Kick Start 2021 - Round G

Analysis: Dogs and Cats

We need to determine if all the dogs will be able to eat food given the condition that an animal can only eat if the animal before it has already eaten. We do not worry if some cats are hungry as long as all the dogs are able to eat. Let us assume that the last (or rightmost) dog is at position p. We do not need to worry about the cats in positions $[p+1, N]$. Since an animal can only eat food if the animal before it has already eaten, we need to make sure that all cats and dogs in positions $[1, p]$ are able to eat food.

Test Set 1

We are given $\mathbf{M} = 0$. This means whenever a dog eats food, no cat foods magically appear. If $\mathbf{S}$ contains all 0 i.e., no dog is present, the answer is `YES`. Otherwise, we find the position of the last (or rightmost) dog, p. Let the total number of dogs i.e., the number of 1 in $\mathbf{S}$ be X and the number of cats before the rightmost dog i.e., the number of 0 in $[1, p-1]$ be Y. The answer is `YES` if $\mathbf{D} \geq X$ and $\mathbf{C} \geq Y$ and `NO` otherwise.

Complexity : $O(\mathbf{N})$ *per test case*

Test Set 2

We are given $\mathbf{M} \geq 0$. This means whenever a dog eats food, $\mathbf{M}$ cat food portions are added. We iterate over positions $[1, N]$ and whenever we see a cat we feed it with the cat food i.e., subtract 1 from $\mathbf{C}$ and whenever a dog appears we feed it with the dog food, i.e., subtract 1 from $\mathbf{D}$. We need to ensure that the dog can only eat if the animal before it has already eaten, and to do so, we first make sure that $\mathbf{D} \geq 1$, as 1 food item is needed to feed this dog and $\mathbf{C} \geq 0$. If this is not true, our answer is `NO`. Otherwise we subtract 1 from $\mathbf{D}$ and add $\mathbf{M}$ to $\mathbf{C}$. If the above condition is satisfied for all the dogs, our answer is `YES`. Note that this algorithm will also work for Test Set 1 but not vice versa.

Sample Code(C++)

```
bool have_dogs_eaten(string S, int N, int D, long long C, int M) {
  for(int i = 0; i < N; i++) {
    if (S[i] == '1') {
      if(D <= 0 || C < 0) {
        return false;
      }
      D--;
      C += M;
    } else {
      C--;
    }
  }
  return true;
}
```

Note that variable `C` is long long since maximum possible value does not fit in 32-bit data type.

Complexity : $O(\mathbf{N})$ *per test case*