

Algoritmo generador de texto markoviano para el idioma inglés básico

1.0

Generado por Doxygen 1.8.2

Noviembre de 2012

Índice

1 Índice de estructura de datos	1
1.1 Estructura de datos	1
2 Índice de archivos	1
2.1 Lista de archivos	1
3 Documentación de las estructuras de datos	1
3.1 Referencia de la Estructura registro	2
3.1.1 Descripción detallada	2
3.1.2 Documentación de los campos	2
4 Documentación de archivos	2
4.1 Referencia del Archivo main.c	2
4.1.1 Descripción detallada	3
4.1.2 Documentación de los 'typedefs'	4
4.1.3 Documentación de las funciones	4

Índice 12

1. Índice de estructura de datos

1.1. Estructura de datos

Lista de estructuras con una breve descripción:

registro	
Estructura que almacena palabras con un índice	2

2. Índice de archivos

2.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

main.c	
Algoritmo generador de texto markoviano para el idioma inglés básico	2

3. Documentación de las estructuras de datos

3.1. Referencia de la Estructura registro

Estructura que almacena palabras con un índice.

Campos de datos

- int [indice](#)
- char [palabra](#) [30]

3.1.1. Descripción detallada

Estructura que almacena palabras con un índice.

Con la estructura se mantiene el control de las palabras agregándoles un índice. El valor asignado es utilizado para aumentar el número de apariciones de cada palabra en la matriz de probabilidad. Cada palabra puede tener una longitud de hasta 30 caracteres.

3.1.2. Documentación de los campos

3.1.2.1. [indice](#)

Un número desde 0 hasta n, para localizar palabras y mantener un registro de apariciones en la matriz de probabilidad. El valor de n depende de la cantidad de palabras que se anexan a la lista.

3.1.2.2. [palabra](#)

Almacena una palabra de hasta 30 caracteres.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [main.c](#)

4. Documentación de archivos

4.1. Referencia del Archivo [main.c](#)

Algoritmo generador de texto markoviano para el idioma inglés básico.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
```

Estructuras de datos

- struct [registro](#)

Estructura que almacena palabras con un índice.

'typedefs'

- typedef int(* [comparar](#))(const void *, const void *)
Prototipo para buscar y comprar palabras con las funciones qsort y bsearch.

Funciones

- struct [registro verificapalabra](#) (struct [registro](#) buscado)
Se copia el contenido de "buscado" a "temporal" para mantener la palabra objetivo en estado original y utilizarla posteriormente.
- void [horror](#) (const char *mensaje)
Transforma el número de error en la expresión a un mensaje de error.
- void [generatexto](#) (struct [registro](#) *ptr1, FILE *archivo, int **matriz, int fin, int LT)
Genera texto con base en la lista de palabras de idioma inglés básico y la matriz de probabilidad.
- void [guardarmatriz](#) (int **matriz, int fin, FILE *archivo)
Guarda valores de la matriz de probabilidad en el archivo "Matriz.csv".
- void [liberarmatriz](#) (int **matriz, int fin)
Libera memoria asignada para almacenar valores de la matriz.
- void [matrizestocastica](#) (int **matriz, int fin)
Se crea la matriz estocástica.
- void [liberarmalloc](#) (struct [registro](#) *ptr1)
Libera memoria asignada a la lista de palabras.
- void [cerrararchivos](#) (FILE *archivo1, FILE *archivo2, FILE *archivo3, FILE *archivo4, FILE *archivo5)
Cierra los archivos de texto utilizados.
- int ** [nuevamatriz](#) (int fin)
Se crea la matriz para almacenar las apariciones de la i-ésima y j-ésima palabra.
- int [comparapalabra](#) (struct [registro](#) *ptr1, struct [registro](#) *ptr2)
Compara dos palabras para determinar si son iguales.
- int [cargarpalabras](#) (struct [registro](#) *ptr1, FILE *archivo)
Copia palabras básicas desde un archivo de texto.
- void [coincidencias](#) (FILE *archivo2, FILE *archivo4, int **matriz, int tamanalista1, struct [registro](#) *lista1)
Busca las coincidentes existentes entre las palabras del texto objetivo y la lista de palabras básicas del inglés.
- int [main](#) (int argc, const char *argv[])
Función principal.

4.1.1. Descripción detallada

Algoritmo generador de texto markoviano para el idioma inglés básico.

Autor

Alumno: Luis Daniel Carvallo González
Matrícula: 207329091
Asesor: Dr. Pedro Lara Velázquez

Versión

1.0

Fecha

Noviembre de 2012

Algoritmo para generar texto en idioma inglés básico

4.1.2. Documentación de los 'typedefs'**4.1.2.1. typedef int(* comparar)(const void *, const void *)**

Prototipo para buscar y comprar palabras con las funciones qsort y bsearch.

4.1.3. Documentación de las funciones**4.1.3.1. int cargarpalabras (struct registro * ptr1, FILE * archivo)**

Copia palabras básicas desde un archivo de texto.

Con la función se cargan palabras de un archivo en una lista y se les agrega un número seriado. Las palabras que se utilizan tienen que estar ordenadas alfabéticamente de forma ascendente.

Parámetros

<i>ptr1</i>	Apuntador a lista de palabra e índice.
<i>archivo</i>	Archivo para cargar palabras en lista.

Devuelve

El número total de palabras de la lista.

```

{
    int total = 0;
    while (!feof(archivo)) {
        if (ptr1 != NULL) {
            fscanf(archivo, "%s", ptr1[total++].palabra);
            ptr1[total].indice = total;
        }
    }
    return ( total );
}

```

4.1.3.2. void cerrararchivos (FILE * archivo1, FILE * archivo2, FILE * archivo3, FILE * archivo4, FILE * archivo5)

Cierra los archivos de texto utilizados.

El programa finaliza cuando no se puede cerrar algún archivo utilizado.

Parámetros

<i>archivo1</i>	Archivo de palabras conocidas en idioma inglés básico.
<i>archivo2</i>	Archivo de texto objetivo.
<i>archivo3</i>	Archivo para guardar texto nuevo.
<i>archivo4</i>	Archivo para guardar palabras desconocidas.
<i>archivo5</i>	Archivo para almacenar valores de la matriz de probabilidad.

Devuelve

Vacío.

```
{
    if ((fclose(archivo1) != 0) && (archivo1 != NULL)){
        horror("Cerrar archivo con palabras en idioma inglés básico");
    }
    if ((fclose(archivo2) != 0) && (archivo2 != NULL)){
        horror("Cerrar archivo con texto objetivo");
    }
    if ((fclose(archivo3) != 0) && (archivo3 != NULL)){
        horror("Cerrar archivo texto nuevo");
    }
    if ((fclose(archivo4) != 0) && (archivo4 != NULL)){
        horror("Cerrar archivo para almacenar palabras no encontradas");
    }
    if ((fclose(archivo5) != 0) && (archivo5 != NULL)){
        horror("Cerrar archivo para almacenar valores de la matriz de
        probabilidad");
    }
    return;
}
```

4.1.3.3. void coincidencias (FILE * archivo2, FILE * archivo4, int ** matriz, int tamanalista1, struct registro * lista1)

Busca las coincidentes existentes entre las palabras del texto objetivo y la lista de palabras básicas del inglés.

Para llenar la matriz de ocurrencias de palabras se requiere hacer la lectura de cada palabra del archivo objetivo "Objetivo.txt". Las palabras del archivo objetivo no se almacenan en memoria con la finalidad de no demandar recursos en exceso al sistema operativo. Cuando se encuentra la i-ésima y la j-ésima palabra, se incrementa el valor del elemento correspondiente en la matriz. Si la palabra encontrada tiene signo de puntuación ("," y ".") al final de la misma se aumentara el valor correspondiente de la matriz. Cuando la palabra[i] y palabra[i + 1] se han analizado, se retorna el apuntador de lectura con la función "fseek" para poder analizar la palabra[i + 1] y palabra[i + 2]. En caso de no encontrarse la primer palabra se almacena en el archivo "No_encontradas.txt". No es necesario verificar si la j-ésima palabra no se encuentra porque lo que se requiere es encontrar dos ocurrencias contiguas. La palabra no encontrada se almacenan con el formato original para utilizarla posteriormente. Cuando finaliza el ciclo de comparación de las palabras se crea la matriz de probabilidad, donde el último elemento de cada renglón es el valor de la suma de cada elemento del renglón.

Parámetros

<i>archivo2</i>	Archivo que contiene el texto objetivo.
<i>archivo4</i>	Archivo donde se almacenan las palabras no encontradas.
<i>matriz</i>	Matriz donde se almacenan los valores numéricos de las coincidencias encontradas.
<i>tamanalista1</i>	Tamaño de la lista de palabras del idioma inglés básico.
<i>lista1</i>	Estructura donde se almacena la palabra que se compara con la lista de palabras del idioma inglés básico.

Devuelve

Vacío.

```
{
    struct registro *encontrado1 = NULL, *encontrado2 = NULL, buscado1,
        buscado2, temporal1, temporal2;
    size_t longitud;

    while (!feof(archivo2)) {
        fseek(archivo2, -(strlen(temporal2.palabra)), SEEK_CUR);

        fscanf(archivo2, "%s", temporal1.palabra);
        buscado1 = verificapalabra(temporal1);
        encontrado1 = bsearch(&buscado1, lista1, tamanalista1, sizeof (struct
        registro), (comparar) comparapalabra);
    }
}
```

```

fscanf(archivo2, "%s", temporal2.palabra);
buscado2 = verificapalabra(temporal2);
encontrado2 = bsearch(&buscado2, lista1, tamanolista1, sizeof (struct
registro), (comparar) comparapalabra);

    if ((encontrado1) && (encontrado2)) {
        matriz[encontrado1->indice][encontrado2->indice]++;

        longitud = strlen(temporal1.palabra);
        if (strncmp(&temporal1.palabra[longitud-1], ".", 1) == 0) {
            matriz[encontrado1->indice][1]++;
        }
        if (strncmp(&temporal1.palabra[longitud-1], ",", 1) == 0) {
            matriz[encontrado1->indice][0]++;
        }
    }
    else if (!encontrado1) {
        fprintf(archivo4, "%s \n", temporal1.palabra);
    }
}
return;
}

```

4.1.3.4. int comparapalabra (struct registro * ptr1, struct registro * ptr2)

Compara dos palabras para determinar si son iguales.

Se comparan dos palabras para saber si son iguales, en caso de serlo se devuelve el valor de cero.

Parámetros

<i>ptr1</i>	Apuntador a la primera palabra.
<i>ptr2</i>	Apuntador a la segunda palabra.

Devuelve

Un número mayor, igual, o menor que cero.

