Day 5: Trademark Contest Analysis

Tyler Marks

Bucharest, Romania

June 25, 2025

C. Corrupted Odyssey

Statement

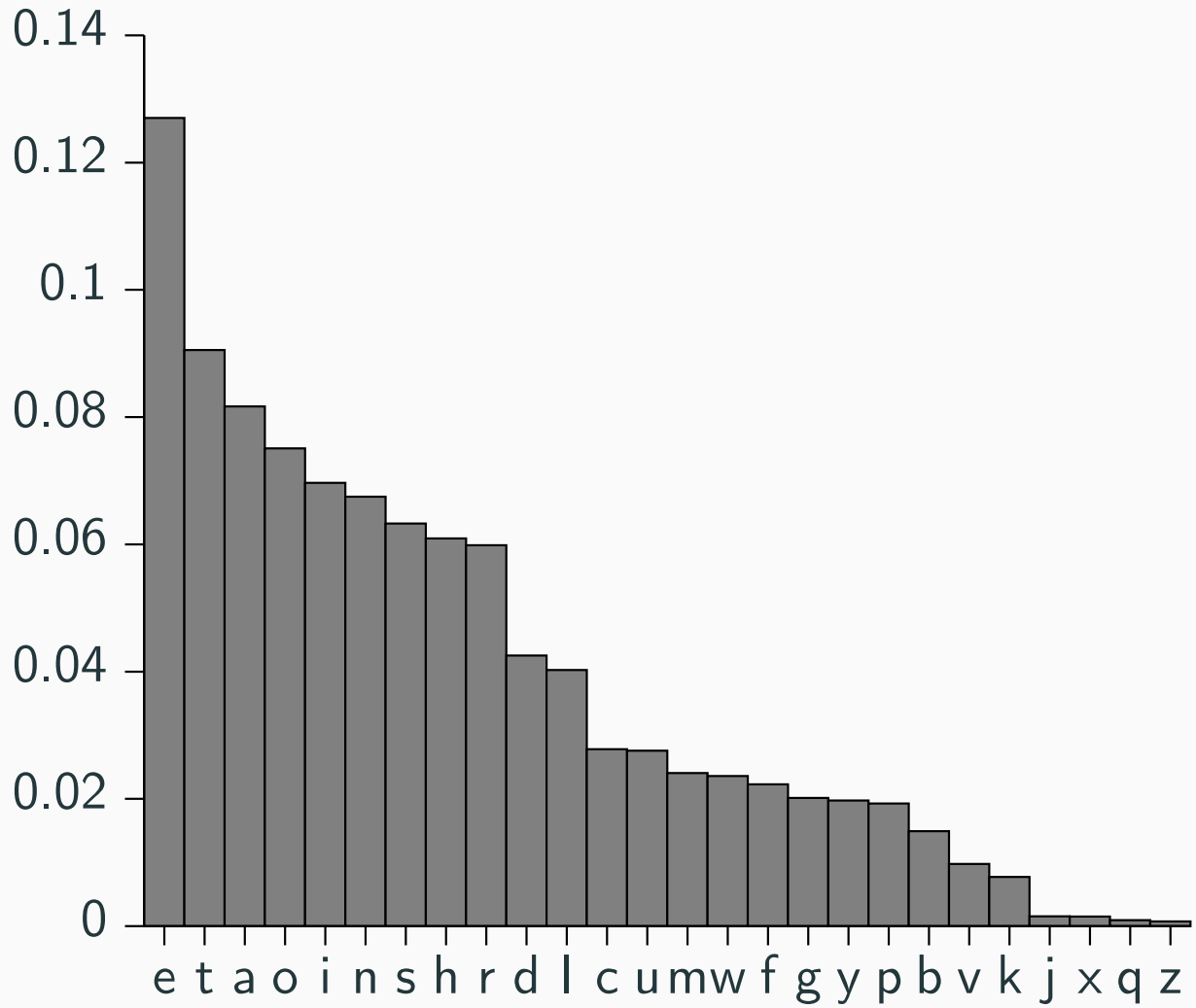
You're given two strings:

- A random string;
- Obscured substring of The Odyssey;

Find out which is which.

