Bài 1. TYPING (7 điểm)

- **Subtask 1:**
  - Với câu hỏi loại 0, ta chỉ cần thử xem nếu bắt đầu bằng tay trái và tay phải thì sẽ phải đổi tay bao nhiêu lần rồi, chọn cách cần ít bước hơn. $O(N)$
- **Subtask 2:**
  - Với loại câu hỏi thứ nhất, ta vẫn có thể làm như subtask trước
  - Với loại câu hỏi thứ hai, nhận thấy, Tài phải đổi tay mỗi khi chuyển từ một chữ "A" sang chữ "I" hoặc ngược lại. Tạo mảng tổng dồn s_i là số lần mà tài phải chuyển tay ít nhất khi đi từ đầu xâu tới vị trí i. Đáp án của một đoạn $[l, r]$ chính là $s_r - s_l$, ta có thể duyệt r rồi lưu lại tổng những s_l thỏa mãn. $O(N)$
- **Subtask 3:**
  - Với việc có những chữ cái "T", ta sẽ không thể tính được như trên.
  - Xét xâu: *"TTTITTA"*
  - Ta nhận thấy rằng các xâu sau đều có số lần chuyển tay như nhau:
    - *"TTTITTA"*
    - *"TTITTA"*
    - *"TITTA"*
    - *"ITTA"*
  - Chính bởi vì những ký tự "T" trước "I" không làm ảnh hưởng đến kết quả. Vì vậy ta có thể bỏ hết những ký tự "T" ra khỏi xâu và lưu thêm với các ký tự còn lại một giá trị a_i có bao nhiêu ký tự "T" ngay liền trước vị trí i. Ví dụ a_A của xâu *"TTTITTA"* là bằng 2.
  - Bây giờ công thức của chúng ta khi tính đáp án cho một đoạn $[l, r]$ chính là
    - $(a_l + 1) * (s_r - s_l)$
    - $a_l * s_r + s_r - s_l$
  - Sử dụng tổng dồn ta có thể duyệt qua các vị trí r và tính được đáp án. $O(N)$

BÀI 2. BIENDOI (7 điểm)

- **Subtask 1:**
  - Code theo mô tả đề bài. $O(M * Q)$
- **Subtask 2:**
  - Lúc này không có truy vấn đảo ngược, nhưng truy vấn thứ 3 sẽ là trở ngại vì có thể mất $O(M)$ mỗi lần thực hiện.
  - Có hai cách để tối ưu đoạn này:
    - Đầu tiên là cách phổ thông hơn, ta sẽ lưu một cây Segment Tree với mỗi ngăn sách. Cây này của chúng ta cần hỗ trợ 3 thao tác: thêm, bớt và nghịch đảo. Hai thao tác đầu là thao tác cơ bản, còn thao tác thứ 3, ta có thể lưu thêm ở mỗi đỉnh của Segment Tree một giá trị $flip$ là hiện nút nay đang bị đảo hay không. $O\big(Q * log(M)\big)$
    - Ngoài ra ta có thể sử dụng cấu trúc bitset cho mỗi ngăn sách, lúc này ta có thể thực hiện thao tác nghịch đảo một cách đơn giản bằng cách xor bitset của một tủ sách với một bitset khác chứa toàn 1. $O(Q * \frac{M}{32})$
- **Subtask 3:**

- Truy vấn nghịch đảo trông có vẻ phức tạp lúc đầu nhưng nếu ta để ý kỹ thì bài này không yêu cầu trả lời truy vấn một cách trực tiếp mà ta sẽ có trước dữ liệu của tất cả các truy vấn.
- Có thể nhận ra rằng, nếu coi các trạng thái là một đỉnh trên đồ thị thì đồ thị này sẽ có dạng một cái cây. Cây có gốc tại đỉnh 0, là trạng thái ban đầu khi tủ sách rỗng. Mỗi khi thực hiện một trong 3 thao tác đầu, ta sẽ nối đỉnh hiện tại với một đỉnh mới bằng một cạnh ghi lại nội dung của thao tác cập nhật này. Khi gặp một thao tác đảo ngược, ta sẽ đơn giản gán đỉnh hiện tại thành đỉnh của thao tác thứ k.
- Sau đó ta sẽ DFS từ đỉnh 0, thực hiện các thao tác khi đi đến một cạnh và đảo ngược lại thao tác khi đi ra khỏi cạnh đó. Ở đây ta vẫn sẽ sử dụng một trong hai cấu trúc dữ liệu như ở subtask trên để hỗ trợ. $O\big(Q * log(M)\big)$ hoặc $O(Q * \frac{M}{32})$

Bài 3. TREE (6 điểm)

