Reporte practicas Teoría Computacional

ESCUELA SUPERIOR DE CÓMPUTO

Colin Heredia Luis Antonio
Juárez Martínez Genaro
Grupo: 2CM5

4 de marzo de 2020

Reportes de las practicas.

Practica 1 (Combinaciones)

En esta practica se se debe dar un numero n dado por el usuario o generado para obtener todas las posibles combinaciones de 0's y 1's. Se llenara un archivo todas las combinaciones posibles, contando de diferentes maneras los unos que existen dentro de ese archivo para generar una gráfica con ellos.

MainP1.cpp

```
/*
 Author : Colin Heredia Luis Antonio

 Version 1.0
 Descripcion: Mostrar las posibles combinaciones del conjunto binario dado
                        los parametros por el usuario o de manera automatica

 */

#include <iostream>
#include "Comb.hpp"      // Clase Comb.h

using namespace std;
int main()
{
        int k = 0;
        int operacion=0; // operacion que se va aplicar.
        Comb comb;

        cout << "\n Teclea el numero del metodo a utilziar. Alfabeto : {1,0} " <<
            endl;
        cout << " 1.- Generar numero aleatorio. \n 2-. Introducir un numero
            manualmente." << endl;
        cout << "\n Opcion : ";
        cin >> operacion;

        if(operacion == 1)
        {
                // caso en el que se debe generar aleatoriamente el numero
                k = comb.genValor();
                cout<<"Valor Genreado : " << k << endl;
                cout<<"\n\nE={"<<"e,";
                for(int i = 2; i<=k; i++)
                {
                        comb.newComb(i);
                        comb.startCombinacion27();
                }
                cout<<"}"<<endl;
                //comb.leerUnos(100,"./txt/Universo27.txt");
        }else
        {
                cout<<"\n Teclea el valor de k para calcular el universo : ";
                cin >> k;
                // el numero sera ingresado manualmente.
```

```
                cout<<"Valor Digitado : " << k << endl;
                if(!(k <= 0 || k >= 28))
                {

                        cout<<"\n\nE={"<<"e,";
                        for(int i = 2; i<=k; i++)
                        {
                                comb.newComb(i);
                                comb.startCombinacion27();
                        }
                        cout<<"}"<<endl;

                        //comb.leerUnos(5,"./txt/Universo.txt");
                }else
                        cout<<"El numero debe ser entero entre 1 a 27"<<endl;
        }
        return 0;
}
```

Comb.cpp

```
/*
        Clase Comb
        Author : Colin Heredia Luis Antonio
*/
#include <iostream>
#include <cmath>        // funcniones matematicas Pow
#include <cstdlib>      // donde se encuentra la funcion RAND y SRAND
#include <ctime>        // time para la funcion de uso de tiempo
#include <fstream>      // para los archivos
#include <iomanip> // parametros de manupulacions
#include <bits/stdc++.h> // funcion reverse.
#include "Comb.hpp"
using namespace std;


/**
 *  Destructor
 */
Comb::~Comb(){}
/**
 *  Constructor inicializacion de binarios
 */
Comb::Comb()
{
        binario[0] = '0';
        binario[1] = '1';
}
/**
 * [Comb::newComb description]
 * Vuelve inicializar las variables el valor de k lo establece a nuestra k
 * de nuestra clase y manda el valor a la funcion calcular K.
 * @param k [description]
 */
void Comb::newComb(int k)
{
```

```
        // constructor vacio
        this->k = 0;
        this->x = 0;
        binario[0] = '0';
        binario[1] = '1';
        this->k = k;
        this->x = calcularX(k);

}
/**
 * [Comb::calcularX description]
 * Obtiene el valor de X que son el numero de combinaciones obtenidas
 * por ejemplo 2^5 = 32
 * @param  numero [valor del exponente]
 * @return        [regresa el valor de elevar 2 al numero]
 */
int Comb::calcularX(int numero)
{
        return pow(2,numero);
}

/**
 * [Comb::genValor description]
 * Genera un valor aleatorio dentro de un rango
 * @return [regresa el valor random generado]
 */
int Comb::genValor() const{
        // funcion de time , recupera la hora del sistema
        // lo cual hace que la semilla que le demos a srand sea aleatoria
        // dado que en teoria siempre sera una semilla distinta
        srand(time(NULL)); // le damos la semilla a srand
        int random = (rand()%100000)+1;

        return random;
}

/**
 * [Comb::startCombinacion description]
 * Empieza a realizar las combinaciones lo que hice fue convertir los valores
 * de 0 hasta el numero de combinaicones X convertirlos a binario.
 * los escribe dentro del archivo Universo y las combinaciones de las cadenas
 */
void Comb::startCombinacion()
{
        int aux;
        this->cadena.clear();

        for(int i = 0; i<x; i++)
        {
                aux = i;
                for(int j = 0; j<k; j++)
                {
                        cadena += binarioResiduo(aux);
                        aux /= 2;
                        reverse(cadena.begin(),cadena.end());
```

```
                }
                escribeUniverso(cadena);
                cadena += ",";
                escribeCombinacion(cadena);
                cout<<cadena;
                this->cadena.clear();
        }
}
/**
 * [Comb::startCombinacion27 description]
 * Algoritmo para la combinaicon de k=27 cambia al momento de
 * escribir en el archivo.
 */
void Comb::startCombinacion27()
{
        int aux;
        this->cadena.clear();

        for(int i = 0; i<x; i++)
        {
                aux = i;
                for(int j = 0; j<k; j++)
                {
                        cadena += binarioResiduo(aux);
                        aux /= 2;
                        reverse(cadena.begin(),cadena.end());
                }
                escribeUniverso27(cadena);
                cadena += ",";
                escribeCombinacion(cadena);
                cout<<cadena;
                this->cadena.clear();
        }
}

/**
 * [Comb::binarioResiduo description]
 * Funcion para devolver el residuo al dividir entre 2
 * esto funciona para convertir a binario.
 * @param  numero [numero a convertir]
 * @return        [regresa el valor de 0 o 1]
 */
char Comb::binarioResiduo(int numero)
{
        if(!(numero == 0))
                numero%=2;
        if(numero>0)
                return binario[1];
        else
                return binario[0];
}

/**
 * [Comb::escribeCombinacion description]
 * Escribe la cadena en un archivo de texto Combinaciones.txt
```

```
 * @param cadCom [cadena a escribir]
 */
void Comb::escribeCombinacion(string cadCom)
{
        // primer archivo de texto llamado Combinaciones.txt dentro de txt
        ofstream txtComb;
        txtComb.open("./txt/Combinaciones.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtComb<<cadCom;
        txtComb.close();
}
/**
 * [Comb::escribeUniverso description]
 * Funcion va concatenando la cadena en el archivo Universo. en este no se agregan
    ','
 * todo se va concatenando directamente.
 * @param cadCom [description]
 */
void Comb::escribeUniverso(string cadCom)
{
        // algoritmo para guardar las combinacione del universo
        // primer archivo de texto llamado Combinaciones.txt dentro de txt
        ofstream txtComb;
        txtComb.open("./txt/Universo.txt",ios::in|ios::app); // entrada y a adir
        // salto de linea
        txtComb<<cadCom;
        txtComb.close();
}
/**
 * [Comb::escribeUniverso27 description]
 * Esta funcion es la misma que escribeUniverso
 * Solo que aqui se escribe en Universo27.txt
 * @param cadCom [cadena a escribir]
 */
void Comb::escribeUniverso27(string cadCom)
{
        // algoritmo para guardar las combinacione del universo
        // primer archivo de texto llamado Combinaciones.txt dentro de txt
        ofstream txtComb;
        txtComb.open("./txt/Universo27.txt",ios::in|ios::app); // entrada y a adir
        // salto de linea
        txtComb<<cadCom;
        txtComb.close();
}
/**
 * [Comb::segmentarVeinte description]
 * escribe el valor de unos en el archivo de segmentar veinte
 * que son los numeros contados, esto se escribe en un archivo llamado
    segmentosVeinte.txt
 * @param numero [numeor a escrbir]
 */
void Comb::segmentarVeinte(int numero)
{
        ofstream txtComb;
```

```
        txtComb.open("./txt/segmentosVeinte.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtComb<<numero<<endl;
        txtComb.close();
}
/**
 * [Comb::segnmentarSesenta description]
 * Escribe los numeros de unos contados en segmentos de sesenta
 * esto en un archivo llamdo segmentoSesenta.txt
 * @param numero [numero a escrbier]
 */
void Comb::segnmentarSesenta(int numero)
{
        ofstream txtComb;
        txtComb.open("./txt/segmentoSesenta.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtComb<<numero<<endl;
        txtComb.close();
}
/**
 * [Comb::segmentarCien description]
 * Escribe el numeor de unos contados en segmentos de cien en cien
 * esto en un archivo llamado segmentoCien.txt
 * @param numero [Valor contado]
 */
void Comb::segmentarCien(int numero)
{
        ofstream txtComb;
        txtComb.open("./txt/segmetoCien.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtComb<<numero<<endl;
        txtComb.close();
}
/**
 * [Comb::segmentarOtro description]
 * Se puede segmenetar en diferentes valores la cadena universo
 * esta funcion es si no esta dentro de los parametros dados en la rpactica
 * si queremos segmentar la cadena en otros valores podemos hacerlo
 * esto escribira el valor dentro de un archivo otroSegmento.txt
 * @param numero [valor a escribir]
 */
void Comb::segmentarOtro(int numero)
{
        ofstream txtComb;
        txtComb.open("./txt/otroSegmento.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtComb<<numero<<endl;
        txtComb.close();
}
/**
 * [Comb::leerUnos description]
```

```
 * La funcion lee un archivo dado por su path y lo manda a contar sus unos
 * con un segmento determinado
 * @param segmentos    [leer en segmentos]
 * @param pathArchivo [direccion del archivo a leer]
 */
void Comb::leerUnos(int segmentos,string pathArchivo)
{
        string numeros;
        int unos = 0;
        int contador = 0;
        ifstream archivo(pathArchivo);
        if(archivo.fail())
                cerr<<"Error al abrir el archivo"<<endl;
        else{
                getline(archivo,numeros);
                contarUnos(numeros,segmentos);
        }
}
/**
 * [Comb::contarUnos description]
 * Cuenta los unos dentro de una cadena dada.
 * y el numero de segmentos dados
 * @param cadena     [cadena a leer]
 * @param segmentos [segmentos de unos a contar]
 */
void Comb::contarUnos(string cadena,int segmentos)
{
        int unos=0;
        for(int i = 0; i<cadena.size(); i++)
        {
                if(cadena[i] == binario[1])
                        unos++;
                cout<<" i="<<i<<" Valor cadena: "<<cadena[i]<<" Valor de unos: "<<
                    unos<<endl;
                if(((i+1)%segmentos) == 0)
                {
                        agregarUnos(unos,segmentos);
                        unos = 0; // reiniciamos el contador.
                }
        }
        if(unos != 0)
                agregarUnos(unos,segmentos);
}
/**
 * [Comb::agregarUnos description]
 * determina en donde escribir el numero de unos contados
 * Ya sea de 20,60,100 u otro. en su respectivo archivo
 * @param unos       [Unos que ha contado]
 * @param segmentos [segmentos en los que va contando]
 */
void Comb::agregarUnos(int unos,int segmentos)
{
        switch(segmentos)
        {
                case 20:
```

```
                        segmentarVeinte(unos);
                break;
                case 60:
                        segnmentarSesenta(unos);
                break;
                case 100:
                        segmentarCien(unos);
                break;
                default:
                        segmentarOtro(unos);
                break;
        }
}
```

