

【华为OD机考 统一考试机试C卷】数据最节约的备份方法 (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：数据最节约的备份方法

有若干个文件，使用刻录光盘的方式进行备份，假设每张光盘的容量是500MB，求使用光盘最少的文件分布方式

所有文件的大小都是整数的MB，且不超过500MB；文件不能分割、分卷打包

输入描述：

一组文件大小的数据

输出描述：

使用光盘的数量

不用考虑输入数据不合法的情况；假设最多100个输入文件。

用例1

输入：

1 | 100,500,300,200,400

输出:

1 | 3

说明:

(100,400),(200,300),(500) 3张光盘即可。

输入和输出内容都不含空格。

用例2

输入:

1 | 1,100,200,300

输出:

1 | 2

python

```
1 |
2 |
3 | from typing import List
4 |
5 | # 定义一个函数, 用于检查是否能够将所有文件分配到指定数量的光盘中
6 | def check(mid: int, files: List[int]) -> bool:
7 |     # 创建一个长度为mid的光盘列表, 每个光盘的容量为500MB
8 |     discs = [500 for _ in range(mid)]
9 |
10 |    # 遍历文件列表, 从最大的文件开始分配
11 |    for i in range(len(files) - 1, -1, -1):
12 |        file_size = files[i]
13 |
14 |        # 将光盘列表按照容量从小到大排序
15 |        discs.sort()
16 |
17 |        # 如果当前最大容量的光盘可以容纳该文件, 则将该文件分配到该光盘中
18 |
```

```
19     if discs[mid - 1] >= file_size:
20         discs[mid - 1] -= file_size
21     else:
22         # 如果当前最大容量的光盘无法容纳该文件, 则返回False
23         return False
24
25     # 如果所有文件都成功分配到光盘中, 则返回True
26     return True
27
28 # 获取输入的文件大小数据
29 files = list(map(int, input().split(',')))
30
31 # 将文件大小列表按照从小到大排序
32 files.sort()
33
34 # 初始化最小和最大光盘数量
35 min_discs = 0
36 max_discs = len(files) + 1
37
38 # 使用二分法查找最少光盘数量
39 while min_discs < max_discs:
40     mid = (min_discs + max_discs) // 2
41
42     # 检查是否能够将所有文件分配到mid个光盘中
43     if check(mid, files):
44         max_discs = mid
45     else:
46         min_discs = mid + 1
47
48 # 输出最少光盘数量
49 print(min_discs)
```

javascript

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

```
6   });
7
8   // 定义一个函数, 用于检查是否能够将所有文件分配到指定数量的光盘中
9   function check(mid, files) {
10    // 创建一个长度为mid的光盘列表, 每个光盘的容量为500MB
11    let discs = new Array(mid).fill(500);
12
13    // 遍历文件列表, 从最大的文件开始分配
14    for (let i = files.length - 1; i >= 0; i--) {
15      let file_size = files[i];
16
17      // 将光盘列表按照容量从小到大排序
18      discs.sort();
19
20      // 如果当前最大容量的光盘可以容纳该文件, 则将该文件分配到该光盘中
21      if (discs[mid - 1] >= file_size) {
22        discs[mid - 1] -= file_size;
23      } else {
24        // 如果当前最大容量的光盘无法容纳该文件, 则返回false
25        return false;
26      }
27    }
28
29    // 如果所有文件都成功分配到光盘中, 则返回true
30    return true;
31  }
32
33  rl.on('line', (input) => {
34    // 获取输入的文件大小数据
35    let files = input.split(',').map(Number);
36
37    // 将文件大小列表按照从小到大排序
38    files.sort();
39
40    // 初始化最小和最大光盘数量
41    let min_discs = 0;
42    let max_discs = files.length + 1;
43
44    // 使用二分法查找最少光盘数量
45    while (min_discs < max_discs) {
46      --
```

```
47     let mid = Math.floor((min_discs + max_discs) / 2);
48
49     // 检查是否能够将所有文件分配到mid个光盘中
50     if (check(mid, files)) {
51         max_discs = mid;
52     } else {
53         min_discs = mid + 1;
54     }
55 }
56
57 // 输出最少光盘数量
58 console.log(min_discs);
59
60 r1.close();
61 };
```

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  bool check(int mid, vector<int>& files) {
8      vector<int> discs(mid, 500);
9
10     for (int i = files.size() - 1; i >= 0; i--) {
11         int file_size = files[i];
12
13         sort(discs.begin(), discs.end());
14
15         if (discs[mid - 1] >= file_size) {
16             discs[mid - 1] -= file_size;
17         } else {
18             return false;
19         }
20     }
21 }
```

