Pots of Gold Game Problem using Dynamic Programming

Đặng Nhật Minh

Ngày 2 tháng 1 năm 2025

Tóm tắt nội dung

Trò chơi Pots of Gold là một bài toán điển hình về trò chơi hai người, trong đó các người chơi lần lượt lấy các nồi vàng (pots of gold) từ hai đầu của một dãy. Mỗi người đều cố gắng tối đa hóa số vàng thu được cho riêng mình, giả sử cả hai cùng chơi tối ưu. Bài viết này trình bày tổng quan về trò chơi, lý do nghiên cứu, các khái niệm liên quan, ví dụ từng bước, và tập trung phân tích các cách tiếp cận bằng dynamic programming. Các phiên bản giải pháp bao gồm từ đệ quy cơ bản đến phương pháp dynamic programming với bảng 2 chiều và cuối cùng là tối ưu hóa không gian bộ nhớ xuống còn $O(n)$.

1 Giới thiệu

Bài toán *Pots of Gold* mô tả một trò chơi hai người: có một hàng gồm n nồi vàng, mỗi nồi chứa một số lượng vàng nhất định. Hai người chơi (A và B) thay phiên nhau lấy **một** nồi từ **một trong hai đầu** của hàng. Mục tiêu là tính xem người chơi A (lượt đi trước) có thể đảm bảo tối đa bao nhiêu vàng cho bản thân, giả sử cả hai đều chơi *tối ưu*.

1.1 Lịch sử ngắn gọn

Trò chơi này có quan hệ gần gũi với các bài toán *chọn phần tử theo lượt* trong lý thuyết trò chơi hai người, thường xuất hiện trong lĩnh vực **AI game theory**, phân tích độ phức tạp thuật toán và thi lập trình. Việc sắp xếp các nồi vàng theo hàng, cùng với lựa chọn từ **hai đầu**, là ví dụ trực quan giúp minh họa hai khái niệm: *Dynamic Programming* và *minimax*.

1.2 Đặc điểm chính

Một số điểm nổi bật của vấn đề:

- **Cả hai người chơi cùng chơi tối ưu**: Mỗi người đều hướng đến việc tối đa hóa số vàng họ thu được.
- **Lựa chọn từ hai đầu**: Người chơi chỉ được phép lấy từ đầu bên trái hoặc đầu bên phải của dãy.
- **Ứng dụng Dynamic Programming**: Cách tiếp cận đệ quy đơn giản có độ phức tạp hàm mũ, nhưng có thể giảm xuống dạng đa thức khi sử dụng DP.
- **Tối ưu bộ nhớ**: Ngoài cách lưu trữ bảng 2 chiều $O(n^2)$, ta có thể giảm còn $O(n)$ nhờ kỹ thuật quản lý chỉ số khéo léo.

2 Động lực (Motivation)

2.1 Phát biểu Bài toán

Cho một mảng n nồi vàng, với chỉ số i chứa coins[i] đồng vàng. Hai người chơi lần lượt lấy một nồi từ *một trong hai đầu* (trái hoặc phải). Mỗi người chơi luôn chọn phương án đem lại lợi ích tối đa cho mình. Nhiệm vụ là tìm **tổng số vàng** mà người chơi A (chơi trước) có thể đảm bảo có được, dưới điều kiện đối thủ (B) cũng chơi tối ưu.

2.2 Ứng dụng Thực tế

Mặc dù đây chỉ là trò chơi minh hoạ, ý tưởng "lựa chọn tối ưu luân phiên" này có mặt trong nhiều bài toán thực tế:

- **Quản lý đầu tư hoặc tài nguyên**: Khi hai doanh nghiệp (hoặc hai người) cùng tiếp cận một tập nguồn lực hữu hạn và phải chọn lần lượt.
- **Đàm phán chia tài sản**: Mỗi người lần lượt chọn món đồ giá trị từ hai đầu danh sách.
- **Lịch biểu và bảo mật**: Phân chia hạng mục hoặc suất thời gian trong điều kiện cạnh tranh.

Trong các trường hợp này, dynamic programming cho phép mô hình hoá chiến lược **tối ưu** khi phải ra quyết định có tính cạnh tranh.

3 Bối cảnh & Công việc Liên quan

Trong phân tích độ phức tạp, bài toán hai người chơi thay phiên nhau chọn trong một cấu trúc thường gắn với:

- **Lý thuyết Minimax**: Phân tích kết quả trong trò chơi hai người có tổng không đổi (zero-sum).
- **Interval DP (Dynamic Programming trên đoạn)**: Giải quyết các bài toán có cấu trúc đoạn như nhân ma trận, chia đa giác, v.v.

Bên cạnh đó, có một số phương pháp khác như:

- **Đệ quy có nhớ (memoization)**: Duy trì kết quả trung gian trong một bảng 2 chiều.
- **Dynamic Programming dạng bảng (bottom-up DP)**: Tính kết quả từ các đoạn nhỏ đến đoạn lớn.
- **Thuật toán tham lam (greedy)**: Thường không đảm bảo tối ưu *khi đối thủ cũng tối ưu.*

4 Giao diện (Giả định và Dữ liệu Đầu vào)

Vì đây chủ yếu là một **thuật toán** thay vì một cấu trúc dữ liệu phức tạp, phần này nêu các giả định về đầu vào, đầu ra:

1. **Mảng đầu vào**: Chứa các giá trị vàng, coins$[0 \dots n-1]$.
2. **Số lượng nồi**: $n = \text{length(coins)}$.

