Competitive Programming Guild 2024-25 *Application*

Karthik Kashyap

14 September 2024

Contact Details:

1. **Name:** Karthik Kashyap K
2. **Roll Number:** EE23B030™
3. **Contact Number:** 8073978167

I am aware that this was supposed to be a Google form response to the application, but a Google form is not very much appealing when it comes to highlighting points, so I have made the application into a LaTeX. In case, the Google form was taken as a means to the application was for convenience sake, the same answers can be found there as well. Also, any related files pertaining to this app can be found in this GitHub Repository.

Part 1: About Me

This section contains the HR part of the application, being the first page of the Google form application.

§1 Experience with CP

I have been doing CP on Codeforces for about 10 months now, as I started it back in the early November of last year. I have solved about 380 problems on Codeforces and given several contests on it when I do not have any assignments or quizzes.

I have accounts in Codeforces and Codechef, but I have only used Codechef for 2 contests of so, only because my friend recommended it to me. In order to write a contest for the Part-2 of this application, I made an account in AtCoder, and wrote one contest in it.

- Codeforces ID: PixlatedBatman, I have a max rating of 1483 on Codeforces, being a Specialist.
- CodeChef ID: pixlatdbatman, I have a max rating of 1491 on Codechef, being 2 star.
- AtCoder ID: PixlatedBatman, I have a *Provisional* rating of 56, due to that one contest.

§2 Motivation to do CP

CP is mostly interesting for 2 things, one being the logical part of it, that is figuring out how the problem is to be solved, and the second being the code part of it, which is to put the said logic as a program. That being said, both of these go hand in hand, by which I mean, when finding a logic for it, we don't just find any logic, we try to find a logic which is most efficient, in both memory and time wise, and then code it with the necessary algorithms and structures needed for it. This is what is the most interesting part of it, the fun of doing both things right. Which is why, sometimes when I'm bored, I open up CF instead, to upsolve a contest I missed, or open up the problemset and solve a few interesting problems.

Again, like all things, if it's done too much, it becomes a little repetitive, and hence one might say they get bored of it or get burned out. This did happen to me once, when I tried upsolving like 1 or 2 previous contests in a day- I got a little bored after two weeks. This is why one mustn't overdo things, and do things in the amount that can get them to be invested, in my case about 4 problems in a day(first few problems in a contest) and ever since, I haven't gotten much bored of CF.

§3 Expectations of CP Guild

1. I have learnt quite a bit and have been invested in CF already, but there are a few topics, like Trees, complex graphs, complex algorithms, that I don't quite know much about, and would like to learn about, but every time I do try to learn it, I either learn very less from it, or not learn it because of the complexity. Hence I would like to join the Guild to learn these things, as these are mostly what holds me back from solving D,E problems in Div. 2 contests most of the times.

2. As part of contributing to the team, I would say I am quite good at DSA, and I can help contribute in that aspect, as DSA is sometimes confusing to people who don't quite know it fully. Other topics like Game Theory, Greedy, Recursions, etc. are topics that I would say I am good at.

§4 Commitments and Plans

PORs: I am a Coordinator in both the Mathematics Club, and the Programming Club. I have no other volunteer positions or projects, and my Sem is not too acads heavy. Hence, I will have quite a lot of free time to spend on CP, like about an hour or two for about 3 days on the weekdays, and much more on the weekends if necessary.

I also have no travel plans or anything as such in my vacations, so for the requirement in December, I will be quite free, so I will be ready to invest a lot of time for CP if needed, without any problems.

Part 2: Time to CP!

This section is about the CP part of the application, which is the second page of the Google form and is pertaining to self-analysis and learning new topics. For better presentability, this section is only on this LaTeX.

§1 Task 1: Self-Analysis of Contests

For this section, I have written two contests since the beginning of the application, the *AtCoder Beginner Contest 371* and the *Codeforces Round 972 (Div. 2)*. The performance and self-analysis for both of these have been added below:

§1.1 AtCoder Beginner Contest 371

Here are the standings for this contest and my ranking was about 4419. My ID is PixlatedBatman.

