Question: https://leetcode.com/problems/task-scheduler/

solution: pretty much based on intution on how we would schedule the tasks normally on paper.

always schedule the tasks that have more count first; later for wait time schedule other low count tasks

if no low count tasks avaialble; use idle time.

data structure needed:

use a dict to initially story the task and count of each task

later add the count values along with task to max-heap (Priority queue). this gives better retrieving of tasks.

use `add\_back` as a storage to add back the counts to `pq` once we use extractMax().

Implementation:

outer loop until the Priority queue all counts are zero

use an innner for-loop to grab 'n' tasks from the max-heap and decrement the counts of max heap.

implementation of the max heap queue may vary here:

as we don't have direct way to decrement the counts in max-heap; we poll() the element the from maxheap

and store the counts to list(if counts > 0); and later add back the list to `pq` after the end of while loop.

time complexity: O(result) --> As we only compute till the time counter is incremented.

space: O(tasks\_count)

Code:

class Solution {

public int leastInterval(char[] tasks, int n) {

Map<Character, Integer> task\_counter = new HashMap<>();

for(Character task: tasks) {

task\_counter.put(task, task\_counter.getOrDefault(task,0) + 1);

}

// add the frequency of all tasks to pq (Priority-Queue)

// use reverseOrder to account for max-heap implementation

PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

pq.addAll(task\_counter.values());

int time = 0;

while(pq.size() > 0) {

List<Integer> add\_back = new ArrayList<>();

// loop for grabbing n tasks from pq or for idle itme only

for(int i= 0; i <= n; i++) {

// for grabbing n tasks from pq

if(pq.size() > 0) {

int count = pq.poll(); // retreives max and removes an element from pq.

count--;

if(count > 0) { // needs add\_back

add\_back.add(count);

}

}

time += 1; // accounts for idle time and real task too.

if(pq.size() == 0 && add\_back.size() == 0) { // this accounts when the last run of tasks is done and no more tasks to ensure no idle runs happen when no tasks to schecule.

break;

}

}

pq.addAll(add\_back); // re-arrange the max-heap proeprty by adding the add\_back counts to pq.

}

return time;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>