CS112 - Luyện Tập 3

Hồ Hữu Tây - 24520029
Dương Hoàng Việt - 24520036

December 15, 2025

1 Phân Tích Bài Toán

Bài toán yêu cầu lựa chọn một tập hợp con các tác vụ và sắp xếp thứ tự thực hiện chúng trên một CPU để đạt được **tổng lợi nhuận lớn nhất** (P_{max}), với điều kiện mỗi tác vụ được chọn phải được hoàn thành trước hoặc đúng **hạn chót** (t_i).

Tham số Tác vụ

Mỗi tác vụ i có các tham số sau:

- Thời gian chạy (Run time): r_i
- Hạn chót (Deadline): t_i
- Lợi nhuận (Profit): p_i

Mục tiêu

Tối đa hóa tổng lợi nhuận:

$$P_{max} = \max_{\text{Tập S}} \sum_{i \in S} p_i$$

với S là tập hợp các tác vụ được chọn và $C_i \leq t_i$ cho mọi $i \in S$ (trong đó C_i là thời điểm hoàn thành tác vụ i).

Chiến lược Sắp xếp Cốt lõi

Trong các bài toán Lập lịch đơn máy với điều kiện hạn chót, nếu một tập hợp tác vụ S là khả thi, thứ tự tối ưu để đảm bảo tất cả tác vụ hoàn thành đúng hạn là **Sắp xếp theo Hạn chót Sớm nhất (Earliest Deadline First - EDF)**: sắp xếp các tác vụ theo thứ tự tăng dần của t_i. Chiến lược này được áp dụng cho mọi tập con tác vụ được chọn.

2 Thiết Kế Thuật Toán theo Subtask

Mã nguồn được thiết kế để xử lý hai giới hạn đầu vào khác nhau:

2.1 Subtask 1: Giới hạn nhỏ ($N \leq 20$)

Với $N \leq 20$, bài toán được giải bằng phương pháp vét cạn kết hợp Bitmask.

Thuật toán

1. Duyệt qua tất cả 2^N tập hợp con (mask) của N tác vụ.
2. Đối với mỗi tập con S:
   - Sắp xếp S theo quy tắc EDF (t_i tăng dần).
   - Tính tổng thời gian hoàn thành C_i theo thứ tự sắp xếp.
   - Nếu $\forall i \in S$, $C_i \leq t_i$, thì tập S là khả thi.
   - Cập nhật P_{max} nếu $\sum_{i \in S} p_i > P_{max}$.

Độ phức tạp

- Số lượng tập con: $O(2^N)$.
- Sắp xếp và kiểm tra mỗi tập con: $O(N \log N + N)$.
- Độ phức tạp tổng thể: $O(2^N \cdot N \log N)$. Với $N = 20$, độ phức tạp này là khoảng $2 \cdot 10^7$, đảm bảo AC (Accepted) trong thời gian cho phép.

2.2 Subtask 2: Giới hạn lớn ($N \leq 500$)

Với $N \leq 500$, ta áp dụng Quy hoạch động (Dynamic Programming - DP) kết hợp kỹ thuật lọc trạng thái (Pruning).

Tiền xử lý

Tất cả N tác vụ được sắp xếp trước theo thứ tự **EDF** (t_i tăng dần).

Định nghĩa Trạng thái DP

Ta định nghĩa $DP[i]$ là tập hợp các trạng thái $\mathbf{s} = (\text{time}, \text{profit})$ đạt được sau khi xét i tác vụ đầu tiên (trong danh sách đã sắp xếp).

- time: Tổng thời gian CPU đã sử dụng.
- profit: Tổng lợi nhuận đạt được.

Công thức Chuyển Trạng thái

Xét tác vụ thứ i (T_i). Từ mỗi trạng thái $\mathbf{s}_{old} = (\text{time}_{old}, \text{profit}_{old}) \in DP[i-1]$, ta có hai chuyển trạng thái:

1. **Không chọn** T_i: Trạng thái mới $\mathbf{s}_{new} = (\text{time}_{old}, \text{profit}_{old})$.
2. **Chọn** T_i: Chỉ khả thi nếu $\text{time}_{old} + r_i \leq t_i$. Trạng thái mới $\mathbf{s}_{new} = (\text{time}_{old} + r_i, \text{profit}_{old} + p_i)$.

Lọc Trạng thái (Pruning)

Để ngăn chặn sự bùng nổ trạng thái, ta chỉ giữ lại các trạng thái **Pareto optimal** trong $DP[i]$. Nếu $\mathbf{s}_1 = (\text{time}_1, \text{profit}_1)$ và $\mathbf{s}_2 = (\text{time}_2, \text{profit}_2)$, ta loại bỏ $\mathbf{s}_1$ nếu:

$$\text{time}_2 \leq \text{time}_1 \quad \text{và} \quad \text{profit}_2 \geq \text{profit}_1$$

(Trạng thái $\mathbf{s}_2$ tốt hơn $\mathbf{s}_1$ vì hoàn thành sớm hơn hoặc cùng lúc mà lợi nhuận không kém hơn).

