

【华为OD机考 统一考试机试C卷】项目排期 (C++ Java JavaScript Python C语言)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录 (C卷 + D卷 + B卷 + A卷) + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

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。假定各个需求直接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 $0 < M < 30$ ； $0 < X_m < 200$

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

用例

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

给定一系列任务的工作量和一定数量的工人，计算完成所有任务所需的最少天数，使得每个工人分配到的任务总工作量不超过这个天数。这是一个典型的搜索问题，可以通过回溯法和二分查找结合来解决。

1. 排序和反转任务数组:

- 使用 `Arrays.sort(tasks)` 对任务数组进行升序排序，然后通过一个循环将数组反转，使其成为降序。这样做是为了优先分配工作量大的任务，从而更高效地利用工人的工作时间。

2. 二分查找:

- 为了找到完成所有任务所需的最少天数，使用二分查找确定这个最小值。设置两个指针 `l` 和 `r`，分别表示可能的最短时间的下界和上界。`l` 初始化为数组中的最大值（即最大的单个任务工作量），`r` 初始化为所有任务工作量的总和。
- 在 `l` 小于 `r` 的条件下进行循环，计算中间值 `mid`，并使用 `canFinish` 函数检查是否可以在 `mid` 天内完成所有任务。
- 如果可以完成，则将上界 `r` 设置为 `mid`，否则将下界 `l` 设置为 `mid + 1`。
- 当 `l` 和 `r` 相遇时，`l` 即为所求的最少天数。

3. 回溯法:

- `canFinish` 函数使用回溯法来检查在给定的时间限制 `limit` 内是否可以完成所有任务。
- 创建一个长度为工人数量 `k` 的数组 `workers`，用于记录每个工人的当前工作量。
- 使用 `backtrack` 函数递归地尝试为每个任务分配工人，直到所有任务都被分配或者无法在时间限制内完成分配。

- 在 `backtrack` 函数中，如果当前工人可以在时间限制内完成当前任务，则将任务分配给他，并递归地尝试分配下一个任务。
- 如果分配成功，则返回 `true`；如果当前路径无法成功分配所有任务，则回溯到上一个状态，尝试其他可能的分配方案。
- 如果所有方案都无法成功，则返回 `false`。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <numeric>
5 #include <sstream>
6 using namespace std;
7 // 回溯法
8 bool backtrack(vector<int>& tasks, vector<int>& workers, int index, int limit) {
9     // 如果所有任务都已分配, 则返回true
10    if (index >= tasks.size()) {
11        return true;
12    }
13
14    // 获取当前任务的工作量
15    int current = tasks[index];
16    // 尝试将当前任务分配给每个员工
17    for (int i = 0; i < workers.size(); i++) {
18        // 如果当前员工可以在时间限制内完成这项任务
19        if (workers[i] + current <= limit) {
20            // 分配任务给当前员工
21            workers[i] += current;
22            // 继续尝试分配下一个任务
23            if (backtrack(tasks, workers, index + 1, limit)) {
24                return true;
25            }
26            // 回溯, 取消当前的任务分配
27            workers[i] -= current;
28        }
29
30        // 如果当前员工没有任务或者加上当前任务刚好达到时间限制, 则不需要尝试其他员工
31        if (workers[i] == 0 || workers[i] + current == limit) {
32            break;
33        }
34    }
```

```
34     }
35
36     // 如果无法分配当前任务, 则返回false
37     return false;
38 }
39 // 检查是否可以在给定的时间限制内完成所有任务
40 bool canFinish(vector<int>& tasks, int k, int limit) {
41     // 创建一个数组来记录每个员工的工作量
42     vector<int> workers(k, 0);
43     // 使用回溯法检查是否可以完成
44     return backtrack(tasks, workers, 0, limit);
45 }
46 // 计算完成所有任务所需的最少天数
47 int minimumTimeRequired(vector<int>& tasks, int k) {
48     // 将任务按工作量降序排序
49     sort(tasks.begin(), tasks.end(), greater<int>());
50
51     // 使用二分查找确定完成所有任务的最短时间
52     int l = tasks[0], r = accumulate(tasks.begin(), tasks.end(), 0);
53     while (l < r) {
54         int mid = (l + r) / 2;
55         // 检查当前时间限制是否足够完成所有任务
56         if (canFinish(tasks, k, mid)) {
57             r = mid;
58         } else {
59             l = mid + 1;
60         }
61     }
62
63     // 返回最短完成时间
64     return l;
65 }
66
67
68
69
70 int main() {
71     // 使用cin读取输入
72     vector<int> tasks;
73     string input;
74
75 }
```

