Práctica 3

IBERO
Ciudad de México • Tijuana

Código Morse (Máquina de Estados)

Integrantes del equipo

César Mauricio Arellano Velásquez
Allan Jair Escamilla Hernández
Raúl González Portillo

Profesor

César Arturo Ángeles Ruiz

Materia

Taller de Desarrollo de Aplicaciones

Introducción:

El código Morse, es un código o sistema de comunicación que permite la comunicación telegráfica a través de la transmisión de impulsos eléctricos de longitudes diversas o por medios visuales, como luz, sonoros o mecánicos. Este código consta de una serie de puntos, rayas y espacios, que al ser combinados entre sí pueden formar palabras, números y otros símbolos.

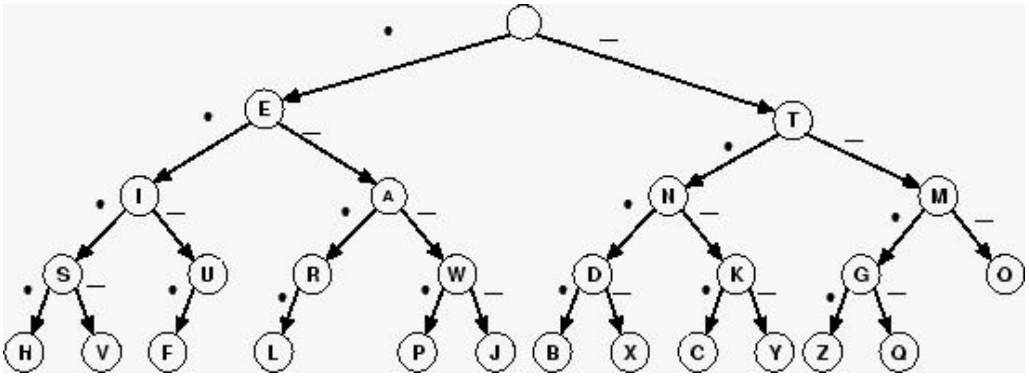
Fue desarrollado por Alfred Vail mientras colaboraba en 1835 con Samuel Morse en la invención del telégrafo eléctrico. Vail creó un método según el cual cada letra o número era transmitido de forma individual con un código consistente en rayas y puntos, es decir, señales telegráficas que se diferencian en el tiempo de duración de la señal activa. La duración del punto es la mínima posible. Una raya tiene una duración de aproximadamente tres veces la del punto. Entre cada par de Símbolos de una misma letra existe una ausencia de señal con duración aproximada a la de un punto. Entre las letras de una misma palabra, la ausencia es de aproximadamente tres puntos.

Para la separación de palabras transmitidas el tiempo es de aproximadamente tres veces el de la raya. Morse reconoció la idoneidad de este sistema y lo patentó junto con el Telégrafo eléctrico. Fue conocido como American Morse Code y fue utilizado en la primera transmisión por Telégrafo. En sus comienzos, el alfabeto Morse se empleó en las líneas telegráficas mediante los tendidos de cable que se fueron instalando. Más tarde, se utilizó también en las transmisiones por radio, sobre todo en el mar y en el aire, hasta que surgieron las emisoras y los receptores de radiodifusión mediante voz.

Código Morse, para el alfabeto inglés.

A ●−	J ●−−−	S ●●●
B −●●●	K −●−	T −
C −●−●	L ●−●●	U ●●−
D −●●	M −−	V ●●●−
E ●	N −●	W ●−−
F ●●−●	O −−−	X −●●−
G −−●	P ●−−●	Y −●−−
H ●●●●	Q −−●−	Z −−●●
I ●●	R ●−●	

Árbol binario.

