The 2015 ”Orz Panda” Cup Programming Contest Editorial

Orz Pandas

August 26, 2015

1 Array of Orz Pandas

This problem is from Master Jie and seems too difficult for Orz Pandas. If you can solve it, all Orz Pandas would come to Orz you. Here is Master Jie's approch:
We can change the subscript of sum, and get

$$f(l,r) = \sum_{k_1 \le A[i] \le k_2} [l \le i \le r]$$

Certainly, it's equal to

$$\sum_{A[i]} [l \le i \le r] - \sum_{A[i] < k_1} [l \le i \le r] - \sum_{A[i] > k_2} [l \le i \le r]$$

The first term is simply $(r - l + 1)$. After sorting the querys by k_1, we can evaluate the second term $\sum_{A[i]<k_1} [l \le i \le r]$ of all m querys in $O(mlogn)$ time, by inserting elements of array A into a segment tree or binary index tree, in ascending order. Similarly, we can also get the third term by inserting elements in descending order. The total time complexity is $O(nlogn + mlogm + mlogn)$.

2 Bidirectional Orz Pandas

Most Orz Pandas agree that Problem B is easy. Certainly, $S_N[k]$ is defined by following recursion

$$S_N[k] = \begin{cases} 1 \oplus S_{N-1}[k] & k \le |S_{N-1}| \\ 1 & k = |S_{N-1}| + 1 \\ S_{N-1}[|S_N| - k + 1] & k > |S_{N-1}| + 1 \end{cases}$$

However, a recursive algorithm using this recursion has an $O(N)$ time complexity. This is too slow. But we can notice that $|S_{60}| = 1152921504606846975 > 10^{18}$, and $k \le 10^{18}$, so we have equation

$$S_N[k] = 1 \oplus S_{N-1}[k] \ (n \ge 60, k \le 10^{18})$$

Because $1 \oplus 1 = 0$, we get

$$S_N[k] = \begin{cases} 1 \oplus S_{59}[k] & (n = 60, 62, ..., k \le 10^{18}) \\ S_{59}[k] & (n = 61, 63, ..., k \le 10^{18}) \end{cases}$$

N is reduced to 59, then we can simply use an $O(N)$ approach.

3 Combining Orz Pandas

With the definition of **period**, the problem is to select K numbers in $T_1, T_2, ..., T_N$ of which the LCM (Lowest Common Multiple) is minimal.
Orz Pandas can only solve this problem when $1 \le K \le N \le 50$. But God Wan thinks this problem is too simple, and the Orz Pandas are too young, sometimes naive. Here is God Wan's solution:
Let $f[i]$ be the maximum M fulfill there are M numbers in $T_1, T_2, ..., T_N$, and of which each is a factor of i. Then the answer is the minimal i fulfill $f[i] \ge K$.

We can get $f[i]$ by following approach: initialize f with zero, then for each T_j, increase $f[T_j], f[2 \times T_j], f[3 \times T_j], ...$ by one. The time complexity is

$$O(\sum_j \lfloor \frac{maxans}{T_j} \rfloor)$$

It's $O(maxans \times logn)$ in average, and $O(maxans \times N)$ in worst case (when $T_j \equiv 1$). So this problem can be solved even if N is 1000.

4 Dominator Orz Pandas

This problem of Master Nuo is so difficult that no Orz Pandas can solve it. Let $cnt[u]$ be the number of vertexs in the subtree rooted u, and $f[u]$ be the number of different ways to distribute $cnt[u]$ Orz Pandas in the subtree rooted u. Then we may guess

$$f[u] = \prod_{v \in child(u)} f[v]$$

However this is wrong. Since two subtree rooted different child of u are independant with each other, we can swap the id (ability) of two Orz Pandas in different subtrees. This will produce a new distribution. The total number of different distribution produced by swapping is exactly

$$\frac{(cnt[u]-1)!}{\prod_{v \in child(u)} cnt[v]!}$$

Then if u is not important, we can swap the id of Orz Panda at u with any other Orz Pandas in the subtree rooted u. So

$$f[u] = \frac{cnt[u]!}{\prod_{v \in child(u)} cnt[v]!} \prod_{v \in child(u)} f[v] \ (u \ is \ not \ important)$$

Otherwise

$$f[u] = \frac{(cnt[u]-1)!}{\prod_{v \in child(u)} cnt[v]!} \prod_{v \in child(u)} f[v] \ (u \ is \ important)$$

We can use dfs to determine the order of evaluating f. The answer is $f[1]$. The time complexity is $O(n)$.

