Daniel F. Perez-Ramirez

Altran Kattis Programming Test Fedback

In general I had a lot of fun solving the tasks. I considered the difficulty of these appropriate: problem A (easy) dealt with basics of programming. Problem B (medium) needed some Software engineering and algorithms background for solving. Problem C (hard) wasn't (in my opinion) that much about software engineering complexity, but to find clever simplifications on the math and choosing a programming language for getting performance targets.

Below are some remarks on each problem in the order in which I decided to solve them.

Problem A: election day (easy)

Solved in python due to its simplicity. No general remarks. I took the chance to document and structure the code as I usually do at work mostly respecting PEP8 style.

Problem C: factorial (hard)

- I started solving this task in python. I first noticed that the "math.factorial()" function get stuck with numbers bigger than 100'000, so I had to transition to an alternative calculation of the factorial.
- On the way, I identified that multiples of 10 and 5 just kept adding zeros a the end of the factorial result, so I filtered these out. I could then calculate the factorial of 1'000'000 is less than 2 seconds (target <= 1sec), but the program still got stuck when evaluating 10'000'000! (max-value input).
- As only the las 3 digits before the trailing zeros where of interest for the task result, I decided to trim all intermediate results to have only an "empirically-estimated amount of digits", so e.g. if the intermediate result was 54321 and the "empirically-estimated amount of digits" was 4, the trimmed result would be 4321.
- With this, I could calculate the result for max-value input in 10-13 seconds. With those results, I knew it was time to change to C++ to decrease execution time.
- I re-wrote the code to C++, increasing the "empirically-estimated amount of digits" and using 'unsigned long long' for caching the intermediate result and I was able to solve the task.

Problem B: honeycomb (medium)

With 24h to deadline, I decided to start with C++ right away. From the problem statement one could assert it was a search optimization problem.
I decided to break the task in 2 parts: discretization of the environment as a graph and implementation of a graph search algorithm.

- For the discretization of the environment I wrote a class which generated a honeycomb grid dependent on the edge length. The class contained an unordered-map with the cells as keys and a vector of the cell-neighbors as values (as the maximum edge length was 20, this did not hurt the memory requirements).
- For the graph search, I chose at the beginning Breadth-First-Search as it is guaranteed to converge to a solution if existent. As it was also to be evaluated if the N possible steps taken by the ant in the honeycomb could fulfill the task, I implemented a short modification to the BFS allowing it to keep track of the shortest distance to the explored nodes (sort of a hybrid of Dijkstra), which solved the task.