Topics: Frequency analysis.

Random string has uniform letter frequencies.
Real English text is typically biased:



There are many ways to use the bias in English text. For instance:

- Compute the following sum for both texts:

$$\sum_{c \in \Sigma} f_c^2$$

  The more uniform the character distribution is, the smaller this sum will be. Therefore, the smaller sum corresponds to the randomly generated text;

- Compute the range for both texts:

$$\max_{c \in \Sigma}(f_c) - \min_{c \in \Sigma}(f_c)$$

  The item with the smaller range correspond to the randomly generated string;

Note: f_c denotes the frequency of character c in the corresponding text.

Time: $O(n)$
Memory: $O(\Sigma)$

H. Shell Shock

Statement

You are constructing a graph on n nodes. Determine the maximum number of shortest paths you can construct.

Topics: Constructive.

To construct the maximum number of shortest paths, we will create "layers" of nodes such that the length of the shortest path is equal to the number layers in the graph.

Each layer can only connect to the layers before and after it, and every node in one layer will connect to every node in an adjacent layer, since adding edges spanning two adjacent layers will only affect the number of shortest paths, not the length of the shortest paths.

The number of shortest paths in this construction is equal to the product of each layer size. Thus we can reduce the problem to the largest product that can be achieved such that sum of the integers in the product is equal to $n - 2$.

The following can be said about the size of each layer:

- Since $x + 1 > x \cdot 1$, it will always be more optimal to combine a layer composed of one node into some other layer;
- Since $2 \cdot (x - 2) \geq x$ for all $x \geq 4$, we can split 2 nodes from any layer with at least 4 nodes, without decreasing the answer. Thus all layers should be composed of 2 or 3 nodes;
- Since $2^3 < 3^2$, it is never optimal to have more than two layers with two nodes;

All together, at most two layers should have 2 nodes and all of the remaining layers should have 3 nodes.

The optimal graph would therefore create layers of two until the number of remaining nodes is divisible by three, then create layers of three until all nodes are used. The only exception to this is when $n = 3$, which has only one extra node forcing any construction to have at most one shortest path.

F. Knapsack One Million

Statement

Items have a uniformly randomly generated weight and value. You want to find the maximum value you can achieve with all knapsacks between 1 and 1000000 capacity.

Topics: Probability Analysis, Dynamic Programming.

Pitfall:

- Greedily only using the best items sorted by the ratio of their value to their weight;
- Verdict: WA or TLE;
- In the case seeded with 126, the only item with weight 1 has a value 44, which makes it the 11450th best item by ratio. This means that taking less than 11450 items will be inaccurate, especially for small capacities, but taking 11450 or more items will be too slow;
- This is just the first instance of a case like this. Several hundred more cases exist in higher seeds;

To correctly solve this problem we will prune items. Item (w_i, v_i) will be pruned if their exists some other item (w_j, v_j) such that $w_i \geq w_j$ and $v_i \leq v_j$.

This pruning will result in $O(\lg n)$ items remaining, allowing us to solve the problem using the standard knapsack DP in $O(n \lg n)$.

One way of approaching the pruning is:

- Sort all items by weight;
- Only take an item if it has a larger value than the previous taken value;

K. Urban Horizons

Statement

You are given a set of graphs, each with n nodes. For each graph you will randomly choose two nodes and connect them with an edge. For each graph, determine the expected number of edges you must add such that the graph becomes a single connected component.

Topics: Dynamic Programming.

For some multiset of edges, E, we have the following recurrence for computing the expected value:

$$f(E) = \frac{1}{n^2} \sum_{u=1}^{n} \sum_{v=1}^{n} 1 + f(E \cup \{(u, v)\})$$

Storing the full graph is going to be to inefficient. Instead we can store the sizes of components.

Two graphs with the same multiset of component sizes have the same expected value and so all of the states can be represented by a partition of N.

