LP II- Artificial Intelligence

Practical No- 3

3. Implement Greedy search algorithm for any of the following application:
   - I. Selection Sort
   - II. Minimum Spanning Tree
   - III. Single-Source Shortest Path Problem
   - IV. Job Scheduling Problem
   - V. Prim's Minimal Spanning Tree Algorithm
   - VI. Kruskal's Minimal Spanning Tree Algorithm
   - VII. Dijkstra's Minimal Spanning Tree Algorithm

Problem: Solve the following job scheduling with deadlines problem using the greedy method. Number of jobs N = 4. Profits associated with Jobs : (P1, P2, P3, P4) = (100, 10, 15, 27). Deadlines associated with jobs (d1, d2, d3, d4) = (2, 1, 2, 1)

Solution:

What is a greedy Strategy?

A Greedy Strategy is an algorithmic approach where we make the best possible choice at each step without worrying about the overall consequences. We hope that these local best choices will lead to a globally optimal solution.

Algorithm :

1. Input the data and constraints.
2. Define a greedy selection criterion.
3. Sort or organize the data using this criterion.
4. Initialize result/solution.
5. Loop through the sorted data:
   - If the current item is valid, add it to the result.
6. Return the result.

In job sequencing problems, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

Examples

Problem 1: Solve the following job scheduling with deadlines problem using the greedy method. Number of jobs N = 4. Profits associated with Jobs : (P1, P2, P3, P4) = (100, 10, 15, 27). Deadlines associated with jobs (d1, d2, d3, d4) = (2, 1, 2, 1)

Solution:

Sort all jobs in descending order of profit.

So,

P = (100, 27, 15, 10), J = (J1, J4, J3, J2) and D = (2, 1, 2, 1).

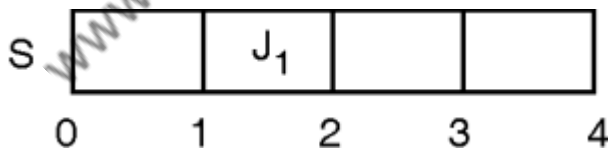
We shall select one by one job from the list of sorted jobs, and check if it satisfies the deadline. If so, schedule the job in the latest free slot. If no such slot is found, skip the current job and process the next one. Initially,

Profit of scheduled jobs, SP = 0

Iteration 1:

Deadline for job J1 is 2. Slot 2 (t = 1 to t = 2) is free, so schedule it in slot 2.

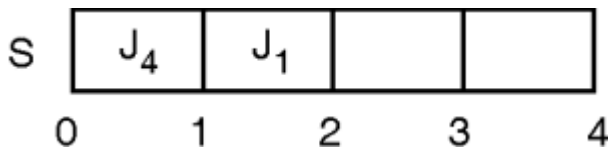
Solution set S= {J1}, and Profit SP = {100}



Iteration 2:

Deadline for job J4 is 1. Slot 1 (t = 0 to t = 1) is free, so schedule it in slot 1.

Solution set S = {J1,J4}, and Profit SP = {100, 27}



Iteration 3:

Job J3 is not feasible because the first two slots are already occupied and if we schedule J3 any time later t = 2, it cannot be finished before its deadline 2. So job J3 is discarded,

Solution set S = {J1,J4}, and Profit SP = {100, 27}

Iteration 4:

Job J2 is not feasible because the first two slots are already occupied and if we schedule J2 any time later t = 2, it cannot be finished before its deadline 1. So job J2 is discarded,

Solution set S = {J1,J4}, and Profit SP = {100, 27}

With the greedy approach, we will be able to schedule two jobs {J1, J4}, which gives a profit of 100 + 27 **= 127 units.**

Problem 2:

Solve the following instance of "job scheduling with deadlines"

problem : n = 7, profits (p1, p2, p3, p4, p5, p6, p7) = (3, 5, 20, 18, 1, 6, 30) and deadlines

(d1, d2, d3, d4, d5, d6, d7) = (1, 3,4, 3, 2, 1,2). Schedule the jobs in such a way to get

maximum profit.