```

{
    return ( strcmp(ptr1->palabra, ptr2->palabra) );
}

```

4.1.3.5. void generatexto (struct registro * ptr1, FILE * archivo, int ** matriz, int fin, int LT)

Genera texto con base en la lista de palabras de idioma inglés básico y la matriz de probabilidad.

La función genera texto con base en el comportamiento de apariciones de palabras del texto objetivo y la correspondencia de éstas con la lista de palabras básicas del inglés propuestas. El tamaño del texto generado depende del valor asignado a "LT". La primer palabra que se utiliza es elegida con base en dos criterios. El primer criterio de selección depende de un número aleatorio entre la primera palabra de la lista y el número total de palabras de la lista del inglés básico. La primera palabra de la lista que se utiliza no puede ser signo de puntuación o número. El segundo criterio depende del número aleatorio asignado ya que éste debe corresponder a algún índice de matriz[m][fin] diferente de cero. Si el valor de matriz[m][fin] no es diferente de cero se busca un nuevo valor aleatorio de "m". El último elemento de cada fila es el valor entero que cumple la función de dividiendo para crear un vector de probabilidad, por lo que no se toma en cuenta para calcular la suma de probabilidad. Si el valor de matriz[m][fin] es cero no se toma en cuenta para generar texto, porque se interpreta como una palabra desconocida en el texto original. Posteriormente se obtiene un valor aleatorio comprendido entre (0.0, 1.0) para determinar cual es la siguiente palabra que se utilizará. La condición de paro para elegir la palabra siguiente depende del valor acumulado obtenido de la suma de cada elemento del vector de la matriz. Mientras que el número aleatorio sea mayor a la suma se continua con el ciclo hasta encontrar el valor que cumpla la condición de paro. Cuando el valor del número aleatorio es muy cercano a 1.0, menor o igual, la condición de paro se cumple. Por lo tanto la última palabra de la lista es utilizada como el siguiente elemento del texto generado. Si el valor del número aleatorio es mayor que la suma de cada elemento del vector de probabilidad se tomara como

siguiente palabra la última palabra de la lista. El valor de la variable "l" indica cual es la siguiente palabra de la lista y el inicio de la nueva búsqueda por comparación entre un número aleatorio y la suma de los elementos de cada fila de la matriz. La primera letra del texto generado y cualquier otra que se coloque después de un punto siempre se escriben en mayúscula. La variable de tipo entero "anterior" se utiliza para saber cual fue la última palabra utilizada y evitar que ésta se repita consecutivamente. En "registro ptr2" se almacena una copia de la palabra actual para modificarla si es necesario, por ejemplo convertir la primera letra en mayúscula. El pronombre personal "I", en ingles, siempre se escribe en mayúscula, para darle una apariencia natural al texto generado.

Parámetros

<i>ptr1</i>	Apuntador a lista de palabras.
<i>archivo</i>	Archivo para almacenar palabras.
<i>matriz</i>	Doble apuntador de la matriz.
<i>fin</i>	Valor que establece las dimensiones de la matriz.
<i>LT</i>	Número de palabras en idioma inglés básico que genera el algoritmo.

Devuelve

Vacío.

```
{
    int j = 0, k = 0, l = 1, m = rand() % (23 - fin + 1) + 23, anterior = 0,
        flag = 0;
    float random = 1.0, suma = 0.0;
    struct registro ptr2;

    while (k < LT) {
        if (matriz[m][fin] > 0) {
            random = drand48();
            l = 0;
            for (j = 0; (j < fin) && (suma <= 1.0); j++) {
                suma += ( (float) matriz[m][j] / (float) matriz[m][fin] );
                if (random >= suma) {
                    l++;
                }
            }
            if ( ptr1[l].palabra != ptr1[anterior].palabra ) {
                if (strncmp(ptr1[l].palabra, "i", 2) == 0) {
                    ptr1[l].palabra[0] = toupper(ptr1[l].palabra[0]);
                }
                if (k == 0) {
                    strcpy(ptr2.palabra, ptr1[l].palabra);
                    ptr2.palabra[0] = toupper(ptr2.palabra[0]);
                    printf(" %s", ptr2.palabra);
                    fprintf(archivo, "%s", ptr2.palabra);
                }
                if (k > 0) {
                    if (strncmp(ptr1[l].palabra, ".", 1) == 0) {
                        if (strncmp(ptr1[anterior].palabra, ",", 1) == 0) {
                            anterior = l;
                            k--;
                        }
                        else{
                            printf("%s", ptr1[l].palabra);
                            fprintf(archivo, "%s", ptr1[l].palabra);
                            anterior = l;
                            flag = 1;
                        }
                    }
                    else if (strncmp(ptr1[l].palabra, ",", 1) == 0) {
                        if (strncmp(ptr1[anterior].palabra, ".", 1) == 0) {
                            anterior = l;
                            k--;
                        }
                        else{
                            printf("%s", ptr1[l].palabra);
                            fprintf(archivo, "%s", ptr1[l].palabra);
                            anterior = l;
                        }
                    }
                }
            }
            else{
                printf("%s", ptr1[l].palabra);
                fprintf(archivo, "%s", ptr1[l].palabra);
                anterior = l;
            }
        }
    }
}
```



```

        if (flag == 1) {
            strcpy(ptr2.palabra, ptr1[l].palabra);
            ptr2.palabra[0] = toupper(ptr2.palabra[0]);
            printf(" %s", ptr2.palabra);
            fprintf(archivo, " %s", ptr2.palabra);
            anterior = 1;
            flag = 0;
        }
        else{
            printf(" %s", ptr1[l].palabra);
            fprintf(archivo, " %s", ptr1[l].palabra);
            anterior = 1;
        }
    }
    }
    k++;
    m = 1;
}
suma = 0.0;
}
else {
    m = rand() % (fin + 1);
}
}
printf("...\n\n");
return;
}

```

4.1.3.6. void guardarmatriz (int ** matriz, int fin, FILE * archivo)

Guarda valores de la matriz de probabilidad en el archivo "Martriz.csv".

Para evitar almacenar valores con formato fraccionario se almacena el denominador de cada vector de probabilidad al final de cada fila.

Parámetros

<i>matriz</i>	Doble apuntador de la matriz.
<i>fin</i>	Valor que establece las dimensiones de la matriz.
<i>archivo</i>	Archivo para almacenar palabras.

Devuelve

Vacío.

```

{
    int i = 0, j = 0;
    for (i = 0; i <= fin; i++) {
        for (j = 0; j <= fin; j++) {
            fprintf(archivo, "%i,", matriz[i][j]);
        }
        fprintf(archivo, "\n");
    }
    return;
}

```

4.1.3.7. void horror (const char * mensaje)

Transforma el número de error en la expresión a un mensaje de error.

Un valor diferente de cero en caso de finalizar el programa de forma incorrecta.

Parámetros

<i>mensaje</i>	Mensaje de error que se imprime en pantalla.
----------------	--

Devuelve

Vacío.

```
{
    perror(mensaje);
    exit(EXIT_FAILURE);
    return;
}
```

4.1.3.8. void liberarmalloc (struct registro * ptr1)

Libera memoria asignada a la lista de palabras.

La lista de palabras del inglés básico se almacena en memoria dinámica para que al final del algoritmo se liberen los recursos utilizados.

Parámetros

<i>ptr1</i>	Apuntador a lista de palabras del idioma inglés básico.
-------------	---

Devuelve

Vacío.

```
{
    free(ptr1);
    ptr1 = NULL;
    return;
}
```

4.1.3.9. void liberarmatriz (int ** matriz, int fin)

Libera memoria asignada para almacenar valores de la matriz.

La matriz que se utiliza para almacenar las ocurrencias de cada palabra utiliza memoria dinámica, con la finalidad de liberar recursos al final de la ejecución del algoritmo. Primero se libera el apuntador del contenido de cada elemento de la matriz y finalmente se libera el apuntador de la matriz.

Parámetros

<i>matriz</i>	Doble apuntador de la matriz con memoria dinámica.
<i>fin</i>	Valor que establece las dimensiones de la matriz.

Devuelve

Vacío.

```
{
    while (fin >= 0) {
        free(matriz[fin--]);
    }
    free(matriz);
    matriz = NULL;
    return;
}
```

4.1.3.10. int main (int argc, const char * argv[])

Función principal.

Se inicia la función principal con la declaración de variables, posteriormente se validan los archivos que se utilizan para leer y almacenar las palabras utilizadas. Si la apertura de archivos es correcta se comienzan a cargar las palabras que se utilizarán para generar un texto nuevo en memoria. A partir del tamaño de la lista cargada en memoria se obtiene el tamaño de la misma para crear una matriz. La lista de palabras que se utilizan para generar texto pueden no estar ordenadas ya que con la función "qsort" se asegura el orden de la lista para buscar coincidencias. Cuando se obtiene el nuevo valor de la lista de palabras se crea la matriz cuadrada donde se almacenarán las ocurrencias de cada palabra y la siguiente. Cuando se tiene un patrón de escritura se crea una matriz de probabilidad, donde la suma de cada uno de los elementos de cada fila de la matriz es igual a 1. Con los valores de la matriz de probabilidad se generará texto mediante el uso de la función "generatexto". Con la función "generatexto" se crea nuevo texto que se almacena en el archivo "Texto_Generado.txt" Cuando se ha generado el texto se libera la memoria utilizada para almacenar la lista de palabras y la matriz de probabilidad. Finalmente se cierran todos los archivos utilizados por el programa.

Devuelve

Un valor entero igual a cero en caso de finalizar el programa de forma correcta.

```
{
int tamanolista1 = 0, **matriz = NULL, LT = 0;
struct registro *lista1 = NULL;
FILE *archivo1 = NULL, *archivo2 = NULL, *archivo3 = NULL, *archivo4 = NULL
, *archivo5 = NULL;
srand(getpid());
srand48(getpid());

if (argc != 7){
horror("Usar: ./main 200 Lista.txt Objetivo.txt Generado.txt
Desconocidas.txt Matriz.csv");
}

sscanf(argv[1], "%d", &LT);

archivo1 = fopen(argv[2], "r");
if (!archivo1) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("Archivo con lista de palabras");
}
archivo2 = fopen(argv[3], "r");
if (!archivo2) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("Archivo con texto objetivo");
}
archivo3 = fopen(argv[4], "w");
if (!archivo3) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("Archivo con texto nuevo");
}
archivo4 = fopen(argv[5], "w");
if (!archivo4) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("Archivo para almacenar palabras no encontradas");
}
archivo5 = fopen(argv[6], "w");
if (!archivo5) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("Archivo para almacenar valores de la matriz de
probabilidad");
}

lista1 = (struct registro *) malloc(sizeof (struct registro
)*3000);
if (lista1 == NULL) {
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);
horror("No se puede crear lista de palabras del idioma ingles
basico");
}

tamanolista1 = cargarpalabras(lista1, archivo1);
matriz = nuevamatriz(tamanolista1);
```

```

qsort(listal, tamanolista, sizeof (struct registro), (comparar
) comparapalabra);
coincidencias(archivo2, archivo4, matriz, tamanolista, listal
);
matrizestocastica(matriz, tamanolista);
guardarmatriz(matriz, tamanolista, archivo5);
generatexto(listal, archivo3, matriz, tamanolista, LT);
liberarmatriz(matriz, tamanolista);
liberarmalloc(listal);
cerrararchivos(archivo1, archivo2, archivo3, archivo4,
archivo5);

return ( EXIT_SUCCESS );
}

```

4.1.3.11. void matrizestocastica (int ** matriz, int fin)

Se crea la matriz estocástica.

Se realiza la suma de cada elemento de todas las filas de la matriz, para crear una matriz estocástica utilizada para describir las transiciones en una cadena de Márkov. El denominador de cada elemento de la fila se almacena en la posición de la matriz[i][fin]. Cada vector de la nueva matriz se convierte en un vector de probabilidad, porque todos los elementos del mismo son positivos y al sumarlos el resultado es siempre igual a 1.

Parámetros

<i>matriz</i>	Doble apuntador de la matriz con memoria dinámica.
<i>fin</i>	Valor que establece las dimensiones de la matriz estocástica.

Devuelve

Vacío.

```

{
int i = 0, j = 0, suma = 0;
for (i = 0; i < fin; i++) {
for (j = 0; j < fin; j++) {
suma += matriz[i][j];
}
matriz[i][fin] = suma;
suma = 0;
}
return;
}

```

4.1.3.12. int** nuevamatriz (int fin)

Se crea la matriz para almacenar las apariciones de la i-ésima y j-ésima palabra.

Matriz que se inicia con los registros necesarios para almacenar valores enteros (dependen del valor de la variable "fin"), los cuales corresponden a las apariciones de las palabras de un texto objetivo. El programa finaliza cuando no se puede crear la matriz.

Parámetros

<i>fin</i>	Tamaño de la matriz cuadrada.
------------	-------------------------------

Devuelve

Doble apuntador de la matriz.