Comb.hpp

```
/**
 *  Author : Colin Heredia Luis Antonio
 *  Version : 1.0
 *  Clase de Comb contiende las funciones y variables a usar en esta practica
 */
class Comb{
public:
        // cosntructor
        explicit Comb();
        ~Comb();
        void newComb(int);
        int calcularX(int); // funcion para devolver el numero de combinaciones.
        int genValor() const;

        // Funcoines para las combinaciones
        void startCombinacion();
        void startCombinacion27();
        char binarioResiduo(int);

        // lectura y escritura de archivos
        void escribeUniverso(std::string);
        void escribeCombinacion(std::string);
        void escribeUniverso27(std::string);
        void segmentarVeinte(int);
        void segnmentarSesenta(int);
        void segmentarCien(int);
        void segmentarOtro(int);

        void leerUnos(int,std::string);
        void agregarUnos(int,int);
        void contarUnos(std::string,int);
        // fin de funciones para la combinacion

private:
        char binario[2];         // valores posibles
        std::string cadena; // arreglo para la cadena
        std::string auxCadena; // para la modificacion de la cadena
        int k;  // valor de k dada o generada
        int x;  // valor d ecombinaciones
};
```

```
C:\Users\colin\Documents\ESCOM\SemestreActual\TeoriaComputacional\Practicas\Practica1(Combinaciones)\C++>practica1.exe

 Teclea el numero del metodo a utilziar. Alfabeto : {1,0}
 1.- Generar numero aleatorio.
 2-. Introducir un numero manualmente.

 Opcion : 1
Valor Genreado : 6


E={e,00,01,10,11,000,010,001,011,100,110,101,111,0000,0010,0100,0110,0001,0011,0101,0111,1000,1010,1100,1110,1001,1011,1101,1111,00000,00100,00010,00110,01000,01100,01010,01110,00001,00101,
00011,00111,01001,01101,01011,01111,10000,10100,10010,10110,11000,11100,11010,11110,10001,10101,10011,10111,11001,11101,11011,11111,000000,000100,001000,001100,000010,000110,001010,001110,0
10000,010100,011000,011100,010010,010110,011010,011110,000001,000101,001001,001101,000011,000111,001011,001111,010001,010101,011001,011101,010011,010111,011011,011111,100000,100100,101000,1
01100,100010,100110,101010,101110,110000,110100,111000,111100,110010,110110,111010,111110,100001,100101,101001,101101,100011,100111,101011,101111,110001,110101,111001,111101,110011,110111,1
11011,111111,}
```

```
 Opcion : 2


 Teclea el valor de k para calcular el universo : 5
Valor Digitado : 5


E={e,00,01,10,11,000,010,001,011,100,110,101,111,0000,0010,0100,0110,0001,0011,0101,0111,1000,1010,1100,1110,1001,1011,1101,1111,00000,00100,00010,00110,01000,01100,01010,01110,00001,00101,
00011,00111,01001,01101,01011,01111,10000,10100,10010,10110,11000,11100,11010,11110,10001,10101,10011,10111,11001,11101,11011,11111,}
```

