Job Scheduling problem using Greedy Algorithm

A MINI PROJECT REPORT

18CSC204J -Design and Analysis of Algorithms Laboratory

Submitted by

Shantanu Sahay [RA2011030010078]

Dhawal Gupta [RA2011030010081]

Under the guidance of

Dr.M.Shobana

Assistant Professor, Department of Networking and Communication

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in Cloud Computing

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report titled "Job Scheduling" is the bonafide work done by Shantanu Sahay (RA2011028010078) and Dhawal Gupta (RA2011028010081) who carried out the mini project work and Laboratory exercises under my supervision for **18CSC204J -Design and Analysis of Algorithms Laboratory**. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr.M.Shobana

ASSISTANT PROFESSOR
18CSC204J -Design and Analysis of Algorithms
Course Faculty
Department of Networking and Communications

Signature of the Internal Examiner-I **Signature of the Internal Examiner-II**

ABSTRACT

Job scheduling is the problem of scheduling jobs out of a set of N jobs on a single processor which maximizes profit as much as possible. Consider N jobs, each taking unit time for execution. Each job is having some profit and deadline associated with it. Profit earned only if the job is completed on or before its deadline. Otherwise, we have to pay a profit as a penalty. Each job has deadline di ≥ 1 and profit pi ≥ 0. At a time, only one job can be active on the processor.

The job is feasible only if it can be finished on or before its deadline. A feasible solution is a subset of N jobs such that each job can be completed on or before its deadline. An optimal solution is a solution with maximum profit. The simple and inefficient solution is to generate all subsets of the given set of jobs and find the feasible set that maximizes the profit. For N jobs, there exist 2N schedules, so this brute force approach runs in O(2N) time.

However, the greedy approach produces an optimal result in fairly less time. As each job takes the same amount of time, we can think of the schedule S consisting of a sequence of job slots 1, 2, 3, …, N, where S(t) indicates job scheduled in slot t. Slot t has a span of (t – 1) to t. S(t) = 0 implies no job is scheduled in slot t.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATION

ABBREVIATION	EXPANSION
NLU	Natural Language understanding
J1	JOB 1
J2	JOB 2
J3	JOB 3
J4	JOB 4
J5	JOB 5
J6	JOB 6

LIST OF TABLES

CHAPTER-1

PROBLEM DEFINITION

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?”

CHAPTER-2

PROBLEM EXPLANATION

- 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.

CHAPTER-3

DESIGN TECHNIQUES

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.

CHAPTER-4

ALGORITHM FOR THE PROBLEM

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

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

Table 4.1

Algorithm:

Job-Sequencing-With-Deadline (D, J, n, k)

Step 1: START

Step 2: D(0) := J(0) := 0

Step 3: k := 1

Step 4: J(1) := 1 // means first job is selected

Step 5: for i = 2 … n do

Step 6: r := k

Step 7: while D(J(r)) > D(i) and D(J(r)) ≠ r do

Step 8: r := r – 1

Step 9: if D(J(r)) ≤ D(i) and D(i) > r then

Step 10: for l = k … r + 1 by -1 do

Step 11: J(l + 1) := J(l)

