DOMINIC

APPROACH

Quy hoạch động có lẽ là cách tiếp cận đơn giản nhất cho bài toán này. Do số lượng dấu '_' rất nhỏ, ta quy hoạch động theo vị trí dấu '_' hiện tại, lưu thêm 2 chiều là giá trị của 2 dấu '_' trước đó và chữ 'L' đã xuất hiện trước đó hay chưa.

Lưu mảng f[i][c1][c2][L], trong đó i là vị trí dấu '_' hiện tại, c1 và c2 là giá trị của 2 dấu '_' cuối cùng, L bằng 0 hoặc 1. Bài toán có thể giải được với độ phức tạp 10 * 27 * 27 * 2 * 27

IMPROVEMENT

Trong bài toán này, các chữ cái được chia vào 3 nhóm. Nhóm thứ nhất gồm chữ 'L', nhóm thứ 2 gồm các nguyên âm, nhóm thứ 3 là các phụ âm khác 'L'. Khi quy hoạch động, ta chỉ cần quan tâm tới nhóm của các chữ cái, lưu lại số lượng chữ cái mỗi nhóm đã được sử dụng. Kết quả cuối cùng được suy ra bằng một phép nhân đơn giản. Độ phức tạp cho cách làm này là 10 * 3 * 3 * 2 * 2 * 10 * 10

HAN

APPROACH

Dễ thấy giải thuật tham lam: để thỏa mãn yêu cầu của bài toán, cái kẹo có mã số lớn nhất của HAN phải ghép cặp với cái kẹo có mã số nhỏ nhất của Gisele, cái kẹo có mã số lớn thứ hai của HAN phải ghép cặp với cái kẹo có mã số nhỏ thứ hai của Gisele,... cứ như vậy cái kẹo có mã số lớn thứ i của Han phải ghép cặp với cái kẹo có mã số nhỏ thứ i của Gisele.

Gọi dãy số của Han có là a, dãy số Gisele có là b.

Chứng minh: giả sử cách làm tối ưu tồn tại 2 cặp $a_1 - b_1$, $a_2 - b_2$ sao cho a1 >= a2, b1 >= b2. Tổng lớn nhất là a1+b1. Ta ghép lại a1 với b2, a2 với b1, khi đó a1+b2 <= a1 + b1, a2 + b1 <= a1 + b1. Rõ ràng, cách ghép mới là tốt hơn cách ghép cũ, và cách ghép cặp cũ là không tối ưu. Như vậy, ta suy ra số lớn nhất trong dãy a luôn phải ghép gặp với số nhỏ nhất của dãy b.

IMPLEMENTATION

Với mỗi cặp số mới được đưa ra, chúng ta phải thực hiện ghép cặp lại. Chi phí cho việc sắp xếp lại ít nhất là logN, chi phí ghép cặp lại là N. Độ phức tạp của cả bài toán là N^2 hoặc N^2 logN, tùy cách cài đặt.

Nhưng ta cần để ý rằng, giới hạn của các số đưa ra là rất nhỏ (<= 100). Ta có thể sắp xếp bằng phương pháp đếm phân phối. Khi đó chi phí cho việc sắp xếp lại là O(1) (phép phân phối), chi phí cho việc ghép cặp là 2 * max(a, b) <= 200. Độ phức tạp của bài toán là N * 2 * max (a, b)

VIN

(VUKVN trên vnoi)

Cách làm thứ nhất: Chặt nhị phân kết quả.

Với mỗi khoảng cách mid, đánh dấu tất cả các ô có khoảng cách < mid tới một nhà dân là không thể đi đến. BFS từ nhà của Vin, qua các ô chưa bị đánh dấu, nếu đến được trường học thì đáp án của bài toán >= mid.

Cách làm thứ hai: Sử dụng heap

BFS sử dụng priority_queue thay vì queue thông thường. Khi đưa một ô kề cạnh vào priority queue (heap), đưa thêm khoảng cách của ô đó tới một nhà dân gần nhất làm giá trị so sánh. Sắp xếp heap sao cho ô có giá trị này lớn nhất được ưu tiên lấy ra trước. Dừng BFS khi đến được ô đích. Đáp án bài toán là giá trị nhỏ nhất của các khoảng cách đã lấy ra từ trong heap trong quá trình BFS.

Cài đặt bằng C++: sử dụng priority_queue thay cho queue. Thay vì push ô (u, v) vào queue, ta push pair (k, (u, v)) vào priority_queue, trong đó k là khoảng cách của ô (u, v) đến căn nhà gần nhất. Các bước khác cài đặt như BFS thông thường.

MIA

Để số cuộc gọi là ít nhất có thể, ta luôn muốn kéo dài một cuộc gọi xa nhất có thể. Khi chưa buộc phải ngắt một cuộc gọi, ta coi như cuộc gọi ấy vẫn còn tiếp tục. Khi tổng số cuộc gọi qua một vị trí lớn hơn số cuộc gọi máy đo ở vị trí đó đo được, ta ngắt bỏ các cuộc gọi thừa. Từ đó ta đi đến giải thuật tham lam:

Lưu giá trị S là tổng số cuộc gọi đang có. Ban đầu S bằng 0. Duyệt lần lượt qua các máy đo. Tới máy đo thứ i, nếu $P_i > S$, ta tang $S = P_i$. Nếu $P_i <= S$, ta cộng $(S - P_i)$ vào kết quả, đồng thời giảm S xuống P_i.