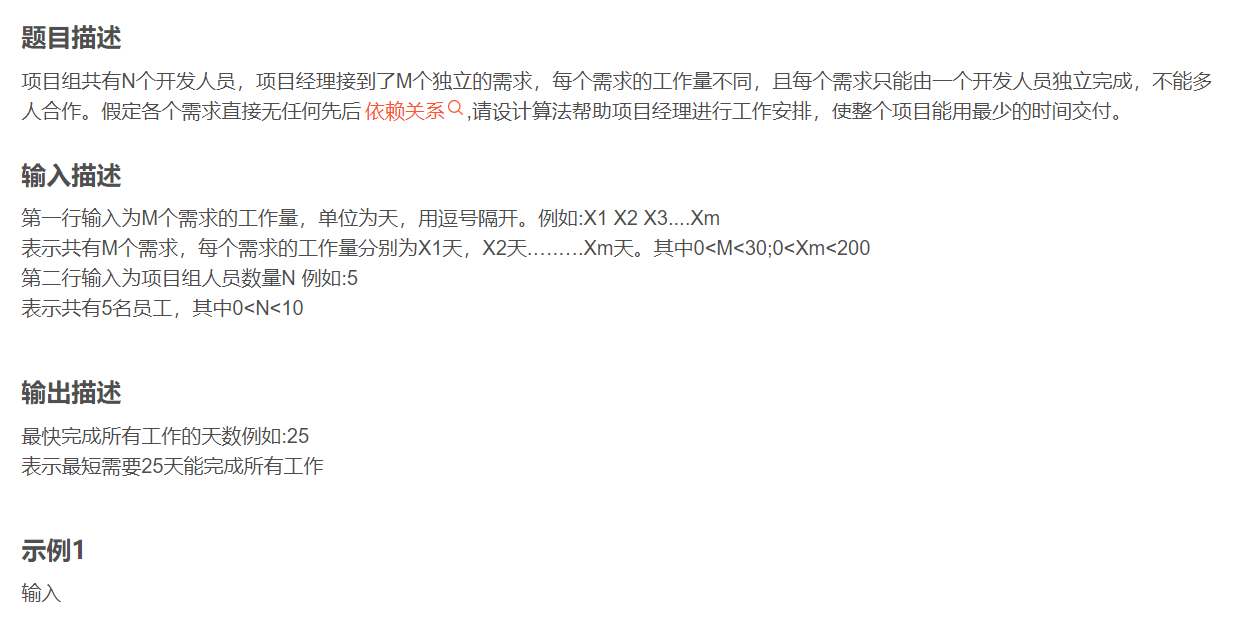
# **E卷-项目排期[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**

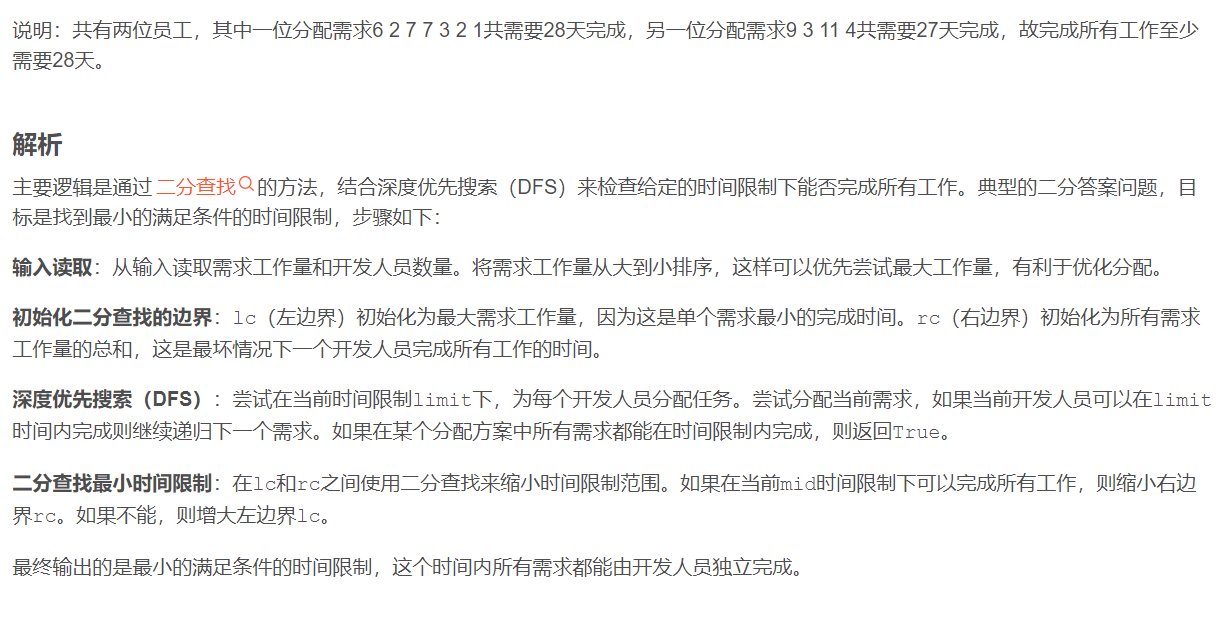


6 2 7 7 9 3 2 1 3 11 4

2



28





import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取M个需求的工作量，单位为天

String[] jobStrings = scanner.nextLine().split(" ");

int[] jobs = new int[jobStrings.length];

for (int i = 0; i < jobStrings.length; i++) {

jobs[i] = Integer.parseInt(jobStrings[i]);

