Day 6 Contest: Grand Prix of Xiaomi

Moscow International Workshop 2020

December 4, 2020

A. Rikka with Game

Keywords: games, graph matchings.

Each player has a safe "never attack" strategy, thus they will only attack if they won't be killed next turn.
For a set of heroes S and a dragon v, let $f(S, v) = 1$ if v will be attacked, and 0 otherwise.
Let E_v be the set of non-friends of v. Then we have $f(S, v) = 1$ iff there is such a $u \in S \cap E_v$ that $f(S - u, u) = 0$.
Observe that $f(S, v) = 1$ also describes winning positions in the *Undirected Vertex Geography (UVG)* game: there is a token in one of the vertices of an undirected graph G. The token can be moved to any of the neighbours of v, but can not visit the same vertex twice. Two players take turns moving the token, and the player who can not move loses.

Proposition. Vertex v of a graph G is a winning starting position in UVG iff v is covered by all maximum matchings of G.
Proof. If a maximum matching M doesn't contain v, the second player can move along edges of M. If M doesn't exist, the first player can choose any matching M' and move along edges of M'. If any of these strategies fails, we can find an alternating chain that violates one of the premises (either M is not maximum, or M' can be transformed so that it doesn't cover v).
All suitable v can be found together with a maximum matching by Edmonds' blossom algorithm in $O(n^3)$.

B. Rikka with Maximum Subsegment Sum

Keywords: divide-and-conquer, sweep-line.

Let us find the sum of answers for subsegments of $[L, R)$. Find the sum for subsegments crossing the middle point $M = (L + R)/2$, and add answers in $[L, M)$ and $[M, R)$ recursively.
For each range $[i, M)$ find its largest subsegment weight a_i and largest suffix weight b_i.
Similarly, for each range $[M, j)$ find its largest subsegment weight c_j and largest prefix weight d_j.
Now, for a range $[i, j) \subseteq [L, R)$ the answer is $\max(a_i, c_j, b_i + d_j)$.

Let us find the total answer for ranges such that $a_i \geqslant c_j, b_i + d_j$. This is equivalent to $a_i \geqslant c_j$, $a_i - b_i \geqslant d_j$.
In the plane create red points $(a_i, a_i - b_i)$ and blue points (c_j, d_j). Now for each blue point (x, y) let us find total a_i of all red points to its top-right.
This is a standard sweep-line with RSQ. The other two cases are processed similarly (take care of tie-breaking in max to avoid double-counting).
Complexity is $O(n \log^2 n)$.

C. Rikka with Random Tree

Keywords: probabilities, DP.

We want to compute the expected value of $\sum_v w_v(n - w_v)$, where w_v is the weight of subtree of v.
Consider vertices $v, 2v, 3v, \ldots$. Let p_i be the probability of the vertex iv being in the subtree of v. $p(1) = 1$.
$p_i = \frac{1}{\sigma_0(i)-1} \sum_{j|i} p_j$ — we have to connect to a multiple of v, provided that it itself is in the subtree of v. Here $\sigma_0(i)$ is the number of divisors of i.
To compute expected value of $w(v)^2$, group together pairs of i, j by their GCD, and compute the total coefficient $\sum_{i,j} \frac{1}{(\sigma_0(i)-1)(\sigma_0(j)-1)}$.
For each GCD k, find the probability $p_2(k)$ that iv, jv are in the same subtree when $\mathrm{GCD}(i, j) = k$. Consider all options of connecting iv, jv to $i'v, j'v$. These can be separated by GCD of i', j'. With this we can obtain another recurrence for p_2.

D. Rikka with New Year's Party

Keywords: hashes, suffix array.

Let ord_s be unique characters of s in order of their appearances from left to right. Let $canon(s)$ be the string s, where all occurences of $ord_s[i]$ are replaced with i.
First, let us find ord_{suf_i} for all i, where suf_i is the i-th suffix of s. ord_{suf_i} can be obtained from $ord_{suf_{i+1}}$ simply by moving s_i to the beginning.

We want to build a "suffix array" of $canon(suf_i)$. To do that, we will just sort $canon(suf_i)$ with a fast lexicographic comparison.
To compare $canon(suf_i)$ and $canon(suf_j)$, let us find the largest common prefix with binary search and hashes.
To find the hash of $canon(suf_i)$ let us store prefix sums of $Q^j \bmod P$ for all occurences of c.
Then the hash of suf_i is equal to $\sum_p p \cdot h_{i,ord_s[p]} \bmod P$, where $h_{i,c}$ is the total hash of all occurences of c in suf_i.

Since we have the "suffix array", and can compute LCPs, the number of distinct substrings is found as total length of suffixes, minus LCP lengths of all adjacent pairs in the suffix array.
Lexicographic comparison actually takes $O(\alpha \log n)$, where $\alpha = 26$. Thus, the **complexity** is $O(\alpha n \log^2 n)$.

E. Rikka with Subsequence

Keywords: ad-hoc.

If x is even, then $a = b = c = x/2$ is optimal.
Start by taking $a = \lfloor x/11 \rfloor$. Then $b = x - a = 10a + (x \bmod 11)$. If $x \bmod 11 \neq 10$, then $c = a$, and this answer is optimal.
Let 10^d be the largest digit of x. If $x \geqslant 2.1 \cdot 10^d$, we can subtract 10^d from x (which changes $x \bmod 11$), apply the same solution, and then add 10^d to a.
If x starts with $20\ldots$, we can do the same to $x - 2 \cdot 10^d$, and add 10^d to both a and b.
If x starts with 1, the best $len(c)$ we can hope for is $len(x) - 2$. We can do this by again applying the solution to $x - 9 \cdot 10^{d-1}$, and adding $9 \cdot 10^{d-1}$ to a (this will just put 9 as highest digit of a).

