INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Genetic Algorithms

Practica 3:

Profa.: MORALES GUITRON SANDRA LUZ

Grupo: 3CM5

Alumno:

Salcedo Barrón Ruben Osmair.

MEXICO, D.F. a 11 de octubre del 2018

Introducción

Propuesto por DeJong, es posiblemente el método más utilizado desde los orígenes de los Algoritmos Genéticos [Blickle and Thiele, 1995].

A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta, de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población está ordenada en base al ajuste por lo que las porciones más grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo [0...1] y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Es un método muy sencillo, pero ineficiente a medida que aumenta el tamaño de la población (su complejidad es O(n^2). Presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez.

Desarrollo

En este algoritmo de ruleta, implementamos 4 arreglos uno para población inicial, de individuos seleccionados por ruleta, de cruza y de mutación. Primero se llena aleatoriamente el arreglo de población inicial con series de 5 bits después se hace el algoritmo de ruleta en este se genera un numero aleatorio entre 0 y el valor de la aptitud total de la población inicial, después se acumula la actitud de la población y este se detiene cuando supera el valor aleatorio generado. Por ultimo el padre será el individuo que genere que se sobrepase el valor aleatorio generado.

Una vez teniendo la población de selección de padres, se realiza una cruza de individuos y al obtener la población de individuos después de la cruza se necesita realizar una mutación. Para este caso será una mutación del 10 % de la población. Nuestra población total es de 32 elementos. Una vez que se tenga la población mutada, se establece esta como población inicial, para realizar el algoritmo de ruleta en la siguiente generación.

GENETIC ALGORITHMS

Generation: 1

N	inicial		x^2	seleccion	cruza	mutacion
1	11110	30		11101	11000	11000
2	10011	19		11000	11101	11101
3	00101	5		10001	10101	10101
4	01101	13		00101	00001	00001
5	00110	6		11101	11110	11110
6	10000	16		00110	00101	00101
7	11001	25		10110	10101	10101
8	11000	24		11101	11110	11110
9	11101	368		00101	00011	00011
10	00010	2		00011	00101	00101
11	10010	18		11000	11010	11010
12	00111	7		00010	00000	00000
13	00001	1		00010	00011	00011
14	11101	29		10011	10010	10010
15	10000	16		00101	00001	00001
16	01110	14		10001	10101	10101
17	00011	961		10000	10110	10110
18	01010	10		10110	10000	10000
19	01101	13		00110	00110	00110
20	01001	9		01110	01110	01110
21	00011	3		00010	00001	00001
22	00010	2		00001	00010	00010
23	11110	30		00101	00001	00011
24	00100	4		10001	10101	10101
25	10111	0		00101	00001	00001
26	00101	5		11001	11101	11111
27	10100	20		00110	00000	00000
28	10110	22		10000	10110	10110
29	11101	29		11000	11101	11101
30	10001	17		11101	11000	11100
31	10001	17		11101	11101	11101
32	10001	17		11101	11101	11101

