# ACM-HK Programming Contest 2023

## Contest Analysis

Zhejiang University

06.10.2023

# A. Putata Strikes Back

## Description

Given three string sequences $P, Q, R$. Find the number of pairs of string $(A, B)$ such that $A$ is a prefix of $P_i$, $B$ is a prefix of $Q_j$ and $AB$ is a substring of $R_k$.

# A. Putata Strikes Back

## Solution

Build Aho-Corasick Automaton on $P$ and $Q^R$, here $Q^R$ means reverse every string in $Q$. Let's denote them as $A_P$ and $A_Q$.
For string $S \in R$ of length $l$, assume $pre(S, i), suf(S, i)$ means $S_1 S_2 \dots S_i$ and $S_i S_{i+1} \dots S_l$.
Assume for some $i$, $pre(S, i)$ matches node $x$ in $A_p$ and $suf(S, i+1)$ matches node $y$ in $A_q$, then all pair of nodes $(u, v)$ such that $u$ is the ancestor of $x$ on the fail tree of $A_P$ and $v$ is the ancestor of $y$ on the fail tree of $A_Q$ represents a legal $(A, B)$.

# A. Putata Strikes Back

## Solution

Then the problem is reduced to, given two trees and several pairs of nodes $(x, y)$, count the number of $(u, v)$ where there exist a given pair $(x, y)$ such that $u$ is ancestor of $x$ in the first tree and $v$ is ancestor of $y$ in the second tree.

Let's enumerate $u$, then for all pairs $(x, y)$ such that $x$ is in subtree of $u$, the number of possible $v$ is equal to the number of nodes in the union of path between $y$ and root. Assume such union is $S(u)$, then we can get $S(u)$ by merging all $S(t)$ such that $t$ is a son of $u$. Thus we can solve this by heuristic merging (small to large technique) or segment tree merging. The time complexity is $\Theta(n \log^2 n)$ or $\Theta(n \log n)$.

# B. Xor is Add

## Description

Find a permutation $p$, where $p_i \oplus i = p_i + 1$.

# B. Xor is Add

## Solution

Since $p_i + i = 2 \cdot (p_i \ \& \ i) + (p_i \oplus i)$, then the constraint equivalent to $p_i \ \& \ i = 0$.

Consider the most significant bit of $n$. Assume it is $k$, then we can pair all $x$ such that $x \geq 2^k$ with $y = 2^{k+1} - 1 - x$, which means $p_x = y, p_y = x$.

And the problem is reduced to the same problem with a smaller $n' = 2^{k+1} - n$. The reduction ends when $n = 1$ or $n = 0$.

The time complexity is $\Theta(n)$.

# C. Permutation Compression II

## Description

Given a permutation of length $n$. You can delete arbitrary number of elements which is a prefix maximum. Find the smallest number of elements you will delete so that the number of prefix maximums is maximized.

# C. Permutation Compression II

## Solution

We can get any increasing subsequence of the permutation after some operations. We can delete the elements one by one from front to back that are not in the increasing subsequence but have the largest prefix value. Then the answer must be the longest increasing subsequence of the permutation. Assume the LIS is $i_1, i_2, \ldots i_k$, then for all $i_t \leq u \leq i_{t+1}$, if $p_u \geq p_{i_t}$, we have to delete $p_u$, so the number of deletion is determined, and it is equal to the number of such elements. We can use DP to find the LIS with smallest number of such elements, or we can also notice that we just have to greedily select the LIS appearing in the permutation as early as possible. Since this can make $p_{i_t}$ as large as possible, it is correct. The time complexity is $\Theta(n \log n)$.

# D. Simple Calculation

## Description

Given $n, k, t$. Define $g(x)$ as $k \cdot \phi(x)$. Calculate $g^{(t)}(n)$.

# D. Simple Calculation

## Solution

Since $\varphi$ function is a multiplicative function, we can consider each prime factor individually.

For each prime factor from $n$ and $g^{(\cdot)}(n)$, it will disappear after at most $2\log(n)$ iterations. The remaining effect of $k$ to the result (after $2\log(n)$ rounds) is a geometric progression. So we can simulate the first $\min(2\log(\max(n,k)),t)$ iterations and do the remaining rounds by using quick power.

A typical implementation is to use `std::map` to maintain the prime factors and their factorization results.

The time complexity is $\mathcal{O}(\sqrt{n}\log(n)+\sqrt{k}\log(k)+\log(t))$

# E. Surreal Number

## Description

There are $n$ piles of stones. Putata can take any positive amount of stones from one pile each turn, and Budada can take one stone each turn. Putata goes first. Person who takes away the last stone wins. Find the winner under the best strategy.

# E. Surreal Number

## Solution

If every pile has only one stone then the result is fixed.
Otherwise Putata can use his first operation to make the total number of remaining stones even. After that he can take 1 stone each time to win.
Time complexity $\Theta(n)$.

# F. Yet Another Modify and Query Problem

## Description

Given a sequence, support the following operations.

- Change $A_x$ into $y$.
- Query the smallest $i \in [l, r]$ to replace $A_i$ with $v$, so the relative size between all adjacent elements in $A$ will not change.

# F. Yet Another Modify and Query Problem

## Solution 1

For each $i$, the number that $A_i$ can be replaced by is an interval. There are four cases:

- $A_{i-1} \leq A_i \leq A_{i+1}$: $v \in [A_{i-1}, A_{i+1}]$.
- $A_{i-1} \leq A_i > A_{i+1}$: $v \in [\max(A_{i-1}, A_{i+1}+1), \infty]$.
- $A_{i-1} > A_i \leq A_{i+1}$: $v \in [-\infty, \min(A_{i-1}-1, A_{i+1}])$.
- $A_{i-1} > A_i > A_{i+1}$: $v \in [A_{i+1}+1, A_{i-1}-1]$.

Let $g(i)$ be the interval of $A_i$, noticing for an interval $[l, r]$, the union of $g(i)$ that $i \in [l, r]$ is at most two intervals. Use segment tree to maintain such intervals and do binary search on segment tree, the time complexity is $\Theta(n \log n)$.

# F. Yet Another Modify and Query Problem

## Solution 2

Assume we want to replace some element in $[l, r]$, we only have to consider the first three consecutive monotonic sequence.
The proof is a bit complex, the main idea is to discuss all the situations.
To implement this, use `std::set` to maintain all position $i$ such that $A_{i-1} \leq A_i > A_{i+1}$ or $A_{i-1} > A_i \leq A_{i+1}$, and check the first four such positions and binary search on each monotonic sequence segmented by these positions.
The time complexity is also $\Theta(n \log n)$.

# G. Tag Game

## Description

There is a directed graph. Each edge can be went through in two ways:
1. use $t$ time to go through, but there is a $\frac{p}{100}$ probability to be teleported to vertex $1$ after going through the edge.
2. use $c$ time to go through.

Calculate the minimum expected time for Putata to move from vertex $1$ to vertex $n$.

# G. Tag Game

## Solution

When standing on a vertex, the best strategy will always be the same. So if Putata has been teleported to vertex $1$, then he should still follow the same path back. This part can be calculated by using the sum of geometric progression formula.

The whole process can be maintained by using Dijkstra's Algorithm. The time complexity is $\Theta(m\log(n))$.

Binary search for the answer and run Dijkstra's Algorithm from vertex $n$ to vertex $1$ can also pass, but you should optimize your constant since this solution has a time complexity of $\Theta(m\log(n)\log(\frac{n\cdot maxw}{\varepsilon}))$, larger than the standard solution.

# Thanks!