

Pruebas de escritorio de algoritmos de búsqueda

Alejandro erazo mejia

Karen andrea navas

jonathan imbaqui

esteban ortiz

Universidad Santiago de Cali | Análisis de algoritmos | 19 de octubre del 2019

# Pruebas de escritorio de algoritmos de búsqueda

Vamos a emplear el primer párrafo del libro “Pulgarcito, Hermanos Grim” para desarrollar las pruebas de escritorio de las técnicas y algoritmos:

* Tablas de hashing.
* Técnicas para hallar el camino más corto en un grafo.
  + Dijsktra.
  + Floyd y Warshall.
* Knuth Morris Pratt.
* Boyer Moore.
* Fuerza bruta.

## Parrafo:

Érase una vez un pobre campesino. Una noche se encontraba sentado, atizando el fuego, mientras su esposa hilaba sentada junto a él. Ambos se lamentaban de hallarse en un hogar sin niños.

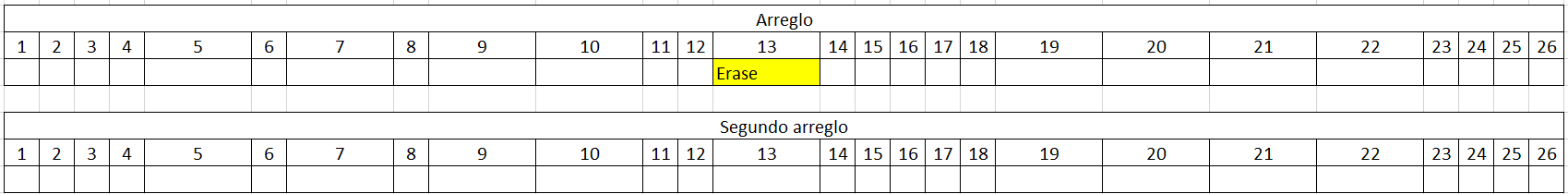
# Tablas de hashing

Para emplear el algoritmo de tablas de hashing vamos a usar 10 palabras del párrafo, sacando de cada palabra una llave la cual dará la posición en el arreglo que será almacenada la palabra, las llaves que presenten colisiones serán almacenadas en otra tabla.

1. Erase
2. Pobre
3. Campesino
4. Noche
5. Encontraba
6. Sentado
7. Atizando
8. Fuego
9. Mientras
10. Esposa

El algoritmo coge palabra por palabra y la convierte en número por medio del código ASCII el cual tiene establecido un número para cada letra o carácter, esto se hace para sacar una llave por medio de una operación matemática, en este caso se saca el residuo de cada número y se toma como llave (numero%23), posterior a ello se guarda cada palabra en el lugar del arreglo que la llave indique. Para evitar las colisiones de llaves, se guarda la palabra que tenga una llave similar en un segundo arreglo y si existe una tercera o más colisiones se guarda la palabra en el segundo arreglo en la siguiente posición al de la colisión.

1. Erase
2. 6911497115101
3. 13
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Pobre
2. 8011198114101
3. 21
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Campesino
2. 6797109112101115105110111
3. 7
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Noche
2. 7811199104101
3. 10
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Encontraba
2. 6911099111110116114979897
3. 9
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Sentado
2. 8310111011697100111
3. 5
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Atizando
2. 6511610512297110100111
3. 21
4. Condición: Como ya existe una palabra en la casilla 21, se verifica que no exista en el segundo arreglo, como no existe, guarda la palabra.



1. Fuego
2. 70117101103111
3. 19
4. Condición: La palabra no existe en el primer arreglo según la posición asignada, se guarda en el primer arreglo.



1. Mientras
2. 7710510111011611497115
3. 7
4. Condición: Como ya existe una palabra en la casilla 7, se verifica que no exista en el segundo arreglo, como no existe, guarda la palabra.



1. Esposa
2. 6911511211111597
3. 21
4. Condición: Como ya existe una palabra en la casilla 21, se verifica que no exista en el segundo arreglo, como si existe una palabra en la casilla 21 del segundo arreglo, se guarda la palabra en la casilla siguiente que es la 22.

