tsp&旅行商问题

旅行商问题，即TSP问题（Traveling Salesman Problem）是数学领域中著名问题之一。

假设有一个旅行商人要拜访N个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。

路径的选择目标是要求得的路径路程为所有路径之中的最小值。

using namespace std;

const int city\_num = 10;//城市数量

const int unit\_num = 100;//群体规模

int ps = 10;//变异概率

const int genmax = 500;//最大迭代数

//城市间距离映射 最优解权值=10

int length\_table[10][10] = {

{0,1,1272,2567,1653,2097,1425,1177,3947,1},

{1,0,1,2511,1633,2077,1369,1157,3961,1518},

{1272,1,0,1,380,1490,821,856,3660,385},

{2567,2511,1,0,1,2335,1562,2165,3995,933},

{1653,1633,380,1,0,1,1041,1135,3870,456},

{2097,2077,1490,2335,1,0,1,920,2170,1920},

{1425,1369,821,1562,1041,1,0,1,4290,626},

{1177,1157,856,2165,1135,920,1,0,1,1290},

{3947,3961,3660,3995,3870,2170,4290,1,0,1},

{1,1518,385,993,456,1920,626,1290,1,0}

};

class Unit

{

public:

int path[city\_num];//个体的路径信息

int length;//个体价值

};

class Group

{

public:

Unit group[unit\_num];

Unit best;

int best\_gen;

Group()

{

best.length = 0x3f3f3f3f;

best\_gen = 0;

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

{

bool flag[city\_num] = {};

for(int j = 0; j < city\_num; j++)

{

int t\_city = rand()%city\_num;

while(flag[t\_city])

t\_city = rand()%city\_num;

flag[t\_city] = true;

group[i].path[j] = t\_city;

}

}

}

//对每个个体进行评估

void assess()

{

for(int k = 0; k < unit\_num; k++)

{

int rel = 0;

for(int i = 1; i < city\_num; i++)

rel += length\_table[group[k].path[i-1]][group[k].path[i]];

rel += length\_table[group[k].path[city\_num-1]][group[k].path[0]];

group[k].length = rel;

}

}

//根据评估结果对个体进行排序

void unit\_sort()

{

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

{

for(int j = i+1; j < unit\_num; j++)

{

if(group[i].length > group[j].length)

{

Unit temp;

memcpy(&temp, &group[i], sizeof(Unit));

memcpy(&group[i], &group[j], sizeof(Unit));

memcpy(&group[j], &temp, sizeof(Unit));

}

}

}

}

//交叉

Unit cross(Unit &father, Unit &mother)

{

int l = rand()%city\_num;

int r = rand()%city\_num;

if(l > r)

swap(l, r);

bool flag[city\_num] = {};

for(int i = l; i <= r; i++)

flag[father.path[i]] = true;

Unit son;

int pos = 0;

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

{

while(flag[mother.path[pos]])

pos++;

son.path[i] = mother.path[pos++];

}

for(int i = l; i <= r; i++)

son.path[i] = father.path[i];

for(int i = r+1; i < city\_num; i++)

{

while(flag[mother.path[pos]])

pos++;

son.path[i] = mother.path[pos++];

}

return son;

}

//突变

void mutation(Unit &t)

{

int proport = rand() % 100;

if(proport > ps)

return;

int one = rand()%city\_num;

int two = rand()%city\_num;

while(two != one)

two = rand()%city\_num;

swap(t.path[one], t.path[two]);

}

//输出信息

void print()

{

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

{

printf("第%d个个体，路径信息：", i);

for(int j = 0; j < city\_num; j++)

printf("%d ", group[i].path[j]);

printf(";总权值：%d;\n", group[i].length);

}

printf("最优个体，路径信息：");

for(int j = 0; j < city\_num; j++)

printf("%d ", group[0].path[j]);

printf(";总权值：%d;\n", group[0].length);

}

//种群进化

void work()

{

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

{

//如果进化层数大于20，加大变异的概率

if(i > 20)

ps \*= 3;

assess();//评估

unit\_sort();//根据评估结果排序

if(best.length > group[0].length)

{

memcpy(&best, &group[0], sizeof(group[0]));

best\_gen = i;

}

for(int j = 0; j+2 < unit\_num; j+=3)

group[j+2] = cross(group[j], group[j+1]);

for(int j = 0; j < city\_num; j++)//变异(从1开始，保留最优)

mutation(group[j]);

}

}

};

Unit group[unit\_num];//种群变量

Unit bestone;//记录最短路径

int generation\_num;//记录当前达到了第几代

int main()

{

srand((int)time(0));

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

{

Group g;

g.work();

printf("第%d次求解。路径：", i+1);

for(int j = 0; j < city\_num; j++)

printf("%d ", g.best.path[j]);

printf(";总权值：%d; 第%d代;\n", g.best.length, g.best\_gen);

}

return 0;

}

