IT003 - Cấu trúc dữ liệu và Giải thuật

Assignment 2
Method Report

Sinh viên : *Trần Nhật Khoa - 22520591*

THÀNH PHỐ HỒ CHÍ MINH

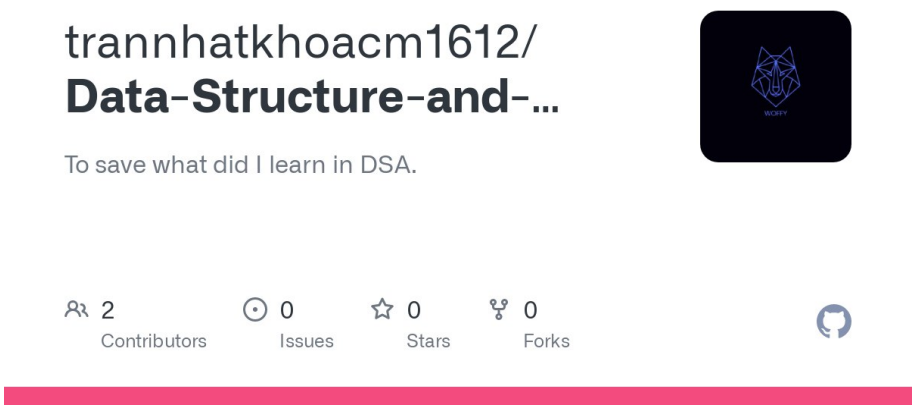
4 - 2023

Mục lục

1 Lời nói đầu

Thầy có thể chuyển nhanh đến mục mình muốn khi nhấp vào mục tương ứng trên **Mục lục** và em xin phép không để code ở trong báo cáo mà sẽ lưu nó ở trên trang Github, dưới đây là link em lưu tất cả những bài code mình đã code trên **Wecode**, ngoài ra ở mỗi mục sẽ có link riêng khi nhấp thầy có thể đến được trang chứa code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm`



2 Advance

2.1 Xếp hàng

Phát biểu lại bài toán: Cho một mảng số tự nhiên lưu các giá trị giá trị từ *1* đến *n*. Cho *m* lần gọi, mỗi lần sẽ gọi một số **x** có trong mảng, sau đó đổi vị trí của x lên vị trí đầu. Sau m lần gọi, xuất ra vị trí của các số trong mảng.Hãy in ra mảng **cuối cùng** sau khi đã thực hiện các thao tác trên.

Giải thuật: Ý tưởng của em chính là sử dụng một mảng **arr** để chứa các số được gọi(cụ thể là *m* số), số nào được gọi trước thì khi xuất ra sẽ ra sau, tương tự như cấu trúc **Stack**. Sau đó dùng một mảng kiểu *bool* có giá trị **True** để đại diện cho toàn bộ mảng, ngoài ra những số được gọi sẽ được đánh dấu là **False**. Tiếp theo ta sẽ ưu tiên xuất mảng **arr** theo thứ tự như đã nói, sau đó sẽ xuất những số còn lại theo thứ tự từ *1* đến *n*, số nào có đại diện trong mảng *bool* là **True** thì xuất ra.

Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/xephang.cpp`

2.2 Xếp hàng 2

Phát biểu lại bài toán: Cho một mảng số tự nhiên lưu các giá trị giá trị từ *1* đến *n*. Cho *m* lần gọi, mỗi lần sẽ gọi một số **x** có trong mảng, sau đó đổi vị trí của x lên vị trí đầu. Sau m lần gọi, xuất ra vị trí của các số trong mảng.Hãy in ra **phần tử cuối cùng** của mảng sau mỗi lần gọi.

Giải thuật: Khác với bài trước khi yêu cầu là in ra phần tử cuối cùng của mảng sau mỗi lần gọi thì ta không thể cải tiến hay sử dụng lại giải thuật ở trên. Ý tưởng của em là sử một mảng có độ dài $\mathbf{m + n}$ để lưu các giá trị ban đầu từ vị trí $\mathbf{m + 1}$ đến $\mathbf{m + n}$(index bắt đầu từ **1**). Mỗi lần gọi giá trị **x** ta sẽ thêm nó vào vị trí kế vị trí đầu, sau khi thêm như thế thì vị trí cũ của số đó ta sẽ gán giá trị là **0**. Khi đó ta chỉ cần ngược lại mảng, chỗ nào có giá trị **khác 0** thì xuất giá trị. Nhưng việc tìm kiếm giá trị của **x** trong mảng lại khá tốn thời gian, đòi hỏi ta phải tìm kiếm tuần tự(vì mảng bị xáo trộn). Do đó từ ban đầu em đã dùng một mảng để lưu lại **index** của từng số, sau đó chỉ cần truy suất đến khi muốn tìm.

Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-andAlgorithm/blob/main/Wecode/xephang2.cpp`

2.3 Kiểm kê 2

Phát biểu lại bài toán: Cho mảng số tự nhiên(có thể **trùng lặp**). Hãy xuất ra **tần suất xuất hiện** của các số trong mảng theo tự **tăng dần tần suất**, nếu **tần xuất bằng nhau** thì **giảm dần giá trị** của số tự nhiên đó.

Giải thuật: Ta sẽ sắp xếp mảng này theo thứ tự tăng dần(hoặc giảm dần) theo giá trị, khi đó những số có giá trị bằng nhau sẽ xếp kế nhau, ta sẽ đếm tần số từ đây. Ngoài ra em sẽ code lại hàm **HeapSort**(hôm đó em nghe thầy nói dị mà không biết sao lại dùng heap mà không sort, nếu được thì mong thầy giải thích dùm em với ạ) và do độ dài của một số tự nhiên có thể lên đến **100** nên em sẽ sử dụng chuỗi để lưu, khi đó cần cập nhật lại hàm so sánh hai **numstring** mà không dùng toán tử so sánh thông thường.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/Kiemke2.cpp`

2.4 Binary Search 2

Phát biểu lại bài toán: Cho một mảng **không được sắp xếp** và các truy vấn có dạng **x,y**. Với $\mathbf{x = 1}$ thì trả về vị trí đầu tiên mà **y** xuất hiện trong mảng, và nếu $\mathbf{x = 2}$ thì trả về vị trí cuối cùng mà **y** xuất hiện trong mảng.

Giải thuật: Ý tưởng của em là sẽ sắp xếp lại vị trí của các số trọng mảng và cũng lưu lại index khi thay đổi(bằng cách khi **swap** trong hàm **sort** thì ta cũng **swap** mảng **index**. Với mỗi truy vấn, em sẽ lấy ra hết tất cả những vị trí mà **y** xuất hiện, rồi truy suất thông qua mảng **index** theo yêu cầu của từng truy vấn.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/Binary_search_2.cpp`

3 Basic

3.1 POINT2D và POINT3D

Phát biểu lại bài toán: Cho một tập **N** điểm **(x,y) hoặc (x,y,z) với POINT3D** trên mặt phẳng Oxy hoặc Oxyz. Sắp xếp và in ra các điểm **tăng dần theo x**, nếu **x bằng nhau** thì sắp xếp các điểm **giảm dần theo y**, với **POINT3D** thì thêm ràng buộc là nếu **y bằng nhau** thì sắp xếp các điểm **tăng dần theo z**.

Giải thuật: Em sẽ tạo một **Struct POINT2D** với 2 atribute là tọa độ **x,y** và

Struct POINT3D với 3 atribute là tọa độ **x,y,z** , sau đó định nghĩa lại toán tử so sánh để tiện cho việc xắp xếp. Cuối cùng kết hợp toán tử đó ta code lại hàm **Quicksort** theo yêu cầu bài toán được dễ dàng hơn.
Link code:

POINT2D:
`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/POINT2D_template.cpp`
POINT3D:
`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/POINT3d.cpp`

3.2 Task

Tóm tắt lại bài toán: Bài toán yêu cầu tìm vị trí ngồi của Bob trong phòng thi sao cho Bob **cùng đề** với Alice và **gần nhất** với Alice. Phòng thi có n học sinh, các bàn xếp dọc thành một hàng, mỗi bàn 2 người ngồi, và có k đề khác nhau. Các đề được phát bắt đầu từ vị trí 1 của bàn 1 rồi đến vị trí 2 của bàn 1, tiếp theo là vị trí 1 của bàn 2, rồi tới vị trí 2 của bàn 2,...,lần lượt các đề 1, đề 2, đề 3, ..., cho tới đề k rồi lại quay lại đề 1, đề 2,... cho tới khi tất cả các học sinh đều nhận được đề.

