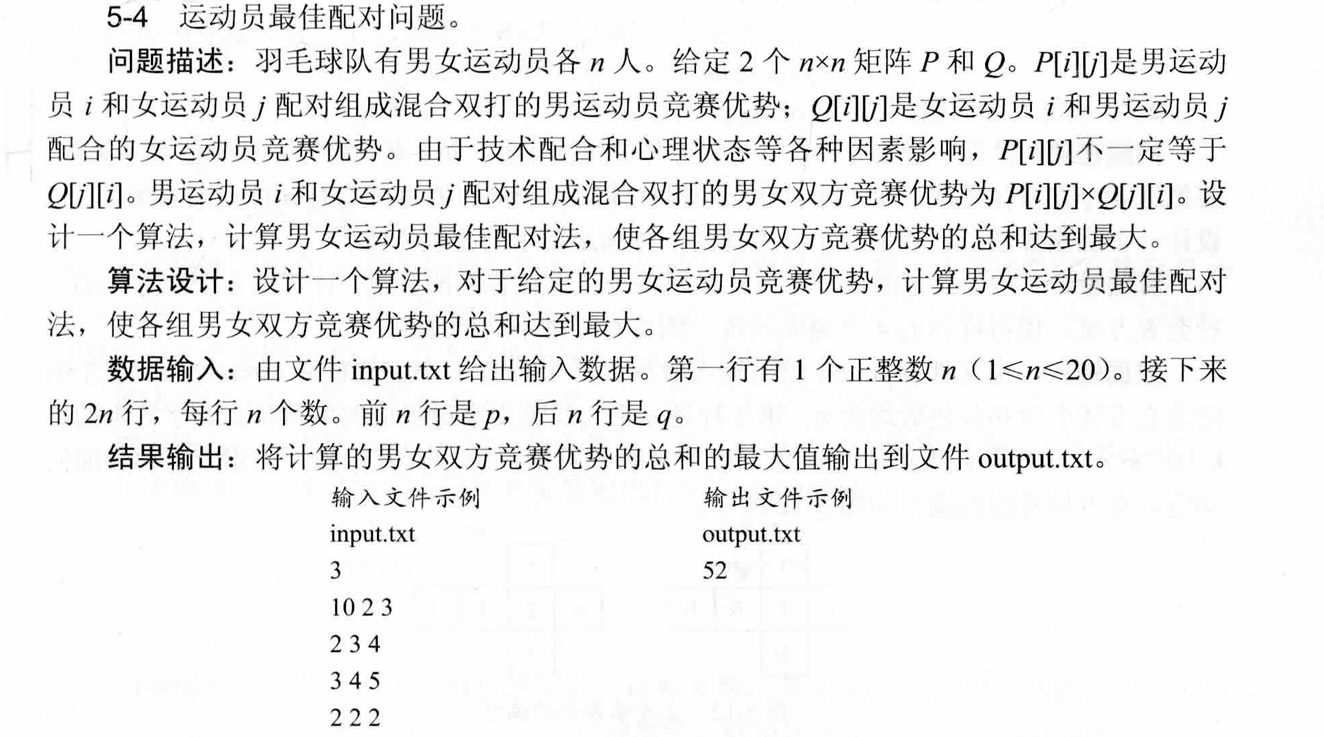
**第五章算法实现题**

学号：2209060322 姓名：梁桐 班级：计算机2203

**5-4**



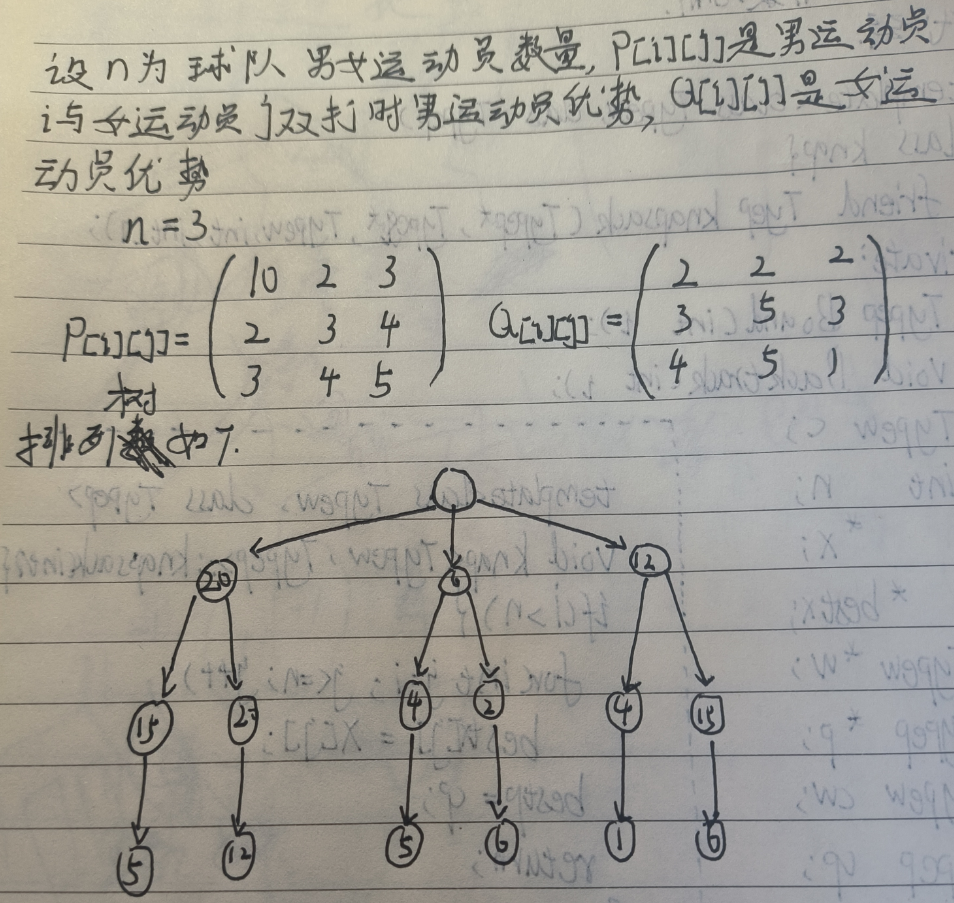
**解题思路：**

在处理男女人员搭配问题时，以羽毛球队为例，存在男女运动员的匹配安排。鉴于运动员不可重复选择这一条件，此问题的解空间树为排列树，据此可运用排列树回溯法模板进行算法设计。

设n为羽毛球队男女运动员的数量，其中，P[i][j]表示男运动员i与女运动员j配对组成混合双打时男运动员的竞赛优势，Q[i][j]表示女运动员i与男运动员j配合时女运动员的竞赛优势。

本题采用男运动员选择女运动员的方法构建排列树，以G表示女运动员，排列树的层数与男运动员相对应。

关于剪枝函数，在人员尚未完全选定之时，无法确定最终结果是否优于最优解，故在回溯过程中无需剪枝函数进行剪枝。仅当某一分支运行至叶子结点时，才需判断可行解与最优解的关系，以考虑是否更新最优解。输出结果应为包含运动员编号的集合，以x[i]表示。初始数组存放原始编号，经算法处理后，输出符合最优值的编号。



