题目地址(1834. 单线程 CPU)

https://leetcode-cn.com/problems/single-threaded-cpu/

题目描述

给你一个二维数组 tasks ，用于表示 n 项从 0 到 n - 1 编号的任务。其中 tasks[i] = [enqueueTimei, processingTimei] 意味着第 i 项任务将

现有一个单线程 CPU ，同一时间只能执行 最多一项 任务，该 CPU 将会按照下述方式运行：

如果 CPU 空闲，且任务队列中没有需要执行的任务，则 CPU 保持空闲状态。
如果 CPU 空闲，但任务队列中有需要执行的任务，则 CPU 将会选择 执行时间最短 的任务开始执行。如果多个任务具有同样的最短执行时间，则选择下标最小的
一旦某项任务开始执行，CPU 在 执行完整个任务 前都不会停止。
CPU 可以在完成一项任务后，立即开始执行一项新任务。

返回 CPU 处理任务的顺序。

示例 1：

输入：tasks = [[1,2],[2,4],[3,2],[4,1]]
输出：[0,2,3,1]
解释：事件按下述流程运行：
- time = 1 ，任务 0 进入任务队列，可执行任务项 = {0}
- 同样在 time = 1 ，空闲状态的 CPU 开始执行任务 0 ，可执行任务项 = {}
- time = 2 ，任务 1 进入任务队列，可执行任务项 = {1}
- time = 3 ，任务 2 进入任务队列，可执行任务项 = {1, 2}
- 同样在 time = 3 ，CPU 完成任务 0 并开始执行队列中用时最短的任务 2 ，可执行任务项 = {1}
- time = 4 ，任务 3 进入任务队列，可执行任务项 = {1, 3}
- time = 5 ，CPU 完成任务 2 并开始执行队列中用时最短的任务 3 ，可执行任务项 = {1}
- time = 6 ，CPU 完成任务 3 并开始执行任务 1 ，可执行任务项 = {}
- time = 10 ，CPU 完成任务 1 并进入空闲状态

示例 2：

输入：tasks = [[7,10],[7,12],[7,5],[7,4],[7,2]]
输出：[4,3,2,0,1]
解释：事件按下述流程运行：
- time = 7 ，所有任务同时进入任务队列，可执行任务项 = {0,1,2,3,4}
- 同样在 time = 7 ，空闲状态的 CPU 开始执行任务 4 ，可执行任务项 = {0,1,2,3}
- time = 9 ，CPU 完成任务 4 并开始执行任务 3 ，可执行任务项 = {0,1,2}
- time = 13 ，CPU 完成任务 3 并开始执行任务 2 ，可执行任务项 = {0,1}
- time = 18 ，CPU 完成任务 2 并开始执行任务 0 ，可执行任务项 = {1}
- time = 28 ，CPU 完成任务 0 并开始执行任务 1 ，可执行任务项 = {}
- time = 40 ，CPU 完成任务 1 并进入空闲状态

提示：

```
tasks.length == n
1 <= n <= 105
1 <= enqueueTimei, processingTimei <= 109
```

前置知识

- 模拟
- 堆

标签

- 模拟
- 堆

难度

- 中等

入选理由

- 难度适中，同时联动了后面的专题《堆》

公司

- 暂无

思路

对于这道题，直接模拟即可。模拟就是直接按照题目描述写代码就行。

简单模拟题直接模拟就行， 中等模拟题则通常需要结合其他知识点。对于这道题来说， 就需要大家结合 **堆** 来完成。

题目说我们需要按照任务的先后顺序处理任务，并且：

- 如果当前没有正在处理任务时，直接处理。
- 如果当前正在处理任务， 则将其放入任务队列。处理完成之后从任务队列拿任务，而拿任务的依据就是**任务** 的时间长短，具体来说就是优先拿任务时长短的。

根据上面的描述，我们可以发现应该先对 task 按照开始时间进行排序。由于排序会破坏原有的顺序，而题目的返回是排序前的索引，因此排序后仍然需要维护排序前的索引。

另外任务队列中每次都取时间最短，这提示我们使用堆来存任务队列，并用任务时长做为 key，这是因为堆特别适合处理**动态极值**问题。如果不太熟悉堆也没关系，我们后面会讲解，如果现在做不出来，大家也可以到时候回过头来再做这道题。

那么如何用代码模拟上述过程呢?