5 Equation of Orz Pandas

Notice that

$$\begin{aligned} x^{2n} + y^{2n} &= (x^n + y^n)^2 - 2(xy)^n \\ x^{2n-1} + y^{2n-1} &= (x^n + y^n)(x^{n-1} + y^{n-1}) - (x+y)(xy)^{n-1} \\ x^{2n+1} + y^{2n+1} &= (x^n + y^n)(x^{n+1} + y^{n+1}) - (x+y)(xy)^n \end{aligned}$$

Let $ans[n]$ be $(x^n + y^n)$. Then

$$\begin{aligned} ans[2n] &= ans[n]^2 - 2B^n \\ ans[2n-1] &= ans[n]ans[n-1] - AB^{n-1} \\ ans[2n+1] &= ans[n]ans[n+1] - AB^n \end{aligned}$$

So we can always represent $(ans[n], ans[n+1])$ using $(ans[\lfloor \frac{n}{2} \rfloor - 1], ans[\lfloor \frac{n}{2} \rfloor])$ or $(ans[\lfloor \frac{n}{2} \rfloor], ans[\lfloor \frac{n}{2} \rfloor + 1])$, and A, B. At each time the each number of the pair $(n, n+1)$ would be reduced about a half, so the time complexity of this recursive approach is $O(logn)$.
Some Orz Pandas can also solve this problem with matrix.

6 Fighting Orz Pandas

Just sort all Orz Pandas. Then we can use binary search to find the Orz Panda to beat for each srO Panda. The time complexity is $O((N+M)logN)$.

7 Gold of Orz Pandas

Let S be the set of all quests. Define a subset A of S *independent* if we can finish all quests in A. Certainly, if A is *independent*, we can always finish all quests in A in ascending order of dead line.
Define $N_t(A)$ is the number of tasks in A whose deadline is t or earlier, we have

Lemma 7.1. *For any $A \subseteq S$, the following statements are equivalent.*
1. A is independant.
2. For $t = 0, 1, 2, ..., n$, $N_t(A) \le t$.
3. All quests in A can be finished in order of monotonically increasing dead lines.

Proof. If $\exists t : N_t(A) > t$, there is no way to finish quests in A since there are more than t quests must finish before t. So (1) implies (2). If (2) holds, using method of induction on subsets of A including quests of which the dead line is not exceed $1, 2, 3..., n$ we can get (3). And certainly (3) implies (1). □

Let I be the set of all *independent* subsets of S. Then we have

Lemma 7.2. *The ordered pair (S, I) is a matroid.*

Proof. Certainly, if $A \in I$ and $B \subseteq A$, $B \in I$, so I is *hereditary.* Now we need to proof (S, I) satisfies the *exchange property.* If $A, B \in I$ and $|B| > |A|$, find a largest k such that $N_t(B) \le N_t(A)$. Since $N_0(B) = N_0(A)$, k always exists. Since $|B| = N_n(B) \ne N_n(A) = |A|$, we must have $k < n$ and $\forall j \in (k, n] : N_j(B) > N_j(A)$. So B must contains more tasks with dead line $k + 1$ than A does. Let a_i be a task in $B - A$ with dead line $k + 1$. Let $A' = A \cup a_i$. For $0 \le t \le k$, we have $N_t(A') = N_t(A) \le t$ since $A \in I$. For $k < t \le n$, we have $N_t(A') \le N_t(B) \le t$, since $B \in I$. So $A' \in I$.
Now using the fact that I is *hereditary* and (S, I) satisfies the *exchange property*, (S, I) is a matroid. □