**代码：**

***#include <stdio.h>***

***#include <stdlib.h>***

***// 判断是否为一个完整解***

***int isCompleteSolution(int c[], int K) {***

***int i, flag;***

***int arr[21] = {0};***

***// 统计每个位置出现的次数***

***for (i = 1; i <= K; i++) {***

***arr[c[i]] += 1;***

***}***

***// 检查是否每个位置恰好出现一次***

***for (i = 1, flag = 1; i <= K; i++) {***

***if (arr[i]!= 1) {***

***flag = 0;***

***break;***

***}***

***}***

***return flag;***

***}***

***// 判断是否为部分解***

***int isPartialSolution(int c[], int K) {***

***int i, flag;***

***int arr[21] = {0};***

***// 统计每个位置出现的次数***

***for (i = 1; i <= K; i++) {***

***arr[c[i]] += 1;***

***}***

***// 检查是否存在重复出现的位置或未出现的位置***

***for (i = 1, flag = 0; i <= K; i++) {***

***if (arr[i] > 1) {***

***flag = 0;***

***break;***

***} else if (arr[i] == 0) {***

***flag = 1;***

***}***

***}***

***return flag;***

***}***

***// 计算给定解情况下男女运动员竞赛优势的总和***

***int calculateTotalAdvantage(int c[], int f[21][21], int K) {***

***int i, totalAdvantage = 0;***

***// 累加每个配对的竞赛优势***

***for (i = 1; i <= K; i++) {***

***totalAdvantage += f[i][c[i]];***

***}***

***return totalAdvantage;***

***}***

***int main() {***

***int i, j;***

***int N;***

***FILE \*inputFile, \*outputFile;***

***// 打开输入文件***

***inputFile = fopen("input.txt", "r");***

***if (inputFile == NULL) {***

***printf("无法打开输入文件input.txt\n");***

***return 1;***

***}***

***// 读取输入文件中的n***

***fscanf(inputFile, "%d", &N);***

***int p[21][21], q[21][21], f[21][21];***

***// p记录男运动员的竞赛优势、q记录女运动员的、f是男女运动员匹配后的。数组下标从1开始。***

***// 读取男运动员竞赛优势数据***

***for (i = 1; i <= N; i++) {***

***for (j = 1; j <= N; j++) {***

***fscanf(inputFile, "%d", &(p[i][j]));***

***}***

***}***

***// 读取女运动员竞赛优势数据并计算匹配后的优势***

***for (i = 1; i <= N; i++) {***

***for (j = 1; j <= N; j++) {***

***fscanf(inputFile, "%d", &(q[i][j]));***

***f[j][i] = q[i][j] \* p[j][i];***

***}***

***}***

***fclose(inputFile);***

***int combination[21] = {0};***

***// 用于记录男女队员的匹配，combination【i】中的i代表男队员、combination【i】代表女队员***

***int k = 1, currentAdvantage = 0, optimalAdvantage = 0;***

***// k为队员编号，currentAdvantage是当前解的优势值，optimalAdvantage是最优的优势值***

***// 回溯搜索过程***

***while (k >= 1 && k <= N) {***

***while (combination[k] <= N) {***

***combination[k]++;***

***// 如果是完整解，计算优势并更新最优解***

***if (isCompleteSolution(combination, N)) {***

***currentAdvantage = calculateTotalAdvantage(combination, f, N);***

***if (currentAdvantage > optimalAdvantage) {***

***optimalAdvantage = currentAdvantage;***

***}***

***break;***

***} else if (isPartialSolution(combination, N)) {***

***k++;***

***}***

***}***

***combination[k] = 0;***

***k--;***

***}***

***// 打开输出文件***

***outputFile = fopen("output.txt", "w");***

***if (outputFile == NULL) {***

***printf("无法打开输出文件output.txt\n");***

***return 1;***

***}***

***// 将最优优势值输出到文件***

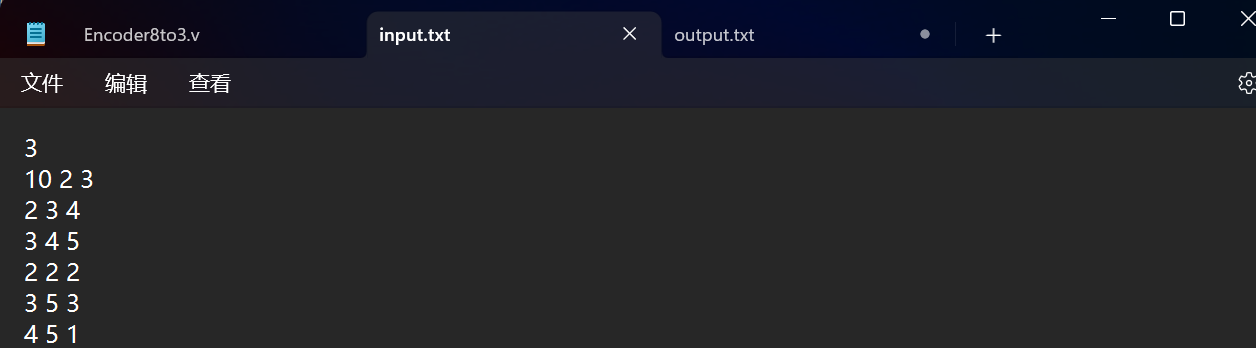
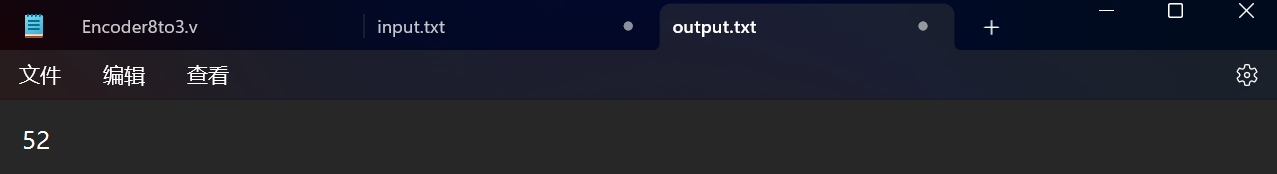
***fprintf(outputFile, "%d", optimalAdvantage);***