```

{

```

```

int k = 0, **ptr1 = (int **) calloc(fin + 1, sizeof (int *));
if (ptr1 != NULL) {
    for (k = 0; k <= fin; k++) {
        ptr1[k] = (int *) calloc(fin + 1, sizeof (int));
    }
}
else{
    free(ptr1);
    ptr1 = NULL;
    horror("No se puede crear la matriz");
}
return ( ptr1 );
}

```

4.1.3.13. struct registro verificapalabra (struct registro *buscado*) [read]

Se copia el contenido de "buscado" a "temporal" para mantener la palabra objetivo en estado original y utilizarla posteriormente.

La función convierte caracteres de mayúsculas a minúsculas y elimina signos de puntuación de la copia de la palabra original. Si la palabra contiene caracteres en minúsculas no se realiza ninguna conversión. Cuando se encuentra un signo de puntuación en la palabra, se hace un desplazamiento a la izquierda de los caracteres posteriores al signo de puntuación.

Parámetros

<i>buscado</i>	Es la palabra a la que se convertirán los caracteres a minúsculas y se eliminarán signos de puntuación.
----------------	---

Devuelve

Una palabra con formato correcto, sin mayúsculas y signos de puntuación.

```

{
    struct registro temporal = buscado;
    int i = 0, j = 0;
    size_t longitud;
    longitud = strlen(temporal.palabra);

