Hyperion Introduction to Competitive Programming

Stephen Huan

April 5, 2020

1 General Tips

For Python, use "PyPy 3.6 (7.2.0)". It should be basically a drop in substitute for Python 3, and PyPy is ***much*** faster than Python. Instead of using the input function, replace it with

```
import sys
input = sys.stdin.readline
```

sys.stdin.readline is something like 2-5x faster than input. For C++, use

```
std::ios::sync_with_stdio(false);
std::cin.tie(NULL);
```

2 A. Big-O

A single if ... elif ... else conditional statement suffices.
Be careful whether something is $<$ or $\leq$!

3 B. Penalty

We can use a for loop over each problem, adding its contribution to the total penalty as we go. Each problem i adds $t_i + Pn_i$ to the total penalty.

4 C. SumNum

The idea is going to be to add as large as possible digits to the front of the number, and then all zero's after. Adding as large as possible digits initially guarentees that one, the resulting constructed number is actually the specified length (if we added zeros to the start, it would be smaller) and that we don't run out of space to reach the sum (for a case like $L = 3$ and $T = 27$). However, we can't add all 9's, since it might add up to greater than the sum. Therefore, the next digit d we add is going to be $\min(9, T)$. Then, update T by subtracting d.

5 D. Amazon

The general set of problems is called "integer knapsack". In this specific case, each item has a "value" of 1, since we only care about how many items we have. Therefore, we can get the most items by always buying the cheapest items first. Sort the items by price, and purchase as much as you can afford. If each item costs c, and you have D dollars, you can afford $N = \lfloor D/c \rfloor$ items. "Buy" these items by adding them to the total, and decrease the amount of money you have by calculating $D - Nc$.

6 E. Copyycat

Maintain a set of elements we've already seen. If an element is already in this set when we go to add it, then it must be a duplicate. How do sets work? Hashing.

7 F. Close Game

The first observation is that Daniel must have the largest number in the array. If Justin had it, then it would be impossible for all of Daniel's numbers to be greater than or equal to Justin's. The second is that reverse sorting will help us. Suppose we sort the array in decreasing order, with the largest numbers at the beginning. We know that the absolute largest number goes to Daniel. Then, consider the second largest number. It can go to either Daniel or Justin. If it goes to Daniel, then nothing of note happens. However, if it goes to Justin, then all the numbers in the array *after* it must go to Justin. Since all of Daniel's numbers must be greater than or equal to Justin's numbers, if Justin gets the second largest number than Daniel can't get the third largest or the fourth or any number after. Thus, we can process each number, seeing what the result of a game where we assign it to Justin would be, but continuting by actually assigning it to Daniel.

In order to "simulate" giving a number and all the numbers after it to Justin efficiently, we can first precompute a prefix sum. The prefix sum maintains the running sum of the array at each particular index. Specifically, prefix[i + 1] is equal to prefix[i] + a[i], where a is the array. prefix[0] = 0 by definition.

Our final algorithm is to iterate over each number. Compute the absolute difference between Daniel's current sum and the prefix sum at this position, and see whether that difference is less than the current best. Then, add the number to Daniel's sum.

8 G. Snowflakes

Generate a new array of pairs, where the first number in each pair is the difference between sucessive elements in the snowflake array, and the second number is the index.

Then, sort this array of pairs by the first number and, iterating in order, report that count and the index which it occurs.

9 H. Aspiring Artist

Recall the APCS lab on floodfill.

10 I. Snowman

The first observation is that even though the sets {4, 1, 2} and {4, 2, 1} are different, we don't actually care because their sum is the same.

The number of total sets, ignoring permutations, is going to be $\binom{N}{K}$. If we compute the sum of the heights of each array, then we can just multiply that sum by $K!$ to get our final answer. The problem is that $\binom{N}{K}$ is gigantic, so we can't actually generate each set and compute its sum.

Instead, we realize that each element is represented in the sets the same number of times - as in, the first element is in as many sets as the second element, because the sets are not generated based off position. If there are $\binom{N}{K}K!$ cases, where each case accounts for permutations, then that simplifies to $\frac{N!}{K!(N-K)!}K! = \frac{N!}{(N-K)!}$ cases, which I'll call C. Each case has K numbers in it, so there are KC total numbers, and since each number is represented evenly, each number occurs $\frac{KC}{N}$ number of times.

So to compute the sum over all sets, we can instead compute the sum of the array and multiply it by $\frac{KC}{N}$. However, we have to account for the modulo. Instead of calculating the factorial outright, modulo at each step. What about the division? Your first reaction might be modulo multiplicative inverses, but since M is not guarenteed to be prime, the mutiplicative inverse does not necessarily exist. Instead, note that $C = \frac{N!}{(N-K)!}$. Thus, $\frac{C}{N} = \frac{(N-1)!}{(N-K)!}$. In order to compute $\frac{(N-1)!}{(N-K)!}$, cancel the denominator. $\frac{(N-1)!}{(N-K)!} = (N-1)(N-2)...(N-K+1)$.

11 J. Favorite

Binary search.

12 K. Snow Angels

Connected Components.

13 L. Dryer Capacity

Dynamic Programming.

14 M. Hot Chocolate

Recall that $\int_b^a f(x) = F(b) - F(a)$ where $\frac{\mathrm{d}F}{\mathrm{d}x} = f$. Since a prefix sum is basically a discrete integral, think of the prefix sum as the antiderivative of the array. Thus, the sum between two indexes i and j in an array is prefix[j + 1] - prefix[i].

This makes a lot of intuitive sense - the subtraction cancels out the lower part, leaving only the segment you care about.