Generation: 2

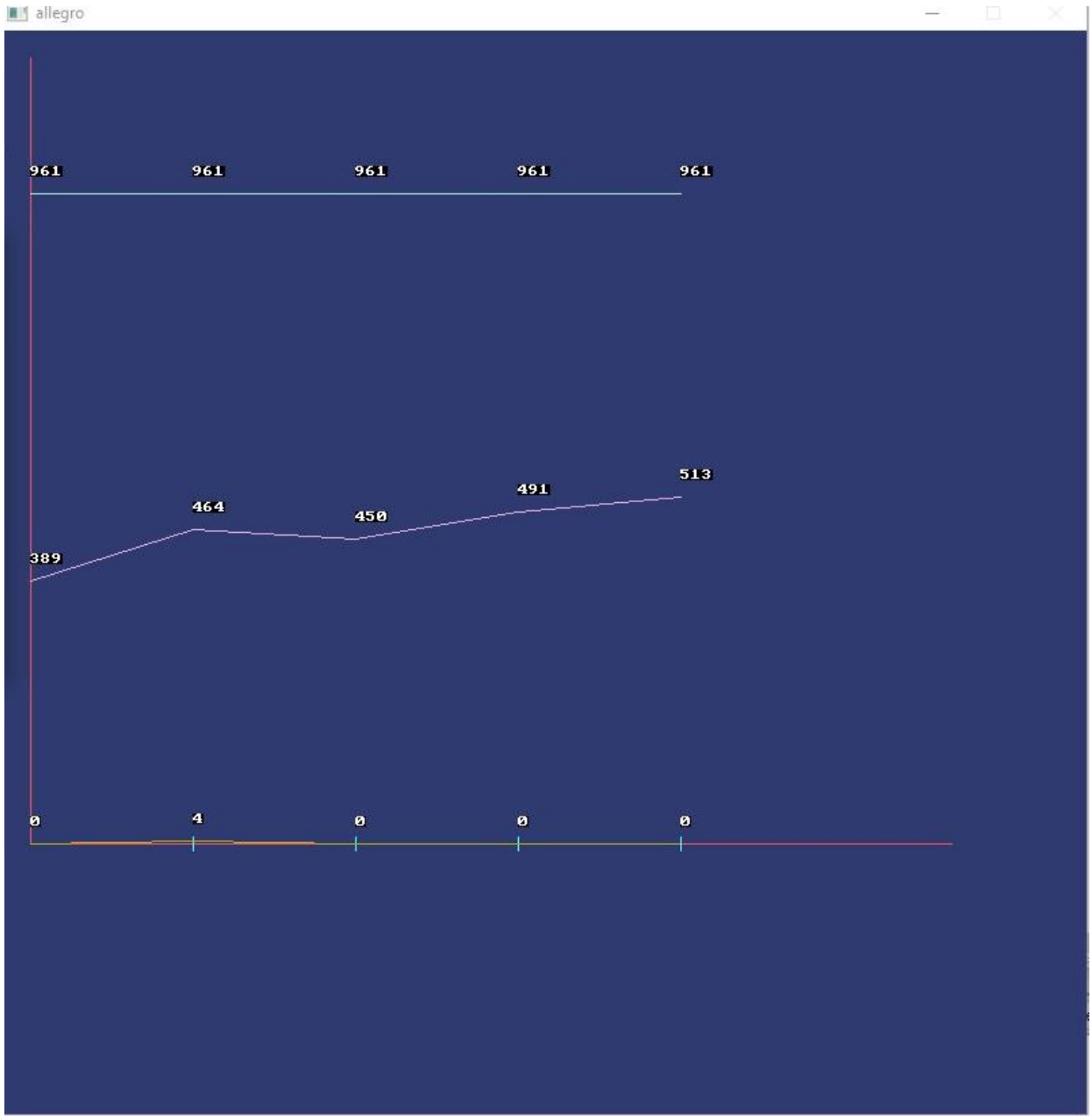
N	inicial		x^2	seleccion	cruza	mutacion
1	11000	24		11101	11101	11111