Solution:

Given that,

Jobs	**j1**	**j2**	**j3**	**j4**	**j5**	**j6**	**j7**
Profit	3	5	20	18	1	6	30
Deadline	1	3	4	3	2	1	2

Sort all jobs in descending order of profit.

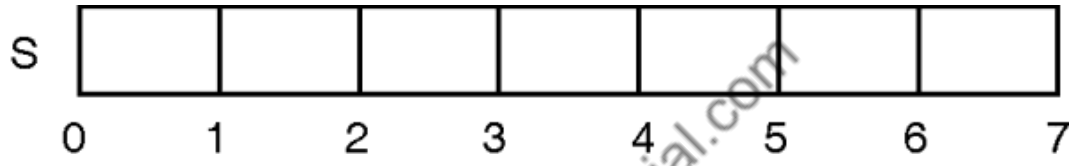
So, P = (30, 20, 18, 6, 5, 3, 1),

J = (J7, J3, J4, J6, J2, J1, J5) and

D = (2, 4, 3, 1, 3, 1, 2).

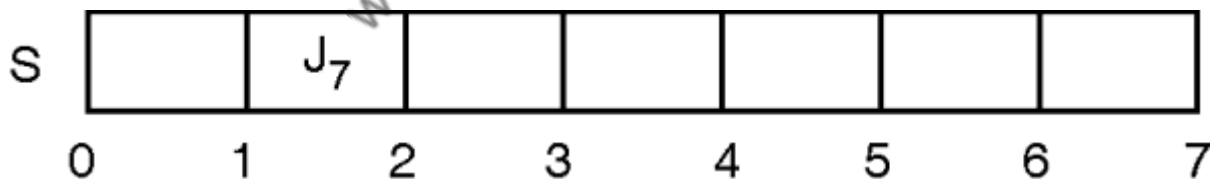
We shall select one by one job from the list of sorted jobs J, and check if it satisfies the deadline. If so, schedule the job in the latest free slot. If no such slot is found, skip the current job and process the next one. Initially,

Profit of scheduled jobs, SP = 0



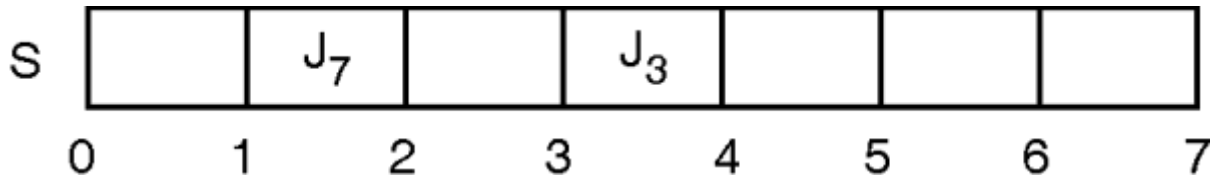
Iteration 1:

Deadline for job J7 is 2. Slot 2 (t = 1 to t = 2) is free, so schedule it in slot 2. Solution set S = {J7}, and Profit SP = {30}



Iteration 2:

Deadline for job J3 is 4. Slot 4 (t = 3 to t = 4) is free, so schedule it in slot 4. Solution set S = {J7, J3}, and Profit SP = {30, 20}



Iteration 3:

Deadline for job J4 is 3. Slot 3 (t = 2 to t = 3) is free, so schedule it in slot 3.

