Flash Sort

Flash Sort là một thuật toán sắp xếp dựa trên sự phân bố của dữ liệu, cụ thể là phân bố đều, được Karl-Dietrich Neubert phát minh năm 1998. Nếu chúng ta không xác định được phân bố của một tập dữ liệu, khi dữ liệu có khả năng phân bố không đều, thì độ phức tạp có thể trở thành O(n2).

Idea

Thuật toán Flash Sort gồm ba công việc: phân lớp, hoán vị và sắp xếp cục bộ.

- Phân lớp giúp xác định kích thước của từng lớp phần tử.
- Hoán vị giúp hoán vị các phần tử giữa các lớp này.
- Sắp xếp cục bộ để sắp xếp các phần tử trong cùng một lớp.

Idea chung của Flash Sort là chia để trị, tương tự như Merge Sort hay Quick Sort. Thay vì chia thành hai mảng con đến khi nào không chia được nữa thì Flash Sort chia thành m phân lớp.

Classification

Giả sử các phần tử trong danh sách là `a[i]` và được phân bố đều rải rác. Chúng ta cần tìm vị trí của phần tử `a[i]` trong một lớp dữ liệu. Việc tính toán này có thể tính dựa trên giá trị của phần tử đó mà không cần thực hiện các phép so sánh.

Đầu tiên, ta đi tìm hệ số `m` đại diện cho số phân lớp. Sử dụng công thức sau với `n` là số phần tử và `c` là một hằng số tùy chỉnh.

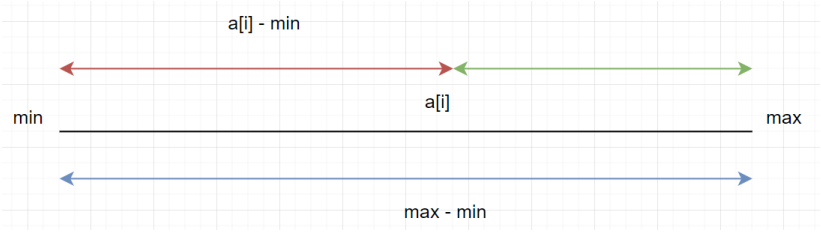
$$m = C * n$$

- Độ phức tạp thời gian và không gian: $O(1)$.

Tiếp theo tìm giá trị nhỏ nhất `min` và lớn nhất `max`.

- Độ phức tạp thời gian $O(n)$ và không gian $O(1)$.

Sau đó ta đi tìm phân lớp của phần tử `a[i]` bất kỳ. Có biểu đồ sau:



Ta lấy đoạn màu đỏ chia cho đoạn màu xanh dương được

$$\frac{a[i] - min}{max - min}$$

Sau đó nhân biểu thức trên cho `m - 1` với `m` là số lượng phân lớp, lý do trừ đi 1 là vì các phân lớp bắt đầu từ 0. Suy ra được công thức tính phân lớp `k` cho phần tử `a[i]` bất kỳ:

$$k(a[i]) = int\left[\frac{(m-1)(a[i] - min)}{max - min}\right]$$

Kết quả sẽ có giá trị từ 0 đến m. Trung bình sẽ có `n/m` phần tử ở mỗi lớp.

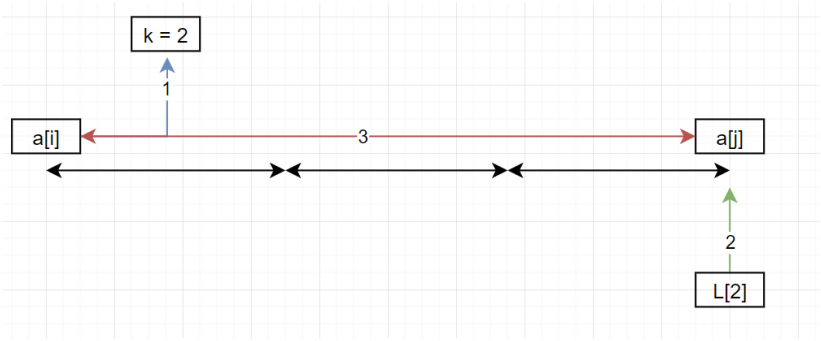
- Do cần phải tìm phân lớp cho tất cả các phần tử nên độ phức tạp thời gian là $O(n)$, độ phức tạp không gian là $O(1)$.

Ta cũng tiến hành đếm số lượng phần tử mỗi phân lớp trong lúc tìm phân lớp của phần tử `a[i]`. Khi đã có được số lượng phần tử này rồi thì ta cần tính vị trí cuối cùng của từng phân lớp (tính tích lũy tương tự Counting Sort). Vị trí cuối cùng này sẽ được lưu trong mảng phụ `L[m]`. Việc tính tích lũy này sẽ phục vụ cho việc chèn phần tử `a[i]` nằm sai phân lớp về đúng vị trí phân lớp của nó do `L[m]` quy định.

- Độ phức tạp thời gian $O(m)$, độ phức tạp không gian $O(m)$.

Permutation

Bước này là bước quan trọng của thuật toán, nó sẽ kiểm tra giá trị `k` của phần tử `a[i]` bất kỳ. Nếu đúng chỗ thì xét tiếp phần tử tiếp theo, sai chỗ thì sẽ hoán vị với phần tử đang nằm ở phân lớp `k` mà nó đáng lẽ phải ở đó.



Giả sử trong hình minh họa trên, giá trị phân lớp `k` của `a[i]` là 2 (Bước 1). Ta tìm vị trí mà phân lớp 2 cho phép hoán vị là `L[2]` (Bước 2). `L[2]` đang trỏ đến `a[j]`. Ta sẽ hoán vị `a[i]` và `a[j]` (Bước 3).