【二分查找】-项目排期

题目描述与示例

题目描述

项目组共有 `N` 个开发人员，项目经理接到了 `M` 个独立的需求，每个需求的工作量不同，且每个需求只能由一个开发人员独立完成，不能多人合作。假定各个需求之间无任何先后依赖关系，请设计算法帮助项目经理进行工作安排，使整个项目能用最少的时间交付。

输入描述

第一行输入为 `M` 个需求的工作量，单位为天，用逗号隔开。

例如： `X1 X2 X3 … Xm` 。表示共有 `M` 个需求，每个需求的工作量分别为 `X1` 天， `X2` 天… `Xm` 天。

其中 `0 < M < 30` ； `0 < Xm < 200`

第二行输入为项目组人员数量 `N`

输出描述

最快完成所有工作的天数

示例

输入

```
1 6 2 7 7 9 3 2 1 3 11 4
2 2
```

输出

```
1 28
```

说明

共有两位员工，其中一位分配需求 `6 2 7 7 3 2 1` 共需要 `28` 天完成，另一位分配需求 `9 3 11 4` 共需要 `27` 天完成，故完成所有工作至少需要 `28` 天。

解题思路

又是一道描述相当晦涩的题目（很想骂人）

用比较简洁的数学语言来描述就是，将数组 `X1, X2, X3, ... ,Xm` 分为 `N` 部分（并非子数组，不要求连续），设每一部分的和为 `sum1, sum2, ..., sumN`，要求找到一种分配方式使得 `max(sum1, sum2, ..., sumN)` 最小。

用一句简单的话来说，就是**最小化各部分求和的最大值**。这种设问一定要想到用二分来完成。

将问题转化为，我们需要找到一个阈值 `k`（一个人的最大工作量），并将原数组 `nums` 可以被分为 `N` 部分（分配给 `N` 个人），使得这 `N` 部分的各自求和的最大值都不会超过 `k`（每个人各自的工作量不会超过 `k`）。

显然 `k` 的选择是一个二段性问题：

- 当 `k` 非常小时，原数组 `nums` 无论怎么分配都无法完成要求
- 当 `k` 非常大时，原数组 `nums` 无论如何分配都可以完成要求
- 必然存在一个临界值 `k`，使得存在 `nums` 的分配结果恰好完成要求。

我们希望找到这个阈值 `k`，因此需要对 `k` 进行二分查找，二分查找的范围为 `[max(nums), sum(nums)]`。当

- `k = max(nums)` 时，能够被分为 `m = len(nums)` 部分（需要 `m` 个人来完成所有工作）
- `k = sum(nums)` 时，能够被分为 `1` 部分（只由 `1` 个人可以完成所有工作）

而上述二分过程的贪心子问题为：当我们选择了阈值 `k` 时，数组 `nums` 能否被分割不超过 `N` 部分？

这个问题就和 【贪心】2023B-数据最节约的备份方法 几乎完全一致了，其代码为

```python
def sub_question(k, nums, m, N):
    ans = 0
    check = [0] * m
    for i in range(m):
        if check[i] == 1:
            continue
        ans += 1
        cur_sum = 0
        j = i
        while j < m:
            if check[j] == 1:
                j += 1
                continue
            if nums[j] + cur_sum > k:
                j += 1
            else:
                cur_sum += nums[j]
                check[j] = 1
                j += 1
    return ans <= N
```

初始化左闭右开区间 `left = max(nums)` ， `right = sum(nums) + 1` ，进行二分。

计算 `mid = (left + right) // 2` 。当

- `sub_question(mid, nums, m, N)` 为 `True` 时，说明当选择了阈值 `k = mid` 时，可以将任务分配给 `N` 个人（组数小于等于 `N` ）。此时的 `mid` 的选择偏大，区间应该左移， `right` 左移。
- `sub_question(mid, nums, m, N)` 为 `False` 时，说明当选择了阈值 `k = mid` 时，无法将任务分配给 `N` 个人（组数大于 `N` ）。此时的 `mid` 的选择偏小，区间应该右移， `left` 右移。

故结合贪心子问题，整体的二分代码为

```python
nums = list(map(int, input().split()))
m = len(nums)
N = int(input())
nums.sort(reverse = True)
left, right = max(nums), sum(nums)+1
while left < right:
    mid = (right + left) // 2
    if sub_question(mid, nums, m, N):
        right = mid
    else:
        left = mid + 1
```

```
12
13 print(left)
```

