Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Proyecto Terminal

Algoritmo para la predicción de mensajes de texto en español escritos en teléfono celular

Alumno: Alejandro Morales León
Matricula: 204204789

Trimestres
09P - 09O

Asesor: Dr. Francisco Javier Zaragoza Martínez
Número económico: 20197

Índice de contenido

Introducción

Complementando la elaboración del proyecto terminal se desarrolló el presente reporte, en el cual es posible consultar los aspectos técnicos que se tomaron en cuenta para el diseño e implementación del sistema que tiene como objetivo la predicción de mensajes de texto en español escritos en teléfono celular.

Desarrollo del sistema

El sistema está compuesto principalmente por los siguientes módulos:

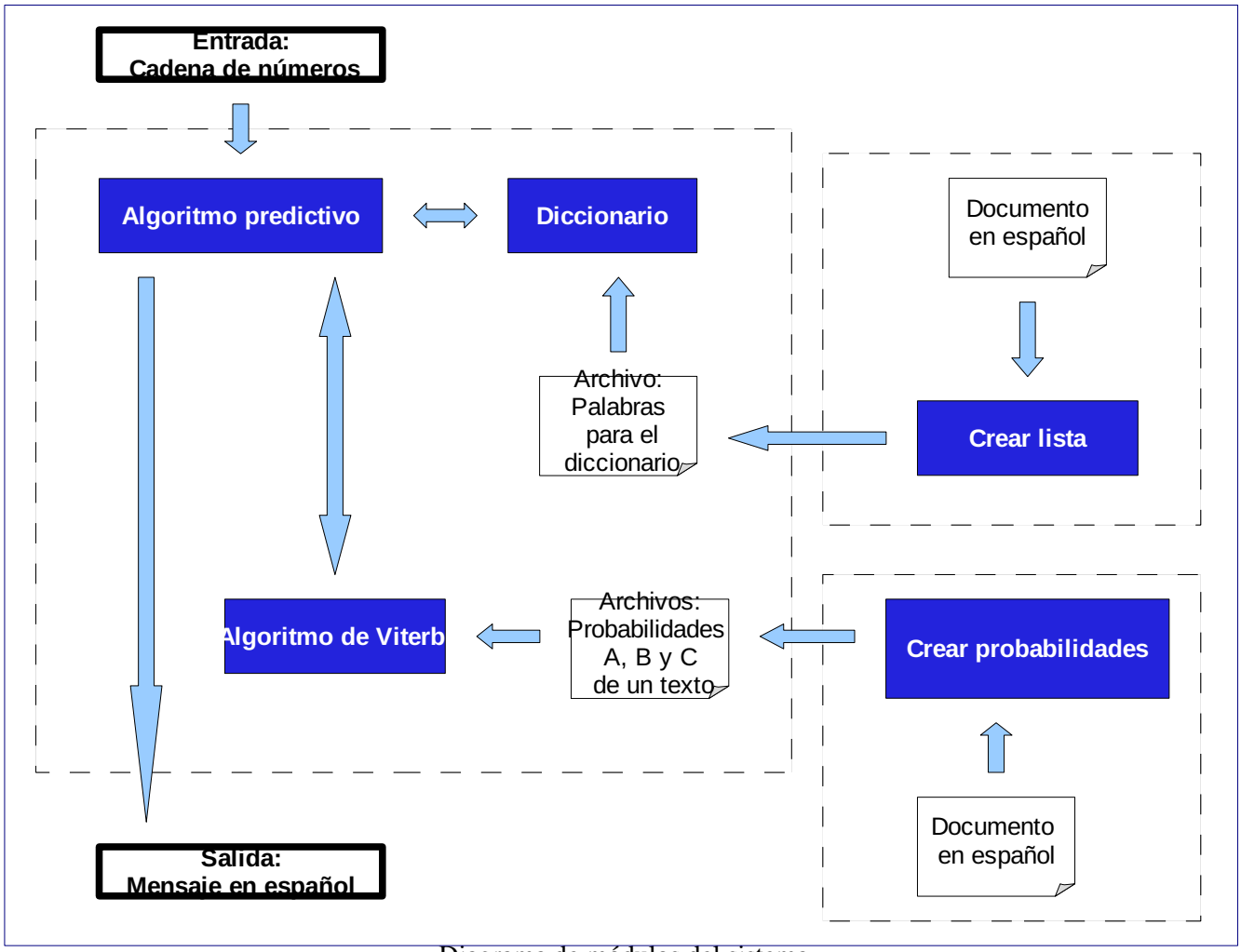


Diagrama de módulos del sistema

El sistema tiene la siguiente interacción con el usuario

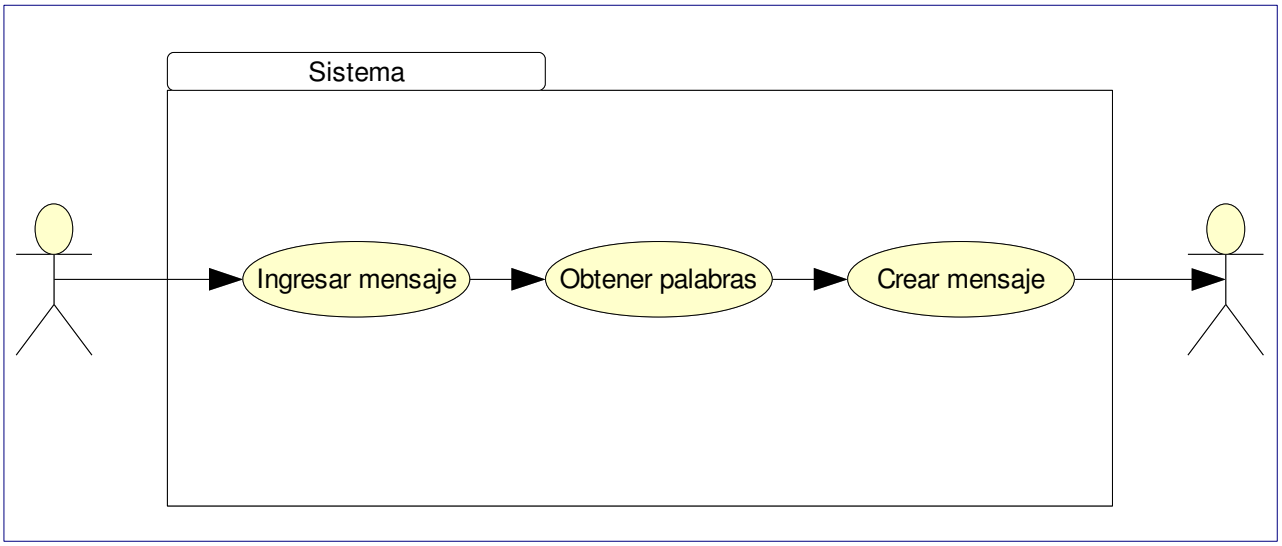


Diagrama de casos de uso

El usuario ingresa un mensaje codificado, puede ser por teclado o en un archivo, por lo que el sistema obtiene una lista de palabras correspondientes a cada código, y obtiene el mensaje correcto en español, escribiéndolo en un archivo de texto.

