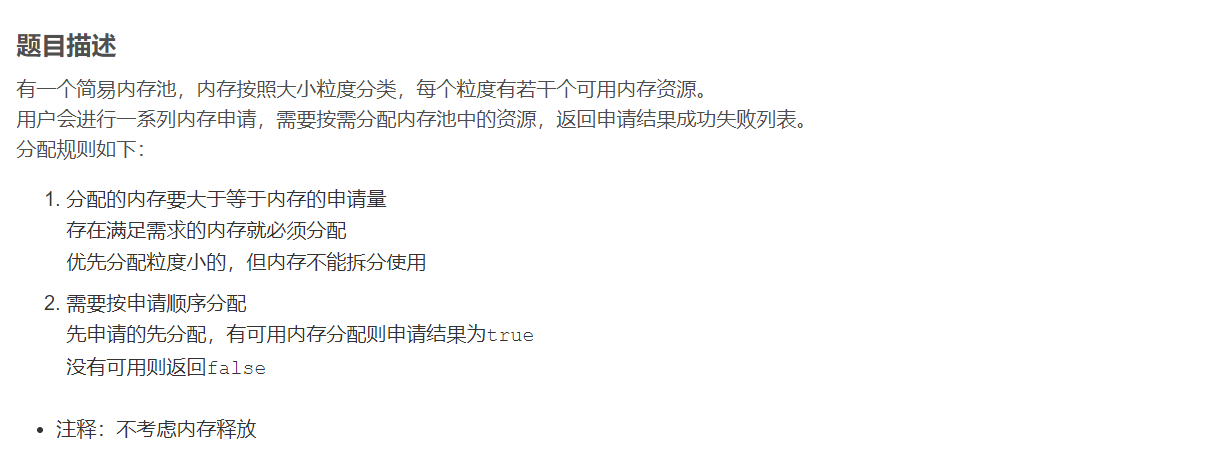
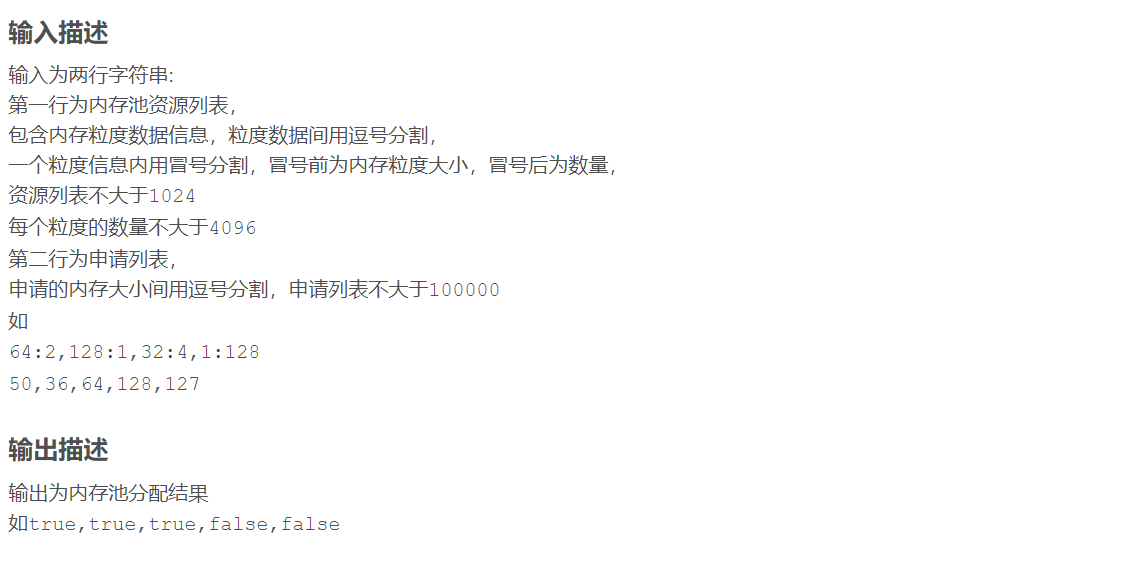
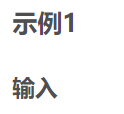
# **E卷-内存资源共享[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**