Combinaciones: Bloc de notas

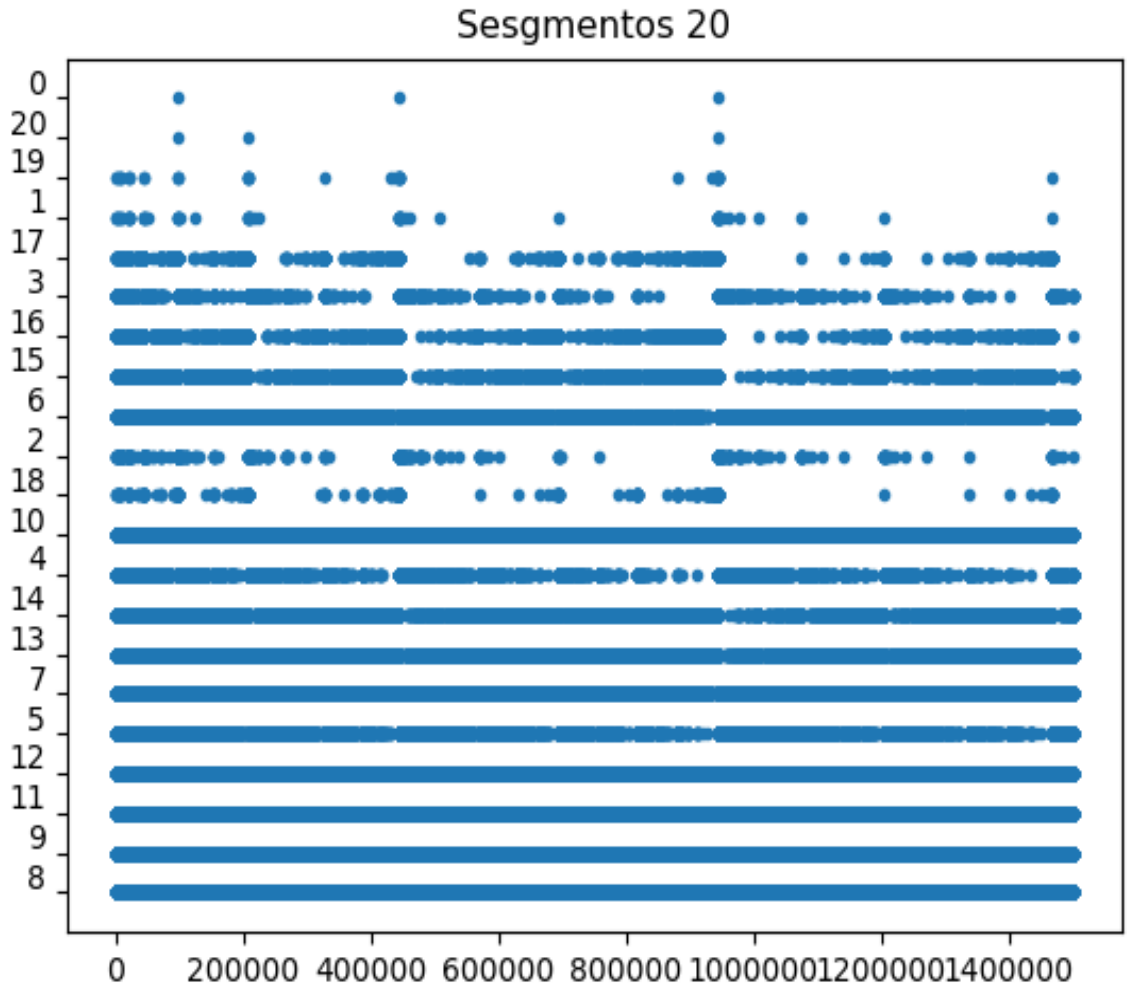
Archivo Edición Formato Ver Ayuda

```
00,01,10,11,000,010,001,011,100,110,101,111,0000,0010,0100,0110,0001,0011,0101,0111,1000,101(
```

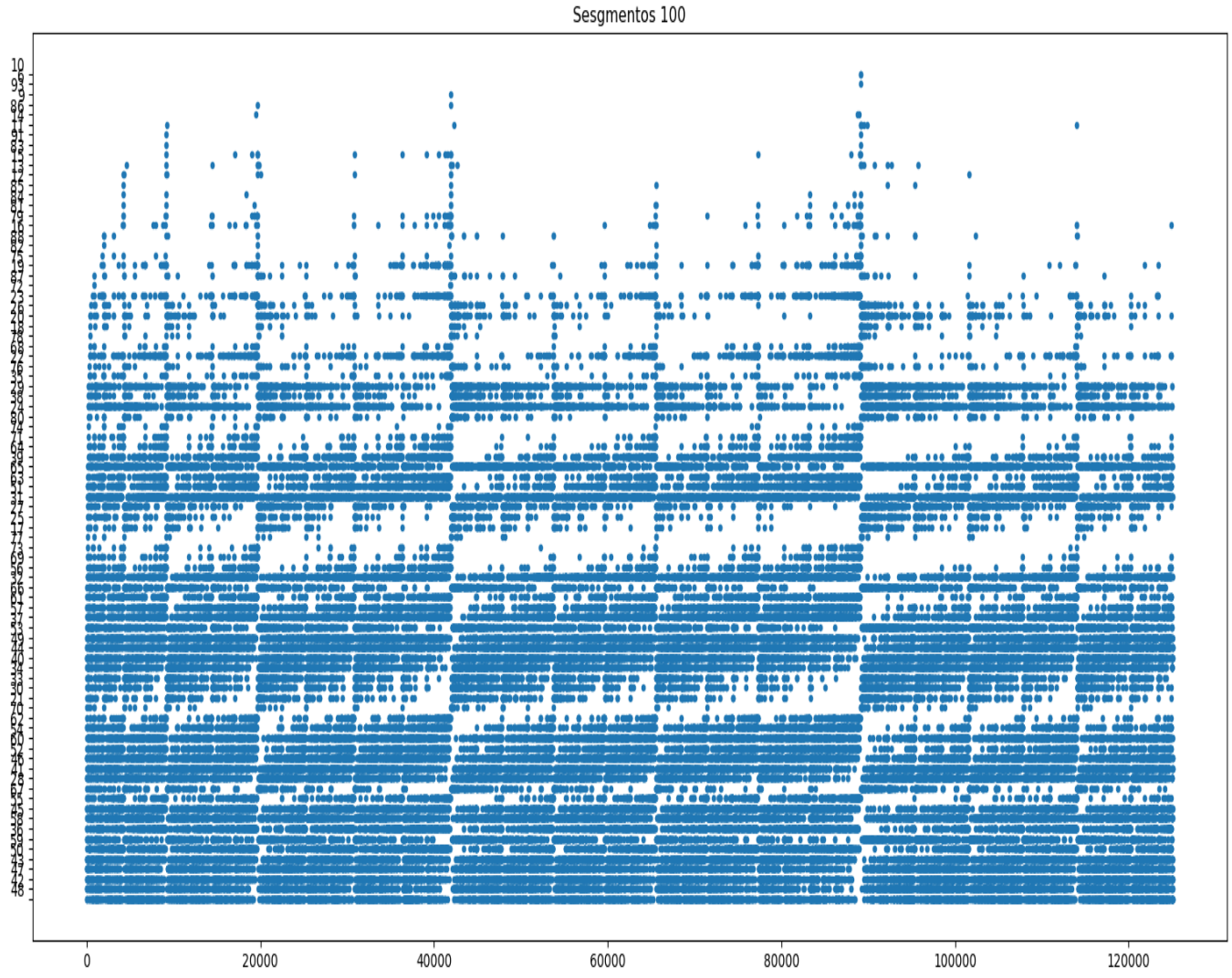
Universo27: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
0001101100001000101110011010111100000010010001100001001101010111100010101100111010011011110
```


Sesgmentos 20
0
20
19
1
17
3
16
15
6
2
18
10
4
14
13
7
5
12
11
9
8
0
200000
400000
600000
800000
1000000
1200000
1400000


Sesgmentos 60
0
25000
50000
75000
100000
125000
150000
175000



Practica 2 (Números Primo)

En esta practica se se debe dar un numero n dado por el usuario o generado.Con el obtener todos los números primos que hay de 2 hasta n. Convertirlos a binario y contar los unos

MainP2.cpp