}

// 读取项目组人员数量

int k = scanner.nextInt();

// 将工作量从大到小进行排序

Arrays.sort(jobs);

reverseArray(jobs);

int lc = jobs[0], rc = Arrays.stream(jobs).sum();

// 使用二分查找法找到完成所有工作的最短时间

while (lc < rc) {

int mid = (lc + rc) >> 1;

if (can(jobs, k, mid)) {

rc = mid;

} else {

lc = mid + 1;

}

}

System.out.println(lc);

}

// 辅助函数：检查是否可以在时间限制 x 内完成所有工作

private static boolean can(int[] jobs, int k, int limit) {

int[] vis = new int[k];

return dfs(jobs, vis, 0, limit);

}

// 深度优先搜索（DFS）判断是否能在限定的时间 limit 内完成工作

private static boolean dfs(int[] jobs, int[] vis, int i, int limit) {

if (i >= jobs.length) return true;

int now = jobs[i];

for (int j = 0; j < vis.length; j++) {

if (vis[j] + now <= limit) {

vis[j] += now;

if (dfs(jobs, vis, i + 1, limit)) return true;

vis[j] -= now;

}

if (vis[j] == 0 || vis[j] + now == limit) break;

}

return false;

}

// 辅助函数：反转数组

private static void reverseArray(int[] arr) {

int left = 0, right = arr.length - 1;

while (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}



# 读取第一个输入行，表示M个需求的工作量，并将其转换为整数列表

jobs = [int(x) for x in input().split()]

# 读取第二个输入行，表示项目组的人数

k = int(input())

# 将需求的工作量从大到小进行排序，这样可以优先处理大的工作量

jobs.sort(reverse=True)

# 初始化最小时间限制为需求中的最大工作量，最大时间限制为所有需求工作量总和

lc, rc = jobs[0], sum(jobs)

# 深度优先搜索（DFS）判断是否能在限定的时间limit内完成工作

def dfs(vis, i, limit):

# 如果所有工作都已经分配完毕，返回True

if i >= len(jobs):

return True

now = jobs[i]

# 遍历所有开发人员，尝试分配当前的工作

for j in range(len(vis)):

# 如果当前开发人员加上这个工作后仍然在时间限制以内，则分配该工作

if vis[j] + now <= limit:

vis[j] += now

# 递归检查下一个工作分配情况

if dfs(vis, i + 1, limit):

return True

# 如果不成功，回溯当前分配

vis[j] -= now

# 如果当前开发人员没有工作或者当前开发人员加上这个工作正好等于时间限制，则跳出循环

if vis[j] == 0 or vis[j] + now == limit:

break

return False

# 检查是否可以在x时间限制内完成所有工作

def can(x):

vis = [0] \* k # 初始化每个开发人员的工作时间为0

return dfs(vis, 0, x)

# 使用二分查找法找到完成所有工作的最短时间

while lc < rc:

mid = (lc + rc) >> 1 # 计算中间值

if can(mid): # 检查是否可以在mid时间内完成所有工作

rc = mid # 如果可以，缩小右边界

else:

lc = mid + 1 # 如果不可以，增大左边界

print(lc) # 输出最短时间



#include <bits/stdc++.h>

using namespace std;

// 辅助函数：深度优先搜索DFS

bool dfs(const vector<int>& jobs, vector<int>& vis, int i, int limit) {

if (i >= jobs.size()) return true; // 若已分配所有工作，返回true

int now = jobs[i];

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

if (vis[j] + now <= limit) {

vis[j] += now;

if (dfs(jobs, vis, i + 1, limit)) return true;

vis[j] -= now;

}

if (vis[j] == 0 || vis[j] + now == limit) break;

}

return false;

}

// 辅助函数：检查是否可以在时间限制x内完成工作

bool can(const vector<int>& jobs, int k, int limit) {

vector<int> vis(k, 0); // 初始化每个开发人员的工作时间为0

return dfs(jobs, vis, 0, limit);

}

int main() {

string line;

getline(cin, line); // 读取任务的工作量

vector<int> jobs;

int num;

stringstream ss(line);

while (ss >> num) {

jobs.push\_back(num);