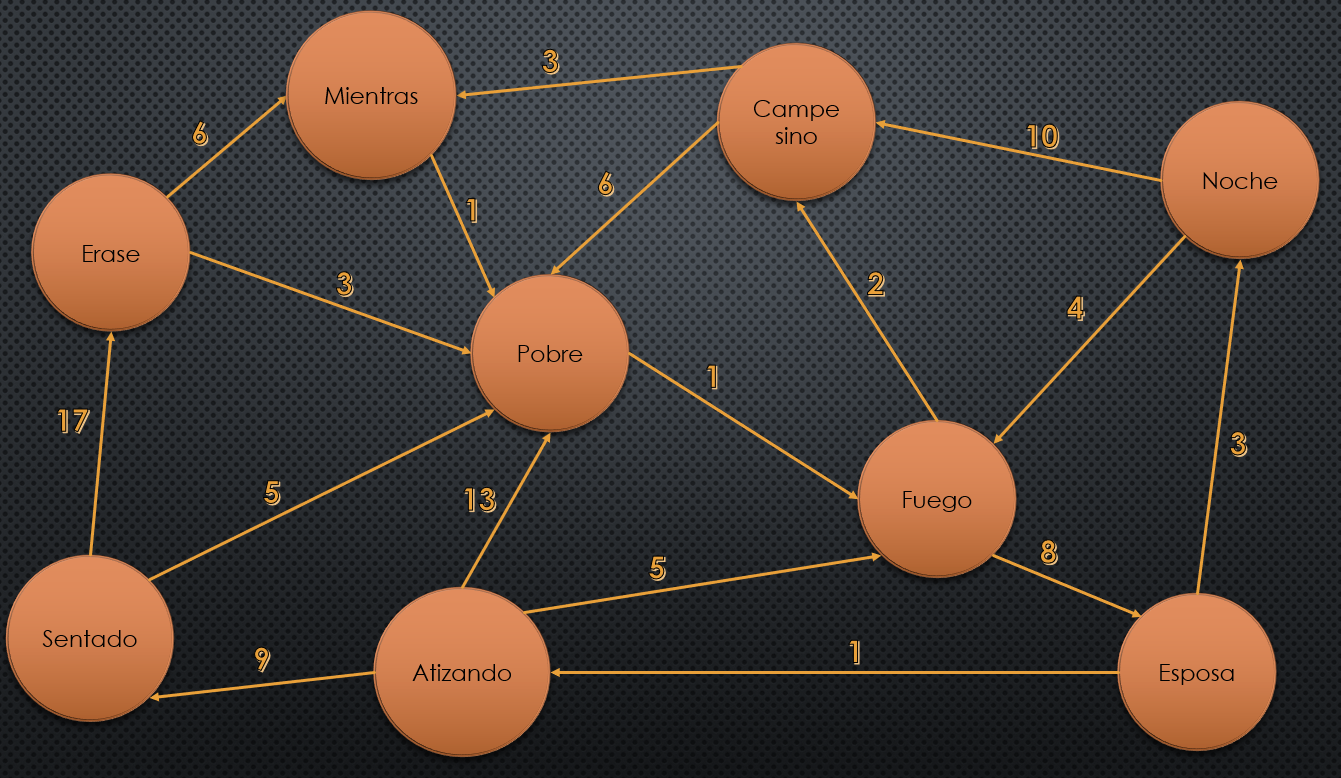


## Conclusiones

1. Al emplear tablas de hashing para el almacenamiento de una gran cantidad de información se hace mucho más eficiente la búsqueda de estos datos a diferencia de otros algoritmos que requiere recorrer casi todo el arreglo para encontrar la información requerida.
2. Al emplear esta técnica tenemos que lidiar con que la información guardada siempre estará desordenada y que existan colisiones entre llaves.

