ICPC Kanpur Regional 2019 Solutions

Setter
Mohammad Kaykobad
Hasin Rayhan Dewan Dhruboo
Anik Sarker

Tester
Suchan Park
Jatin Yadav

Editorialist
Jatin Yadav
Anik Sarker

Is It a Giveaway

Problem

You are given a range $[L, R]$ in which you choose K random real numbers. What is the expected value of the number of distinct numbers?

Is It a Giveaway

Solution

- If $L \neq R$, all chosen elements would be distinct as there are infinite real numbers between L and R. So, the answer is K.
- If $L = R$, all chosen elements must be equal to L, so the answer is 1.

Convolution

Problem

Given two arrays $A[1..n]$ and $B[1..m]$, Let's define,

$$Score(A, B) = \sum_{i=1}^{n} \sum_{j=1}^{m} A_i \times B_j$$

You will have to -

- perform range additions on both the arrays
- output $Score(A, B)$.

Solution

- We can rewrite -

$$Score(A, B) = \sum_{i=1}^{n} A_i \times \sum_{j=1}^{m} B_j$$

- Lets keep two variables : $f = \sum_{i=1}^{n} A_i$ and $g = \sum_{j=1}^{m} B_j$.
- For range additions, update f or g.
- For $Score(A, B)$, output $f \times g$.

Prime-partite graph

Problem

Consider a graph with n nodes $1, 2, \ldots n$. There is an undirected edge between i and j if and only if i is a prime divisor of j or vice-versa. Given Q queries of the form a, b, find the shortest distance between a and b in this graph.

Prime-partite graph

Solution

- Let P be the set of primes in $[n]$, and $Q = [n] \setminus P$
- The graph is a bipartite graph, where each edge is between a node in P and a node in Q. In any path, the nodes must alternate between prime and not prime.
- The primes $> \frac{n}{2}$ are connected to no other node.
- Let sp_r be the smallest prime factor of r
- Consider a query a, b.
- If $a = b$, the answer is 0. Else if either of a or b is a prime $> \frac{n}{2}$, the answer is -1.

Prime-partite graph

Solution...

If exactly one of a and b is a prime, let it be a without loss of generality. The remaining cases are the following:

1. a and b are both primes:
   - If $ab \leq n$, the answer is 2 : $a \to ab \to b$.
   - Else, the answer is 4 : $a \to 2a \to 2 \to 2b \to b$
2. a is a prime but b is not:
   - If a divides b, the answer is 1.
   - Else if $sp_b \times a \leq n$, the answer is 3 : $a \to sp_b \times a \to sp_b \to b$
   - Else, the answer is 5 : $a \to 2a \to 2 \to 2sp_b \to sp_b \to b$

Prime-partite graph

Solution...

3. a and b are both not primes:
   - If $\gcd(a, b)! = 1$, the answer is 2 : $a \to sp_{\gcd(a,b)} \to b$.
   - Else if $sp_a \times sp_b \leq n$, the answer is 4 :
     $a \to sp_a \to sp_a \times sp_b \to sp_b \to b$
   - Else the answer is 6 :
     $a \to sp_a \to 2sp_a \to 2 \to 2sp_b \to sp_b \to b$

Walk 4 steps

Problem

You are given a weighted undirected simple graph.

Let $f(X)$ = number of walks of length 4 such that the maximum weight of an edge on the walk is exactly X.

For each query X, calculate $f(X)$.

Walk 4 steps

Solution

- Let $g(X)$ = number of walks of length 4 such that the maximum weight of an edge on the walk is at most X.
- There can be only 100 distinct X values.
- Precompute for all possible $g(X)$ values.

Walk 4 steps

Solution

- For each X, do the following :
- build an adjacency matrix keeping only the edges having weight $\leq X$.
- Calculate the 4^{th} power of the matrix and sum all the elements of the matrix to calculate $g(X)$.
- $f(X) = g(X) - g(X-1)$

IPL begins

Problem

There are p cricketers and q footballers to be accomodated in r rooms, such that:

- No room is empty.
- A room may contain either only footballers or only cricketers, not both.
- No room can have exactly 1 cricketer.

