Câu 1: WISEQ

Subtask 1: N ≤ 20, W ≤ S

- Sinh nhị phân; so sánh tổng các dãy con tăng với W

Subtask 2: N ≤ 50, W = S

- Quy hoạch động
- Gọi f[i] là độ dài dãy con tăng dài nhất tính từ 1 và kết thúc tại i.
- Công thức QHĐ:

$$f[i] = \max(f[i], f[j] + 1) \text{ với } (j < i \text{ và } A[j] < A[i])$$

Subtask 3: N ≤ 50, W ≤ S

- Quy hoạch động Top – down
- Tính và lưu 3 trạng thái i, j và sum với i là vị trí hiện tại, j là vị trí số đã được chọn phía trước, sum là tổng hiện tại.
- Nếu bỏ qua vị trí hiện tại thì ta tính trạng thái (i + 1, j, sum)
- Với A[j] < A[i] && sum + A[i] <= W thì ta tính trạng thái (i + 1, i, sum + A[i]) và cộng thêm 1
- Cuối cùng lấy MAX của 2 trạng thái trên.

Subtask 4: N ≤ 500, W ≤ S

- Quy hoạch động đảo nhãn
- Gọi DP[i][l] là tổng nhỏ nhất có giá trị của phần tử lớn nhất là i và dãy con tăng dài nhất có độ dài là l
- Công thức QHĐ:

$$DP[A[i]][l] = \min(DP[A[i]][l], DP[j][l - 1] + A[i]);$$

(Với DP[j][l – 1] được xác định và j < A[i])

- Ta cập nhật kết quả nếu DP[i][l] được xác định. (Với i là giá trị của phần tử lớn nhất trong dãy con tăng dài nhất có độ dài l)

Subtask 5: N ≤ 50000, W = S

- Quy hoạch động kết hợp với cấu trúc dữ liệu đặc biệt (Binary Indexed Tree)
- Cải tiến từ Subtask 2
- Gọi dp[i] là độ dài dãy con tăng dài nhất tính từ 1 và kết thúc tại i.
- Công thức QHĐ:

$$dp[i] = \max(dp[i], dp[j] + 1) \text{ với } (j < i \text{ và } A[j] < A[i])$$

- Để có tìm được f[j] lớn nhất mà thỏa mãn đề bài thì ta xây dựng 1 cây BIT để có thể dễ dàng lấy được kết quả.
- Sau khi xây dựng được cây Binary Indexed Tree thì bây giờ ta sẽ tính

$$dp[i] = get(A[i] - 1) + 1;$$

Hàm get để tính dp[j] thỏa mãn điều kiện A[j] < A[i];

- Cuối cùng ta cập nhật giá trị dp[i] vào vị trí A[i] trong cây Binary Indexed Tree:

$$update(A[i], dp[i]);$$

Câu 2: Chi phí nhỏ nhất

*Nhận xét chung: Do $c_i \geq 2c_{i-1}$, $2 \leq i \leq K$ nên ta có $K \leq \log2(\max(c_i))$.

Sub 1: Ở sub này, $K \leq \log2(2000) \sim 10$ khá nhỏ đủ để ta sinh tất cả 2^K khả năng chọn các đồng xu, sau đó chỉ cần sử dụng thuật toán DFS để kiểm tra tính liên thông của đồ thị nếu sử dụng các đồng xu được chọn. ĐPT $O(K * 2^K * (N + M))$.

Sub 2: Sub này tương tự như sub 1, tuy nhiên do $N, M \leq 10^5$ khá lớn nên yêu cầu cài đặt thông minh, hợp lý để qua được toàn bộ test. Gợi ý: có thể sử dụng các phép toán xử lý bit khi kiểm tra khả năng thông qua của một đường đi. ĐPT $O(2^K * (N + M))$.

