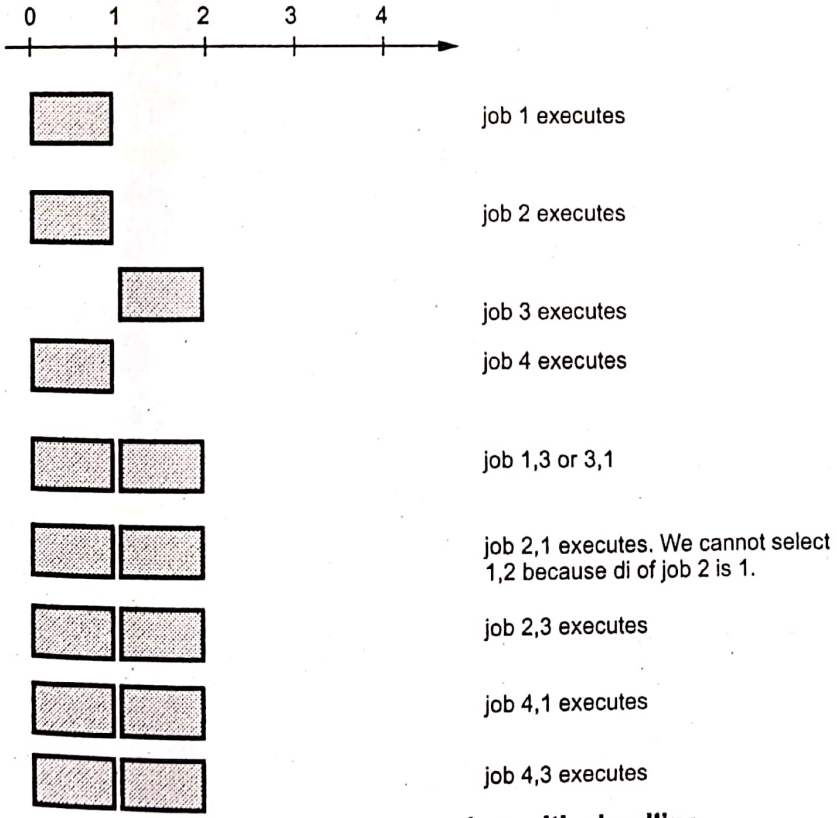


Fig. 5.10.1 Job sequencing with deadline

Each job takes only one unit of time. Deadline of job means a time on which or **before which the job has to be executed.** The sequence {2,4} is not allowed because both have deadline 1. If job 2 is started at 0 it will be completed on 1 but we cannot start job 4 on 1 since deadline of job 4 is 1. The feasible sequence is a sequence that allows all jobs in a sequence to be executed within their deadlines and highest profit can be gained.

- The optimal solution is a feasible solution with maximum profit.
- In above example sequence 3,2 is not considered as $d_3 > d_2$ but we have considered the sequence 2,3 as feasible solution because $d_2 < d_3$.
- We have chosen job 1 first then we have chosen job 4. The solution 4,1 is feasible as the order of execution is 4 then 1. Also $d_4 < d_1$. If we try {1,3,4} then it is not a feasible solution, hence reject 3 from the set. Similarly if we add job 2 in the sequence then the sequence becomes {1,2,4}. This is also not a feasible solution hence reject it. Finally the feasible sequence is 4,1. This sequence is optimum solution as well.

Example 5.10.1 *Using greedy algorithm find an optimal schedule for following jobs with n = 7 profits: (P1,P2,P3,P4,P5,P6,P7) = (3,5,18,20,6,1,38) and deadline (d1,d2,d3,d4,d5,d6,d7) = (1,3,3,4,1,2,1)* **GTU : Winter-11, Marks 7**

Solution :

Step 1 :

We will arrange the profits p_i in ascending order. Then corresponding deadlines will appear.

Profit	38	20	18	6	5	3	1
Job	p_7	p_4	p_3	p_5	p_2	p_1	p_6
Deadline	1	4	3	1	3	1	2

Step 2 :

Create an array J [ ] which stores the jobs. Initially J [ ] will be

1	2	3	4	5	6	7
0	0	0	0	0	0	0

J [7]

Step 3 :

Add i^{th} job in array J [ ] at the index denoted by its deadline. Di.

First job is p_7. The deadline for this job is 1. Hence insert p_7 in the array j[ ] at 1st index.

1	2	3	4	5	6	7
p_7						

Step 4 :

Next job is p_4. Insert it in array J [ ] at index 4.

1	2	3	4	5	6	7
p_7			p_4			

Step 5 :

Next job is p_3. Its has a deadline 3. Therefore insert it at index 3

1	2	3	4	5	6	7
p_7		p_3	p_4			

Step 6 : Next job is p_5, it has deadline 1. But as 1 is already occupied and there is no empty slot at index < J [1] . Hence just discard job p_5. Similarly job p_5 will get discarded.

