TravelingSalesman.cpp

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Case::Case(float\*\* cities) {

amountCities = sizeof(cities)/sizeof(cities[0]);

genome = new int[amountCities+1];

}

Case::~Case(){

delete[] genome;

{

float Case::getFitness()

{

}

void Case::setGenome(){

int e, g;

srand(time(Null));

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

g=i;

e= generateRandomNumber(0, amountCities)

for(int d=0;d<=g;d++){

if(e==genome[d]){

g=g-g;

i=i-1;;

}

while((g==i)&&(e!=genome[d])&&(d==i)){

genome[i]=e;

}

}

}

genome[amountCities+1]=genome[0];

}

int\* Case::getGenome()

{

return genome;

}

void TravellingSalesman::sort(Case\* population, int populationSize)

{

bool swapped;

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

swapped = false;

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

if (population[j].getFitness > population[j+1].getFitness){

swap(&populationSize[j], &populationSize[j+1]);

swapped = true;

}

}

if (swapped == false)

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Case Class \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class Case() {

private: //@author Jan Pablo Murillo - B95447

int\* genome;

int amountCities;

float\*\* cities;

public:

Case(float\*\* cities); //@author Jan Pablo Murillo - B95447 ^ int cities -> 'size'

~Case(); //@author Jan Pablo Murillo - B95447

/// Methods

void setGenome(); //@author Jan Pablo Murillo - B95447

float getFitness(); //@author Jimena Gdur - B93250

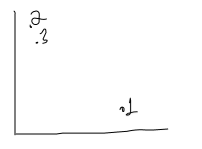
int\* getGenome(); //@author Jan Pablo Murillo - B95447

};

Primero te voy a explicar más o menos la tarea. Entonces la cosa es que tenemos una persona que tiene que hacer un recorrido verdad. Este tiene una lista de ciudades por las que solo puede pasar una ves, y volver a la primera. Entonces solo puede repetir la primera ciudad. El profe nos da unos casos de prueba, que son matrices y tienen coordenadas (x,y) como si fuera un eje cartesiano. Entonces tratamos esto como si fueran puntos en un eje cartesiano y tenemos que encontrar la mejor ruta, es decir en la que la persona recorre la menor distancia. Entonces digamos que tengo las ciudades:

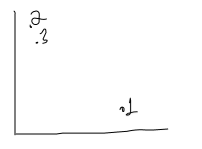
1: (24,6), 2: (3, 20), 3: (4,15)….

Tengo que buscar cuál es el mejor orden en que puedo escoger las ciudades, de forma que cuando sume la distancia que recorrió sea la mejor.



La forma que el profe quiere que logremos esto es usando un algoritmo que es muy parecido al de la genética. Este funciona así:

1. Se crea una lista de individuo que escogen un orden de ciudades al azar, por ejemplo: 1,3,2,1 (recordando que tiene que volver a 1).
2. Se calcula para cada individuo cuánto recorrió, haciendo Pitágoras. En la foto solo calcule la distancia entre el punto 1 y 3, pero se hace para: (3 y 2) y (2 y 1). Este ejemplo es tomando el orden del individuo específico que puse en el punto 1.





1. Entonces se guarda esta distancia en una variable que se llama “fitness” y la lista de individuos (cada uno con su fitness) se ordena de menor a mayor.
2. Ahora con esto podemos ejecutar una época, una época o “epoch” crea una nueva generación de individuos (te dicen cuántos individuos va a tener cada época), seleccionando los mejores de la época pasada (la cantidad específica te la dan como parámetro, este proceso se llama *elitism*), seleccionando nuevos (la cantidad específica te la dan como parámetro, este proceso se llama *spontaneous*), y el resto de los campos (poblaciónDeEpoca - cantidad*Elitism* – cantidad*Spontaneous*) obtienen mutando algunos de la época pasada (el proceso de mutación sigue las instrucciones en la foto).