**北京邮电大学软件学院**

**2016-2017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： 实验项目三——分支限界法**

**项目完成人：**

**姓名：沈嘉樑 学号：2014212065**

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

**日 期： 2016年 11 月 18 日**

1. **实验目的**
2. 深刻理解并掌握分支限界法的设计思想；
3. 提高应用分支限界法设计算法的技能；
4. **实验内容**

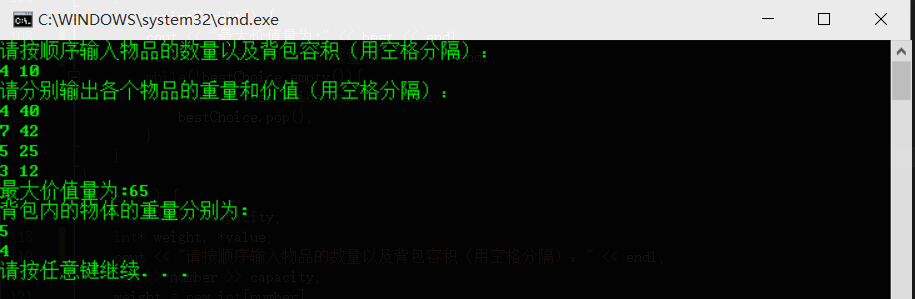
用分支限界法解决0/1背包问题、任务分配问题和八数码问题

1. **实验环境**

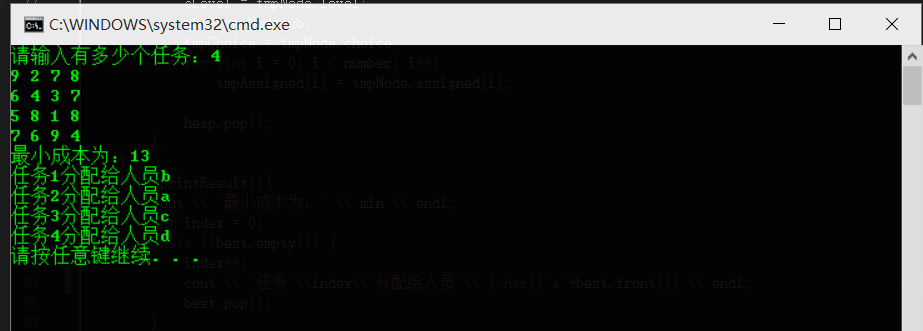
windows 10、vs2015

1. **实验结果**

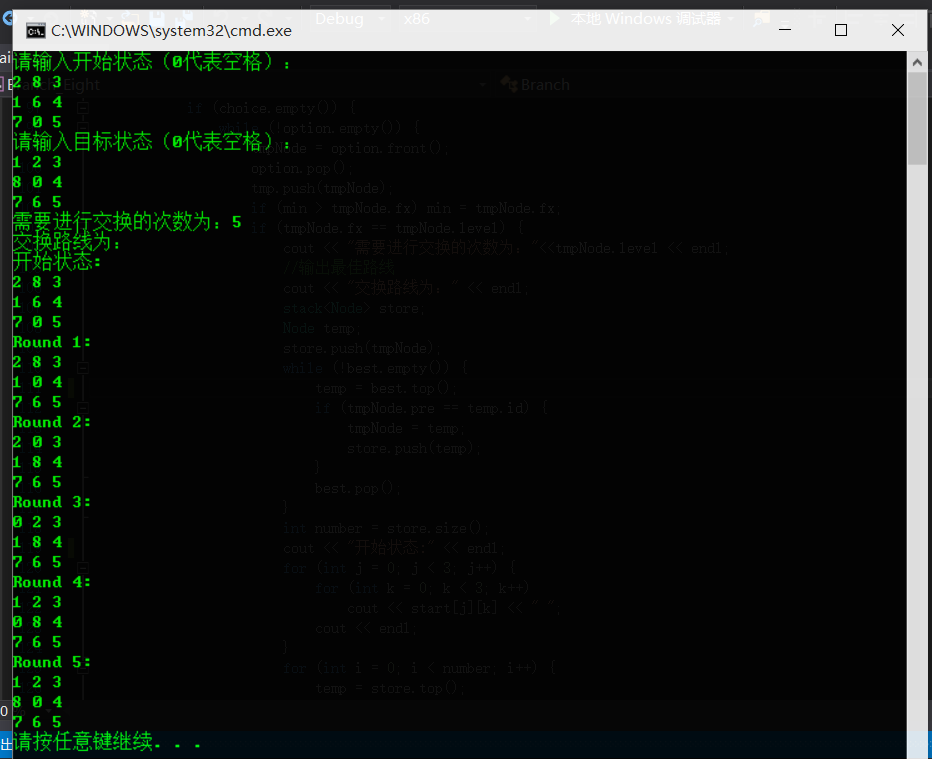
* 0/1背包问题实验结果



* 任务分配问题



* 八数码问题



1. **附录**

* 0/1背包问题主要代码

struct Node {

int level;

double weight;

double value;

double ub;

stack<int> choice;

};

double maxBound(int currentLevel) {

double spare = capacity - currentWeight;

double bound = currentValue;

while (currentLevel < (number -1) && weight[currentLevel] <= spare ) {

spare -= weight[currentLevel];

bound += value[currentLevel];

currentLevel++;

}

if (currentLevel < number - 1) {

bound += spare \* (value[currentLevel] / weight[currentLevel]);

}

return bound;

}

void addNode(stack<int> currentChoice,int currentWeight\_, int currentValue\_, int ub\_, int level\_) {

Node newNode;

newNode.weight = currentWeight\_;

newNode.value = currentValue\_;

newNode.level = level\_;

newNode.ub = ub\_;

newNode.choice = currentChoice;

if (level\_ <= number - 1)

heap.push(newNode);

}

void bestResult() {

int index = 0;

double ub = maxBound(index);

int tmpWeight;

Node tmpNode;

stack<int> tmpChoice;

while (true) {

tmpWeight = currentWeight + weight[index];

if (tmpWeight < capacity) {

tmpChoice.push(weight[index]);