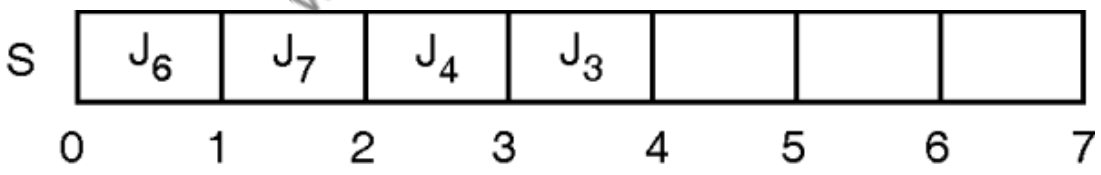
Solution set S = {J7, J3, J4}, and Profit SP = {30, 20, 18}

Iteration 4:

Deadline for job J6 is 1. Slot 1 (t = 0 to t = 1) is free, so schedule it in slot 1.

Solution set S = {J7,J3,J4,J6}, and Profit

SP = {30, 20, 18, 6}



First, all four slots are occupied and none of the remaining jobs has a deadline of less than 4. So none of the remaining jobs can be scheduled. Thus, with the greedy approach, we will be able to schedule four jobs {J7,J3,J4,J6}, which give a profit of (30 + 20 + 18 + 6) = **74 units.**

```
public class Job {
      String id;
         int deadline;
         int profit;
         Job(String id, int deadline, int profit) {
             this.id = id;
             this.deadline = deadline;
             this.profit = profit;
         }
}
public class jobsequencing {
     static int minValue(int x, int y) {
        return (x < y) ? x : y;
    }
   static void jobSequencingWithDeadline(Job[] jobs,
int n) {
        // Find the maximum deadline
        int dmax = 0;
        for (int i = 0; i < n; i++) {
            if (jobs[i].deadline > dmax) {
                dmax = jobs[i].deadline;
            }
        }
        // Create a time slot array to keep track of
free slots
        int[] timeslot = new int[dmax + 1];
        Arrays.fill(timeslot, -1);
        int filledTimeSlot = 0;
        int maxProfit = 0;
        for (int i = 0; i < n; i++) {
            int k = minValue(dmax, jobs[i].deadline);
            while (k >= 1) {
                if (timeslot[k] == -1) { // If slot is
free
```

```
                    timeslot[k] = i;
                    filledTimeSlot++;
                    break;
                }
                k--;
            }
            // Stop if all slots are filled
            if (filledTimeSlot == dmax) {
                break;
            }
        }
        // Display the selected jobs
        System.out.print("\nRequired Jobs: ");
        for (int i = 1; i <= dmax; i++) {
            if (timeslot[i] != -1) {
                System.out.print(jobs[timeslot[i]].id);
                if (i < dmax && timeslot[i + 1] != -1)
{
                    System.out.print(" ---> ");
                }
            }
        }
        // Calculate total profit
        for (int i = 1; i <= dmax; i++) {
            if (timeslot[i] != -1) {
                maxProfit += jobs[timeslot[i]].profit;
            }
        }
        System.out.println("\nMax Profit: " +
maxProfit);
    }
    public static void main(String[] args) {
        Job[] jobs = {
            new Job("j1", 2, 100),
            new Job("j2", 1, 10),
```

```
            new Job("j3", 2, 15),
            new Job("j4", 1, 27),
        };
        int n = jobs.length;
        // Manual sorting by profit in descending order
(bubble sort)
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (jobs[j].profit < jobs[j +
1].profit) {
                    // swap jobs[j] and jobs[j + 1]
                    Job temp = jobs[j];
                    jobs[j] = jobs[j + 1];
                    jobs[j + 1] = temp;
                }
            }
        }
        System.out.println("Jobs\tDeadline\tProfit");
        for (int i = 0; i < n; i++) {
            System.out.println(jobs[i].id + "\t" +
jobs[i].deadline + "\t\t" + jobs[i].profit);
        }
        jobSequencingWithDeadline(jobs, n);
    }
}
```


Output:

```
Jobs Deadline Profit
j1   2        100
j4   1        27
j3   2        15
j2   1        10
Required Jobs: j4 ---> j1
Max Profit: 127
```