```
/*
        Author: Colin Heredia Luis Antonio
        Versio : 1.0
        Descripcion: Encontrar los numeros primos de 0 a n, conviertiendo a binario
        y contando los unos para al final graficarlos.
*/
#include <iostream>
#include "Primos.hpp"

using namespace std;
int main()
```

```
{
        int n = 0;
        int operacion=0; // operacion que se va aplicar.
        Primos prim;
        cout << "\n Teclea el numero del metodo a utilziar. " << endl;
        cout << " 1.- Generar numero aleatorio. \n 2-. Introducir un numero
            manualmente." << endl;
        cout << "\n Opcion : ";
        cin >> operacion;

        if(operacion == 1)
        {
                // modo automatico
                n = prim.genValor();
                cout<< "\nValor Generado : "<< n <<endl;
                prim.calcularPrimos(n);

        }else
        {
                cout<<"Teclea un valor para n : ";
                cin>>n;
                // modo manual
                if(n >= 2 && n <= 1000000)
                {
                        cout<<"\nValor Digitado : " << n <<endl;
                        prim.calcularPrimos(n);
                }else
                        cout<<"El numero debe ser entero mayor a 1 y menor o igual
                            a 100,000 "<<endl;
        }

        return 0;
}
```

Primos.cpp

```
/*
        Clase Primos
        Author : Colin Heredia Luis Antonio
        Version: 1.0
*/
#include <iostream>
#include <cstdlib>      // donde se encuentra la funcion RAND y SRAND
#include <ctime>        // time para la funcion de uso de tiempo
#include <fstream>      // para los archivos
#include <iomanip> // parametros de manupulacions
#include <bits/stdc++.h> // funcion reverse.
#include "Primos.hpp"

using namespace std;

// construcores y desctructor
Primos::~Primos(){}
Primos::Primos()
{
        // constructor clase primos
```

```
        binario[0] = '0';
        binario[1] = '1';
}
/**
 * [Primos::genValor description]
 * Genera un valor distinto en un determinador rango
 * @return [regresa el numero generado]
 */
int Primos::genValor() const{
        srand(time(NULL)); // le damos la semilla a srand
        int random = (rand()%100000)+1; // generamos un numero entre 1 y 10
        return random;
}

/**
 * [Primos::escribeBinariosPrimos description]
 * Escribe en un archivo llamado BinariosPrimos los numeros primos binarios
 * @param binario [cadena que se va escribir en el carhcivo]
 */
void Primos::escribeBinariosPrimos(string binario)
{
        cadenaJunta += " Numero Binario : " + binario;
        ofstream txtPrimos;
        txtPrimos.open("./txt/BinariosPrimos.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtPrimos<<binario<<",";
        txtPrimos.close();
}
/**
 * [Primos::escribeNumerosPrimos description]
 * escribe en un archivo los numeros primos encontrados
 * a ade en la cadena para imprimir el numero primo
 * @param primos [numeor primo]
 */
void Primos::escribeNumerosPrimos(int primos)
{
        cadenaJunta += " Decimal: " + to_string(primos);
        ofstream txtPrimos;
        txtPrimos.open("./txt/NumerosPrimos.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txtPrimos<<primos<<",";
        txtPrimos.close();
}
/**
 * [Primos::escribeUnosBinarios description]
 * Escribe los unos contados dentro de un binario
 * @param unos [binario]
 */
void Primos::escribeUnosBinarios(int unos)
{
        cadenaJunta += " Numero de unos : " + to_string(unos);
        ofstream txtPrimos;
        txtPrimos.open("./txt/UnosBinarios.txt",ios::in|ios::app); // entrada y
```

```
            a adir
        // salto de linea
        txtPrimos<<unos<<endl;
        txtPrimos.close();
}


/**
 * [Primos::leerUnos description]
 * lee un arhcivo con un path y cuenta los unos dentro de ese archivo
 * manda la cadena a contarunos quien se encarga de contarlos
 * @param pathArchivo [description]
 */
void Primos::leerUnos(string pathArchivo)
{
        // abrimos el archivo y contamos los segmentos
        // abirmo el archivo y contamos los unos que hayan
        string numeros;
        int unos = 0;
        int contador = 0;
        ifstream archivo(pathArchivo);
        if(archivo.fail())
                cerr<<"Error al abrir el archivo"<<endl;
        else{
                getline(archivo,numeros);
                contarUnos(numeros);
        }
}
/**
 * [Primos::contarUnos description]
 * Funcion se encarga de contar los unos que existan dentro de la cadena
 * @param cadena [cadena a leer]
 */
void Primos::contarUnos(string cadena)
{
        int unos=0;
        for(int i = 0; i<cadena.size(); i++)
                if(cadena[i] == binario[1])
                        unos++;
        if(unos != 0)
                escribeUnosBinarios(unos);
}
/**
 * [Primos::calcularPrimos description]
 * Calcula los numeros primos dentro de un rango dado por el usuario
 * pasa por el isPrimo quien determina si es primo
 * Si es primo lo escribe en numerosPrimos y en bianrio
 * @param n [Numeor hasta n a calcular]
 */
void Primos::calcularPrimos(int n)
{
        for(int i = 2; i<= n; i++)
        {
                if(isPrimo(i))
                {
```

```
                        escribeNumerosPrimos(i);
                        escribeBinariosPrimos(conversorBinario(i));
                        imprimeValores();
                }
        }
}
/**
 * [Primos::imprimeValores description]
 * Solo imprime en pantalla los valores escritos en diferentes archivos
 */
void Primos::imprimeValores()
{
        cout<<cadenaJunta<<endl;
        cadenaJunta.clear();
}
/**
 * [Primos::isPrimo description]
 * Determina si el valor dado es un numero primo
 * @param  numero [numero a evaluar]
 * @return        [true or false si es primo o no]
 */
bool Primos::isPrimo(int numero)
{
        bool isPrimo = true;
        int contador = 2;

