Kỹ thuật Two-pointer

Kỹ thuật Two-pointer là gì?

Two-pointer, như tên gọi của nó, là một kỹ thuật sử dụng hai con trỏ nhằm xử lý các yêu cầu bài toán trên một mảng. Bởi vì Two-pointer là một *phương pháp*, thay vì một *thuật toán*, cho nên Two-pointer sẽ không có một "khung" nhất định, áp dụng cứng cho mỗi lần ta sử dụng phương pháp. Thay vào đó, ta chỉ có là một tư tưởng chung, và tùy vào từng bài toán mà ta sẽ triển khai nó như thế nào.

Tư tưởng chung: Two-pointer là một phương pháp nhằm giảm thiểu độ phức tạp (một cách rất lớn) cho một bài toán yêu cầu người làm bài phải tìm ra một đoạn con liên tiếp thỏa mãn một hoặc một số điều kiện nhất định trên một dãy dữ liệu (như là mảng, chuỗi hay danh sách liên kết, ...) được cho trước.

Bài *Books*

Tóm tắt đề

Với bài này, đề yêu cầu chúng ta phải chọn ra một đoạn nhiều quyển sách liên tiếp nhất, sao cho tổng thời gian đọc các quyển sách đó không được vượt quá một giới hạn thời gian cho trước.

Như vậy, điều đầu tiên ta có thể thấy là ta phải chọn một đoạn liên tiếp. Do đó, nếu ta chọn một cách ngắt quãng thì bài chúng ta sẽ không đúng với yêu cầu đề.

Hướng giải quyết "ngây thơ"

Với hướng giải quyết này, chúng ta sẽ cần dùng hai vòng lặp lồng nhau. Vòng lặp đầu tiên bên ngoài sẽ là vòng lặp dùng để chọn quyển sách mà chúng ta sẽ bắt đầu đọc. Vòng lặp thứ hai bên trong sẽ là vòng lặp dùng để thử đọc các cuốn sách.

Như vậy, ta có thể có đoạn code như sau:

C++:

```cpp
int max_book = 0;

for (int start_book = 0; start_book < N; start_book++) {
    int read_time = 0;
    for (int end_book = start_book; end_book < N; end_book++) {
        read_time += A[end_book];
        if (read_time <= T) {
            max_book = max(max_book, end_book - start_book + 1);
        }
    }
}
```

Java:

```java
int maxBook = 0;

for (int startBook = 0; startBook < N; startBook++) {
    int readTime = 0;
```

```
    for (int endBook = startBook; endBook < N; endBook++) {
        readTime += A[endBook];
        if (readTime <= T) {
            maxBook = Math.max(maxBook, endBook - startBook + 1);
        }
    }
}
```

Python:

```
max_book = 0

for start_book in range(N):
    read_time = 0
    for end_book in range(start_book, N):
        read_time += A[end_book]
        if read_time <= T:
            max_book = max(max_book, end_book - start_book + 1)
```

Độ phức tạp thuật toán: $O(N^2)$ – với N là số lượng quyển sách.

Giải thích: Thấy rằng vòng lặp vòng lặp `end_book` (vòng lặp trong cùng) sẽ lặp $N - start_book$ lần. Bên cạnh đó, vòng lặp `start_book` sẽ lặp từ 0 đến $N - 1$. Vì vậy, tổng số lần lặp của ta sẽ là:

$N + (N-1) + (N-2) + ... = \frac{N \cdot (N+1)}{/2} = \frac{N^2+N}{2} = \frac{1}{2} \cdot (N^2 + N)$

Như vậy, ta được: $O(\frac{1}{2} \cdot (N^2 + N)) = O(N^2 + N) = O(N^2)$.

Nhìn lại yêu cầu đề bài, thấy rằng số lượng quyển sách có thể lên đến 10^5, thì độ phức tạp thời gian sẽ không thể nào chạy trong thời gian quy định là 2 giây. Vì thế, ta cần phải tìm cách giải quyết tốt hơn.

Sử dụng kỹ thuật Two-pointer

Quan sát: Giả sử ta chọn được một đoạn các quyển sách $A[start\_book\ ..\ end\_book]$. Thấy rằng nếu ta lấy thêm phần tử $end_book + 1$ mà tổng $A[start\_book\ ..\ end\_book + 1]$ lớn hơn giới hạn T thì theo cách giải quyết bên trên thì chúng ta sẽ phải đưa $start\_book$ lên 1 đơn vị và cho end_book chạy lại từ $start\_book$ đi lên.