Mã nguồn thực hiện điều này bằng cách sắp xếp các trạng thái theo time tăng dần, và chỉ giữ lại trạng thái khi profit tăng (kỹ thuật lọc theo time và profit).

Ngoài ra, mã nguồn sử dụng một giới hạn cứng MAX_STATES = 10000 (kỹ thuật Heuristic) để đảm bảo độ phức tạp thời gian.

Truy vết

Sau khi tính toán $DP[N]$, tìm trạng thái có profit lớn nhất. Sử dụng thông tin prev (trạng thái trước) và chosen (tác vụ i có được chọn hay không) để truy vết ngược từ N về 1, thu thập danh sách các id tác vụ đã chọn.

2.3 Tính Phù Hợp của Phương Pháp cho Subtask 2

Giải thích về Tính Phù hợp của Phương pháp

Đối với Subtask 2, giới hạn số lượng tác vụ là $N \leq 500$, và các tham số thời gian/lợi nhuận (r_i, t_i, p_i) có thể lên đến 10^{12}.

a. **Loại bỏ Vét cạn:** Phương pháp vét cạn (Bitmask) của Subtask 1 có độ phức tạp $O(2^N)$ là không khả thi (2^{500} là quá lớn), buộc phải tìm kiếm phương pháp có độ phức tạp đa thức hoặc bán đa thức.

b. **Lựa chọn Quy hoạch động (DP):** Bài toán Lập lịch đơn máy với tối đa hóa lợi nhuận $(1|\text{prec}, r_i|\sum w_i U_i)$ là bài toán NP-hard. Tuy nhiên:

- Bằng cách áp dụng thứ tự sắp xếp theo **Hạn chót Sớm nhất (EDF)**, ta chuyển bài toán thành một cấu trúc có thể áp dụng DP. Sắp xếp theo EDF (t_i tăng dần) là điều kiện tiên quyết để đảm bảo tính tối ưu của lời giải.
- Ta sử dụng DP theo cấu trúc DP[i] (xét i tác vụ đầu tiên) với trạng thái $(\text{time}, \text{profit})$.

c. **Vấn đề và Giải pháp (Lọc Trạng thái):** DP truyền thống có thể bùng nổ vì time và profit là số rất lớn ($\leq 10^{12}$). Số lượng trạng thái có thể lên tới 2^N trong trường hợp xấu nhất.

- **Kỹ thuật Lọc Trạng thái (State Pruning):** Đây là chìa khóa để thuật toán hoạt động. Ta chỉ lưu trữ các trạng thái $\mathbf{s} = (\text{time}, \text{profit})$ là **Pareto Optimal**.
- Cụ thể, nếu tồn tại $\mathbf{s}'$ thỏa mãn $\text{time}' \leq \text{time}$ và $\text{profit}' \geq \text{profit}$, thì trạng thái $\mathbf{s}$ bị loại bỏ. Việc này giúp giảm đáng kể số lượng trạng thái cần xét ở mỗi bước, khiến DP có thể chạy được trong thực tế.

Chi tiết Heuristic

Mặc dù phương pháp chính là DP, quá trình lọc trạng thái (Pruning) bao gồm một **Heuristic** quan trọng để khống chế độ phức tạp.

Tên Thuật toán Heuristic Heuristic được sử dụng là **Giới hạn số lượng trạng thái** (State Space Limiting).

Giải thích tại sao chọn Heuristic đó Trong lý thuyết DP cho các bài toán NP-hard, đôi khi không gian trạng thái M (số trạng thái tối ưu Pareto) vẫn có thể lớn theo cấp số mũ hoặc đa thức bậc cao. Để đảm bảo chương trình chạy trong giới hạn thời gian thực tế (time limit per test : $5s$), ta cần đặt một giới hạn cứng cho số lượng trạng thái được lưu trữ giữa các bước DP:

- **Mục đích:** Đảm bảo độ phức tạp thời gian xấp xỉ $O(N \cdot M \log M)$, trong đó M là một hằng số nhỏ.
- **Lý do chọn:** Phương pháp này là cách tiếp cận phổ biến để giải các bài toán NP-hard/Pseudo-polynomial trong các cuộc thi lập trình (ví dụ: các biến thể của bài toán Knapsack). Nó chuyển đổi thuật toán từ lý thuyết phức tạp sang một thuật toán **gần tối ưu** và **hiệu quả về mặt thời gian**.

Độ phức tạp

- Sắp xếp ban đầu: $O(N \log N)$.
- DP: N bước. Mỗi bước xử lý $O(M)$ trạng thái, sắp xếp $O(M \log M)$ (với M là số lượng trạng thái sau lọc).
- Độ phức tạp tổng thể (với Heuristic): $O(N \cdot M \log M)$. Với $N = 500$ và $M = 10000$, độ phức tạp là $\approx O(6.5 \times 10^7)$, rất hiệu quả.