31	11100	28	00101	00110	00110
32	11110	30	11110	11101	11101

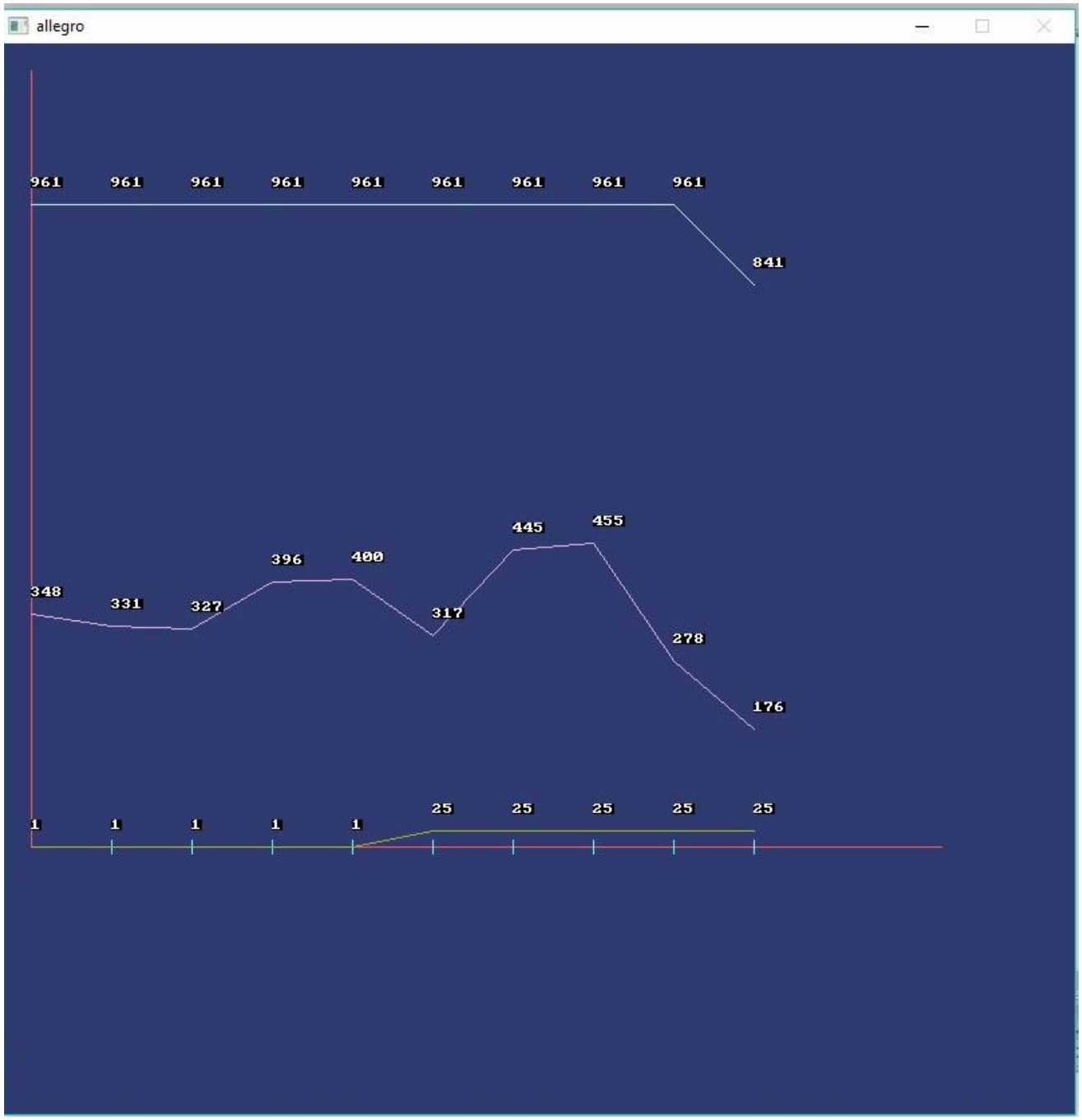
Generation: 5

N	inicial	x^2	seleccion	cruza	mutacion
1	00101	5	00001	00001	00001
2	10101	21	11001	11001	11001
3	00011	3	10101	10001	10001
4	00100	4	00001	00101	00101
5	11110	30	00101	00101	00101
6	11001	25	00101	00101	00101
7	10101	21	11101	11101	11101
8	00001	1	11101	11101	11101
9	11001	25	10101	10101	10101
10	00101	5	11101	11101	11101
11	00110	6	11101	11110	11110
12	11001	25	00110	00101	00101
13	00101	413	11101	11110	11110
14	11100	28	00110	00101	00101
15	11101	29	11001	11110	11110
16	11101	29	00110	00001	00001
17	00001	1	10101	10101	10101
18	11101	29	10101	10101	10101
19	11101	29	00001	00011	00011
20	10101	21	11011	11001	11101
21	11101	961	11101	11110	11110
22	00110	6	00110	00101	00101
23	10011	19	11101	11110	11111
24	11011	27	00110	00101	00101
25	00100	4	11101	11001	11001
26	10101	21	11001	11101	11101
27	00101	5	00101	00110	00110
28	10101	21	00110	00101	00101
29	11101	1	10101	10101	10101
30	11101	29	00101	00101	00101
31	00110	6	11101	11101	11101
33	11101	29	00101	00101	00111