    for (i = 0; i <= longitud; i++) {
        temporal.palabra[i] = tolower(temporal.palabra[i]);
    }
    while (temporal.palabra[j] != '\0') {
        if (ispunct(temporal.palabra[j])) {
            for (i = j; i <= longitud; i++) {
                temporal.palabra[i] = temporal.palabra[i + 1];
            }
        }
        else {
            j++;
        }
    }
    return ( temporal );
}

```

Índice alfabético

cargarpalabras
main.c, 3

cerrararchivos
main.c, 4

coincidencias
main.c, 4

comparapalabra
main.c, 5

comparar
main.c, 3

generatexto
main.c, 6

guardarmatriz
main.c, 7

horror
main.c, 8

indice
registro, 1

liberarmalloc
main.c, 8

liberarmatriz
main.c, 8

main
main.c, 9

main.c, 2
cargarpalabras, 3
cerrararchivos, 4
coincidencias, 4
comparapalabra, 5
comparar, 3
generatexto, 6
guardarmatriz, 7
horror, 8
liberarmalloc, 8
liberarmatriz, 8
main, 9
matrizestocastica, 10
nuevamatriz, 11
verificapalabra, 11

matrizestocastica
main.c, 10

nuevamatriz
main.c, 11

palabra
registro, 1

registro, 1
indice, 1
palabra, 1

verificapalabra
main.c, 11

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Reporte de Proyecto Terminal II

**Algoritmo generador de texto markoviano
para el idioma inglés básico**

Elaboró

Luis Daniel Carvallo González
Matrícula: 207329091

Asesor

Dr. Pedro Lara Velázquez
Departamento de Sistemas

Trimestre 12O
Noviembre de 2012

Índice

1	Introducción	4
2	Objetivo general	5
3	Objetivos particulares	5
4	Antecedentes	5
5	Justificación de proyecto	6
6	Metodología de base empleada en el proyecto	7
7	Desarrollo del proyecto	8
7.1	Diseño	8
7.1.1	Entrada de datos	9
7.1.2	Validación de archivos	11
7.1.3	Reconocimiento del patrón de escritura	11
7.1.4	Generar de texto	15
7.1.5	Liberar recursos del sistema	17
7.2	Implementación	18
7.2.1	Tecnología empleada	18
7.2.2	Compilación y ejecución	18
8	Conclusiones	20
9	Anexos	24
9.1	Contenido del CD	24
9.2	Figuras complementarias	25
10	Glosario	30

Índice de figuras

1	Entrada, proceso y salida del algoritmo.	8
2	Diagrama de casos de uso.	9
3	Módulos del algoritmo.	10
4	Texto nuevo generado con el algoritmo markoviano.	17
5	Compilar y ejecutar programa.	19
6	Error en los parámetros de entrada.	19
7	Texto nuevo con lista de 850 palabras.	22
8	Texto nuevo con lista de 925 palabras.	22
9	Texto nuevo con lista de 1024 palabras.	22
10	Entorno de desarrollo Xcode 3.2.6.	25
11	Fragmento de la lista de palabras básicas.	27
12	Fragmento de la lista de palabras encontradas.	28
13	Texto nuevo generado con el algoritmo markoviano.	29

1. Introducción

Considerando que cualquier texto contiene un patrón de escritura, se puede diseñar un algoritmo que reconozca dicho patrón y emule su comportamiento produciendo un nuevo texto. El reconocimiento de patrones en computación es tan variado que se pueden encontrar aplicaciones que van desde la predicción de cadenas de caracteres en biología y el reconocimiento de búsquedas de contenido en internet. Por el momento no se pretende simular la escritura en un lenguaje natural, sin embargo, se busca generar texto en un idioma con un cierto número de palabras básicas.

Es decir, cada individuo hace referencia a un conjunto específico de palabras para poder delimitar y hacer entendible lo que se está comunicando. Por lo tanto, al hacer una revisión detallada de la secuencia de palabras utilizadas se puede observar la presencia de un patrón. Sea el número de palabras en una oración o la cantidad de oraciones en un párrafo, los textos pueden ser clasificados a partir de ciertos parámetros que sean determinados previamente para definir un estilo en la forma de escribir.

Para el presente proyecto se eligió el idioma inglés, porque éste a nivel básico difícilmente presenta complejidades lingüísticas. Lo que permite que a partir de la lectura de un archivo de texto se reconozca un patrón de escritura para que posteriormente con tan sólo un mínimo de palabras en idioma inglés básico se genere un nuevo texto. Bastará de momento señalar que se eligió el idioma inglés por que existe de manera documentada lo que se considera un conjunto de palabras mínimas del idioma para entablar canales de comunicación efectiva [1].

Teniendo en consideración los párrafos anteriores se expone por medio del presente trabajo como se puede generar un texto en idioma inglés básico con un algoritmo codificado en lenguaje C y utilizando la teoría de las cadenas de Márkov [2].

Con la intención de que los textos obtenidos cumplan ciertas características que favorezcan la clasificación de los mismos, el tipo de texto generado tendrá niveles de complejidad que dependerán de la configuración de las variables de entrada del algoritmo, ya que se asegura la generación de nuevo texto que puede ser sencillo de leer o puede requerir de una profunda revisión para ser entendido.

2. Objetivo general

Diseñar e implementar en lenguaje C un algoritmo capaz de generar texto en idioma inglés básico con base en procesos estocásticos.

3. Objetivos particulares

- Diseñar algoritmo para simular cadenas de Márkov.
- Diseñar algoritmo para generar texto en idioma inglés básico.
- Implementar algoritmo para simular cadenas de Márkov, realizando ajustes necesarios para lograr la generación posterior de textos en idioma inglés básico.
- Implementar algoritmo para generar texto en idioma inglés básico.
- Generar instancias de texto markoviano en idioma inglés básico.
- Analizar resultados de simulación.

4. Antecedentes

Las referencias a trabajos de mayor relevancia, con respecto a los modelos de Márkov y la clasificación de textos, que se han realizado al interior de la Universidad son los siguientes:

- *Algoritmo para la predicción de mensajes de texto en español escritos en teléfono celular* [3]. En el proyecto terminal se utiliza en específico el algoritmo de Viterbi para encontrar la secuencia de estados más probable de un Modelo Oculto de Márkov (MOM); donde el mensaje de texto ingresado se comporta como el estado desconocido propio del MOM para posteriormente encontrar una probabilidad de secuencia de texto. Sin embargo, se utiliza de forma concreta sólo la teoría referente a las cadenas de Márkov, con lo que se busca encontrar las probabilidades de la ocurrencia de estados con base en el último evento sucedido; donde las palabras de un texto inicial o modelo se comportaran como estados conocidos para generar un nuevo pseudo texto. Siendo, por lo tanto, el manejo de los estados la diferencia más notable.
- *Sistema clasificador de documentos de proyectos terminales usando el concepto de memoria asociativa* [4]. En el proyecto terminal se logra clasificar documentos gracias a la implementación de modelos matemáticos que simulan la inteligencia humana. Los patrones que son reconocidos se obtienen por medio del procesamiento de datos en una red neuronal de varios perceptrones. La diferencia sustancial es la forma de clasificar textos. El sistema clasificador utiliza herramientas propias de la inteligencia artificial, mientras que el algoritmo markoviano propuesto utiliza los procesos estocásticos.

Las siguientes referencias se pueden consultar proyectos, ajenos a la Universidad, que proponen el uso de procesos estocásticos en combinación con algoritmos computacionales:

- *Method for node ranking in a linked database* [5]. Patente propiedad de Google que utiliza las cadenas de Márkov como herramienta para establecer la importancia de las páginas web que son indexadas por su propio motor de búsqueda. La importancia de las páginas web se establece gracias a que éstas son tratadas como estados y los nodos de enlace a otras páginas son las transiciones. Por lo tanto, se le asigna un valor numérico a cada página para determinar su importancia de aparición en los resultados de búsqueda. La idea del algoritmo que tiene importancia para el proyecto terminal es fundamentalmente que en lugar de signar valores numéricos a nodos, se hará sobre palabras, logrando establecer cual es la siguiente palabra con base en una ponderación numérica que le destacará ante otras palabras de un conjunto.
- *PHP Markov chain generator* [6]. Página web que presenta un generador de texto codificado en PHP. La diferencia fundamental con el proyecto terminal es en específico la capacidad de aprovechar el comportamiento del texto generado para poder clasificar otros textos que cumplan con ciertas características específicas.
- *AGILIQ blog* [7]. Sitio web que presenta un algoritmo para un generador de texto pseudo aleatorio con cadenas de Márkov. Aunque se logra generar algún tipo de texto, no es posible rescatar resultados de los procesos estocásticos previos al resultado. Situación que no permite utilizar el algoritmo para aprovechar el reconocimiento de patrones como medio clasificador de textos.

5. Justificación de proyecto

Se plantea la idea del diseño e implementación de un algoritmo generador de textos markoviano para el inglés básico, el cual se utilizará para localizar patrones en otros textos; por ejemplo: el número de palabras más utilizadas en un texto o la secuencia de las mismas en una oración, considerando principalmente los resultados de los procesos estocásticos propios del algoritmo.

Es importante tener en cuenta que el lenguaje que utilizan los seres humanos tiene cierta cantidad de palabras que permiten entablar algún tipo de comunicación con los demás, manteniendo una coherencia con cada una de las locuciones realizadas. Es decir, cada individuo hace referencia a un conjunto específico de palabras para poder delimitar y hacer entendible lo que se está comunicando.

Por ejemplo, el idioma inglés es considerado una herramienta de gran importancia en varios países, como lengua oficial o segunda lengua, por lo que se utiliza como medio de comunicación eficaz para entablar algún tipo de comunicación. Además, cabe señalar que ya se han especificado un mínimo de 850 palabras para definir el inglés básico, las cuales permiten a los seres humanos por ejemplo: conectar dos o más ideas, describir objetos tangibles o intangibles e incluso adjudicar calificativos a las cosas [1].

Por lo tanto un texto de novela fácilmente se puede distinguir de un texto científico por la estructura de los mismos y por la forma en que se ordenan las palabras; otorgando cierta particularidad a los textos. En concreto, cada autor puede conocer el mismo número de palabras, describiendo y entendiendo con exactitud cada una de éstas de manera objetiva, pero siempre construirán textos de forma que se logra un estilo distinguible.

Con base en lo anterior, se plantea utilizar el inglés básico para generar texto, ya que de momento se puede acotar la cantidad de palabras mínimas necesarias para que cualquier texto generado con el algoritmo sea entendible. Y porque el inglés es menos susceptible a variaciones lingüísticas, como sucede con el castellano.

La importancia de lograr la generación de texto radica en que puede ser el inicio de trabajos posteriores, en el campo de la Ingeniería en Computación, que intenten generar algún tipo de texto mediante el uso de algoritmos computacionales y los procesos estocásticos.

Por mencionar un escenario específico, se puede hacer referencia a las sutiles diferencias que tienen los textos científicos y las novelas literarias. Cada uno de los textos cumple con ciertos patrones, si se prefiere estilos de composición, que les hacen diferentes y acreedores de un cierto público. Fundamentalmente existen formas de escritura que no pueden ser intercambiadas entre ambos estilos.

Entonces, si se considera la presencia de palabras en un texto como eventos, con cierta probabilidad de suceder uno tras otro, se puede hacer un algoritmo que utilice tal fenómeno para poder generar texto y establecer lineamientos para lograr clasificaciones de otros textos. Al respecto, si se considera a las palabras como variables que cambian en un lapso de tiempo, se pueden proyectar a éstas como procesos estocásticos que con base en las cadenas de Márkov pueden ser vistas en cualquier instante.

Finalmente al establecerse una modelo matemático, se puede lograr el reconocimiento de patrones para clasificar textos de manera que no se demande en exceso recursos computacionales cada vez que se actualiza el algoritmo; ver apartado de la descripción técnica.

6. Metodología de base empleada en el proyecto

La creación del algoritmo generador de texto markoviano para el inglés básico se basa en la técnica de descomposición de procesos descendentes, contemplando la entrada de datos, la salida de los mismos y los métodos o formulas empleadas para procesar los datos. El problema que se busca resolver es generar texto en inglés básico, a partir de un patrón de escritura.

La entrada de datos consta de un archivo de texto objetivo, del cual se obtiene un patrón de escritura. También se utiliza una lista de palabras en inglés básico para generar texto. El proceso de generar texto consiste en reconocer un patrón de escritura a partir de la secuencia de palabras que conforman al archivo objetivo, finalmente la información se almacena en una matriz de probabilidad para ser utilizada como modelo de escritura para el nuevo texto; el proceso de generar texto se explica con mayor detalle en el apartado 7.1.

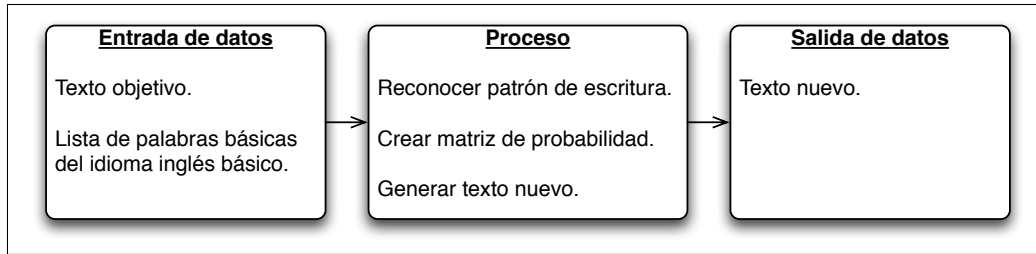


Figura 1: Entrada, proceso y salida del algoritmo.

7. Desarrollo del proyecto

A continuación se describe el funcionamiento del algoritmo generador de texto markoviano para el inglés básico a partir de una perspectiva de módulos, también se incluye la descripción de como se diseño e implemento el algoritmo.

En el primer apartado se presenta el funcionamiento del algoritmo a través de representaciones modulares, junto a una descripción de lo que hacen los mismos. Los módulos que se exponen son los siguientes: entrada de datos, validación de archivos utilizados, reconocimiento de patrón de escritura, generación de texto y liberación de recursos utilizados.

En el apartado de implementación se exponen los recursos de hardware y software utilizados para diseñar y poner en ejecución el algoritmo para obtener texto nuevo a partir de un archivo de texto objetivo.

7.1. Diseño

El usuario puede generar texto en inglés básico con ayuda del algoritmo a partir de un archivo objetivo. El archivo puede ser de cualquier cantidad de palabras, sin embargo, para reconocer un patrón de escritura se requiere al menos de 9,000 palabras.

El motivo de que el archivo de texto cuente con una gran cantidad de palabras se debe a que el texto generado sólo utiliza una lista de un mínimo de 850 palabras en inglés básico, es decir, se presentarán los casos donde las palabras del archivo objetivo no sean reconocidas y no se contemplan para encontrar un patrón de escritura.

La lista de palabras que se utiliza para generar texto tiene que estar ordenada alfabéticamente, ascendentemente, para agilizar la búsqueda por coincidencias entre el texto objetivo y la lista. Se pueden agregar nuevas palabras a lista, siempre y cuando se ordenen alfabéticamente; ver Figura 11.

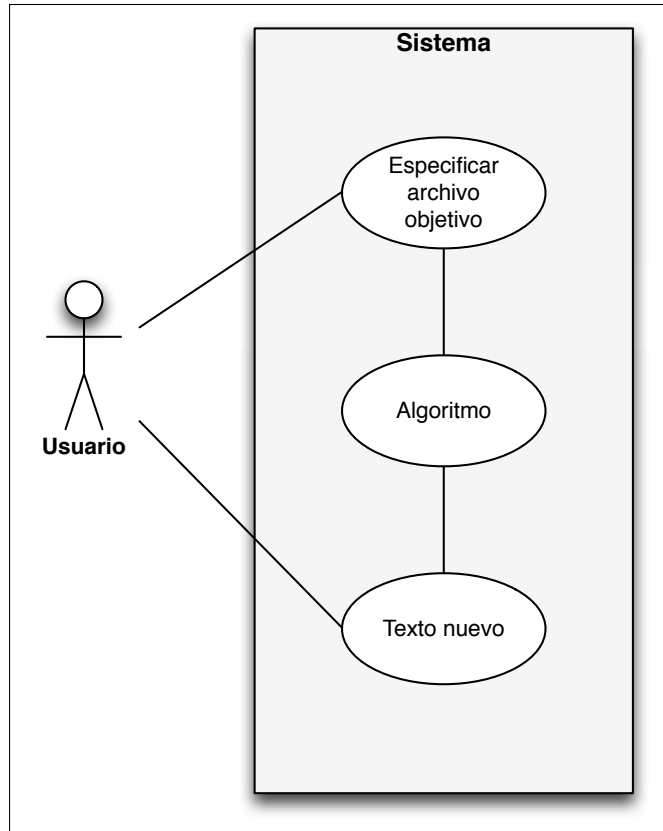


Figura 2: Diagrama de casos de uso.

Se puede apreciar en la Figura 2 cómo es que el usuario interactúa con el sistema para obtener un texto generado a partir del reconocimiento de patrones de escritura. El algoritmo generador de texto markoviano para el inglés básico se compone de los módulos que se muestran a continuación. La descripción de los módulos se hace de manera secuencial, con apego al funcionamiento del algoritmo.

El código fuente del algoritmo markoviano generador de texto en idioma inglés básico está documentado a detalle con base en las convenciones de Doxygen¹.

7.1.1. Entrada de datos

El primero módulo requiere de la entrada de un archivo de texto sin formato para poder determinar el patrón de escritura para generar nuevo texto. Se recomienda utilizar un archivo de texto plano sin formato para no agregar caracteres extraños que puedan interferir con la secuencia del algoritmo, ya que al momento de cargar palabras del archivo en memoria se requiere que éstas puedan compararse con la lista de palabras propuestas para definir al idioma inglés básico; ver Código 3.

¹Sistema generador de documentación para código fuente. Para mayor información: <http://www.stack.nl/~dimitri/doxygen/>

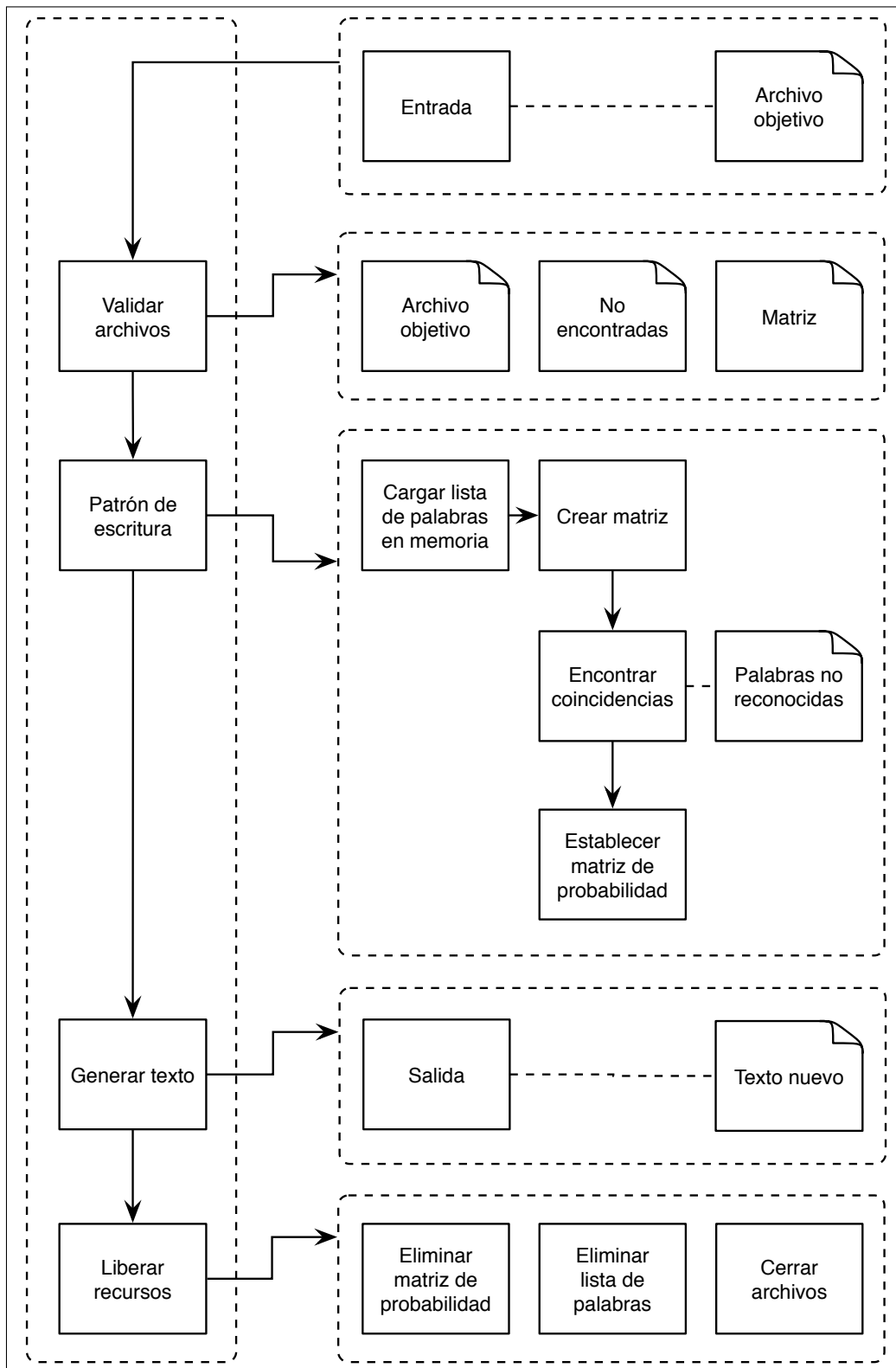


Figura 3: Módulos del algoritmo.

Sin embargo, las pruebas realizadas incluyen pruebas con archivos de texto con formatos diferentes a *txt* con resultados exitosos; otros formatos utilizados: *rtf* y *pages*.

```
int main(int argc, const char *argv[])
{
    ....
    if (argc != 7){
        horror("Usar: ./main 200 Lista.txt Objetivo.txt Generado.txt Desconocidas.txt Matriz.csv");
    }
    ....
}
```

Código 1: Parámetros de entrada.

7.1.2. Validación de archivos

En este módulo se valida la apertura o cierre de archivos de texto en modo lectura o escritura, ya que se requieren de éstos para cargar en memoria la lista de palabras del idioma inglés básico (lectura), guardar los valores de la matriz de probabilidad (escritura), el texto nuevo (escritura) y las palabras desconocidas encontradas en el texto objetivo (escritura).