Nhận xét: Thực sự việc cho end_book chạy lại kể từ $start\_book$ sau khi tăng $start\_book$ lên là một nước đi không cần thiết. Thay vào đó, ta có thể chỉ cần tăng $start\_book$ cho đến khi đoạn $A[start\_book\ ..\ end\_book]$ thỏa lại điều kiện bài toán là được rồi.

Chứng minh:

Giả sử tổng $A[start\_book\ ..\ end\_book]$ đã là tối đa, và nếu ta lấy thêm phần tử $end_book + 1$ thì tổng $A[start\_book\ ..\ end\_book + 1]$ sẽ lớn hơn T.

Đặt $current\_max = end\_book - start\_book + 1$, thấy rằng nếu ta tăng $start\_book$ lên 1 đơn vị thì end_book của chúng ta lúc này chắc chắn sẽ chạy được đến end_book lúc trước vì các giá trị trong mảng của chúng ta không có âm, cho nên nếu ta bỏ đi phần tử tại $start\_book$ thì nếu tổng $A[start\_book\ ..\ end\_book]$ đã nhỏ hơn hoặc bằng T rồi thì tổng $A[start\_book + 1\ ..\ end\_book]$ sẽ chắc chắn nhỏ hơn T.

Như vậy, việc mình đưa *end_book* về lại *start_book* và tăng lên sẽ khiến chúng ta sinh ra rất nhiều thao tác thừa. Thay vào đó, ta hãy giữ *end_book* tại vị trí cũ nếu tổng $A[start\_book \,..\, end\_book]$ bị lố T và cứ tăng *start_book* lên cho đến khi tổng $A[start\_book \,..\, end\_book] \leq T$ lại là được.

Code mẫu:

C++:

```
int max_book = 0;

int start_book = 0;
int read_time = 0;

for (int end_book = 0; end_book < N; end_book++) {
    // thử lấy end_book
    read_time += A[end_book];

    // nếu tổng từ start_book -> end_book bị lố T thì ta chỉ cần
    // loại start_book mà không cần thay đổi end_book
    while (read_time > T) {
        read_time -= A[start_book];
        start_book++;
    }

    // đến đây thì read_time chắc chắn <= T
    max_book = max(max_book, end_book - start_book + 1);
}
```

Java:

```
int maxBook = 0;

int startBook = 0;
int readTime = 0;

for (int endBook = 0; endBook < N; endBook++) {
    // thử lấy end_book
    readTime += A[endBook];

    // nếu tổng từ start_book -> end_book bị lố T thì ta chỉ cần
    // loại start_book mà không cần thay đổi end_book
    while (readTime > T) {
        readTime -= A[startBook];
        startBook++;
    }

    // đến đây thì read_time chắc chắn <= T
    maxBook = Math.max(maxBook, endBook - startBook + 1);
}
```

Python:

```
max_book = 0

start_book = 0
read_time = 0

for end_book in range(N):
    # thử lấy end_book
    read_time += A[end_book]

    # nếu tổng từ start_book -> end_book bị lố T thì ta chỉ cần
    # loại start_book mà không cần thay đổi end_book
    while read_time > T:
        read_time -= A[start_book]
        start_book += 1

    # đến đây thì read_time chắc chắn <= T
    max_book = max(max_book, end_book - start_book + 1)
```

Độ phức tạp thuật toán: $O(N)$ – với N là số lượng quyển sách.

Giải thích: Thấy rằng ta có hai vòng lặp $O(N)$ lồng nhau, nhưng cả hai vòng lặp này ta thấy rằng chúng chạy "độc lập" với nhau, không có vòng lặp nào phụ thuộc vào nhau cả, vì thể ta sẽ thấy hai biến *start_book* và *end_book* của chúng ta luôn luôn tăng, không bao giờ bị gán lại về một giá trị nhỏ hơn để sau đó chạy lên như cách làm cũ của chúng ta. Vì thế, độ phức tạp của chúng ta sẽ được giảm rất nhiều lần, về chỉ còn $O(2 \cdot N) = O(N)$.