***fclose(outputFile);***

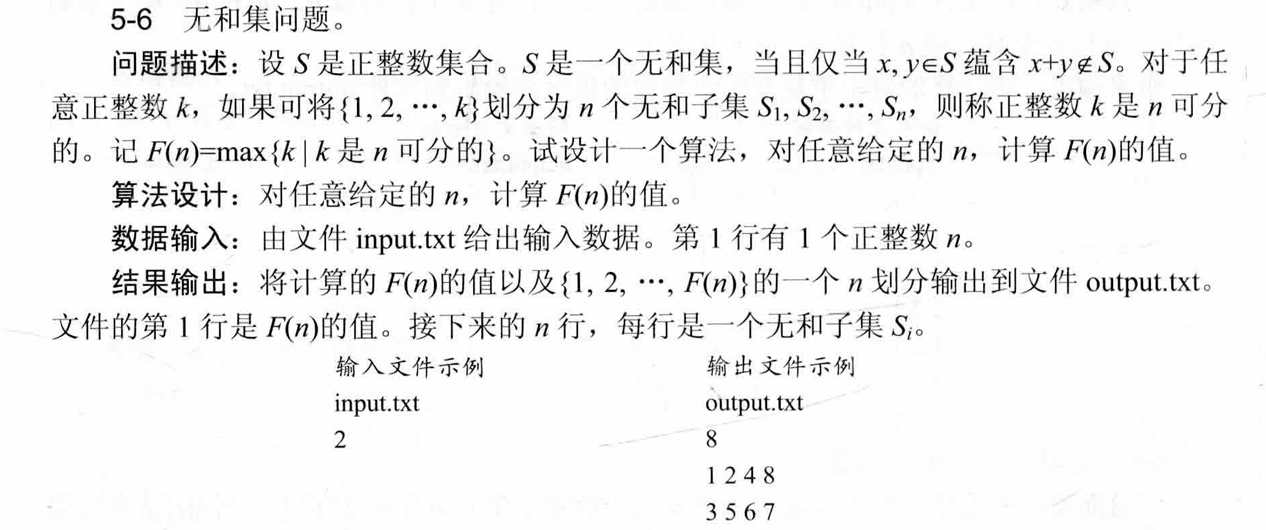
***return 0;***

***}***

**运行结果：**



**5-6**



**解题思路：**

整体思路

该算法的核心思想是通过深度优先搜索（DFS）的方式，逐步尝试将整数依次划分到不同的无和子集中，不断探索所有可能的划分组合，以找到能将整数集合 {1, 2,..., k} 划分成 n 个无和子集的最大的 k 值（即 F(n)）以及对应的一种划分方案。

无和子集：

无和子集是指集合中的任意两个元素之和不属于该集合本身。例如，若集合 S = {1, 3, 5}，对于集合 S 中的任意两个元素，如 1 + 3 = 4，4 不属于集合 S；1 + 5 = 6，6 不属于集合 S；3 + 5 = 8，8 不属于集合 S，所以 S 是一个无和子集。在算法中，始终要保证划分出来的每个子集都满足这一性质。

1. 初始化阶段

在 main 函数中，首先从输入文件 input.txt 读取正整数 n，这个 n 代表要将整数集合划分成的无和子集的个数。然后将用于记录最大可划分值的变量 maxDivisibleValue 初始化为 0，并对存储划分中间状态的二维数组 subsetArray 进行初始化，即将每个子集的元素个数（subsetArray[i][0]）初始化为 0，表示每个子集初始时为空集。

2. 深度优先搜索（DFS）阶段 exploreDivisions 函数

该函数是整个算法的核心，通过递归调用实现深度优先搜索，不断尝试不同的划分可能性。

每次调用 exploreDivisions 函数时，传入一个当前要处理的整数 currentNumber。

判断是否找到更优划分：在函数开始处，首先检查当前要处理的整数 currentNumber 是否大于已记录的最大可划分值 maxDivisibleValue。如果是，这意味着找到了一个可能更好的划分情况，因为已经能够将更多的整数成功划分到无和子集中了。此时，就将当前 subsetArray 数组中存储的划分方案复制到 finalResult 数组中，以保存当前最优结果，并更新 maxDivisibleValue 为 currentNumber。