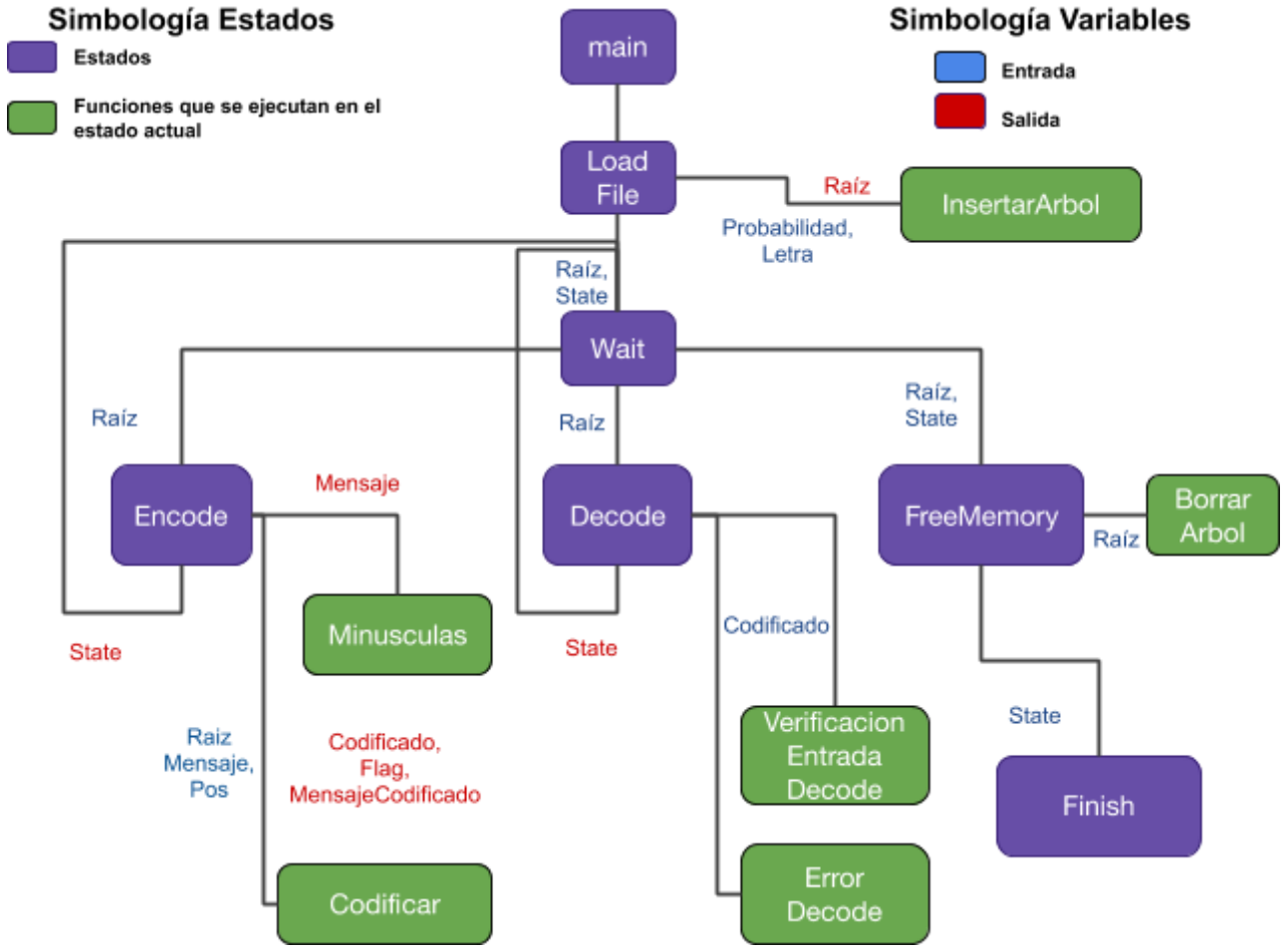


Objetivos:

- Comprender el funcionamiento de las máquinas de estados y los apuntadores a funciones a grandes rasgos.
- Codificar y decodificar mensajes a través de la lógica del código Morse y árboles binarios.

Análisis

Diagrama IPO



Entradas Procesos y Salidas

Entradas

Nombre	Descripción
Raíz	Almacena la dirección del nodo Raíz.

State	Determina el siguiente estado de la máquina.
Mensaje	Caracter particular de la cadena se piensa codificar. / Cadena a convertir en letras minúsculas.
Pos	Mantiene la posición actual del arreglo.
Codificado	Cadena a verificar.
Probabilidad	Almacena la probabilidad de aparición de la letra.
Letra	Caracter a insertar en el árbol binario.

Estados / Procesos

Nombre	**Descripción**
LoadFile	Estado que lee los datos de un archivo para generar el árbol binario que nos permitirá codificar y decodificar los mensajes.
InsertarArbol	Función que insertar en el árbol binario las letras y sus probabilidades.
Wait	Estado a la que siempre se regresa, después de realizar algún proceso, funciona como estado de espera.
Encode	Estado en donde se codifica el mensaje que usuario ingresa.
Minusculas	Función que se manda a llamar para convertir todo el mensaje a minúsculas.
Codificar	Función que se manda a llamar para realizar el proceso de codificación de un mensaje determinado por el usuario.
Decode	Estado en donde se decodifica el mensaje que usuario ingresa.
VerificacionEntradaDecode	Función que verifica la entrada de la cadena en la opción de decodificar.
ErrorDecode	Función que imprime un mensaje de error.

