**北京邮电大学软件学院**

**2020-2021学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称： 实验三：分支限界法**

**项目完成人：**

**姓名：\_\_王衔飞\_\_学号：\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2020年 11 月 15 日**

1. **实验目的**

1、 深刻理解并掌握 分支限界 法的设计思想 ，比较与回溯法的不同之处

2、 提高应用分支限界法设计算法的技能；

3、 理解这样一个观点：好的限界函数不仅计算简单，还要保证最优解在搜索空间中，更重要的是能在搜索的早期对超出目标函数界的结点进行丢弃，减少搜索空间，从而尽快找到问题的最优解。

1. **实验内容**

运用分支限界编程解决基本题，学有余力者继续解决补充题

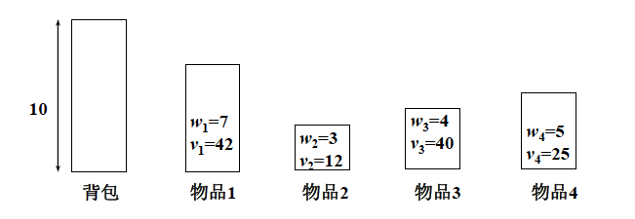
1. **实验环境**

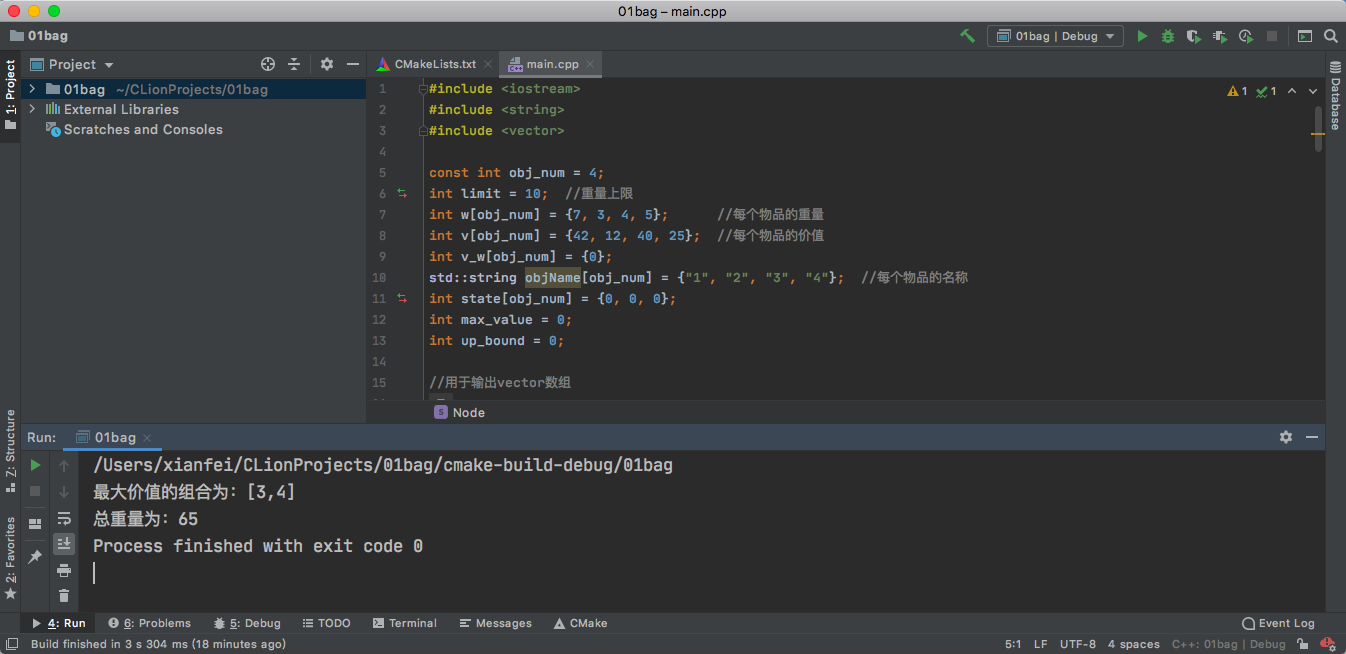
CLion 2020.2 + macOS10.13 + gcc9

语言：C++ 17

1. **实验结果**

0/1背包问题：完成实验任务，测试数据如图，程序得出最佳选择方案是选择物品3和物品4，总价值为65。所用算法的时间复杂度为O(2n)





