Bài tập thực hành tuần 1 - IT003.L21.KHTN

Sinh viên thực hiện: Nguyễn Duy Đạt - 20520435

Lưu ý: Trong các bài tập này các bạn sẽ sử dụng các cấu trúc dữ liệu tuyến tính để giải quyết. Các bạn sử dụng thư viện được cung cấp sẵn trong C++ nếu có thể (mảng tĩnh `array`, mảng động `vector`, các thư viện hỗ trợ sắp xếp `sort, stable_sort, partital_sort`, các thư viện hỗ trợ tìm kiếm `binary_search`, bảng băm). Các bạn chỉ nên trình bày ở dạng ý tưởng vắn tắt bao gồm sử dụng cấu trúc dữ liệu gì và thực hiện các bước nào.

Bài tập 1: Sắp xếp mảng gồm N phần tử. Trong đó, $\uparrow$ và $\downarrow$ lần lượt ký hiệu cho việc sắp xếp theo thứ tự tăng dần và giảm dần.

1. N các bộ dữ liệu có cấu trúc (`integer: tuổi` $\uparrow$, `string: tên` $\downarrow$, `string: họ` $\uparrow$), trong đó, biến dữ liệu nào đứng trước ưu tiên sắp xếp trước.
2. N các phân số ($\frac{\texttt{tử số}}{\texttt{mẫu số}}$) $\uparrow$.

Lời giải:
1.1
Bước 1: Sử dụng thư viện algorithm, sử dụng struct với các biến tuổi, tên, họ.
Bước 2: Viết hàm comp(kiểu bool) so sánh 2 đối tượng với 3 thuộc tính ở trong struct, dùng câu lệnh if else với thứ tự so sánh lần lượt là tuổi, tên, họ.
Bước 3: Nhập n bộ dữ liệu, sử dụng sort(a,a+n,comp) nếu sắp xếp $\uparrow$ và sort(a,a+n, greater<kiểu dữ liệu>()) nếu $\downarrow$ với a là mảng chứa 3 thuộc tính của đối tượng.
1.2
Bước 1: Sử dụng thư viện algorithm, sử dụng struct với các biến tử số, mẫu số(Tu,Mau).
Bước 2: Dùng Bubble Sort, 2 vòng for lồng nhau : vòng for đầu tiên chạy từ 0 đến n-1, vòng for thứ hai chạy từ n-1 đến i.
Bước 3: Kiểm tra nếu (a[j].Tu/a[j].Mau)<(a[j-1].Tu/a[j-1].Mau) thì đổi vị trí với a là mảng nhập vào giá trị tử số, mẫu số.

Bài tập 2: Xét thuật toán Quick Sort các số nguyên theo thứ tự tăng dần. Chúng ta quy ước rằng thuật toán phân hoạch với chốt p của thuật toán Quick Sort đặt các phần tử $< p$ về phía bên trái và $\geq p$ về phía bên phải. Lưu ý rằng các phần tử bằng p luôn đặt về phía bên phải. Hãy đưa ra một trường hợp sao cho thuật toán Quick Sort sử dụng thuật toán phân hoạch như vậy thực thi với thời gian $O(n^2)$, thậm chí ngay cả khi sử dụng thuật toán chọn chốt p ngẫu nhiên. Cuối cùng, hãy đưa ra một phương án để giải quyết vấn đề này.
Lời giải:
- Trường hợp thuật toán Quick Sort thực thi với thời gian $O(n^2)$: Đó là khi quá trình phân vùng luôn chọn phần tử lớn nhất hoặc nhỏ nhất làm chốt. Xét trường hợp trên, trong đó phần tử cuối cùng được chọn làm chốt, trường hợp xấu nhất xảy ra khi mảng đã được sắp xếp ở thứ tự tăng hoặc giảm.
- Phương án giải quyết: Kết hợp Insertion Sort khi mảng con có 20-25 phần tử, có xu hướng nhanh hơn khi xử lí các danh sách nhỏ.

Bài tập 3: Giả sử rằng bạn được cho một mảng S chưa được sắp xếp gồm n số nguyên 32-bit. Giải quyết các bài toán con bên dưới với thuật toán tốt nhất mà bạn có thể nghĩ và phân tích độ phức tạp của thuật toán. Chúng ta có ràng buộc của bài toán như sau $1 \leq n \leq 10^5$.

1. Đưa ra các giá trị xuất hiện nhiều nhiều hơn một lần theo thứ tự tăng dần.
2. Tìm hai số nguyên $a, b \in S$ sao cho $a + b = v$ với v là số nguyên cho trước.

