BÁO CÁO THUẬT TOÁN DSA – TUẦN 2

Nguyễn Đình Trung Kiên – 24120195

Bài 1: Linear Search

- **Mô tả:** Sử dụng cách tìm kiếm tuần tự để tìm kiếm một phần tử K trong mảng các số nguyên được cho trước. Nếu tìm thấy, trả về chỉ số vị trí K xuất hiện đầu tiên trong mảng, ngược lại trả về -1 để biểu thị rằng K không tồn tại trong mảng.
- **Hướng giải quyết:**
  - Sử dụng vòng lặp for với biến đếm i để duyệt qua từng phần tử của mảng.
  - Ở mỗi lần duyệt qua phần tử thứ i, kiểm tra xem K có ở vị trí đó không:
    + Nếu có, trả về chỉ số i tại đó ngay lập tức và kết thúc hàm, K đã được tìm thấy.
    + Nếu không, tiếp tục duyệt mảng.
  - Khi đã duyệt hết phần tử mà hàm chưa kết thúc, đồng nghĩa với việc K không được tìm thấy dù mảng đã được duyệt toàn bộ. Khi đó, ta trả về -1.
- **Độ phức tạp:**
  - Ở trường hợp xấu nhất, ta phải duyệt toàn bộ mảng với n phần tử mới có thể tìm được K hoặc kết luận K không tồn tại trong mảng.
  - Vì vậy độ phức tạp của thuật toán là O(n).

Bài 2: Linear Search with Setinel.

- **Mô tả:** Tối ưu thuật toán tìm kiếm tuyến tính đơn thuần ở bài 1 bằng việc đặt phần tử lính canh vào cuối mảng.
- **Hướng giải quyết:**
  - Ý tưởng của thuật toán này là tiết kiệm chi phí một lần so sánh điều kiện của thuật toán cũ, đó là điều kiện i < n (xem xét biến đếm i đã được duyệt tới hết mảng hay chưa).
  - Ta thay thế phần tử K cần tìm với phần tử cuối mảng. Khi biến đếm duyệt mảng, chắc chắn sẽ dừng lại một cách hợp lệ chỉ với một điều kiện là kiểm tra xem đã duyệt đến phần tử K hay chưa. Tiếp đến, ta có thể kết luận dựa vào chỉ số của K có phải là phần tử cần tìm hay không.

- Một trường hợp khác ta có thể kết luận ngay từ đầu là khi phần tử K nằm ở cuối mảng (phần tử ta thay thế).

- **Độ phức tạp:**
  - Dù được tiết kiệm một điều kiện mỗi vòng lặp nhưng trường hợp xấu nhất vẫn là khi ta phải duyệt hết mảng đến khi gặp phần tử lính canh.
  - Do đó, thuật toán có độ phức tạp: O(n).

Bài 3:

- **Mô tả:** Một mảng số nguyên đôi một khác nhau gồm n phần tử được sắp xếp theo thứ tự tăng dần, sau đó được xoay với số lần từ 1 đến n. Trả về phần tử nhỏ nhất của mảng, sử dụng thuật toán với độ phức tạp O(logn).
- **Hướng giải quyết:**
  - Từ yêu cầu phải giải cùng độ phức tạp O(logn), ta nghĩ đến thuật toán Binary Search.
  - Ở bài toán này, dù mảng được sắp xếp đã bị xoay (không còn bảo toàn sự sắp xếp ban đầu) nhưng khi ta đã xác định được phần tử nào đó ở giữa mảng là thỏa điều kiện hay không thì ta có thể bỏ hết nửa còn lại và chỉ xét ở nửa tiếp theo như thuật toán Binary Search.
  - Một phần tử ở giữa mảng có thể:
    + Đang nằm trên nửa lớn hơn và là nửa đã bị xoay về phía đầu mảng.
    + Đang nằm trên nửa bé hơn (là nửa ban đầu nhưng sau khi xoay đã nằm ở cuối mảng). Ở trường hợp này, phần tử đó **cũng có thể** là phần tử bé nhất.
  - Để xác định được điều trên, ta so sánh với phần từ bên phải ngoài cùng (arr[right]). Bởi arr[right] là phần tử nối giữa phần đã bị xoay về đầu mảng và phần ban đầu bị đẩy về phía sau mảng.
    + Khi phần tử đang xét lớn hơn arr[right], phần tử đó đang nằm trong phần đã bị xoay, ta bỏ hết phía bên trái của phần tử đang xét, vì chắc chắn phần tử nhỏ nhất không thể nằm trong phần này.
    + Khi phần tử đang xét bé hơn arr[right], phần tử đó đang nằm trong phần mảng không bị xoay và bị đẩy về cuối mảng, ta rút gọn phần bên phải phần tử đó nhưng không bỏ phần tử đang xét, vì nó có thể là phần tử bé nhất.