Step 12: J(r + 1) := i

Step 13: k := k + 1

Step 14: END

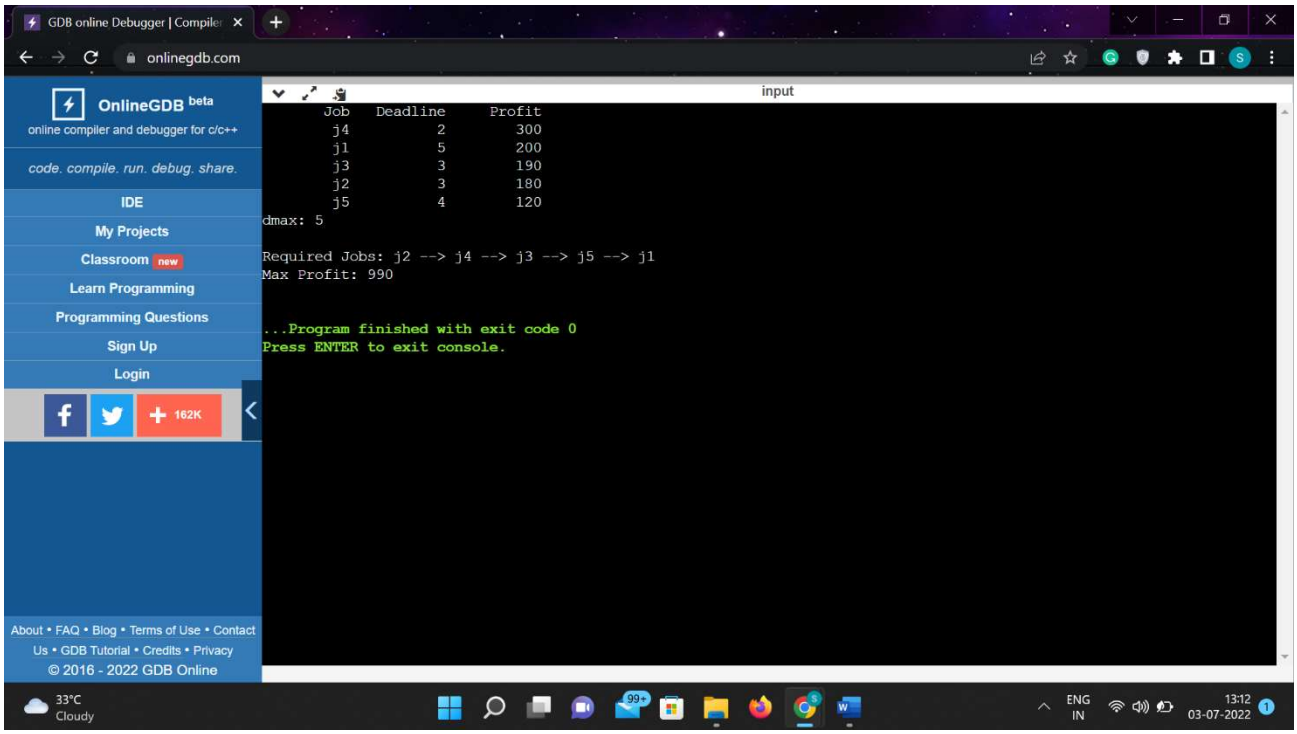


Fig 4.1

CHAPTER-5

EXPLANATION OF ALGORITHM

Step-01:

Value of maximum deadline = 5.

So, draw a Gantt chart with maximum time on Gantt chart = 5 units as shown

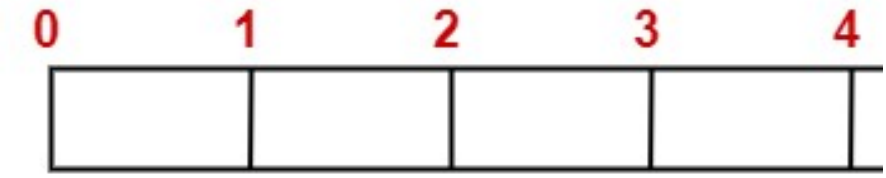


Table 5.1

Now,

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.

Step-02:

We take job J4.

Since its deadline is 2, so we place it in the first empty cell before deadline 2 as-

Table 5.2

Step-03:

We take job J1.

Since its deadline is 5, so we place it in the first empty cell before deadline 5 as-

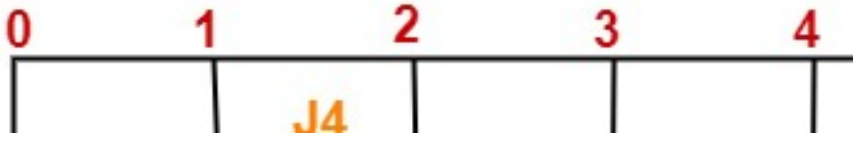


Table 5.3

Step-04:

We take job J3.

Since its deadline is 3, so we place it in the first empty cell before deadline 3 as-

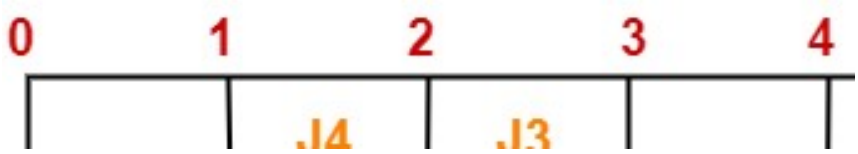


Table 5.4

Step-5:

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-

Table 5.5

Step-06:

Now, we take job J5.