        while((contador<numero) && (isPrimo))
        {
                if((numero%contador)!= 0)
                        contador++;
                else
                        isPrimo = false;
        }
        return isPrimo;
}
/**
 * [Primos::conversorBinario description]
 * convierte un valor decimal dado que seria primo a un valor
 * binario equivalente. Llama la funcion contar unos para que cuente
 * cuantos unos tiene el numero primo en binario
 * @param  decimal [decimal a convertir]
 * @return         [cadena en binario del numero primo]
 */
string Primos::conversorBinario(int decimal)
{
        cadena = "";
        while(decimal > 0)
        {
                cadena += binarioResiduo(decimal);
                decimal /= 2;
        }
        // revertimos la cadena
        reverse(cadena.begin(),cadena.end());
        contarUnos(cadena);
        return cadena;
```

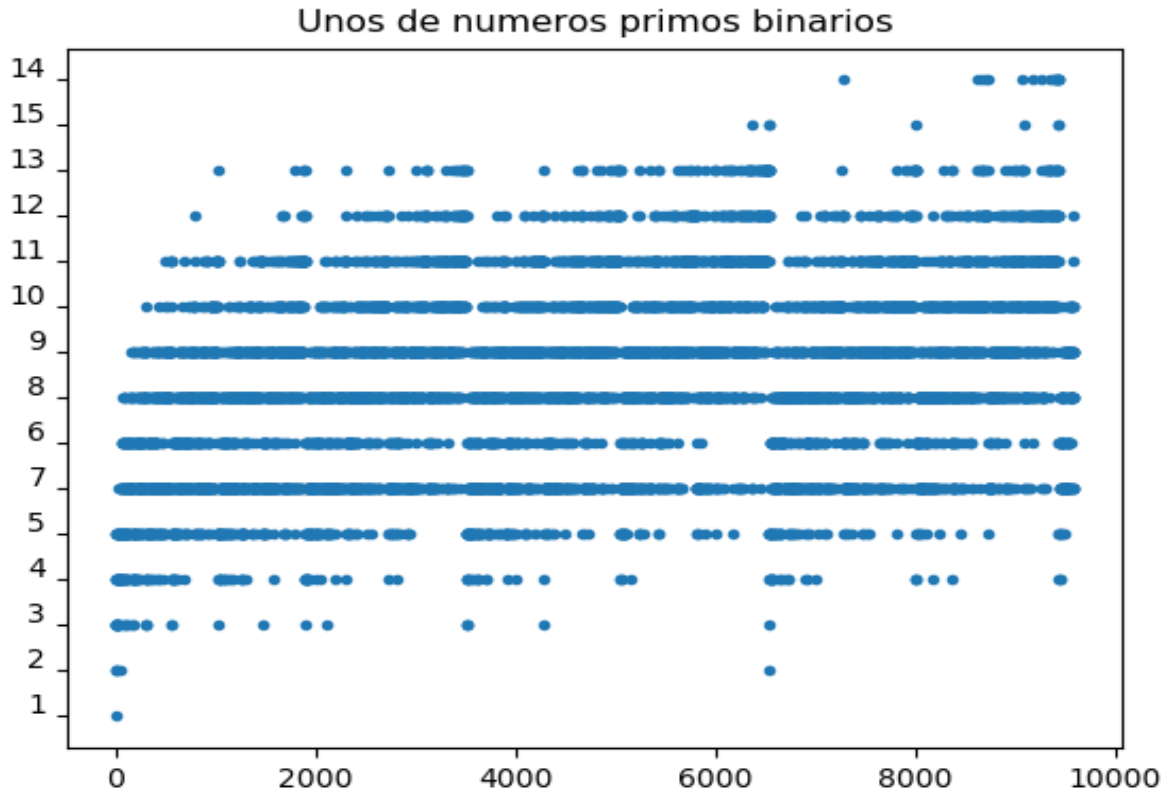
```
}
/**
 * [Primos::binarioResiduo description]
 * Devuelve el residuo para la conversion de binario
 * @param  numero [numero a convertir]
 * @return        [valor 0 o 1 del residuo]
 */
char Primos::binarioResiduo(int numero)
{
        if(!(numero == 0))
                numero%=2;
        if(numero>0)
                return binario[1];
        else
                return binario[0];
}
```

Primos.hpp

```
/*
        Clase ppara calculo de primos y escritura de archivos de texto
*/
class Primos{
public:
        explicit Primos();
        ~Primos();
        bool isPrimo(int);
        int genValor() const;
        void start();
        void escribeBinariosPrimos(std::string);
        void leerUnos(std::string);
        void contarUnos(std::string);
        void escribeUnosBinarios(int);
        void escribeNumerosPrimos(int);
        void calcularPrimos(int);
        char binarioResiduo(int);
        void imprimeValores();
        std::string conversorBinario(int);
private:
        char binario[2];
        std::string cadena;
        std::string cadenaJunta;
};
```

```
 Teclea el numero del metodo a utilziar.
 1.- Generar numero aleatorio.
 2-. Introducir un numero manualmente.

 Opcion : 1

Valor Generado : 13
 Decimal: 2 Numero de unos : 1 Numero Binario : 10
 Decimal: 3 Numero de unos : 2 Numero Binario : 11
 Decimal: 5 Numero de unos : 2 Numero Binario : 101
 Decimal: 7 Numero de unos : 3 Numero Binario : 111
 Decimal: 11 Numero de unos : 3 Numero Binario : 1011
 Decimal: 13 Numero de unos : 3 Numero Binario : 1101
```

```
 Teclea el numero del metodo a utilziar.
 1.- Generar numero aleatorio.
 2-. Introducir un numero manualmente.

 Opcion : 2
Teclea un valor para n : 50

Valor Digitado : 50
 Decimal: 2 Numero de unos : 1 Numero Binario : 10
 Decimal: 3 Numero de unos : 2 Numero Binario : 11
 Decimal: 5 Numero de unos : 2 Numero Binario : 101
 Decimal: 7 Numero de unos : 3 Numero Binario : 111
 Decimal: 11 Numero de unos : 3 Numero Binario : 1011
 Decimal: 13 Numero de unos : 3 Numero Binario : 1101
 Decimal: 17 Numero de unos : 2 Numero Binario : 10001
 Decimal: 19 Numero de unos : 3 Numero Binario : 10011
 Decimal: 23 Numero de unos : 4 Numero Binario : 10111
 Decimal: 29 Numero de unos : 4 Numero Binario : 11101
 Decimal: 31 Numero de unos : 5 Numero Binario : 11111
 Decimal: 37 Numero de unos : 3 Numero Binario : 100101
 Decimal: 41 Numero de unos : 3 Numero Binario : 101001
 Decimal: 43 Numero de unos : 4 Numero Binario : 101011
 Decimal: 47 Numero de unos : 5 Numero Binario : 101111
```



Practica 3 (Protocolo)

En esta practica se generan 100,000 cadenas de 32 bits. El cual el autómata determinista las evaluara, creando un archivo con las cadenas validas e invalidas de igual forma se mostrara el recorrido de estados. En esta practica utilice la libreria miniwin la cual se puede descargar gratis desde su pagina principal para la elaboración gráfica. para su compilación es "g++ Main3.cpp Aautomata.cpp -mwindows -o practica3

MainP3.cpp

```
/**
 *
 * Author: Colin Heredia Luis Antonio
 * Version : 1.0
 * Descripcion: Protocolo prorama que cree 100,000 cadenas de 31 bits y revise con
    un automata
 *      Si pertenece la cadena.
 *
 */

#include <iostream>
#include <cstdlib>      // donde se encuentra la funcion RAND y SRAND
#include <ctime>        // time para la funcion de uso de tiempo
#include "miniwin.h"
```

```
#include "Automata.hpp"

using namespace std;

int main(void)
{
        /* code */
        srand(time(NULL)); // le damos la semilla a srand
        Automata autoCadenas;

        return 0;
}
```

Automata.cpp

```
#include <iostream>
#include <cstdlib>      // donde se encuentra la funcion RAND y SRAND
#include <ctime>        // time para la funcion de uso de tiempo
#include <fstream>      // para los archivos
#include <cstring>
#include <sstream>
#include <iomanip> // parametros de manupulacions
#include "miniwin.h"
#include "Automata.hpp"

using namespace std;
using namespace miniwin;

/**
 *  Constructor. Se pinta el automata directamente al ser llamado
 */
Automata::Automata() {
        inicializaVariabres();
        pintarTodo();
        // genera las cadenas
        generarCadenas();
        leerArchivoMandarAutomata();
        //automata("11101111110111111101111110111111010,");
}
/**
 * [Automata::inicializaVariabres description]
 * Funcion que inicializa la funcion de transicion, los estados
 * alfabeto para el funcinamiento del automata.
 */
void Automata::inicializaVariabres()
{
        numCadena=1;
        // cadenas
        palAuxiliar = "";
        // estados
        q0 = 0; // estado inicial
        qA = q0; //posicion del estado.