addNode(tmpChoice, tmpWeight, currentValue + value[index], ub, index+1);

if (best < (currentValue + value[index])) {

best = currentValue + value[index];

bestChoice = tmpChoice;

}

}

ub = maxBound(index + 1);

if (ub > best) {

addNode(tmpChoice, currentWeight, currentValue, ub, index + 1);

}

if (heap.empty())

return;

tmpNode = heap.top();

currentValue = tmpNode.value;

currentWeight = tmpNode.weight;

index = tmpNode.level;

ub = tmpNode.ub;

tmpChoice = tmpNode.choice;

heap.pop();

}

}

* 任务分配问题主要代码

struct Node {

bool assigned[MAX];

int ub;

int level;

queue<int> choice;

};

void addNode(queue<int> currentChoice, int ub\_, bool assigned\_[MAX], int level) {

Node n;

n.choice = currentChoice;

n.level = level;

n.ub = ub\_;

for (int i = 0; i < number; i++)

n.assigned[i] = assigned\_[i];

if (level <= number-1)

heap.push(n);

}

void bestResult() {

int cLevel = -1;

bool tmpAssigned[MAX];

int ub = 0;

int temp;

Node tmpNode;

queue<int> tmpChoice;

queue<int> tmp;

for (int i = 0; i < number; i++)

tmpAssigned[i] = false;

while (true) {

cLevel++;

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

if ((!tmpAssigned[i]) && (ub + cost[cLevel][i] <= min)) {

tmpAssigned[i] = true;

tmpChoice.push(i);

addNode(tmpChoice, ub + cost[cLevel][i], tmpAssigned, cLevel);

if (cLevel == number - 1) {

min = ub + cost[cLevel][i];

best = tmpChoice;

}

tmpAssigned[i] = false;

while (!tmpChoice.empty()) {

temp = tmpChoice.front();

tmp.push(temp);

tmpChoice.pop();

}

int n = tmp.size();

for (int i = 0; i < n - 1; i++) {

tmpChoice.push(tmp.front());

tmp.pop();