3. **Các lượt chơi**: Hai người chơi A và B, thay nhau, mỗi lượt lấy từ một trong hai đầu.
4. **Đầu ra**: Tổng vàng tối đa mà người chơi A có thể *đảm bảo* chiếm được.

Giả sử giá trị vàng không âm và $n \geq 1$.

5 Ví dụ Minh hoạ Thuật toán

Ta xét ví dụ $n = 4$:

$$\text{coins} = [4,\ 7,\ 2,\ 9].$$

1. **Người chơi A** có thể lấy 4 (bên trái) hoặc 9 (bên phải).
2. Giả sử A lấy 4, dãy còn lại [7, 2, 9]. **Người chơi B** muốn tối đa số vàng, nên chọn giữa 7 và 9.
3. Giả sử B lấy 9, dãy còn lại [7, 2], lúc này A chỉ còn lựa chọn giữa 7 và 2, v.v.

Mỗi lựa chọn đều có chiến thuật sâu hơn do đối thủ cũng chơi tối ưu. Phương pháp **dynamic programming** sẽ giúp ta tìm được nước đi tốt nhất mà không phải liệt kê hoàn toàn mọi khả năng.

6 Phân tích: Giải pháp Dynamic Programming

6.1 Công thức truy hồi

Xét hàm $F(i, j)$ biểu diễn số vàng *tối đa* người chơi hiện tại có thể lấy từ đoạn nồi vàng chỉ số i đến j. Lưu ý rằng sau khi người chơi hiện tại lấy xong, người chơi kế tiếp cũng chơi tối ưu:

$$F(i,j) = \max\Big\{\text{coins}[i]+\min(F(i+2,j),\ F(i+1,j-1)),\quad \text{coins}[j]+\min(F(i+1,j-1),\ F(i,j-2))\Big\}.$$

Thành phần $\min(\cdot)$ trong công thức hàm ý: sau khi ta chọn một đầu, đối phương sẽ chọn tiếp (cũng tối ưu) và để lại *kết quả xấu nhất* cho ta. Điều này phản ánh việc đối phương cũng tìm cách **hạn chế** số vàng ta có thể nhận được sau này.

6.2 Giả mã (Pseudocode) cho phương pháp Đệ quy có nhớ

Độ phức tạp thời gian: $O(n^2)$, vì có $O(n^2)$ cặp (i, j) và mỗi cặp chỉ tốn $O(1)$ thời gian khi đã lưu kết quả.
Độ phức tạp không gian: $O(n^2)$ cho bảng lưu trữ `dp` và $O(n)$ cho ngăn xếp đệ quy.

6.3 Dynamic Programming Duyệt Bảng (Bottom-Up) và Tối ưu xuống $O(n)$ Bộ nhớ

Dùng công thức truy hồi tương tự, ta có thể xây dựng kết quả từ các đoạn ngắn (độ dài 1) dần lên đến cả dãy (độ dài n). Giải pháp duyệt bảng thông thường dùng một mảng 2 chiều $dp[i][j]$, chiếm $O(n^2)$ bộ nhớ. Tuy nhiên, nhờ quan sát rằng $F(i, j)$ chỉ phụ thuộc vào $F(i+2, j)$, $F(i+1, j-1)$ và $F(i, j-2)$, ta chỉ cần duy trì **một vài đường chéo** của mảng. Bằng cách ghi đè cẩn thận, chúng ta có thể rút gọn bộ nhớ xuống $O(n)$.

- **Độ phức tạp thời gian**: $O(n^2)$ (vẫn cần duyệt qua tất cả các đoạn con).
- **Độ phức tạp không gian**: $O(n)$ với cách lưu trữ tối ưu.

Algorithm 1 Dynamic Programming đệ quy có nhớ (Memorized DP) cho bài toán Pots of Gold

Require: coins$[0..n-1]$: Mảng giá trị vàng
Ensure: Trả về số vàng tối đa mà người chơi A có thể giành

```
function MaxCoins(i, j, dp, coins)
    if i > j then
        return 0
    end if
    if dp[i][j] ≠ −1 then
        return dp[i][j]
    end if
    leftPick ← coins[i] + min(MaxCoins(i + 2, j, dp, coins), MaxCoins(i + 1, j − 1, dp, coins))
    rightPick ← coins[j] + min(MaxCoins(i + 1, j − 1, dp, coins), MaxCoins(i, j − 2, dp, coins))
    dp[i][j] ← max(leftPick, rightPick)
    return dp[i][j]
end function
function solvePotsOfGold(coins)
    n ← length(coins)
    dp ← 2D array of size n × n, initialized to −1
    return MaxCoins(0, n-1, dp, coins)
end function
```

7 Kết luận

Bài toán Pots of Gold thể hiện rõ cách một trò chơi hai người tưởng chừng đơn giản lại có thể dẫn đến độ phức tạp rất lớn nếu chỉ dùng đệ quy thông thường. Nhờ **Dynamic Programming**, ta có thể tìm được lời giải trong thời gian đa thức, và với kỹ thuật lập chỉ mục khéo léo, bộ nhớ có thể giảm từ $O(n^2)$ xuống $O(n)$. Đây là ví dụ sinh động về **Optimal Strategy** trong lý thuyết trò chơi (Game Theory), đồng thời minh họa tầm quan trọng của dynamic programming trong các tình huống cạnh tranh hai người.

Tài liệu tham khảo

- Mark A. Weiss, 3rded., 2005. *Data Structures and Algorithm Analysis in C++*
- *https://www.techiedelight.com/pots-gold-game-dynamic-programming*