        // sigma
        binario[0]='0';
```

```
        binario[1]='1';

        // Tabla de transicion.
        // primer columna
        funcionT[0][0] = 1; // [0][0] => q1
        funcionT[1][0] = 0; // [1][0] => q0
        funcionT[2][0] = 3; // [2][0] => q3
        funcionT[3][0] = 2; // [3][0] => q2
        // segunda columna
        funcionT[0][1] = 2; // [0][1] => q2
        funcionT[1][1] = 3;      // [0][1] => q3
        funcionT[2][1] = 0; // [0][1] => q0
        funcionT[3][1] = 1; // [0][1] => q1

        refresca();
}
void Automata::pintarTodo()
{
        vredimensiona(1200,600);
        // DAtos
        texto(200,0,"ESCOM - Colin Heredia Luis Antonio - Practica 3 Teoria
            computacional");
        // dibujamos los estados
        dibujarEstados();
        // dibujanod las fechas
        dibujarLinea(120,125,270,125,"0",true);
        dibujarLinea(120,170,270,170,"0",false);
        dibujarLinea(120,275,270,275,"0",true);
        dibujarLinea(120,320,270,320,"0",false);
        refresca();
        espera(1000);

        // flechas hacia arriba y abajo
        dibujarLineaArribaAbajo(80,170,80,275,"1",true);
        dibujarLineaArribaAbajo(110,170,110,275,"1",false);
        dibujarLineaArribaAbajo(280,170,280,275,"1",true);
        dibujarLineaArribaAbajo(310,170,310,275,"1",false);
        refresca();
        espera(1000);
        // sifnificado de colores
        texto(500,100,"SIGNIFICADO DE COLORES");
        texto(750,100,"SECUENCIA DE ESTADOS");
        texto(500,320,"CADENA NUMERO : "+to_string(numCadena));
        dibujarPuntosignificado(500,150,VERDE,": Estado valido");
        dibujarPuntosignificado(500,200,ROJO,": Estado invalido");
        dibujarPuntosignificado(500,250,CYAN,": Posicion actual");
        refresca();
}
/**
 * [Automata::automata description]
 * Esta funcion recorre la palabra caracter por caracter hasta encontrar
 * el , que le dice que ua termino de leer la cadena. La funcion al recorrer
 * Cada caracter lo pasa a la funcion transicion quien me determina el estado
 * al que debo devolverle a qA
 * @param  palabra [La palabra a analizar de 1 y 0 s]
```

```
 */
void Automata::automata(std::string palabra)
{
        int is = 0;
        texto(80,350,"CADENA A ANALIZAR");
        texto(80,380,palabra);
        for(int i = 0; i<=palabra.size(); i++)
        {
                if(palabra[i] == ','){
                        break;
                }

                is +=13;
                palAuxiliar = palabra[i];
                char c = palabra[i];
                qA = transicion(qA,c,is);
                switch(qA)
                {
                        case 0:
                                // dibujamos los estados
                                espera(500);
                                dibujarEstados();
                                dibujarEstadoActual("qA",100,150);
                                refresca();
                        break;
                        case 1:
                                espera(500);
                                dibujarEstados();
                                dibujarEstadoActual("qA",300,150);
                                refresca();
                        break;
                        case 2:
                                espera(500);
                                dibujarEstados();
                                dibujarEstadoActual("qA",100,300);
                                refresca();
                        break;
                        case 3:
                                espera(500);
                                dibujarEstados();
                                dibujarEstadoActual("qA",300,300);
                                refresca();
                        break;
                }
        }

        color(AMARILLO);
        if(qA != 0){
                texto(80,400,"LA CADENA ES VALIDA");
                escribeCadenaCorrecta(palabra);
                numCadena++;
        }
        else{
                texto(80,400,"LA CADENA ES INVALIDA");
                escribeCadenaIncorrecta(palabra);
```

```
                numCadena++;
        }

        refresca();
        espera(1500);
        repintar();
}
/**
 * [Automata::repintar description]
 * Funcion para borrar el buffer y volver a pintar todo
 */
void Automata::repintar()
{
        borra();
        espera(500);
        refresca();
        pintarTodo();
}
/**
 * [Automata::leerArchivoMandarAutomata description]
 * Leer las cadenas y mandarlas al automata para verificarlos
 * y decidir si son validas o no.
 */
void Automata::leerArchivoMandarAutomata()
{
        int unos = 0;
        int contador = 0;
        //ifstream archivo("./txt/cadenas32BitsPruebas.txt");
        ifstream archivo("./txt/cadenas32Bits.txt");
        if(archivo.fail())
                cerr<<"Error al abrir el archivo"<<endl;
        else{
                while(getline(archivo,cadenas,'\n'))
                        automata(cadenas);
        }
}
/**
 * [Automata::dibujarEstados description]
 * Dibyja los circulos en verde y rojo para el recorrido
 * del automata
 */
void Automata::dibujarEstados()
{
        // dibujamos los estados
        dibujarEstado("q0",100,150,false);
        dibujarEstado("q1",300,150,true);
        dibujarEstado("q2",100,300,true);
        dibujarEstado("q3",300,300,true);
}
/**
 * [Automata::transicion description]
 * Esta es la funcion de transicion recibe la letra char
 * Esa letra la convierte a string y luego a un int.
 * el cual se usa como indice junto con qA siendo el estado
 * y el indice lo cual en nuestra tabla de funcion de transicion
```

```
 * nos dira que es lo que hay que devolver.
 * @param  qA     [Estado actual]
 * @param  letra [caracter leido]
 * @param  toi    [contador para imprimir correctamente]
 * @return        [regresa el siguiente estado para qA]
 */
int Automata::transicion(int qA,char letra,int toi)
{
        int qR=0;
        int indice =0;
        string aux;
        aux.push_back(letra);
        istringstream iss (aux);
        indice = stoi(aux);
        qR = funcionT[qA][indice];
        color(AMARILLO);
        texto(800,105+toi,aux+"-->q"+to_string(qR));
        escribeEstado("q"+to_string(qR)); // escribe el estado en el archivo
        return qR;
}

/**
 * [Automata::dibujarPuntosignificado description]
 * Dibuja un circulo con una descripcion a un lado para darle signidicado al color
 * @param x       [posicion respectiva]
 * @param y       [posicion respectiva]
 * @param a       [color del circulo especificado dentro del miniwin.h]
 * @param cadena [descripcion del colo]
 */
void Automata::dibujarPuntosignificado(float x,float y,COLOR a,string cadena)
{
        color(a);
        circulo_lleno(x,y,10);
        texto(x+15,y-10,cadena);
}

/**
 * [Automata::dibujarEstado description]
 * Dibujja un circulo con el nombre del estado si es final lo pinta en verde si no
    en blanco.
 * @param estado  [String con el nombre del estado genralmente qn]
 * @param x       [Posicion en X deonde pintar]
 * @param y       [Posicion en Y donde pintar]
 * @param isFinal [Verifica si es un estado final o no]
 */
void Automata::dibujarEstado(string estado,float x,float y ,bool isFinal)
{
        color(ROJO);
        if(isFinal)
                color(VERDE);
        circulo_lleno(x,y,30);
        color(NEGRO);
        texto((x-10),(y-15),estado);
}
/**
```

```
 * [Automata::dibujarEstadoActual description]
 * Dibujja un circulo con el nombre del estado qA
 * @param estado  [String con el nombre del estado genralmente qn]
 * @param x       [Posicion en X deonde pintar]
 * @param y       [Posicion en Y donde pintar]
 */
void Automata::dibujarEstadoActual(string estado,float x,float y)
{
        color(CYAN);
        circulo_lleno(x,y,30);
        color(NEGRO);
        texto((x-10),(y-15),estado);
}

