

Bài A. CNTTREE

File dữ liệu vào: **stdin**
File kết quả: **stdout**
Hạn chế thời gian: 1 giây

tags: encode, hash, map

Mã hóa của cây gốc x được định nghĩa như sau:

- Mã hóa các nút con trực tiếp của x . Giả sử y_1, y_2, \dots, y_k là các nút con trực tiếp của x , đã sắp xếp theo thứ tự tăng dần mã của chúng
- Mã của x là $S_x = (S_{y_1} S_{y_2} \dots S_{y_k})$, tức là ghép mã của các con lại và thêm mở đóng ngoặc hai đầu

Khi đó hai cây đẳng cấu khi và chỉ khi cùng mã. Có thể hash hoặc map để đếm.

Bài B. SDINO

File dữ liệu vào: **stdin**
File kết quả: **stdout**
Hạn chế thời gian: 1 giây

tags: mergesort, quicksort, divide and conquer

Mỗi thuật toán sắp xếp có 1 tư tưởng cốt lõi. Ở bài này các thuật toán sắp xếp có sử dụng phép swap đều không khả thi vì rất khó để thiết kế phép đổi chỗ nhanh chóng. Một số thuật không dùng phép đổi chỗ là:

+Sắp xếp chọn:

- Dẩy toàn bộ lên Q
- Dùng lệnh HC để chuyển 1 số từ đầu Q xuống cuối Q
- Khi gặp phần tử lớn nhất thì chọn, cho xuống S
- đpt $O(n^2)$

+Sắp xếp chèn:

- Duy trì Q là dãy tăng
- Dùng HC để các phần tử trong Q chạy vòng tròn
- Chèn $S.\text{top}()$ vào vị trí thích hợp
- đpt $O(n^2)$

+Sắp xếp trộn: Linh hồn của sắp xếp trộn là phép trộn. Nếu thiết kế được phép trộn thì sẽ sắp xếp trộn được
-Thủ tục merge(n, m): Yêu cầu Q đang rỗng, n phần tử đầu của S tăng, m phần tử tiếp theo của S tăng. Nhiệm vụ là trộn $n + m$ phần tử đó thành dãy tăng và để vào đầu S , trả lại Q rỗng

- Ăn n lần C
- Trộn hai dãy, một dãy ở S và một dãy ở Q , kết quả trộn cho vào sau Q
- Ăn $n + m$ lần H, thu được S giảm
- Ăn $n + m$ lần C, ăn $n + m$ lần H, thu được S tăng
- đpt $O(n + m)$

-Thủ tục sort(n): Yêu cầu Q đang rỗng. Nhiệm vụ là sắp xếp lại n phần tử trên cùng của S và trả lại Q rỗng

- Đặt $a = n/2, b = n - a$
- sort(a)
- Ăn a lần C, ăn a lần H để đảo ngược a phần tử trên cùng
- Ăn n lần C, ăn n lần H để đảo ngược n phần tử trên cùng
- sort(b)
- merge(b, a)
- đpt $O(n \log n)$

+Sắp xếp nhanh: Linh hồn của sắp xếp nhanh là phép chia dãy làm 2 phần, một phần lớn hơn khóa pivot và một phần nhỏ hơn hoặc bằng.

- Đầu tiên là chọn khóa pivot, nên chọn trung vị để dãy được chia đôi
- Dẩy n phần tử lên Q
- Duyệt qua n phần tử một lượt, $<$ pivot thì cho xuống S , ngược lại thì cho ra sau Q
- Dưa $n/2$ phần tử trở lại S

*Cả hai thuật sắp xếp cuối đều có đpt $O(n \log n)$ nhưng cần tính chính xác hằng số cài đặt và tối ưu. Hằng số bằng 3 là AC

Bài C. CXTREE

File dữ liệu vào: **stdin**
File kết quả: **stdout**
Hạn chế thời gian: 1 giây

tags: math, combination

Gọi f_n là số cây n đỉnh thỏa mãn. Có định nghĩa 1, trong các nút kề 1 sẽ có 1 nhánh chứa đỉnh 2. Gọi m là số đỉnh trong nhánh con đó. Khi đó m chạy từ 1 đến $n - 1$ và trọng số cạnh là $m(n - m)$ phải không quá k . Số cây thỏa mãn TH này là: $C_{n-2}^{m-1} \times f_m \times m \times f_{n-m}$, vì ta chọn $m - 1$ đỉnh để đặt vào nhánh này, đám này có f_m cách sắp xếp, và chọn ra 1 đỉnh để nối với 1, đám còn lại có f_{n-m} cách xếp. Cho m chạy và tính được f_n

Bài D. DGRAPH

File dữ liệu vào: **stdin**
File kết quả: **stdout**
Hạn chế thời gian: 1 giây

tags: LCA, BIT, DFS

Dùng hai đối tượng S và T để quản lý hai tập: tập đỉnh phát và tập đỉnh thu. Mỗi khi có thêm một kết nối $a = (s, t)$, ta cần tính số kết nối làm nhiễu a và bị a làm nhiễu.

Gọi p_t là đỉnh nằm trước t trên $\text{Path}(s, t)$. Số kết nối làm nhiễu a chính là số đỉnh phát nằm cùng phía với p_t so với t . Để làm được điều này ta có thể sử dụng binary index tree để cài đặt đối tượng S , quản lý trên thứ tự duyệt cây.

Gọi $f(s)$ là số kết nối sẽ bị nhiễu nếu tại s có thêm một đỉnh phát. Lúc này $f(s)$ có thể được cập nhật lại sau mỗi thay đổi. Đối tượng T cũng chỉ cần cài đặt bằng binary index tree là đủ.