Find the number of ways in which the players can be given rooms. All rooms are considered identical. That is, two ways are distinct if and only if there exist two players who are in the same room in one way but in different rooms in the other.

IPL begins

Solution

- To answer a query, we can iterate on the number of rooms occupied by cricketers.
- Let $f(n, k)$ be the number of ways to divide n cricketers into k identical rooms such that no cricketer is alone in a room.
- $f(n, k) = kf(n-1, k) + (n-1)f(n-2, k-1)$, as there are two cases:
  1. If cricketer 1 has exactly one roommate, this gives $(n-1)f(n-2, k-1)$ ways, as there are $n-1$ ways to choose the roommate and $f(n-2, k-1)$ ways for the other $n-2$ cricketers.
  2. If cricketer 1 has > 2 roommates, we can first divide the remaining $n-1$ cricketers into k rooms and then assign cricketer 1 to any of the k rooms. The total number of ways is $kf(n-1, k)$

Solution

- Let $g(n, k)$ be the number of ways to divide n footballers into k identical rooms. It is the same as stirling number of second kind.
- $g(n, k) = kg(n-1, k) + g(n-1, k-1)$, as there are two cases:
  1. Footballer 1 is alone. This gives $g(n-1, k-1)$ ways.
  2. Footballer 1 has atleast one roommate. divide the remaining $n-1$ footballers into k rooms and then assign footballer 1 to any of the k rooms. The total number of ways is $kg(n-1, k)$
- The answer to a query is $\sum_{k=0}^{r} f(p, k)g(q, r-k)$

IPL begins

Analysis

- The precomputation of both f and g takes time $O(N^2)$
- Every query takes time $O(r)$

K-Prob

Problem

Given a string S, find for each L, the probability that K random substrings of size L chosen at random are all equal.

K-Prob

Solution

- Let $t_1, t_2, \ldots t_r$ be all the distinct substrings of length L. Let these have a frequency of $f_1, f_2, \ldots f_r$ respectively. Then the answer for L is clearly $\frac{\sum f_i^K}{(n-L+1)^K}$
- Let $p_1, p_2, \ldots p_n$ be indices arranged in lexicographic order of the suffixes. This ordering can be found using a suffix array algorithm. You also store the longest common prefix of the suffixes starting at p_i and p_{i+1} for all i. This array of LCP of neighbouring suffixes in the suffix array is called the LCP array.
- Consider the length L substrings ordered lexicographically. They will follow the same order as the suffix array.
- Strings of length L starting at p_i and p_{i+1} are distinct if and only if $LCP[i] < L$.

K-Prob

Solution

- We can maintain equal substrings of length L as a single component in a DSU having an edge between i and $i+1$ if and only if $LCP[i] \geq L$.
- Then, the frequencies $f_1, \ldots f_r$ are same as the sizes of connected components in this DSU.
- Iterate over L from n to 1, and add all edges corresponding to $LCP[i] = L$ when processing length L. Also maintain sum of K^{th} powers of sizes of connected components in the DSU.
- The complexity is $O(N \log N)$ for finding the suffix array. The TL is enough to allow $O(N \log^2 N)$ suffix array algorithm also.

Solution

- There is also a solution using Suffix Automaton.
- We can count the number of occurrence for each 'endpos' class in its corresponding node.
- Since each 'endpos' class covers a continuous range over length of sub-strings, we can do simple range-updates.
- The Complexity is $O(N)$.

Weighted Necklace

Problem

There are K different colors of beads. The i^{th} colored beads have weight w_i. For each color, there are infinite beads. Given q queries, each containing an integer S, you have to find the number of different circular necklaces of $n = 2^k$ beads upto rotation, with a total weight of S.

Weighted Necklace

Solution

- We can use Burnside's lemma. The group acting is rotation group having elements $R_0, R_1, \ldots R_{n-1}$, where R_i denotes clockwise rotation by i.
- Consider one such element R_i. Let $g = 2^r = \gcd(n, i)$. A necklace A is fixed by R_i if and only if $A_j = A_{(j+i) \mod n}$ for all i.
- Since we can rotate any number of times, this means $A_j = A_{j \mod g}$ for all j.
- Consider the different groups of indices modulo g. Then the necklace A is fixed by R_i if and only if, for all groups, the colors of all beads in that group is the same.

