【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：

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

输出：

```
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：

```
1,100,200,300
```

输出：

```
2
```

python

```python


from typing import List

# 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
def check(mid: int, files: List[int]) -> bool:
    # 创建一个长度为mid的光盘列表，每个光盘的容量为500MB
    discs = [500 for _ in range(mid)]

    # 遍历文件列表，从最大的文件开始分配
    for i in range(len(files) - 1, -1, -1):
        file_size = files[i]

        # 将光盘列表按照容量从小到大排序
        discs.sort()

        # 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

```

```
        if discs[mid - 1] >= file_size:
            discs[mid - 1] -= file_size
        else:
            # 如果当前最大容量的光盘无法容纳该文件，则返回False
            return False

    # 如果所有文件都成功分配到光盘中，则返回True
    return True

# 获取输入的文件大小数据
files = list(map(int, input().split(',')))

# 将文件大小列表按照从小到大排序
files.sort()

# 初始化最小和最大光盘数量
min_discs = 0
max_discs = len(files) + 1

# 使用二分法查找最少光盘数量
while min_discs < max_discs:
    mid = (min_discs + max_discs) // 2

    # 检查是否能够将所有文件分配到mid个光盘中
    if check(mid, files):
        max_discs = mid
    else:
        min_discs = mid + 1

# 输出最少光盘数量
print(min_discs)
```

javascript

```
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
```

```
});

// 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
function check(mid, files) {
  // 创建一个长度为mid的光盘列表，每个光盘的容量为500MB
  let discs = new Array(mid).fill(500);

  // 遍历文件列表，从最大的文件开始分配
  for (let i = files.length - 1; i >= 0; i--) {
    let file_size = files[i];

    // 将光盘列表按照容量从小到大排序
    discs.sort();

    // 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
    if (discs[mid - 1] >= file_size) {
      discs[mid - 1] -= file_size;
    } else {
      // 如果当前最大容量的光盘无法容纳该文件，则返回False
      return false;
    }
  }

  // 如果所有文件都成功分配到光盘中，则返回True
  return true;
}

rl.on('line', (input) => {
  // 获取输入的文件大小数据
  let files = input.split(',').map(Number);

  // 将文件大小列表按照从小到大排序
  files.sort();

  // 初始化最小和最大光盘数量
  let min_discs = 0;
  let max_discs = files.length + 1;

  // 使用二分法查找最少光盘数量
  while (min_discs < max_discs) {
```

```
    let mid = Math.floor((min_discs + max_discs) / 2);

    // 检查是否能够将所有文件分配到mid个光盘中
    if (check(mid, files)) {
      max_discs = mid;
    } else {
      min_discs = mid + 1;
    }
  }

  // 输出最少光盘数量
  console.log(min_discs);

  rl.close();
});
```

C++

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

using namespace std;

bool check(int mid, vector<int>& files) {
   vector<int> discs(mid, 500);

   for (int i = files.size() - 1; i >= 0; i--) {
       int file_size = files[i];

       sort(discs.begin(), discs.end());

       if (discs[mid - 1] >= file_size) {
           discs[mid - 1] -= file_size;
       } else {
           return false;
       }
   }

```

```
    return true;
}

int main() {
    vector<int> files;
    string input;
    getline(cin, input);

    size_t pos = 0;
    string token;
    while ((pos = input.find(',')) != string::npos) {
        token = input.substr(0, pos);
        files.push_back(stoi(token));
        input.erase(0, pos + 1);
    }
    files.push_back(stoi(input));

    sort(files.begin(), files.end());

    int min_discs = 0;
    int max_discs = files.size() + 1;

    while (min_discs < max_discs) {
        int mid = (min_discs + max_discs) / 2;

        if (check(mid, files)) {
            max_discs = mid;
        } else {
            min_discs = mid + 1;
        }
    }

    cout << min_discs << endl;

    return 0;
}
```

java

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

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] filesString = scanner.nextLine().split(",");
        int[] files = new int[filesString.length];
        for (int i = 0; i < filesString.length; i++) {
            files[i] = Integer.parseInt(filesString[i]);
        }
        Arrays.sort(files);
        int minDiscs = 0;
        int maxDiscs = files.length + 1;
        while (minDiscs < maxDiscs) {
            int mid = (minDiscs + maxDiscs) / 2;
            if (check(mid, files)) {
                maxDiscs = mid;
            } else {
                minDiscs = mid + 1;
            }
        }
        System.out.println(minDiscs);
    }

    public static boolean check(int mid, int[] files) {
        int[] discs = new int[mid];
        Arrays.fill(discs, 500);
        for (int i = files.length - 1; i >= 0; i--) {
            int fileSize = files[i];
            Arrays.sort(discs);
            if (discs[mid - 1] >= fileSize) {
                discs[mid - 1] -= fileSize;
            } else {
                return false;
            }
        }
        return true;
```

```
    }
}
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_FILES 100  // 假设最多100个输入文件
#define DISC_CAPACITY 500  // 光盘容量

// 检查是否可以用mid数量的光盘来备份所有文件
int check(int mid, int files[], int file_count) {
    int discs[MAX_FILES];  // 存储每张光盘剩余容量
    for (int i = 0; i < mid; i++) {
        discs[i] = DISC_CAPACITY;
    }

    for (int i = file_count - 1; i >= 0; i--) {
        int file_size = files[i];

        // 对光盘剩余容量进行排序，以便尽可能地填满每张光盘
        for (int j = 0; j < mid - 1; j++) {
            for (int k = j + 1; k < mid; k++) {
                if (discs[j] > discs[k]) {
                    int temp = discs[j];
                    discs[j] = discs[k];
                    discs[k] = temp;
                }
            }
        }

        // 如果最大的光盘剩余容量都放不下当前文件，则返回失败
        if (discs[mid - 1] < file_size) {
            return 0;
        } else {
            // 否则放入该文件，并更新光盘剩余容量
            discs[mid - 1] -= file_size;
        }
```

```
    }

    return 1;  // 所有文件都能放入光盘中，返回成功
}

int main() {
    int files[MAX_FILES];  // 存储文件大小
    int file_count = 0;    // 文件数量
    char input[1000];      // 输入字符串

    // 读取输入
    scanf("%[^\n]", input);

    // 解析输入字符串，提取文件大小
    char *token = strtok(input, ",");
    while (token != NULL) {
        files[file_count++] = atoi(token);
        token = strtok(NULL, ",");
    }

    // 对文件大小进行排序
    for (int i = 0; i < file_count - 1; i++) {
        for (int j = i + 1; j < file_count; j++) {
            if (files[i] > files[j]) {
                int temp = files[i];
                files[i] = files[j];
                files[j] = temp;
            }
        }
    }

    // 使用二分查找来确定最少需要的光盘数量
    int min_discs = 1;
    int max_discs = file_count;
    while (min_discs < max_discs) {
        int mid = min_discs + (max_discs - min_discs) / 2;
        if (check(mid, files, file_count)) {
            max_discs = mid;
        } else {
            min_discs = mid + 1;
```

```
        }
    }

    // 输出最少需要的光盘数量
    printf("%d\n", min_discs);

    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

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

用例9

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

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
No No No
CSDN @算法大师