Entrada

La entrada al programa podrá ser de dos maneras, por teclado y en un archivo de texto, la cual consistirá de un conjunto de números separados por espacios, tabulaciones o nuevas líneas, los cuales corresponden a los códigos para teclado de celular.

```
4652 68636 7272 4642427
```

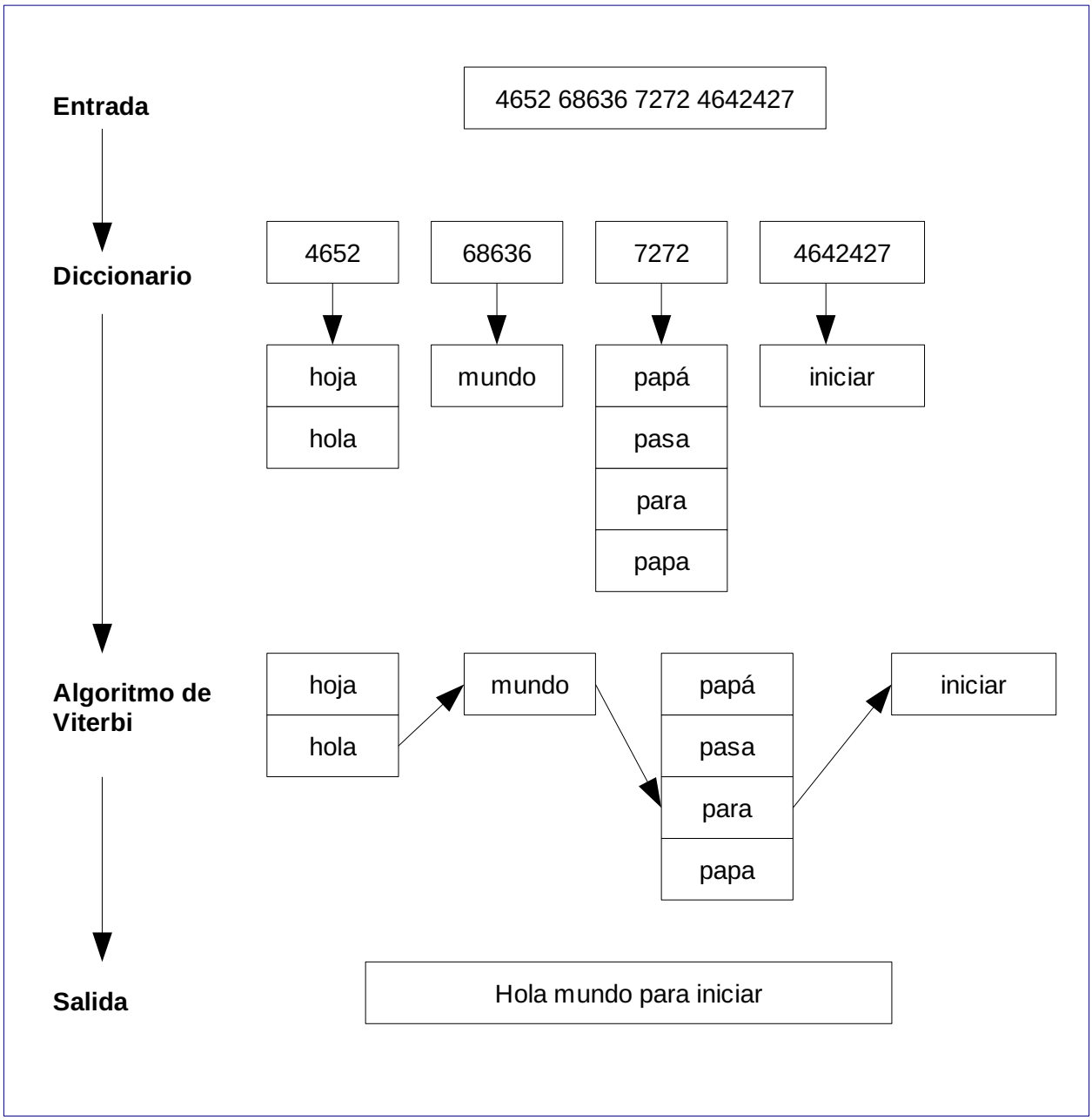
Entrada

Algoritmo predictivo

Este modulo de software es el encargado de recibir la entrada, por teclado o en archivo, que consiste en un conjunto de códigos, haciendo uso de la estructura de datos diccionario se obtendrán sus correspondientes palabras para cada código, y de entre los posibles caminos, con el algoritmo de

Viterbi se seleccionará el que mayor probabilidad tenga, y por ultimo genera un archivo de salida con el mensaje y la probabilidad de éste.

Si se ingresa algún código que no existe en nuestro diccionario, se colocará como su correspondiente palabra el mismo código. Como recomendación para proyectos futuros sería interesante encontrar palabras que pudieran ajustarse de alguna manera al código y hacer de esta manera un sistema más eficiente.



Algoritmo predictivo

Diccionario

Para el sistema tenemos la necesidad de mantener el acceso a un conjunto de códigos con sus determinadas palabras, de tal manera que dado un conjunto de elementos {k1, k2, ..., kN}, todos distintos entre sí, se desea almacenarlos en una estructura de datos que permita la implementación eficiente de las operaciones:

- búsqueda(k): dado un elemento k, conocido como llave de búsqueda, encontrarlo dentro del conjunto o decir que no está.
- inserción(k): agregar un nuevo elemento k al conjunto.
- eliminación(k): eliminar el elemento k del conjunto.

Estas operaciones describen al diccionario. Ahora tenemos el problema de cómo implementarlo, si no existieran limitaciones de espacio, se podría implementar simplemente utilizando la clave de un elemento como índice en una tabla de búsqueda directa (un arreglo); y si no existieran limitaciones en el tiempo de proceso, se podría implementar utilizando una lista enlazada con un requerimiento de espacio mínimo. Una tabla de dispersión es una estructura de datos que intenta encontrar un balance entre estos dos extremos, puesto que nuestro universo de datos podría crecer de manera significante y nuestra capacidad de proceso es mínima (posible futura implementación en celular), ésta se convierte en nuestra mejor opción. En lugar de utilizar la clave como un índice directamente, como sería el caso de una tabla de búsqueda directa, el índice se calcula a partir de la clave utilizando una función de acceso.

Aunque el costo de una búsqueda en una tabla de dispersión de n elementos en el peor de los casos es el mismo que el costo de una búsqueda en una lista enlazada $\Theta(n)$, en la práctica resulta mucho más eficiente. Pero el tiempo esperado de una búsqueda de un elemento en una tabla de dispersión es O(1).

Una tabla de dispersión *T* requiere tan solo $\Theta([K])$ espacio para almacenar un conjunto *K* de claves, un elemento con clave k se almacena en el espacio *h(k),* donde *h* es la función de acceso. La siguiente figura muestra la idea que se menciona. Para nuestro caso K es el conjunto de k códigos para celular

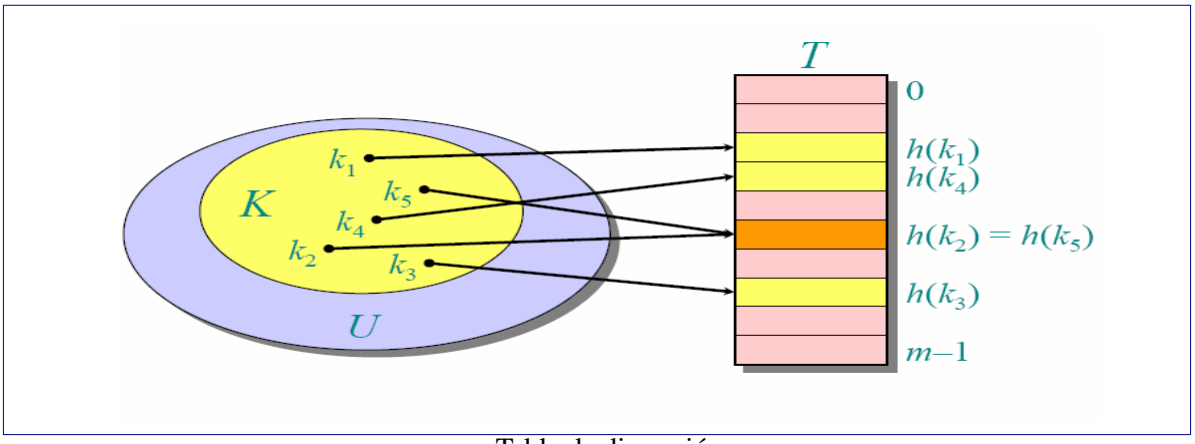
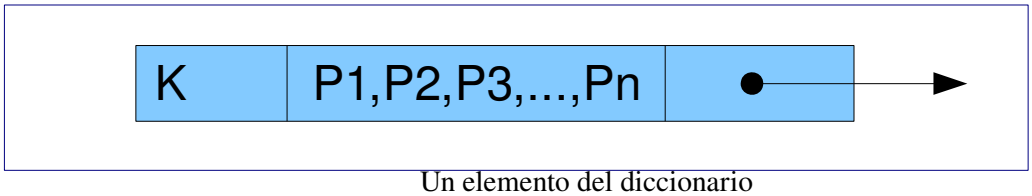