/**
 * [Automata::dibujarLinea description]
 * Dibujar una linea de un luhgar a otro y al final le agrega un > para simular la
    fecha
 * @param x_ini [Posicion inicial X]
 * @param y_ini [posicion inicial Y]
 * @param x_fin [Posicion final X]
 * @param y_fin [Posicion final Y]
 * @param caracter [caracter a colocar a la mitad de la flecha];
 * @param der_iz un boleano para detectar de que lado apunta la flecha;
 */
void Automata::dibujarLinea(float x_ini, float y_ini, float x_fin, float y_fin,
   string caracter,bool der_izq)
{
        color(BLANCO);
        linea(x_ini,y_ini,x_fin,y_fin);

        if(der_izq)
                texto(x_fin,(y_fin-8),">");
        else
                texto(x_ini,(y_ini-8),"<");

        color(BLANCO);
        texto(((x_ini+x_fin)/2),((y_ini+y_fin)/2),caracter);
}

/**
 * [Automata::dibujarLineaArribaAbajo description]
 * Dibujar una linea de un luhgar a otro y al final le agrega un > para simular la
    fecha
 * @param x_ini [Posicion inicial X]
 * @param y_ini [posicion inicial Y]
 * @param x_fin [Posicion final X]
 * @param y_fin [Posicion final Y]
 * @param caracter [caracter a colocar a la mitad de la flecha];
 * @param arr_abj un boleano para detectar de que lado apunta la flecha;
 */
void Automata::dibujarLineaArribaAbajo(float x_ini, float y_ini, float x_fin, float
    y_fin,string caracter,bool arr_abj)
{
        color(BLANCO);
```

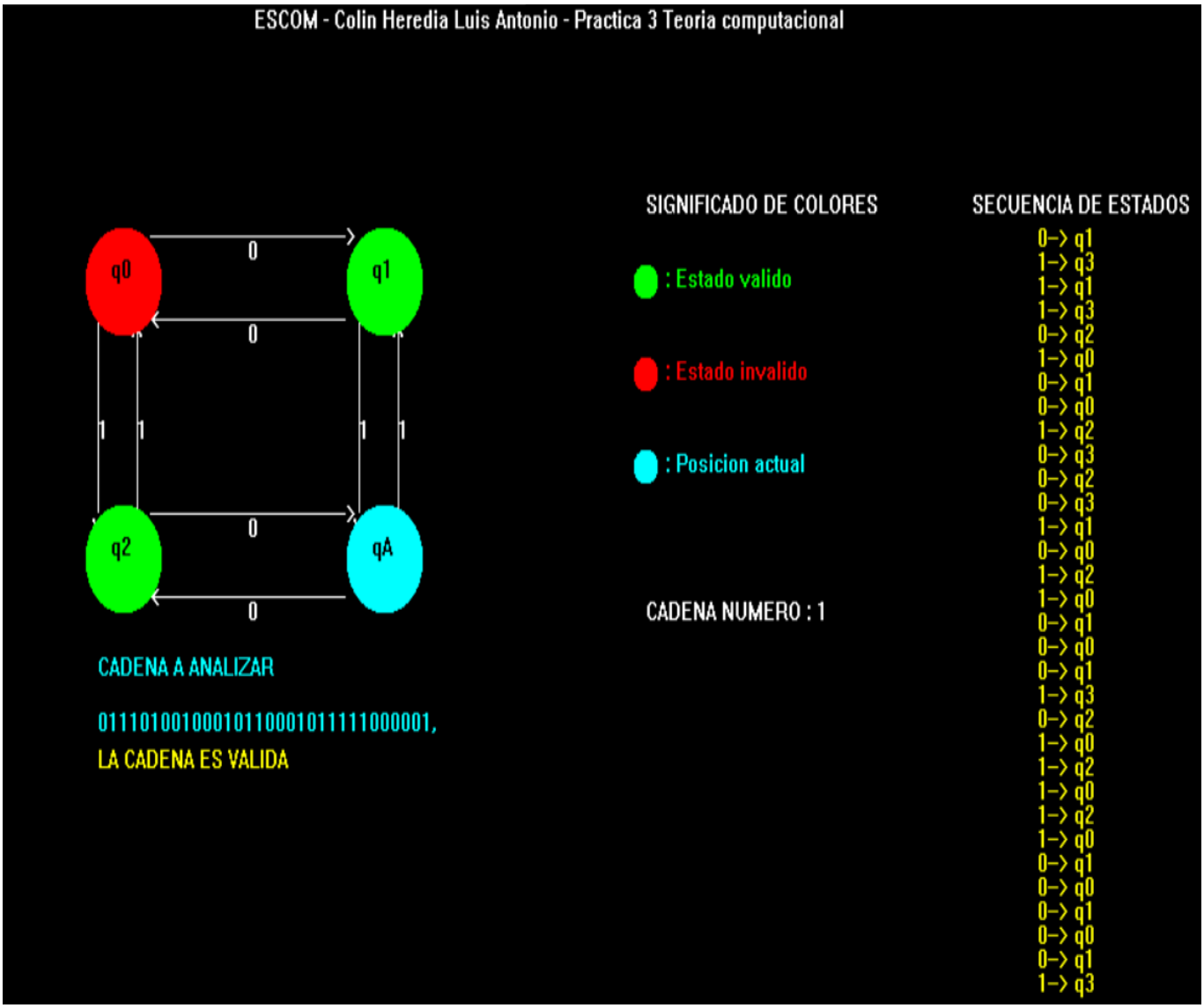
```
        linea(x_ini,y_ini,x_fin,y_fin);

        if(arr_abj)
                texto((x_fin-4),y_fin,"V");
        else
                texto((x_ini-4),y_ini-3,"A");

        color(BLANCO);
        texto(((x_ini+x_fin)/2),((y_ini+y_fin)/2),caracter);
}
/**
 * [Automata::genValor description]
 * Genera un valor aleatorio de 0 y 1
 * @return [devuelve el valor generado]
 */
char Automata::genValor() const{

        int random = (rand()%2);