FreeMemory	Estado en donde se libera memoria del árbol binario.
BorrarArbol	Función que libera memoria del árbol binario.
Finish	Estado que da fin al programa.

Salidas

Nombre	**Descripción**
Raiz	Actualiza la dirección de memoria del nodo raíz del árbol binario en determinadas funciones como **LoadFile, InsertarArbol, FreeMemory.**
Codificado	Guarda un caracter codificado.
Flag	Retorna una bandera en caso de que algo haya salido mal.
MensajeCodificado	Almacena el mensaje ya codificado.

Código
Archivo state_machine.h

```
/*
    * @author: César Mauricio Arellano Velásquez, Allan Jair Escamilla Hernández, Raúl González Portillo.
    * @date:   19/Noviembre/2019
    * @file:   state_machine.h
    * @brief:  Motor de la máquina de estados.
    */
    #ifndef state_machine_h
    #define state_machine_h


    // Incluyendo las bibliotecas
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <unistd.h>
```

```c
#include <ctype.h>


// Enum de los estados de la maquina de estados
typedef enum States{
   LOAD_FILE,
   WAIT,
   ENCODE,
   DECODE,
   FREE_MEMORY,
   FINISH,
   STATE_NUM
} States;


//Estructura árbol binario.
typedef struct DefArbol{
   double Probabilidad;
   char Letra;
   struct DefArbol *izq, *der;
} TipoNodo;


// Estructura que contendra etiqueta y funcion
typedef struct StateMachine{
   States State;
   void (*process)(States* State, TipoNodo** Raiz);
}StateMachine;



// Estructuras de las funciones
int VerificacionEntradaDecode(char Codificado[]);
void InsertarArbol(TipoNodo **Raiz, double Probabilidad, char Letra);
void BorrarArbol(TipoNodo *Raiz);
void Codificar(char Mensaje, char Codificado[], int Pos, TipoNodo *Raiz, int
*Flag, char MensajeCodificado[]);
void Minusculas(char Mensaje[]);
```

```
// Prototipos de las funciones ejecutables de la maquina de estados
void LoadFile(States *State, TipoNodo** Raiz);
void Wait(States* State, TipoNodo** Raiz);
void Encode(States* State, TipoNodo** Raiz);
void Decode(States* State, TipoNodo** Raiz);
void FreeMemory(States* State, TipoNodo** Raiz);
void Finish(States* State, TipoNodo** Raiz);

States MachineInitialiser(void); // Inicializador de la maquina de estados
#endif // !state_machine_h
```