Tabla de dispersión

El punto es que la función de acceso reduce el rango de índices en el arreglo que se manipula. En lugar de $|U|$ valores, solamente se manipulan m índices, con el correspondiente ahorro de espacio.

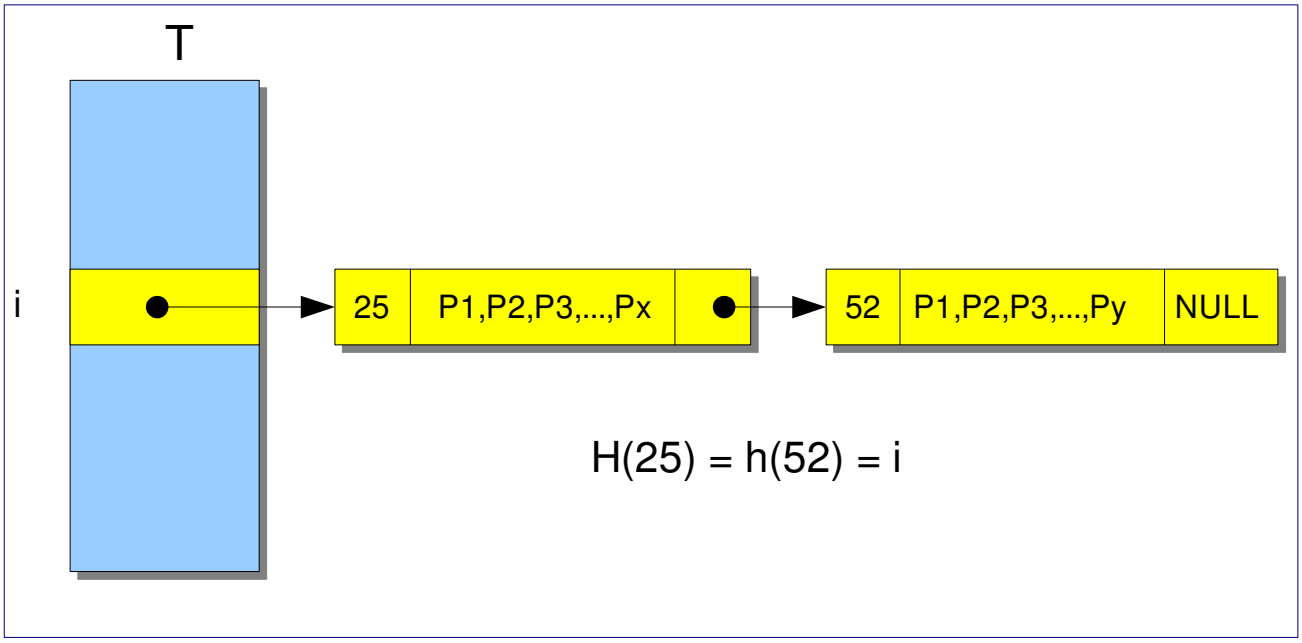
Cada elemento de nuestro diccionario, contiene la llave k correspondiente al código, su conjunto de n palabras P representadas por un arreglo dinámico de cadenas de caracteres, y un apuntador al siguiente elemento en el caso de existir más de uno que al aplicarle la función de acceso den la misma posición en la tabla de dispersión, a lo cual se le conoce como colisión.



Un elemento del diccionario

Resolución de colisiones por encadenamiento

La técnica de encadenamiento soluciona las colisiones insertando todos los elementos con el mismo valor de acceso en la misma celda, usando una lista enlazada. La celda i contiene sólo un puntero a la cabeza de la lista que contiene todos los elementos almacenados con valor de acceso i.



Técnica de encadenamiento

Podemos realizar las operaciones de inserción eliminación y búsqueda utilizando la función de acceso.

- **Inserción**
  Insertar elemento x en la cabeza de la lista con T [h(llave[x])]

- **Búsqueda**
  Buscar clave T [h(k)] en la lista

- **Eliminación**
  Eliminar elemento x con clave T [h(llave[x])] de la lista

La operación que nos es de mayor interés es la búsqueda pues ésta es la operación que determinará la eficiencia del sistema en términos del tiempo de ejecución. Para cada uno de los códigos que integran la entrada se requiere ejecutar una búsqueda en la tabla de dispersión siendo esta operación de gran importancia.

Análisis de búsqueda

Analicemos el factor de carga

$$\alpha = n / m$$

n = número de elementos en la tabla T
m = número de celdas en la tabla T o número de listas vacías

Este factor indica el número promedio de elementos por celda, pudiendo ser

$$\alpha<1 \qquad \alpha=1 \qquad \alpha>1$$

El peor caso para la tabla de dispersión, la cual hace uso de la técnica de encadenamiento es que todos los códigos almacenados tengan el mismo valor de acceso, para lo cual al haber almacenado todos los códigos integrantes del diccionario, tendremos una sola lista enlazada en alguna de las celdas que contiene todos los elementos y por lo tanto el costo de la búsqueda es $O(n)$.

El costo promedio depende de cuán uniformemente la función de acceso distribuya el conjunto de códigos entre las m celdas de la tabla que representa nuestro diccionario.

Si la tabla T tiene m celdas, entonces para j = 0,...,m-1 indicaremos que el número de elementos de determinado slots de la tabla T es la siguiente $T[j] = n_j$, entonces $n = n_1 + ... + n_{m-1}$

El valor promedio de n_j lo denotaremos por $E[n_j] = \alpha = n/m$
Si asumimos que podemos calcular la función de acceso en un tiempo de O(1) entonces el tiempo de búsqueda dependerá de la longitud de $n_{h(k)}$ en la posición de la tabla T[h(k)]

Existen dos casos de análisis

Búsqueda insatisfactoria, si no se encuentra el elemento con código k en la tabla T, y búsqueda satisfactoria, si se encuentra el elemento con código k en la tabla T

Búsqueda insatisfactoria

Teorema : Una búsqueda insatisfactoria toma el tiempo de $\theta(1 + \alpha)$

Prueba :

Para decir que el elemento con código k no se encuentra en la tabla T, buscaremos hasta el final de la lista T[h(k)], cuya longitud esperada es $E[n_{h(k)}] = \alpha$, Agregando el costo de calcular la función de acceso el tiempo esperado es $\theta(1 + \alpha)$.

Búsqueda satisfactoria

Teorema : Una búsqueda satisfactoria toma el tiempo de $\theta(1 + \alpha)$ en el caso promedio

Prueba:

Se asume que el código que se busca puede ser cualquiera de los n códigos almacenados en la lista con igual probabilidad. Como al insertar un elemento nuevo se agrega al principio de la lista de la celda correspondiente, el número promedio de comparaciones efectuadas durante una búsqueda de ese elemento es igual a 1 más el número de elementos subsiguientes que se insertan en esa celda.