Now expressing the DP with a state as a multiset of sorted integers, where each integer is equal to the size of the component:

$$f(P) = \sum_{i=1}^{|P|} \frac{P_i^2}{n^2}(1+f(P))+2\cdot\sum_{i=1}^{|P|}\sum_{j=1}^{i-1} \frac{P_i P_j}{n^2}(1+f((P-\{P_i,P_j\})\cup\{P_i+P_j\}))$$

Solving for $f(P)$:

$$f(P) = \frac{1}{n^2-\sum_{i=1}^{n} P_i}(\sum_{i=1}^{n} P_i+2\cdot\sum_{i=1}^{|P|}\sum_{j=1}^{i-1} P_i P_j(1+f((P-\{P_i,P_j\})\cup\{P_i+P_j\})))$$

Time: $O(cm + p_n n^3 \lg n)$
Memory: $O(p_n)$

Note: p_n denotes the nth partition number. For $n = 22$, $p_n = 1002$, thus this solution is feasible for the bounds in this problem.

I. Stepping Stones

Statement

You have a set of convex polygons representing stones and you are given several queries. In each query, you are given two parallel lines representing a river and an L, R range of stones. For each query, Determine if you can translate each stone in the range such that you can cross the river while remaining within at least one stone.

Topics: Radial Sorts, Extreme Vertex, Data Structures.

For now, let's say we are given a single river and we are allowed to use every stone:

- We want to maximize the effect of each stone, so find the furthest point in both directions along the perpendicular;
  - This can be done with a binary search or an offline sweep;
- The vector spanning these two points corresponds to the vector that contributes the most distance towards the other side (i.e. has the largest projection onto the perpendicular);
- It is possible to cross the river, if and only if this vector added to a point on one river bank brings us to the opposite side of the other river bank;

There are several approaches to solving this problem with queries. The first we will explore is an online approach:

- Adding these vectors together is equivalent to convolving the polygons together first using Minkowski Sums, then finding the extreme vertices;
- Store a merge sort tree where a node covering the range $[lo, hi]$ stores the convolution of all polygons in that range;
  - Each point will be in at most $O(\lg(n))$ nodes;
- For a queried range of polygons, break it into $O(\lg(n))$ ranges, where each range corresponds to a node in the tree;
- For each range, find the extreme vertices (using binary search) of the corresponding node's convolution and add the difference vector to the total;
- Check if the total successfully crosses over the opposite bank;

Time: $O(P \lg n + q \lg n \lg P)$
Memory: $O(P \lg P)$
Where P is the total number of points.

Alternatively, this problem can be solved with an offline approach:

- Each vertex will be the extreme for all perpendicular directions where the angle of the river is between the angles of the two incident sides to the vertex;
- Store events corresponding to when an extreme vertex changes and when a query occurs;
- Sort the events by angle;
- Maintain the differences in extreme points using a Fenwick Tree to be able to easily query ranges of stones;

Time: $O((P+q)\lg(P+q))$
Memory: $O(P+q)$

A. Canonical Palindromes

Statement

You can perform the following operation on an array:

- Pick an index;
- Remove the item at that index and the reflected index and store their values;
- Swap the values;
- Reinsert them into the array such that the distance from the start of the array to one of the items is equal to the distance from the other item to the end of the array;

For each subarray query, output the minimum number of operations to transform the subarray into a palindrome that is initially non-decreasing then non-increasing.

Topics: Strings, Palindromes, LIS.

Observation

It is impossible for the operation to increase the number of indices, i ($1 \leq i \leq n$), such that a_i is equal to a_{n-i+1}. Thus, if the array is not palindromic, then it is impossible to transform it into a canonical palindrome.

Reduction

Since the array must already be palindromic, the operation performs two identical, but mirrored, removal and re-insertions. Thus, making a canonical palindrome is equivalent to sorting the second half of the subarray in non-increasing order using removal and re-insertions.

The number of operations to sort an array in non-increasing order is equal to the size of the array minus the length of the longest non-increasing subsequence.

Pitfall:

- This problem now can be reduced to range LIS, which in the general case is solveable in $O(n \lg^2 n)$ time;
  - The technique used is a lot of code and has a bad constant factor;
- Verdict: TLE;

For more information on this technique, reference the following:

- Semi-local string comparison: algorithmic techniques and applications;
- ko_osaga's codeforces blog on range LIS;

To solve this problem in $O(n \lg n)$ time, the palindromic property of the subarrays must be used.