Giải thuật: Ta sẽ chia ra hai trường hợp đó là số mã đề là số **chẵn** và là số **lẻ**, sau đó quét hết tất cả trường hợp có thể xảy ra. Em sẽ ví dụ với trường hợp số đề là số chẵn,: nếu **Alice ngồi ở vị trí 1** trên bàn thì Bob có thể ngồi vào bàn kế tiếp vị trí 2, Nnếu không được ngồi ở bàn kế tiếp thì chọn bàn phía trước của Alice và ngồi ở hàng bên phải của bàn đó,Nếu không được ngồi ở bàn phía trước của Alice thì trả về **false**; nếu Alice ngồi ở vị trí 2 trên bàn. và không khó để ta quét trường hợp của số đề lẻ.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/task.cpp`

3.3 Trộn 2 mảng

Phát biểu lại bài toán: Trộn 2 mảng có thứ tự không giảm thành một mảng có thứ tự không giảm.

Giải thuật: Ý tưởng của em là **duyệt qua cả hai mảng a và b đồng thời**, so sánh phần tử hiện tại của hai mảng và **đưa phần tử nhỏ hơn vào mảng kết quả c**. Ban đầu, ta khởi tạo con trỏ **indexB** và **indexC** với giá trị tương ứng là **-1** và **0**. Duyệt qua mảng a, với mỗi phần tử a[i], ta kiểm tra nếu có phần tử trong mảng b **nhỏ hơn hoặc bằng a[i]** thì ta đưa phần tử đó vào

mảng kết quả c. Con trỏ indexB sẽ chỉ vào **phần tử cuối cùng** trong mảng b được đưa vào mảng c. Sau khi duyệt xong mảng a, nếu trong mảng b còn phần tử chưa được đưa vào mảng kết quả c, ta đưa các phần tử đó vào mảng c.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/an_512.cpp`

3.4 Maxstring

Phát biểu lại bài toán: Bài toán yêu cầu tìm xâu số lớn nhất có thể được tạo ra bằng cách sắp xếp lại các chữ số trong xâu ban đầu và xóa đi một số ký tự sao cho xâu mới thu được là số chia hết cho 3.

Giải thuật: Ý tưởng của em đầu tiên là tính tổng các chữ số trong xâu **slock**. Dựa vào **phần dư** của tổng này khi chia cho 3, ta sẽ chọn một trong hai trường hợp: Nếu phần dư là 1 thì thuật toán sẽ lần lượt xóa khỏi xâu các ký tự mang giá trị mod 3 là 1, nếu chưa đủ để xóa được một số ký tự thì xóa lần lượt các ký tự mod 3 là 2 cho đến khi đủ số ký tự cần xóa, nếu phần dư là 2: thuật toán tương tự như trên nhưng đảo ngược trật tự xóa ký tự mod 3 là 1 và mod 3 là 2.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/vu33.cpp`

3.5 VQFlower

Tóm tắt lại bài toán: Cho một mảng gồm các số, ta sẽ đưa ra mảng sao cho số lượng phần tử khác biệt là nhiều nhất.

Giải thuật: Ta sẽ code lại thuật toán **Quicksort**, sau đó sắp xếp lại mảng rồi dựa vào đó tìm ra số lượng màu khác nhau, nếu chưa đủ vị trí k, thì ta thêm tiếp vào các số còn lại.
Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/VQ44_template.cpp`

3.6 Kiểm kê

Tóm tắt lại bài toán: Đơn giản là ta sẽ đếm số lượng phần tử **khác nhau** của mảng và trả về số lượng đó.

Giải thuật: Ý tưởng của em là sắp xếp lại mảng đó, và dựa vào đó để đếm.

Lưu ý ta cũng sẽ cập nhật lại cách so sánh chuỗi vì mã hàng có thể có độ dài rất lớn.

Link code:

`https://github.com/trannhatkhoacm1612/Data-Structure-and-Algorithm/blob/main/Wecode/kiemke.cpp`

– Hết –