Para encontrar el número promedio de elementos examinados al buscar un elemento se calcula el promedio de elementos que se insertaron posteriormente en la lista donde éste se insertó. Para encontrar el número promedio de comparaciones de cualquier búsqueda se promedia sobre los n elementos. Como la probabilidad de que un elemento se inserte en una celda cualquiera es 1/m, el costo C(n,m) de una búsqueda en la tabla es:

$$
\begin{aligned}
C(n,m) &= \frac{1}{n}\sum_{i=1}^{n}\left(1+\overbrace{\sum_{j=i+1}^{n}\frac{1}{m}}^{\text{elementos insertados posteriormente}}\right) && (1)\\
&= 1+\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=i+1}^{n}\frac{1}{m}\right) && (2)\\
&= 1+\frac{1}{n\,m}\sum_{i=1}^{n}(n-i) && (3)\\
&= 1+\frac{1}{n\,m}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right) && (4)\\
&= 1+\frac{1}{n\,m}\left(n^2-\frac{n(n+1)}{2}\right) && (5)\\
&= 1+\frac{n}{m}-\frac{n+1}{2m} && (6)\\
&= 1+\frac{n-1}{2m} && (7)\\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2\,n} && (8)\\
&= \Theta(1+\alpha) && (9)
\end{aligned}
$$

Comprobación de la búsqueda satisfactoria

A la conclusión que podemos llegar después de este análisis es que si el número de celdas en la tabla es al menos proporcional al número de elementos, entonces n = *O*(m) y

$$\alpha = n / m \leq c\ m / m = c = O(1)$$

Como una operación para insertar toma tiempo constante, entonces el costo de cualquier operación sobre un diccionario implementado con la técnica de encadenamiento es de orden *O(1)*.

Función de Acceso

Escoger una buena función de acceso quiere decir que la función de dispersión debe distribuir los datos uniformemente por la tabla, de tal manera que los valores de las claves no deben afectar esta distribución; lamentablemente en la práctica no podemos hacer esto pues es imposible conocer de antemano las distribuciones de probabilidad de las clave al insertarlas en una tabla.

La llave que se utiliza para obtener la dirección numérica en la tabla, es un número correspondiente a la codificación de las palabras en teléfono celular, el cual puede sobrepasar para algunos casos su capacidad de representación de números enteros de la máquina, para resolver esto, se almacena en una

cadena de caracteres, un número para cada posición en el arreglo. Se suman sus correspondientes códigos ascii y del resultante se obtiene el modulo de la división entre el total de posiciones en la tabla de dispersión. De esta manera se obtiene un entero comprendido entre 0 y m-1, con m = número de posiciones en la tabla.

Ejemplo: Utilizando la palabra "estudiante" en un diccionario con 517 posiciones
Su codificación para teléfono celular es 3788342683
sumando el código ASCII de cada elemento 51+55+56+56+51+52+50+54+56+51 = 532
obteniendo el modulo 532 % 517 = 15
La posición en la que se situará la palabra estudiante será en la 15 de la tabla de dispersión.

Comunicación entre el algoritmo predictivo y el diccionario

El algoritmo predictivo hace uso de la interfaz del diccionario, la cual incluye las siguientes funciones:

Función de acceso	int fa(void *arg, tMatrizHash dic);
Libera la memoria de un elemento	void liberarmem(void *arg);
Ingresar un elemento al diccionario	int hashIn(token *x, tMatrizHash mh);
Libera la memoria del diccionario	void EliminarDiccionario(tMatrizHash mh);
Compara dos llaves	int comparar(void *arg1, void *arg2);
Lee el archivo con las listas de los códigos y palabras y los almacena en una tabla de dispersión que representa al diccionario	void LlenarDiccionario(tMatrizHash *diccionario);
Busca en el diccionario un código determinado y obtiene sus correspondientes palabras	void *ObtenerPalabras(void *x, tMatrizHash mh);
Obtiene un número primo a partir del tamaño de la tabla de dispersión	int numeroPrimo(int n);
Crea una tabla de dispersión que representa al diccionario	void CrearDiccionario(tMatrizHash *mh);

