Data Structure lab6

吴嘉骜 21307130203

2023 年 11 月 12 日

Objective

The objective of this lab is to understand the spirit of Dynamic Programming and to apply it to solve problems.

Experiment environment

Windows 11 VsCode Python 3.11.5 64-bit

Task description

Suppose you have one machine and a set of n jobs $a_1, a_2, \ldots, a_n$ to process on that machine. Each job a_j has a processing time t_j, a profit p_j, and a deadline d_j. The machine can process only one job at a time, and job a_j must run uninterruptedly for t_j consecutive time units. If job a_j is completed by its deadline d_j, you receive the profit p_j, but if it is completed after its deadline, you receive the profit of 0.
Give an algorithm to find the schedule that obtains the maximum amount of profit, assuming that all processing times are integers between 1 and n. Please write code and analyze time complexity.

1 Algorithm

We use a dynamic programming (DP) approach to solve the job scheduling problem for maximum profit. First we sort the jobs by their deadlines. This is because it makes sense to consider earlier deadlines first for scheduling, and it will be useful when we build up the DP table next.
We then construct a 2D DP table dp with size $(n+1) \times (d_n+1)$, where n is the number of jobs and d_n is the maximum deadline. The $dp(i,j)$ element represents the maximum profit that can be obtained by scheduling jobs $1, 2, \ldots, i$ adhering to deadlines, within total time j, or j slots. Note that $i = 0$ represents the case where there are no jobs to schedule, and $j = 0$ represents the case where there is no time slot to schedule the jobs. Both cases result in a profit of 0.

For $dp(i,j), i \geq 1$, consider whether we want to schedule job i or not:
If we do not schedule job i, then the maximum profit is the same as scheduling jobs $1, 2, \ldots, i-1$ within j slots, which is $dp(i-1,j)$.
If we schedule job i, assume that the latest time slot that a_i can start is t', then it should satisfy $t' \leq j - t_i$ and $t' \leq d_i - t_i$, which leads to $t' = min\{j, d_i\} - t_i$.
When $t' \geq 0$, we claim that $dp(i,j) = p_i + dp(i-1,t')$. Fisrt notice that a_i is the job with the latest deadline among jobs $1, 2, \ldots, i$, and we always arrange it at the latest possible time slot. If we schedule a_i

ahead of t', say $t'' < t'$, the profit will be $p_i + dp(i-1, t'') \leq p_i + dp(i-1, t')$, since $dp(i-1, t'') \leq dp(i-1, t')$. Second, we prove that $p_i + dp(i-1, t')$ maximizes the profit. As shown above, if $p_i + dp(i-1, t'')$ may achieve a higher profit, then $t'' \geq t'$. As t' is the latest possible time slot for a_i, there must be no jobs in $1, \ldots, i-1$ scheduled after t', which means that we can use $dp(i-1, t')$ to replace the profit achieved before t''. Therefore, $p_i + dp(i-1, t')$ is the maximum profit.
Conversely, When $t' < 0$, it means that to arrange a_i we have to start it before time slot 0, which is impossible. In this case, $dp(i, j) = dp(i-1, j)$.

In conclusion, we have the following recurrence relation:

$$dp(i,j) = \begin{cases} dp(i-1, j) & \text{if } t' < 0 \\ \max\{dp(i-1, j), p_i + dp(i-1, t')\} & \text{if } t' \geq 0 \end{cases}$$

When the dp table is filled, the maximum profit is $dp(n, d_n)$, and we can trace back to find the schedule. We start from $dp(n, d_n)$, and if $dp(n, d_n) \neq dp(n-1, d_n)$, it means that a_n is scheduled. Add a_n to the schedule, and move to $dp(n-1, d_n - t_n)$. If $dp(n, d_n) = dp(n-1, d_n)$, it means that a_n is not scheduled, and we move to $dp(n-1, d_n)$. Repeat this process until we reach $dp(0, 0)$.