}

}

}

if (heap.empty())

return;

tmpNode = heap.front();

cLevel = tmpNode.level;

ub = tmpNode.ub;

tmpChoice = tmpNode.choice;

for (int i = 0; i < number; i++)

tmpAssigned[i] = tmpNode.assigned[i];

heap.pop();

}

}

* 八数码问题主要代码

struct Node {

int id;//记录当前节点的序号

int pre;//记录上一个节点的序号

int lastMoved;//记录上一次移动的点

int state[3][3];//记录状态

int fx;

int level;//记录层次

};

void addNode(int level\_, int lastMoved\_, int id\_, int pre\_) {

Node n;

n.pre = pre\_;

n.id = id\_;

n.level = level\_;

n.lastMoved = lastMoved\_;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

n.state[i][j] = tmpNode.state[i][j];

n.fx = 0;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if ((n.state[i][j] != finish[i][j]) && (finish[i][j] != 0)) {

n.fx++;

}

n.fx = n.fx + level\_;

option.push(n);

}

int findFree(int state[3][3]) {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3;j++)

if (state[i][j] == 0) {

return i\*10+j;

}

}

void bestResult() {

int cLevel = 0;

int position[2];

int p;

queue<Node> tmp;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++) {

tmpNode.state[i][j] = start[i][j];

}

tmpNode.id = index;

tmpNode.pre = -1;

tmpNode.lastMoved = -1;

tmpNode.fx = INT\_MAX;

tmpNode.level = 0;

while (true) {

p = findFree(tmpNode.state);

position[0] = p / 10;

position[1] = p % 10;

for (int i = position[0] - 1; i <= position[0] + 1; i++) {

if ((i >= 0) && (i <= 2) && (i!=position[0]) && (tmpNode.lastMoved != (i\*10+position[1]))) {

index++;

tmpNode.state[position[0]][position[1]] = tmpNode.state[i][position[1]];

tmpNode.state[i][position[1]] = 0;

addNode( cLevel + 1, p, index, tmpNode.id);

tmpNode.state[i][position[1]] = tmpNode.state[position[0]][position[1]];

tmpNode.state[position[0]][position[1]] = 0;

}

}

for (int i = position[1] - 1; i <= position[1] + 1; i++) {

if ((i >= 0) && (i <= 2) && (i != position[1]) && (tmpNode.lastMoved != (i + position[0]\*10))) {

index++;

tmpNode.state[position[0]][position[1]] = tmpNode.state[position[0]][i];

tmpNode.state[position[0]][i] = 0;

addNode( cLevel + 1,p, index, tmpNode.id);

tmpNode.state[position[0]][i] = tmpNode.state[position[0]][position[1]];

tmpNode.state[position[0]][position[1]] = 0;

}

}

int min = INT\_MAX;

if (choice.empty()) {

while (!option.empty()) {

tmpNode = option.front();

option.pop();

tmp.push(tmpNode);

if (min > tmpNode.fx) min = tmpNode.fx;

if (tmpNode.fx == tmpNode.level) {

cout << "需要进行交换的次数为："<<tmpNode.level << endl;

//输出最佳路线

cout << "交换路线为：" << endl;

stack<Node> store;

Node temp;

store.push(tmpNode);

while (!best.empty()) {

temp = best.top();

if (tmpNode.pre == temp.id) {

tmpNode = temp;

store.push(temp);

}

best.pop();

}

int number = store.size();

cout << "开始状态:" << endl;

for (int j = 0; j < 3; j++) {

for (int k = 0; k < 3; k++)

cout << start[j][k] << " ";

cout << endl;

}

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

temp = store.top();

store.pop();

cout << "Round " << i + 1 << ":" << endl;

for (int j = 0; j < 3; j++) {

for (int k = 0; k < 3; k++)

cout << temp.state[j][k] << " ";

cout << endl;

}

}

return;

}

}

while (!tmp.empty()) {

tmpNode = tmp.front();

tmp.pop();

if (min == tmpNode.fx) {

choice.push(tmpNode);

best.push(tmpNode);

}

}

}

tmpNode = choice.front();

choice.pop();

cLevel = tmpNode.level;

}

}

};