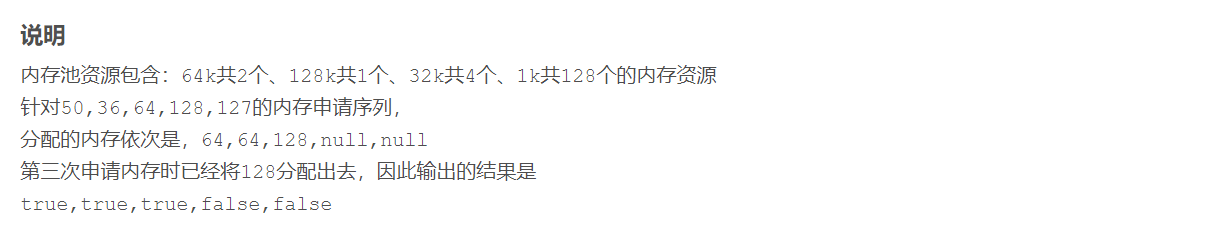


64:2,128:1,32:4,1:128

50,36,64,128,127



true,true,true,false,false





import java.util.ArrayList; // 导入ArrayList类

import java.util.List; // 导入List接口

import java.util.Scanner; // 导入Scanner类

public class Main {

static ArrayList<int[]> arr = new

ArrayList<>(); // 创建一个存储内存池资源的ArrayList，每个元素是内存粒度和数量的二元数组

static int[] query; // 创建一个存储申请内存大小的数组

// 用于读取输入数据的方法

static void read() {

Scanner sc = new Scanner(

System.in); // 创建一个Scanner对象以读取用户输入

String[] tmp =

sc.nextLine().split(","); // 按照逗号分割第一行字符串，获取内存池资源列表

// 遍历内存池资源列表，解析每个内存粒度和数量，并存储到arr列表中

for (String x : tmp) {

int f = x.indexOf(":"); // 找到冒号的位置

arr.add(new int[] {Integer.parseInt(x.substring(0, f)), Integer.parseInt(x.substring(f + 1))}); // 解析内存粒度和数量，并存储到arr中

}

tmp = sc.nextLine().split(","); // 按照逗号分割第二行字符串，获取申请内存列表

int n = tmp.length; // 获取申请列表的长度

query = new int[n]; // 初始化申请内存大小数组

// 遍历申请列表，将每个申请内存大小转换为整数并存储到query数组中

for (int i = 0; i < n; ++i) {

query[i] = Integer.parseInt(tmp[i]);

}

}

// 用于处理内存分配逻辑的方法

static void work() {

arr.sort((x, y) -> (x[0] -

y[0])); // 按照内存粒度从小到大排序内存池资源列表

ArrayList<Boolean> res = new

ArrayList<>(); // 创建一个存储分配结果的ArrayList，每个元素是布尔值

// 遍历申请内存列表，逐个处理内存申请

for (int x : query) {

boolean f = false; // 初始化当前申请的分配结果为false

// 遍历内存池资源列表，寻找符合条件的内存资源

for (int j = 0; j < arr.size(); ++j) {

int[] y = arr.get(j); // 获取当前的内存粒度和数量

// 判断当前内存资源是否可用且内存粒度大于等于申请量

if (y[1] > 0 && y[0] >= x) {

f = true; // 找到可用内存，则将分配结果设为true

y[1] -= 1; // 内存数量减1

arr.set(j, y); // 更新内存池资源列表

break; // 终止内层循环，直接分配当前找到的内存资源

}

}

res.add(f); // 将当前申请的分配结果添加到结果列表中

}

// 输出分配结果列表，每个结果用逗号分隔

for (int i = 0; i < res.size(); ++i) {

if (i != 0) System.out.print(","); // 输出逗号分隔符

System.out.print(res.get(i)); // 输出当前结果

}

}

// 主程序入口

public static void main(String[] args) {

read(); // 调用读取输入数据的方法

work(); // 调用处理内存分配逻辑的方法

}

}



import sys

# 定义全局变量保存内存池资源和申请需求

arr = []

query = []

# 读取输入数据

def read():

global arr, query

# 读取并分割内存池资源列表

inputs = sys.stdin.read().split()

first\_line = inputs[0]

second\_line = inputs[1]

arr = []

for x in first\_line.split(","):

size, count = x.split(":")

arr.append((int(size), int(count)))

query = list(map(int, second\_line.split(",")))

# 分配内存的方法

def work():

arr.sort() # 按照内存粒度大小进行排序

res = [] # 存储分配结果

# 遍历每个内存申请需求

for x in query:

f = False # 标记是否找到合适内存

# 遍历内存池资源列表

for i in range(len(arr)):

size, count = arr[i]

# 判断是否有可用且合适的内存资源

if count > 0 and size >= x:

f = True

arr[i] = (size, count - 1)

break

res.append(f)

print(",".join(map(str, res)).lower())

if \_\_name\_\_ == "\_\_main\_\_":

read()

work()



#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// 存储内存池资源和申请需求的全局变量

vector<pair<int, int>> arr;

vector<int> query;

// 读取输入数据的方法

void read() {

string line;

// 读取内存池资源列表

getline(cin, line);

stringstream ss(line);

string temp;

while (getline(ss, temp, ',')) {

int pos = temp.find(':');

int size = stoi(temp.substr(0, pos));

int count = stoi(temp.substr(pos + 1));

arr.push\_back({size, count});

}

// 读取申请内存列表

getline(cin, line);

ss.clear();

ss.str(line);

while (getline(ss, temp, ',')) {

int size = stoi(temp);

query.push\_back(size);

}

}

// 分配内存的方法

void work() {

// 按照内存粒度大小进行排序

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

vector<bool> res; // 存储分配结果

// 遍历每个申请需求

for (int x : query) {

bool f = false; // 标记是否找到合适内存

// 遍历内存池资源列表

for (auto& y : arr) {

int size = y.first;

int& count = y.second;