```
22     return true;
23 }
24
25 int main() {
26     vector<int> files;
27     string input;
28     getline(cin, input);
29
30     size_t pos = 0;
31     string token;
32     while ((pos = input.find(',')) != string::npos) {
33         token = input.substr(0, pos);
34         files.push_back(stoi(token));
35         input.erase(0, pos + 1);
36     }
37     files.push_back(stoi(input));
38
39     sort(files.begin(), files.end());
40
41     int min_discs = 0;
42     int max_discs = files.size() + 1;
43
44     while (min_discs < max_discs) {
45         int mid = (min_discs + max_discs) / 2;
46
47         if (check(mid, files)) {
48             max_discs = mid;
49         } else {
50             min_discs = mid + 1;
51         }
52     }
53
54     cout << min_discs << endl;
55
56     return 0;
57 }
```

java

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3 import java.util.List;
4 import java.util.Scanner;
5
6 public class Main {
7     public static void main(String[] args) {
8         Scanner scanner = new Scanner(System.in);
9         String[] filesString = scanner.nextLine().split(",");
10        int[] files = new int[filesString.length];
11        for (int i = 0; i < filesString.length; i++) {
12            files[i] = Integer.parseInt(filesString[i]);
13        }
14        Arrays.sort(files);
15        int minDiscs = 0;
16        int maxDiscs = files.length + 1;
17        while (minDiscs < maxDiscs) {
18            int mid = (minDiscs + maxDiscs) / 2;
19            if (check(mid, files)) {
20                maxDiscs = mid;
21            } else {
22                minDiscs = mid + 1;
23            }
24        }
25        System.out.println(minDiscs);
26    }
27
28    public static boolean check(int mid, int[] files) {
29        int[] discs = new int[mid];
30        Arrays.fill(discs, 500);
31        for (int i = files.length - 1; i >= 0; i--) {
32            int fileSize = files[i];
33            Arrays.sort(discs);
34            if (discs[mid - 1] >= fileSize) {
35                discs[mid - 1] -= fileSize;
36            } else {
37                return false;
38            }
39        }
40        return true;
41    }
```



```
41 |     }  
42 | }
```

C语言

```
1  #include <stdio.h>  
2  #include <stdlib.h>  
3  #include <string.h>  
4  
5  #define MAX_FILES 100 // 假设最多100个输入文件  
6  #define DISC_CAPACITY 500 // 光盘容量  
7  
8  // 检查是否可以用mid数量的光盘来备份所有文件  
9  int check(int mid, int files[], int file_count) {  
10     int discs[MAX_FILES]; // 存储每张光盘剩余容量  
11     for (int i = 0; i < mid; i++) {  
12         discs[i] = DISC_CAPACITY;  
13     }  
14  
15     for (int i = file_count - 1; i >= 0; i--) {  
16         int file_size = files[i];  
17  
18         // 对光盘剩余容量进行排序, 以便尽可能地填满每张光盘  
19         for (int j = 0; j < mid - 1; j++) {  
20             for (int k = j + 1; k < mid; k++) {  
21                 if (discs[j] > discs[k]) {  
22                     int temp = discs[j];  
23                     discs[j] = discs[k];  
24                     discs[k] = temp;  
25                 }  
26             }  
27         }  
28  
29         // 如果最大的光盘剩余容量都放不下当前文件, 则返回失败  
30         if (discs[mid - 1] < file_size) {  
31             return 0;  
32         } else {  
33             // 否则放入该文件, 并更新光盘剩余容量  
34             discs[mid - 1] -= file_size;  
35         }  
36     }
```

```
36     }
37
38     return 1; // 所有文件都能放入光盘中, 返回成功
39 }
40
41 int main() {
42     int files[MAX_FILES]; // 存储文件大小
43     int file_count = 0;   // 文件数量
44     char input[1000];     // 输入字符串
45
46     // 读取输入
47     scanf("%[^\\n]", input);
48
49     // 解析输入字符串, 提取文件大小
50     char *token = strtok(input, ",");
51     while (token != NULL) {
52         files[file_count++] = atoi(token);
53         token = strtok(NULL, ",");
54     }
55
56     // 对文件大小进行排序
57     for (int i = 0; i < file_count - 1; i++) {
58         for (int j = i + 1; j < file_count; j++) {
59             if (files[i] > files[j]) {
60                 int temp = files[i];
61                 files[i] = files[j];
62                 files[j] = temp;
63             }
64         }
65     }
66
67     // 使用二分查找来确定最少需要的光盘数量
68     int min_discs = 1;
69     int max_discs = file_count;
70     while (min_discs < max_discs) {
71         int mid = min_discs + (max_discs - min_discs) / 2;
72         if (check(mid, files, file_count)) {
73             max_discs = mid;
74         } else {
75             min_discs = mid + 1;
76         }
77     }
```

```
77 |     }  
78 | }  
79 |  
80 | // 输出最少需要的光盘数量  
81 | printf("%d\n", min_discs);  
82 |  
83 | return 0;  
   | }  
   | }
```

完整用例

用例1

100,500,300,200,400

用例2

500,500,500,500

用例3

100,100,100,100,100,100

用例4

300,200,100

用例5

400,400,400,400,400

用例6

100,200,300,400,500,100,200,300,400,500

用例7

400,400,400,400,400,400,400,400,400,400

用例8

500,500,500,500,500,500,500,500,500

用例9

200,200,200,200,200,200,200,200,200

用例10

300,200,100,300,200,100,300,200,100

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：数据最节约的备份方法
 - 输入描述：
 - 输出描述：
 - 用例1
 - 用例2
 - python
 - javascript
 - C++
 - java
 - C语言
 - 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10

机考真题 华为OD



CSDN @算法大师