En el siguiente diagrama de secuencia se puede apreciar mejor la comunicación

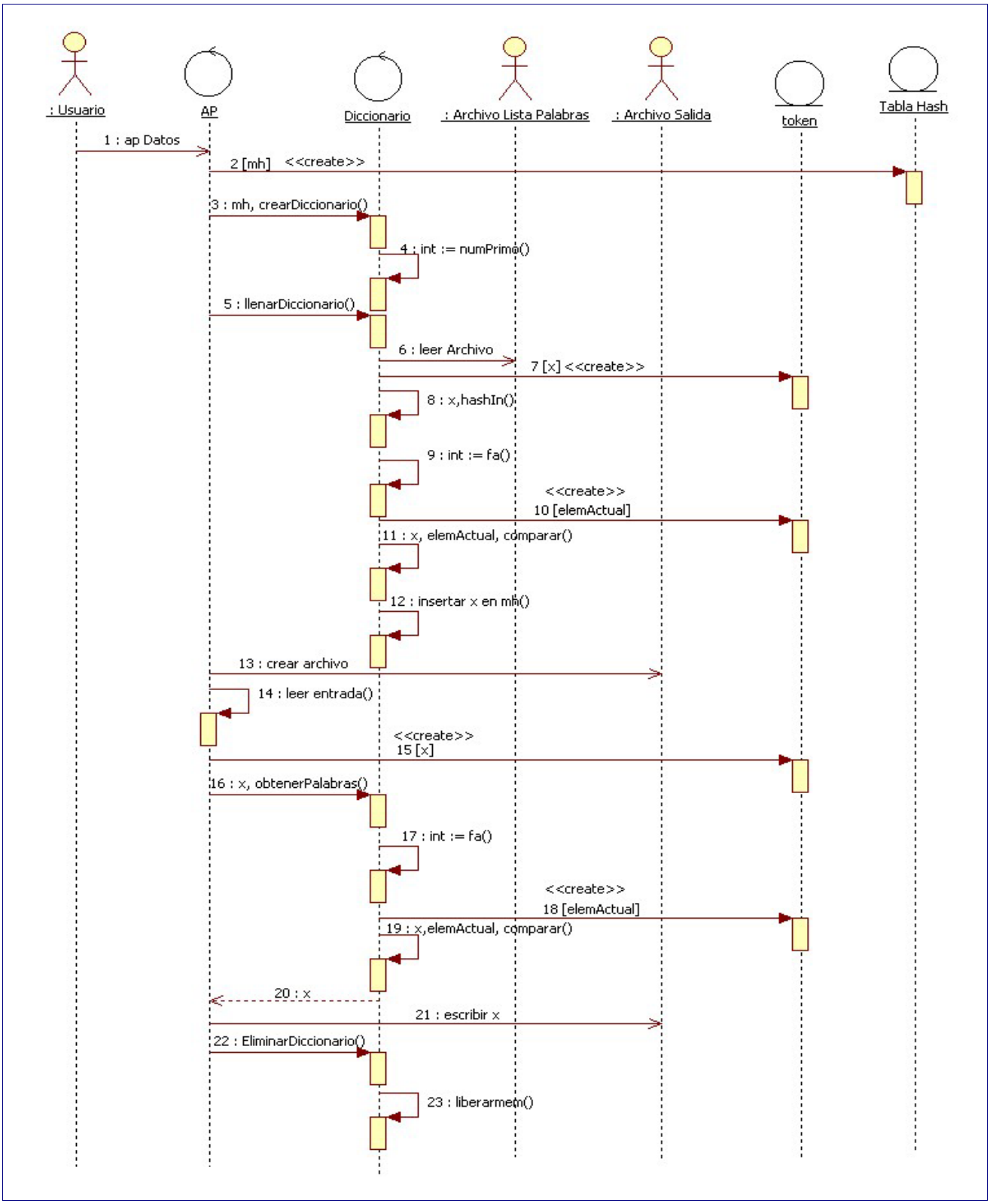


Diagrama de secuencia: ap y diccionario

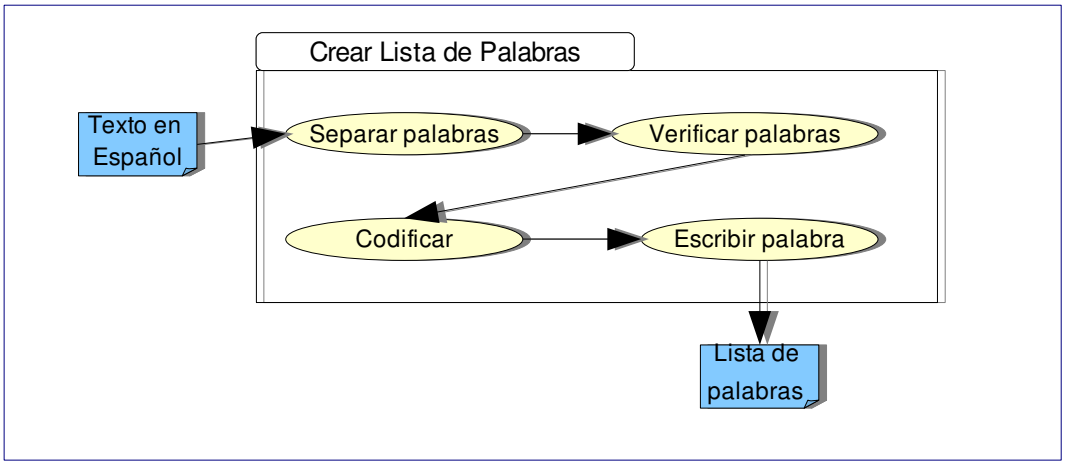
Lista de palabras utilizadas por el diccionario (Crear lista)

El sistema utiliza una lista de palabras y sus respectivas codificaciones en un teléfono celular, esta lista de palabras se selecciona dependiendo hacia qué usuarios está destinado y el uso que se le dará a dicha predicción, de tal manera que si se quiere utilizar en un contexto informal como son los mensajes de texto reales enviados entre los usuarios comunes, tendríamos que requerir una lista de palabras utilizadas por dichos usuarios, un ejemplo de este tipo de palabras sería:

Palabra:	*Palabra español:*
kom	como
t	te
q	que
xq	por que
dond	donde
x	por
tard	tarde
c	se
vakciones	vacaciones

Así que si queremos que el sistema sea dedicado a usuarios cuya forma de comunicación utilice palabras bien escritas del español tenemos que utilizar una lista con todas o las más utilizadas de este idioma.

Para propósito de este proyecto se utilizan palabras del idioma español obtenidas del periódico en línea *La Jornada.* Una vez que tenemos la fuente, hay que hacer una lista con todas las palabras del conjunto de estos documentos y obtener la codificación para teléfono celular de cada una de estas palabras, labor que podría ser muy ardua si se pretende utilizar una gran cantidad de documentos, de tal manera que se creo un módulo más, cuyo objetivo es automatizar esta tarea y de esta manera poder utilizar una gran cantidad y diversidad de documentos.



Casos de uso: Crear lista

Este modulo recibe un archivo de texto en español, separa las palabras para que queden aisladas de signos de puntuación, verifica si se trata de una palabra, esto quiere decir que contenga únicamente letras, a continuación obtiene la codificación de cada palabra y las escribe en un nuevo archivo, el cual podrá ser utilizado por el diccionario.

Es preciso recomendar que los números estén escritos con letra en este y todos los demás archivos que se utilicen para que el programa los tome en cuenta.

Algoritmo de Viterbi

El algoritmo de Viterbi es un caso particular del algoritmo de Programación Dinámica que permite encontrar las secuencia de estados más probable en un Modelo oculto de Márkov (MOM), $S = (q_1, q_2, \ldots, q_T)$, a partir de una observación $O = (o_1, o_2, \ldots, o_T)$, es decir, obtiene la secuencia óptima que mejor *explica* la secuencia de observaciones.

En el MOM se tienen las siguientes probabilidades:

- El conjunto de probabilidades A = {aij} de transiciones entre palabras, es decir, aij es la probabilidad de estar en la palabra j en el instante t si en el instante anterior t − 1 se estaba en el la palabra i.
- El conjunto de probabilidades B = {bj} de emisión de palabras, es decir, la probabilidad de observar una determinada palabra para cierto código.
- Las probabilidades iniciales C = {ci}, donde ci es la probabilidad de que el primer estado sea el estado Qi, es decir, la probabilidad de que las palabras correspondientes al primer código estén en primera posición.

Probabilidades A

Probabilidades de transición entre palabras, de manera que, en cada instante de tiempo las palabras correspondientes son distintas dependiendo del código, se necesitó una matriz de tres dimensiones para almacenarlas Aijk donde:

- i representa el número de transiciones entre los códigos que integran el mensaje
- j representa el número de palabras en un instante t-1
- k representa el número de palabras en un instante t

Tenemos que P(w,x)= Ap/Tot, la probabilidad de estar en la palabra w en el instante t si en el instante anterior t-1 se estaba en la palabra x, esto es igual a Ap/Tot donde Ap es el número de apariciones de la transición (w,x) entre el número total de las transiciones de x con las demás palabras.

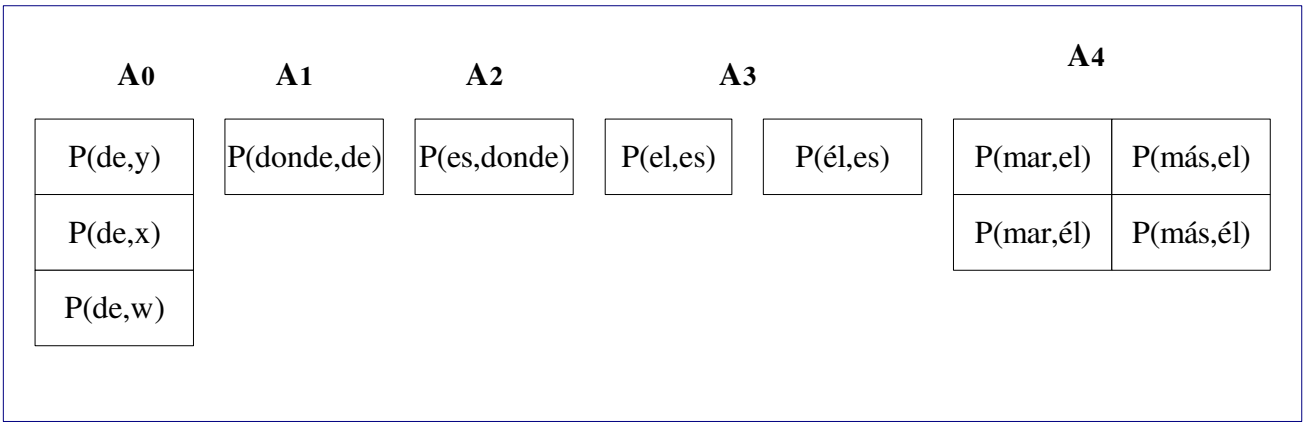
Ejemplo. Si tenemos lo siguiente como entrada:

9 33 36633 37 35 627

Las palabras correspondientes son

9 y x w
33 de
36633 donde
37 es
35 el él
627 mar más

por lo tanto vamos a tener la siguiente matriz A



Matriz de probabilidades A

De no existir ninguna de las probabilidades en alguna de las transiciones entonces se les coloca probabilidad 1/Ntransiciones, donde Ntransiciones es el número de transiciones que se generan con las palabras de dos códigos. En el ejemplo anterior, si todas las probabilidades de A0 fueran 0 entonces a cada una se le asignaría 1/3.