尝试划分到各个子集：接下来，通过一个循环遍历所有的无和子集（个数由 n 即 numSubsets 确定）。对于每个子集：

添加当前整数尝试：先将当前整数 currentNumber 尝试添加到该子集（将其赋值给 subsetArray[i][subsetArray[i][0] + 1]），这相当于在当前划分方案中考虑将 currentNumber 放入这个子集。

判断添加是否可行：然后调用 canAddToSubset 函数来判断这种添加是否可行，即添加后该子集是否依然保持无和集的性质。如果 canAddToSubset 函数返回 1，表示添加是可行的。

继续探索后续划分：若添加可行，就将该子集的元素个数加一（subsetArray[i][0] += 1），然后递归调用 exploreDivisions 函数继续尝试下一个整数（currentNumber + 1），这意味着基于当前的划分情况（已经将 currentNumber 成功添加到某个子集），继续探索后续整数的划分可能性，形成深度优先搜索的递归过程。

回溯操作：在递归调用返回后，需要进行回溯操作，将该子集的元素个数减一（subsetArray[i][0]--），这是因为在递归调用过程中可能已经尝试了多种后续划分情况，现在要回到之前的状态，以便尝试其他可能的添加方式或子集组合。例如，可能之前将 currentNumber 放入了这个子集并继续探索了后续情况，但现在要尝试将 currentNumber 放入其他子集或者改变当前子集的其他元素组合，所以要恢复到添加 currentNumber 之前的子集状态。

移除添加的整数（若不可行）：如果 canAddToSubset 函数返回 0，表示添加当前整数到该子集不可行，那么就将刚才添加的元素（subsetArray[i][subsetArray[i][0] + 1]）重置为 0，以便进行下一轮尝试，即继续考虑将 currentNumber 放入其他子集或者等待下一次循环重新尝试这个子集。