- Đến khi đã rút gọn về chỉ còn một phần tử (left = right) thì đó chính là phần tử bé nhất mà ta cần tìm.

- **Độ phức tạp:** O(logn).

Bài 4:

- **Mô tả:** Từ một mảng số nguyên dương n phần tử là n các kiện hàng cần được giao đi trong *days* ngày. Tính tải trọng nhỏ nhất của tàu để thỏa mãn điều kiện trên.
- **Hướng giải quyết:**
  - Sử dụng Binary Search để thử lần lượt các tải trọng khả thi trong khoảng từ bé nhất đến lớn nhất:

    + Tải trọng của tàu ít nhất phải chở được kiện hàng nặng nhất.

    + Tải trọng lớn nhất có thể là tổng tất cả kiện hàng.

  - Từ ý tưởng của Binary Search, ta có được những tải trọng trong khoảng từ bé nhất đến nhỏ nhất. Ở mỗi tải trọng có được, ta thực hiện thuật toán kiểm tra xem có thể giao hàng đi trong số ngày ít hơn hoặc bằng số ngày quy định hay không:

    + Nếu thỏa điều kiện, ta tiếp tục giảm tải trọng tối đa đến mức ở tải trọng đang xét để tìm xem còn giá trị nào nhỏ hơn có thể thỏa mãn điều kiện không, không bỏ qua giá trị này vì có thể là tải trọng ta cần tìm.

    + Nếu không thỏa điều kiện, ta tăng tải trọng lên và bỏ qua hẳn giá trị đó. Bởi khi này tải trọng tối đa là quá thấp, lượng hàng được giao đi trong một ngày là không cao nên số ngày phải tăng lên cao hơn yêu cầu.

  - Tiếp tục thu hẹp khoảng tìm kiếm của thuật toán đến khi không còn phần tử nào trong khoảng thì phần tử nằm tại vị trí nhỏ nhất là phần tử cuối cùng cần tìm đến.
- **Độ phức tạp thuật toán:**
  - Ở mỗi bước Binary Search đến khi tìm được phần tử cuối cùng, ta liên tục chia đôi khoảng tìm kiếm theo thuật toán, có thể thấy ta sẽ duyệt $\log_2(\text{maxWeight} - \text{minWeight})$ lần.
  - Ở mỗi lần duyệt, ta lại phải duyệt cả mảng để kiểm tra xem điều kiện giao hàng trong số ngày quy định: duyệt n lần.

- Vậy suy ra, thuật toán có độ phức tạp O(nlog(maxWeight - minWeight)) cho toàn bộ thuật toán.

Bài 5:

- **Mô tả:** Từ một mảng các số nguyên dương n phần tử được cho trước cùng một số nguyên dương *target,* trả về chiều dài nhỏ nhất của một mảng con trong mảng được cung cấp có tổng các phần tử lớn hơn hoặc bằng *target.*
- **Hướng giải quyết:**
  - Ta sử dụng thuật toán Binary Search để tìm ra chiều dài một mảng con bé nhất thỏa điều kiện đề bài.
  - Ta thấy:
    + Chiều dài mảng con bé nhất là 1.
    + Chiều dài mảng con lớn nhất có thể là n.
  - Từ một khoảng tìm kiếm tăng dần như trên, ta sử dụng Binary Search để có được các chiều dài thử nghiệm để kiểm tra:
    + Với mỗi chiều dài thỏa điều kiện, ta có thể bỏ tất cả các giá trị lớn hơn nó và thu hẹp khoảng cách tìm kiếm.
    + Khi một chiều dài bé hơn không thỏa, ta cũng có thể bỏ hết tất cả những giá trị bé hơn và cả chính chiều dài không thỏa đó.
  - Duyệt liên tục thuật toán đến khi tìm thấy duy nhất một giá trị chiều dài thỏa điều kiện, cũng là giá trị bé nhất cần tìm.
  - Để kiểm tra điều kiện, ta sử dụng mảng cộng dồn được tạo lập từ mảng ban đầu, mỗi phần tử của mảng cộng dồn đó là tổng của tất cả phần tử từ vị nó của nó trở về trước trong mảng ban đầu. Với cách này, ta sẽ không cần phải duyệt tuyến tính để tìm ra tổng của một mảng con, trường hợp xấu nhất có thể phải duyệt hầu như toàn bộ mảng.
- **Độ phức tạp:**
  - Với thuật toán Binary Search: $\log_2 n$.
  - Với thuật toán kiểm tra: 2n.
  - Ta có độ phức tạp thuật toán trên là O(nlogn).