Step 7 : Next job is p_2. Its has a deadline 3. There is an empty slot at index < J [3] Therefore insert it at index 2

1	2	3	4	5	6	7
p_7	p_2	p_3	p_4			

Step 8 : Thus we now obtain the job sequence as **7-2-3-4** with the profit of **81.**

Example 5.10.2 *Using greedy algorithm find an optimal schedule for following jobs with n = 5 profits : (P1, P2, P3,P4,P5) = (3,5,18,20,38) and deadline : (d1,d2,d3,d4,d5) = (1,3,3,4,1)* **GTU : Winter-14, Marks 7**

Solution : Step 1 : We will arrange profits Pi in descending order. The corresponding deadlines are also mentioned.

Profit	38	20	18	5	3
Job	P5	P4	P3	P2	P1
Deadline	1	3	3	4	1

Step 2 : Create an array J[ ] which stores the job. Initially J [ ] will be.

1	2	3	4	5
0	0	0	0	0

J[5]

Step 3 : Add i^{th} job in array J [ ] at index denoted by its deadline Di. First job is P5 in array J[ ] at 1^{st} index, because its deadline is 1.

1	2	3	4	5
P5				

Step 4 : Next job is P_4. Insert it at index 3.

1	2	3	4	5
P5		P4		

Step 5 : Next job is P3, but its deadline is 3 and it is not possible to insert P3 at 3. Hence just discard it.

Step 6 : Insert P_2 at 4^{th} index as deadline of P_2 is 4.

1	2	3	4	5
P5		P4	P2	

Step 7 : Just discard P1 whose deadline is 1.

Step 8 : Thus we now obtain the job sequence as **5-4-2** with profit of **63.**

Example 5.10.3 *Using greedy algorithm find an optimal schedule for following jobs with n = 6.*

Profits : (Pl, P2, P3, P4, P5, P6) = (20, 15, 10, 7, 5, 3)

Deadline : (d1, d2, d3, d4, d5, d6) = (3 , 1, 1, 3, 1, 3) **GTU : Winter-15, Marks 7**

Solution :

Step 1 : We will arrange the profits P_i in descending order. Then correspondin deadlines will appear.

Profit	20	15	10	7	5	3
Job	P1	P2	P3	P4	P5	P6
Deadline	3	1	1	3	1	3

Step 2 : Create an array J [ ] which stores the jobs. Initially J [ ] will be

1	2	3	4	5	6
0	0	0	0	0	0

J[6]

tep 3 : Add i^{th} job in array J [ ] at index denoted by its deadline d_i. First job is P1. The deadline for this job is 3. Hence insert P1 at index 3 in J [ ] array.

1	2	3	4	5	6
		P1			

tep 4 : Next job is P2. Insert it at index 1 in J [ ] array.

1	2	3	4	5	6
P2		P1			

tep 5 : Next job is P3 having deadline 1. There is no empty slot at index 1 in J [ ] array. Hence discard P3.

tep 6 : Next job is P4 having deadline 3. The index 3 in J [ ] array is already occupied but there is empty slot at index 2 in J [ ] array. Hence insert P4 at index 2.

1	2	3	4	5	6
P2	P4	P1			

tep 7 : Next job is P5 whose deadline is 1. Discard it. Similarly discard the next job i.e. P6 as its deadline is 3.

tep 8 : Thus we now obtain the job sequence as **2 – 4 – 1** with profit 15 + 7 + 20 = **42**.

5.10.1 Algorithm

The algorithm for job sequencing is as given below.

```
lgorithm Job_seq(D,J,n)

Problem description : This algorithm is for job sequencing using Greedy method.
D[i] denotes ith deadline where 1 ≤ i ≤ n
J[i] denotes ith job
D[J[i]] ≤ D[J[i+1]]
nitially
0]←0;
0]←0;
1]←1;
ount←1;
t ←2 to n do

←count;
hile(D[J[t]] > D[i]) AND D[J[t]]!=t)) do t←t-1;
(D[J[t]] ≤ D[i])AND(D[i] > t)) then
```

```
{
      //insertion of ith feasible sequence into J array
      for s←count to (t+1) step -1 do
      J[s+1] ← J[s];
      J[t+1]←I;
      count←count+1;
 }//end of if
}//end of while
return count;
}
```

The sequence of J will be inserted if and only if D[J[t]]!=t. This also means that the job J will be processed if it is in within the deadline.

The computing time taken by above Job_seq algorithm is $O(n^2)$, because the basic operation of computing sequence in array J is within two nested for loops.