**代码：**

***#include <stdio.h>***

***#include <stdlib.h>***

***#define MAX\_SIZE 100***

***// 用于存储划分过程中各子集信息的二维数组***

***int subsetArray[MAX\_SIZE][MAX\_SIZE];***

***// 用于保存最终有效划分结果的二维数组***

***int finalResult[MAX\_SIZE][MAX\_SIZE];***

***int numSubsets;***

***int maxDivisibleValue;***

***// 判断给定整数是否能添加到指定子集而保持无和集性质***

***int canAddToSubset(int numToAdd, int subsetIndex) {***

***int i, j;***

***for (i = 1; i <= subsetArray[subsetIndex][0]; i++) {***

***for (j = i + 1; j <= subsetArray[subsetIndex][0]; j++) {***

***if (subsetArray[subsetIndex][i] + subsetArray[subsetIndex][j] == numToAdd) {***

***return 0;***

***}***

***}***

***}***

***return 1;***

***}***

***// 深度优先搜索函数，用于探索所有可能的划分组合***

***void exploreDivisions(int currentNumber) {***

***int i;***

***if (currentNumber > maxDivisibleValue) {***

***// 将当前有效的划分复制到最终结果数组***

***for (i = 0; i < numSubsets; i++) {***

***for (int j = 0; j <= subsetArray[i][0]; j++) {***

