Báo Cáo Nhóm 6

Phân tích và thiết kế thuật toán(24520002)

Nhóm: Nhóm 14

Thành viên:
Đặng Phú Duy 24520010

Mai Quốc Anh 24520002

Ngày 15 tháng 12 năm 2025

1 Tóm tắt đề bài

Server của Fucs nhận được N tác vụ cần xử lý. Do tài nguyên CPU hữu hạn, các tác vụ chỉ có thể được thực hiện **tuần tự**. Khi CPU bắt đầu xử lý một tác vụ thì tác vụ đó phải chạy đến khi hoàn thành (không được ngắt quãng).

Mỗi tác vụ i có ba tham số:

- Thời gian chạy: r_i
- Hạn hoàn thành (deadline): t_i
- Lợi nhuận: p_i

Ta có thể **chọn hoặc không chọn** mỗi tác vụ. Nếu tác vụ i được chọn thì nó phải hoàn thành tại thời điểm không muộn hơn t_i. Nếu hoàn thành sau t_i thì tác vụ đó thất bại và không mang lại lợi nhuận.

Mục tiêu là chọn một tập con các tác vụ và sắp xếp thứ tự thực hiện sao cho tổng lợi nhuận thu được là lớn nhất.

Dữ liệu vào

- Dòng đầu tiên chứa số nguyên dương N.
- N dòng tiếp theo, mỗi dòng gồm ba số nguyên r_i, t_i, p_i.

Dữ liệu ra

- Dòng thứ nhất ghi tổng lợi nhuận lớn nhất.
- Dòng thứ hai ghi dãy chỉ số (ID) các tác vụ được chọn theo thứ tự thực hiện.

Ràng buộc

- Subtask 1: $N \leq 20$, yêu cầu nghiệm tối ưu tuyệt đối.
- Subtask 2: $N \leq 500$, $1 \leq r_i, t_i, p_i \leq 10^{12}$, cho phép nghiệm xấp xỉ.

2 Phương pháp thiết kế thuật toán

2.1 Subtask 1: Thiết kế thuật toán tối ưu tuyệt đối

Phương pháp sử dụng: Brute Force kết hợp kiểm tra hoán vị

Do giới hạn của subtask 1 là $N \leq 20$, ta có thể duyệt toàn bộ không gian nghiệm để tìm ra lời giải tối ưu tuyệt đối.

Nguyên lý

Mỗi tác vụ có thể được **chọn hoặc không chọn**. Với mỗi tập con các tác vụ được chọn, ta cần tìm một thứ tự thực hiện sao cho:

- Các tác vụ được xử lý tuần tự
- Tổng thời gian chạy tại thời điểm hoàn thành mỗi tác vụ không vượt quá deadline của tác vụ đó

Nếu tồn tại một thứ tự hợp lệ thì tổng lợi nhuận của tập con đó được tính bằng tổng p_i của các tác vụ trong tập.

Cách áp dụng vào bài toán

Thuật toán được thực hiện theo các bước sau:

1. Duyệt tất cả các tập con của N tác vụ (tổng cộng 2^N tập con).
2. Với mỗi tập con, duyệt tất cả các hoán vị thứ tự thực hiện các tác vụ trong tập con.
3. Với mỗi hoán vị, mô phỏng quá trình xử lý:
    - Cộng dồn thời gian chạy
    - Kiểm tra tại mỗi tác vụ xem thời điểm hoàn thành có vượt deadline hay không
4. Nếu hoán vị hợp lệ, tính tổng lợi nhuận và cập nhật nghiệm tốt nhất.

Do duyệt toàn bộ không gian nghiệm nên thuật toán đảm bảo tìm được **nghiệm tối ưu tuyệt đối** cho subtask 1.

Subtask 2: Phương pháp heuristic

Phương pháp thiết kế được sử dụng: Đối với Subtask 2, nhóm sử dụng phương pháp **heuristic**, cụ thể là **Beam Search** (tìm kiếm theo chiều rộng có giới hạn).

Nguyên lý của phương pháp: Beam Search là một biến thể của tìm kiếm theo chiều rộng, trong đó tại mỗi bước chỉ giữ lại một số lượng hữu hạn các trạng thái tốt nhất (gọi là *beam*) thay vì duyệt toàn bộ không gian trạng thái. Mỗi trạng thái đại diện cho một nghiệm tạm thời. Việc giới hạn số trạng thái giúp giảm đáng kể chi phí tính toán, đồng thời vẫn giữ được nhiều phương án tiềm năng để tránh rơi vào nghiệm cục bộ. Beam Search không đảm bảo tìm nghiệm tối ưu tuyệt đối, nhưng thường cho nghiệm gần đúng có chất lượng cao trong thời gian ngắn, phù hợp với các bài toán lớn.

Cách áp dụng vào bài toán: Trong bài toán lập lịch tác vụ, mỗi trạng thái trong Beam Search biểu diễn một lịch thực hiện tạm thời, được mô tả bởi hai đại lượng: tổng thời gian thực thi các tác vụ đã chọn và tổng lợi nhuận thu được.