- **Subtask 1:**
  - Duyệt qua các tất cả các cặp đường đi, với mỗi cặp thì lại duyệt qua tất cả các đỉnh trên đường đi để kiểm tra. $O(N^3)$
- **Subtask 2:**
  - Ở subtask này, giá trị của a_i đều bằng 1, vậy nên mọi cặp đường đi có độ dài lớn hơn 2 đều thỏa mãn. Vì vậy đáp án là tổng số cặp trừ đi số cặp đường đi có độ dài bằng 2 nên đáp án sẽ là $N * (N-1)/2 - (N-1)$. $O(1)$
- **Subtask 3:**
  - Subtask này đồ thị của chúng ta có hình một đường thẳng, bài toán trở thành: *"Cho một mảng độ dài N, tính số dãy con có tổng các phần tử lớn hơn hai lần giá trị lớn nhất"*.
  - Có thể tính trước mỗi phần tử là giá trị lớn nhất trong đoạn nào sử dụng stack, sau đó dùng cấu trúc dữ liệu để tính. Cách này có thể qua được những test sinh ngẫu nhiên nhưng độ phức tạp trong trường hợp tệ nhất vẫn là $O(N^2 logN)$.
  - Ở đây ta sẽ sử dụng chia để trị để giải bài toán này. Gọi hàm đệ quy $F(1,n)$, ta sẽ tính số đoạn thỏa mãn đi qua $mid = \frac{1+n}{2}$ sau đó gọi đệ quy xuống $F(1, mid)$ và $F(mid+1, n)$ để tính tương tự...
  - Xét hàm $F(l,r)$, ta sẽ có $mid - l + 1$ đoạn con ở phía bên tay trái và $r - mid$ đoạn con ở phía bên tay phải. Ta sẽ phải đếm số cách để ghép một đoạn bên trái với một đoạn bên phải sao cho tổng hai đoạn lớn hơn hai lần giá trị lớn nhất.
  - Xét hai đoạn con (s_c, m_c) và (s_p, m_p), giả sử $m_c \leq m_p$ hai đoạn ghép lại được khi $s_c + s_p > 2 * m_p$.
  - Ta sẽ gộp các đoạn con từ cả hai phía và sắp xếp lại tăng dần theo giá trị lớn nhất, sau đó sử dụng cấu trúc dữ liệu để tính số lượng thỏa mãn. Làm cách trên có thể sẽ bị ghép sai hai đoạn nằm cùng một phía, để khử chỗ này, ta chỉ cần tính tương tự cho riêng các phần tử ở mỗi bên (đây sẽ là

những đoạn thỏa mãn nhưng cùng phía), sau đó trừ đi giá trị tính được. $O(nlogn^2)$

- **Subtask 4:**
  - Ở subtask này, ta có thể nghĩ đến việc sử dụng Centroid Decomposition (Phân rã trọng tâm cây) để áp dụng chia để trị trên cây. Ở đây với mỗi lần duyệt một đỉnh centroid mới, bài toán của chúng ta sẽ trở thành: *"Cho một cây có gốc, tính số cặp đỉnh đi qua gốc thỏa mãn yêu cầu đề bài"*.
  - Ta có thể giải bài toán này bằng cách quy hoạch động trên cây. Ta sẽ có một nhánh chính và những nhánh con (là các cây con của con trực tiếp của gốc), ban đầu nhánh chính rỗng, ta sẽ xét từng nhánh con và gộp dần vào. Sau khi xét xong một nhánh con thì nhánh chính của chúng ta sẽ có một số những cặp chỉ số (s, m) nghĩa là tồn tại một đường đi có tổng từ gốc đến nó là s và đỉnh có trọng số lớn nhất là m. Khi gộp hai nhánh lại với nhau, điều ta cần làm là xác định xem với một cặp (s_c, m_c) của nhánh chính và (s_p, m_p) của nhánh phụ khi ghép lại có phải là một đường đi thỏa mãn hay không. Và điều này thỏa mãn khi $s_c + s_p > 2 * max(m_c, m_p)$.
  - Ở đây ta có thể tách ra làm hai trường hợp, một là $m_c < m_p$, hai là ngược lại, sau đó sử dụng hai cấu trúc BIT2D để tính toán. $O(NlogN^3)$
- **Subtask 5:**
  - Ta vẫn sẽ sử dụng ý tưởng của subtask trước, nhưng tối ưu phần đếm như subtask thứ 3 để đạt được độ phức tạp tốt hơn.
  - Thay vì ta phải gộp từng nhánh một và phải xét hai trường hợp $m_c < m_p$ hoặc $m_c > m_p$, ta có thể gộp hết tất cả các cặp chỉ số (s, m) lại và sắp xếp tăng dần theo m. Vì m tăng dần nên ta có thể sử dụng duy nhất một cấu trúc BIT.
  - Nhưng nếu gộp hết tất cả lại để tính một lần, ta sẽ bị tính sai những cặp chỉ số thỏa mãn mà lại nằm cùng một nhánh con. Ta có thể xử lý việc này đơn giản bằng cách tính số lượng các cặp trong cùng một nhánh con thỏa mãn và lấy tổng trừ đi số lượng này. $O(NlogN^2)$

---------- HẾT ----------