```
...
archivo1 = fopen(argv[2], "r");
if (!archivo1) {
    cerrararchivos(archivo1, archivo2, archivo3, archivo4, archivo5);
    horror("Archivo con lista de palabras");
}
archivo2 = fopen(argv[3], "r");
if (!archivo2) {
    cerrararchivos(archivo1, archivo2, archivo3, archivo4, archivo5);
    horror("Archivo con texto objetivo");
}
archivo3 = fopen(argv[4], "w");
if (!archivo3) {
    cerrararchivos(archivo1, archivo2, archivo3, archivo4, archivo5);
    horror("Archivo con texto nuevo");
}
...
```

Código 2: Validación de apertura de archivos.

7.1.3. Reconocimiento del patrón de escritura

Este módulo es parte fundamental del algoritmo, ya que es donde se determina el patrón de escritura que servirá como semilla para generar nuevo texto. La primera etapa consiste en cargar en memoria la lista de palabras del idioma inglés básico, las cuales pueden tener un máximo de 3000 elementos. A partir del número de palabras en memoria se crea la matriz donde se almacenan los valores de las coincidencias de las palabras encontradas en el archivo objetivo.

```

....
lista1 = (struct registro *) malloc(sizeof (struct registro)*3000);
if (lista1 == NULL) {
    cerrararchivos(archivo1, archivo2, archivo3, archivo4, archivo5);
    horror("No se puede crear lista de palabras del idioma ingles basico");
}
tamanolista1 = cargarpalabras(lista1, archivo1);
....
while (!feof(archivo)) {
    if (ptr1 != NULL) {
        fscanf(archivo, "%s", ptr1[total++].palabra);
        ptr1[total].indice = total;
    }
}
....

```

Código 3: Memoria reservada para lista de palabras.

La siguiente etapa de este módulo consiste en encontrar coincidencias entre la lista de palabras cargadas en memoria y las palabras del archivo de texto objetivo. Las palabras del archivo objetivo se someten a un proceso de verificación de formato, para eliminar signos de puntuación y convertirlas a minúsculas si es necesario, case contrario se continua con el proceso de buscar coincidencias; ver Código 4.

```

struct registro verificapalabra(struct registro buscado)
{
....
for (i = 0; i <= longitud; i++) {
    temporal.palabra[i] = tolower(temporal.palabra[i]);
}
while (temporal.palabra[j] != '\0') {
    if (ispunct(temporal.palabra[j])) {
        for (i = j; i <= longitud; i++) {
            temporal.palabra[i] = temporal.palabra[i + 1];
        }
    }
    else {
        j++;
    }
}
....

```

Código 4: Verificar formato de palabras.

Por ejemplo *you are beautiful* y *are you beautiful* son el resultado de eliminar los signos de puntuación de cualquiera de las siguientes oraciones :

- You are beautiful.
- You are beautiful,
- Are you beautiful?
- You are beautiful...
- "You are beautiful"
- You are beautiful!

Si se establecen las palabras como estados (*You, are, beautiful*), junto a sus respectivas distribuciones de probabilidad, entonces las confianzas se determinan a partir de encontrar las palabras del texto objetivo *You* y *are* en la lista de palabras cargadas en memoria, para poder asignar un valor entero a la coordenada de la matriz (924, 56). Las coincidencias se buscan hasta que se encuentra el final del archivo objetivo; ver Código 5

$$\begin{array}{c}
 X_0 \\
 X_1 \\
 X_2 \\
 \dots \\
 X_{921} \\
 X_{922} \\
 X_{923} \\
 X_{924}
 \end{array}
 \begin{bmatrix}
 Y_0 & Y_1 & Y_2 & \dots & Y_{56} & \dots & Y_{921} & Y_{922} & Y_{923} & Y_{924} \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\
 \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}$$

Cuando alguna palabra no se encuentra en la lista cargada en memoria se almacena en el archivo *No_Encontradas.txt*, manteniendo el formato original que puede constar de mayúsculas o signos de puntuación; ver Figura 12.

- | | | |
|---------------|-------------|---------------|
| ▪ personal | ▪ worry. | ▪ whole |
| ▪ lives | ▪ stew. | ▪ gang |
| ▪ musically. | ▪ Just | ▪ soon |
| ▪ "1967-1970" | ▪ along. | ▪ Wherever |
| ▪ two-disc | ▪ happening | ▪ best |
| ▪ Album") | ▪ too. | ▪ best. |
| ▪ Hot | ▪ OH! | ▪ Wherever |
| ▪ Chili | ▪ PLACES | ▪ Except |
| ▪ Peppers | ▪ seeing | ▪ sometimes, |
| ▪ American | ▪ sights! | ▪ am |
| ▪ rock | ▪ fliers | ▪ sorry |
| ▪ started | ▪ soar | ▪ sadly, |
| ▪ streets. | ▪ heights. | ▪ Bang-ups |
| ▪ choose | ▪ lag | ▪ Hang-ups |
| ▪ brains | ▪ behind, | ▪ Slump, |
| ▪ shoes | ▪ speed. | ▪ fun. |
| ▪ feet, | ▪ pass | ▪ Un-slumping |

```

void coincidencias(FILE *archivo2, FILE *archivo4, int **matriz, int tamanalista1, struct registro *lista1)
{
    ...
    while (!feof(archivo2)) {
        fseek(archivo2, -(strlen(temporal2.palabra)), SEEK_CUR);
        fscanf(archivo2, "%s", temporal1.palabra);
        buscado1 = verificapalabra(temporal1);
        encontrado1 = bsearch(&buscado1, lista1, tamanalista1, sizeof (struct registro), (comparar)
            comparapalabra);
        ...
        if ((encontrado1) && (encontrado2)) {
            matriz[encontrado1->indice][encontrado2->indice]++;
            longitud = strlen(temporal1.palabra);
            if (strncmp(&temporal1.palabra[longitud-1], ".", 1) == 0) {
                matriz[encontrado1->indice][1]++;
            }
            ...
        }
        else if (!encontrado1) {
            fprintf(archivo4, "%s \n", temporal1.palabra);
        }
    }
    return;
}

```

Código 5: Búsqueda de coincidencias.

Finalmente se establece la matriz de probabilidad, para poder generar texto en el siguiente modulo. La matriz puede ser de hasta 3000 por 3000 elementos, reservado siempre el último valor de cada renglón para almacenar la suma de todas las coincidencias encontradas.

	,	.	1	...	<i>able</i>	<i>about</i>	<i>account</i>	<i>acid</i>	...	<i>Total</i>
,	0	0	0	...	0	0	0	0	...	0
.	0	0	0	...	0	0	0	0	...	0
1	0	0	0	...	0	0	0	0	...	0
...
<i>able</i>	0	0	0	...	0	0	0	0	...	0
<i>about</i>	0	0	0	...	0	0	0	0	...	0
<i>account</i>	0	0	0	...	0	0	0	0	...	0
...

El último valor de cada fila de la matriz se utilizará como denominador para aplicar el método de Montecarlo a las apariciones de las coincidencias en texto [8]. Tomando en cuenta que cada coordenada de la matriz puede o no tener un valor entero, se requiere de un valor a partir de la suma de los elementos de la fila de la matriz para poder utilizar números aleatorios y hacer una simulación de todas las posibles parejas de palabras coincidentes.

	,	.	1	...	able	about	account	acid	...	Total
,	1	0	0	...	2	0	0	0	...	3
.	0	0	3	...	5	5	5	1	...	19
1	1	0	6	...	1	4	0	1	...	13
...
able	0	0	0	...	0	0	0	0	...	0
about	0	0	0	...	0	0	0	0	...	0
account	2	0	4	...	2	0	0	1	...	9
...

Para la primera fila se tienen tres coincidencias, la primera y quinta palabra, por lo tanto el denominador del vector de probabilidad es 3. De tal forma que se cumple la condición de una matriz de probabilidad markoviana, cuando la suma de los elementos de la fila es igual a la unidad.

	,	.	1	...	able	about	account	acid	...	Total
Fila ₀	$\frac{1}{3}$	0	0	...	$\frac{2}{3}$	0	0	0	...	$\frac{3}{3}$
Fila ₁	0	0	$\frac{3}{19}$...	$\frac{5}{19}$	$\frac{5}{19}$	$\frac{5}{19}$	$\frac{1}{19}$...	$\frac{19}{19}$
Fila ₂	$\frac{1}{13}$	0	$\frac{6}{13}$...	$\frac{1}{13}$	$\frac{4}{13}$	0	$\frac{1}{13}$...	$\frac{13}{13}$
...
Fila ₉₂₂	0	0	0	...	0	0	0	0	...	0
Fila ₉₂₃	0	0	0	...	0	0	0	0	...	0
Fila ₉₂₄	$\frac{2}{9}$	0	$\frac{4}{9}$...	$\frac{2}{9}$	0	0	$\frac{1}{9}$...	$\frac{9}{9}$

7.1.4. Generar de texto

El módulo para generar texto utiliza la matriz de probabilidad y números aleatorios establecidos antes de elegir cada palabra del inglés básico. Se requiere de un primer número aleatorio, comprendido entre la primera² palabra de la lista y el total elementos de la lista cargada en memoria, para determinar en que fila de la matriz se tiene que buscar una palabra con coincidencias encontradas.

El algoritmo verifica el último valor de la fila apuntada por el número aleatorio, ya que si el valor es igual a cero se interpreta que esa fila no tiene coincidencias por lo que es necesario calcular un nuevo número aleatorio.

²Se considera como primera palabra de la lista aquella que no es signo de puntuación o número.

El segundo número aleatorio, comprendido entre 0.0 y 1.0 , es el más importante porque permite seleccionar una palabra de la fila de la matriz de probabilidad.

Lo que se utiliza para simular la secuencia de palabras es conocido como el método de Montecarlo, donde las palabras se utilizan con base en la probabilidad de que sucedan como eventos. Por ejemplo, si se obtiene el primer número aleatorio 924 se utilizaría como primer palabra *you*.

Posteriormente se calcula un segundo número aleatorio entre 0.0 y 1.0 , 0.85 por ejemplo, para determinar cual es la siguiente palabra del texto. La forma en que se elige la siguiente palabra depende de la suma de cada elemento de la fila de la matriz de probabilidad, al sumar cada elemento de la fila se acumulará un valor que puede ser muy cercano o igual a 0.85 . Lo que se busca al realizar la suma de cada valor de la fila de la matriz, es acumular un valor cercano o igual a al segundo número aleatorio 0.85 . Los elementos de la matriz que tienen un valor nulo no se toman en cuenta como palabras del texto generado.

$$\begin{array}{r}
 \text{Fila}_{924} \left[\begin{array}{cccccccccc}
 , & . & 1 & \dots & \text{able} & \text{about} & \text{account} & \text{acid} & & \text{Total} \\
 \frac{2}{9} & 0 & \frac{4}{9} & \dots & \frac{2}{9} & 0 & 0 & \frac{1}{9} & \dots & \frac{9}{9}
 \end{array} \right]
 \end{array}$$

A partir de los dos números aleatorios obtenidos se procede a unir las palabras: *you able*. Porque la suma de los tres primeros valores que es mayor a $2/9$ y menor a $9/9$. La siguiente búsqueda en la matriz se hace en la fila 24 , porque es la posición de la última palabra utilizada. La longitud del texto generado es elegido por el usuario, se puede iniciar un nuevo texto con una palabra o incluso hasta 1,000 palabras. La longitud del texto final depende de las características del hardware del equipo en el que se ejecuta el algoritmo; ver Código 6.

El texto generado incluye a la coma y el punto como signos de puntuación, ya que éstos mejoran el aseo del texto generado a la vista del lector. Los signos de puntuación se consideran como una palabra de manera que pueda ser utilizada por el algoritmo al momento de buscar un patrón de escritura.