Presumably, many other constructions are possible.

F. Rikka with Lake

Keywords: shortest path, unions & intersections.

Special points = vertices of the polygon, and the start $(0, 0)$.
Find the shortest distance to travel from $(0, 0)$ to each of the special points. Any reachable point can be reached by following the shortest path to a special point and then going straight (without intersecting the polygon), thus we have an oracle "if an arbitrary point is reachable".
If the distance to a special point p is $d < k$, then the region reachable from p is a part of the circle C of radius $k - d$ visible from p.
Construct a planar graph of all polygon sides and circles subdivided by pairwise intersections. Any face/edge of this graph is either completely inside the boundary (then the shortest distance to any point is $< k$), completely outside the boundary (the shortest distance to any point is $> k$), or on the boundary (the distance to any point is exactly k).
Now we can either find the total area of reachable faces, or the total directed area under boundary edges.
Complexity: who cares?

G. Rikka with Game Theory

Keywords: subset DP.

Suppose we have placed all values of $f = 0, \ldots, k-1$, and the set of labelled vertices is S. We must have that any vertex outside of S has at least one neighbour with $f = 0, \ldots, k-1$, since it has $f \geqslant k$.
Consider options of choosing a subset S' and placing k in all its vertices. S' is valid if:

- S' is an independent set (no vertices of S' are adjacent);
- any vertex outside of $S \cup S'$ has a neighbour in S'.

Note that this criterion is independent of the exact value of k.

We can now compute dp_S — the number of ways to legally assign numbers $0, \ldots, k-1$ to vertices of a subset S for some k.
Transitions are $S \to S \cup S'$ as described above. To check the two conditions described above, precompute all

independent sets, as well as sets of neighbours of all subsets S'.
Complexity is $O(3^n)$.

H. TODO

I. Rikka with RCPC

Keywords: RMQ.

Let dp_i be the lowest damage we can receive in first i days if we answer messages on the last day. Consider transitions from dp_j. We are supposed to have $A = sa_i - sa_j$ at day i, where sa_i are prefix sums of a_i.

- If $sa_i - sa_j \leqslant T$ and $i - j < k$, the cost is $dp_j + sa_i - sa_j$.
- If $sa_i - sa_j \leqslant T$ and $i - j \geqslant k$, the cost is dp_j.
- If $sa_i - sa_j > T$ and $sa_{i-1} - sa_j \leqslant T$, the cost is $dp_j + 2(sa_i - sa_j)$.
- If $sa_{i-1} - sa_j > T$, the transition is illegal since A would overflow before day i.

This gives at most three interesting ranges of j, and in each range the cost is $k \cdot sa_i + (dp_j - k \cdot sa_j)$, where $k = 0, 1, 2$. We can either maintain RMQs of $dp_j - k \cdot sa_j$ for each k, or notice that endpoints of each range are monotonous as i increases. The latter allows us to maintain window maximums of $dp_j - k \cdot sa_j$ with queues.
In the end, try all unanswered suffixes and calculate damage accordingly using dp_i.
Complexity is $O(n)$. $O(n \log n)$ is probably fine too.

J. Rikka with Book

Keywords: subset DP.

Let dp_S be the largest overhang when books in subset S are stacked in some order.
Consider placing the stack S on top of another book x. It is optimal to place S so that it overhangs the book x by dp_S.
Also, the center of mass of the new stack has to be above the table edge. The shift depends only on the length and mass of the book x, and total mass of S.
With this, largest answer can be found in $O(n2^n)$ with subset DP.

K. Rikka with Composite Number

Keywords: brute-force.

There are less than 200 prime numbers reachable in the process, and all of them are less then $2 \cdot 10^9$. We can just brute-force the possibilities, and check if numbers are prime in $O(\sqrt{n})$.

L. Rikka with Generals

Keywords: DP.

Suppose that x and $x + 1$ were swapped first. They cannot be swapped again without complete backtracking. After the swap the vertex containing x can be swapped with any adjacent $x - 1, x - 2, \ldots$ in order. Similarly, $x + 1$ can be swapped with $x + 2, \ldots$.
For each initial swap $x, x+1$ find the largest range $[l, r]$ so that x can be swapped with l, and $x+1$ can be swapped with r. The ranges have to be independent for different pairs $x, x + 1$.
The number of combinations of disjoint ranges can be with DP.

M. Rikka with Employees

Keywords: divide-and-conquer.

For each subtree, we have to reach a state when only vertices in this subtree are present.
Sort vertices by their in-time in a DFS. Instead of subtrees, we now have to reach several subsegments.
We will use divide-and-conquer. Suppose we are in a state when only elements in a range $[L, R)$ are present. If $R - L = 1$, we don't have to do anything (other than possibly register a single-element segment).
Otherwise, consider all segments we need to reach that contain both M and $M+1$, for $M = \lfloor (L+R)/2 \rfloor$. All these segments are nested, thus we can register them by decreasing of their length, by gradually removing elements from left and right.
Now, fill the segment $[L, R)$ back. All yet unreached subsegments of $[L, R)$ are either inside $[L, M)$ or inside $[M, R)$. Process both options recursively: remove one half of the segment and process the other.

On each level of recursion each element will be touched at most four times. Thus, the number of queries is $\sim 4n \log_2 n$. Further optimizations are possible, but not needed.