Archivo main.c

```
/
*
    * @author: César Mauricio Arellano Velásquez, Allan Jair Escamilla Hernández,
    Raúl González Portillo.
    * @date:   19/Noviembre/2019
    * @file:   main.c
    * @brief:  Implementación de máquinas de estados y árboles binarios para poder
    codificar y decodificar mensajes en código morse.
    */
    // Incluyendo el motor de la máquina de estados.
    #include "state_machine.h"

    int main(void){
       // Inicializacion de variable de estado al estado inicial.
       static States MachineState = LOAD_FILE;
       TipoNodo* Raiz = NULL;
       // Instancia de pares {Label,funcion} de la maquina de estados.
       StateMachine Machine[] = {
           {LOAD_FILE, LoadFile},
           {WAIT, Wait},
           {ENCODE, Encode},
           {DECODE, Decode},
```

```
        {FREE_MEMORY, FreeMemory},
        {FINISH, Finish}
    };

    for(;;){
        if(MachineState > STATE_NUM){
            puts("Error");
            return 0;
        }else{
            // Manda Ejecutar la funcion machine.process() correspondiente a la
etiqueta MachineState.
                (*Machine[MachineState].process)(&MachineState, &Raiz);
        }
    }

    return 0;
}
/* * Estado que lee los datos de un archivo para generar el árbol binario que nos
permitirá codificar y decodificar los mensajes.
  * @param States* State Determina el siguiente estado de la máquina.
  * @param TipoNodo **Raiz Apuntador donde se generará el árbol binario.
*/
void LoadFile(States* State, TipoNodo** Raiz){
    printf("Cargando archivo...");
    FILE *Archivo;
    char Renglon[100], Letra;
    double Probabilidad;
    Archivo = fopen("ordenletras.txt", "rt");
    if (Archivo == NULL){
        printf("El archivo no existe\n");
        printf("Saliendo del programa...");
        exit(0);
    }else{
        while (fgets(Renglon, 50, Archivo) != NULL){
```

```
            Renglon[strlen(Renglon)] = '\0';
            sscanf(Renglon, "%c, %lf", &Letra, &Probabilidad);
            InsertarArbol(Raiz, Probabilidad, Letra);
        }
    }
    fclose(Archivo);
    *State = WAIT;
    printf("Archivo cargado!\n");
}
/* * Estado a la que siempre se regresa, después de realizar algún proceso,
funciona como estado de espera.
  * @param States* State Determina el siguiente estado de la máquina.
  * @param TipoNodo **Raiz mantiene el registro del árbol binario.
*/
void Wait(States* State, TipoNodo** Raiz){
    int Opcion;
    printf("(1) Codificar mensaje.\n(2) Decodificar mensaje.\n(3) Salir.\nIngresar
opción-> ");
    //__fpurge(stdin);
    scanf(" %d", &Opcion);
    switch(Opcion){
        case 1:
            *State = ENCODE;
            break;
        case 2:
            *State = DECODE;
            break;
        case 3:
            *State = FREE_MEMORY;
            break;
        default:
            printf("Esta opción no existe!");
    }
}
/* * Estado en donde se codifica el mensaje que usuario ingresa.
```

```
  * @param States* State Determina el siguiente estado de la máquina.
  * @param TipoNodo **Raiz mantiene el registro del árbol binario.
*/
void Encode(States* State, TipoNodo** Raiz){
   char Mensaje[150], Codificado[150], MensajeCodificado[150];
   int i = 0, Flag = 0;
   printf("Ingresar frase que desea codificar a morse-> ");
   scanf(" %[^\n]", Mensaje);
   Minusculas(Mensaje);
   printf("Codificando mensaje...");
   puts("Mensaje codificado con éxito!\n");
   printf("\nEl mensaje codificado es: ");
   while(Mensaje[i] != '\0'){
       Codificar(Mensaje[i], Codificado, 0, *Raiz, &Flag, MensajeCodificado);
       Flag = 0;
       printf("%s ", MensajeCodificado);
       Codificado[0] = '\0';
       i++;
   }
   printf("\n");
   *State = WAIT;
}
/* * Función que se manda a llamar para realizar el proceso de codificación de un mensaje determinado por el usuario.
  * @param char Mensaje Caracter particular de la cadena se piensa codificar.
  * @param char Codificado[] Guarda un caracter codificado.
  * @param int Pos Mantiene la posición actual del arreglo.
  * @param int *Flag Retorna una bandera en caso de que algo haya salido mal.
  * @param char MensajeCodificado[] Almacena el mensaje ya codificado.
  * @param TipoNodo *Raiz Motor de búsqueda para codificar.
*/
void Codificar(char Mensaje, char Codificado[], int Pos, TipoNodo* Raiz, int* Flag, char MensajeCodificado[]){
   if (Mensaje == Raiz->Letra){
       *Flag = 1;
```

```
        strcpy(MensajeCodificado, Codificado);
    }
    if(Raiz != NULL && Raiz->Letra != Mensaje && *Flag == 0){
        if(Raiz->izq != NULL){
            Codificado[Pos] = '.';
            Codificado[Pos + 1] = '\0';
            Codificar(Mensaje, Codificado, Pos + 1, Raiz->izq, Flag,
MensajeCodificado);
        }
        if(Raiz->der != NULL){
            Codificado[Pos] = '-';
            Codificado[Pos + 1] = '\0';
            Codificar(Mensaje, Codificado, Pos + 1, Raiz->der, Flag,
MensajeCodificado);
        }
    }
}
/* * Función que se manda a llamar para convertir todo el mensaje a minusculas.
  * @param char Mensaje[] Cadena a convertir en letras minúsculas.
*/
void Minusculas(char Mensaje[]){
    int i = 0;
    while (Mensaje[i] != '\0'){
        Mensaje[i] = tolower(Mensaje[i]);
        i++;
    }
}
/* * Función que verifica la entrada de la cadena en la opción de decodificar.
  * @param char Mensaje[] Cadena a verificar.
*/
int VerificacionEntradaDecode(char Codificado[]) {
 for (int i = 0; i < strlen(Codificado); i++) {
    if(Codificado[i] != '.' && Codificado[i] != '-' && Codificado[i] != ' ')
      return 1;
 }
```