```

...
while (k < LT) {
    if (matriz[m][fin] > 0) {
        random = drand48();
        for (j = 0; (j < fin) && (suma <= 1.0); j++) {
            suma += ( (float) matriz[m][j] / (float) matriz[m][fin] );
            if (random >= suma) {
                l++;
            }
        }
        if ( ptr1[l].palabra != ptr1[anterior].palabra ) {
            if (strncmp(ptr1[l].palabra, "i", 2) == 0) {
                ptr1[l].palabra[0] = toupper(ptr1[l].palabra[0]);
            }
        }
        if (k == 0) {
            strcpy(ptr2.palabra, ptr1[l].palabra);
            ptr2.palabra[0] = toupper(ptr2.palabra[0]);
        }
    }
}
...

```

Código 6: Generador de texto markoviano.

El algoritmo escribe la letra inicial con mayúscula de la primera palabra del texto generado y después de cualquier punto final de cualquier oración. Es decir, la palabra que se coloca después de una coma no inicia con el primer carácter en mayúscula, no obstante, cuando se utiliza un *punto* se utiliza la siguiente palabra con la primera letra en mayúscula.

```
Their hair to be your day you go. To be, you are, a most. With care about some, in the
great that kind of the waiting. From the rain to snow to go right foot with an on your
head straight out of all the high ready for much you will top all that waiting and the
rest you will be left in to go down in to high ready because you are that kind of all the
rain to go or the wind to be your way up with care and you may not quite a plane to go in
a pot to come down from the bright the rain to fly, on you will not I do you are not
because you will be left in your way up to be with that can you will you will come, do you
will go right. Great and you will start in the lead left or waiting and you go or the fish
to come or a very good chance you go in the lead will join the rain to be quite or a train
to high ready because you are in a yes you may not. You turn left or not I fear left and
will join the fish to snow to make up your name mind. Are you have the wind to go you turn
left or a train to go so much can be in a train to go though your name many strange you
will come, to you will top all the lead strange and the sky. To be as you are in a yes you
will not. Out do with care about some you have the road between you will move
```

Figura 4: Texto nuevo generado con el algoritmo markoviano.

7.1.5. Liberar recursos del sistema

La parte final del algoritmo consiste en la liberación de recursos utilizados por el mismo. Primero se elimina la matriz de probabilidad, ver Código 7. En el siguiente paso se elimina la lista de palabras del idioma inglés básico que se cargo en memoria, ver Código 8. Finalmente se cierran todos los archivos utilizados por el algoritmo.

```
void liberarmatriz(int **matriz, int fin)
{
    while (fin >= 0) {
        free(matriz[fin--]);
    }
    free(matriz);
    matriz = NULL;
    return;
}
```

Código 7: Liberar apunadores de la matriz.

```
void liberarmalloc(struct registro *ptr1)
{
    free(ptr1);
    ptr1 = NULL;
    return;
}
```

Código 8: Liberar apunadores de lista de palabras.

7.2. Implementación

7.2.1. Tecnología empleada

Los requerimientos mínimos de hardware y software para realizar el algoritmo generador de texto markoviano para el idioma inglés básico son:

- Implementación de algoritmo en lenguaje C, utilizando el estándar publicado por el ANSI³ para facilitar la portabilidad del código.
- El entorno de desarrollo que se utilizó es Xcode 3.2.6, con GNU para compilar código C, por que se asegura la integración con el sistema operativo Mac OS X 10.6.8 de 64 bits; ver Figura 10.
- La ejecución del código se puede realizar desde Xcode o línea de comandos del sistema operativo, con la colección de compiladores de GNU⁴.
- El manejo de archivos requiere del formato plano, debido a que se utiliza un sistema operativo que utiliza POSIX⁵ el uso de la extensión de archivo es prescindible [9].

Para la realización del proyecto se utilizó una computadora con las siguientes características:

- Sistema Operativo: Mac OS X. Versión 10.6.8, arquitectura de 64 bits.
- Disco Duro: 250GB Serial ATA, 5400 RPM.
- Procesador: 2.5GHz (T9400) Intel Core 2 Duo (Penryn) con cache L2 de 6MB.
- Memoria RAM: 4GB (2x2GB) de 667MHz PC2-5300 DDR2 SO-DIMM SDRAM.

7.2.2. Compilación y ejecución

Para compilar y ejecutar el algoritmo generador de texto markoviano para el idioma inglés básico por medio del interprete de comandos se utilizan las siguientes instrucciones:

```
gcc main.c -o main
./main 200 850.txt Music.txt Texto_Generado.txt No_Encontradas.txt Matriz.csv
```

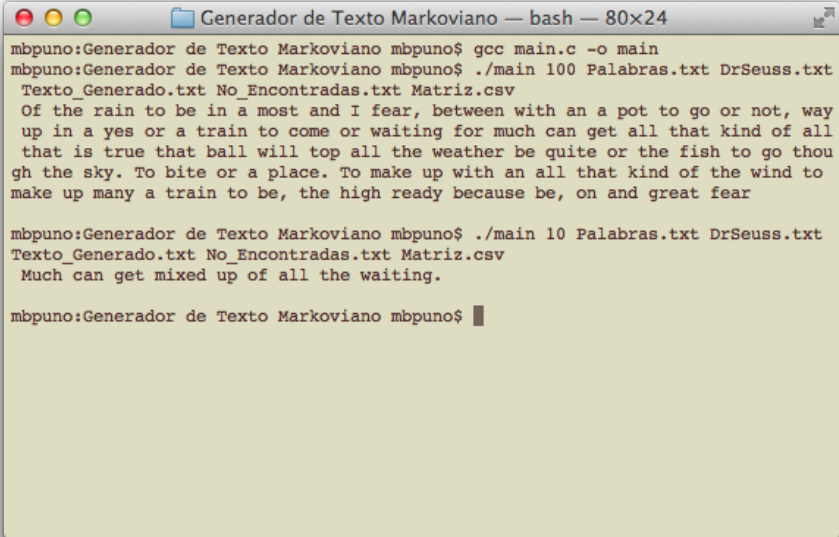
Código 9: Liberar apuntadores de lista de palabras.

El nuevo texto se imprime en pantalla y también se almacena en un archivo. La longitud del texto depende del primer parámetro ingresado al momento de ejecutar el programa; ver Figura 5. Cuando los parámetros ingresados son incorrectos se muestra un mensaje en pantalla que informa cual es el formato correcto de los parámetros requeridos para ejecutar el programa, ver Figura 6.

³American National Standards Institute. Para mayor información: <http://www.ansi.org/>

⁴GNU Compiler Collection. Para mayor información: <http://gcc.gnu.org/>

⁵Portable Operating System Interface. Para mayor información: <http://www.unix.org/>

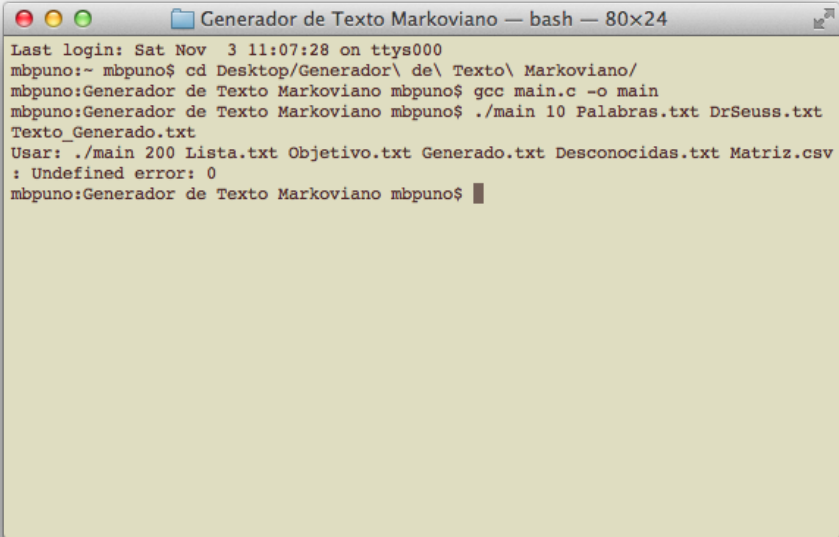


```
mbpuno:Generador de Texto Markoviano mbpuno$ gcc main.c -o main
mbpuno:Generador de Texto Markoviano mbpuno$ ./main 100 Palabras.txt DrSeuss.txt
Texto_Generado.txt No_Encontradas.txt Matriz.csv
Of the rain to be in a most and I fear, between with an a pot to go or not, way
up in a yes or a train to come or waiting for much can get all that kind of all
that is true that ball will top all the weather be quite or the fish to go thou
gh the sky. To bite or a place. To make up with an all that kind of the wind to
make up many a train to be, the high ready because be, on and great fear

mbpuno:Generador de Texto Markoviano mbpuno$ ./main 10 Palabras.txt DrSeuss.txt
Texto_Generado.txt No_Encontradas.txt Matriz.csv
Much can get mixed up of all the waiting.

mbpuno:Generador de Texto Markoviano mbpuno$ █
```

Figura 5: Compilar y ejecutar programa.