The pseudocode for the algorithm is as follows:

Algorithm 1: Maximize Profit Schedule Algorithm

```
   Result: Maximum profit and job schedule
   Input: List of jobs with processing time, profit, and deadline
1  Sort jobs by deadline;
2  Initialize DP table with zeros;
3  for each job $i$ from 1 to $n$ do
4      for each time $j$ from 1 to latest deadline do
5          Compute the latest start time $t'$ for job $i$;
6          if $t' < 0$, job $i$ cannot be started then
7              DP value is the same as without job $i$;
8          else
9              DP value is the max of without job $i$ or with job $i$ plus its profit;
10         end
11     end
12 end
13 Trace back to find the schedule;
14 return maximum profit and schedule;
```

2 Coding and Experiment

The implemented code in `jobschedule.py` is shown below:

```python
import numpy as np
def max_profit_schedule(jobs):
    '''
    Find the schedule to maximize the profit given a list of jobs.

    Parameters:
      - jobs: a list of jobs, where each job is a tuple of (processing time, profit,
        deadline)

    Returns:
      - max profit: the maximum profit that can be obtained by scheduling the jobs
      - schedule: a list of job indexes that gives the maximum profit (in
       chronological order)
    '''
    # Sort jobs according to deadlines
    jobs.sort(key=lambda x: x[2])
    n = len(jobs)
    ddl_latest = jobs[-1][2] # latest deadline
    dp = np.zeros((n+1, ddl_latest+1), dtype=int) # dynamic programming table

    for i in range(1,n+1): # i: job index+1
      for j in range(1, ddl_latest+1): # j: deadline
        ti = jobs[i-1][0]
        pi = jobs[i-1][1]
        di = jobs[i-1][2]
        tmp = min(di, j) - ti # tmp: the latest time to start job i
        if tmp < 0: # cannot start job i
            dp[i][j] = dp[i-1][j]
        else: # can start job i
            dp[i][j] = max(dp[i-1][j], dp[i-1][tmp] + pi) # max profit of job i

    max_profit = dp[n][ddl_latest]
    # Trace back through the dp table to find the jobs that were scheduled
    schedule = []
    j = ddl_latest
    for i in range(n,0,-1):
      if dp[i][j] != dp[i-1][j]: # job i-1 is scheduled
```

```
            schedule.append(i-1)
            j = j - jobs[i-1][0] # move to the time slot before the start of this job
    schedule.reverse()
    return max_profit, schedule
```

We test the code with the given test cases and the results are shown below:

```
  # test
sample_jobs = [
  [(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4)],
  [(3, 100, 4), (1, 80, 1), (2, 70, 2), (1, 10, 3)],
  [(4, 100, 4), (2, 75, 3), (3, 50, 3), (1, 25, 1)],
  [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (2, 50, 3)],
  [(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4), (2, 50, 3), (1, 80, 2)],
  [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (2, 50, 3), (1, 80, 2), (4, 90,
     4)],
  [(3, 60, 3), (2, 100, 2), (1, 20, 2), (2, 40, 4), (4, 50, 4)],
  [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (4, 50, 4), (1, 80, 2), (4, 90,
     4)],
  [(3, 60, 3), (2, 100, 2), (1, 20, 2), (2, 40, 2), (4, 50, 4), (5, 70, 5)],
  [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (4, 50, 4), (5, 70, 5), (3, 90,
     4)]
]
for jobs in sample_jobs:
  max_profit, schedule = max_profit_schedule(jobs)
  print(max_profit, [jobs[idx] for idx in schedule])

# Output:
160 [(1, 100, 2), (2, 60, 3)]
180 [(1, 80, 1), (3, 100, 4)]
100 [(1, 25, 1), (2, 75, 3)]
160 [(1, 100, 2), (2, 60, 3)]
220 [(1, 100, 2), (1, 80, 2), (2, 40, 4)]
180 [(1, 100, 2), (1, 80, 2)]
140 [(2, 100, 2), (2, 40, 4)]
180 [(1, 100, 2), (1, 80, 2)]
100 [(2, 100, 2)]
190 [(1, 100, 2), (3, 90, 4)]
```

3 Analysis

The initial sorting of the jobs takes $O(n \log n)$ time.
The main loop takes $O(n \cdot d_n)$ time, where d_n is the maximum deadline, since all the operations inside the loop take constant time.
The trace back loop takes $O(n)$ time, since it traces back at most n steps.
Then the total time complexity is $O(n \log n + n \cdot d_n)$.
When d_n is larger than n, the time complexity is $O(n \cdot d_n)$ since $n \cdot d_n$ dominates $n \log n$.
When d_n is smaller than n, the time complexity can also be reduced to $O(n \cdot d_n)$ because sorting n integer numbers in range $[1, d_n]$ can be implemented in $O(n + d_n)$ time using counting sort.
So, the time complexity is $O(n \cdot d_n)$.

The space complexity is also $O(n \cdot d_n)$, since the DP table has size $(n + 1) \times (d_n + 1)$. If we don't want the schedule, we can reduce the space complexity to $O(d_n)$ by only keeping the previous row of the DP table.