- Since the subarrays that are able to be transformed are palindromic, there are at most $O(n)$ unique transformable subarrays;
- These unique subarrays must form a tree structure where each node corresponds to a subarray and an edge corresponds to adding a value to the beginning and end of the parent's corresponding subarray;

There are many ways to construct this tree using tools such as hashing or eertrees/palindromic trees.

Once the tree is obtained, we need to find the corresponding length of the longest non-increasing subsequence for each corresponding half subarray in the tree.

- DFS from the root node;
- Each time an edge is traversed, append the corresponding value to the list and attempt to add it to the longest non-increasing subsequence;
- Store a stack of changes to the data structure maintaining the DP;
- After the child's subtree is traversed, undo the addition of the corresponding edge value;

Time: $O(n \lg n + q)$ or $O(n(\lg n + \lg \Sigma) + q \lg n)$
Memory: $O(n)$

B. Cocktail Sort

Statement

You are sorting an array where in each iteration you:

- Do a forward sweep, swapping two adjacent elements if they are inverted;
- Then do a backward sweep, swapping two adjacent elements if they are inverted;

After each iteration of the array, if it is sorted, the algorithm terminates. Determine how many operations are performed if an operation is defined as two elements being compared in a sweep or swapping two elements.

Topics: Sorting Algorithms.

The pseudocode provided in the statement is stable. Thus we can coordinate compress the pairs (a_i, i) to transform the given array into a permutation.

The '`swap`' and '`comp`' functions can be counted independently:

- The number of times '`swap`' will be used is equal to the number of inversions in the permutation;
- The number of times '`comp`' will be used can be computed from the number of iterations needed to sort the array;

So to find the total number of operations, we must compute the number of iterations the algorithm will need to sort the permutation.

For each threshold $t \in [1, n]$, transform the array into a binary sequence by setting all values less than t to 0 and all values greater than or equal to t to 1. These sequences will be referred to as "partitions"

- Let k be the maximum number of iterations needed to sort any partition.
- The cocktail sorting algorithm with k iterations can be represented as a sorting network;
- If all partitions can be sorted in the network, then the original permutation can be sorted as well;
- Thus, the maximum number of iterations to sort any partition is equal to the number of iterations to sort the permutation;
- Proof of this is similar to proving the 0-1 principle for sorting networks;

Let's say we have the following binary sequence:

0	0	0	1	1	1	0	1	0	0	1	0

When performing the forward pass on the binary sequence, the rightmost one of each contiguous block of ones will be moved to the next block of ones or the end of the array if there is no next block.

0	0	0	1	1	0	1	1	0	0	1	0
0	0	0	1	1	0	1	0	0	1	1	0
0	0	0	1	1	0	1	0	0	1	0	1

For the backwards pass, the leftmost zero of each contiguous block of zeros will be moved to the earlier block of zeros.

0	0	0	1	1	0	1	0	0	0	1	1
0	0	0	1	1	0	0	1	0	0	1	1
0	0	0	0	1	1	0	1	0	0	1	1

Thus, each iteration will swap the leftmost one and right most zero.

For each threshold t, there will be $t-1$ zeros, and thus the sorted sequence should have a prefix of $t-1$ zeros followed by $n-t+1$ ones.

The number of these 0/1 swaps needed to sort the sequence will therefore be the number of ones in the prefix of length $t-1$, which can be maintained in a Binary Index Tree.

Time: $O(n \lg n)$
Memory: $O(n)$

E. Just Tree Diameter

Statement

You are given a weighted tree and several updates. In each update, you will add x to every edge's weight in the tree. You want to output the maximal sum of weights of any path in the tree for the initial tree and all queries.

Topics: Centroid Decomposition, Convex Hull Trick.

For one of the paths in the tree, let's define w as the total initial weight among all edges on the path and k as the number of edges in the path.

If we add a total of x to every edge weight in the tree, the new weight can be modeled by the linear function $k \cdot x + w$.

To consider the linear functions of all possible paths, centroid decomposition can be used.

For the current step in the centroid decomposition, recursively compute for each subtree the lines corresponding to the weight of a path from some node in the subtree to the centroid.