# Dijsktra

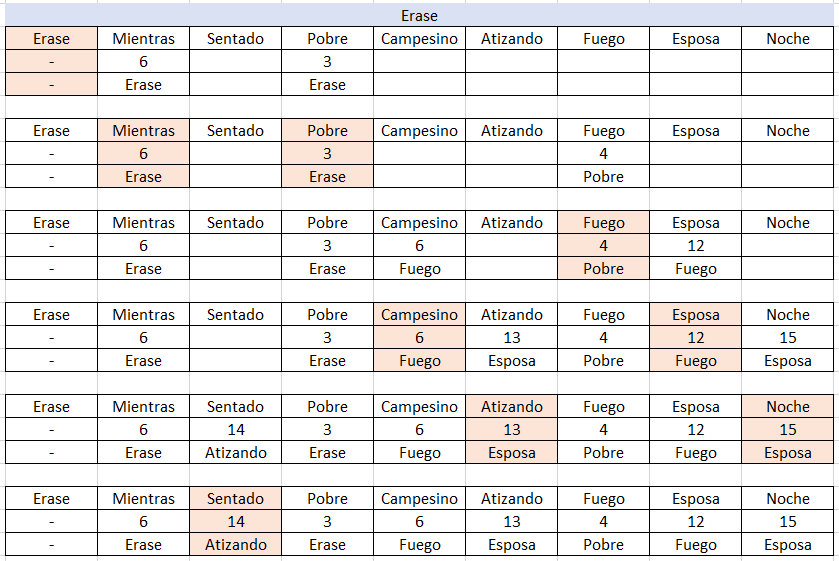
Para encontrar el camino más corto entre 2 nodos, vamos a trabajar con el siguiente grafo:



Tomando como referencia “Sentado” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



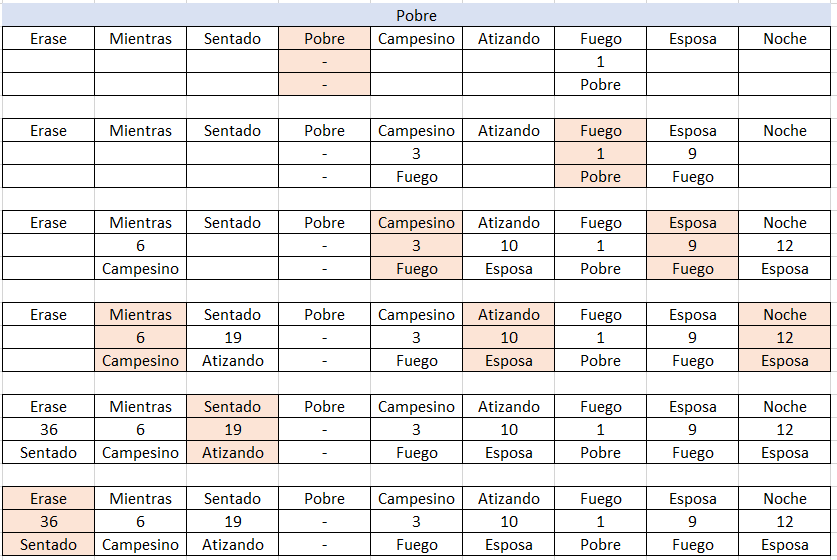
Tomando como referencia “Erase” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



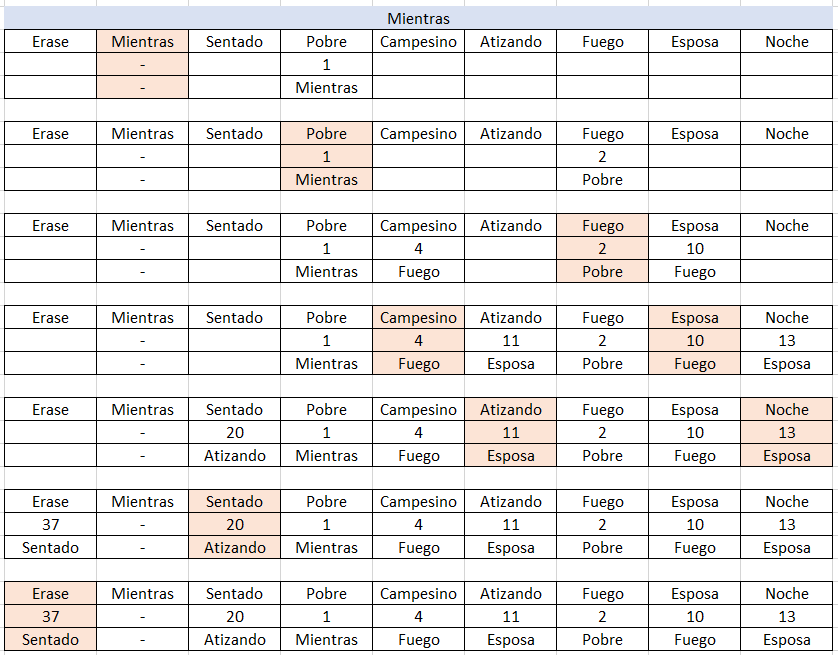
Tomando como referencia “Atizando” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



