Books

Blue 11 - Lecture 02: Algorithmic Complexity

Tóm Tắt Đề Bài

Tóm Tắt Đề Bài

Dữ kiện:

- Cho một giá sách gồm N cuốn sách nằm liên tiếp nhau và thời gian rảnh T.
- Cuốn sách thứ i sẽ tiêu tốn A[i] thời gian để đọc.
- Quy tắc đọc:
  - Chọn một cuốn sách bất kì trên giá sách.
  - Đọc sách liên tục từ trái sang phải kể từ vị trí đã chọn sao cho sử dụng hết thời gian rảnh T.

Yêu cầu:

- Hãy tìm số lượng cuốn sách được đọc nhiều nhất.

Giải Thích Ví Dụ

Ví Dụ 1

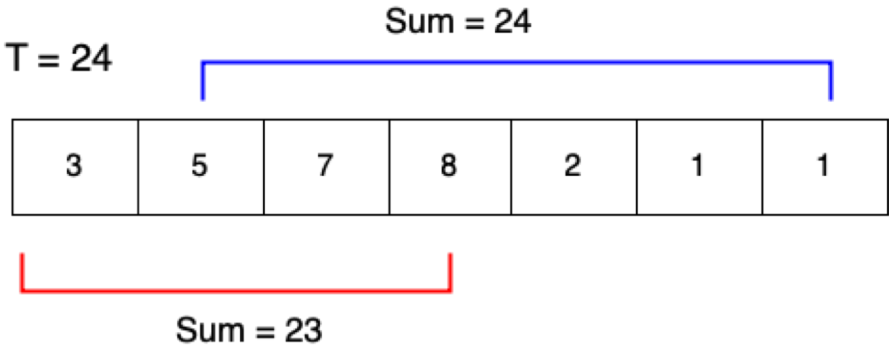
Input	Output	Giải thích ví dụ
N = 4, T = 5 A = [3, 1, 2, 1]	3	Bắt đầu đọc tại vị trí 1: [3, 1] Bắt đầu đọc tại vị trí 2: [1, 2, 1] Bắt đầu đọc tại vị trí 3: [2, 1] Bắt đầu đọc tại vị trí 4: [1] Ta chọn trường hợp đọc tại vị trí 2 cho kết quả độ dài lớn nhất là 3.

Ví Dụ 2

Input	Output	Giải thích ví dụ
N = 3, T = 3 A = [2, 2, 3]	1	Bắt đầu đọc tại vị trí 1: [2] Bắt đầu đọc tại vị trí 2: [2] Bắt đầu đọc tại vị trí 3: [3] Ta chọn trường hợp đọc tại vị trí bất kì đều cho kết quả độ dài lớn nhất là 1

Hướng Dẫn Giải

Cách 1



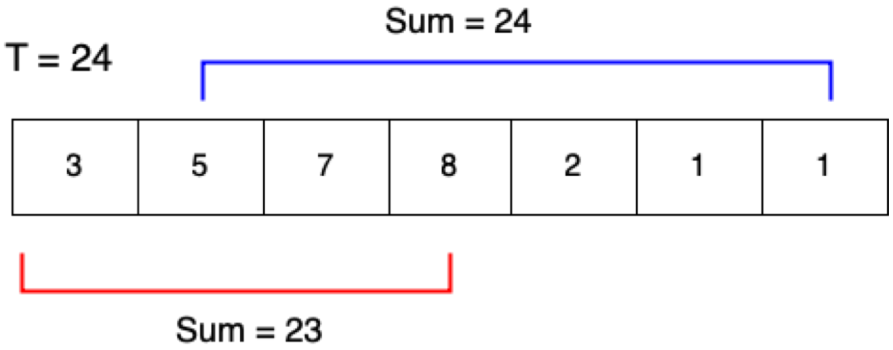
- Duyệt cố định vị trí bắt đầu **i** và kết thúc **j**:
  - Tính tổng **sum** tất cả phần tử trong đoạn **[i, j]**
  - Nếu **sum <= T**, ta cập nhật **best = max(best, j - i + 1)**

