实验题目：A\*解决罗马尼亚路径问题

实验代码：

#include<iostream>

#include<vector>

#include<memory.h>

#include<stack>

#include<algorithm>

#define A 0

#define B 1

#define C 2

#define D 3

#define E 4

#define F 5

#define G 6

#define H 7

#define I 8

#define L 9

#define M 10

#define N 11

#define O 12

#define P 13

#define R 14

#define S 15

#define T 16

#define U 17

#define V 18

#define Z 19

using namespace std;

//Astar算法种的评估函数是f=g+h;

//其中g是从始点出发到当前点的距离，h是当前点到目标点的距离

//本题中h已经给定了，所以就不需要进行计算了

//h[20]就是20个节点对应的h的数值

int h[20] =

{ 366,0,160,242,161,

178,77,151,226,244,

241,234,380,98,193,

253,329,80,199,374 };

//以下是Node节点，对应Astar算法中的节点，g h f对应Astar中g h f，

//name对应此节点的编号，也就是以上预定义的数值

//这里重载了<号，便于根据f排序

struct node

{

int g;

int h;

int f;

int name;

node(int name, int g, int h)

{

this->name = name;

this->g = g;

this->h = h;

this->f = g + h;

};

bool operator <(const node &a)const

{

return f < a.f;

}

};

//以下是一个图结构，用来存储图的相关信息

//默认用一个20\*20的数组来存储图的信息

//构造函数将所有数值都初始化为-1

//类提供getEdge和addEdge方法

class Graph

{

public:

Graph()

{

memset(graph, -1, sizeof(graph));

}

int getEdge(int from, int to)

{

return graph[from][to];

}

void addEdge(int from, int to, int cost)

{

if (from >= 20 || from < 0 || to >= 20 || to < 0)

return;

graph[from][to] = cost;

}

void init()

{

addEdge(O, Z, 71);

addEdge(Z, O, 71);

addEdge(O, S, 151);

addEdge(S, O, 151);

addEdge(Z, A, 75);

addEdge(A, Z, 75);

addEdge(A, S, 140);

addEdge(S, A, 140);

addEdge(A, T, 118);

addEdge(T, A, 118);

addEdge(T, L, 111);

addEdge(L, T, 111);

addEdge(L, M, 70);

addEdge(M, L, 70);

addEdge(M, D, 75);

addEdge(D, M, 75);

addEdge(D, C, 120);

addEdge(C, D, 120);

addEdge(C, R, 146);

addEdge(R, C, 146);

addEdge(S, R, 80);

addEdge(R, S, 80);

addEdge(S, F, 99);

addEdge(F, S, 99);

addEdge(F, B, 211);

addEdge(B, F, 211);

addEdge(P, C, 138);

addEdge(C, P, 138);

addEdge(R, P, 97);

addEdge(P, R, 97);

addEdge(P, B, 101);

addEdge(B, P, 101);

addEdge(B, G, 90);

addEdge(G, B, 90);

addEdge(B, U, 85);

addEdge(U, B, 85);

addEdge(U, H, 98);

addEdge(H, U, 98);

addEdge(H, E, 86);

addEdge(E, H, 86);

addEdge(U, V, 142);

addEdge(V, U, 142);

addEdge(I, V, 92);

addEdge(V, I, 92);

addEdge(I, N, 87);

addEdge(N, I, 87);

}

private:

int graph[20][20];

};

bool list[20];//记录节点是否在openList中

vector<node> openList;//对应Astar算法中的oepnList

bool closeList[20];//对应Astar算法中的closeList

stack<int> road;//用来保存最终结果并输出

int parent[20];//用来记录每个节点对应的父节点

//解释一下为什么没用priority\_queue，虽然优先队列可以自动排序，但是问题在于它不能遍历

//因为Astar算发会涉及到修改openList表的数值，而队列是不能遍历的所以我采用vector来存放节点，在增加或删除节点后，调用sort进行排序

void A\_star(int goal,node &src,Graph &graph)

{

//首先将始点放进openList

openList.push\_back(src);

sort(openList.begin(), openList.end());

//一下开始遍历openList

while (!openList.empty())

{

node current = openList[0];//取出第一个节点，第一个节点的f值是最小的

if (current.name == goal)//如果是目标节点，那么结束循环

break;

openList.erase(openList.begin());//将节点从openList表中拿出

list[current.name] = false;

sort(openList.begin(), openList.end());//对openList重新排序

closeList[current.name] = true;//将节点加入closeList中

//以下开始遍历和当前节点相邻的点

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

{

if (graph.getEdge(current.name, i) != -1)//如果值不等于-1，则代表相邻

{

if (closeList[i])//如果已经在closeList表中，则查下下一个点

continue;

else if (list[i])//如果在openList表中

{

//以下进行的操作是，从openList中找到这个节点

int num = 0;

for (vector<node>::iterator it = openList.begin(); it != openList.end(); it++)

{

if (it->name == i)

break;

num++;

}

//如果经过current到达i点更好的话，也就是g值更小，则更新这个节点

if (current.g + graph.getEdge(current.name, i) < openList[num].g)

{

openList[num].g = current.g + graph.getEdge(current.name, i);

openList[num].f = openList[num].g + openList[num].h;

parent[i] = current.name;

list[i] = true;

}

}

else

{

//如果这个点既不在openList也不在closeList则创建一个新点，将这个点加入openList

node newNode(i, graph.getEdge(current.name, i), h[i]);

parent[i] = current.name;

openList.push\_back(newNode);

sort(openList.begin(), openList.end());

}

}

}

}

}

//这个函数的目的是输出结果，将所有的节点的编号放入栈中，这样就能倒转了，

//然后计算从始点到目标点的cost

void print\_result(Graph &graph)

{

int p = openList[0].name;

int lastNodeNum;

road.push(p);

while (parent[p] != -1)

{

road.push(parent[p]);

p = parent[p];

}

lastNodeNum = road.top();

int cost = 0;

cout << "solution: ";

while (!road.empty())

{

cout << road.top() << "-> ";

if (road.top() != lastNodeNum)

{

cost += graph.getEdge(lastNodeNum, road.top());

lastNodeNum = road.top();

}

road.pop();

}

cout << "end" << endl;

cout << "cost:" << cost;

}

int main()

{

Graph graph;

graph.init();

int goal = B;

node src(A, 0, h[A]);

list[A] = true;

memset(parent, -1, sizeof(parent));

memset(list, false, sizeof(list));

A\_star(goal, src, graph);

print\_result(graph);

return 0;

}

实验结果：