3. Tìm hai số nguyên $a, b \in S$ sao cho $a + b = v$ với v là số nguyên cho trước với giả định rằng mảng S đã được sắp xếp.

4. Đưa ra các số nguyên trong mảng S có giá trị nằm trong $[a..b]$ với hai số nguyên $a \neq b$. Kết quả đưa ra theo thứ tự tăng dần.

5. Tìm giá trị trung vị của mảng S. Giả sử rằng n là số lẻ.

6. Tìm phần tử xuất hiện nhiều nhiều hơn $\frac{n}{2}$ lần trong mảng S.

Lời giải:
3.1
Bước 1: Sử dụng Hashing, lưu trữ các phần tử của S và số lượng của chúng trong 1 bảng băm.
Bước 2: Sau khi lưu trữ số đếm, duyệt lại mảng S và xuất ra các phần tử xuất hiện nhiều hơn 1 lần.
Bước 3: Tạo mảng để lưu các phần tử được xuất ra ở bước 2 và dùng sort sắp xếp ↑.
Độ phức tạp của Hashing : $O(n)$
3.2
Bước 1: Sử dụng Hàm Băm, dùng hashmap để kiểm tra giá trị a của mảng S, nếu có tồn tại 1 giá trị v-a mà khi thêm vào giá trị trước đó ta được v.
Bước 2: Khởi tạo bảng băm trống S.
Bước 3: Thực hiện xét s[i] trong S, nếu s[v-s[i]] được đặt thì in ra cặp số thỏa mãn (a,b) với a tương ứng s[i].
Bước 4: Chèn s[i] vào S.
Độ phức tạp : $O(n)$
3.3
Bước 1: Dùng lặp for i từ 1 đến n, gọi a là s[i] thuộc S.
Bước 2: Chặt nhị phân- tìm phần tử v-s[i] (tức là b) có trong mảng S hay không?.Nếu có thì in ra cặp (a,b) thỏa mãn.
Độ phức tạp : $O(nlog(n))$
3.4
Bước 1: Sử dụng thư viện algorithm, sort các phần tử trong mảng S ↑ .
Bước 2: Duyệt từng phần tử s[i], nếu s[i]≥a và b≥s[i] thì xuất ra. Sau cùng ta được 1 dãy số nguyên thuộc mảng S thỏa mãn yêu cầu.
Độ phức tạp : $O(n)$
3.5
Bước 1: Sử dụng thư viện algorithm, sort các phần tử trong mảng S ↑ .
Bước 2: Vì n lẻ, n=2k+1 thì s[k+1] là số trung vị cần tìm.Suy ra, s[k+1]=s[(n+1)/2].
Bước 3: Duyệt từng phần tử s[i]. Kiểm tra nếu i = (n+1)/2 thì xuất s[i].
Độ phức tạp : $O(n)$
3.6
Bước 1: Sử dụng Hashing, lưu trữ các phần tử của S và số lượng của chúng trong 1 bảng băm.
Bước 2: Sau khi lưu trữ số đếm, duyệt lại mảng S và xuất ra các phần tử xuất hiện nhiều hơn n/2 lần.
Độ phức tạp của Hashing : $O(n)$

Bài tập 4: Giả sử rằng bạn được cho một mảng 2D gồm các số nguyên A có kích thước $n \times n$. Giải quyết các bài toán con bên dưới với thuật toán tốt nhất mà bạn có thể nghĩ và phân tích độ phức tạp của thuật toán. Chúng ta có ràng buộc của bài toán như sau $1 \leq n \leq 10^4$.

1. Xoay mảng 2D một góc 90 độ theo chiều kim đồng hồ.

2. Chuyển vị mảng 2D (đổi hàng và cột).

Lời giải:
4.1
Bước 1: Chuyển vị ma trận (hàng đổi cho cột), sử dụng 2 vòng for lồng nhau (i chạy từ 0 đến n, j

chạy từ 0 đến i) sau đó đổi chỗ 2 phần tử a[i][j] và a[j][i].
Bước 2: Lấy đối xứng ma trận ở bước 1 theo trục tung, sử dụng 2 vòng for lồng nhau (i chạy từ 0 đến n, j chạy từ 0 đến n/2) sau đó đổi chỗ 2 phần tử a[i][j] và a[i][n-i-1].
Độ phức tạp : $O(n^2)$
4.2
Bước 1: Sử dụng 2 vòng for lồng nhau (i chạy từ 0 đến n, j chạy từ 0 đến n).
Bước 2: Tạo mảng chuyển vị mới để lưu ở bước 1 B[i][j] =A[j][i]
.