If two paths in the same subtree have the same slope, we only care about the path with the greater initial weight, so we can store for each slope, what the maximal initial weight is. Use these lines to construct the convex hull.

For each pair of subtrees, we need to add together the lines in each subtree.

- To add two of our convex hulls, store a pointer to the current x in each line container. For the current pair of pointers, add the two corresponding lines together. Increment the pointers based off of the earliest appearing intersection point;
- Doing this for every pair of line containers is $O(n^2)$. To do this more efficiently, use a divide and conquer;
- Add each of the resulting lines to a global container that will be used to answer the queries;

For each query, store the sum of values added to every edge so far, and query the line container for the answer.

Time: $O(n \lg^2 n)$
Memory: $O(n)$

G. Puzzle Boy

Statement

Given a set of cards with positive or negative integers on them, choose the maximum number of cards such that no subset of chosen cards has a product whose absolute value is a perfect square. For all possible choices, determine the maximum product you can achieve.

Topics: Matroids, XOR Basis.

For a number to be a perfect square, each power in the number's prime factorization must be even.

Multiplying two numbers is equivalent to summing the powers of each prime factor, but since we only care about the parity, we can maintain this sum under modulo two. That is, each number can be viewed as a bitmask where a bit is on if the corresponding prime occurs an odd number of times and some product is a perfect square if the xor of the corresponding masks is 0.

For their to be no product whose absolute value is a perfect square, no mask should be able to be represented by the XOR of some subset of other bitmasks in the set. Thus the optimal set must form an XOR basis.

For now, let's only consider positive numbers:

- We can view this problem as finding the largest set of independent bitmasks such that the sum of their weights is maximal;
  - Since our goal is to maximize the product of chosen values, it is equivalent to maximizing the sum of the logs of chosen values;
  - Thus the weight of each mask can be viewed as the log of the corresponding value;
- Linearly independent vectors can be represented as a matroid, thus we can greedily add the items to the basis in descending order of weight;

Now for handling negative numbers:

- First, construct the basis, maximizing the magnitude of the product of items taken;
- If this product is already, positive, the answer has been found;
- Otherwise, we must make the product positive by swapping some items in the basis;
  - It can be proven that it is always suboptimal to swap 2 or more elements;
- To swap an element in the basis with an element not already in the basis, find the linear combination of basis elements that equals the element being added in;
  - The item originally in the basis and the item trying to be swapped in must have opposite signs to change the sign of the overall product;
- Of all potential swaps, the optimal swap will have a ratio as close to 1 as possible;
- If no swaps are possible, the answer must be negative. Thus, the magnitude of the product must instead be minimized, which can be done with the greedy basis construction;

D. Dots and Boxes?

Statement

Two people are playing a game on a grid of dots. In one move, two adjacent dots can be connected with a segment. When some contiguous set of cells is enclosed by some segments, that region can no longer be played in. Once a player can't make a move, they lose. Given the initial state of the board, determine who wins.

Topics: Grundy Numbers, BCCs.

Reduction

Construct the dual graph of the grid:

- Create a node for every cell. If the segment between two adjacent cells has not been created yet, create an edge between the two corresponding node;
- Create a node representing the outside. Any cell on the edge of the grid, that doesn't have a segment drawn on the border, gets connected to the outside node;

The problem is now equivalent to Green Hackenbush!

Let's begin by solving this problem where the dual graph is a tree:

- Start at the root node of the tree;
- Recursively compute the Grundy Number for each child's subtree;
  - If the current node has no children (i.e. it is a leaf), the Grundy number is 0;
- Each child will correspond to a pile in the current node's nim game. The size of the pile will be equal to the child's Grundy number plus one to represent the edge connecting the child to the current node;
- Our Grundy number will be the xor sum of each of the pile sizes that have been constructed;

To solve this problem for a general graph:

- A simple cycle of k_1 nodes (and k_1 edges) can be collapsed into a single node with k_1 self loops attached to it.
- After collapsing one cycle, if the new node is still apart of another cycle with k_2 additional edges, we can again collapse the cycle into a single node, now with $k_1 + k_2$ self loops. This can keep propagating until no cycles remain;
- Thus, any 2-edge connected component can be collapsed into a single node where each edge originally in the component becomes a self-loop on the new node;
- After all 2-edge connected components are collapsed, a tree remains where some nodes have self-loops;
- We can perform the same Grundy number computation as in the tree case, where each self loop is treated as a pile of size 1 for the node it is attached to;

Time: $O(nm)$
Memory: $O(nm)$

L. Yet Another XOR Problem

Statement

You are going to add circles to the coordinate plane. After each circle is added, determine the area of the set of points covered by an odd number of circles.