To talk about my self-analysis on this contest, first I will talk about a few mistakes that I made, which cost me a bit, and some improvements I can make so that they do not repeat:

1. I started the contest 15 min late. It usually does not happen for 8PM contests, but just before this contest, I decided to go for a tea break, but my order at HFC came late and I met one of my friends along the way, so I got a little delayed. Moreover, this was only a 100 min contest rather than the usual 2 hr contest.

2. I used about 5 minutes on the A problem, which is way more than it should normally take. This is because it was a simple case of Hard-coding a few parts of the problem, but I was trying to find efficiency, which was unnecessary.

3. B was done relatively quickly, so there is not much I can improve on that.
4. C problem was based on graphs. It might not be that hard, and some parts of the graphs were defined in the problem, but even then I couldn't understand the problem statement, so I had to leave it. As mentioned before, I need to learn to solve graph-related problems.
5. I could get the method to do the D problem quite easily. The only issue I faced was that I thought when binary searching through a sorted array, using the lower_bound would give me the index of element lower than the value I'm searching for, when the element is not found in the array, but after a bit of using GeeksforGeeks, I realized that both upper_bound and lower_bound give the next element. Wasted about 5 to 10 min of debugging time because of that.
6. For the E problem, because it involves double summation, I tried to solve it mathematically, and then found an equation that depends on n, but then when the time was about to run out, I found an error with it and could not fix it. Also, I was not able to upsolve it, because of the next contest was just after this, and I had to go to dinner.
7. I hadn't checked the rest of the problem statements, which I need to probably do, just in case I can solve those better.
8. Overall, this being my first contest, gave me a +56 rating change on AtCoder, which I am not sure if it is good or not, as I am not familiar with AtCoder rating system.

§1.2 Codeforces Round 972 (Div. 2)

Here are the standings for this contest and my ranking was about 2830. My ID is PixlatedBatman.

To talk about my self-analysis on this contest, first I will talk about a few mistakes that I made, which cost me a bit, and some improvements I can make so that they do not repeat:

1. To not repeat the same mistake as I did for the above contest, I went to dinner early and reached on time for the contest.
2. Coming to the A problem, I wanted to solve it as quick as possible, to correct my mistake from the previous contest. But after I submitted my code with my logic, I got *Wrong answer on Test 1*, which is too depressing and embarrassing, so I had to take more time to correct it and submit the right one. Luckily, as it was wrong on Test 1, I was not given any penalty for it.
3. For the problem B1, I should have realised that working on B2, and submitting the same code for B1 should have been the optimal strategy, as only the constraints on the variables differ. Due to not thinking of this, I had written it only considering the B1 problem.
4. Like I said about B1 problem, even though the algorithm was about the same, except for applying binary search and such, I had to re-write the initial syntaxes for it, as it was different from B1, which could have been optimised. The rest was smooth, and could solve the question quick.
5. Problem C: My nemesis. As soon as I read the problem, I realised that it involves some sort of DP, but wasn't getting much of a clear idea as to whether to optimise for the check of the sequence of *Narek* or whether to reduce the score of ChatGPT as much as possible (This is related to problem, I am not talking about GPT's usage). Hence, I was stuck at this for a long time, and even in the end I was not able to come up with a solution for this.
6. I did look at the further problems unlike the previous contest, but realised that C is the most do-able looking problem out of them, so I left those.
7. Another slight issue I had during this contest, and most of the other contests, is that I have the standings tab open all the time, and I periodically keep checking my standings in the contest to check on my progress. But for this case, what I also did was that I noticed that none of my other friends who were in

Expert rating were also unable to solve C, or any other further problems, and this put my confidence on getting this question down. Hence, I probably should stop checking for my standings, and getting demotivated from that. (P.S: One of my friends managed to solve C at the $1:50$ mark, but at that point it was too late for me to get the confidence to solve it.)

§2 Task 2: Learning new topics

For this section, we were given the a number of topics to learn, and to explain it out, along with the code, use cases, blackbox, and an example problem where it is used. DP is always an interesting topic to explore, and hence I picked the topic of Digit DP. I have explained these on my own words after learning them, so if there is any sort of small ambiguity in what I have said, I apologize in advance.