旅行商售货员问题的分支限界算法：完成实验任务，测试数据如图所示，程序运行得出正确路径。所用算法的时间复杂度O(n²)其中n为边的个数

图片包含 手表, 钟表, 不同, 游戏机

描述已自动生成

图形用户界面

描述已自动生成

1. **附录**

（附上实验文档，尽量有：问题分析、设计方案、算法原理、流程、设计图、程序、仿真结果、运行结果、心得等，具体内容根据实验要求来定。源代码请附在这里。源代码排版请特别注意，用5号字体，行间距为单倍行距。注意节省空间，不要浪费纸张。）

**背包问题：**

算法设计：考虑最好情况，即背包中装入的全部是价值最大的物品且可以将背包装满，由此得到评估函数的上界。根据当前背包容量和未被选择的物品中单位质量价值最大的物品来填充。

算法的具体过程为：先考虑放入该物品，计算放入时的ub1，然后再考虑不放入该物品，并计算其ub2，之后比较两个界限，在当前背包未满的情况下，选择ub 值较大的一个节点进行搜索，并将其ub 值赋值给此节点。

代码：

#include <iostream>

#include <string>

#include <vector>

const int obj\_num = 4;

int limit = 10; //重量上限

int w[obj\_num] = {7, 3, 4, 5}; //每个物品的重量

int v[obj\_num] = {42, 12, 40, 25}; //每个物品的价值

int v\_w[obj\_num] = {0};

std::string objName[obj\_num] = {"1", "2", "3", "4"}; //每个物品的名称

int state[obj\_num] = {0, 0, 0};

int max\_value = 0;

int up\_bound = 0;

//用于输出vector数组

template<typename T>

std::ostream &operator<<(std::ostream &os, std::vector<T> vec) {

os << "[";

if (!vec.empty()) {

std::copy(vec.begin(), vec.end() - 1, std::ostream\_iterator<T>(os, ","));

os << vec.back();

}

os << "]";

return os;

}

struct Node {

int w,v;

Node(int w, int v) : w(w), v(v) {}

};

int upbound(Node n, int k) {//界限函数

int ub = 0;

for (int i = k; i < obj\_num; i++) {

if ((limit - n.w) \* v\_w[i] > ub)

ub = (limit - n.w) \* v\_w[i] + n.v;

}

return ub;

}

void search(Node node, int i) {

if (i == obj\_num) return; //递归结束条件

int ub1, ub2, w\_ = w[i], v\_ = v[i];

//加入物体i

Node node1(node.w + w\_, node.v + v\_);

ub1 = upbound(node1, i + 1);

//不加入物体i

Node node2(node.w, node.v);

ub2 = upbound(node2, i + 1);

//当两种情况都未将背包装满时

if (node1.w <= limit && node2.w <= limit) {

if (ub1 >= ub2 && ub1 < up\_bound) {

state[i] = 1;

max\_value = node1.v;

search(node1, i + 1);

} else if (ub1 < ub2 && ub2 < up\_bound) {

max\_value = node2.v;

search(node2, i + 1);

}

}

//如果加入此物品将背包装满

else if (node1.w > limit && node2.w <= limit) {

max\_value = node2.v;

search(node2, i + 1);

}

}

int main() {

for (int i = 0; i < obj\_num; i++) {

v\_w[i] = v[i] / w[i]; //计算v/w的值

up\_bound += v[i]; //上界为所有价值之和

}

search(Node(0, 0), 0); //计算最小边界，最大边界

std::cout << "最大价值的组合为：";

std::vector<std::string> maxObjs;

for (int i = 0; i < obj\_num; i++)

if (state[i] == 1)

maxObjs.push\_back(objName[i]);

std::cout << maxObjs;

std::cout << "\n总重量为：" << max\_value;

return 0;