Bài 6:

- **Mô tả:** Từ một mảng n số dương được sắp xếp tăng dần và một số nguyên dương *target,* kiểm tra xem có cặp số nào trong mảng có tổng bằng số *target* ấy không.
- **Hướng giải quyết:**
  - Để tránh việc sử dụng một vòng lặp đôi với độ phức tạp $O(n^2)$ thì ta sử dụng thuật toán khác hiệu quả hơn.
  - Ta vẫn sẽ duyệt một vòng lặp từ đầu mảng đến khi con số duyệt tới lớn hơn một nửa số *target* vì chắc chắn phía sau đó không thể có số nào tổng với số đó mà bằng *target* được.
  - Với mỗi lần duyệt, ta sẽ tìm kiếm với Binary Search với khoảng tìm kiếm là từ sau số đó đến hết mảng. Ở mỗi lần duyệt, ta kiểm tra xem số đó có cộng với số ta có được từ vòng lặp và có tổng bằng với *target* hay không:
    + Nếu đã tìm thấy, ta trả về ngay lập tức.
    + Nếu không tìm thấy và tổng đó lớn hơn giá trị *target,* ta có thể thu hẹp phạm vi tìm kiếm về những số bé hơn và ngược lại.
  - Nếu khi đã tìm đến khi không còn phần tử nào để thử và vẫn chưa trả về, điều đó có nghĩa là không có cặp số nào tạo thành tổng *target* cần tìm.
- **Độ phức tạp:**
  - Duyệt một nửa mảng để có được một giá trị cơ sở cho phép thử ở Binary Search: n/2 + 1 lần.
  - Thuật toán Binary Search với mỗi giá trị cơ sở: $\log_2 n$.
  - Thuật toán có độ phức tạp là: O(nlogn).

Bài 7:

- **Mô tả:** Từ một mảng số nguyên cho trước, in ra màn hình các bộ ba phần tử trong mảng có tổng bằng 0, không in ra các bộ 3 trùng nhau và có thể in bộ 3 theo thứ tự tùy ý.
- **Hướng giải quyết:**
  - Cùng ý tưởng tìm đến một tổng cụ thể từ một mảng như bài trên, ta sử dụng thuật toán Binary Search để tránh việc phải lặp tới 3 vòng lặp lồng nhau.
  - Ta sử dụng 2 vòng lặp lồng nhau để tìm ra đôi 2 phần tử cơ sở cho phép thử. Ở vòng lặp này, ta sẽ loại bỏ các bộ 3 trùng nhau bằng cách không xét đến các giá trị trùng nhau ở vòng lặp ngoài. Bởi khi đã có một phần

tử trong bộ ba không trùng thì 2 phần tử còn lại cũng sẽ không trùng vì cả 3 phụ thuộc vào nhau.

- Từ tổng của bộ đôi có được từ vòng lặp lồng nhau, ta sử dụng Binary Search để tìm một phần tử tạo thành tổng 0 với 2 phần tử trên. Khoảng tìm kiếm sẽ từ sau phần tử ở vòng lặp 2 đến cuối mảng, vì chắc chắn các phần tử trước đó đã được xét đến ở các lần lặp trước đó.
- Nếu phần tử đang xét thỏa điều kiện tạo thành tổng 0, ta trả về ngay. Nếu không, thu hẹp về các phần tử bé hơn nếu tổng vẫn lớn hơn 0 và ngược lại.

- **Độ phức tạp:**
  - Sử dụng vòng lặp lồng nhau: n^2.
  - Duyệt phần mảng còn lại bằng Binary Search để tìm đến phần tử tạo thành tổng 0: $\log_2 n$.
  - Độ phức tạp của thuật toán là: $O(n^2 \log n)$.