实验八

实验题目：遗传算法

实验目的：熟悉遗传算法的实现过程

实验内容：遗传算法实现旅行商问题

实验程序及结果：

//population 种群

//Chromosome 染色体

//survival 存活

//crossover rate 交叉率

//mutation rate 突变率

//probability 概率

#include "stdio.h"

#include "stdlib.h"

#include "windows.h"

#include "time.h"

#define cityNum 10 //城市数量（基因数量）（染色体长度）

#define popSize 10 //种群大小（尺寸）

#define croRate 0.85 //交叉概率

#define mutRate 0.1 //变异概率

#define MAX 999 //进化代数

//定义染色体的结构

struct Chrom

{

int cityArr[cityNum]; //染色体的基因编码

char name; //染色体的名称

float adapt; //染色体的适应度

int dis; //染色体的路径长度

};

struct Chrom genes[popSize]; //定义基因库（结构体数组）

struct Chrom genesNew[popSize]; //重新建立一个新的种群

struct Chrom temp; //定义临时公用结点

char names[cityNum] = { 'A','B','C','D','E','F','G','H','I','J' }; //城市名称

int distance[cityNum][cityNum] = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }, //城市距离矩阵

{ 1, 0, 1, 2, 3, 4, 5, 6, 7, 8 },

{ 2, 1, 0, 1, 2, 3, 4, 5, 6, 7 },

{ 3, 2, 1, 0, 1, 2, 3, 4, 5, 6 },

{ 4, 3, 2, 1, 0, 1, 2, 3, 4, 5 },

{ 5, 4, 3, 2, 1, 0, 1, 2, 3, 4 },

{ 6, 5, 4, 3, 2, 1, 0, 1, 2, 3 },

{ 7, 6, 5, 4, 3, 2, 1, 0, 1, 2 },

{ 8, 7, 6, 5, 4, 3, 2, 1, 0, 1 },

{ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 } }; //最优解为18

void initGroup()

{

//初始化基因库

int i, j, k;

int t = 0;

int flag = 0;

srand(time(NULL));//初始化随机种子,防止随机数每次重复，常常使用系统时间来初始化,当srand()的参数值固定的时候，rand()获得的数也是固定的

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

{

//使用临时结点开始赋值

temp.name = names[i];

temp.adapt = 0.0f;

temp.dis = 0;

//产生10个不相同的数字

for (j = 0; j < cityNum;)

{

t = rand() % cityNum; //随机产生0-9的数

flag = 1;

for (k = 0; k < j; k++)

{

if (genes[i].cityArr[k] == t)

{

flag = 0;

break;

}

}

if (flag)

{

temp.cityArr[j] = t;

genes[i] = temp;//存入结构体数组，产生一个个体

j++;

}

}

}

}

//计算种群所有染色体的个体适应度

void popFitness()

{

int i, n1, n2;

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

{

genes[i].dis = 0;

for (int j = 1; j < cityNum; j++)

{

n1 = genes[i].cityArr[j - 1];

n2 = genes[i].cityArr[j];

genes[i].dis += distance[n1][n2];

}

genes[i].dis += distance[genes[i].cityArr[0]][genes[i].cityArr[cityNum - 1]];

genes[i].adapt = (float)1 / genes[i].dis; //每条染色体的路径总和(个体适应度)

}

}

//返回最优秀的一条染色体

int chooseBest()

{

int choose = 0;

float best = 0.0f;

best = genes[0].adapt;

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

{

if (genes[i].adapt < best)

{

best = genes[i].adapt;

choose = i;

}

}

return choose;

}

// 选择操作

void \_select()

{

float biggestSum = 0.0f;

float adapt\_pro[popSize];

float pick = 0.0f;

int i;

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

{

biggestSum += genes[i].adapt; // 总概率

}

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

{

adapt\_pro[i] = genes[i].adapt / biggestSum; // 概率数组

}

// 轮盘赌

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

{

pick = (float)rand() / RAND\_MAX; // 0到1之间的随机数

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

{

pick = pick - adapt\_pro[j];

if (pick <= 0)

{

genesNew[i] = genes[j];

break;

}

}

}

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

{

genes[i] = genesNew[i];

}

}

// 交叉操作

void cross()

{

float pick;

int choice1, choice2;

int pos1, pos2;

int temp;

int conflict1[popSize]; // 冲突位置

int conflict2[popSize];

int num1;

int num2;

int index1, index2;

int move = 0;

int i,j,k; // 当前移动的位置

while (move < popSize - 1)

{

pick = (float)rand() / RAND\_MAX; // 用于决定是否进行交叉操作

if (pick > croRate) //两条染色体是否相爱

{

move += 2;

continue; // 本次不进行交叉

}

// 采用部分映射杂交

choice1 = move; // 用于选取杂交的两个父代

choice2 = move + 1; // 注意避免下标越界

pos1 = rand() % popSize;

pos2 = rand() % popSize;

while (pos1 > popSize - 2 || pos1 < 1)//如果位置在开头或结尾(因为全部交换无意义)

{

pos1 = rand() % popSize;