```
75     getline(cin, input);
76     istringstream iss(input);
77     int value;
78     while (iss >> value) {
79         tasks.push_back(value);
80     }
81     int N;
82     cin >> N;
83
84     // 输出最快完成所有工作的天数
85     cout << minimumTimeRequired(tasks, N) << endl;
86     return 0;
}
```

Java

```
1  import java.util.Arrays;
2  import java.util.Scanner;
3
4  public class Main {
5      public static void main(String[] args) {
6          // 使用Scanner读取输入
7          Scanner scanner = new Scanner(System.in);
8          // 读取第一行输入, 即需求的工作量, 并以空格分隔
9          String[] workloads = scanner.nextLine().split(" ");
10         // 读取第二行输入, 即项目组人员数量
11         int N = Integer.parseInt(scanner.nextLine());
12         // 创建一个数组来存放每个需求的工作量
13         int[] tasks = new int[workloads.length];
14
15         // 将输入的工作量转换为整数并存入数组
16         for (int i = 0; i < workloads.length; i++) {
17             tasks[i] = Integer.parseInt(workloads[i]);
18         }
19
20         // 输出最快完成所有工作的天数
21         System.out.println(minimumTimeRequired(tasks, N));
22     }
23
24     // 计算完成所有任务所需的最少天数
25 }
```

```
25 public static int minimumTimeRequired(int[] tasks, int k) {
26     // 将任务按工作量升序排序
27     Arrays.sort(tasks);
28     // 将排序后的数组反转, 使之成为降序
29     int low = 0, high = tasks.length - 1;
30     while (low < high) {
31         int temp = tasks[low];
32         tasks[low] = tasks[high];
33         tasks[high] = temp;
34         low++;
35         high--;
36     }
37
38     // 使用二分查找确定完成所有任务的最短时间
39     int l = tasks[0], r = Arrays.stream(tasks).sum();
40     while (l < r) {
41         int mid = (l + r) / 2;
42         // 检查当前时间限制是否足够完成所有任务
43         if (canFinish(tasks, k, mid)) {
44             r = mid;
45         } else {
46             l = mid + 1;
47         }
48     }
49
50     // 返回最短完成时间
51     return l;
52 }
53
54 // 检查是否可以在给定的时间限制内完成所有任务
55 private static boolean canFinish(int[] tasks, int k, int limit) {
56     // 创建一个数组来记录每个员工的工作量
57     int[] workers = new int[k];
58     // 使用回溯法检查是否可以完成
59     return backtrack(tasks, workers, 0, limit);
60 }
61
62 // 回溯法
63 private static boolean backtrack(int[] tasks, int[] workers, int index, int limit) {
64     // 如果所有任务都已分配, 则返回true
65     --
```

```
66     if (index >= tasks.length) {
67         return true;
68     }
69
70     // 获取当前任务的工作量
71     int current = tasks[index];
72     // 尝试将当前任务分配给每个员工
73     for (int i = 0; i < workers.length; i++) {
74         // 如果当前员工可以在时间限制内完成这项任务
75         if (workers[i] + current <= limit) {
76             // 分配任务给当前员工
77             workers[i] += current;
78             // 继续尝试分配下一个任务
79             if (backtrack(tasks, workers, index + 1, limit)) {
80                 return true;
81             }
82             // 回溯, 取消当前的任务分配
83             workers[i] -= current;
84         }
85
86         // 如果当前员工没有任务或者加上当前任务刚好达到时间限制, 则不需要尝试其他员工
87         if (workers[i] == 0 || workers[i] + current == limit) {
88             break;
89         }
90     }
91
92     // 如果无法分配当前任务, 则返回false
93     return false;
94 }
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  // ...
```