Probabilidades B

Probabilidades de emisión de una determinada palabra, en este caso se necesito de una matriz de dos dimensiones Bij, donde:

- i representa el número de códigos que integran el mensaje
- j representa el número de palabras para cada código

Tenemos que P(w)= Ap/Tot, la probabilidad de observar una determinada palabra w para cierto código. Esto es igual a Ap/Tot donde Ap es el número de apariciones de la palabra w entre el número total de apariciones de todas las palabras correspondientes al mismo código.

Continuando con el ejemplo anterior se crea la siguiente matriz B

i \ j	0	1	2
0	P(y)	P(x)	P(w)
1	P(de)		
2	P(donde)		
3	P(es)		
4	P(el)	P(él)	
5	P(mar)	P(más)	

Matriz de probabilidades B

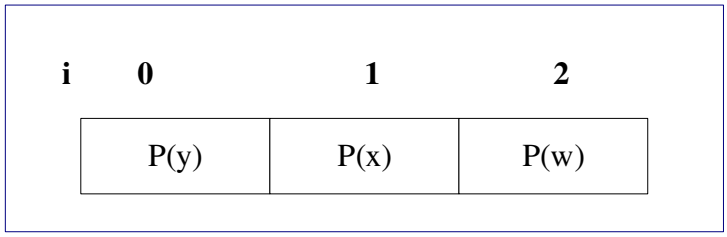
Probabilidades C

La probabilidad para las palabras correspondientes al primer código que estén en primera posición, de manera que para el texto ingresado únicamente se considerará el primer código como el inicial, las palabras que correspondan a éste tendrán dicha probabilidad, así que se necesita de un vector Ci para almacenar esta información, donde

- i representa el número de palabras para el código inicial

Tenemos que P(w)= Ap/Tot, la probabilidad de observar una determinada palabra w para el código inicial, esto es igual a Ap/Tot donde Ap es el número de apariciones de la palabra w al inicio entre el número total de apariciones al inicio de todas las palabras correspondientes al mismo código.

Continuando con el ejemplo anterior se obtiene el siguiente vector C



Matriz de probabilidades C

De no existir ninguna de las probabilidades correspondientes a las palabras de un código entonces se les coloca probabilidad 1/Npalabras, donde Npalabras es el número de palabras correspondientes al código inicial. Si en el ejemplo anterior las probabilidades de y, *x* y *w* fueran 0 entonces se les coloca probabilidad 1/3.

Algoritmo

Consideremos la variable $\delta_t(i)$ que se define como:

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_t = i, o_1, o_2, \ldots, o_t|\mu)$$

$\delta_t(i)$ es la probabilidad del mejor camino hasta el estado *i* habiendo visto las *t* primeras observaciones.

Esta función se calcula para todos los estados e instantes de tiempo.

$$\delta_{t+1}(i) = [\max_{1 \leq j \leq N_t} \delta_t(j) a_{tji}] b_{t+1i}$$

Puesto que el objetivo es obtener las secuencia de estados más probable, será necesario almacenar el argumento que hace máxima la ecuación anterior en cada instante de tiempo *t* y para cada estado *j* y para ello utilizamos la variable . $\varphi_t(i)$

Inicialización

$$\delta_1(i) = c_i b_{1i}$$

donde $1 \leq i \leq N_1$

Recursión

$$\delta_{t+1}(i) = [\max_{1 \leq j \leq N_t} \delta_t(j) a_{tji}] b_{t+1i}$$

donde t=1, 2, ... , T-1, $1 \leq i \leq N_{t+1}$

$$\varphi_t(i) = \arg \max_{1 \leq j \leq N_t} \delta_t(j) a_{tji}$$

donde t=1, 2, ... , T-1, $1 \leq i \leq N_{t+1}$

Terminación

$$q_T = \arg \max_{1 \leq j \leq N_T} \delta_T(j)$$

Reconstrucción de la secuencia más probable

$$q_t = \varphi_t(q_{t+1})$$

donde t=T-1, T-2, ... , 1

Ejemplo

Para el ejemplo anterior: 9 33 36633 37 35 627

Obtenemos las siguientes matrices de probabilidades

A0	**A1**	**A2**	**A3**		**A4**	
2/153	1/1	1/1	2/37	0/37	0/183	1/183
0/153					0/1	0/1
0/153						

Matriz de probabilidades A

i \ **j**	**0**	**1**	**2**
0	155/157	1/157	1/157
1	407/407		
2	8/8		
3	38/38		
4	184/185	1/185	
5	4/36	32/36	

Matriz de Probabilidades B

i	**0**	**1**	**2**
	23/23	0/23	0/23

Matriz de Probabilidades C

Inicialización:

$\delta_1(1)=1*(155/157)=155/157$
$\delta_1(2)=0*(1/157)=0$
$\delta_1(3)=0*(1/157)=0$

Recursión:

$\delta_2(1)=max[(155/157)*(2/153),\ 0*0,\ 0*0]*(407/407)=310/24021$
$\varphi_1(1)=1$

$\delta_3(1)=max[(310/24021)*1]*(8/8)=310/24021$
$\varphi_2(1)=1$

$\delta_4(1)=max[(310/24021)*1]*(38/38)=310/24021$
$\varphi_3(1)=1$

$\delta_5(1)=max[(310/24021)*(2/37)]*(184/185)=114080/164423745$
$\varphi_4(1)=1$

$\delta_5(2)=max[(310/24021)*(0/37)]*(1/185)=0$
$\varphi_4(2)=1$

$\delta_6(1)=max[(114080/164423745)*(0/183),\ 0*(0/1)]*(4/36)=0$
$\varphi_5(1)=1$

$\delta_6(2)=max[(114080/164423745)*(1/183),\ 0*(0/1)]*(32/36)=3650560/1083223632060$
$\varphi_5(2)=1$

Terminación

$q_6=arg\ max[0,\ 3650560/1083223632060]= 2$

Reconstrucción de la secuencia más probable

$q_5= \varphi_5(q_6)= \varphi_5(2)=1$
$q_4= \varphi_4(q_5)= \varphi_4(1)=1$
$q_3= \varphi_3(q_4)= \varphi_3(1)=1$
$q_2= \varphi_2(q_3)= \varphi_2(1)=1$
$q_1= \varphi_1(q_2)= \varphi_1(1)=1$

i \ j	1	2	3
1	155/157	0	0
2	310/24021		
3	310/24021		
4	310/24031		
5	114080/164423745	0	
6	0	3650560/ 1083223632060	

δt(i) Probabilidad del mejor camino

i \ j	1	2	3
1	y	x	w
2	de		
3	donde		
4	es		
5	el	él	
6	mar	más	

Palabras del posible mensaje

De manera que la secuencia reconstruida es qt= 1, 1, 1, 1, 1, 2 , nuestro mensaje será:

“ y de donde es el más “

Con una probabilidad de: 0.00000337

Crear probabilidades

El algoritmo de Viterbi hace uso de probabilidades, las cuales ya se han mencionado anteriormente (matriz A, matriz B y vector V). Dichas probabilidades se obtienen de documentos en español y que además las palabras que lo integran deben de estar en el diccionario para que el algoritmo predictivo las tome en cuenta, de lo contrario éstas serán omitidas y sus probabilidades no tendrán sentido.