}

while (pos2 > popSize - 2 || pos2 < 1)

{

pos2 = rand() % popSize;

}

if (pos1 > pos2)

{

temp = pos1;

pos1 = pos2;

pos2 = temp; // 交换pos1和pos2的位置

}

for (int j = pos1; j <= pos2; j++)// 逐个交换顺序

{

temp = genes[choice1].cityArr[j];

genes[choice1].cityArr[j] = genes[choice2].cityArr[j];

genes[choice2].cityArr[j] = temp;

}

num1 = 0;

num2 = 0;

if (pos1 > 0 && pos2 < popSize - 1)//分三段

{

for (int j = 0; j < pos1; j++)

{

for (int k = pos1; k <= pos2; k++)

{

if (genes[choice1].cityArr[j] == genes[choice1].cityArr[k])

{

conflict1[num1] = j;

num1++;

}

if (genes[choice2].cityArr[j] == genes[choice2].cityArr[k])

{

conflict2[num2] = j;

num2++;

}

}

}

for (int j = pos2 + 1; j < popSize; j++)

{

for (int k = pos1; k <= pos2; k++)

{

if (genes[choice1].cityArr[j] == genes[choice1].cityArr[k])

{

conflict1[num1] = j;

num1++;

}

if (genes[choice2].cityArr[j] == genes[choice2].cityArr[k])

{

conflict2[num2] = j;

num2++;

}

}

}

}

if ((num1 == num2) && num1 > 0)

{

for (int j = 0; j < num1; j++)

{

index1 = conflict1[j];

index2 = conflict2[j];

temp = genes[choice1].cityArr[index1]; // 交换冲突的位置

genes[choice1].cityArr[index1] = genes[choice2].cityArr[index2];

genes[choice2].cityArr[index2] = temp;

}

}

move += 2;

}

}

//变异操作

void mutation()

{

double pick;

int pos1, pos2, temp;

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

{

pick = (float)rand() / RAND\_MAX; // 用于判断是否进行变异操作

if (pick > mutRate)

{

continue;

}

pos1 = rand() % popSize;

pos2 = rand() % popSize;

while (pos1 > popSize - 1)

{

pos1 = rand() % popSize;

}

while (pos2 > popSize - 1)

{

pos2 = rand() % popSize;

}

int a = genes[i].dis;

temp = genes[i].cityArr[pos1];

genes[i].cityArr[pos1] = genes[i].cityArr[pos2];

genes[i].cityArr[pos2] = temp;

popFitness();//更新数据

//此步骤的作用在于检查是否变异后得到的个体比变异前更优秀了，如若往坏的方向变化了，那还不如不变异了

//（强制返回，虽然有点违背事物的客观发展规律，但为了增强程序的收敛性，该操作还是有必要的）（偷笑）

if (genes[i].dis > a)

{

temp = genes[i].cityArr[pos1];

genes[i].cityArr[pos1] = genes[i].cityArr[pos2];

genes[i].cityArr[pos2] = temp;

}

}

}

int main()

{

char c = 0;

printf("\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 遗传算法求解TSP(旅行商)问题 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

initGroup(); //初始化

popFitness(); //更新数据

//输出配置信息

printf("\n\t\t基因长度:%d", cityNum);

printf("\t种群大小:%d", popSize);

printf("\t交叉概率:%.2f", croRate);

printf("\t变异概率:%.2f", mutRate);

printf("\t进化代数:%d", MAX);

printf("\t预设最优解：18");

printf("\n\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

//输出距离矩阵

printf("\n\t\t--------- 城市距离矩阵 ---------\n");

printf("\t\t");

int i, j;

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

{

for (j = 0; j < cityNum; j++)

{

printf(" %d", distance[i][j]);

}

printf("\n\t\t");

}

printf("--------------------------------");

//输出路径信息

printf("\n\t\t-------- 初始种群基因库 --------\n");

printf("\t\t ");

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

{

printf(" %c", genes[i].name);

}

printf("\n\t\t");

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

{

printf("%c", genes[i].name);

for (j = 0; j < cityNum; j++)

{

printf(" %d", genes[i].cityArr[j]);

}

printf("\n\t\t");

}

printf("--------------------------------\n");

do

{

printf("\n\t\t寻求最优解中：");

//通过不断进化，直到达到定义的进化代数

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

{

\_select();

cross();

mutation();

popFitness();//更新数据

int temp = (int)MAX / 20;

if (i % temp == 0)

{

printf("▊");

Sleep(200);

}

}

printf("完成");

printf("\n\n\t\t最优路径：");

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

{

printf("%d-->", genes[chooseBest()].cityArr[i]);

}

printf("%d", genes[chooseBest()].cityArr[0]);

printf("\n\n\t\t路径长度:%d", genes[chooseBest()].dis);

printf("\n\n\t\t是否再试一次?(Y/y) 是/(N/n) 否：");

fflush(stdin);

c = getchar();

fflush(stdin);

if (c == 'N' || c == 'n')

{

break;

}

} while (1);

printf("\n\t\t");

return 0;

}