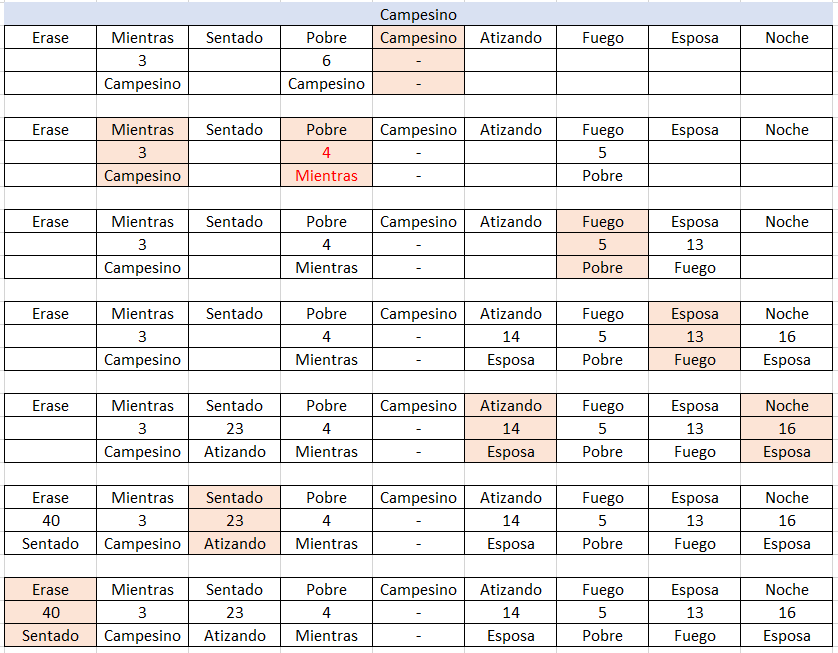
Tomando como referencia “Pobre” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



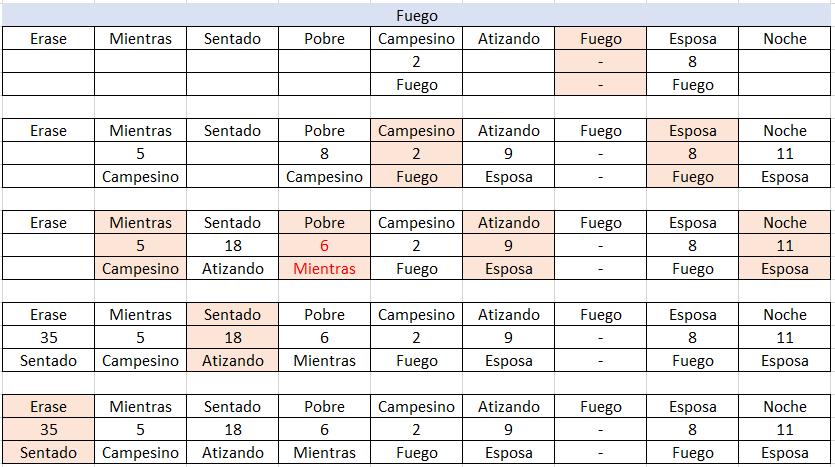
Tomando como referencia “Mientras” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



Tomando como referencia “Campesino” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



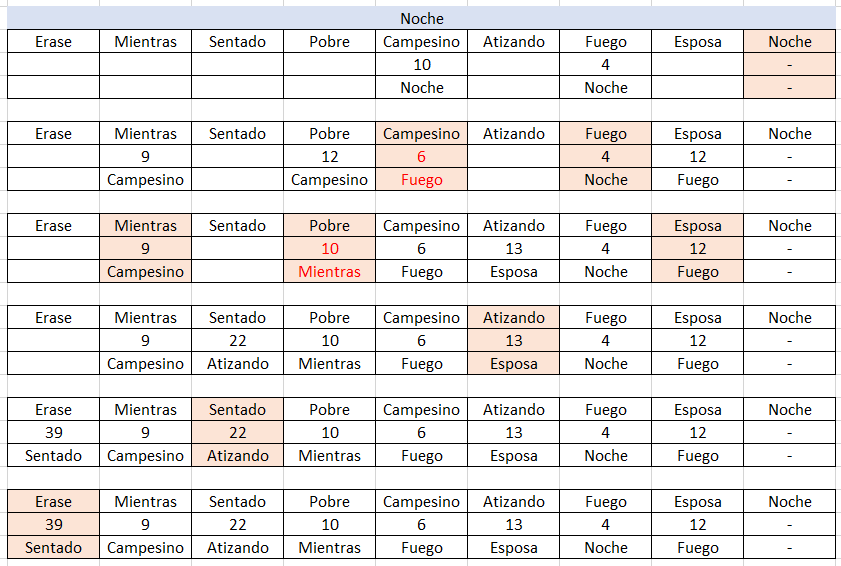
Tomando como referencia “Fuego” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