Các tác vụ được sắp xếp trước theo **thứ tự deadline tăng dần**. Việc này giúp đảm bảo rằng tại mỗi bước, lịch thực thi đang xét là hợp lệ đối với các tác vụ trước đó và việc kiểm tra ràng buộc deadline khi thêm tác vụ mới trở nên đơn giản.

Tại mỗi tác vụ, từ mỗi trạng thái hiện có, thuật toán sinh ra hai phương án:

- Không chọn tác vụ: trạng thái giữ nguyên tổng thời gian và lợi nhuận.
- Chọn tác vụ: tổng thời gian tăng thêm thời gian chạy của tác vụ và tổng lợi nhuận tăng thêm lợi nhuận tương ứng; phương án này chỉ được chấp nhận nếu tổng thời gian không vượt quá deadline của tác vụ.

Hai tập trạng thái sinh ra được hợp nhất bằng **kỹ thuật merge tuyến tính** theo tổng thời gian thực thi, giúp tránh việc sắp xếp lại tốn kém. Sau đó, thuật toán áp dụng **dominance pruning** để loại bỏ các trạng thái kém hiệu quả, tức là những trạng thái bị chi phối bởi trạng thái khác có thời gian thực thi không lớn hơn nhưng lợi nhuận lớn hơn hoặc bằng.

Cuối cùng, để kiểm soát kích thước không gian tìm kiếm, thuật toán chỉ giữ lại một số lượng tối đa các trạng thái tốt nhất thông qua tham số *beam width*. Nhờ đó, Beam Search được áp dụng hiệu quả cho bài toán có kích thước lớn và cho nghiệm gần đúng chất lượng cao, đáp ứng yêu cầu của Subtask 2.

3 Tính phù hợp của phương pháp :

3.1 Subtask 1 :

Trong subtask 1, số lượng tác vụ bị giới hạn nhỏ ($N \leq 20$) và yêu cầu bắt buộc là phải tìm được **nghiệm tối ưu tuyệt đối**. Do đó, việc ưu tiên tính chính xác hơn hiệu năng là hoàn toàn phù hợp.

Phương pháp **brute force duyệt toàn bộ không gian nghiệm** đáp ứng trực tiếp yêu cầu này vì:

- Thuật toán xem xét **mọi tập con** tác vụ có thể chọn và **mọi thứ tự thực hiện** tương ứng.
- Không bỏ sót bất kỳ phương án hợp lệ nào, do đó luôn tìm được nghiệm tối ưu toàn cục.

Mặc dù độ phức tạp tính toán lớn (giai thừa), nhưng với $N \leq 20$, không gian tìm kiếm vẫn đủ nhỏ để xử lý trong thời gian cho phép của subtask 1. Việc áp dụng các phương pháp xấp xỉ hoặc heuristic trong trường hợp này là không cần thiết và có thể dẫn đến nghiệm không tối ưu, vi phạm yêu cầu đề bài.

Vì vậy, phương pháp brute force là **phù hợp nhất** cho subtask 1 do:

- Đảm bảo tối ưu tuyệt đối
- Phù hợp với kích thước dữ liệu nhỏ
- Cách triển khai đơn giản và dễ kiểm chứng tính đúng đắn

3.2 Subtask 2 :

Đối với Subtask 2, bài toán có số lượng tác vụ lớn ($N \leq 500$) cùng với các giá trị thời gian chạy, deadline và lợi nhuận rất lớn, khiến không gian nghiệm trở nên cực kỳ rộng. Trong điều kiện này, các phương pháp tìm nghiệm tối ưu tuyệt đối như brute force hoặc quy hoạch động không khả thi do vượt quá giới hạn thời gian và bộ nhớ. Vì vậy, việc sử dụng phương pháp xấp xỉ là cần thiết.

Thuật toán được sử dụng: Heuristic – Beam Search.

Beam Search là một thuật toán heuristic phù hợp với các bài toán tối ưu tổ hợp có không gian nghiệm lớn, cho phép cân bằng giữa chất lượng nghiệm và chi phí tính toán. Khác với các heuristic ngẫu nhiên như Genetic Algorithm hay Simulated Annealing, Beam Search có tính **xác định** (deterministic), không phụ thuộc vào yếu tố ngẫu nhiên, giúp kết quả ổn định và dễ kiểm soát trong môi trường chấm tự động.

Lý do lựa chọn Beam Search:

- Beam Search cho phép duy trì nhiều phương án lập lịch song song, giúp tránh rơi vào nghiệm cục bộ như các phương pháp greedy đơn thuần.
- Thuật toán tận dụng được cấu trúc của bài toán lập lịch bằng cách sắp xếp các tác vụ theo deadline, từ đó kiểm soát tốt ràng buộc thời gian.
- Việc kết hợp với dominance pruning giúp loại bỏ sớm các trạng thái kém hiệu quả, giảm đáng kể kích thước không gian tìm kiếm.
- Beam Search cho phép kiểm soát trực tiếp độ phức tạp thông qua tham số *beam width*, đảm bảo thuật toán chạy trong giới hạn thời gian cho phép.

