Đại học quốc gia TP.HCM
Trường đại học công nghệ thông tin

Ngành Khoa học Máy tính

Môn học: Phân tích & Thiết kế thuật toán

BÁO CÁO BÀI TẬP: TRÒ CHƠI HAI MẢNG (DELEGAME)

Sinh viên thực hiện:
Võ Lê Ngọc Thịnh
Vũ Minh Phương

Giảng viên hướng dẫn:
Nguyễn Thanh Sơn

Thành phố Hồ Chí Minh, Tháng 1 năm 2026

Contents

1 Phương pháp thiết kế thuật toán

Để giải quyết bài toán **DELEGAME**, chúng tôi sử dụng kết hợp hai phương pháp chính: **Tư duy ngược (Backward Induction)** trong Lý thuyết trò chơi và **Chiến lược Tham lam (Greedy Strategy)**.

1.1 Nguyên lý thiết kế

- **Lý thuyết trò chơi (Game Theory) - Tư duy ngược:** Đây là một trò chơi đối kháng có tổng bằng 0 (Zero-sum game), hữu hạn, không có chu trình và thông tin hoàn hảo. Trạng thái thắng thua của trò chơi có thể được xác định bằng cách suy luận từ trạng thái cuối cùng (khi mảng B đã được duyệt hết hoặc không còn nước đi hợp lệ) ngược về trạng thái bắt đầu.

  Thay vì mô phỏng trò chơi từ lượt đầu tiên (vốn sẽ tạo ra cây trò chơi với độ phức tạp bùng nổ), ta xét từ phần tử cuối cùng của mảng B. Nếu ta biết điều kiện để thắng ở lượt $i+1$, ta có thể suy ra hành động tối ưu cần thực hiện ở lượt i.

- **Chiến lược Tham lam (Greedy):** Tại mỗi bước, người chơi luôn muốn thực hiện một nước đi "tối ưu cục bộ" để ép đối thủ vào thế khó nhất có thể. Trong bài toán này, khi chọn một vị trí k thỏa mãn $A[k] = B[i]$, toàn bộ phần hậu tố từ k trở đi sẽ bị xóa. Do đó, mảng còn lại cho đối thủ là đoạn tiền tố $A[0 \dots k-1]$. Để hạn chế tối đa không gian hoạt động của đối thủ ở lượt sau, người chơi hiện tại cần để lại đoạn tiền tố **ngắn nhất có thể**. Điều này dẫn đến quyết định tham lam: luôn chọn vị trí xuất hiện **sớm nhất** (chỉ số nhỏ nhất) của $B[i]$ trong mảng A.

1.2 Cách áp dụng vào bài toán

1. **Tiền xử lý:** Duyệt mảng A một lần để lưu trữ vị trí xuất hiện **đầu tiên** của mỗi giá trị số nguyên. Ta sử dụng cấu trúc Hash Map (hoặc mảng tra cứu trực tiếp) để đảm bảo thời gian truy xuất là $O(1)$.

2. **Quy trình duyệt ngược:** Gọi $Threshold$ là "ngưỡng thua". Nếu đến lượt của một người chơi mà độ dài mảng A hiện tại nhỏ hơn hoặc bằng $Threshold$, họ sẽ thua (vì đối thủ đã chặn đường thắng ở các lượt sau).

   Ta duyệt mảng B từ phần tử cuối cùng $B[m]$ về $B[1]$:

   - Khởi tạo $Threshold = \infty$ (sau lượt cuối cùng thì không còn ai chơi nữa, nên coi như trạng thái vô cùng an toàn).
   - Tại mỗi phần tử $B[i]$, tìm vị trí xuất hiện đầu tiên của nó trong A, gọi là pos.
   - **Kiểm tra điều kiện:** Nếu $pos < Threshold$, nghĩa là người chơi tại lượt i có thể thực hiện nước đi này và sau khi cắt mảng (còn lại độ dài pos), đối thủ ở lượt $i+1$ sẽ bị rơi vào thế thua (vì độ dài mảng còn lại nhỏ hơn ngưỡng cần thiết để đối thủ thắng).
   - **Cập nhật:** Nếu điều kiện thỏa mãn, ta cập nhật $Threshold = pos$. Ngược lại, nếu $pos \geq Threshold$ hoặc không tìm thấy $B[i]$ trong A, người chơi tại lượt này không thể ép đối thủ thua, tức là họ sẽ thua $\rightarrow$ gán $Threshold = \infty$.

3. **Kết luận:** Sau khi duyệt về đến đầu mảng B, nếu $Threshold \neq \infty$, Alice (người đi đầu) có chiến thuật thắng. Ngược lại, Bob thắng.

2 Tính phù hợp của phương pháp và Chứng minh toán học

Phương pháp Greedy kết hợp với Quy hoạch động ngược (Backward Induction) là lựa chọn tối ưu cho bài toán này. Dưới đây là chứng minh chặt chẽ về tính đúng đắn của giải thuật dựa trên tính đơn điệu của không gian trạng thái.

2.1 Định nghĩa bài toán dưới dạng toán học

Gọi $Win(i, L)$ là hàm boolean trả về `True` nếu người chơi tại lượt thứ i (đang xét phần tử B_i) có chiến thuật thắng chắc chắn khi mảng A hiện tại có độ dài L (tức là đoạn tiền tố $A[0 \dots L-1]$).

Bài toán yêu cầu xác định giá trị của $Win(1, n)$.