Sub 3: Ta thấy $K \leq \log2(10^{18}) \sim 59$ là không còn đủ nhỏ để sinh toàn bộ khả năng chọn các đồng xu nữa. Vậy hãy thử xem xét điều kiện $c_i \geq 2c_{i-1}$, $2 \leq i \leq K$ ta sẽ nhận thấy một điều rất quan trọng: $c_i > \sum c_j$, $1 \leq j \leq i - 1$. Chứng minh:

Giả sử ta có $c_i = 2c_{i-1}$, $c_1 = 1$ hay $c_i = 2^{i-1}$, khi đó $\sum c_j = 2^{i-1} - 1 < c_i = 2^{i-1}$, $1 \leq j \leq i - 1$. Vì vậy khi $c_i > 2c_{i-1}$ khẳng định trên vẫn luôn đúng. Từ đó suy ra đpcm.

Từ nhận xét trên, ta nhận ra rằng việc phải chọn đổi cả i - 1 đồng xu đầu tiên vẫn lời hơn việc phải chọn đổi một đồng xu thứ i. Do đó nảy sinh ra thuật toán tham lam để giải quyết bài toán như sau: Giả sử ban đầu ta chọn đổi hết K đồng xu, duyệt các đồng xu theo mệnh giá giảm dần, ta thử bỏ chọn đổi đồng xu đang duyệt và kiểm tra tính liên thông của đồ thị. Nếu đồ thị vẫn đảm bảo tính liên thông thì ta sẽ không chọn đổi đồng xu này, ngược lại ta bắt buộc phải chọn đổi. Việc còn lại là cài đặt thông minh, hiệu quả theo gợi ý ở sub 2 để qua toàn bộ test. ĐPT O(K * (N + M)).

Câu 3: Keyboard

- Nhận xét là xâu s thực sự không quá quan trọng, chỉ cần biết được $c[x][y]$ là số lần ta đang từ kí tự x trong xâu s mà chuyển sang kí tự tiếp theo là y. Kết quả cho một bàn phím nhất định sẽ là:
  $sum(abs(pos[x] - pos[y]) \times (c[x][y] + c[y][x]))$.
- Bây giờ nhận thấy là nếu ta biết được x sẽ đứng trước hay sau y trong bàn phím thì ta có thể tách cái tổng ra làm hai phần chỉ liên quan đến $pos[x]$ và $pos[y]$, từ đó có thể tính được các giá trị đóng góp vào độ trễ của $pos[x]$ và $pos[y]$ mà không quan tâm đến giá trị còn lại.
- Cụ thể, ta sẽ đi xây dựng bàn phím từ trái sang phải. Nếu ta hiện tại có xâu bàn phím là a và tập các số chưa được thêm vào là S thì có thể mô tả các cách chọn như sau:
  + Với mỗi $x \in S$, nếu ta thêm x vào xâu a thì chi phí mà x phải trả là:

$$\sum_{y \in a} pos[x] \times (c[x][y] + c[y][x])$$

$$+$$

$$\sum_{y \in (S \setminus \{x\})} -pos[x] \times (c[x][y] + c[y][x])$$

(Vì x xuất hiện sau mỗi kí tự trong a nên cộng vị trí của x cho mỗi lần di chuyển giữa một kí tự trong a đến x và vì x xuất hiện trước các kí tự trong $(S\backslash\{x\})$ nên phải trừ đi vị trí của x.)

- Điều này cũng có nghĩa rằng xâu a là gì không quan trọng, chỉ cần biết tập hợp các chữ cái có trong a.
- Bài toán bây giờ trở thành từ tập rỗng, thêm dần các kí tự, mỗi lần thêm thì sẽ phải trả chi phí nhất định tùy vào việc đang thêm vào tập nào, hỏi cách thêm thế nào mà tốn ít chi phí nhất. Thì đây là một bài quy hoạch động được, cái chính là lưu các tập như thế nào.
- Việc lưu tập là cái chỗ mà bitmask có tác dụng. Cụ thể, ta lưu lại một dãy bit sao cho bit 0 thì phần tử đó không có trong tập, bit 1 thì phần tử đó có trong tập. Thao tác thêm từ một tập thì đơn giản là bật một cái bit lên để chuyển đến tập mới. Rõ ràng thì mỗi bitmask sẽ ứng với một số, vì thế ta có thể lưu trữ các tập bằng một mảng số nguyên mà không tốn kém nhiều chi phí để đọc / ghi vào các tập.
- Biết cách lưu tập hợp và biết cách chuyển trạng thái thì ta có thể thực hiện quy hoạch động một cách đơn giản.