Cách 1

```
for i:= 0 to N-1 do:
    for j:= i to N - 1 do:
        sum := 0
        for k:= i to j do:
            sum += a[k]
        if (sum <= T):
            best = max(best, j - i + 1)
```

Độ phức tạp: **O(N^3)**
Không đáp ứng được yêu cầu thời gian của đề bài

Cách 2



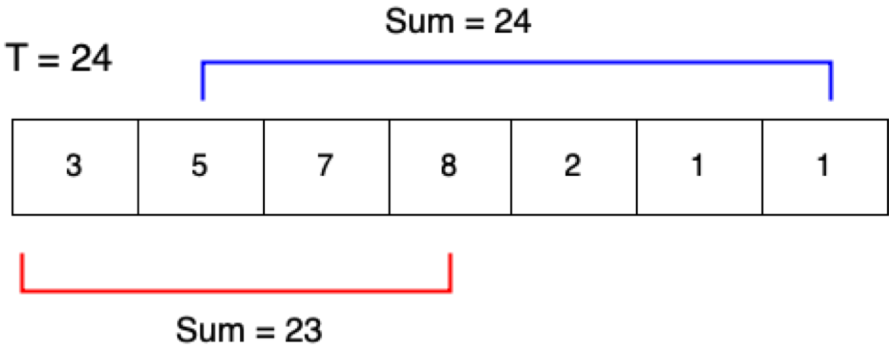
- Với mỗi phần tử thứ **i**:
  - Ta cần tìm phần tử thứ **j lớn nhất** sao cho $\sum A[i..j] \leq T$
  - Cập nhật **best = max(best, j - i + 1)**

Cách 2

```
for i:= 0 to N-1 do:
    sum = 0
    for j:= i to N - 1 do:
        sum += A[j]
        if (sum <= T):
            best = max(best, j - i + 1)
        else break
```

Độ phức tạp: **O(N^2)**
Không đáp ứng được yêu cầu thời gian của đề bài

Cách Tối Ưu

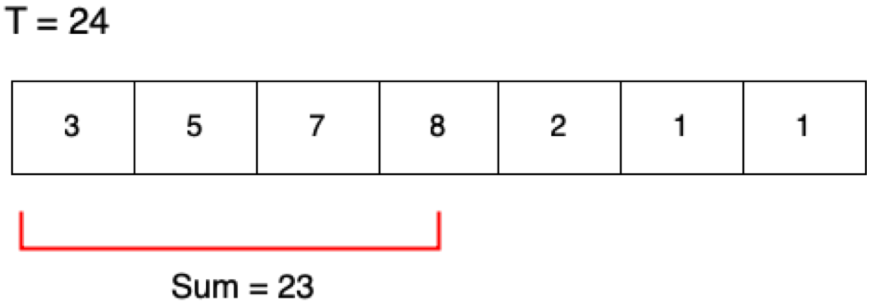


- Cải tiến:
  - Giả sử ta đã tính được **(i, j)** thoả mãn điều kiện, vậy kết quả bài toán của vị trí **i + 1** thì **j** ảnh hưởng như thế nào ? Ta tạm gọi vị trí kết quả đó là **j'**

$$\sum A[i+1\ ..j'] = \sum A[i\ \ldots\ j] - A[i] + \sum A[j+1 \ldots\ j']$$

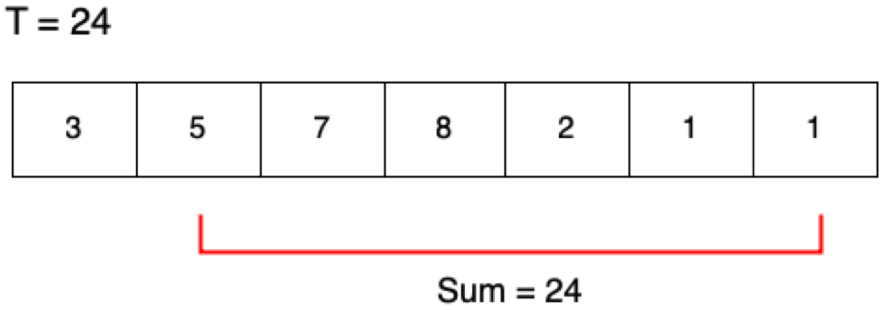
  - Ta cần **lưu lại biến vị trí kết quả của bài toán** trước để giảm thiểu số lần phải lặp để tính vị trí kết quả hiện tại