Para tener un seguimiento adecuado de acuerdo al diccionario, se utilizaron documentos obtenidos del periódico en línea *La Jornada.* Una vez que tenemos un archivo con varias noticias hay que generar tres archivos, uno para cada matriz de probabilidades (A,B y C).

Probabilidades A

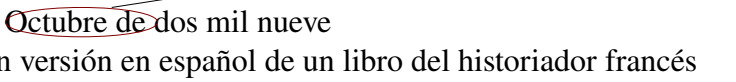
Para generar las probabilidades A, fue necesario separar el documento en oraciones, las cuales se delimitan por signos de puntuación, a continuación se realiza un conteo de todas las transiciones entre palabras, contando como transición al conjunto de dos palabras que se encuentran dentro de una oración. Una vez que se tiene el registro de todas las transiciones se almacenan en un archivo, colocando cada una de las palabras en forma de lista y enseguida de cada una se colocan las palabras con las cuales éstas tienen transición y el número de veces que se observaron, al final de cada línea se coloca el número total de transiciones de la palabra correspondiente a la línea. A continuación se ve un ejemplo:

Documento de La Jornada

Trece de Octubre de dos mil nueve

■ Presentan versión en español de un libro del historiador francés
“ No juzgo a la fotografía ; sólo constato su expansión y eficacia ” : Michel Frizot

Separar en oraciones



Trece de Octubre de dos mil nueve
Presentan versión en español de un libro del historiador francés
No juzgo a la fotografía
sólo constato su expansión y eficacia
Michel Frizot

Archivo generado

trece de 1 1
de octubre 1 dos 1 un 1 3
octubre de 1 1
dos mil1 1
mil nueve 1 1
nueve 0
presentan versión 1 1
versión en 1 1
en español 1 1
español de1 1
un libro 1 1
libro del 1 1
del historiador 1 1

historiador francés 1 1
francés 0
no juzgo 1 1
juzgo a 1 1
a la 1 1
la fotografía 1 1
fotografía 0
sólo constato 1 1
constato su 1 1
su expansión 1 1
expansión y 1 1
y eficacia 1 1
eficacia 0
michel frizot 1 1
frizot 0

Probabilidades B

De igual manera el documento se separa en oraciones pero para obtener estas probabilidades se suma el número de veces que aparecen las palabras en el texto y a continuación se crea el archivo y se escriben las palabras en forma de lista y enfrente de cada palabra se coloca el número de apariciones. Continuando con el ejemplo anterior obtenemos el siguiente archivo

trece 1
de 3
octubre 1
dos 1
mil 1
nueve 1
presentan 1
versión 1
en 1
español 1
un 1
libro 1
del 1
historiador 1
francés 1
no 1
juzgo 1
a 1
la 1
fotografía 1

sólo 1
constato 1
su 1
expansión 1
y 1
eficacia 1
Michel 1
Frizot 1

Probabilidades C

Para crear este archivo se utilizan las oraciones separadas y se sigue el mismo proceso que en las probabilidades B pero con la diferencia que ahora solo se cuentan las apariciones de palabras al inicio de las oraciones y obtenemos el siguiente archivo:

trece 1
de 0
octubre 0
dos 0
mil 0
nueve 0
presentan 1
versión 0
en 0
español 0
un 0
libro 0
del 0
historiador 0
francés 0
no 1
juzgo 0
a 0
la 0
fotografía 0
sólo 1
constato 0
su 0
expansión 0
y 0
eficacia 0
Michel 1
Frizot 0

Salida

La salida es generada por el *Algoritmo predictivo y es* almacenada en un archivo *salida.txt* la cual esta conformada por un texto en español cuya probabilidad fue la máxima, de igual manera se muestra dicha probabilidad.

Ejemplo de salida para la entrada: 9 33 36633 37 35 627 4676782683

y de donde es el más importante

probabilidad del texto: 1.053153E-07

Pruebas

Pruebas de Crear lista

Se obtuvieron varios documentos que corresponden a noticias en texto plano de la página web www.jornada.unam.mx, los cuales sirvieron como entrada al modulo para obtener la lista de los códigos y palabras, de primera instancia se opto por utilizar documentos pequeños para revisar de manera manual que todas las palabras del texto estuvieran en el diccionario creado. Una vez que se comprobó con varios documentos, éstos se integraron en uno solo para verificar que funcionara de manera correcta con documentos de mayor tamaño, para lo cual se obtuvo un resultado satisfactorio.

Ejemplo de noticia y su diccionario creado a partir de ésta:

13 de Octubre de 2009
■ Presentan versión en español de un libro del historiador francés
“No juzgo a la fotografía; sólo constato su expansión y eficacia”: Michel Frizot
■ Se trata de un proceso que permite trascender la noción de arte, expresa a La Jornada
■ En mi país, en cada época aparece la inquietud respecto de si esa disciplina es un arte, dice

Diccionario:

33 de
6288273 octubre
773736826 presentan
8377466 versión
36 en
3772665 español
86 un
54276 libro
335 del
44786742367 historiador
3726237 francés
66 no
58946 juzgo
2 a
52 la
3686472342 fotografía
7656 sólo
26678286 constato
78 su
397267466 expansión
9 y
33422242 eficacia
642435 michel
374968 frizot

73 se
87282 trata
7762376 proceso
783 que
7376483 permite
8727236337 trascender
662466 noción
2783 arte
3977372 expresa
5676232 jornada
64 mi
7247 país
2232 cada
37622 época
2727323 aparece
467843883 inquietud
73773286 respecto
74 si
372 esa
3472475462 disciplina
37 es
3423 dice

Pruebas de Crear probabilidades

Los documentos que se utilizaron son los mismos que se obtuvieron de *La Jornada* en línea. De igual manera que las pruebas anteriores, se utilizaron documentos pequeños para verificar que todas las palabras estuvieran presentes en los tres documentos y manualmente verificar que las palabras se estuvieran contando de manera correcta, una vez que se comprobó que el programa funcionaba de manera correcta para casos de entrada pequeños, se creo un solo documento integrando todos. Una manera de verificar que todas las palabras estén en los archivos, es utilizar los mismos documentos que se utilizaron en *Crear lista* y comparar el diccionario con dichos archivos, pues la lista de palabras que generan son muy parecidas.

Pruebas del algoritmo predictivo y del diccionario

Las pruebas que se le realizaron a estos dos programas fue en conjunto pues los resultados que da el *algoritmo predictivo* dependen en gran medida del resultado que genere el *Diccionario.*

El módulo debe de generar una lista con los códigos de entrada y sus respectivas palabras en caso de existir tales y el mismo código si no existen palabras correspondientes a él, por lo tanto para comprobar un buen funcionamiento es necesario incluir en la entrada estos dos casos.

Utilizamos como entrada el mismo conjunto de códigos que integran el diccionario y se le agregó un conjunto de códigos que no lo integran para lo cual se obtuvo de manera correcta una copia del diccionario más el conjunto de códigos que no están en él. Dicho diccionario incluye distintos casos que pudieran ser de motivo de error, como:

Códigos con solo una palabra correspondiente. 33 de

Con más de una. 3323 debe edad

Códigos que colisionen con la función de acceso 335 del y 227 bar (fa=155)

Ejemplo del diccionario completo como entrada, por supuesto la entrada real es de un tamaño mucho mayor :

Diccionario	Entrada	Salida
33 de 6288273 octubre 773736826 presentan 8377466 versión 36 en 3772665 español 86 un 54276 libro	33 6288273 773736826 8377466 36 3772665 86 54276 22	33 de 6288273 octubre 773736826 presentan 8377466 versión 36 en 3772665 español 86 un 54276 libro 22 22

Al igual que la prueba en la que se utiliza todo el diccionario se realizaron pruebas con diferentes entradas, desde archivo y desde teclado, en las cuales se incluyen códigos existentes e inexistentes en el diccionario obteniendo un resultado correcto en todos los casos.