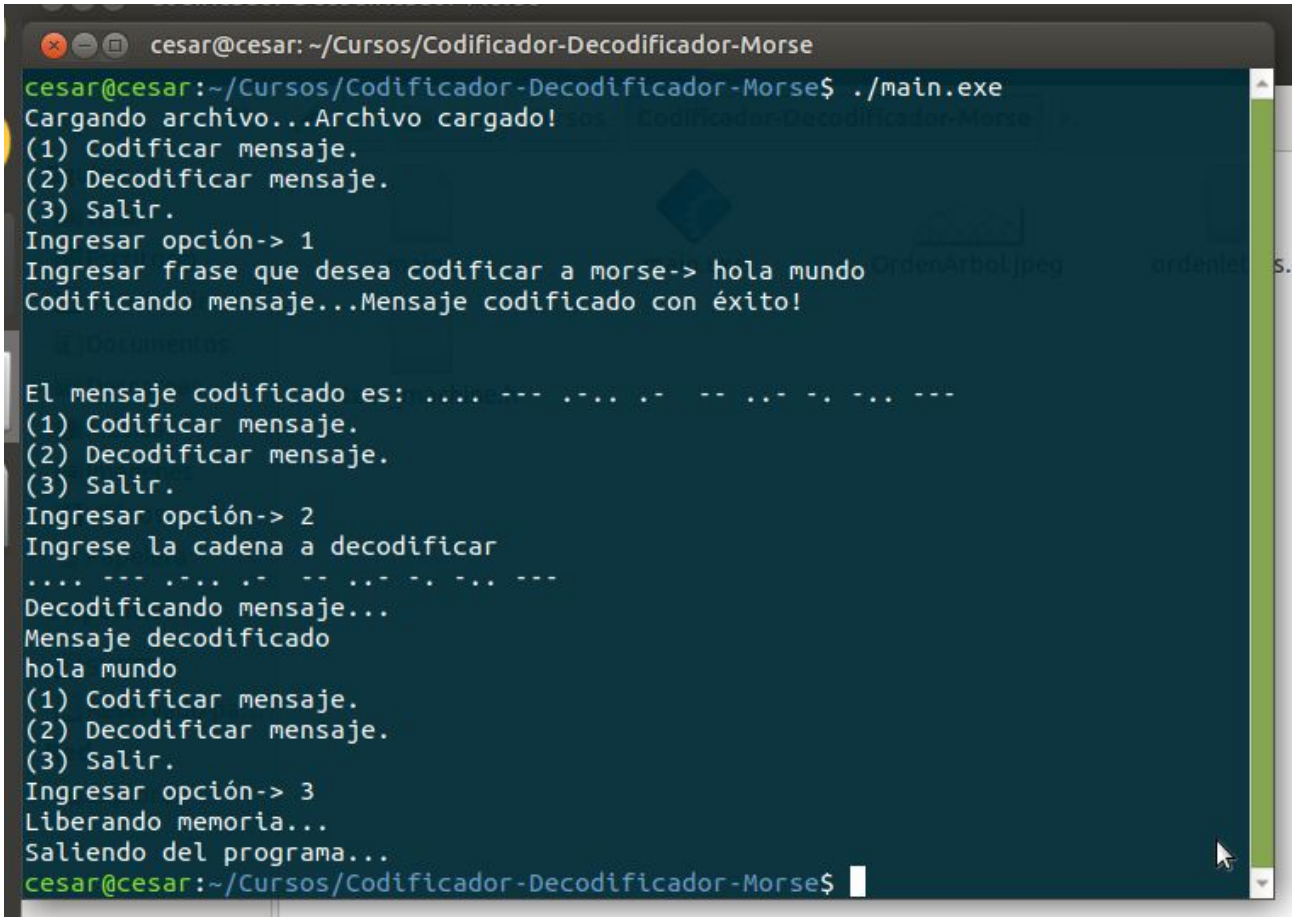
```
 return 0;
}


/* * Función que imprime un mensaje de error.*/
void ErrorDecode() {
 printf ("Revise que el mensaje está correctamente codificado\n");
 return;
}
/* * Estado en donde se decodifica el mensaje que usuario ingresa.
  * @param States* State Determina el siguiente estado de la máquina.
  * @param TipoNodo **Raiz mantiene el registro del árbol binario.
*/
void Decode(States* State, TipoNodo** Raiz) {
 int i = 0, j = 0;
 char Codificado[150], Decodificado[150];
 TipoNodo *Temp;
 printf ("Ingrese la cadena a decodificar\n");
 scanf (" %[^\n]", Codificado);
 printf("Decodificando mensaje...\n");
 if(VerificacionEntradaDecode(Codificado))
   printf ("No parece que el mensaje esté codificado\n");
 else
 {
   while (Codificado[i] != '\0')
   {
     if ((Codificado [i - 1] == ' ' && Codificado [i] != ' ') || (i == 0))
     {
       Temp = *Raiz;
       for (; Codificado[i] != ' ' && Codificado[i] != '\0'; i++)
       {
         if (Codificado[i] == '.')
         {
           if (Temp -> izq == NULL)
             ErrorDecode;
```

```
            Temp = Temp -> izq;
          }
          if (Codificado[i] == '-')
          {
            if (Temp -> der == NULL)
              ErrorDecode;
            Temp = Temp -> der;
          }
        }
        Decodificado [j] = Temp -> Letra;
        j++;
      }
      else
        if (Codificado [i - 1] == ' ')
        {
          Decodificado [j] = ' ';
          j++;
        }
      i++;
    }
    Decodificado [j] = '\0';
    printf("Mensaje decodificado\n");
    puts(Decodificado);
  }
  *State = WAIT;
}
/* * Estado en donde se libera memoria del árbol binario.
   * @param States* State Determina el siguiente estado de la máquina.
   * @param TipoNodo **Raiz mantiene el registro del árbol binario.
*/
void FreeMemory(States* State, TipoNodo** Raiz){
    printf("Liberando memoria...\n");
    BorrarArbol(*Raiz);
    *State = FINISH;
```

```
}
/* * Estado que da fin al programa.
  * @param States* State Determina el siguiente estado de la máquina.