Topics: Green's Theorem, Segment Trees.

We will use Green's theorem to relate line integrals and areas:

$$\oint_C P\,dx + Q\,dy = \iint_R \frac{\partial Q}{\partial y} - \frac{\partial P}{\partial x}\,dA$$

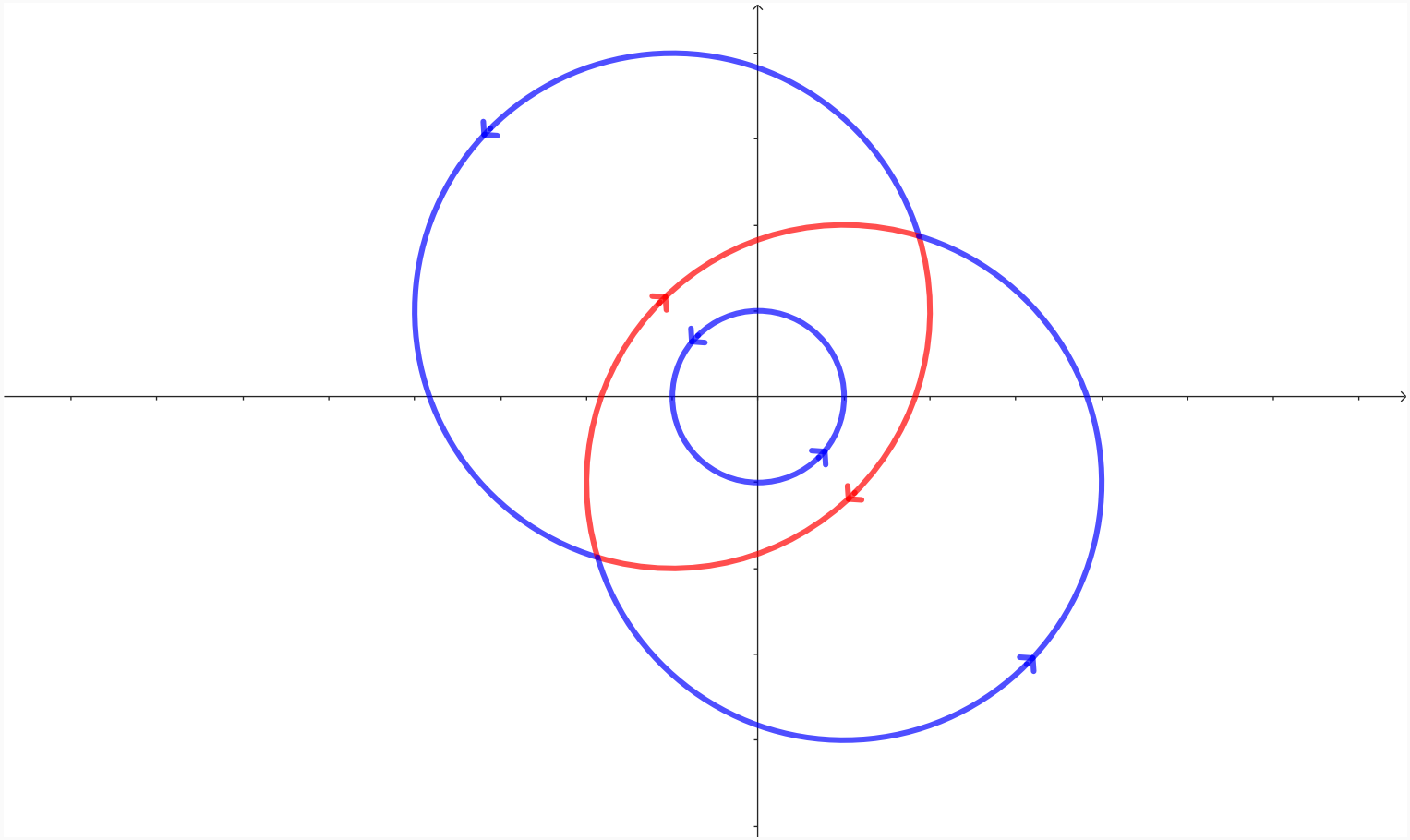
To relate this to area, any vector field that results in $\frac{\partial Q}{\partial y} - \frac{\partial P}{\partial x} = 1$ is viable. For ease, let's use $\langle P, Q\rangle = \langle 0, x\rangle$. This means we can represent the area as:

