Problem Tutorial: "Total Eclipse"

The vertices with the larger brightness will be removed later than those with the smaller brightness. Let's reverse the whole procedure: Sort vertices by brightness in non-increasing order, add vertices to the graph one by one, and merge vertices into connected components. When a vertex x is added, for each edge (x, y), if y is already added and y is not connected with x, merge x and y into a connected component, and set the parent of the root of the component y to x using DSU.

Finally we will get several rooted trees. For a vertex x, it will be operated for b_{parent_x} times before b_x becomes the minimum brightness. Thus $answer = \sum_{i=1}^{n} b_i - b_{parent_i}$.

Overall time complexity is $O((n + m) \log n)$.

Problem Tutorial: "Visual Cube"

Calculate the size of the picture and the locations of each object carefully, then you can draw the intended picture.

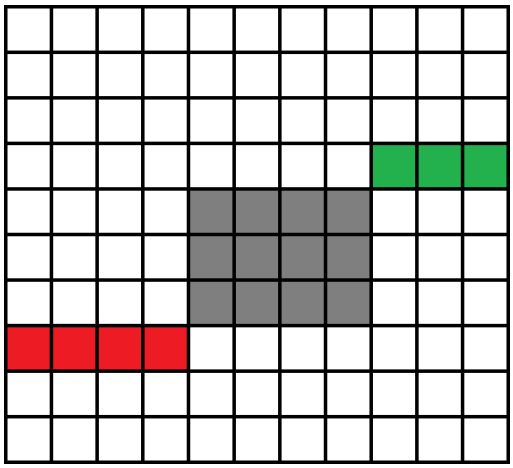
Problem Tutorial: "Count on a Tree II Striking Back"

It is well-known that the expected value of the minimum value among k uniform random real numbers in $[0, 1]$ is $\frac{1}{k+1}$. For a set T of size k, if we assign a random integer weight to each element in T, get the minimum weight of these k integers, and run this procedure several times, the smaller average minimum weight will show the larger value of k.

In this problem, assign a random integer weight to each color, find the minimum weight on the chain, and run this procedure k times. This way, we can know which chain has the larger number of distinct colors.

Overall time complexity is $O(nk + mk \log^2 n)$.

Problem Tutorial: "Diamond Rush"



Assume the gray area is blocked, we can find that we will always go through a red cell or a green cell. Let $f_{i,j}$ denote the best path going through the cell (i, j), $g_{i,j} = \max(f_{i,1}, f_{i,2}, \ldots, f_{i,j})$, and $h_{i,j} = \max(f_{i,j}, f_{i,j+1}, \ldots, f_{i,n})$. Then we can answer a query using g and h in $O(1)$ comparisons.

Note that the value of a path may be extremely large, but fortunately we can store them using persistent segment tree with hashing on each node, such that we can copy in $O(1)$, and modify a bit or do a comparison in $O(\log n)$.

Overall complexity is $O((n^2 + q) \log n)$.

Problem Tutorial: "New Equipments"

Let's build a directed graph with $n + |R| + 2$ vertices (n workers, $|R|$ pieces of equipment, the source S and the sink T) and the following arcs:

- $S \to$ each worker, the capacity is 1 and the cost is 0.
- Each piece of equipment $\to T$, the capacity is 1 and the cost is 0.
- For the i-th worker, find the smallest n values of $a_i \times j^2 + b_i \times j + c_i$, add the corresponding j into set R, add an arc $i \to j$, the capacity is 1 and the cost is $a_i \times j^2 + b_i \times j + c_i$.

Now we will get a tiny graph, run MCMF (minimum cost maximum flow) on it and get the answer. There is always an optimal solution only using these pieces of equipment.

Problem Tutorial: "The Missing Pet"

Denote $s_{i,j}$ as all possible routes ending at (i, j), $p(k)$ as the possibility of route k, $len(k)$ as the length of route k, $f_{i,j} = \sum_{k \in s_{i,j}} p(k)$, and $g_{i,j} = \sum_{k \in s_{i,j}} p(k) \cdot len(k)$. The answer will be g, and we can calculate g using f and Gauss-Jordan Elimination. Calculating f is similar to g, we need to use Gauss-Jordan Elimination. There will be $O(n^2)$ variables and $O(n^2)$ equations, the straightforward algorithm is $O(n^6)$, which is unfortunately too slow to pass this problem.

Let's treat the cells with a hole and the cells in the first row as **important cells**. It turns out that, if the values of important cells are known, we can calculate the values of all the remaining cells using those equations. Assume there are cnt ($cnt \le n + k$) important cells, finally there will be exactly cnt unused equations, just run Gauss-Jordan Elimination on it.

Overall complexity is $O((n + k)^3)$.

