Selection Sort

Table of contents

Idea

Lặp qua các vị trí từ 0 đến n - 1. Ở mỗi lần lặp i tìm phần tử cực trị (nhỏ nhất hoặc lớn nhất) trong khoảng từ i đến n - 1. Sau đó thay thế phần tử a_i với phần tử cực trị vừa tìm. Trong trường hợp xếp mảng tăng dần thì phần tử cực trị là phần tử nhỏ nhất.

Properties

Điểm thú vị của selection là có thể sort trong khoảng từ 0 đến k (k < n). Ví dụ khi tuyển sinh, ta chỉ tuyển 500 học sinh đầu trong 1000 học sinh. Chúng ta sắp xếp theo tên của các học sinh, khi sắp xếp đến số lượng 500 thì ngừng, không cần duyệt qua toàn bộ dữ liệu đầu vào. Vậy Selection Sort nó có thể ngừng sort tại vị trí mong muốn nào đó trong quá trình sort, giúp tiết kiệm chi phí.

Điểm mạnh Dễ hiểu, dễ cài đặt, dùng trong **(prototype?)**. Tức là khi mình build một hàm lớn, cần sắp xếp dữ liệu, chúng ta chọn Selection Sort để làm điều đó. Sau này, khi chúng ta cần tối ưu hóa thời gian thực hiện thì thay thế bằng các thuật toán khác mà không làm ảnh hưởng đến kết quả xử lý.

Đồng thời selection sort còn làm việc rất tốt với các mảng dữ liệu nhỏ và phân bố ngẫu nhiên. Là một thuật toán inplace tiết kiệm bộ nhớ.

Điểm yếu Mảng dữ liệu lớn và phân bố có gần như có thứ tự sẽ làm cho thuật toán chạy chậm và không hiệu quả.

> Khi nào nên sử dụng?

- Khi danh sách đầu vào là nhỏ
- Bộ nhớ bị giới hạn

Complexity Analysis

Tham khảo opengenus.

Lúc ban đầu mảng có hai phần, phần được sắp xếp và phần chưa sắp xếp, gọi kích thước của chúng lần lượt là S1 và S2. Ở mỗi lần lặp, kích thước S1 tăng lên 1 và S2 giảm đi 1. Do đó S2 = n - S1.

Độ phức tạp thời gian cho việc tìm cực trị của mảng chưa sắp xếp có kích thước S2 là $O(S2)$. Với vòng lặp i bất kỳ, S1 sẽ là i - 1, do đó S2 = n - S1 = n - i + 1.

Như vậy độ phức tạp của việc tìm phần tử cực trị sẽ là $O(n-i+1)$ và $O(1)$ cho công việc hoán vị.

Tổng độ phức tạp sẽ là

$$\sum_{i=1}^{n+1} O(n-i+1) + \sum_{i=1}^{n+1} O(1)$$

Do ở bước i thì S1 = i - 1, mà ta lặp đến khi nào S1 = n nên sẽ có n + 1 bước.

Ta có

$$\sum_{i=1}^{n+1} O(1) = 1+1+1+\ldots = n+1 = O(n)$$

Và

$$\sum_{i=1}^{n+1} O(n-i+1) = n+(n-1)+\ldots+1+0 \ = 1+2+\ldots+n = \frac{n(n+1)}{2} = O(n^2)$$

Do quy luật áp đảo mà độ phức tạp tổng cộng sẽ là $O(n^2)$.

Worst case Giả sử mảng đã sắp có thứ tự $a_1, a_2, \ldots a_n$. Và đẩy a_1 về cuối, ta có mảng $a_2, a_3, \ldots, a_n, a_1$. Lúc này thuật toán sẽ rơi vào trường hợp xấu nhất khi mà chỉ xảy ra đúng một lần hoán vị. Worst case có:

$\frac{n(n\ +\ 1)}{2}$ số lần so sánh.

n lần hoán vị trong tổng số các vòng lặp.

Do đó mà độ phức tạp là $O(n^2)$.

Best case Trường hợp tốt nhất có thể xem như không xảy ra bất kỳ sự hoán vị nào, tức là khi mảng đã được sắp xếp. Best case có:

$\frac{n(n+1)}{2}$ số lần so sánh.

0 lần hoán vị trong tổng số các lần lặp.

Average case Phân tích chi tiết độ phức tạp trung bình ở đây.

Complexity

Cases	Complexity
Best case	$O(n^2)$
Worst case	$O(n^2)$
Average case	$O(n^2)$

Space Complexity: $O(1)$.

Code

```
void selectionSort(int *a, int n)
{
    // Phần tử kế cuối đã tự sắp xếp
    for(int i = 0;i < n - 1; i++)
    {
        // Tìm phần tử nhỏ nhất trong mảng chưa sắp xếp
        int min = i;
        for(int j = i + 1; j < n; j++)
        {
            if(a[j] < a[min])

                min = j;
        }
        swap(a[i], a[min]);
    }
}
```