Domino

Subtask 1: với subtask này ta đơn giản chỉ cần duyệt thử lần lượt các cách di chuyển của các quân domino và kiểm tra xem quân domino tiếp theo có thể dịch chuyển là quân nào, đánh dấu các quân đã dịch chuyển và từ đó tính được kết quả. Với mỗi ô trống có tối đa 3 quân domino khác có thể dịch chuyển vào ô này, trừ ô đầu tiên sẽ có 4 cách, và số lượng quân domino tối đa là 12 nên số phép duyệt chỉ là 3^{12}.

Subtask 2: Vì chỉ có tối đa 2 quân domino có thể dịch chuyển, do đó ta duyệt các quân domino cạnh ô trống ban đầu, và mỗi quân domino di chuyển được, ta duyệt các quân domino có thể di chuyển đến ô trống khi di chuyển quân domino thứ nhất, chỉ cần kiểm tra được quân nào có thể di chuyển vào ô trống là xong.

Subtask 3: Với một ô trống (x, y) ta sẽ biết được domino nào có thể di chuyển vào ô này, và khi một quân domino đó dich chuyển vào ô (x, y) thì nó sẽ tạo ra một ô trống mới $(x^{'}, y')$, cứ như vậy lại có những quân domino khác có thể di chuyển vào $(x^{'}, y')$. Do đó ta có thể quy bài toán này về đồ thị có hướng như sau:

- Coi mỗi ô của bảng là một đỉnh của đồ thị
- Ô (x, y) có cạnh nối đến ô $(x^{'}, y')$ nếu có quân domino có thể di chuyển vào ô (x, y) và để lại ô trống là $(x^{'}, y')$.
- Từ đây, từ ô trống đầu tiên, ta sẽ loang được đến các ô trống khác, và sẽ biết được các quân domino nào có thể di chuyển được.

Even Subsequences

Subtask 1: Với subtask này ta có thể làm thuật toán $O(n^2)$ đơn giản là duyệt từng đoạn (i, j) và tìm max min bằng cách cố định i và chạy j.

Subtask 2: Vì chỉ có 3 giá trị của a_i là 1, 2, 3, những dãy thỏa mãn là những dãy con liên tiếp hoặc chỉ gồm những số giống nhau, hoặc chỉ chứa những số có giá trị là 1 và 3. Như vậy những dãy không thỏa mãn phải chứa hoặc là 1 và 2, hoặc là 2 và 3. Nếu đếm được số dãy con liên tiếp có phải chứa hai số 1 và 2 là s_{12}, số dãy con liên tiếp phải chứa hai số 2 và 3 là s_{23}, thì kết quả số dãy thỏa mãn là:

$$ans = \frac{n \times (n+1)}{2} - s_{12} - s_{23}$$

Vậy giờ ta cần tìm cách tính được s_{12} và s_{23}. Xét tính s_{12}:

- Gọi s_1 là số dãy con liên tiếp chỉ chứa số 1, s_2 là số dãy con liên tiếp chỉ chứa số 2.
  - Ta có thể tính được s_1 bằng cách tìm các đoạn $[i, j]$ mà $\forall i \leq k \leq j$ có $a_k == 1$ và $a_{i-1} \neq 1$, $a_{j+1} \neq 1$ (1), số dãy chỉ chứa số 1 trong đoạn $[i, j]$ là:

    $$d(1, i, j) = \frac{(j - i + 1) \times (j - i + 2)}{2}$$

  - Vậy số lượng s_1 sẽ là tổng của các $d(1, i, j)$
  - Tương tự tính được s_2
- Gọi $\overline{s_{12}}$ là số dãy con liên tiếp có thể chứa hoặc số 1 hoặc số 2 hoặc cả 1 và 2. Tương tự ta cũng có thể tính được $\overline{s_{12}}$ như tính s_1. Sau khi tính được $\overline{s_{12}}$, s_1 và s_2 ta có thể tính được s_{12}:

  $$s_{12} = \overline{s_{12}} - s_1 - s_2$$

Tương tự tính được s_{23}. Vì ta có thể tìm được các đoạn $[i, j]$ thỏa mãn (1) trong $O(n)$, và số lần cần tìm sẽ là khi tính s_1, s_2, s_3, $\overline{s_{12}}$ và $\overline{s_{23}}$ nên thuật toán chỉ là $O(n)$.

Subtask 3: Có thể có nhiều cách làm để đạt được full điểm bài này, ở đây tôi đề xuất thuật toán chia để trị vào giải quyết được bài toán này:

- Giả sử ta đang tính đến đoạn $[l, r]$ và chia ra thành 2 phần là $[l, mid]$ và $[mid + 1, r]$, giờ ta cần đếm số cặp (i, j) với $l \leq i \leq mid < j \leq r$ sao cho thỏa mãn.
- Ta xây dựng 4 mảng sau:
  - $maxL[i]$ là số lớn nhất trong đoạn $[i, mid]$
  - $minL[i]$ là số nhỏ nhất trong đoạn $[i, mid]$
  - $maxR[i]$ là số lớn nhất trong đoạn $[mid + 1, r]$
  - $minR[i]$ là số nhỏ nhất trong đoạn $[mid + 1, r]$
- Nếu ta duyệt i và với mỗi i đếm xem có bao nhiêu giá trị j thỏa mãn, tuy nhiên ta thấy rằng nếu xét tất cả $mid < j \leq r$ thì rất khó xác định được max và min của đoạn $[i, j]$:
  - do đó ta chỉ xét j với $maxL[i] \geq maxR[j]$ để dễ dàng xác định được max và min của đoạn $[i, j]$
  - và để tránh xét thiếu các j mà $maxL[i] < maxR[j]$ thì ta chỉ cần làm thêm là xét từng j và đếm số lượng i thỏa mãn, để tránh hai lần tính bị trùng lặp các cặp (i, j) thì ta chỉ xét những giá trị i với $maxL[i] < maxR[j]$.

- Với i đang xét ta sẽ tìm được j_m là giá trị lớn nhất có $maxL[i] \geq maxR[j_m]$, ta biết min của các số bên trái là $minL[i]$, vậy giá trị j xác định min trong đoạn $[i, j]$ sẽ hoặc chính là $minL[i]$ hoặc là $minR[j]$:
  - Gọi j_s là vị trí lớn nhất mà $minL[i] \leq minR[j_s]$, vậy những $mid < j \leq j_s$ thì min của đoạn $[i, j]$ sẽ là $minL[i]$, vậy nếu $maxL[i] - minL[i]$ là số chẵn thì số lượng j thỏa mãn chính là $j_s - mid$, ngược lại thì 0 có giá trị j nào thỏa mãn.
  - Với $j_s < j \leq j_m$ thì min của đoạn $[i, j]$ sẽ là $minR[j]$, vậy các giá trị j thỏa mãn sẽ phụ thuộc vào $maxL[i]$ là chẵn hay lẻ. Do đó ta cần một mảng cnt để đếm số lượng giá trị $minR[j]$ là chẵn hay lẻ. Từ đó có thể đếm được có bao nhiêu giá trị $j_s < j \leq j_m$ thỏa mãn.
  - mảng cnt sẽ được cập nhật theo j_m và j_s, khi ta duyệt i giảm dần thì j_m và j_s sẽ tăng dần nên ta sẽ cần cập nhật cnt với tất cả i trong $r - mid$ lần.
- tương tự ta cũng có thể đếm i khi duyệt j theo cách trên.

Simulation

Dễ dàng nhận thấy rằng ta có thể tách biệt tọa độ x và y, do đó ta có thể quy bài toán ban đầu về bài toán sau: cho dãy số x_i có n phần tử, xem xét các truy vấn sau:

- thay đổi giá trị của x_i
- đếm xem có bao nhiêu giá trị i mà $\sum_{1\leq j<i} x_j$ và $\sum_{1\leq j\leq i} x_j$ khác dấu nhau.

Gọi $p_i = \sum_{1\leq j\leq i} x_j$, xét bài toán với dãy p_i làm sao đếm được số lượng i mà p_{i-1} và p_i khác dấu nhau, ta cần tìm cách để khi x_i thay đổi thì vẫn có thể đếm được một các nhanh nhất.

- Xây dựng hai tập L và R như sau: xét từng cặp (p_{i-1}, p_i) gọi $a = \max(p_{i-1}, p_i)$, $b = \min(p_{i-1}, p_i)$, thêm a vào tập R và thêm b vào tập L
- Gọi số phần tử nhỏ hơn d của tập L là $L(d)$
- Gọi số phần tử nhỏ hơn d của tập R là $R(d)$
- Số lượng i thỏa mãn p_{i-1}, p_i trái dấu sẽ là $L(0) - R(0)$ (lưu ý rằng các giá trị p_i trong bài toán này luôn là số lẻ nên ta ko cần quan tâm đến việc có $p_i = 0$ hay không).

Ta có thêm nhận xét sau: khi x_1 thay đổi một lượng là d thì tất cả các p_i sẽ đều thay đổi một lượng là d, nếu ta hình dùng thì các điểm này được tịnh tiến một lượng là d so với trục tọa độ, ta có thể hình dung theo cách khác thì là trục tọa độ tịnh tiến một lượng là $-d$ so với các điểm này. Như vậy khi tịnh tiến một lượng là d thì số lượng i thỏa mãn sẽ là $L(-d) - R(-d)$.

Từ những nhận xét trên, ta có thuật toán cho bài toán như sau:

- Xây dựng mảng p_i với dữ liệu vector ban đầu
- Vì khi con trỏ đứng ở vị trí i thì chỉ các điểm $i+1$ bị tịnh tiến cùng một lượng d nên tập L và R sẽ duy trì các phần tử của đoạn $[i+1, n]$
- ta cập nhật d là mức thay đổi của các điểm trong đoạn $[i+1, n]$, d sẽ là tổng độ thay đổi của các vector từ 1 đến i.
- giá trị của p_j với $1 \leq j \leq i$ ta cập nhật theo đúng với lượng mà các điểm này bị thay đổi, và vì i mỗi lần thay đổi chỉ 1 đơn vị nên ta có thể tính được số cặp (p_{j-1}, p_j) với $1 \leq j \leq i$ cắt qua trục tọa độ, nên ta dùng t để duy trì số lượng này. Lưu ý với thao tác 'B' ta cần thay đổi p_i ứng với độ thay đổi là d cùng với các p_k với $k > i$.
- Với mỗi truy vấn 'Q' ta có kết quả sẽ là $t + L(-d) - R(-d) + f(p_i, p_{i+1} + d)$ (với $f(a, b)$ là hàm kiểm tra xem a và b có trái dấu hay không).

