**暨南大学本科实验报告专用纸**

课程名称 算法分析与设计实验 成绩评定

实验项目名称 指导教师 李展 实验地点 N116

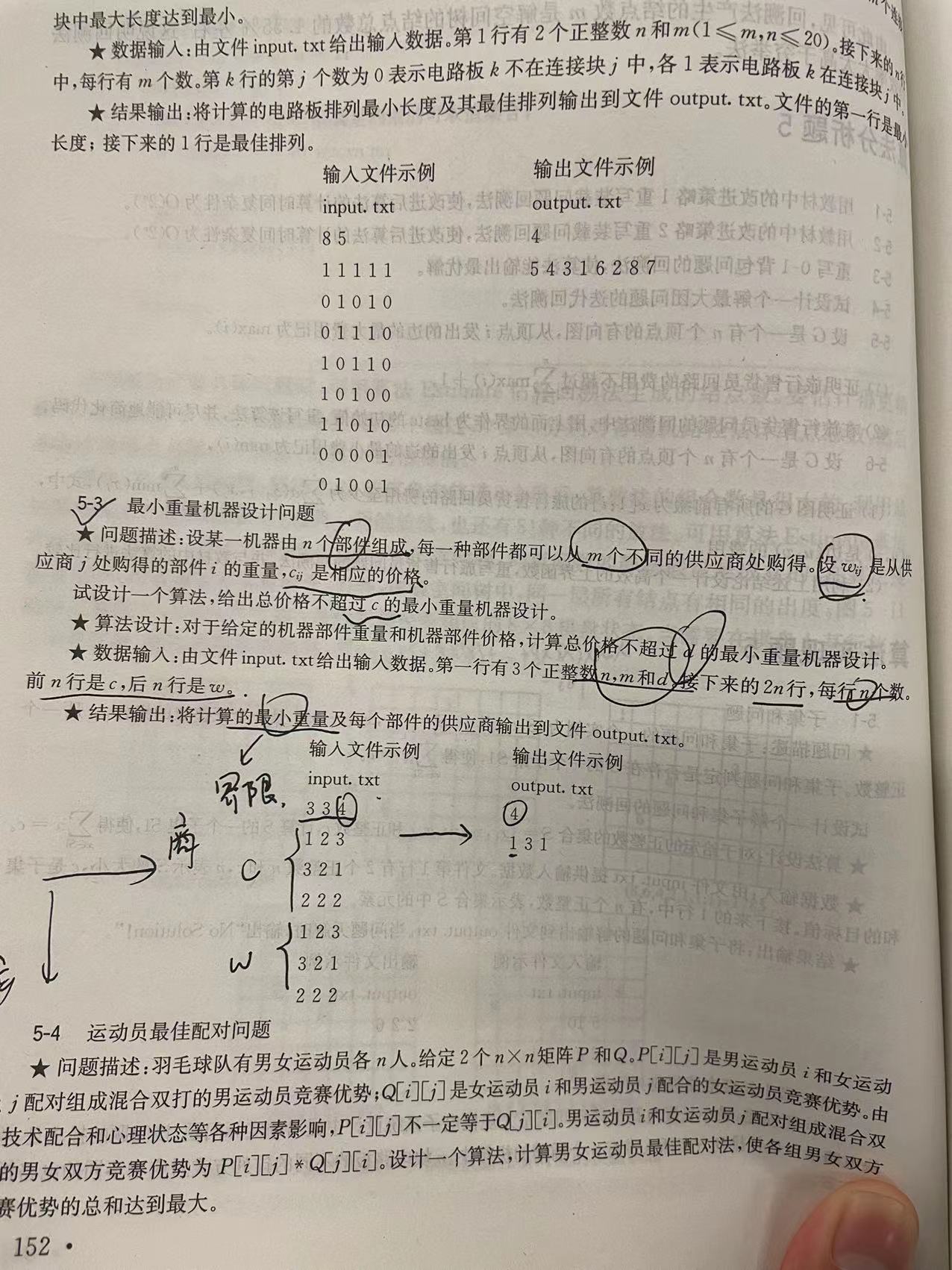
实验项目编号 4-7 实验项目类型 综合型

学生姓名 张瑞鹏 学号 2020101124

学院 信息科学技术学院 系 计算机 专业 计算机科学与技术

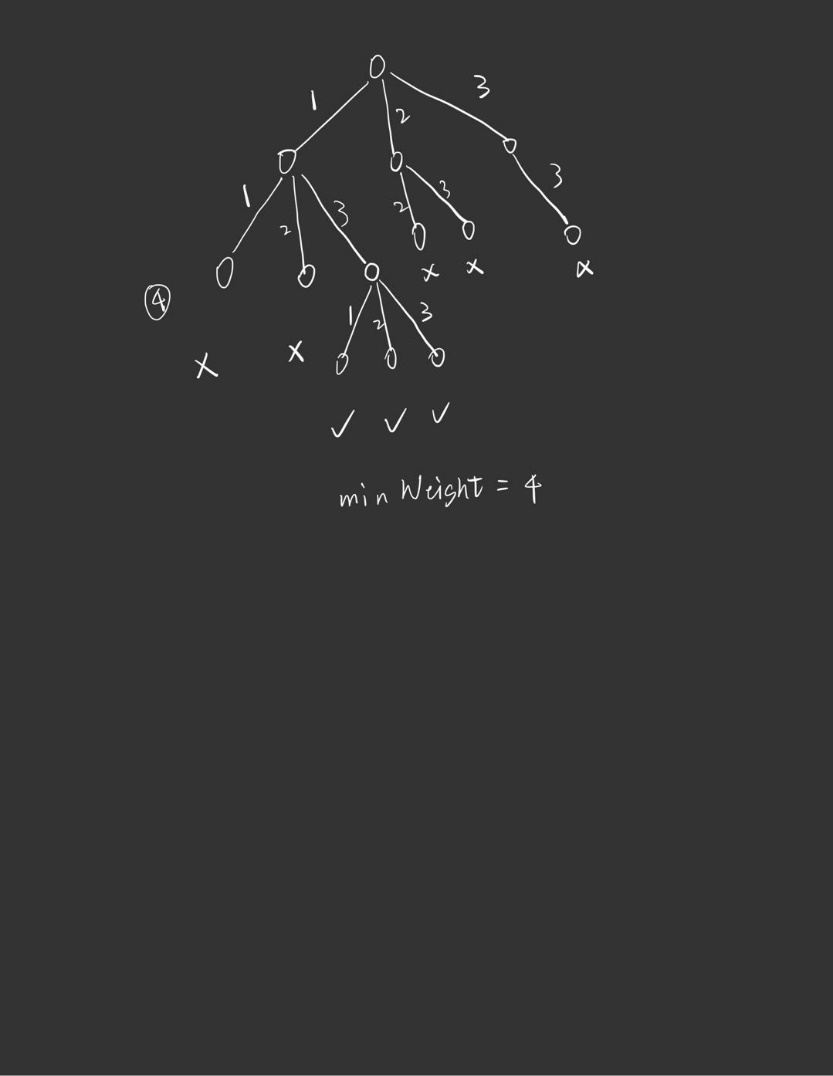
实验时间 2022 年 月 日 午～ 月 日 下 午

**（一）问题描述**



**（二）算法思路**（用文字简单说明）

**采用回溯剪枝的算法，递归方法比较简单，非递归算法写起来复杂很多，但是思路都是差不多。都是基于深度优先搜索的枚举，通过限制函数还有界限函数来剪枝，使得枚举有条不紊。**



(c + cost[t - 1][i - 1]) <= limit) && ((w + weight[t - 1][i - 1]) < bestw)

这就是分别两个函数，剪枝结果如图所示。

**（三）算法实施步骤和流程**（伪代码/流程图等方式描述）

**递归：**

void traceback(int\* x, int\* bestlist, int t, int& c, int& w, int& bestw) {

if (t > n) {

bestw = w;

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

bestlist[i] = x[i];

}

}

for (int i = 1; i <= m; i++) {

if (((c + cost[t - 1][i - 1]) <= limit) && ((w + weight[t - 1][i - 1]) < bestw)) {

c += cost[t - 1][i - 1];

w += weight[t - 1][i - 1];

x[t - 1] = i;

traceback(x, bestlist, t + 1, c, w, bestw);

c -= cost[t - 1][i - 1];

w -= +weight[t - 1][i - 1];

}

}

return;

