模拟退火算法解决旅行商问题：

1. 产生一条新的遍历路径P(i+1)，计算路径P(i+1)的长度L( P(i+1) )

2. 若L(P(i+1)) < L(P(i))，则接受P(i+1)为新的路径，否则以模拟退火的那个概率接受P(i+1) ，然后降温

3. 重复步骤1，2直到满足退出条件

产生新的遍历路径的方法有：

1. 随机选择2个节点，交换路径中的这2个节点的顺序。

2. 随机选择2个节点，将路径中这2个节点间的节点顺序逆转。

3. 随机选择3个节点m，n，k，然后将节点m与n间的节点移位到节点k后面。

#include <iostream>

#include <string.h>

#include <fstream>

#include <iterator>

#include <algorithm>

#include <limits.h>

#include <math.h>

#include <stdlib.h>

#include <time.h>

using namespace std;

const int nCities = 99; //城市数量

const double SPEED = 0.98;//退火速度

const int INITIAL\_TEMP = 1000;//初始温度

const int L = 100 \* nCities;//Markov 链的长度

struct node

{

int num;

int x;

int y;

}nodes[nCities];

struct unit //一个解

{

double length;//代价，总长度

int path[nCities];//路径

bool operator < ( const struct unit &other) const

{

return length < other.length;

}

};

unit bestone = {INT\_MAX, {0} };//最优解

double length\_table[nCities][nCities];//distance

void init\_dis(); // create matrix to storage the Distance each city

void SA\_TSP();

void CalCulate\_length(unit &p);//计算长度

void print( unit &p);//打印一个解

void getNewSolution(unit &p);// 从邻域中获去一个新解

bool Accept(unit &bestone, unit &temp, double t);//新解以Metropolis 准则接受

int main(int argc, char\* argv[])

{

init\_dis();

SA\_TSP();

CalCulate\_length(bestone);

print(bestone);

return 0;

}

//stl 中 generate 的辅助函数对象

class GenbyOne {

public:

GenbyOne (int \_seed = -1): seed(\_seed){}

int operator() (){return seed += 1;}

private:

int seed;

};

void SA\_TSP()

{

srand(time(0));

int i = 0;

double r = SPEED;

double t = INITIAL\_TEMP;

const double t\_min = 0.001; //温度下限，若温度达到t\_min ，则停止搜索

//choose an initial solution ~

unit temp;

generate(temp.path, temp.path + nCities, GenbyOne(0));

random\_shuffle(temp.path, temp.path + nCities);

CalCulate\_length(temp);

memcpy(&bestone, &temp, sizeof(temp));

// while the stop criterion is not yet satisfied do

while ( t > t\_min )

{

for (i = 0; i < L; i++)

{

getNewSolution(temp);

if(Accept(bestone,temp, t))

{

memcpy(&bestone, &temp, sizeof(unit));

}

else

{

memcpy(&temp, &bestone, sizeof(unit));

}

}

t \*= r; //退火

}

return;

}

bool Accept(unit &bestone, unit &temp, double t)

{

if (bestone.length > temp.length)

{

return true;

}

else

{

if ((int)(exp((bestone.length- temp.length) / t) \* 100) > (rand() % 101) )

{

return true;

}

}

return false;

}

void getNewSolution(unit &p)

{

int i = rand() % nCities;

int j = rand() % nCities;

if (i > j)

{

int t = i;

i = j;

j = t;

}

else if (i == j)

{

return;

}

int choose = rand() % 3;

if ( choose == 0)

{//交换

int temp = p.path[i];

p.path[i] = p.path[j];

p.path[j] = temp;

}

else if (choose == 1)

{//置逆

reverse(p.path + i, p.path + j);

}

else

{//移位

if (j+1 == nCities) //边界处不处理

{

return;

}

rotate(p.path + i, p.path + j, p.path + j + 1);

}

CalCulate\_length(p);

}

void init\_dis() // create matrix to storage the Distance each city

{

int i, j;

ifstream in("source.txt");

for (i = 0; i < nCities; i++)

{

in >> nodes[i].num >> nodes[i].x >> nodes[i].y;

}

for (i = 0; i < nCities; i++)

{

length\_table[i][i] = (double)INT\_MAX;

for (j = i + 1; j < nCities; j++)

{

length\_table [i][j] = length\_table[j][i] =sqrt(

(nodes[i].x - nodes[j].x) \* (nodes[i].x - nodes[j].x) +

(nodes[i].y - nodes[j].y) \* (nodes[i].y - nodes[j].y) );

}

}

}

void CalCulate\_length(unit &p)

{

int j = 0;

p.length = 0;

for (j = 1; j < nCities; j++)

{

p.length += length\_table[ p.path[j-1] ][ p.path[j] ];

}

p.length += length\_table[ p.path[nCities - 1] ][ p.path[0] ];

}

void print( unit &p)

{

int i;

cout << "代价是：" << p.length << endl;

cout << "路径是：";

for (i = 0; i < nCities; i++)

{

cout << p.path[i] << " ";

}

cout << endl;

}