}

程序输出：

最大价值的组合为：[3,4]

总重量为：65

Process finished with exit code 0

**旅行商售货员问题的分支限界算法：**

算法设计：找出与每个城市连接的两条最小的路径来代表进入和离开的路程，在每次选择下一个地点后，将其路程值替换掉原来的值中的较大的一个，以此来设计界限。搜索函数从没有的点中选择下一个要去的城市，在list数组中存放已经走过的点，遍历每个可以选择的点，并计算其下界值，如果其下界值在界限之内，则要对此节点继续进行搜索。递归结束的条件为所有地点都选择过。

代码：

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

const int n=5;

int cost\_matrix[n][n]= {

{INT\_MAX,3,1,5,8},

{3,INT\_MAX,6,7,9},

{1,6,INT\_MAX,4,2},

{5,7,4,INT\_MAX,3},

{8,9,2,3,INT\_MAX}};

char symbol[]= {'1','2','3','4','5'};

int least\_cost=INT\_MAX;

int up\_bound=1+2+3+7+3;

int low\_bound=((1+3)+(3+6)+(1+2)+(3+4)+(2+3))/2;

std::vector<int> list;

int candidate\_option[n][2]= {{1,3},{3,6},{1,2},{3,4},{2,3}};

int calc\_lb(int start, int end, int candidate\_option\_[n][2]) { //计算下界

int lowbound = 2\*cost\_matrix[start][end];

if(cost\_matrix[start][end] > candidate\_option\_[start][0]) lowbound += candidate\_option\_[start][0];

if(cost\_matrix[start][end] == candidate\_option\_[start][0]) lowbound += candidate\_option\_[start][1];

if(cost\_matrix[start][end] > candidate\_option\_[end][0]) lowbound += candidate\_option\_[end][0];

if(cost\_matrix[start][end] == candidate\_option\_[end][0]) lowbound += candidate\_option\_[end][1];

//加上剩余没有选过的行的最小的两个元素

for(int i=0;i<n;i++) if(i!=start&&i!=end) lowbound += candidate\_option\_[i][0] + candidate\_option\_[i][1];

return lowbound/2;

}

void search(int root) {

int lb=INT\_MAX;

list.push\_back(root);

for(int i=0;i<n;i++) { //遍历没有走过的地点

if(std::find(list.begin(), list.end(), i) == list.end()) { // 取上界

lb = ceil(calc\_lb(root, i, candidate\_option));

if((lb<up\_bound&&lb>=low\_bound)||(list.size()==4&&lb==up\_bound)) {

if(list.size()==4) { // 当走过5个地点的时候结束递归

list.push\_back(i);

least\_cost=lb;

throw "exit";

}

int temp1=INT\_MAX,temp2=INT\_MAX;

if(cost\_matrix[root][i]>candidate\_option[root][0]) {

temp1=candidate\_option[root][1];

candidate\_option[root][1]=cost\_matrix[root][i];

}

if(cost\_matrix[root][i]>candidate\_option[i][0]) {

temp2=candidate\_option[i][1];

candidate\_option[i][1]=cost\_matrix[root][i];

}

search(i);

list.erase(std::find(list.begin(), list.end(), i)); // 回溯,删除i

if(temp1!=INT\_MAX) candidate\_option[root][1]=temp1;

if(temp2!=INT\_MAX) candidate\_option[i][1]=temp2;

}

}

}

}

int main() {

try {

search(0);

}catch (...) {

std::cout << "方案为："<<std::endl;

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

std::cout << symbol[list.at(i)]<<"->";

}

std::cout << symbol[list.at(0)] <<std::endl;

std::cout << "代价为：" << least\_cost;

}

}

程序输出：

方案为：

1->3->5->4->2->1

代价为：16

Process finished with exit code 0

心得：通过本次实验进一步实践和了解了分支界限法。