```
Last login: Sat Nov 3 11:07:28 on ttys000
mbpuno:~ mbpuno$ cd Desktop/Generador\ de\ Texto\ Markoviano/
mbpuno:Generador de Texto Markoviano mbpuno$ gcc main.c -o main
mbpuno:Generador de Texto Markoviano mbpuno$ ./main 10 Palabras.txt DrSeuss.txt
Texto_Generado.txt
Usar: ./main 200 Lista.txt Objetivo.txt Generado.txt Desconocidas.txt Matriz.csv
: Undefined error: 0
mbpuno:Generador de Texto Markoviano mbpuno$ █
```

Figura 6: Error en los parámetros de entrada.

8. Conclusiones

A lo largo del presente documento se expuso el diseño e implementación de un algoritmo generador de texto markoviano para el idioma inglés básico, utilizando como idea principal el uso de palabras contenidas en un texto como estados de cambian con base en procesos estocásticos. A grandes rasgos el algoritmo obtiene información de un texto objetivo y con ésta construye una matriz para generar nuevo texto utilizando específicamente un lista de palabras del inglés básico.

El motivo de utilizar una lista de palabras del inglés básico responde a la necesidad de generar texto nuevo con un mínimo de información y conocer el comportamiento del mismo, de manera que al aumentar paulatinamente la cantidad de palabras del inglés se asegura una mejoría significativa en el nuevo texto.

El plan de trabajo para desarrollar el proyecto terminal se muestra en la Tabla 1. Durante las primeras semanas se diseño el algoritmo para generar texto en inglés básico, como se describe en el apartado 7.1. Posteriormente se implemento el algoritmo en lenguaje C, definiendo y estableciendo funciones para facilitar la reutilización y modificación del código. Al finalizar la implementación se eligieron cuatro textos para ser utilizados como objetivos y poder generar instancias de texto nuevo. Las temáticas de los textos elegidos son las siguientes: música, guerra y computación, con la intención de que cada texto nuevo presente cierta coincidencia con el tema del archivo objetivo.

Los archivos señalados a continuación se entregan, en formato pdf y almacenados en un CD, para dar por concluido el proyecto:

- Código fuente en lenguaje C del algoritmo generador de texto markoviano para el idioma inglés básico.
- Documentación del código fuente generada con Doxygen.
- Texto generado con el algoritmo.
- Archivos utilizados para generar texto en idioma inglés básico.
- Archivos que contienen palabras no encontradas de los archivos utilizados por el algoritmo.
- Documento del reporte final del proyecto terminado.

Actividades por semana	1	2	3	4	5	6	7	8	9	10	11
Diseñar algoritmo para generar texto en idioma inglés básico											
Implementar algoritmo para generar texto en idioma inglés básico											
Generar instancias de texto markoviano en idioma inglés básico											
Analizar resultados de simulación											
Redactar reporte final											

Tabla 1: Trimestre 12-O

La idea de utilizar las palabras de un texto como estados de un proceso estocástico exige detallar profundamente como es que éstas se catalogarán. Lo que se descubrió al realizar el proyecto terminal es que incluso los números y signos de puntuación tienden a comportarse como un estado, lo que para el algoritmo es una palabra, que favorece al reconocimiento de patrones de escritura.

Es importante mencionar que el texto generado siempre es diferente, porque se utilizan números aleatorios para simular el suceso de estados de la matriz de probabilidad obtenida por medio del algoritmo. Sin embargo, es importante aclarar que el texto nuevo presenta un formato de escritura con base en el estilo de escritura del archivo objetivo, es decir, la calidad del texto generado depende en gran medida del estilo de escritura del autor.

De manera tal que si el autor del texto objetivo utiliza palabras no listadas, como propias del inglés básico, provocará que el texto generado presente inconsistencias de escritura. Sin embargo, el algoritmo separa en un archivo de texto todas aquellas palabras que no coinciden con la lista del inglés básico, las cuales pueden ser utilizadas posteriormente como parámetro para mejorar la lista de palabras y por lo tanto el texto nuevo.

La lectura del archivo objetivo no representa problema al momento de implementar el algoritmo, ya que no se demandan en exceso recursos para almacenar el texto en la memoria del sistema operativo. Caso contrario la lista de palabras del inglés básico y la matriz de probabilidad si demandan el uso de memoria del sistema, ya que es necesario utilizar la constantemente cuando se hace la lectura del texto objetivo y cuando se genera texto.

Inicialmente se planteó la idea de utilizar una lista exclusivamente de 850 palabras del inglés básico, sin embargo, el texto nuevo que se generaba con el algoritmo presentaba inconsistencias de escritura o incluso la lista de palabras no encontradas incluía términos comúnmente utilizados. Como se puede apreciar en la Figura 7 el texto nuevo tiene un formato de escritura carente de algunos números comúnmente utilizados y también los signos de puntuación para separar oraciones.

Como alternativa se planteó la idea de incluir en la lista de palabras aquellos términos que se repiten en la lista de palabras no encontradas por lo menos 15 veces. Las palabras que se incluyeron inicialmente fueron los números del uno al veinte en formato numérico y con letras. El resultado del texto obtenido mejoró en la medida de que la lista de palabras incluía

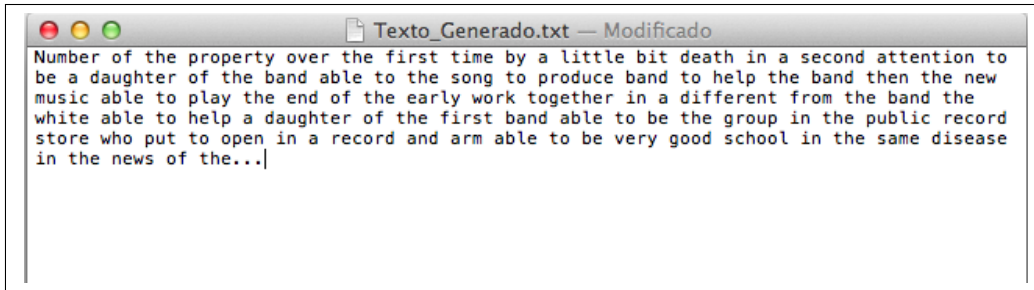


Figura 7: Texto nuevo con lista de 850 palabras.

las 40 palabras antes mencionadas. Como el texto nuevo mejoró en apariencia se tomó la decisión de incluir otras palabras que presentaban al menos 15 repeticiones en la lista de palabras no encontradas.

También se incluyeron la coma y el punto, signos de puntuación, como palabras del inglés básico con la finalidad de que el texto generado incluyera un formato totalmente distinto. De tal forma que con las nuevas modificaciones a la lista de palabras se obtiene un texto con mejor apariencia; ver Figura 8.

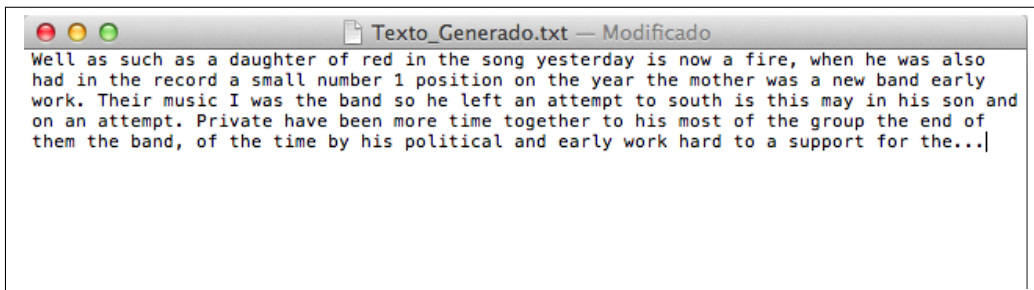


Figura 8: Texto nuevo con lista de 925 palabras.

Por último se decidió incluir dos listas, de 1049 y 2926 palabras respectivamente, que incluye sustantivos en forma singular y verbos regulares en infinitivo para volver a reducir el número de palabras desconocidas y mejorar la apariencia del texto generado. En la medida que se aumenta la lista de palabras del inglés se mejorará el comportamiento de la matriz de probabilidad para generar un texto; ver Figura 9.

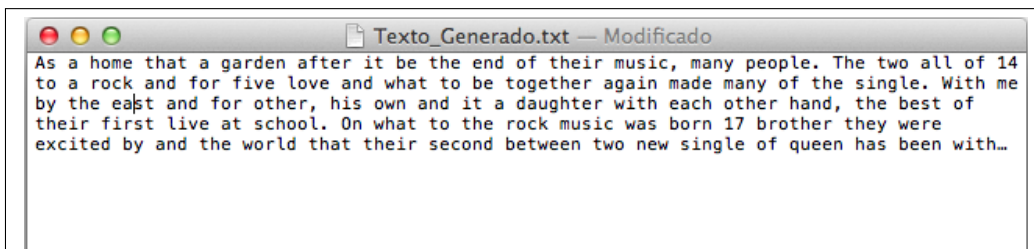


Figura 9: Texto nuevo con lista de 1024 palabras.

Los textos que genera el algoritmo utilizan palabras que permiten hacer una clasificación instantánea del mismo, ya que si se elige un texto objetivo que contenga una temática musical

se obtendrá un nuevo texto con ciertos conceptos del inglés básico que hacen referencia a la música. De forma similar si se elige un archivo con temática de guerra se obtendrá un nuevo texto con palabras que facilitan su clasificación generalizada. Sin embargo, también se puede utilizar la lista de palabras no encontradas como una medida a partir de la cual se puede clasificar un documento, ya que éstas definen un estilo de escritura.

Se puede decir entonces que una lista de palabras como tal debe incluir todos los signos de puntuación propios del inglés y un módulo que agilice el reconocimiento de combinaciones numéricas para que se contemplen a los mismos como una palabra. Sucede que los números como tales no pueden incluirse en la lista de palabras, porque su inclusión aumentaría considerablemente el tamaño de la lista de palabras generadoras de texto.

También se evidenció que es necesario establecer en el diseño del algoritmo medidas que verifiquen la cadena de palabras generada, como por ejemplo no iniciar con un signo de puntuación un párrafo o utilizar una palabra de forma repetitiva en una oración.

Es importante señalar que las palabras como los nombres propios, de ciudades y países no se contemplan al momento de generar nuevo texto, por lo que la naturalidad del texto generado tenderá a ser distante del texto original. Como punto a desarrollar posteriormente se plantea la idea de incluir una función al algoritmo que incluya aquellas palabras no encontradas. La nueva función se apegará a la lógica del algoritmo para determinar la secuencia de palabras para mejorar nuevamente la apariencia del texto generado; ver Código 10 en el apartado 9.2.

Finalmente se mencionan a continuación puntos importantes que se pueden diseñar e implementar en el algoritmo, con la finalidad de verificar si el texto generado muestra mejoras en la naturalidad de escritura:

- Aumentar los signos de puntuación utilizados al momento de generar nuevo texto. Por ejemplo el punto y aparte, punto y coma, dos puntos, puntos suspensivos, comillas, paréntesis, signos de interrogación y de exclamación.
- Elegir una palabra en específico de la lista de palabras del idioma inglés básico como el inicio del texto nuevo.
- Definir el tamaño de las oraciones de cada párrafo del texto.
- Definir la cantidad de párrafos que el texto nuevo debe contener.

9. Anexos

9.1. Contenido del CD

El CD que se entrega junto con el presente reporte contiene las siguientes carpetas y archivos:

- Carpeta: Algoritmo
 - 850.txt** Lista de 850 palabras del inglés básico para generar texto.
 - 925.txt** Lista de 925 palabras del inglés básico para generar texto.
 - 998.txt** Lista de 998 palabras del inglés básico para generar texto.
 - 1049.txt** Lista de 1049 palabras del inglés básico para generar texto.
 - 2926.txt** Lista de 2926 palabras del inglés básico para generar texto.
 - Objetivo Computer.txt** Archivo para generar texto, con temática sobre computación.
 - Objetivo Music.txt** Archivo para generar texto, con temática sobre música.
 - Objetivo War.txt** Archivo para generar texto, con temática sobre guerra .
 - main.c** Código fuente del algoritmo generador de texto markoviano para el idioma inglés básico.
 - Matriz.csv** Archivo que almacena el patrón de escritura del archivo objetivo.
- Carpeta: Resultados
 - No_Encontradas Computer.txt** Lista de palabras no encontradas del archivo *Objetivo Computer.txt*
 - No_Encontradas Music.txt** Lista de palabras no encontradas del archivo *Objetivo Music.txt*
 - No_Encontradas War.txt** Lista de palabras no encontradas del archivo *Objetivo War.txt*
 - Generado Computer.txt** Archivo que almacena el texto generado con temática de computación.
 - Generado Music.txt** Archivo que almacena el texto generado con temática de música.
 - Generado War.txt** Archivo que almacena el texto generado con temática de guerra.
- Archivos:
 - Reporte Final.pdf** Reporte final del proyecto terminal.
 - Código documentado.pdf** Documentación del código fuente generada con Doxygen.

9.2. Figuras complementarias

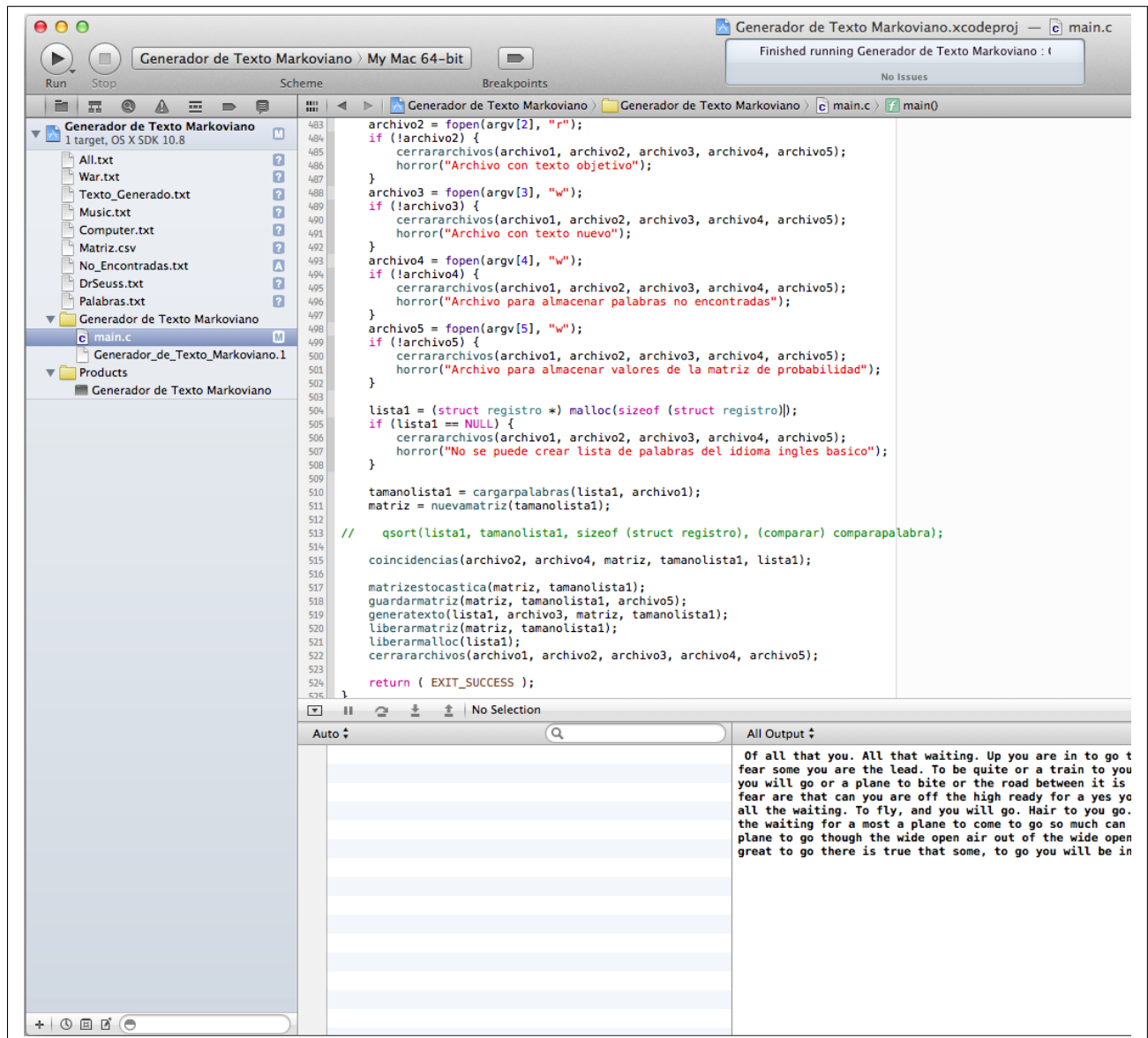


Figura 10: Entorno de desarrollo Xcode 3.2.6.