Cách Tối Ưu



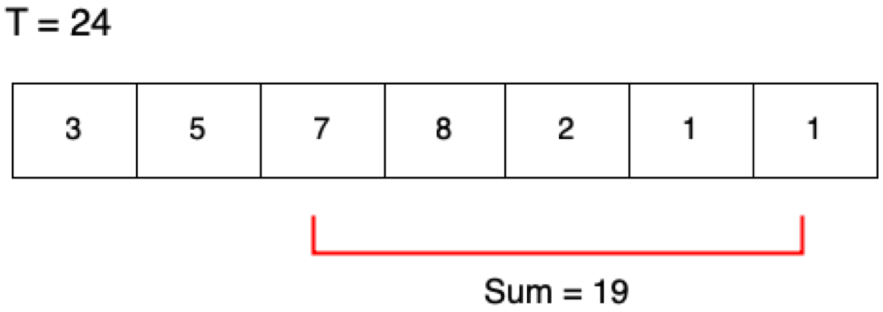
- Tại vị trí i = 1, ta sẽ tìm ra được vị trí j = 4 lớn nhất sao cho Sum = 23 < T = 24

Cách Tối Ưu



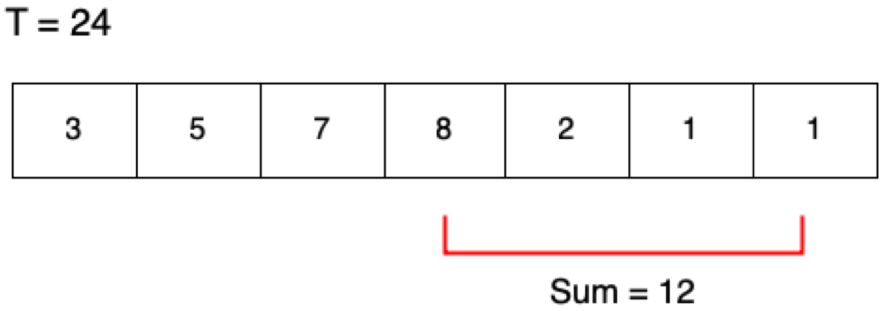
- Tại vị trí i = 1, ta sẽ tìm ra được vị trí j = 4 lớn nhất sao cho Sum = 23 < T
- Tại vị trí i = 2, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 24 <= T

Cách Tối Ưu



- Tại vị trí i = 1, ta sẽ tìm ra được vị trí j = 4 lớn nhất sao cho Sum = 23 < T
- Tại vị trí i = 2, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 24 <= T
- Tại vị trí i = 3, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 19 < T

Cách Tối Ưu



- Tại vị trí i = 1, ta sẽ tìm ra được vị trí j = 4 lớn nhất sao cho Sum = 23 < T
- Tại vị trí i = 2, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 24 <= T
- Tại vị trí i = 3, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 19 < T
- Tại vị trí i = 4, ta sẽ tìm ra được vị trí j = 7 lớn nhất sao cho Sum = 12 < T

Thuật Toán

- Bước 1: Đọc N, T và mảng A.
- Bước 2: Khởi tạo **right** = 0, sum = 0, best = 0
- Bước 3: Duyệt **left** = 0 tới **N - 1**
  - Ta cần tìm vị trí **right'** xa nhất sao cho **sum[left..right') <= T** bằng cách tịnh tiến **right** lên một lượng đơn vị.
  - Cập nhật **best = max(right - left, best)**
  - Chuẩn bị sang vị trí kế tiếp, ta cần trừ **sum** đi một lượng **A[left]**
- Bước 4: Đưa ra kết quả **best**

Độ phức tạp: **O(N)**

Mã Giả

Pseudo Code

```
read(N, T, A)
A[N] = INF

right = 0, sum = 0, maxLength = 0
for left := 0 to N - 1 do :
    while (right <= N) and (sum + A[right] <= T):
        sum += A[right]
        right += 1
    maxLength = max(maxLength, right - left)
    sum -= A[left]

write(maxLength)
```

Thank you