Như vậy, với mỗi trục x và y ta sẽ duy trì các biến như trên, và kết quả là tổng của 2 phần. Tập L và R ta có thể sử dụng BIT động (sử dụng unordered_map) hoặc Trie để có thể truy vấn một cách hiệu quả với bài toán này.

Lottery

Subtask 1: duyệt hoán vị kiểm tra từng hoán vị, độ phức tạp là $O(N! \times N)$.

Subtask 2: Với mỗi đỉnh u ta xuất phát từ đỉnh này và duyệt dfs đến tất cả các đỉnh khác, mỗi đỉnh ta xác định được một đường đi tương ứng với một dãy số may mắn. Độ phức tạp là $O(N^2)$.

Subtask 3: Với đồ thị cho với subtask này, mỗi đỉnh chỉ thuộc nhiều nhất một chu trình đơn, ta có thể sử dụng quy hoạch động như sau:

- Gọi $dp[u][v][l]$ là số đường đi xuất phát từ u đi qua đỉnh v có độ dài là l, với v là đỉnh kề với u, và u và v không thuộc cùng một chu trình
- Gọi $dpc[u][l]$ là số đường đi xuất phát từ u có độ dài là l và các đường đi này phải đi qua một trong các đỉnh của chu trình chứa u.
- Dễ dàng thấy:
$$dp[u][v][l] = dpc[v][l-1] + \sum_{w \in \{e(v) \setminus u\}} dp[v][w][l-1]$$
- Để tính $dpc[u][l]$ ta thấy nếu từ u đi đến v sau đó đi từ v ra các đỉnh khác không thuộc chu trình của u và v, thì nếu chu trình u có số lượng đỉnh là z_u và khoảng cách ngắn nhất giữa u và v là d thì ta có thể đi theo chiều ngược lại từ u đến v với khoảng cách là $z_u - d$ nên ta công thức sau:
$$dpc[u][l] = \sum_{v \in \{c(u) \setminus u\}} \sum_{w \in e(v), w \notin c(u)} (dp[v][w][l-d] + dp[v][w][l-(z_u - d)])$$
với c(u) là tập các đỉnh cùng chu trình với u.
- Sử dụng đệ quy có nhớ xuất phát từ tất cả n đỉnh để tính được 2 mảng trên, để giảm bộ nhớ sử dụng map và vector để lưu thông tin tại từng đỉnh, và sử dụng mảng cộng dồn để có thể giảm thời gian tính toán khi tính $\sum_w$. Độ phức tạp sẽ là $O(N^2)$.

Subtask 4: Để tổng quát hóa bài toán, ta coi các cạnh nối (u, v) là một chu trình nếu u và v không thuộc cùng một chu trình.

- Gọi $g(u)$ là tập chu trình đi qua u
- Gọi $c(k)$ là tập các đỉnh trong chu trình k
- Gọi $z(k)$ là số lượng đỉnh thuộc chu trình k
- Gọi $d(u, v)$ là đường đi ngắn nhất giữa u và v
- Gọi $dp[u][k][l]$ là số đường đi có độ dài l xuất phát từ u và không đi qua đỉnh nào khác của chu trình k, với k là chu trình chứa u. Ta có công thức sau:
$$dp[u][k][l] = \sum_{k' \in \{g(u) \setminus k\}} \sum_{v \in \{z(k) \setminus u\}} (dp[v][k'][l - d(u,v)] + (z(k') > 2) \times dp[v][k'][l - (z(k') - d(u,v))]$$
- vế thứ hai cần điều kiện là chu trình k' phải có nhiều hơn 2 đỉnh nếu không ta chỉ có một lối đi duy nhất giữa u và v.
- Từ công thức trên ta có thể thực hiện như sau:
  - Xây dựng dp là một map với key là (u, k) và value là một vector có độ dài tương ứng với đường đi dài nhất từ u không đi qua chu trình k

- Ta xây dựng hàm $solve(u, k)$ trả về vector $dp[u][k]$
- Với mỗi đỉnh u ta sẽ gọi $solve(u, -1)$ để có trả về vector số đường đi xuất phát từ u, các chu trình sẽ đánh số từ 0 trở đi.
- Với một vector $dp[v][k']$ trả về khi tính $dp[u][k]$ ta có một hàm *update* $dp[u][k]$ theo $dp[v][k']$ với độ dài là $d(u, v)$
- vector kết quả ans khởi tạo ban đầu là 1 vector có $n + 1$ phần tử có giá trị là 0, sau đó dùng hàm update để cập nhật ans với mỗi $dp[u][-1]$.