

Session 3

Task Scheduling Algorithms

A task scheduling problem is formulated as an optimization problem in which we need to determine the set of tasks from the given tasks that can be accomplished within their deadlines along with their order of scheduling such that the profit is maximum. So, this is a maximization optimization problem with a constraint that tasks must be completed within their deadlines.

Objective

✖ Do not copy. Read for understanding and the viva

The main objectives of this session are to:

- Implement a task scheduling algorithm
- Test the algorithm on different problem instances
- Differentiate between various task scheduling approaches based on their performance and efficiency
 - Brute Force Approach
 - Efficient task scheduling algorithm

Question 1

Problem Statement

■ Write in lab record

Apply a brute force approach to schedule three jobs J1, J2 and J3 with service times as 5, 8, 12 respectively. The actual service time units are not relevant to the problems. Make all possible job schedules, calculate the total times spent in jobs by the system. Find the optimal schedule (total time). If there are N jobs, what would be the total number of job schedules?

Mathematical Logic

✕ Do not copy. Read for understanding and the viva

For a sequence of jobs (J_a, J_b, J_c) with service times (t_a, t_b, t_c) :

- J_a finishes at t_a
- J_b finishes at $t_a + t_b$
- J_c finishes at $t_a + t_b + t_c$

The total time spent in jobs by the system is the sum of the finishing times of all jobs, which can be calculated as: $\text{Total Time} = (t_a) + (t_a + t_b) + (t_a + t_b + t_c) = 3*t_a + 2*t_b + t_c$

Implementation



Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

Python

task-scheduling.py

```
1 from itertools import permutations
2
3 def solve_brute_force(jobs):
4     # jobs is a list of tuples: (name, time)
5     all_configs = list(permutations(jobs))
6     best_time = float("inf")
7     best_seq = None
8
9     for config in all_configs:
10         current_finish_time = 0
11         total_system_time = 0
12
13         # Calculate completion time for each job in this specific order
14         for name, time in config:
15             current_finish_time += time
16             total_system_time += current_finish_time
17
18         print(
19             f"Schedule {'-'.join([j[0] for j in config])}: Total Time = {tot
20         )
21
22         # Track the minimum time found
23         if total_system_time < best_time:
24             best_time = total_system_time
25             best_seq = config
26
27     print(f"\nOptimal: {'-'.join([j[0] for j in best_seq])} with Time: {best
28
29
30 job_list = [("J1", 5), ("J2", 8), ("J3", 12)]
31 solve_brute_force(job_list)
```

Question 2

Problem Statement

Implement the task scheduling algorithm on your system to minimize the total amount of time spent in the system for the following problem:

Job	Service Time
1	5
2	10
3	7
4	8

Mathematical Logic

To minimize the total amount of time spent in the system, we can use the Shortest Job First (SJF) scheduling algorithm. This algorithm schedules jobs in order of their service times, starting with the shortest job first. The total time spent in the system can be calculated as follows:

- Schedule the jobs in ascending order of their service times: J1 (5), J3 (7), J4 (8), J2 (10)
- Calculate the finishing times for each job:
 - J1 finishes at 5
 - J3 finishes at $5 + 7 = 12$
 - J4 finishes at $12 + 8 = 20$
 - J2 finishes at $20 + 10 = 30$
- The total time spent in the system is the sum of the finishing times: $5 + 12 + 20 + 30 = 67$

Implementation



Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

Python

task-scheduling.py

```
1 def minimize_total_time(jobs):
2     # Sort jobs by service time (index 1 of the tuple)
3     jobs.sort(key=lambda x: x[1])
4
5     current_time = 0
6     total_spent = 0
7     sequence = []
8
9     for job_id, service_time in jobs:
10         current_time += service_time # Time when this specific job finishes
11         total_spent += current_time # Add this finish time to the system to
12         sequence.append(str(job_id))
13
14     print(f"Optimal Sequence: {' → '.join(sequence)}")
15     print(f"Total Time Spent in System: {total_spent}")
16
17
18 # Input: (Job ID, Service Time)
19 job_list = [(1, 5), (2, 10), (3, 7), (4, 8)]
20 minimize_total_time(job_list)
```

Question 3

Problem Statement

Consider the following jobs, deadlines and profits. Implement the task scheduling algorithm with deadlines to maximize the total profits.

Job	Deadline	Profits
1	3	50
2	4	20
3	5	70
4	3	15
5	2	10
6	1	47
7	1	60

Mathematical Logic

To maximize the total profits while scheduling jobs with deadlines, we can use a greedy approach. We will sort the jobs based on their profits in descending order and then schedule them as late as possible before their deadlines.

1. Sort the jobs based on profits: J3 (70), J7 (60), J1 (50), J6 (47), J2 (20), J4 (15), J5 (10)

2. Schedule the jobs:

- J3 is scheduled at time 5 (profit 70)
- J7 is scheduled at time 4 (profit 60)
- J1 is scheduled at time 3 (profit 50)
- J6 is scheduled at time 2 (profit 47)
- J2 is scheduled at time 1 (profit 20)
- J4 and J5 cannot be scheduled as their deadlines have passed.

3. The total profit is the sum of the profits of the scheduled jobs: $70 + 60 + 50 + 47 + 20 = 247$

4. The optimal schedule is: J2 (1), J6 (2), J1 (3), J7 (4), J3 (5) with a total profit of 247.

Implementation



Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

Python

task-scheduling.py

```
1 def solve_job_sequencing(jobs):
2     # 1. Sort jobs by profit in descending order
3     # (Job ID, Deadline, Profit)
4     jobs.sort(key=lambda x: x[2], reverse=True)
5
6     # 2. Find max deadline to size the timeline
7     max_d = max(job[1] for job in jobs)
8
9     # 3. Initialize slots (0 is ignored, using 1 to max_d)
10    slots = [-1] * (max_d + 1)
11    total_profit = 0
12    count_jobs = 0
13
14    for job_id, deadline, profit in jobs:
15        # 4. Try to place job in the latest available slot from deadline down
16        for j in range(deadline, 0, -1):
17            if slots[j] == -1:
18                slots[j] = job_id
19                total_profit += profit
20                count_jobs += 1
21                break
22
23    # Filter out empty slots for the result
24    scheduled_jobs = [slots[i] for i in range(1, max_d + 1) if slots[i] != -1]
25    return scheduled_jobs, total_profit
26
27
28 # Data: (ID, Deadline, Profit)
29 job_data = [
30     (1, 3, 50),
31     (2, 4, 20),
32     (3, 5, 70),
33     (4, 3, 15),
34     (5, 2, 10),
35     (6, 1, 47),
36     (7, 1, 60),
37 ]
38 sequence, profit = solve_job_sequencing(job_data)
39
40 print(f"Scheduled Sequence: {sequence}")
41 print(f"Maximum Profit: {profit}")
```

Conclusion

In this session, we have explored various task scheduling algorithms and their implementations. We have seen how to apply a brute force approach to schedule jobs and calculate the total time spent in the system. We also implemented the Shortest Job First (SJF) algorithm to minimize the total time spent in the system and a greedy approach to maximize profits while scheduling jobs with deadlines. These algorithms are fundamental in understanding how to efficiently manage tasks in various applications, such as operating systems, project management, and manufacturing processes.

 [Edit this page](#)

[< Previous](#)
Session 2

Session 4 [Next >](#)