Since its deadline is 4, so we place it in the first empty cell before deadline 4 as

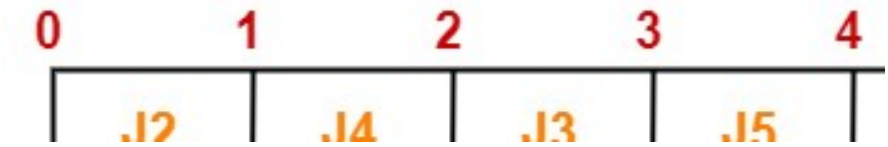


Table 5.6

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.

Now, the given questions may be answered as-

Question 1:

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.

Question 2:

All the jobs are not completed in optimal schedule.

This is because job J6 could not be completed within its deadline.

Question 3:

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

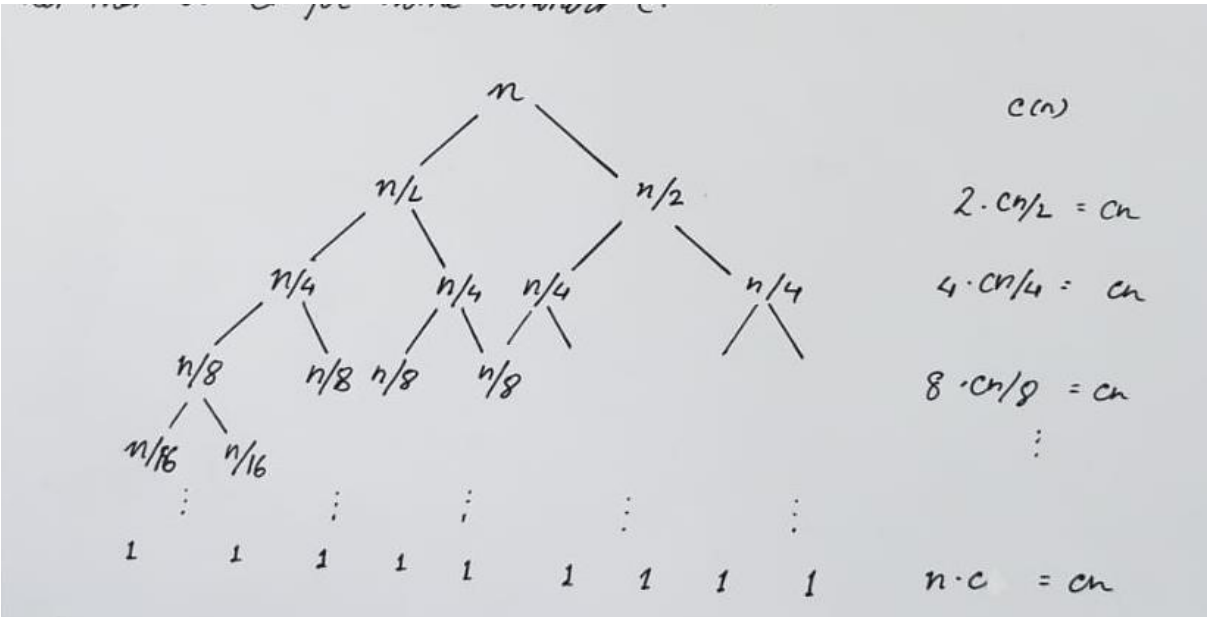
= 990 units

CHAPTER-6

COMPLEXITY ANALYSIS

The time complexity of the given code is the summation of the sorting algorithm's time complexity and the loop's time complexity to maximize the profit.

Considering we use merge sort as the sorting algorithm,



1) Divide sort computes the midpoint of each Submerged. Each of these takes O(1) time .
2) The conquer step recursively sorts 2 subarrays of n/2 elements each .
3) The merge sort merges n elements which take O(n) time. Considering steps 1 to 3, the time complexity is O(n) as the total. Let that be cn for some constant c. From each level, top to bottom, 2 calls merge onto 2 subarrays of length n/2 each. complexity is (2*cn)/2 = cn. now the height of the tree is (log n + 1). Thus, overall complexity is O(nlogn)

To select the items, we need one scan to this sorted list which will take O(n) time.

