Job Sequencing with Deadline: A Greedy Approach

Varun Kumar

July 8, 2025

Contents

Listings

List of Algorithms

1 Introduction

The **Job Sequencing with Deadline** problem involves scheduling jobs to maximize total profit when each job has:

- a **profit** value
- a **deadline**
- takes **one unit of time**

Goal: Maximize total profit by completing jobs within their deadline, assuming only one job can be scheduled at a time.

2 Problem Statement

Given n jobs, each with:

- Job ID
- Deadline (integer)
- Profit

Schedule the jobs to maximize profit such that no two jobs overlap and each job is done before or on its deadline.

3 Approach: Greedy Algorithm

- Sort all jobs in descending order of profit.
- Create a result array to store job sequence and track free slots.
- For each job, find the latest available slot before its deadline.
- If a slot is found, schedule it.

3.1 Problem Statement

Given 5 jobs with deadlines and profits:

Job ID	Deadline	Profit
J1	2	60
J2	1	100
J3	3	20
J4	2	40
J5	1	20

Goal: Schedule jobs to maximize total profit. Each job takes 1 unit of time and must be finished before or on its deadline.

3.2 Step 1: Sort Jobs by Profit (Descending)

Job ID	Deadline	Profit
J2	1	100
J1	2	60
J4	2	40
J3	3	20
J5	1	20

3.3 Step 2: Schedule Jobs in Greedy Manner

We'll use a time slot array. The maximum deadline is 3, so we have 3 time slots: Slot 1, Slot 2, Slot 3.

Job 1: J2: Deadline 1 → Slot 1 is free → Schedule J2 at Slot 1

Job 2: J1: Deadline 2 → Slot 2 is free → Schedule J1 at Slot 2

Job 3: J4: Deadline 2 → Slot 2 is full → Check Slot 1 (already taken) → Can't schedule

Job 4: J3: Deadline 3 → Slot 3 is free → Schedule J3 at Slot 3

Job 5: J5: Deadline 1 → Slot 1 is full → Can't schedule

3.4 Step 3: Final Scheduled Jobs

Time Slot	Job Scheduled
Slot 1	J2
Slot 2	J1
Slot 3	J3

Total Profit = 100 (J2) + 60 (J1) + 20 (J3) = **180**

3.5 Step 4: Timeline Visualization

Slot 1	Slot 2	Slot 3
J2 (100)	J1 (60)	J3 (20)

3.6 Conclusion

Using the greedy strategy:

- We prioritized jobs with the highest profit.
- Placed each job in the latest available slot before its deadline.
- Achieved maximum profit of **180**.

This is an efficient solution with time complexity:

$$O(n \log n + n \cdot d)$$

where d is the maximum deadline.

4 Theoretical Logic Behind Job Sequencing with Deadline Code Implementation

The Job Sequencing with Deadline problem is solved using a **greedy algorithm** aimed at maximizing total profit. Each job has:

- A **deadline** (latest time by which it must be scheduled)
- A **profit** (earned if scheduled before or on the deadline)

Approach Overview

1. Sort all jobs in **descending order of profit**.
2. Initialize a time slot array of size equal to the **maximum deadline**.
3. Iterate over each job in sorted order:
    - Try to place the job in the **latest available slot** on or before its deadline.
    - If such a slot exists, assign the job and accumulate the profit.
4. Continue until all jobs are considered.

Why Greedy Works

The problem satisfies:

- **Greedy choice property:** Locally best choice (most profitable job first) leads to global optimum.
- **Optimal substructure:** Scheduling earlier jobs doesn't prevent optimal scheduling of remaining jobs.

Time and Space Complexity

- **Sorting:** $O(n \log n)$
- **Scheduling:** $O(n \cdot d)$ where d is the max deadline (or $O(n)$ with Disjoint Set Union)
- **Space:** $O(d)$ for the slot array

Slot Allocation Strategy

For each job with deadline d_i, we look for a free slot from d_i to 1. The goal is to place the job as **late as possible** before its deadline to leave earlier slots open for tighter-deadline jobs.

Example: If Job A has deadline 3, we try: Slot 3 $\rightarrow$ Slot 2 $\rightarrow$ Slot 1. This backward check ensures maximum slot availability for other jobs.

Final Result

The algorithm outputs:

- A list of scheduled jobs
- Maximum total profit earned

This solution is efficient and optimal for single-unit jobs under hard deadlines.

4.1 Pseudocode

Algorithm 1 Job Sequencing with Deadline

```
procedure JOBSEQUENCING(jobs)
    Sort jobs by descending profit
    result ← array of size max_deadline
    slot ← array of size max_deadline initialized as False
    for each job in jobs do
        for j = min(job.deadline, n) − 1 to 0 do
            if slot[j] == False then
                result[j] ← job
                slot[j] ← True
                break
            end if
        end for
    end for
    return scheduled jobs from result
end procedure
```

4.2 Python Implementation

```
class Job:
    def __init__(self, id, deadline, profit):
        self.id = id
        self.deadline = deadline
        self.profit = profit

def job_sequencing(jobs):
    jobs.sort(key=lambda x: x.profit, reverse=True)
    max_deadline = max(job.deadline for job in jobs)
    result = [None] * max_deadline
    slot = [False] * max_deadline

    for job in jobs:
        for j in range(min(max_deadline, job.deadline)-1, -1,
           -1):
            if not slot[j]:
                result[j] = job.id
                slot[j] = True
                break

    return [job_id for job_id in result if job_id]
```

Listing 1: Job Sequencing with Deadline in Python

4.3 C++ Implementation

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
struct Job {
    char id;
    int deadline, profit;
};
bool cmp(Job a, Job b) {
    return a.profit > b.profit;
}
vector<char> jobSequencing(vector<Job> &jobs) {
    sort(jobs.begin(), jobs.end(), cmp);
    int max_deadline = 0;
    for (Job &job : jobs) max_deadline = max(max_deadline, job.
       deadline);
    vector<char> result(max_deadline, '\0');
    vector<bool> slot(max_deadline, false);
    for (Job &job : jobs) {
        for (int j = min(job.deadline, max_deadline) - 1; j >=
           0; --j) {
            if (!slot[j]) {
                result[j] = job.id;
                slot[j] = true;
                break;
            }
        }
    }
    vector<char> scheduled;
    for (char c : result) {
        if (c != '\0') scheduled.push_back(c);
    }
    return scheduled;
}
```

Listing 2: Job Sequencing with Deadline in C++

5 Key Points to Remember

1. Greedy approach: always pick highest profit job first.
2. Sorting based on profit is crucial.
3. Use a time-slot array to track free positions.
4. Try to place each job in the latest available slot.
5. Each job takes 1 unit of time.

6 Time and Space Complexity

- **Time:** $O(n \log n + n \cdot d)$ where d is max deadline
- **Space:** $O(d)$ for time slot tracking

7 Real-World Applications

- **CPU Job Scheduling:** Maximize efficiency and throughput
- **Project Deadlines:** Pick best paying contracts under time constraint
- **Cloud Computing:** Efficient resource utilization for tasks
- **Freelancing:** Prioritize clients with highest returns before due dates

8 Conclusion

Job Sequencing with Deadline is a fundamental greedy algorithm that models real-world scheduling and optimization problems. It ensures profit maximization while adhering to time constraints — a key strategy in resource management and systems design.