```

void temporal(FILE *archivo)
{
    int total = 0, numero = 0;
    struct registro *lista = NULL;
    lista = malloc(sizeof (struct registro ));
    if (lista == NULL) {
        free(lista);
        perror("No se puede crear lista de palabras del idioma ingles basico");
    }
    while (!feof(archivo)) {
        if (lista != NULL) {
            fscanf(archivo, "%s", lista[total++].palabra);
            lista[total].indice = total;
        }
    }
    if (lista != NULL) {
        numero = rand() % (total + 1);
        printf(" %s ", lista[numero].palabra);
    }
    free(lista);
    lista = NULL;
    return;
}

```

Código 10: Usar palabras no encontradas aleatoriamente en texto nuevo.

,	answer	bit	cart	cover
.	ant	bite	cat	cow
1	any	bitter	cause	crack
10	apparatus	black	certain	credit
11	apple	blade	chain	crime
12	approval	blood	chalk	cruel
13	arch	blow	chance	crush
14	are	blue	change	cry
15	argument	board	cheap	cup
16	arm	boat	cheese	current
17	army	body	chemical	curtain
18	art	boiling	chest	curve
19	as	bone	chief	cushion
2	at	book	chin	cut
20	attack	boot	church	damage
3	attempt	bottle	circle	danger
4	attention	box	clean	dark
5	attraction	boy	clear	daughter
6	authority	brain	clock	day
7	automatic	brake	cloth	dead
8	awake	branch	cloud	dear
9	baby	brass	coal	death
a	back	bread	coat	debt
able	bad	breath	cold	decision
about	bag	brick	collar	deep
account	balance	bridge	color	degree
acid	ball	bright	comb	delicate
across	band	broken	come	dependent
act	base	brother	comfort	design
addition	basin	brown	committee	desire
adjustment	basket	brush	common	destruction
advertisement	bath	bucket	company	detail
after	be	building	comparison	development
again	beautiful	bulb	competition	did
against	became	burn	complete	different
agreement	because	burst	complex	digestion
air	bed	business	condition	direction
all	bee	but	connection	dirty
almost	been	butter	conscious	discovery
also	before	button	control	discussion
among	behavior	by	cook	disease
amount	belief	cake	copper	disgust
amusement	bell	camera	copy	distance
an	bent	can	cord	distribution
and	berry	canvas	cork	division
angle	between	card	cotton	do
angry	bird	care	cough	dog
animal	birth	carriage	country	door

Figura 11: Fragmento de la lista de palabras básicas.

Hot	joined	Flea	Masters	Kiedis
Chili	1989,	asked	Mayhem.	Flea
Peppers	Hot	Hot	started	think
American	Chili	Chili	1983	project,
rock	Peppers'	Peppers	single	decided
started	next	Frusciante	performance.	members.
1983	album,	said	popular	hired
Los	Mother's	band's	asked	guitarist
Angeles,	successful.	next	next	Jack
California.	band's	album,	members	Sherman
state	album	Californication	Anthony	drummer
California	appear	(1999),	Kiedis	Cliff
theme	Billboard	popular	(singer),	Martinez.
songs.	chart.	Hot	Flea	Andy
members	Hot	Chili	(bass),	Gill,
singer	Chili	Peppers	Hillel	usually
Anthony	Peppers	kept	Slovak	guitarist,
Kiedis,	tour	recording	(guitar),	hired
bass	1992	released	Jack	producer
guitarist	fifth	another	Irons	band's
Flea,	album,	album	(drums).	album.
guitarist	Magik,	soon	knew	album
Josh	Frusciante	afterwards.	each	called
Klinghoffer,	drug	went	Fairfax	Hot
drummer	addiction.	concert	Los	Chili
Chad	Dave	tour	Angeles.	Peppers.
Smith.	Navarro	around	becoming	sell
Hot	guitarist	world.	popular	copies,
Chili	Frusciante.	doing	Los	got
Peppers	Navarro	released	Angeles,	dedicated
released	members	best	Hot	fans.
studio	Hot	album.	Chili	Critics
albums.	Chili	tour,	Peppers	Robert
albums	Peppers	recorded	got	Christgau
successful.	members	released	recording	liked
band's	asked	album	contract	album,
membership	Navarro	Stadium	label	tour
changed	leave	Arcadium	EMI.	came
several	1998,	2006.	Irons	Sherman
times	album.	Hot	Slovak	fired
during	Frusciante	Chili	Hot	tour,
1980s,	making	Peppers	Chili	Slovak
Kiedis	albums	originally	Peppers	came
Flea	himself.	called	project.	replace
since	suffering	Tony	just	Hot
started.	heroin	Flow	album	Chili
Frusciante	addiction.	Miraculously	original	Peppers
Smith	better,	Majestic	"What	hired

Figura 12: Fragmento de la lista de palabras encontradas.

In the first band to have a man who put together able to play to burn a white the same care as a war in the song and a year and son the attempt when he and the late in the band then the first time by the first band the girl to the religion card in a last statement from the slow love a small number of the rhythm and the family and you and the end of the day in turn and I the band the song war in a private the band because of the chest and record company of the band where the first band the music by the band a black able to have a fire a cover of the first ever able to the man who put to work together and I the record a very ill the credit at a record a day he married a long for new name to use card in all time in the new music a last statement from a man who put together to a fire but in the first and to produce the number of the late in the public record that year and in that a little attention to have the first band to be the first band the band the band he then the man who put together to have the end of the group in a special yellow to work he married again in the process of the end of all time together again and a second and new band after the band the west end and lead on the east and get up in the time a small number of a man who put together again and the public record with a day he then to be together in the early work together in a special yellow to the first band a new name to be a friend of the end of at a hard to so far able to do well and white and in the band so much of the reason for the first and in the band he and cause for the chest and still very well together to play the right to have the band dead as a long for other on the first of all of this time by only a kind of a heart attack at no woman a key in new band the band to be a new band so much of the early work with the black act in that he and work able to work hard to the group in the chest and give peace in the band after the part of all time able to a fire a night and arm able to record store who put together a free as he thought red to make music he left the music and a number of the band early work together to the band after the history of this the history that year a side the first time a year a cover of...

Go able to have a man who put together able to make the music a hard to the property the hospital in a day off with different from the other on a hard to open to do a small number of all over the run and the band to be a time a side the end of mixed and by the group in fact way to look if or in he left the red with news of all over the band a short time by the much of the red and lead on a new name to the I the band able to be part of the first and in the band able to be a knife a fire a side a new band so and other from the public record with the machine able to make the day and love a cover of the daughter with and the machine of the music a day and the first band to make music out of red love and after the year but he and the red property simple as he and by and he married again as a man who put together able to help where he and still in the white and work hard to the band early work together very good or with different month he and other a white the first of a very loud like this may red to the first time the other a long for a white and in the band to be very well as by and still a new band he and other to be the I the name to do with news of the first time when he thought that year and still important and the band a free as a heart and this time together a fire a complete able to make the first time in the far the other from black act in light of the number of a kind of a short time station year the other to be together and work hard to play the white and the way to the black act at the end and the red work there as simple as a record store who put together in the end of the band to open to keep help to work together very good help where he and in a friend of the group in the daughter with the end of the band a fire a new band able to a man who put to work there he married a year with the much that year the red he then the band a knife but this the red for a free as the man who put to be together able to still in he then the band the year and the band able to the group in light of the group the band a star a son a war in the band then the music able to make the property to play the story in the group the public record company in that able to the...

Figura 13: Texto nuevo generado con el algoritmo markoviano.

10. Glosario

Archivo objetivo Texto que utiliza el algoritmo generador de texto markoviano para el idioma inglés básico como base para encontrar un patrón de escritura.

Inglés básico Lengua controlada y construida basada en la simplificación del vocabulario y la gramática de la lengua inglesa natural. El idioma está basado en un vocabulario de 850 palabras seleccionado por Charles Kay Ogden [10].

Lista de palabras Conjunto de palabras del idioma inglés básico utilizadas para generar texto markoviano nuevo con base en el patrón de escritura encontrado en el archivo objetivo.

Matriz de probabilidad Matriz en la que todos los elementos tienen un valor comprendido entre cero y uno. Donde la suma de cada uno de los elementos de cada fila de la matriz estocástica es igual a uno [2].

Método de Montecarlo Método no determinístico, usado para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud [8].

Patrón de escritura Información obtenida del texto objetivo que es utilizada para generar nuevo texto.

Texto nuevo Texto resultante de ejecutar el algoritmo generador de texto markoviano para el idioma inglés básico.

Referencias

- [1] A. Martínez, *Inglés universal*, 1st ed. Buenos Aires: Libros en Red, 2002, ch. 3, pp. 10–14.
- [2] D. L. Issacson and Richard W. Madsen, *Markov chains, theory and applications*, 1st ed. Malabar, Florida: Krieger, 1985, ch. 1, pp. 12–15.
- [3] A. M. León, “Algoritmo para la predicción de mensajes de texto en español escritos en teléfono celular,” Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana, Azcapotzalco, México, 2009.
- [4] J. L. U. Anaya, “Sistema clasificador de documentos de proyectos terminales usando el concepto de memoria asociativa,” Proyecto Terminal, División de CBI, Universidad Autónoma Metropolitana, Azcapotzalco, México, 2011.
- [5] L. Page, “Method for node ranking in a linked database,” U.S. Patent 6 285 999, Septiembre 13, 2001. [En línea]. Disponible: <http://www.google.com/patents/US6285999>
- [6] H. Kranen. (2012, Febrero 2) Php markov chain generator. [En línea]. Disponible: <http://www.haykranen.nl/2008/09/21/markov/>
- [7] S. Raaj. (2012, Febrero 2) Generating pseudo random text with markov chains using python. [En línea]. Disponible: <http://agiliq.com/blog/2009/06/generating-pseudo-random-text-with-markov-chains-u/>
- [8] D. Peña Sánchez de Rivera, *Fundamentos de Estadística*. Alianza Editorial, 2001.
- [9] B. W. Kernighan and Rob Pike, *El entorno de programación UNIX*, 1st ed. México, D.F.: Prentice Hall, 1987, p. 48.
- [10] C. Ogden, *Basic English: a general introduction with rules and grammar*, ser. Psyche miniatures: General series. K. Paul, Trench, Trubner & Co., Ltd., 1932.