Pruebas de la integración del programa con el algoritmo de Viterbi

Una vez integrado el programa con el algoritmo de Viterbi se probó con varias entradas pequeñas, las cuales se realizaron a mano para comparar los resultados, un ejemplo de ellos lo tenemos en la sección *Algoritmo de Viterbi – Ejemplo,* en ejemplos de mayor tamaño (hasta 100 Kb) sólo me limité a verificar que el programa no tuviera problemas con la memoria, o existiera algún error obvio

Aquí tenemos alguno de los ejemplos pequeños:

entrada:

3783 37 86 3536756 33 778322 7272 72237 74 8636 3782 8722252636 33 626372 334229

salida:

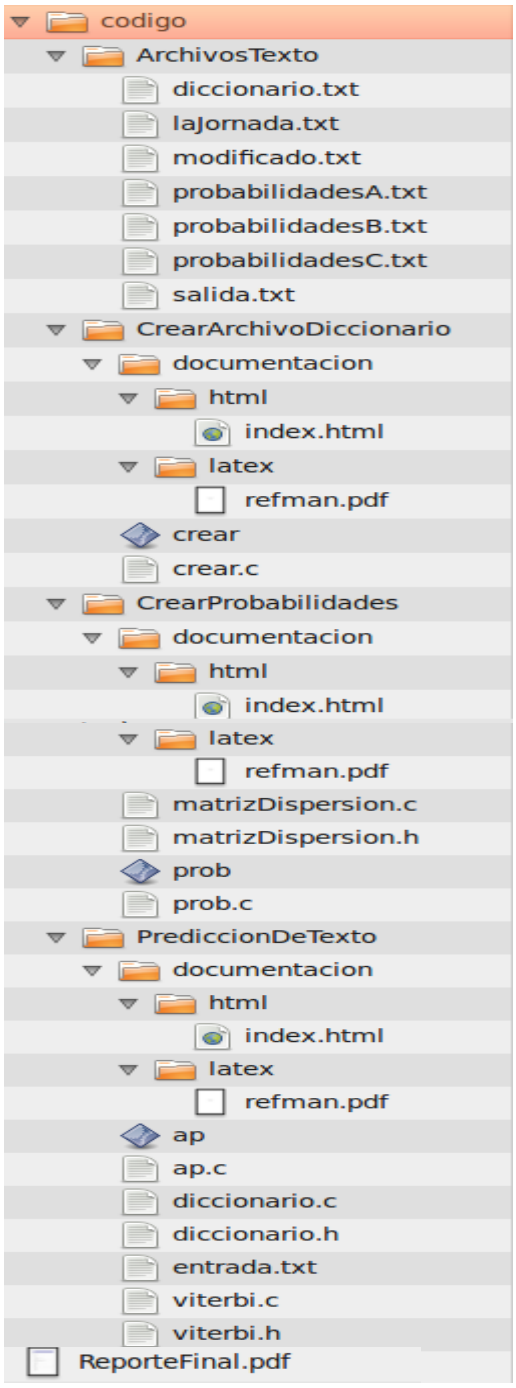
este es un ejemplo de prueba para saber si todo está trabajando de manera eficaz

probabilidad del texto: 2.204778E-08

Anexos

Contenido del CD

El CD que se entregará junto con el presente reporte tendrá las siguientes carpetas y archivos importantes:



codigo: Contiene el código fuente y su documentación

- **ArchivosTexto:** Contiene los archivos de texto
  - **diccionario.txt:** contiene la lista de códigos de celular y sus respectivas palabras
  - **lajornada.txt: c**ontiene noticias de *La Jornada* en línea
  - **modificado.txt:** igual que el anterior pero con los signos de puntuación separados
  - **probabilidadesA.txt:** contiene las probabilidades de transición
  - **probabilidadesB.txt:** contiene las probabilidades de emisión
  - **probabilidadesC.txt:** contiene las probabilidades iniciales
  - **salida.txt:** contiene la salida del programa
- **CrearArchivoDiccionario:** contiene el módulo que crea la lista para el diccionario
  - **documentacion:** documentación del código fuente de este módulo
    - **html:** documentación en html
      - **index.html:** pagina html con la documentación de este módulo
    - **latex:** documentación en latex y pdf
      - **refman.pdf:** documento en pdf de la documentación de este módulo
  - **crear.c:** código fuente de este módulo
  - **crear:** ejecutable de este módulo
- **CrearProbabilidades:** contiene el módulo que crea los archivos de probabilidades
  - **documentacion:** documentación del código fuente de este módulo
    - **html:** documentación en html
      - **index.html:** pagina html con la documentación de este módulo
    - **latex:** documentación en latex y pdf
      - **refman.pdf:** documento en pdf de la documentación de este módulo
  - **matrizDispersion.c:** código fuente de este módulo
  - **matrizDispersion.h:** código fuente de este módulo
  - **prob.c:** código fuente principal de este módulo
  - **prob:** ejecutable de este módulo
- **PrediccionDeTexto:** contiene el módulo principal del algoritmo predictivo
  - **documentacion:** documentación del código fuente de este módulo
    - **html:** documentación en html
      - **index.html:** pagina html con la documentación de este módulo
    - **latex:** documentación en latex y pdf
      - **refman.pdf:** documento en pdf de la documentación de este módulo
  - **ap:** ejecutable de este módulo
  - **ap.c:** código fuente principal de este módulo
  - **diccionario.c:** código fuente de la estructura de datos diccionario
  - **diccionario.h:** código fuente de la estructura de datos diccionario
  - **viterbi.c:** código fuente del algoritmo de Viterbi
  - **viterni.h:** código fuente del algoritmo de Viterbi
  - **entrada.txt:** contiene la entrada al programa

reporteFinal.pdf: El presente documento

Compilación y ejecución

Módulo Crear lista

Este programa utiliza el archivo *lajornada.txt* y crea los archivos *modificado.txt* y *diccionario.txt* en sus respectivas carpetas ya mencionadas.

Compilación

$ gcc crear.c -o crear

Ejecución

$ ./crear

Módulo Crear probabilidades

Este programa utiliza el archivo *modificado.txt* y crea los archivos *probabilidadesA.txt, probabilidadesB.txt* y *probabilidadesC.txt* en sus respectivas carpetas.

Compilación

$ gcc prob.c matrizDispersion.c -o prob -lm

Ejecución

$ ./prob

Módulo Algoritmo predictivo

Este programa utiliza los archivos *doccionario.txt, probabilidadesA.txt, probabilidadesB.txt, probabilidadesC*.txt y opcionalmente entrada.txt, y crea el archivo *salida.txt*

Compilación

$ gcc ap.c diccionario.c viterbi.c -o ap -lm

Ejecución

Existen dos formas de ejecución, con argumentos y sin argumentos

Sin argumentos: Entrada por teclado, a continuación del comando se presiona enter y enseguida se pueden comenzar a escribir los códigos, el programa termina con EOF que en Linux se genera con las teclas Alt + D.

./ap
Entrada (Termina con EOF):

Con argumentos: entrada por archivo, a continuación del comando es necesario especificar el nombre del archivo

./ap nombre_archivo

ejemplo: ./ap entrada.txt