§2.1 Digit DP

Digit DP is used to efficiently find a property relating to the digits of all numbers between two given numbers a and b. Brute forcing this for all the numbers between the numbers, by finding all the digits and finding the property, is very time-complex, the time complexity being $O((a+b)\times 18)$ (there can be a maximum of 18 digits). This does not seem bad, but we have to consider that the inputs can be huge. Considering that inputs are in the range of long long int, which can have 18 digits, this can have about 3×10^{20} iterations in the worst case!!! This is very bad and can be hugely improved by using Dynamic Programming.

§2.1.1 Data Structures and Functions needed

As this is a simple case of a 2D Dynamic Programming, we need just a 2D array or a 2D vector.

As for functions, we can define a few convenience functions like:

- A function to return a vector of all the digits of a given number.
- A function to recursively implement the DP.

§2.1.2 Situations where Digit DP is used

Digit DP can be used in situations where the problem statement is related to the digits of the numbers in a given range. Some examples can be like:

1. Sum of all digits of all the numbers between a and b.
2. The maximum product of digits in the range of $[a,b]$.
3. Number of palindromic numbers between a and b.

As explaining in general can be a bit ambiguous, I will be taking one of the cases where Digit DP can be used and explaining the process. The case I am choosing is the first one mentioned, that is, to find the sum of digits of all numbers in the range $[a,b]$.

§2.1.3 Blackbox and Bonus

To do this, the input, as mentioned above, is two integers a and b, the range where the given operation is to be performed. The output is the required property that is requested.

To perform this, instead of performing operations from a to b, we instead perform operations on all digits from 1 to b and then 1 to $a-1$, and then subtract these numbers to obtain the result. As we can have a maximum of 18 digits, the number of iterations is significantly decreased.

Let us consider the case where we need to find the sum of digits of all numbers from 1 to n. This, as mentioned, can be performed with a 2D DP, with the current sum and the index of the digit under consideration are the dimensions used. With this we can have a recursive function, or even a loop to iterate from the MSD to the LSD of the number. Apart from the current sum and the index, we also need a third variable, let's call this tight, to check for any restriction on the digits we have. Normally, we consider all iterations of digits from 0 to 9. Sometimes, going past a certain digit would result in the number under consideration to be outside the range of $[1, n]$. Hence, when we reach this certain digit, the tight becomes 1, to signify that we no longer need to iterate past that in the current index, and to move on to the next index, and otherwise, tight is to be 0.

One such problem I found, we can use this, and then while having to calculate from a to b instead, we can just take the difference of 1 to b and 1 to $a - 1$. The implementation can be seen below:

Function to return the digits of the given number in a vector:

```
vector<llt> Digits(llt num){                // llt being used for long long int
    vector<llt> digits;

    while(num){
        digits.push_back(num%10);
        num /= 10;
    }

    return digits;
}
```

Function to recurse through all the digits of the given number:

```
llt dp[163][20];

llt Sum_to_N(llt sum, llt ind, llt tight, vector<llt> &digits){
    // Base case
    if(ind == -1) return sum;

    // If the value already exists in the dp
    if(dp[sum][ind] != -1 && tight == 0) return dp[sum][ind];

    // Max integer to be iterated to, based on the value of tight
    llt val = 0, new_tight, k;
    if(tight) k = digits[ind];
    else k = 9;

    // Iterating for all possible digits, and recursing them
    for(int i=0; i<=k; i++){
        if(i == digits[ind]) new_tight = tight;
        else new_tight = 0;

        val += Sum_to_N(sum+i, ind-1, new_tight, digits);
    }

    // If tight is 1, then the val is not to be stored
    if(!tight) dp[sum][ind] = val;

    return val;
}
```

This way, we can call the function *Sum_to_N* for b and $a - 1$, and return the difference whenever necessary.

Time complexity

As we have seen in the implementation above, this code recurses for the variables sum, ind, tight, and then iterates for all digits 0 to 9. This gives it a complexity of $O(10 \times sum \times ind \times tight)$. As the input is in the range of long long int, it can have a max of 18 digits, and hence, the sum can be a max of 18×9, ind has max of 18, and tight can have only two values, namely 0 and 1. Hence, the number of max iterations can be calculated to be 6×10^4, which is way more reduced than the previous brute force case of 3×10^{20} iterations.

The full code can be found in this GitHub Repository.