Code Jam 2018 - Round 1A

Analysis: Bit Party

Test set 1

We might consider enumerating all the ways of assigning bits to cashiers, but with 20 bits and 5 cashiers, there could be as many as 5^{20} ways — too many to check! We need to take advantage of the fact that the bits are interchangeable, and instead find all the ways to *partition* **B** bits among **R** of the **C** cashiers (since we will only be able to use as many cashiers as we have robots). Then, we can compute how much time each of those ways takes, and pick the minimum of those values.

We can calculate the number of ways to partition 20 bits among 5 robots using this method. It turns out there are only (24 choose 4) = 10,626 ways to check. If we have fewer robots than cashiers, then we need to introduce another multiplicative factor of (**C** choose **R**), but this cannot be larger than 10 in test set 1. Each check takes O(**R**) time, which is very small given that **R** is at most 5. So, test set 1 should be solvable well within the 15 second time limit, regardless of your language choice.

Test set 2

To solve this test set, we need to be able to answer the following question: Given a time limit T, is there a possible assignment of bits such that all the robots can finish interacting with their cashiers in no more than T seconds? Let f(T) be the answer to this question.

How can we find f(T)? The maximum number of bits that the i-th cashier can process in not more than T seconds is max(0, min($\mathbf{M}_i$, floor((T - $\mathbf{P}_i$) / $\mathbf{S}_i$))). Let us call this value Capacity_i.

Then, we want to know whether a total of **B** bits can be assigned to **R** robots, and each of those robot to a cashier, such that the number of bits processed by the i-th cashier is not more than Capacity_i. To do this, we can greedily sort the Capacity values into nonincreasing order, and then assign the **R** robots to the first **R** cashiers. f(T) is true if and only if the total number of bits that can be processed by the first **R** cashiers is at least **B**. Therefore, we can compute the value of f(T) for any T in O(**C** log(**C**)) time (which is the time it takes to sort the Capacity values). (Aside: We can even avoid the sort, and instead partition in O(**C**) time, by using introselect, for example.)

Since we want to minimize the time taken for all robots to interact with their cashiers, we want to find the minimum possible value of T such that f(T) is true. That value of T will also satisfy the following:

- f(x) is false for all x < T.
- f(x) is true for all x ≥ T.

Therefore, we can find the value of T using binary search. Since the maximum answer will not be more than O(max(**S**) × **B** + max(**P**)), this solution will run in O(**C** log(**C**) log(max(**S**) × **B** + max(**P**))) time.