Solution

- Let $G_0, G_1, \ldots G_{2^r-1}$ be these groups. Each group has size 2^{k-r}. Let c_j be the color of group j.
- We want $2^{k-r} \sum w_{c_j} = S$.
- Or we want to choose 2^r colors with sum of weights equal to $\frac{S}{2^{k-r}}$
- Consider the polynomial $W(x) = \sum x^{w_i}$. Then we want the coefficient of $x^{\frac{S}{2^{k-r}}}$ in $P(x)^{2^r}$.
- We can precompute first N coefficients of all the polynomials $Q_r(x) = P(x)^{2^r}$ in time $O(N \log^2 N)$ using the recurrence $Q_r(x) = Q_{r-1}(x)^2$.

Solution

- On a query, we iterate over different r such that 2^{k-r} divides S.
- The number of i for which $\gcd(i, n) = 2^r$ is $h(r) = 2^{k-r-1}$ if $r \neq k$ and 1 if $r = k$.
- The answer is $\frac{\sum h(r)[x^{\frac{S}{2^{k-r}}}]Q_r(x)}{n}$
- The overall complexity is $O(N \log^2 N)$ for precomputation and $O(\log N)$ per query, where N is the maximum limit on the weights, n, and S.

Minimize the Distance

Problem

Given two line segments AB and CD, find minimum distance between points on the two segments,, i.e. minimum $|PQ|$ over all points $P \in AB$ and $Q \in CD$.

Minimize the Distance

Solution

- Lets forget about segments, solve the problem for lines.
- We can use 'Vector Differentiation'
- Let $P = A + ab * x$, where $ab = B - A$
- $Q = C + cd * y$, where $cd = D - C$

Minimize the Distance

Solution

- $PQ = P - Q = (A + ab * x) - (C + cd * y)$
- $PQ = ca + ab * x - cd * y$, where $ca = A - C$
- $|PQ|$ will be minimum when $|PQ|^2$ is minimum
- Let $f(x, y) = |PQ|^2 = (ca + ab * x - cd * y)^2$

Minimize the Distance

Solution

- Differentiate f w.r.t both x and y.
- For f to be minimum. $\frac{df}{dx}$ and $\frac{df}{dy}$ must be zero.
- $0 = \frac{df}{dx} = 2 * ab * (ca + ab * x - cd * y)$
- $0 = \frac{df}{dy} = -2 * cd * (ca + ab * x - cd * y)$
- Simplifying we get,
  - $ab * ca = -ab * ab * x + ab * cd * y$
  - $cd * ca = -cd * ab * x + cd * cd * y$

Minimize the Distance

Solution

- Two equations with two unknowns x an y:
  - $ab * ca = -ab * ab * x + ab * cd * y$
  - $cd * ca = -cd * ab * x + cd * cd * y$
- We can use Cramer's rule now.

Minimize the Distance

Solution

- However there are several cases to take care of.
  - Two lines are parallel (Determinant zero)
  - The minimum distance points are outside the segments
- In such cases we need to find for a **terminal point** its minimum distance to the other line segment.

Minimize the Distance

Solution

- We can find minimum distance between a point and a line using similar idea of 'Vector Differentiation'.
- In case the minimum distance point is again outside the segment, we need to check the distance from the point to the **terminal points** of the segment.

Minimize the Distance

Solution

- Remember to check for **coincident line segment pairs** and **zero-length line segments**.
- Also there are ternary search solutions with precomputation and numerical optimizations.

Beauty and The Array

Problem

$$Score(p[1..m], q[1..m]) = \sum_{i=1}^{m} (m - i + 1) * p[i] + q[i]$$

$$Beauty(p[1..n], q[1..n]) = max_{1 \le i \le j \le n} Score(p[i..j], q[i..j])$$

Given two arrays A and B of n integers, you can perform any number of simultaneous left circular rotations on both the arrays. Maximize the Beauty value.