***finalResult[i][j] = subsetArray[i][j];***

***}***

***}***

***maxDivisibleValue = currentNumber;***

***}***

***for (i = 0; i < numSubsets; i++) {***

***subsetArray[i][subsetArray[i][0] + 1] = currentNumber;***

***if (canAddToSubset(currentNumber, i)) {***

***subsetArray[i][0]++;***

***exploreDivisions(currentNumber + 1);***

***subsetArray[i][0]--;***

***}***

***subsetArray[i][subsetArray[i][0] + 1] = 0;***

***}***

***}***

***// 将计算结果输出到output.txt文件***

***void outputResults() {***

***FILE \*outputFile;***

***int i, j;***

***outputFile = fopen("output.txt", "w");***

***if (outputFile == NULL) {***

***printf("无法打开output.txt文件进行写入操作。\n");***

***exit(1);***

***}***

***// 输出F(n)的值***

***fprintf(outputFile, "%d\n", maxDivisibleValue - 1);***

***// 输出每个无和子集***

***for (i = 0; i < numSubsets; i++) {***

***for (j = 1; j <= finalResult[i][0]; j++) {***

***fprintf(outputFile, "%d ", finalResult[i][j]);***

***}***

***fprintf(outputFile, "\n");***

***}***

***fclose(outputFile);***

***}***

***int main() {***

***FILE \*inputFile;***

***int i;***

***// 从input.txt文件读取子集数量***

***inputFile = fopen("input.txt", "r");***

***if (inputFile == NULL) {***

***printf("无法打开input.txt文件进行读取操作。\n");***

***exit(1);***

***}***

***fscanf(inputFile, "%d", &numSubsets);***

***fclose(inputFile);***

***maxDivisibleValue = 0;***

***// 初始化subsetArray数组***

***for (i = 0; i < numSubsets; i++) {***

***subsetArray[i][0] = 0;***

***}***

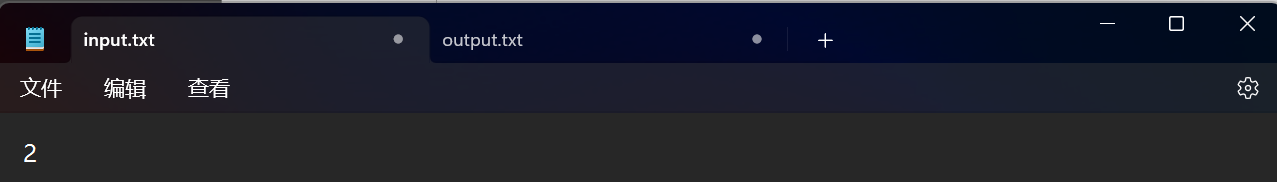
***exploreDivisions(1);***

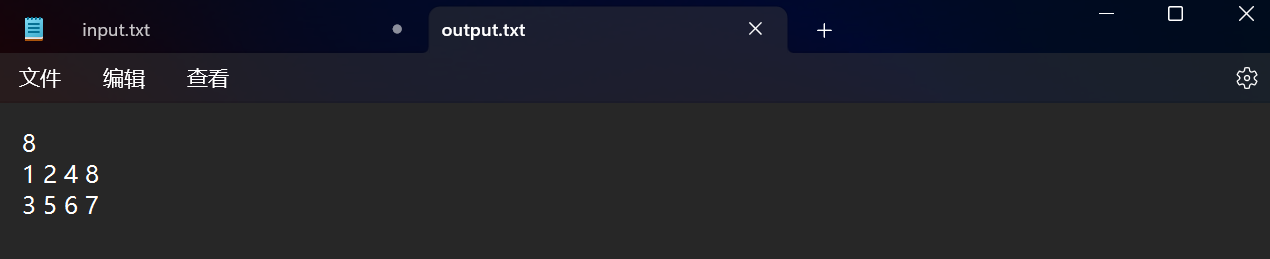
***outputResults();***

***return 0;***

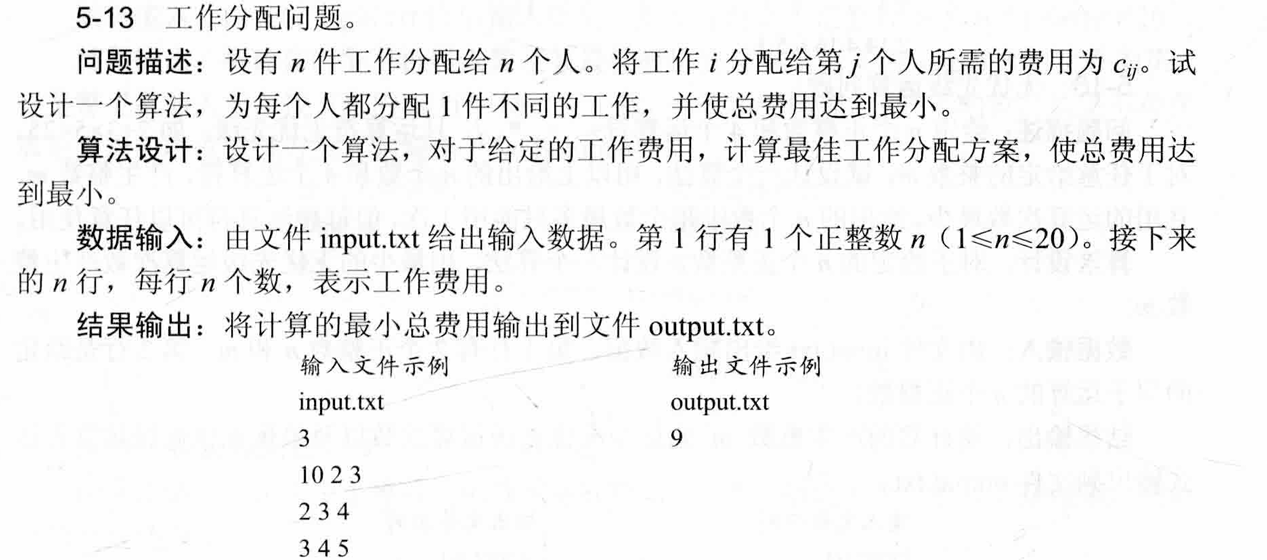
***}***

**运行结果：**





**5-13**



**算法思想：**

以 exploreAssignments 函数实现深度优先搜索，其参数 jobIndex 表示当前正在处理的工作序号。

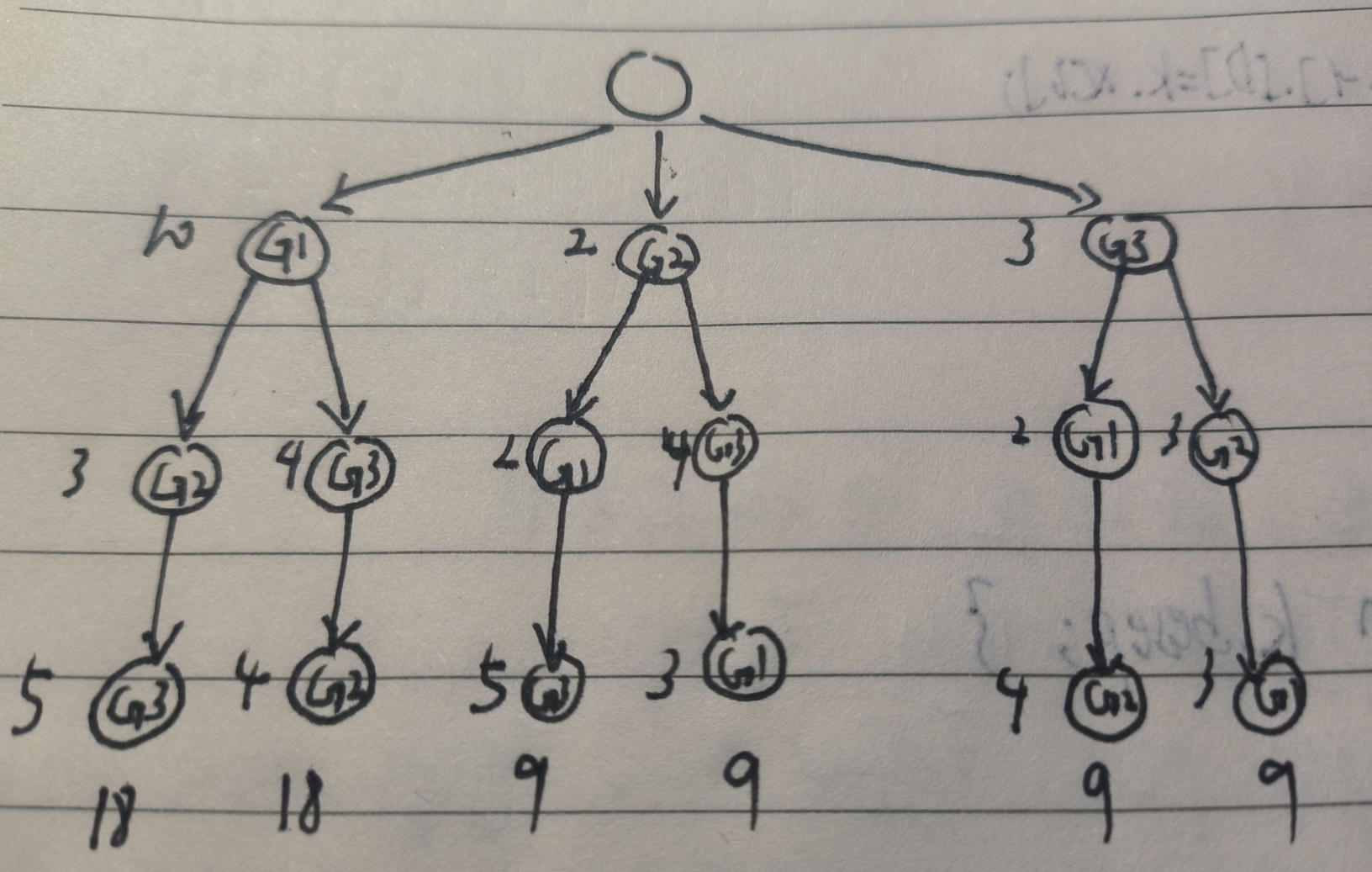
递归终止条件：当 jobIndex 大于等于工作数量 numJobs 时，意味着已经尝试完了所有工作的分配可能性，到达了搜索树的叶子节点。此时，比较当前搜索路径下得到的总费用 currentCost 和已记录的最小总费用 minCost，如果 currentCost 小于 minCost，则更新 minCost 的值。