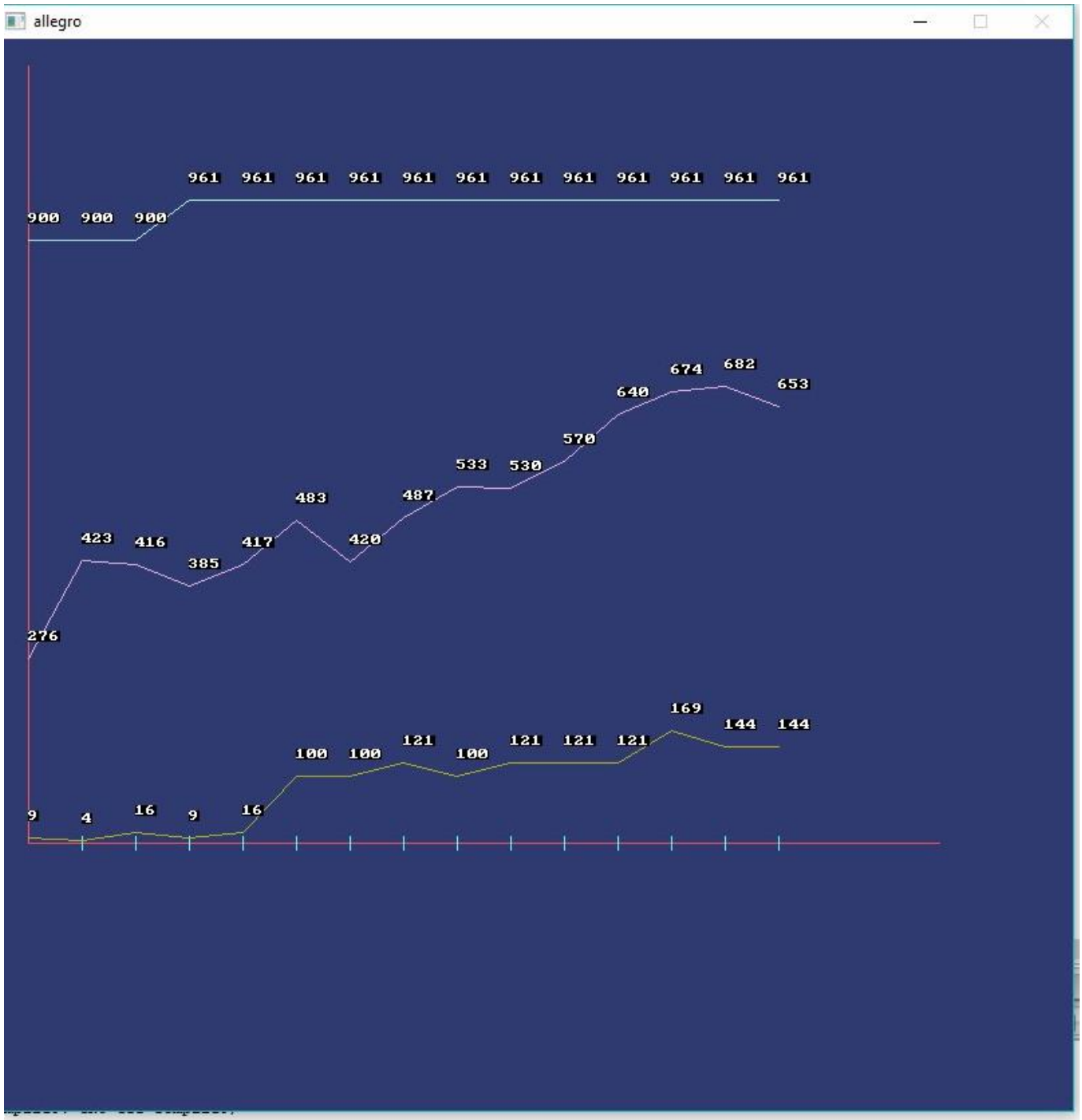
GENETIC ALGORITHMS



GENETIC ALGORITHMS



GENETIC ALGORITHMS



Conclusiones

En esta practica pudimos ver la evolución de los individuos mediante el algoritmo de ruleta, pero debido a que es aleatorio esta selección puede que en algunos casos el individuo seleccionado sea el que tenga menor aptitud con respecto a los otros. Sin embargo, el uso de este algoritmo nos puede ayudar a ver como seria la selección de la naturaleza que no siempre el mas apto es el seleccionado.