Cấu hình tham số và lý do lựa chọn:

Trong cài đặt, thuật toán sử dụng tham số *beam width* với giá trị:

$$\texttt{BEAM\_WIDTH} = 300{,}000$$

Giá trị này được lựa chọn nhằm đảm bảo:

- Beam đủ lớn để giữ lại các phương án lập lịch tiềm năng, giúp nâng cao chất lượng nghiệm gần đúng.

- Đồng thời vẫn đảm bảo giới hạn bộ nhớ và thời gian thực thi (5 giây, 1024 MB) nhờ các kỹ thuật tối ưu như merge tuyến tính và dominance pruning.

Do Beam Search là heuristic xác định, thuật toán không sử dụng các tham số ngẫu nhiên như temperature, mutation rate hay số vòng lặp ngẫu nhiên, giúp quá trình chạy ổn định và dễ phân tích.

Nhờ các đặc điểm trên, Beam Search là phương pháp phù hợp để giải quyết Subtask 2, cho phép tìm được nghiệm gần đúng chất lượng cao trong điều kiện tài nguyên hạn chế.

4 Phân tích độ phức tạp

4.1 Subtask 1 :

Độ phức tạp thời gian

Thuật toán của subtask 1 sử dụng phương pháp brute force, duyệt toàn bộ các tập con tác vụ và mọi hoán vị thứ tự thực hiện của mỗi tập con.

- Có 2^N tập con các tác vụ có thể được chọn.
- Với mỗi tập con có k tác vụ, số hoán vị cần kiểm tra là $k!$.
- Với mỗi hoán vị, ta mô phỏng quá trình thực thi trong $O(k)$ thời gian để kiểm tra deadline và tính lợi nhuận.

Do đó, độ phức tạp thời gian trong trường hợp xấu nhất là:

$$O\left(\sum_{k=1}^{N} \binom{N}{k} \cdot k! \cdot k\right)$$

Ta có thể chặn trên bởi:

$$O(N \cdot N!) = O(N!)$$

Với $N \leq 20$, thuật toán vẫn có thể chạy được trong phạm vi subtask 1, dù không phù hợp cho dữ liệu lớn.

Độ phức tạp không gian

Thuật toán chỉ cần:

- Lưu danh sách các tác vụ đầu vào: $O(N)$
- Lưu một hoán vị hiện tại và các biến mô phỏng thời gian chạy: $O(N)$
- Lưu nghiệm tốt nhất tìm được: $O(N)$

Do đó, độ phức tạp không gian là:

$$O(N)$$

Thuật toán sử dụng rất ít bộ nhớ, nhưng đánh đổi bằng chi phí thời gian rất lớn để đảm bảo tìm được nghiệm tối ưu tuyệt đối.

4.2 Subtask 2 :

Độ phức tạp thời gian

Ban đầu, các tác vụ được sắp xếp theo thứ tự deadline tăng dần, bước này có độ phức tạp:

$$O(N \log N)$$

Sau đó, thuật toán lần lượt xử lý từng tác vụ. Tại mỗi tác vụ, các bước chính bao gồm:

- Sinh hai luồng trạng thái (chọn và không chọn tác vụ) từ beam hiện tại.
- Hợp nhất hai luồng trạng thái bằng kỹ thuật merge tuyến tính, với độ phức tạp $O(B)$.
- Thực hiện dominance pruning bằng cách duyệt một lần qua các trạng thái đã hợp nhất, cũng có độ phức tạp $O(B)$.

Do đó, tổng chi phí xử lý cho mỗi tác vụ là $O(B)$. Với N tác vụ, độ phức tạp thời gian tổng thể của thuật toán là:

$$O(N \cdot B + N \log N)$$

Vì B lớn hơn nhiều so với $\log N$, có thể coi độ phức tạp thời gian xấp xỉ là:

$$O(N \cdot B)$$

Độ phức tạp bộ nhớ

Thuật toán lưu trữ các trạng thái trong beam tại mỗi bước và thông tin truy vết cho mỗi tác vụ. Bộ nhớ chính bao gồm:

- Beam hiện tại, tối đa B trạng thái.
- Dữ liệu truy vết (parent index và task id) cho mỗi trạng thái tại mỗi bước.

Do đó, độ phức tạp bộ nhớ xấp xỉ:

$$O(N \cdot B)$$

Tuy nhiên, nhờ áp dụng dominance pruning và giới hạn beam width, số trạng thái thực tế được lưu thường nhỏ hơn giới hạn tối đa, giúp thuật toán đáp ứng được giới hạn bộ nhớ của bài toán.

Nhìn chung, thuật toán Beam Search có độ phức tạp tuyến tính theo số tác vụ và beam width, phù hợp để giải quyết Subtask 2 trong điều kiện giới hạn thời gian và bộ nhớ.