我们用 time 表示当前的时间，time 从 0 开始，用 pos 记录我们处理到的 tasks。（由于我们进行了排序，因此 pos 从 0 开始处理，当处理完所有的 tasks，模拟结束）

具体来说：

1. 如果任务队列没有任务，那么直接将 time 快进到下一个任务的开始时间 ，这样可以减少时间复杂度。
2. 将 time 之前开始的任务全部加入到任务队列中，表示这些任务都可以被处理了。
3. 从任务队列中取出一个时间最短的进行处理。（这是题目要求的）
4. 重复 1 - 3 直到 n 个任务都被处理完毕。

关键点

- 堆

代码

- 语言支持：Python3，JS，Java，CPP

Python3 Code:

```
class Solution:
    def getOrder(self, tasks: List[List[int]]) -> List[int]:
        tasks = [(task[0], i, task[1]) for i, task in enumerate(tasks)]
        tasks.sort()
        backlog = []
        time = 0
        ans = []
        pos = 0
        for _ in tasks:
            if not backlog:
                time = max(time, tasks[pos][0])
            while pos < len(tasks) and tasks[pos][0] <= time:
                heapq.heappush(backlog, (tasks[pos][2], tasks[pos][1]))
                pos += 1
            d, j = heapq.heappop(backlog)
            time += d
            ans.append(j)
```

```
        return ans
```

JS Code:

```
/**
 * @param {number[][]} tasks
 * @return {number[]}
 */
const getOrder = function (tasks) {
  const queue = new MinPriorityQueue();
  tasks = tasks.map((task, index) => ({
    index,
    start: task[0],
    time: task[1],
  }));
  tasks.sort((a, b) => b.start - a.start);
  const answer = [];
  let time = 0;
  while (tasks.length > 0 || !queue.isEmpty()) {
    // 队列为空，且没有任务能加入队列，直接跳过时间
    if (queue.isEmpty() && tasks[tasks.length - 1].start > time) {
      time = tasks[tasks.length - 1].start;
    }

    // 向队列中加入可执行任务
    while (tasks.length > 0) {
      if (tasks[tasks.length - 1].start <= time) {
        const task = tasks.pop();
        queue.enqueue(task, task.time * 100000 + task.index);
      } else {
        break;
      }
    }

    // 执行任务
    const { element: task } = queue.dequeue();
    time += task.time;
    answer.push(task.index);
  }

  return answer;
};
```

Java Code:

```java
class Solution {
    public int[] getOrder(int[][] tasks) {
        int n = tasks.length;
        int[] ans = new int[n];
        int[][] extTasks = new int[n][3];
        for(int i = 0; i < n; i++) {
            extTasks[i][0] = i;
            extTasks[i][1] = tasks[i][0];
            extTasks[i][2] = tasks[i][1];
        }
        Arrays.sort(extTasks, (a,b)->a[1] - b[1]);
        PriorityQueue<int[]> pq = new PriorityQueue<int[]>((a, b) -> a[2] == b[2] ? a[0] - b[0] : a[2] - b[2]);
        int time = 0;
        int ai = 0;
        int ti = 0;
        while(ai < n) {
            while(ti < n && extTasks[ti][1] <= time) {
                pq.offer(extTasks[ti++]);

            }
            if(pq.isEmpty()) {
                time = extTasks[ti][1];
                continue;
            }
            int[] bestFit = pq.poll();
            ans[ai++] = bestFit[0];
            time += bestFit[2];
        }
        return ans;
    }
}
```

CPP Code :

```cpp

class Solution {
private:
    using PII = pair<int, int>;
    using LL = long long;

public:
    vector<int> getOrder(vector<vector<int>>& tasks) {
        int n = tasks.size();
        vector<int> indices(n);
        iota(indices.begin(), indices.end(), 0);
        sort(indices.begin(), indices.end(), [&](int i, int j) {
            return tasks[i][0] < tasks[j][0];
        });
```

```
        vector<int> ans;
        // 优先队列
        priority_queue<PII, vector<PII>, greater<PII>> q;
        // 时间戳
        LL time = 0;
        // 数组上遍历的指针
        int ptr = 0;

        for (int i = 0; i < n; ++i) {
            // 如果没有可以执行的任务，直接快进
            if (q.empty()) {
                time = max(time, (LL)tasks[indices[ptr]][0]);
            }
            // 将所有小于等于时间戳的任务放入优先队列
            while (ptr < n && tasks[indices[ptr]][0] <= time) {
                q.emplace(tasks[indices[ptr]][1], indices[ptr]);
                ++ptr;
            }
            // 选择处理时间最小的任务
            auto&& [process, index] = q.top();
            time += process;
            ans.push_back(index);
            q.pop();
        }

        return ans;
    }
};
```

复杂度分析

令 n 为数组长度。

- 时间复杂度：$O(nlogn)$
- 空间复杂度：$O(n)$

> 此题解由 [力扣刷题插件] 自动生成。

力扣的小伙伴可以[关注我]，这样就会第一时间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

关注公众号力扣加加，努力用清晰直白的语言还原解题思路，并且有大量图解，手把手教你识别套路，高效刷题。

欢迎长按关注

上一页 下一页