$$A = \oint_C x\,dy$$

Using the linearity of line integrals, we can express the area as the sum of line integrals over each arc composing the shape. For a single arc centered at (c_x, c_y) with radius r and spanning the angles $[\theta_1, \theta_2]$:

$$\oint_{C_i} x\,dy = [c_x r \sin\theta + \frac{r^2}{2}(\theta + \frac{1}{2}\sin(2\theta))]_{\theta_1}^{\theta_2}$$

The curve forming outer boundaries of the shape will be positively oriented, while boundaries forming holes will be negatively oriented.



Any arc that is within or on an odd number of lenses (excluding the lens it is derived from) will be negatively oriented. Otherwise, it will be positive oriented.

Changing orientation is equivalent to negating the value of the line integral.

When two circles intersect, a contiguous range of arcs will be negated, so use a segment tree to allow for range negation, range query. Each circle will have it's own segment tree.

So for each update:

1. Go through all pre-existing circles;
2. Find the range of arcs in the pre-existing circles that get covered and negate all arcs in that range;
3. Find the range of arcs in the circle being added that are covered and negate all arcs in that range;
4. Find the area by checking the sum stored in each segment tree for the entirety of each circle;

Time: $O(n^2 \lg n)$
Memory: $O(n^2)$

Be wary of:

1. Tangential Circles;
2. Duplicate Intersection Points;
3. Duplicate Circles;
4. Nested Circles;
5. Loss of Floating Point Precision;

J. Techno-Trains

Statement

You're given n and k.
Count simple connected graphs s.t.:

1. They have n vertices;
2. They have at most 1 cycle;
3. Some non-intersecting simple paths are colored in one of k colors;

Topics: Generating functions.

Define T_i to be the number of valid rooted trees such that the root of the tree can connect to i $(0 \leq i \leq 2)$ children with a colored edge.

The following is true for their EGF:

$$\begin{cases} T_0 = x \exp T_2, \\ T_1 = (1 + T_1) T_0, \\ T_2 = (1 + kT_1 + \frac{kT_1^2}{2}) T_0 \end{cases}$$

We can find T_0, T_1 and T_2 at once with "online FFT" in $O(n \lg^2 n)$, or Multi-Dimensional Newton's Method in $O(n \lg n)$. Note that the Newton's Method approach uses significantly more FFT calls, so it will have more difficulties with time limit.

Every valid graph is one of the following:

- An "unrooted" valid tree;
- A cycle consisting of valid rooted trees T_2 and "blocks";

EGF for the "unrooted" valid trees U:

$$[x^n]U = \frac{1}{n}[x^n]T_2$$

EGF form:

$$U = \int \frac{T_2}{x} dx$$

A "block" B is a connected sequence of trees:

- Starts with a T_1;
- Contains a few intermediate T_0;
- Ends with a T_1;

EGF for blocks:

$$B = k \cdot \frac{1}{1 - T_0} \cdot T_1^2$$

The following cycles are forbidden:

- Single rooted tree T_2;
- A pair of rooted trees $\frac{T_2^2}{2}$;
- A single block with no intermediate trees $\frac{T_1^2}{k}$;

Divide by 2 for cycle direction:

$$A = U + \frac{1}{2}\left(-\ln(1-(B+T_2)) - T_2 - \frac{T_2^2}{2} - \frac{T_1^2}{k}\right)$$

Time: $O(n \lg^2(n))$ or $O(n \lg(n))$
Memory: $O(n)$