Problem Tutorial: "In Search of Gold"

Binary search for the answer, now we just need to check whether there is a tree whose diameter is not larger than mid.

Let's set the root of the tree to 1. For the subtree of vertex i, its diameter should not exceed mid, and we want to minimize $d(i)$: the distance between i and the farthest vertex in this subtree. Let $f_{i,j}$ ($j \le \min(k, size(i))$) denote the minimum value of $d(i)$ such that there are exactly j values taken from a, the transition is straightforward.

Overall time complexity is $O(nk \log ans)$.

Problem Tutorial: "Dynamic Convex Hull"

Let's answer all queries offline. We can find that each function can update a continuous subsequence of queries. Assume there are q queries, build a segment tree on $[1, q]$. For each function, find the range of queries it can update, and insert it as a tag into $O(\log q)$ nodes on the segment tree.

Now for each node of the segment tree, we will have several functions and several queries. We want to find the answer for each query just using these functions. There are two cases: $a_i \le x$ and $a_i \ge x$.

Take $a_i \le x$ for example: Sort functions by a_i in non-decreasing order. Denote $best(x)$ as the best function for query x. it can be proved that $best(x) \ge best(x - 1)$, so we can use the well-known divide-and-conquer trick to find all the values of $best(x)$ in $O(B \log A)$. Here, A denotes the number of queries and B denotes the number of functions.

Overall time complexity is $O((n + m) \log^2 m)$.

Problem Tutorial: "It's All Squares"

For each query, let's find the bounding box of the polygon, assume its size is $r \times c$, we can easily find the answer in $O(rc)$.

In the worst case, all the polygons are squares, a square of size $k \times k$ will consume $4k$ characters of the input, so the overall complexity is $O(\frac{n|S|}{4})$.

Problem Tutorial: "Walking Plan"

Let $G_{i,j}$ denote the length of the shortest one-way street from i to j, and let $f_{t,i,j}$ be the length of the shortest path from i to j with exactly t streets. Then we have $f_{t,i,j} = \min\limits_{1 \le k \le n} (f_{t-1,i,k} + G_{k,j})$. We can also find that $f_{t,i,j} = \min\limits_{1 \le k \le n} (f_{x,i,k} + f_{t-x,k,j})$. If we treat them as matrices, we have $f_t = f_x \cdot f_{t-x} = G^t$.

Note that $k_i \le 10\,000$, let $A = \lfloor \frac{k}{100} \rfloor$ and $B = k \bmod 100$. We can split each query (s, t, k) into two phases:

- Walk from s to some vertex u, passing exactly $100A$ streets.
- Walk from u to t, passing at least B streets.

Let $a_i = G^{100i}$ and $b_i = G^i$. Then a_i is the length of the shortest path with exactly $100i$ streets, b_i (after running an external Floyd-Warshall) is the length of of the shortest path with at least i streets. Now we can answer a query by trying all possible values of u, thus $ans = \min\limits_{1 \le u \le n} (a_{A,s,u} + b_{B,u,t})$.

Overall time complexity is $O(n^3\sqrt{k} + qn)$.

Problem Tutorial: "King of Hot Pot"

We will show that we can construct the optimal solution incrementally. In other words, there exists a permutation $ord_1, ord_2, \ldots, ord_n$ such that the set $\{ord_1, ord_2, \ldots, ord_k\}$ is an optimal solution for eating k dishes of meat.

Assume we have already known which set of dishes to eat, we will always eat them in non-decreasing order by a_i. Initially $ord = \{\}$, let's insert all the dishes into ord in non-decreasing order by a_i.

Assume now we are going to insert a dish of meat (a, b). Let t_i be the time to finish the first i dishes $ord_1, ord_2, \ldots, ord_i$. If $\max(t_{j-1}, a) + b < t_j$, it means that replacing ord_j by the current dish (a, b) will get a better solution. Note that if $\max(t_{j-1}, a) + b < t_j$ holds, $\max(t_j, a) + b < t_{j+1}$ also holds, so we just need to binary search for the smallest value of j, and insert the current dish (a, b) before ord_j. After insertion of (a, b), we should fix a suffix of t into $\max(t, a) + b$, which means eating one more dish of meat (a, b) in each solution.

In order to speed up the algorithm, we can store ord and t in a binary search tree with lazy tag. Finally the sequence t is the answer. Overall complexity is $O(n \log n)$.

Problem Tutorial: "String Distance"

It can be proved that the insertion is useless, so $dis(A, B) = |A| + |B| - 2\mathrm{LCS}(A, B)$. For each query, let $f_{i,j}$ denote the minimum possible value of k such that $\mathrm{LCS}(B[1..i], A[l..k]) = j$. Then we can find the LCS in $O(m^2)$ for each query.