```
8 // 读取输入
9 rl.on('line', (line) => {
10   if (!this.tasks) {
11     // 第一次输入, 处理任务工作量
12     this.tasks = line.split(' ').map(Number);
13   } else {
14     // 第二次输入, 处理员工数量
15     const N = Number(line);
16     // 输出最快完成所有工作的天数
17     console.log(minimumTimeRequired(this.tasks, N));
18     rl.close();
19   }
20 });
21
22 // 计算完成所有任务所需的最少天数
23 function minimumTimeRequired(tasks, k) {
24   // 将任务按工作量降序排序
25   tasks.sort((a, b) => b - a);
26
27   // 使用二分查找确定完成所有任务的最短时间
28   let l = tasks[0], r = tasks.reduce((a, b) => a + b, 0);
29   while (l < r) {
30     let mid = Math.floor((l + r) / 2);
31     // 检查当前时间限制是否足够完成所有任务
32     if (canFinish(tasks, k, mid)) {
33       r = mid;
34     } else {
35       l = mid + 1;
36     }
37   }
38
39   // 返回最短完成时间
40   return l;
41 }
42
43 // 检查是否可以在给定的时间限制内完成所有任务
44 function canFinish(tasks, k, limit) {
45   // 创建一个数组来记录每个员工的工作量
46   let workers = new Array(k).fill(0);
47   // 使用回溯法检查是否可以完成
48   --
```

```
49     return backtrack(tasks, workers, 0, limit);
50 }
51
52 // 回溯法
53 function backtrack(tasks, workers, index, limit) {
54     // 如果所有任务都已分配, 则返回true
55     if (index >= tasks.length) {
56         return true;
57     }
58
59     // 获取当前任务的工作量
60     let current = tasks[index];
61     // 尝试将当前任务分配给每个员工
62     for (let i = 0; i < workers.length; i++) {
63         // 如果当前员工可以在时间限制内完成这项任务
64         if (workers[i] + current <= limit) {
65             // 分配任务给当前员工
66             workers[i] += current;
67             // 继续尝试分配下一个任务
68             if (backtrack(tasks, workers, index + 1, limit)) {
69                 return true;
70             }
71             // 回溯, 取消当前的任务分配
72             workers[i] -= current;
73         }
74
75         // 如果当前员工没有任务或者加上当前任务刚好达到时间限制, 则不需要尝试其他员工
76         if (workers[i] === 0 || workers[i] + current === limit) {
77             break;
78         }
79     }
80
81     // 如果无法分配当前任务, 则返回false
82     return false;
83 }
```

Python

```
1 # Python版本代码
2 from itertools import combinations
3
```

```
3
4
5 def minimumTimeRequired(tasks, k):
6     # 将任务按工作量降序排序
7     tasks.sort(reverse=True)
8
9     # 使用二分查找确定完成所有任务的最短时间
10    l, r = tasks[0], sum(tasks)
11    while l < r:
12        mid = (l + r) // 2
13        # 检查当前时间限制是否足够完成所有任务
14        if canFinish(tasks, k, mid):
15            r = mid
16        else:
17            l = mid + 1
18
19    # 返回最短完成时间
20    return l
21
22 def canFinish(tasks, k, limit):
23     # 创建一个数组来记录每个员工的工作量
24     workers = [0] * k
25     # 使用回溯法检查是否可以完成
26     return backtrack(tasks, workers, 0, limit)
27
28 def backtrack(tasks, workers, index, limit):
29     # 如果所有任务都已分配, 则返回True
30     if index >= len(tasks):
31         return True
32
33     # 获取当前任务的工作量
34     current = tasks[index]
35     # 尝试将当前任务分配给每个员工
36     for i in range(len(workers)):
37         # 如果当前员工可以在时间限制内完成这项任务
38         if workers[i] + current <= limit:
39             # 分配任务给当前员工
40             workers[i] += current
41             # 继续尝试分配下一个任务
42             if backtrack(tasks, workers, index + 1, limit):
43                 return True
44     return False
```