Tomando como referencia “Esposa” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



Tomando como referencia “Noche” se hallan todos los caminos más cortos para llegar a los diferentes nodos del grafo.



## Conclusiones

1. Este algoritmo permite la búsqueda del camino más corto (de un nodo A, a un nodo B) sin la necesidad de recorrer todos los nodos.
2. Aunque Dijkstra es una buena solución para hallar el camino más corto entre dos nodos se queda atrás contra otros algoritmos que solucionan este problema de una manera más fácil y eficiente.

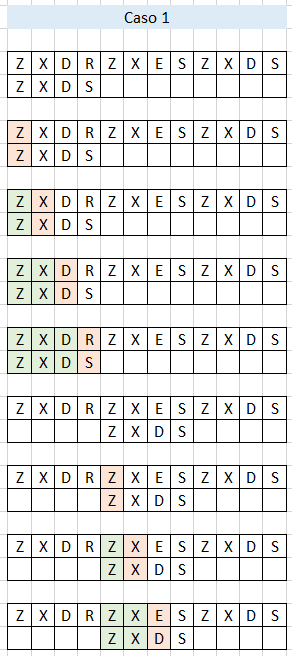
# Knuth Morris Pratt

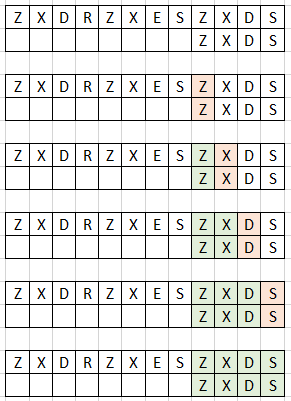
Knuth Morris Pratt utiliza la información que tiene en el patrón justo antes del momento del fallo. Se va colocando el patrón mientras no haya fallo, recorriendo la cadena de izquierda a derecha

Cadena madre: ZXDRZCESZXDS

Cadena: ZXDS

La cadena se coloca debajo de la cadena madre y empieza a verificar letra por letra de izquierda a derecha hasta no encontrar fallos, si encuentra un fallo, se corre la cadena completa hacia la derecha y vuelve a repetir el procedimiento hasta no encontrar fallos.





## Conclusiones

1. El algoritmo de Knuth Morris Pratt es una solución mas que se le da al problema de la búsqueda de una palabra en un texto. Este algoritmo usa información basada en fallos previos, aprovechando la información que la propia palabra a buscar contiene de sí.
2. Aunque el algoritmo de Knuth Morris Pratt no es el mas lento, tampoco es el algoritmo mas eficiente para resolver este problema.