Código

main.cpp

```
#include <allegro.h>
#include "inicia.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <bitset>
#include "ruleta.h"

using namespace std;
int main ()
{
    int k=0,i=0;
    ofstream tabla;
    tabla.open("resultado.txt", fstream::out);
    int num_generaciones;
    printf("Cuantas generaciones:");
    scanf("%d",&num_generaciones);

    bitset<BIT_IND> inicial[NUM_IND];
    bitset<BIT_IND> seleccion[NUM_IND];
    bitset<BIT_IND> cruza[NUM_IND];
    bitset<BIT_IND> mutacion_des[NUM_IND];

    int minimo[num_generaciones];
    int maximo[num_generaciones];
    int generationValuesAverage[num_generaciones];
    int ind_apti[NUM_IND];

    IniciarInd(inicial);
    for ( k = 0; k < num_generaciones; k++){

            int totalAptitude = getTotalAptitude(inicial);

            for(i=0;i<NUM_IND;i++){
                ind_apti[i]=(int)getIndividualAptitude(inicial[i]);
            }

            for ( i = 0; i < NUM_IND; i++)
            {
                seleccion[i] = rouletteSelection(inicial, totalAptitude);
```

```
            }

            for ( i = 0; i < NUM_IND; i+=2)
            {
                cruza[i] = crossAlgorithm(seleccion[i], seleccion[i +
1]);
                cruza[i + 1] = crossAlgorithm(seleccion[i + 1],
seleccion[i]);

            }

            for ( i = 0; i < NUM_IND; i++)
            {
                mutacion_des[i] = cruza[i];
            }

            int mutation_value = NUM_IND / PROBABILITY;

            srand (time(NULL));

            for (i = 0; i < mutation_value; i++)
            {
                int indivual_to_mutate = rand() % NUM_IND;
                mutacion_des[indivual_to_mutate] =
mutationAlgorithm(cruza[indivual_to_mutate]);
            }

            int min_gen_value = getMinGenerationValue(mutacion_des);
            int max_gen_value = getMaxGenerationValue(mutacion_des);
            int gen_average = getGenerationAverage(mutacion_des);

            minimo[k] = min_gen_value;
            maximo[k] = max_gen_value;
            generationValuesAverage[k] = gen_average;

            tabla << "Generation: " << k+1 << endl;
            tabla << "N\t\t
inicial\t\tx^2\t\tseleccion\t\tcruza\t\tmutacion\t\t"<<endl;
            int indice=1;
            for (int i = 0; i < NUM_IND; i++)
            {
                tabla << indice <<"\t\t"<< inicial[i] <<
"\t\t\t"<<ind_apti[i]<<"\t\t\t" << seleccion[i] << "\t\t\t"<< cruza[i] <<
"\t\t\t" << mutacion_des [i] << endl;
                indice+1;
             }

            for (int i = 0; i < NUM_IND; i++)
            {
                inicial[i] = mutacion_des[i];
            }

    }

    int sep=600/num_generaciones;
    /*****************************************/
```

```
    inicia_allegro(800,800);
    int in=20;
    BITMAP *buffer = create_bitmap(800,800);
    clear_to_color(buffer, 0x0a6c92);
    line(buffer, 20, 600, 700, 600,  palette_color[11]);
    line(buffer, 20, 600, 20, 20,  palette_color[11]);
    textout_centre_ex(buffer, font,"histograma", 160, 25, 0xFFFFFF,
0x999999);
    textout_centre_ex(buffer, font,"Generaciones", 160, 650, 0xFFFFFF,
0x999999);
    for(i=0;i<num_generaciones-1;i++){
        line(buffer, in, 600-(minimo[i]/2), in+sep, 600-(minimo[i+1]/2),
0xbde4ff);
        line(buffer, in, 600-(maximo[i]/2), in+sep, 600-(maximo[i+1]/2),
0xe5ffdc);
        line(buffer, in, 600-(generationValuesAverage[i]/2), in+sep, 600-
(generationValuesAverage[i+1]/2),  0xe5b0dc);

        line(buffer, in+sep, 605, in+sep, 595,  palette_color[11]);

        in=in+sep;
    }
    in=20;
    for(i=0;i<num_generaciones;i++){
         textprintf(buffer, font, in, 580-(minimo[i]/2), 0xFFFFFF,
"%d",(minimo[i]));
         textprintf(buffer, font, in, 580-(maximo[i]/2), 0xFFFFFF,
"%d",(maximo[i]));
         textprintf(buffer, font, in, 580-(generationValuesAverage[i]/2),
0xFFFFFF, "%d",(generationValuesAverage[i]));
         textprintf(buffer, font, in, 610, 0xFFFFFF, "%d",(i));
        in=in+sep;
    }




    blit(buffer, screen, 0, 0, 0, 0, 800, 800);
    readkey();

    destroy_bitmap(buffer);

    return 0;
}
END_OF_MAIN ()
```