```
44     # 回溯, 取消当前的任务分配
45     workers[i] -= current
46
47     # 如果当前员工没有任务或者加上当前任务刚好达到时间限制, 则不需要尝试其他员工
48     if workers[i] == 0 or workers[i] + current == limit:
49         break
50
51     # 如果无法分配当前任务, 则返回False
52     return False
53
54 if __name__ == "__main__":
55     # 使用input读取输入
56     tasks = list(map(int, input().split()))
57     N = int(input())
58
59     # 输出最快完成所有工作的天数
60     print(minimumTimeRequired(tasks, N))
```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_TASKS 30 // 定义最大任务数量的常量, 用于设置任务数组的最大长度
6
7  // 用于qsort函数的比较函数, 实现降序排序
8  int compare(const void *a, const void *b) {
9      // 将void指针转换为int指针, 并解引用获取值进行比较
10     return (*(int*)b - *(int*)a);
11 }
12
13 // 回溯法分配任务
14 int backtrack(int *tasks, int *workers, int index, int limit, int k, int taskSize) {
15     // 检查是否所有任务都已分配
16     if (index >= taskSize) {
17         return 1; // 如果是, 返回1表示成功
18     }
19
20     // 获取当前要分配的任务
21     int task = tasks[index];
```

```
21     int current = tasks[index];
22     // 遍历所有员工
23     for (int i = 0; i < k; i++) {
24         // 检查当前员工是否可以在时间限制内完成这个任务
25         if (workers[i] + current <= limit) {
26             // 如果可以, 分配任务并递归尝试分配下一个任务
27             workers[i] += current;
28             if (backtrack(tasks, workers, index + 1, limit, k, taskSize)) {
29                 return 1;
30             }
31             // 如果不成功, 回溯, 即撤销这次任务分配
32             workers[i] -= current;
33         }
34
35         // 如果当前员工没有任务或者加上当前任务刚好达到时间限制, 则不需要尝试其他员工
36         if (workers[i] == 0 || workers[i] + current == limit) {
37             break;
38         }
39     }
40
41     // 如果无法分配当前任务, 返回0表示失败
42     return 0;
43 }
44
45 // 检查是否能在指定时间内完成所有任务
46 int canFinish(int *tasks, int k, int limit, int taskSize) {
47     // 初始化一个记录员工当前任务量的数组
48     int workers[MAX_TASKS] = {0};
49     // 调用回溯法尝试分配任务
50     return backtrack(tasks, workers, 0, limit, k, taskSize);
51 }
52
53 // 计算完成所有任务的最短时间
54 int minimumTimeRequired(int *tasks, int k, int taskSize) {
55     // 先对任务进行降序排序
56     qsort(tasks, taskSize, sizeof(int), compare);
57
58     // 二分查找的左右边界, 左边界为最大单个任务时间, 右边界为所有任务时间总和
59     int l = tasks[0], r = 0;
60     for (int i = 0; i < taskSize; i++) {
61         --
```

```
62     r += tasks[i];
63 }
64
65 // 二分查找最短完成时间
66 while (l < r) {
67     int mid = l + (r - l) / 2;
68     // 检查是否能在mid时间内完成所有任务
69     if (canFinish(tasks, k, mid, taskSize)) {
70         r = mid;
71     } else {
72         l = mid + 1;
73     }
74 }
75
76 // 返回最短完成时间
77 return l;
78 }
79
80 int main() {
81     // 存储任务的数组和任务数量
82     int tasks[MAX_TASKS], taskSize = 0;
83     // 读取一行输入作为任务工作量
84     char input[200];
85     fgets(input, 200, stdin);
86     // 使用strtok分割字符串, 将分割后的数字转换为int存入任务数组
87     char *token = strtok(input, " ");
88     while (token != NULL) {
89         tasks[taskSize++] = atoi(token);
90         token = strtok(NULL, " ");
91     }
92
93     // 读取员工数量
94     int N;
95     scanf("%d", &N);
96
97     // 计算并输出完成所有任务的最短时间
98     printf("%d\n", minimumTimeRequired(tasks, N, taskSize));
99     return 0;
100 }
```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例

解题思路

C++

Java

JavaScript

Python

C语言

机考真题 华为OD



CSDN @算法大师