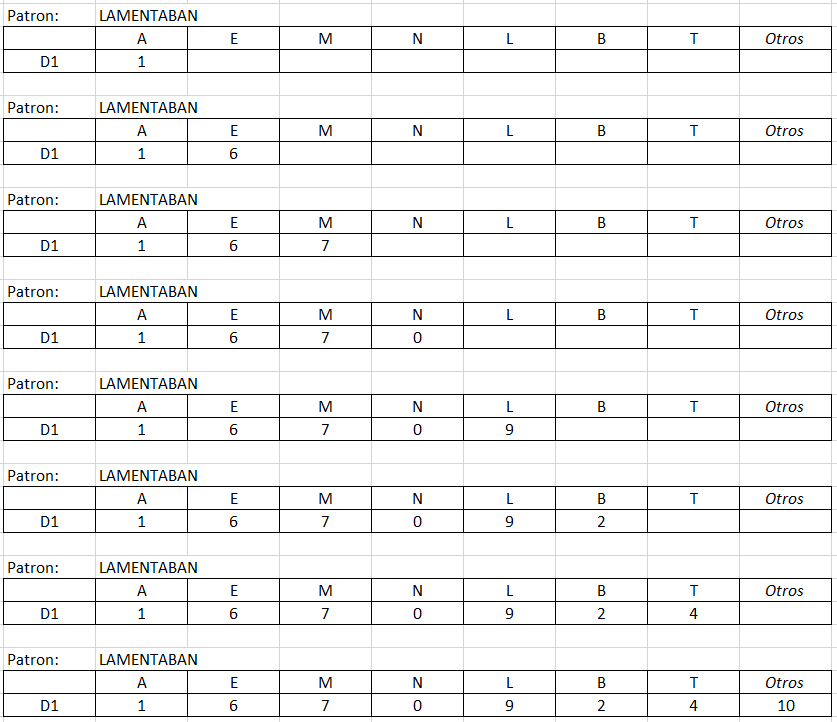
# Boyer Moore

Para hacer las pruebas de escritorio del algoritmo de Boyer Moore vamos hacer un cálculo con prefijo malo y uno con subfijo bueno, en las dos vamos a usar como patrón la palabra “lamentaban” encontrada en el párrafo de nuestro texto.

**Cálculo de la tabla D1 – Con prefijo malo**

Primero se coloca cada ocurrencia de cada carácter solo una vez en la tabla y se deja un último espacio para el resto de caracteres posibles que puedan aparecer en la cadena madre.

En cada carácter se coloca el número de posiciones que el patrón avanza. Para contar las posiciones se empieza desde el fin de la cadena hacia adelante. Si hay una letra repetida, se toma la que está más a la derecha posible. En la casilla de otros se coloca el tamaño total del patrón. Quedando de la siguiente manera:



Cadena madre: SODLIXHLAMENTABANGFWOL

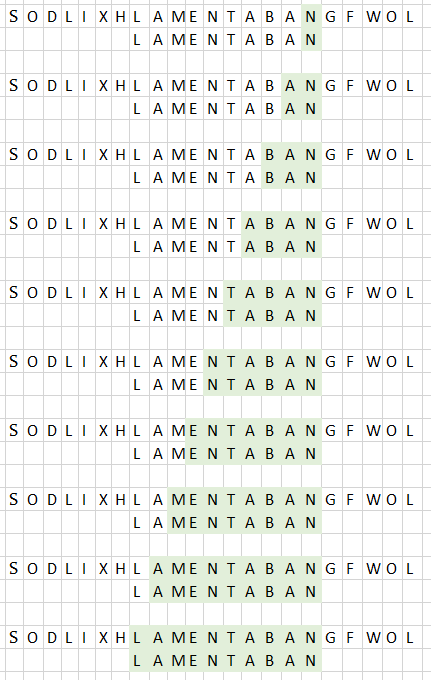
Para probar en la cadena madre con Boyer More se coloca el patrón completo debajo de la cadena madre.



Encontramos que hay un fallo con “M” y “N”, entonces se verifica si la “M” esta en nuestra tabla. Como si se encuentra, se corre la palabra “7” posiciones más. (En caso de que no estuviera, se corren las posiciones de la casilla “otros” en este caso “10”).

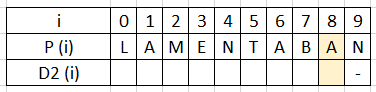


Se verifican las letras “N” y “N”, como hay coincidencia, se sigue recorriendo todo el patrón verificando que todas las letras coincidan.

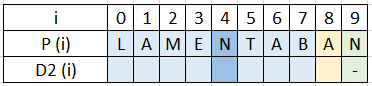


**Cálculo de la tabla D2 – Con subfijo bueno**

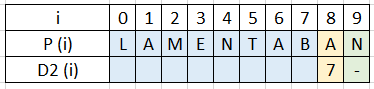
Primero se coloca el patrón de forma ordenada en la tabla, en D2, en el último carácter se coloca una raya “-” porque si el ultimo carácter del patrón falla con la cadena madre, iremos a la tabla D1 que se vio anteriormente. Por lo tanto, el ultimo carácter del patrón siempre se pone con una raya.