  * @param TipoNodo **Raiz mantiene el registro del árbol binario.
*/
void Finish(States* State, TipoNodo** Raiz){
   printf("Saliendo del programa...\n");
   exit(0);
}
/* * Función que insertar en el árbol binario las letras y sus probabilidades.
  * @param TipoNodo **Raiz Almacena orden de las letras del árbol binario.
  * @param double Probabilidad Almacena la probabilidad de aparición de la letra.
  * @param char Letra caracter a insertar en el árbol binario.
*/
void InsertarArbol(TipoNodo **Raiz, double Probabilidad, char Letra){
   TipoNodo *Avanza, *Nuevo;
   Avanza = *Raiz;
   if ((Nuevo = (TipoNodo *)malloc(sizeof(TipoNodo))) == NULL){
       printf("No hay memoria\n");
       exit(0);
   }
   Nuevo->Probabilidad = Probabilidad;
   Nuevo->Letra = Letra;
   Nuevo->izq = NULL;
   Nuevo->der = NULL;
   while (Avanza != NULL){
       if (Probabilidad > Avanza->Probabilidad){ // muevete a la derecha
           if (Avanza->der != NULL)
               Avanza = Avanza->der;
           else{
               Avanza->der = Nuevo;
               return;
           }
       }
```

```
        if (Probabilidad <= Avanza->Probabilidad){ //muevete a la izquierda
            if (Avanza->izq != NULL)
                Avanza = Avanza->izq;
            else{
                Avanza->izq = Nuevo;
                return;
            }
        }
    }
    Avanza = Nuevo;
    *Raiz = Avanza;
}
/* * Función que libera memoria del árbol binario.
  * @param TipoNodo **Raiz Apuntador del árbol binario a liberar memoria.
*/
void BorrarArbol(TipoNodo* Raiz){
    if (Raiz != NULL){
        BorrarArbol(Raiz->izq);
        BorrarArbol(Raiz->der);
        free(Raiz);
    }
}
```

Ejecución del programa



Conclusión:

Gracias a esta práctica comprendimos la importancia de la arquitectura de las máquinas de estados, cuando se pretende desarrollar modularmente y eficientemente software, también pudimos volver a retomar temas importantes como lo es la recursividad, árboles binarios y apuntadores a funciones que nos permitió realizar un algoritmo más óptimo para codificar y decodificar los mensajes.