621. Task Scheduler

Description HintsSubmissionsSolutions

Discuss Editorial Solution Pick One

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks.Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle.

However, there is a non-negative cooling interval **n** that means between two **same tasks**, there must be at least n intervals that CPU are doing different tasks or just be idle.

You need to return the **least** number of intervals the CPU will take to finish all the given tasks.

Example 1:

Input: tasks = ['A','A','A','B','B','B'], n = 2

Output: 8

Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.

Note:

1. The number of tasks is in the range [1, 10000].
2. The integer n is in the range [0, 100].

Seen this question in a real interview before?

The optical solution to this problem is to do the round robin schedule, Consider the case, where say the number of instances of tasks A, B, C, D, E are 6, 1, 1, 1, 1 respectively with n=2(cooling time). If we go by the above method, firstly we give 1 round to each A, B, C, D and E. Now, only 5 instances of A are pending, but each instance will take 3 time units to complete because of cooling time. But a better way to schedule the tasks will be this: A, B, C, A, D, E, ... In this way, by giving turn to the task A as soon as its cooling time is over, we can save a good number of clock cycles.

```
 class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        vector<int> mp(26,0);
        for(char c:tasks)
        {
            mp[c-'A']++;
        }
        sort(mp.begin(),mp.end());
        int times = 0;
        while(mp[25]>0)
        {
            int i=0;
            while(i<=n)
            {
                if(mp[25]==0) break;
                if(i<26 && mp[25-i]>0)
                {
                    mp[25-i]--;
                }
                i++;
                times++;
            }
            sort(mp.begin(),mp.end());
        }
        return times;
    }
};
```