代码

Python

```python
# 题目：【二分查找】2024E-项目排期
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：二分查找/贪心
# 代码看不懂的地方，请直接在群上提问
# 相关题目：【贪心】2023B-数据最节约的备份方法


# 贪心子问题，当选择了阈值k时，如果m个任务nums可以被分配给N个员工，
# 且每一个员工的工作总量不超过k则返回True，否则返回False
# （注意此处nums必须是一个已排序好的逆序数组）
# 该子问题的时间复杂度与nums的长度m相关，为O(m^2)
def sub_question(k, nums, m, N):
    ans = 0
    # 初始化长度为m的check数组，用来表示第i个任务是否已经分配给了某一个员工
    check = [0] * m
    # 遍历所有nums每一个工作量
    for i in range(m):
        # 如果该工作已经由某个员工完成了，则直接跳过
        if check[i] == 1:
            continue
        # 否则，需要一个新的员工
        # 来完成包含工作nums[i]在内的若干工作
        # 故ans更新
        ans += 1
        # 初始化当前员工所做的工作总量为0
        cur_sum = 0
        # 初始化变量j为i，用于修改当前这个员工的工作情况
        j = i
        # 进行内层循环，此处涉及贪心算法
        while j < m:
            # 如果第j个工作已经安排，则j直接递增，跳过第j个工作
            if check[j] == 1:
                j += 1
                continue
```

```
            # 如果第j份工作和当前员工之前的工作量之和超过k
            # 则这个员工不能选择这份工作，j递增
            if nums[j] + cur_sum > k:
                j += 1
            # 如果第j份工作和当前员工之前的工作量之和不超过k
            # 则贪心地将这份工作安排给这个员工
            # 修改cur_sum和check[j]，j递增
            else:
                cur_sum += nums[j]
                check[j] = 1
                j += 1
    # 退出循环时，如果需要的人数ans不超过N，则返回True，否则返回False
    return ans <= N


# 输入m个任务构成的数组
nums = list(map(int, input().split()))
# 获得nums的长度，即任务数量
m = len(nums)
# 输入员工人数N
N = int(input())
# 对nums进行逆序排序，方便后续贪心子问题的计算
nums.sort(reverse = True)
# 初始化左闭右开
left, right = max(nums), sum(nums)+1
while left < right:
    mid = (right + left) // 2
    # 如果选择了mid作为阈值，可以将任务分配给N个人（组数小于等于N）
    # 说明mid的选择偏大，区间应该左移，right左移
    if sub_question(mid, nums, m, N):
        right = mid
    # 如果选择了mid作为阈值，无法将任务分配给N个人（组数多于N）
    # 说明mid的选择偏小，区间应该右移，left右移
    else:
        left = mid + 1

# 退出循环时，存在left = right是恰好可以将任务分配给N个人的阈值k
# left或right即为答案
print(left)
```

Java

```
import java.util.Arrays;
import java.util.Scanner;

```

```java
public class Main {
    // 贪心子问题
    private static boolean subQuestion(int k, int[] nums, int m, int N) {
        int ans = 0;
        int[] check = new int[m];

        for (int i = 0; i < m; i++) {
            if (check[i] == 1) continue;
            ans++;
            int curSum = 0;
            int j = i;
            while (j < m) {
                if (check[j] == 1) {
                    j++;
                    continue;
                }
                if (nums[j] + curSum > k) {
                    j++;
                } else {
                    curSum += nums[j];
                    check[j] = 1;
                    j++;
                }
            }
        }
        return ans <= N;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入m个任务构成的数组
        String[] numsStr = scanner.nextLine().split(" ");
        int m = numsStr.length;
        int[] nums = new int[m];
        for (int i = 0; i < m; i++) {
            nums[i] = Integer.parseInt(numsStr[i]);
        }

        // 输入员工人数N
        int N = scanner.nextInt();

        // 对nums进行逆序排序
        Arrays.sort(nums);
        for (int i = 0; i < m / 2; i++) {
            int temp = nums[i];
            nums[i] = nums[m - i - 1];
```

```
            nums[m - i - 1] = temp;
        }

        // 初始化左闭右开
        int left = nums[0], right = Arrays.stream(nums).sum() + 1;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (subQuestion(mid, nums, m, N)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        // 输出结果
        System.out.println(left);
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <numeric>

using namespace std;

bool subQuestion(int k, vector<int>& nums, int m, int N) {
    int ans = 0;
    vector<int> check(m, 0);

    for (int i = 0; i < m; i++) {
        if (check[i] == 1) continue;
        ans++;
        int curSum = 0;
        int j = i;
        while (j < m) {
            if (check[j] == 1) {
                j++;
                continue;
            }
            if (nums[j] + curSum > k) {
```

```
                j++;
            } else {
                curSum += nums[j];
                check[j] = 1;
                j++;
            }
        }
    }
    return ans <= N;
}

int main() {
    vector<int> nums;
    string input;
    getline(cin, input);
    stringstream ss(input);
    int num;
    while (ss >> num) {
        nums.push_back(num);
    }
    int m = nums.size();
    int N;
    cin >> N;

    sort(nums.begin(), nums.end(), greater<int>());
    int left = nums[0], right = accumulate(nums.begin(), nums.end(), 0) + 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (subQuestion(mid, nums, m, N)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    cout << left << endl;
    return 0;
}

```

时空复杂度

时间复杂度：`O(log(C)m^2)`。贪心子问题的时间复杂度为`O(m^2)`，二分查找的时间复杂度为`O(logC)`，其中`C = sum(nums) - max(nums)`。

空间复杂度：`O(m)`。贪心子问题需要构建长度为`m`的`check`数组。