搜索分支扩展：对于每个工作（由 jobIndex 确定），遍历所有人（由 personIndex 遍历 numJobs 个人），判断如果某个人还没有被分配工作（通过 !assigned[personIndex] 判断），则进行如下操作：

将当前工作分配给该人，即将 assigned[personIndex] 设为 true，并更新当前总费用 currentCost，增加该工作分配给此人产生的费用（通过 currentCost += costMatrix[jobIndex][personIndex]）。

接着判断当前总费用是否小于最小总费用，如果是，则继续向下一个工作（通过调用 exploreAssignments(wa, jobIndex + 1) 对下一个工作进行分配搜索）进行深度优先搜索，探索该分配方案下后续工作分配的可能性。

如果在当前分支搜索完后发现没有得到比 minCost 更小的总费用，就需要进行回溯操作。回溯时，将刚才分配工作的人的状态还原为未分配（即 assigned[personIndex] 设为 false），并从当前总费用中减去刚才分配工作给此人增加的费用（通过 currentCost -= costMatrix[jobIndex][personIndex]），以便尝试其他可能的分配分支。



**代码：**

***#include <iostream>***

***#include <fstream>***

***#include <limits>***

***// 定义一个较大的常量用于初始化最小费用值***

***const int MAX\_INT\_VALUE = std::numeric\_limits<int>::max();***

***// 工作分配结构体，用于存储相关数据***

***struct WorkAssignment {***

***int numJobs; // 工作数量***

***int costMatrix[20][20]; // 存储工作分配给每个人的费用矩阵***

***int minCost; // 记录找到的最小总费用***

***int currentCost; // 记录搜索过程中的当前总费用***

***bool assigned[20]; // 标记每个人是否已被分配工作***

***WorkAssignment() : numJobs(0), minCost(MAX\_INT\_VALUE), currentCost(0) {***

***for (int i = 0; i < 20; i++) {***

***for (int j = 0; j < 20; j++) {***

***costMatrix[i][j] = 0;***

***}***

***assigned[i] = false;***

***}***

***}***

***};***

***// 深度优先搜索函数，用于遍历所有可能的工作分配方案***

***void exploreAssignments(WorkAssignment& wa, int jobIndex) {***

***if (jobIndex >= wa.numJobs) {***

***// 到达叶子节点，比较并更新最小总费用***

***if (wa.minCost > wa.currentCost) {***

***wa.minCost = wa.currentCost;***

***}***

***return;***

***}***

***for (int personIndex = 0; personIndex < wa.numJobs; personIndex++) {***

***if (!wa.assigned[personIndex]) {***

***// 分配工作给当前人***

***wa.assigned[personIndex] = true;***

***wa.currentCost += wa.costMatrix[jobIndex][personIndex];***

***// 如果当前费用小于最小费用，继续搜索下一个工作分配***

***if (wa.currentCost < wa.minCost) {***

***exploreAssignments(wa, jobIndex + 1);***

***}***

***// 回溯，撤销本次分配***

***wa.assigned[personIndex] = false;***

***wa.currentCost -= wa.costMatrix[jobIndex][personIndex];***

***}***

***}***

***}***

***int main() {***

***WorkAssignment workAssignment;***

***// 从文件读取输入数据***

***std::ifstream inputFile("input.txt");***

***if (!inputFile) {***

***std::cerr << "无法打开输入文件input.txt" << std::endl;***

***return 1;***

***}***

***// 读取工作数量***

***inputFile >> workAssignment.numJobs;***

***// 读取工作费用矩阵***

***for (int i = 0; i < workAssignment.numJobs; i++) {***

***for (int j = 0; j < workAssignment.numJobs; j++) {***

***inputFile >> workAssignment.costMatrix[i][j];***

***}***

***}***

***inputFile.close();***

***// 开始深度优先搜索***

***exploreAssignments(workAssignment, 0);***

***// 将结果输出到文件***

***std::ofstream outputFile("output.txt");***

***if (!outputFile) {***

***std::cerr << "无法打开输出文件output.txt" << std::endl;***

***return 1;***

***}***

***outputFile << workAssignment.minCost << std::endl;***

***outputFile.close();***

***return 0;***

***}***

**运行结果：**