2.2 Bổ đề về tính đơn điệu (Monotonicity Lemma)

Phát biểu: Với mọi lượt i và hai độ dài L_1, L_2 sao cho $L_1 < L_2$, ta luôn có:

$$Win(i, L_1) \implies Win(i, L_2)$$

Chứng minh: Nếu $Win(i, L_1)$ là `True`, tức là tồn tại một nước đi $k < L_1$ sao cho $A[k] = B[i]$ và đẩy đối thủ vào thế thua ở lượt $i + 1$. Vì $L_1 < L_2$, đoạn tiền tố độ dài L_1 là tập con của đoạn tiền tố độ dài L_2. Do đó, nước đi hợp lệ k trong không gian L_1 cũng chắc chắn là nước đi hợp lệ trong không gian L_2. $\Rightarrow$ Người chơi vẫn có thể chọn nước đi k đó để chiến thắng. $\Rightarrow$ Khả năng thắng không giảm đi khi không gian tìm kiếm mở rộng.

2.3 Chứng minh tính đúng đắn của chiến thuật Tham lam (Greedy Proof)

Giả thuyết: Tại lượt i, chiến thuật tối ưu là chọn chỉ số k nhỏ nhất thỏa mãn $A[k] = B[i]$ (gọi là k_{min}).

Chứng minh bằng phản chứng (Proof by Contradiction): Giả sử chiến thuật tham lam là **sai**. Điều này có nghĩa là tồn tại một trạng thái mà tại đó:

1. Việc chọn k_{min} dẫn đến người chơi hiện tại bị THUA.
2. Tồn tại một chỉ số khác $k_{opt} > k_{min}$ ($A[k_{opt}] = B[i]$) giúp người chơi hiện tại THẮNG.

Phân tích điều kiện thắng/thua:

- Người chơi chọn k_{min} bị THUA $\iff$ Đối thủ tại lượt $i + 1$ THẮNG với mảng còn lại có độ dài k_{min}. Tức là: $Win(i + 1, k_{min}) =$ `True`.
- Người chơi chọn k_{opt} sẽ THẮNG $\iff$ Đối thủ tại lượt $i + 1$ THUA với mảng còn lại có độ dài k_{opt}. Tức là: $Win(i + 1, k_{opt}) =$ `False`.

Từ hai điều trên, ta có:

$$Win(i + 1, k_{min}) = \texttt{True} \quad \text{và} \quad Win(i + 1, k_{opt}) = \texttt{False}$$

Tuy nhiên, theo giả thiết ban đầu $k_{opt} > k_{min}$. Áp dụng **Bổ đề về tính đơn điệu**, nếu $Win(i + 1, k_{min})$ là `True` thì $Win(i + 1, k_{opt})$ bắt buộc phải là `True`.

Điều này tạo ra mâu thuẫn ($True = False$). $\Rightarrow$ Giả sử ban đầu là sai. $\Rightarrow$ Không tồn tại trường hợp chọn k_{opt} tốt hơn k_{min}. $\Rightarrow$ Chiến thuật chọn k_{min} (vị trí xuất hiện sớm nhất) luôn là tối ưu hoặc tương đương với tối ưu.

2.4 Kết luận

Dựa trên chứng minh trên, việc ta chỉ lưu trữ và xét vị trí xuất hiện đầu tiên (first_occurrence) của mỗi phần tử trong thuật toán là hoàn toàn chính xác và đầy đủ. Phương pháp này giúp giảm độ phức tạp từ việc phải duyệt toàn bộ cây trò chơi xuống còn $O(N + M)$ mà không bỏ sót nghiệm tối ưu nào.

3 Phân tích độ phức tạp

3.1 Độ phức tạp thời gian (Time Complexity)

Thuật toán bao gồm hai giai đoạn chính:

- **Giai đoạn 1 (Tiền xử lý):** Ta duyệt qua mảng A có kích thước n một lần duy nhất để xác định vị trí xuất hiện đầu tiên của các phần tử. Việc chèn vào Hash Map (trong Python là Dictionary) có độ phức tạp trung bình là $O(1)$. $\Rightarrow$ Thời gian: $O(n)$.
- **Giai đoạn 2 (Duyệt ngược):** Ta duyệt qua mảng B có kích thước m từ cuối về đầu. Tại mỗi bước, ta thực hiện thao tác tra cứu trong Hash Map mất $O(1)$ và so sánh số học mất $O(1)$. $\Rightarrow$ Thời gian: $O(m)$.

Tổng độ phức tạp thời gian là:

$$O(n) + O(m) = O(n + m)$$

Với $n, m \leq 2 \times 10^5$, thuật toán thực thi trong thời gian rất ngắn (khoảng vài trăm mili-giây), hoàn toàn thỏa mãn giới hạn 5 giây của đề bài.

3.2 Độ phức tạp không gian (Space Complexity)

- Ta cần lưu trữ mảng A và B: $O(n + m)$.
- Cấu trúc Hash Map first_occurrence lưu trữ vị trí đầu tiên của các số trong mảng A. Trong trường hợp xấu nhất (tất cả phần tử trong A đều khác nhau), kích thước Map là n. $\Rightarrow$ Không gian phụ trợ: $O(n)$.

Tổng độ phức tạp không gian là $O(n + m)$. Với $n, m \leq 2 \times 10^5$ phần tử kiểu số nguyên (4 bytes), bộ nhớ tiêu tốn khoảng vài MB, thấp hơn rất nhiều so với giới hạn 50 MB.