}

**非递归：**

void traceback(int\* x, int\* bestlist, int t, int c, int w, int &bestw) {

while (t>0) {//x内部索引-1，

x[t] += 1;//第t个零件，共有n个 第x[t]个商家

while (((c + cost[t][x[t]]) > limit) || ((w + weight[t][x[t]]) >= bestw)) {

if (x[t] <= m) {

x[t]++;

}

else break;

}

if (x[t] <= m) {

c += cost[t][x[t]];

w += weight[t][x[t]];

if (t == n) {

bestw = w;

for (int i = 1; i <= n; i++) {

bestlist[i] = x[i];

}//之后同一层里面继续走

c -= cost[t][x[t]];

w -= weight[t][x[t]];

}

else {

t++;

x[t] = 0;

}

}

else {

t--;

c -= cost[t][x[t]];

w -= weight[t][x[t]];

}//所有可能都在这里弹出

}

return;

}

**（四）源代码**（通过了编译运行的正确程序）

#include<iostream>

using namespace std;

int weight[4][4] = {

{0,0,0,0},

{0,7,2,2},

{0,3,2,3},

{0,3,5,4}

};

int cost[4][4] = {

{0,0,0,0},

{0,2,4,3},

{0,3,7,1},

{0,8,2,1}

};

int n = 3;

int m = 3;

int limit = 12;

void traceback(int\* x, int\* bestlist, int t, int c, int w, int &bestw) {

while (t>0) {//x内部索引-1，

x[t] += 1;//第t个零件，共有n个 第x[t]个商家

while (((c + cost[t][x[t]]) > limit) || ((w + weight[t][x[t]]) >= bestw)) {

if (x[t] <= m) {

x[t]++;

}

else break;

}

if (x[t] <= m) {

c += cost[t][x[t]];

w += weight[t][x[t]];

if (t == n) {

bestw = w;

for (int i = 1; i <= n; i++) {

bestlist[i] = x[i];

}//之后同一层里面继续走

c -= cost[t][x[t]];

w -= weight[t][x[t]];

}

else {

t++;

x[t] = 0;

}

}

else {

t--;

c -= cost[t][x[t]];

w -= weight[t][x[t]];

}//所有可能都在这里弹出

}

return;

}

int main(void) {

int c = 0, w = 0, bestw = 1000;

int x[4] = { 0 };

int bestlist[4] = { 0 };

cout << "test example:" << endl;

printf("%-3d%-3d%-3d\n\n", n, m, limit);

for (int i = 1; i <= n; i++) {

for (int j = 1; j <=m; j++) {

printf("%-3d", cost[i][j]);

}

cout << endl;

}

printf("\n");

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

printf("%-3d", weight[i][j]);

}

cout << endl;

}

traceback(x, bestlist, 1, c, w, bestw);

cout << "the result is:" << endl;

cout << bestw << endl;

for (int i = 1; i <= n;i++) {

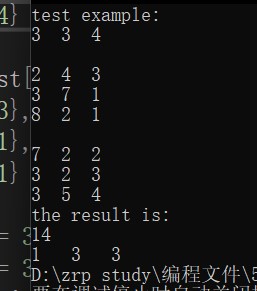
printf("%-3d", bestlist[i]);

}

return 0;

}

**（五）测试结果**（至少有两个以上算例及程序运行结果，截图贴进实验报告）



**（六）实验总结**（至少三句话，可以写复杂度分析、遇到问题、可能的改进措施、心得体会等）

1.回溯算法的技术在于确定剪枝函数的设计。回溯法大致可以分成两种，一种是排列树，一种是子集树，而且子集树又包括了排列树，总体而言，子集树更加的一般。排列树相当于子集树又增加了一项限制，主要体现在最后一步更为复杂。

2.尝试了非递归写法，非递归的难度更加大，因为要更加熟悉模拟计算机压栈的过程，定义的t其实相当于一个栈，经常会出现++，--的操作，意思代表的就是栈的移动，整个设计思路都不太一样，难点在于需要分情况来恢复。

3.时间复杂度方面是O（m\*\*n）是一个相当大的时间复杂度。这是回溯的特点，复杂度感觉都会非常大，但是一般使用回溯的算法感觉都是比较复杂的算法，不具有最优子结构，甚至很多已经被证明是非线性时间能够解决的问题都可以解决，所以相对而言回溯其实实用性还是挺广的。

4.这道题目显然是一个子集树，显然不能用排列数解决，因此审题需要注意要求，不要随便使用排列树。

**暨南大学本科实验报告专用纸(附页)**