}

int k;

cin >> k; // 读取项目组人员数量

// 将工作量从大到小进行排序

sort(jobs.rbegin(), jobs.rend());

int lc = jobs[0], rc = accumulate(jobs.begin(), jobs.end(), 0);

// 使用二分查找法找到完成所有工作的最短时间

while (lc < rc) {

int mid = (lc + rc) >> 1;

if (can(jobs, k, mid)) {

rc = mid;

} else {

lc = mid + 1;

}

}

cout << lc << endl; // 输出最短时间

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 辅助函数：比较函数供qsort使用

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

return \*(int\*)b - \*(int\*)a;

}

// 深度优先搜索DFS

int dfs(int jobs[], int vis[], int n, int i, int limit, int k) {

if (i >= n) return 1; // 若已完成所有任务，返回真

int now = jobs[i];

for (int j = 0; j < k; ++j) {

if (vis[j] + now <= limit) {

vis[j] += now;

if (dfs(jobs, vis, n, i + 1, limit, k)) return 1;

vis[j] -= now;

}

if (vis[j] == 0 || vis[j] + now == limit) break;

}

return 0;

}

// 检查给定时间限制是否可以完成任务

int can(int jobs[], int n, int k, int limit) {

int vis[30] = {0}; // 初始化每个开发人员的工作时间为0

return dfs(jobs, vis, n, 0, limit, k);

}

int main() {

char line[1000];

fgets(line, 1000, stdin); // 读取任务的工作量

int jobs[30], k, n = 0;

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

while (token) {

jobs[n++] = atoi(token);

token = strtok(NULL, " ");

}

scanf("%d", &k); // 读取项目组人员数量

// 将工作量从大到小进行排序

qsort(jobs, n, sizeof(int), cmp);

int lc = jobs[0], rc = 0;

for (int i = 0; i < n; ++i) rc += jobs[i];

while (lc < rc) {

int mid = (lc + rc) / 2;

if (can(jobs, n, k, mid)) {

rc = mid;

} else {

lc = mid + 1;

}

}

printf("%d\n", lc); // 输出最短时间

return 0;

}



function can(jobs, k, limit) {

let vis = new Array(k).fill(0); // 初始化每个开发人员的工作时间为0

function dfs(i) {

if (i >= jobs.length) return true; // 若已分配所有工作，则返回真

let now = jobs[i];

for (let j = 0; j < vis.length; j++) {

if (vis[j] + now <= limit) {

vis[j] += now;

if (dfs(i + 1)) return true;

vis[j] -= now;

}

if (vis[j] === 0 || vis[j] + now === limit) break;

}

return false;

}

return dfs(0);

}

function minTime(jobs, k) {

jobs.sort((a, b) => b - a); // 将工作量从大到小进行排序

let lc = jobs[0];

let rc = jobs.reduce((acc, job) => acc + job, 0);

while (lc < rc) {

let mid = (lc + rc) >> 1;

if (can(jobs, k, mid)) {

rc = mid;

} else {

lc = mid + 1;

}

}

return lc; // 输出最短时间

}

// 读取输入

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let lines = [];

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

lines.push(line);

if (lines.length === 2) {

rl.close();

}

});

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

let jobs = lines[0].split(" ").map(Number);

let k = parseInt(lines[1]);

console.log(minTime(jobs, k));

});



package main

import (

"bufio"

"fmt"

"os"

"sort"

"strconv"

"strings"

)

// 辅助函数：深度优先搜索DFS

func dfs(jobs []int, vis []int, i, limit int) bool {

if i >= len(jobs) {

return true

}

now := jobs[i]

for j := 0; j < len(vis); j++ {

if vis[j]+now <= limit {

vis[j] += now

if dfs(jobs, vis, i+1, limit) {

return true

}

vis[j] -= now

}

if vis[j] == 0 || vis[j]+now == limit {

break

}

}

return false

}

// 辅助函数：检查是否可以在时间限制 x 内完成所有工作

func can(jobs []int, k, limit int) bool {

vis := make([]int, k)

return dfs(jobs, vis, 0, limit)

}

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取M个需求的工作量

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

line = strings.TrimSpace(line)

jobStrings := strings.Split(line, " ")

jobs := make([]int, len(jobStrings))

for i, s := range jobStrings {

jobs[i], \_ = strconv.Atoi(s)

}

// 读取项目组人员数量

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

line = strings.TrimSpace(line)

k, \_ := strconv.Atoi(line)

// 将工作量从大到小进行排序

sort.Sort(sort.Reverse(sort.IntSlice(jobs)))

lc, rc := jobs[0], 0

for \_, job := range jobs {

rc += job

}

// 使用二分查找法找到完成所有工作的最短时间

for lc < rc {

mid := (lc + rc) >> 1

if can(jobs, k, mid) {

rc = mid

} else {

lc = mid + 1

}

}

fmt.Println(lc) // 输出最短时间

}