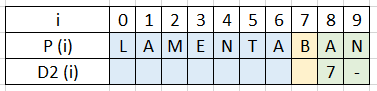
Para buscar un patrón, En este caso la “A” parte la cadena en 2, lo que está a la derecha de la “A” se busca el patrón de la derecha y los que están a la izquierda de la “A” se busca el patrón de la izquierda.



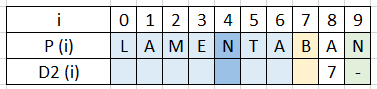
Empezamos a buscar el patrón desde la “B” hasta la “L”. En la posición 4 encontramos un posible patrón “N” y “N”, se comparan los caracteres anteriores al patrón de la derecha y el patrón de la izquierda si son iguales o no, en este caso no lo son, entonces el patrón es válido y se coloca en la casilla de “A” la deferencia de inicio de cada patrón (7 (izquierda) – 1 (derecha)) (si fueran iguales no tendríamos patrón y tendríamos que seguir buscando).



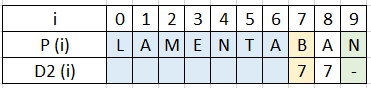
Avanzamos a la siguiente letra y continuamos buscando en el lado izquierdo el patrón “AN”.



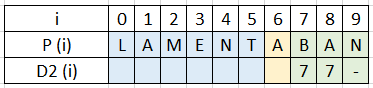
como no se encuentra, reducimos a buscar el patrón “N”



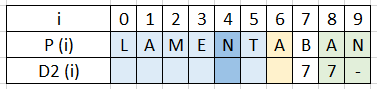
Al encontrarlo al lado izquierdo, se hace la resta y se coloca el numero en la posición “B”.

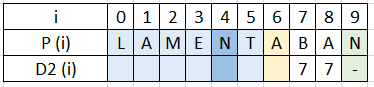


Se avanza a la siguiente letra y continuamos buscando en el lado izquierdo el patrón “BAN”.

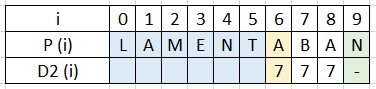


Como no se encuentra, se reduce a “AN”. Como no se encuentra, se reduce a “N”.

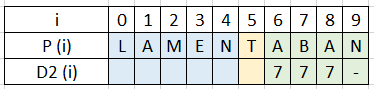




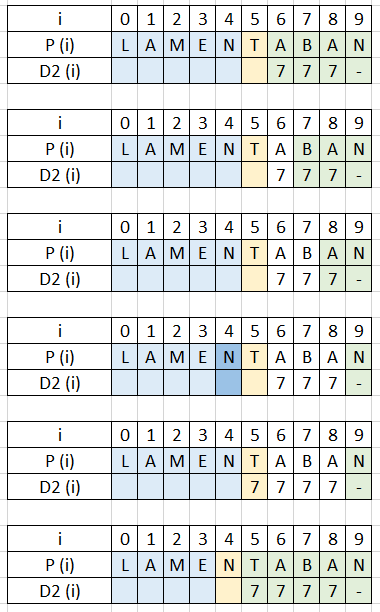
Al encontrarlo al lado izquierdo, se hace la resta y se coloca el numero en la posición “A”.

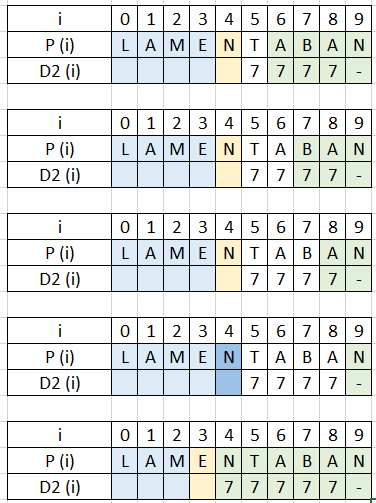


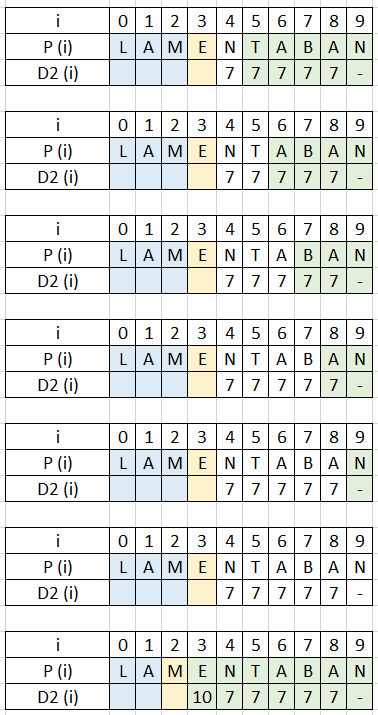
Se avanza a la siguiente letra y continuamos buscando en el lado izquierdo el patrón “ABAN”.