So we can use a greedy algorithm to solve this problem. Sort all quests in order of monotonically decreasing EXP's, then for each quest add it into the set of quests planned to be finish, if possible. We can check the possiblity of finishing a quest using N_t. If we use brute force to maintain N_t we'll get an $O(N^2)$ approach, and using segment tree can produce an $O(NlogN)$ approach.
For more information about matroid, read Orz Pandas' favourite book: *Introduction to Algorithms.*

8 Hexagon of Orz Pandas

We can try all $6! = 720$ orders of edges since we have only six edges. Because the problem is manually judged, we need not to worry about floating point precision of the answer itself. But when we're checking whether an order of edges would make up a *diaoable* hexagon, we must be careful. In fact, we have an important conclusion

Lemma 8.1. *Let $a[i]$ be the length of i-th edge (in counter-clockwise order) of hexagon H. H is* diaoable *if and only if*

$$a[0] - a[3] = a[4] - a[1] = a[2] - a[5]$$

Proof. *H* is *diaoable*, if and only if all angle of H are $\frac{2\pi}{3}$. On the complex plane, we have

$$\sum_{k=0}^{5} a[k]e^{\frac{2k\pi i}{6}} = 0$$

This equation means if we start from vertex v of H, and move along all six edges of a hexagon, we'll return to v. Since $e^{i\pi} = -1$, we have

$$(a[0] - a[3]) + (a[4] - a[1])e^{\frac{4\pi i}{3}} + (a[2] - a[5])e^{\frac{2\pi i}{3}} = 0$$

Since a contains only real number, and $e^{\frac{4\pi i}{3}}$ is the conjugate of $e^{\frac{2\pi i}{3}}$, we must have $a[4]-a[1] = a[2]-a[5] = T$ to make the imaginary part of the sum 0. The real part of the sum is

$$a[0] - a[3] + Tcos(\frac{4\pi}{3}) + Tcos(\frac{2\pi}{3}) = a[0] - a[3] - T = 0$$

So $a[0] - a[3] = T$. □

With this conclusion, we can determine whether an order of edges can make up a *diaoable* hexagon safely (i. e. without any floating point operations).

9 Inserting Orz Pandas

Another hard problem from Master Nuo. Notice that the number of lines is small, we can use bitmask to represent the state of a column. For example, bitmask $10_{(10)} = 1010_{(2)}$ means in this column there is one Orz Panda on line 2, and one Orz Panda on line 4.
Then the constraint of this problem can be represented:
(1) For i-th column, the number of 1's (called *popcount*) in the bitmask must be a[i].
(2) The *xor* of all columns' bitmask must be

$$\sum_{i=0}^{n-1} 2^i b[i+1]$$

Certainly we can use DP to solve this problem. Let $dp[n][k]$ be the number of configurations in the first n lines such that the *xor* of all columns' bitmask is k, we have recursion

$$\begin{array}{ll} dp[0][0] = 1 & \\ dp[0][j] = 0 & (j \neq 0) \\ dp[i][j] = \sum_{popcount(k)=a[i]} dp[n][j \oplus k] & (i > 0) \end{array}$$

There are at most $C_n^{\lfloor n/2 \rfloor}$ bitmasks such that the *popcount* is exactly $a[i]$. So the time complexity is $O(m \times 2^n \times C_n^{\lfloor n/2 \rfloor})$.

10 Judgment of Orz Pandas

Let A, B, C be the position of Orz Pandas and P be the position of the person. P is inside ΔABC if and only if

$$\begin{cases} (\overrightarrow{AO} \times \overrightarrow{AB})(\overrightarrow{AO} \times \overrightarrow{AC}) & \leq & 0 \\ (\overrightarrow{BO} \times \overrightarrow{BA})(\overrightarrow{BC} \times \overrightarrow{AC}) & \leq & 0 \\ (\overrightarrow{CO} \times \overrightarrow{CA})(\overrightarrow{CB} \times \overrightarrow{AC}) & \leq & 0 \end{cases}$$

11 K = 1 Problems of Orz Pandas

This is an easy problem from Light. Scan *Num* from the least significant bit and record the maximum number. For each bit, if it's smaller than the maximal bit among all bits lower than it, we can swap them. The best swap is the swap at highest bit among all swaps. The time complexity is $O(log\ Num)$.