        if(random != 1)
                return binario[0];
        return binario[1];
}
/**
 * [Automata::escribeCadena description]
 * Escribe las cadenas generadas en un archivo dividiendolas por una ,
 * @param cadena [cadena a escribir en el archivo]
 */
void Automata::escribeCadena(string cadena)
{
        ofstream txt;
        txt.open("./txt/cadenas32Bits.txt",ios::in|ios::app); // entrada y a adir
        // salto de linea
        txt<<cadena<<","<<endl;
        txt.close();
}
/**
 * [Automata::escribeEstado description]
 * Escribe en un archivo llamado Estados.txt los estados
 * donde se va recorriendo la transicion del automata
 * @param q [description]
 */
void Automata::escribeEstado(string q)
{
        ofstream txt;
        txt.open("./txt/Estados.txt",ios::in|ios::app); // entrada y a adir
        // salto de linea
        txt<<q<<",";
        txt.close();
}
/**
 * [Automata::escribeCadenaCorrecta description]
 * Escribira las cadenas validads
 * @param cadena [cadena a escribir]
 */
```

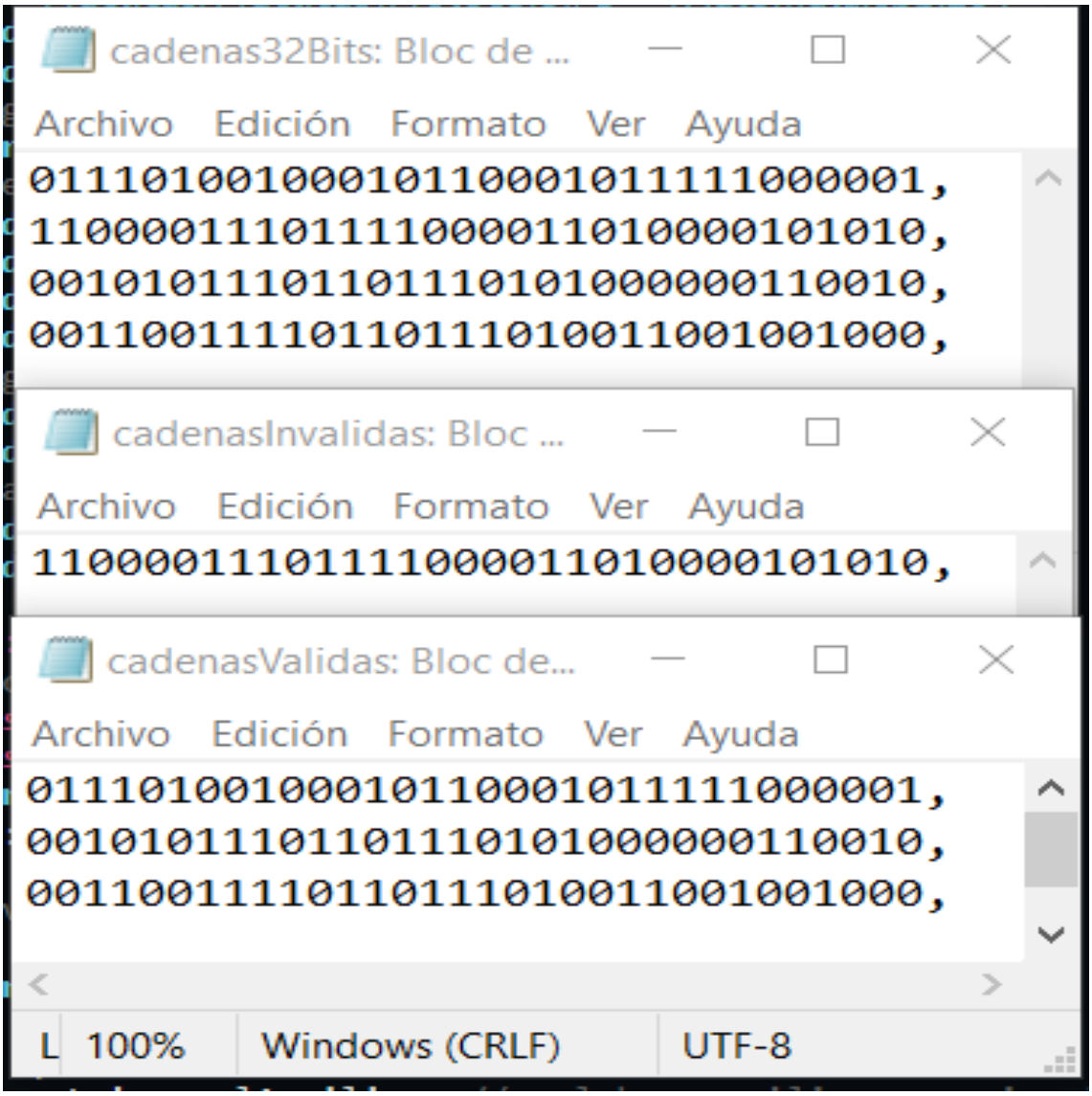
```
void Automata::escribeCadenaCorrecta(string cadena)
{
        ofstream txt;
        txt.open("./txt/cadenasValidas.txt",ios::in|ios::app); // entrada y a adir
        // salto de linea
        txt<<cadena<<endl;
        txt.close();
}
/**
 * [Automata::escribeCadenaIncorrecta description]
 * Escribe la cadenas incorrectas
 * @param cadena [cadena a escribir]
 */
void Automata::escribeCadenaIncorrecta(string cadena)
{
        ofstream txt;
        txt.open("./txt/cadenasInvalidas.txt",ios::in|ios::app); // entrada y
            a adir
        // salto de linea
        txt<<cadena<<endl;
        txt.close();
}
/**
 *  Genera las 100000 cadenas usando la variable constante de la clase
 *  seran 100000 cadenas con 32 bits
 */
void Automata::generarCadenas()
{
        this->cadena = "";
        for(int j=0; j<=MAX; j++){
                for(int i = 0; i<BITS_NUM; i++)
                        cadenaCeros[i] = genValor();

                cadenaCeros[32] = '\0';
                cadena = cadenaCeros;
                escribeCadena(cadena);
        }

}
/**
 *  Destructor de la clase Automata
 */
Automata::~Automata() {}
```

Automata.hpp

```
class Automata {
public:

        explicit Automata();
        ~Automata();
        // dibujar el dise o
        void pintarTodo();
        void dibujarEstado(std::string,float,float,bool);
        void dibujarLinea(float, float, float, float,std::string,bool);
```

```
        void dibujarLineaArribaAbajo(float, float, float, float,std::string,bool);
        void dibujarEstados();
        void dibujarPuntosignificado(float,float,miniwin::COLOR,std::string);
        void dibujarEstadoActual(std::string,float,float);
        void repintar();
        // generar valores
        char genValor() const;
        // escribir la cadenas en archivos
        void escribeCadena(std::string);
        void escribeCadenaCorrecta(std::string);
        void escribeCadenaIncorrecta(std::string);
        void escribeEstado(std::string);
        // generar las 100,000 cadenas de 32 bits
        void generarCadenas();
        void leerArchivoMandarAutomata();
        // automata
        void inicializaVariabres();
        void automata(std::string);
        int transicion(int,char,int);
private:
        // cienmil cadenas de 32 bits
        const int MAX = 100000;
        const int BITS_NUM = 33;
        char cadenaCeros[33];
        std::string cadena;
        int numCadena;
        // variables del automata
        int funcionT[4][2];
        char binario[2]; // esta seria sigma
        int qA;  // resultado q0 q1 , q2 , q3 solo el indice
        int q0; // estado incial
        std::string palAuxiliar; // palabra auxiliar para imprimir
        std::string cadenas; // almacenar las cadeas que leeramos en el archivo
};
```


ESCOM - Colin Heredia Luis Antonio - Practica 3 Teoria computacional
q0
q1
q2
qA
0
0
1
1
1
1
0
0
SIGNIFICADO DE COLORES
: Estado valido
: Estado invalido
: Posicion actual
CADENA NUMERO : 1
CADENA A ANALIZAR
01110100100010110001011111000001,
LA CADENA ES VALIDA
SECUENCIA DE ESTADOS
0-> q1
1-> q3
1-> q1
1-> q3
0-> q2
1-> q0
0-> q1
0-> q0
1-> q2
0-> q3
0-> q2
0-> q3
1-> q1
0-> q0
1-> q2
1-> q0
0-> q1
0-> q0
0-> q1
1-> q3
0-> q2
1-> q0
1-> q2
1-> q0
1-> q2
1-> q0
0-> q1
0-> q0
0-> q1
0-> q0
0-> q1
1-> q3


cadenas32Bits: Bloc de ...
Archivo Edición Formato Ver Ayuda
01110100100010110001011111000001,
11000011101111000011010000101010,
00101011101101110101000000110010,
00110011110110111010011001001000,
cadenasInvalidas: Bloc ...
Archivo Edición Formato Ver Ayuda
11000011101111000011010000101010,
cadenasValidas: Bloc de...
Archivo Edición Formato Ver Ayuda
01110100100010110001011111000001,
00101011101101110101000000110010,
00110011110110111010011001001000,
L 100% Windows (CRLF) UTF-8