ruleta.cpp

```
#include "ruleta.h"
#include <iostream>
#include <time.h>
using namespace std;

void IniciarInd(bitset<BIT_IND> array[]){
      srand (time(NULL));
      for (int i = 0; i < NUM_IND; i++)
```

```
                  array[i] = 1+rand()%(101-1);
}
void printIndividuals(bitset<BIT_IND> array[]){
      for (int i = 0; i < NUM_IND; i++)
            cout << array[i].to_ulong() << endl;
}

int getIndividualValue(bitset<BIT_IND> individual){
      return individual.to_ulong();
}

int getIndividualAptitude(bitset<BIT_IND> individual){
      return (individual.to_ulong()*individual.to_ulong());
}

int getTotalAptitude(bitset<BIT_IND> array[]){
      int total = 0;
      for (int i = 0; i < NUM_IND; i++)
      {
            total += getIndividualAptitude(array[i]);
      }
      return total;
}

bitset<BIT_IND> rouletteSelection(bitset<BIT_IND> array[], int
totalAptitude) {

    int r = rand() % (totalAptitude + 1);
    int add = 0;
    int i;
    for(i = 0; i < NUM_IND && add < r; i++) {
        add += getIndividualAptitude(array[i]);
    }
    return array[i];
}

bitset<BIT_IND> crossAlgorithm(bitset<BIT_IND> &p1, bitset<BIT_IND> &p2)
{

    bitset<BIT_IND> aux = p1;

    for (int i = 0; i <= PUNTO_CRUZA; i++)
    {
      aux.set(PUNTO_CRUZA - i, p2[PUNTO_CRUZA - i]);
    }

    return aux;
}

bitset<BIT_IND> mutationAlgorithm(bitset<BIT_IND> individual){
      bitset<BIT_IND> result = individual;

      int cont = 0;

      while(cont <= MAX_SEARCH_VALUE)
      {
```

```
            int mutation_point = rand() % BIT_IND;
            if(result[mutation_point] == 0){
                  result.set(mutation_point, 1);
                  break;
            }
            cont++;
      }

      return result;
}

int getMinGenerationValue(bitset<BIT_IND> array[]){
      int min = 1000000, aux = 0;

      for (int i = 0; i < NUM_IND; i++)
      {
            aux = getIndividualAptitude(array[i]);

            if(aux < min){
                  min = aux;
            }

      }
      return min;
}

int getMaxGenerationValue(bitset<BIT_IND> array[]){
      int max = 0, aux = 0;

      for (int i = 0; i < NUM_IND; i++)
      {
            aux = getIndividualAptitude(array[i]);

            if(aux > max){
                  max = aux;
            }

      }
      return max;
}

int getGenerationAverage(bitset<BIT_IND> array[]){
      return (getTotalAptitude(array)/NUM_IND);
}
```

inicia.cpp

```
#include "inicia.h"
#include <allegro.h>

void inicia_allegro(int ANCHO_ , int ALTO_){
   allegro_init();
   install_keyboard();

   set_color_depth(32);
```

```
   set_gfx_mode(GFX_AUTODETECT_WINDOWED, ANCHO_, ALTO_, 0, 0);
}

int inicia_audio(int izquierda, int derecha){
    if (install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, NULL) != 0) {
       allegro_message("Error: inicializando sistema de sonido\n%s\n",
allegro_error);
       return 1;
    }

      set_volume(izquierda, derecha);
    return 0;
}
```