Thus, the total time complexity T(n) = O(nlogn) + O(n) = O(nlogn)

CHAPTER-7

CONCLUSION

Design and analysis of algorithms have helped humanity solve real problems using techniques available in Greedy algorithms, dynamic programming, divide and conquer, backtracking, and the searching and sorting algorithm. This has efficiently resulted in time management and maximizing profit value.

REFERENCES

- https://www.gatevidyalay.com/job-sequencing-with-deadlines/
- https://codecrucks.com/job-scheduling-using-greedy-algorithm/#:~:text=Job%20scheduling%20is%20the%20problem,and%20deadline%20associated%20with%20it.
- https://www.geeksforgeeks.org/job-sequencing-problem/
- https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_job_sequencing_with_deadline.htm
- https://www.techiedelight.com/job-sequencing-problem-deadlines/
- https://www.codingninjas.com/codestudio/library/job-scheduling-algorithm
- https://dyclassroom.com/greedy-algorithm/job-sequencing-problem-with-deadline
- https://www.interviewbit.com/blog/job-sequencing-with-deadlines/
- https://www2.cs.sfu.ca/~kabanets/307/sched307.pdf
- https://www.javatpoint.com/activity-or-task-scheduling-problem

APPENDIX

1. CODE

```
#include <stdio.h>
#define MAX 100
typedef struct Job {
  char id[5];
  int deadline;
  int profit;
} Job;
void jobSequencingWithDeadline(Job jobs[], int n);
int minValue(int x, int y) {
  if(x < y) return x;
  return y;
}
int main(void) {
  //variables
  int i, j;
  //jobs with deadline and profit
  Job jobs[100] = {
    {"j1", 5, 200},
    {"j2", 3, 180},
    {"j3", 3, 190},
    {"j4", 2, 300},
    {"j5", 4, 120},
    {"j6", 2, 100},
  };
  //temp
  Job temp;
  //number of jobs
  int n = 5;
  //sort the jobs profit wise in descending order
  for(i = 1; i < n; i++) {
```

```
    for(j = 0; j < n - i; j++) {
      if(jobs[j+1].profit > jobs[j].profit) {
        temp = jobs[j+1];
        jobs[j+1] = jobs[j];
        jobs[j] = temp;
      }
    }
  }
  printf("%10s %10s %10s\n", "Job", "Deadline", "Profit");
  for(i = 0; i < n; i++) {
    printf("%10s %10i %10i\n", jobs[i].id, jobs[i].deadline,
jobs[i].profit);
  }
  jobSequencingWithDeadline(jobs, n);
  return 0;
}
void jobSequencingWithDeadline(Job jobs[], int n) {
  //variables
  int i, j, k, maxprofit;
  //free time slots
  int timeslot[MAX];
  //filled time slots
  int filledTimeSlot = 0;
  //find max deadline value
  int dmax = 0;
  for(i = 0; i < n; i++) {
    if(jobs[i].deadline > dmax) {
      dmax = jobs[i].deadline;
    }
  }
  //free time slots initially set to -1 [-1 denotes EMPTY]
  for(i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
```

```
  }
  printf("dmax: %d\n", dmax);
  for(i = 1; i <= n; i++) {
    k = minValue(dmax, jobs[i - 1].deadline);
    while(k >= 1) {
      if(timeslot[k] == -1) {
        timeslot[k] = i-1;
        filledTimeSlot++;
        break;
      }
      k--;
    }
    //if all time slots are filled then stop
    if(filledTimeSlot == dmax) {
      break;
    }
  }
  //required jobs
  printf("\nRequired Jobs: ");
  for(i = 1; i <= dmax; i++) {
    printf("%s", jobs[timeslot[i]].id);
    if(i < dmax) {
      printf(" --> ");
    }
  }
  //required profit
  maxprofit = 0;
  for(i = 1; i <= dmax; i++) {
    maxprofit += jobs[timeslot[i]].profit;
  }
  printf("\nMax Profit: %d\n", maxprofit);
}
```