// 判断是否有可用且合适的内存

if (count > 0 && size >= x) {

f = true;

--count;

break;

}

}

res.push\_back(f);

}

// 输出结果

for (size\_t i = 0; i < res.size(); ++i) {

if (i != 0) cout << ",";

cout << (res[i] ? "true" : "false");

}

cout << endl;

}

int main() {

read();

work();

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 定义内存池资源和申请列表的全局变量

typedef struct {

int size;

int count;

} Memory;

Memory arr[1024];

int arr\_size = 0;

int query[100000];

int query\_size = 0;

// 读取输入数据的方法

void read() {

char line[4096];

// 读取内存池资源列表

fgets(line, sizeof(line), stdin);

char\* token = strtok(line, ",");

while (token != NULL) {

int pos = strchr(token, ':') - token;

int size = atoi(token);

int count = atoi(token + pos + 1);

arr[arr\_size++] = (Memory) {

size, count

};

token = strtok(NULL, ",");

}

// 读取申请内存列表

fgets(line, sizeof(line), stdin);

token = strtok(line, ",");

while (token != NULL) {

query[query\_size++] = atoi(token);

token = strtok(NULL, ",");

}

}

// 比较函数用于排序

int cmp(const void\* a, const void\* b) {

return ((Memory\*)a)->size - ((Memory\*)b)->size;

}

// 分配内存的方法

void work() {

qsort(arr, arr\_size, sizeof(Memory), cmp); // 按内存粒度大小排序

int res[query\_size]; // 存储分配结果

// 遍历每个申请需求

for (int i = 0; i < query\_size; ++i) {

int x = query[i];

int f = 0; // 标记是否找到合适内存

// 遍历内存池资源列表

for (int j = 0; j < arr\_size; ++j) {

if (arr[j].count > 0 && arr[j].size >= x) {

f = 1;

--arr[j].count;

break;

}

}

res[i] = f;

}

// 输出结果

for (int i = 0; i < query\_size; ++i) {

if (i != 0) printf(",");

printf("%s", res[i] ? "true" : "false");

}

printf("\n");

}

int main() {

read();

work();

return 0;

}



const readline = require("readline");

// 创建Readline接口以读取用户输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

// 监听输入数据并存储到数组中

rl.on("line", (line) => {

input.push(line);

if (input.length === 2) {

rl.close();

}

});

// 关闭Readline接口后处理输入数据

rl.on("close", () => {

let arr = [];

let query = [];

// 解析内存池资源列表

input[0].split(",").forEach((x) => {

const [size, count] = x.split(":").map(Number);

arr.push([size, count]);

});

// 解析申请内存列表

query = input[1].split(",").map(Number);

// 按内存粒度大小排序

arr.sort((a, b) => a[0] - b[0]);

const res = []; // 存储分配结果

// 遍历每个申请需求

for (const x of query) {

let allocated = false; // 标记是否找到合适内存

// 遍历内存池资源列表

for (let i = 0; i < arr.length; i++) {

let [size, count] = arr[i];

// 判断是否有可用且合适的内存

if (count > 0 && size >= x) {

arr[i][1] -= 1;

allocated = true;

break;

}

}

res.push(allocated);

}

// 输出结果

console.log(res.join(","));

});



package main

import (

"bufio"

"fmt"

"os"

"sort"

"strconv"

"strings"

)

// 定义内存池资源和申请列表的全局变量

var arr []struct {

size int

count int

}

var query []int

// 读取输入数据的方法

func read() {

reader := bufio.NewReader(os.Stdin)

// 读取内存池资源列表

line, \_ := reader.ReadString('\n')

line = strings.TrimSpace(line)

for \_, x := range strings.Split(line, ",") {

parts := strings.Split(x, ":")

size, \_ := strconv.Atoi(parts[0])

count, \_ := strconv.Atoi(parts[1])

arr = append(arr, struct {

size int

count int

}{size, count})

}

// 读取申请内存列表

line, \_ = reader.ReadString('\n')

line = strings.TrimSpace(line)

for \_, x := range strings.Split(line, ",") {

val, \_ := strconv.Atoi(x)

query = append(query, val)

}

}

// 分配内存的方法

func work() {

// 按照内存粒度大小进行排序

sort.Slice(arr, func(i, j int) bool {

return arr[i].size < arr[j].size

})

res := make([]bool, len(query)) // 存储分配结果

// 遍历每个申请需求

for i, x := range query {

allocated := false // 标记是否找到合适内存

// 遍历内存池资源列表

for j := range arr {

if arr[j].count > 0 && arr[j].size >= x {

arr[j].count--

allocated = true

break

}

}

res[i] = allocated

}

// 输出结果

for i, v := range res {

if i > 0 {

fmt.Print(",")

}

fmt.Print(v)

}

fmt.Println()

}

func main() {

read()

work()

}