Beauty and The Array

Solution

- Lets assume the left circular rotations are not allowed.
- Let $cSmA(i) = \sum_{j=1}^{i} A(j)$
- $ccSmA(i) = \sum_{j=1}^{i} A(j) \times (i - j + 1)$
- $cSmB(i) = \sum_{j=1}^{i} B(j)$
- For a subarray (u, v), where $u \leq v$, we can write : $Score(u, v) = ccSmA(v) - ccSmA(u-1) - cSmA(u-1) \times v + cSmA(u-1) \times (u-1) + cSmB(v) - cSmB(u-1)$

Beauty and The Array

Solution

- We can rewrite $Score(u, v)$ as :

$$Score(u, v) = m_v \times x_u + c_v + c_u$$

where $m_v = v$,
$x_u = -cSmA(u-1)$,
$c_v = ccSmA(v) + cSmB(v)$,
$c_u = -ccSmA(u-1) + cSmA(u-1) \times (u-1) - cSmB(u-1)$

Beauty and The Array

Solution

- If we want to find the optimal v for a particular u :

$$MAX(u) = \max_{j=u}^{v} Score(u, v)$$

$$MAX(u) = c_u + \max_{j=u}^{v} (m_v * x_u + c_v)$$

- This can be calculated fast using 'Convex Hull Trick'.

Beauty and The Array

Solution

- To handle rotation operations, we will need to also consider scenarios where $u > v$, meaning the subarray considered is from $[u..n]$ and then again $[1..v]$.
- Let $pSmA(i) = \sum_{j=i}^{n} A(j)$
- $ppSmA(i) = \sum_{j=i}^{n} A(j) \times (n - j + 1)$
- $pSmB(i) = \sum_{j=i}^{n} B(j)$
- For a subarray (u, v), where $u > v$, we can write : $Score(u, v) = ccSmA(v) + ppSmA(u) + pSmA(u) \times v + cSmB(v) + pSmB(u)$
- Use 'Convex Hull Trick' similarly to maximize this expression.

Nicks Landing

Problem

Given N persons each having B_i and P_i values. You have to answer Q range queries of the form L R. For each range query, sort the people in $[L...R]$ according to B_i values and calculate a function based on P_i values in that order.

The function is defined as the sum of absolute difference of P_i values between adjacent odd-positioned persons ($OddQ$) and the sum of absolute difference of P_i values between adjacent even-positioned persons ($EvenQ$).

Nicks Landing

Solution

- We don't need to process *OddQ* and *EvenQ* separately.
- The problem can be reduced to finding the sum of absolute difference of P_i between each person and the person next to the person next to him (if exists).
- To apply traditional MO's algorithm here, we need a data structure where we can add new element or remove existing element and maintain active elements in sorted order to efficiently calculate the function value. We can do this with SET in O(log N). The overall complexity will be $O((N+Q)\sqrt{N}\log N)$, which will not pass.

Nicks Landing

Solution

- The key observation is : if we sort all the elements in sorted order of B_i and create a doubly-linked list-like structure from them, it allows us to remove an element from the list in $O(1)$. This idea is also popularly known as the concept of 'Dancing Link'.
- Lets group the queries (L_q, R_q) in $O(\sqrt{N})$ blocks based on L_q and process the queries block by block.

Nicks Landing

Solution

- We need a linked list containing the elements in $[LeftEnd_{CurrentBlock}, N]$ sorted according to their B_i values. This can be built $O(N)$ per block with initial $O(N \log N)$ sorting. It can also be done at just $O(N \log N)$ by processing the blocks from the right to the left.
- Process the queries in a block in decreasing order of R_q.
- Initialize $MO_{Left} = LeftEnd_{CurrentBlock}$ and $MO_{Right} = N$.

Nicks Landing

Solution

- For each query (L_q, R_q), first decrease MO_{Right} until it reaches R_q. Please note, we can remove any element (and its effect to MO_{Ans}) from the linked list in $O(1)$.
- Increase MO_{Left} until it reaches L_q. Removal of an element from linked list can be done in $O(1)$ during this process.
- Current MO_{Ans} value is the desired output for this query.
- Decrease MO_{Left} until it reaches to $LeftEnd_{CurrentBlock}$. This can be done in exact reverse way of increasing MO_{Left}. Since this rollback operation takes place subsequently, we can add back the elements in $O(1)$ so that we reach back the state before MO_{Left} increased.
- Overall complexity : $O((N+Q)\sqrt{N})$.