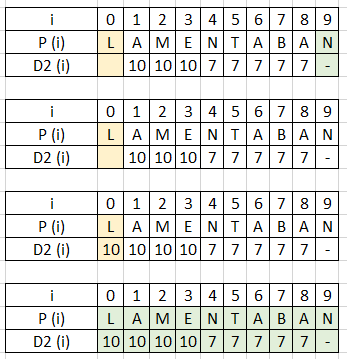
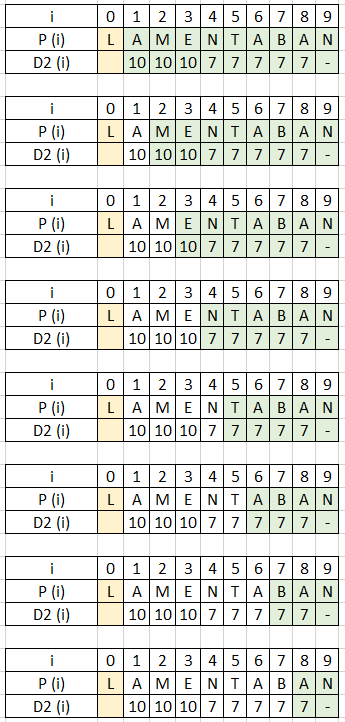
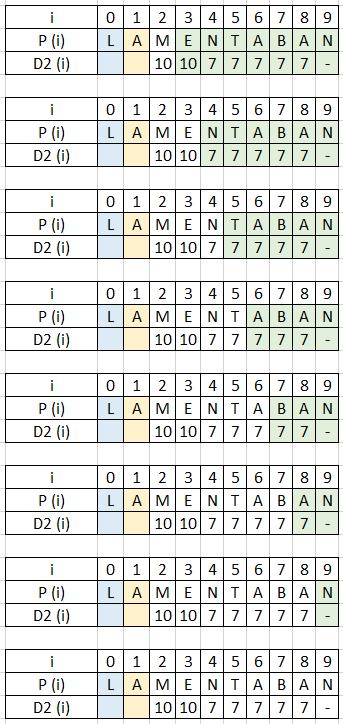
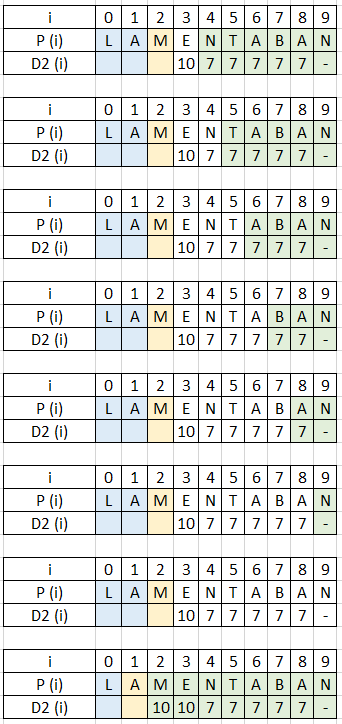


Se continua el procedimiento hasta llenar el D2.







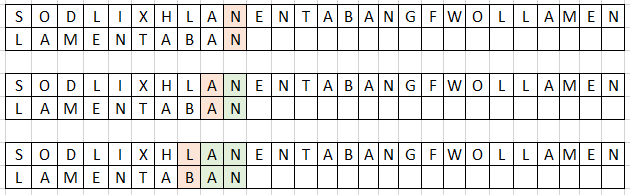


Después de llenar el de D2, lo vamos a comparar con la cadena madre:

Cadena madre: SODLIXHLANENTABANGFWOLLAMEN

Primero se coloca la palabra al comienzo de la cadena madre, al coincidir el ultimo patrón de la cadena con el de la cadena madre nos olvidamos de la tabla D1 y nos centramos en la tabla D2.

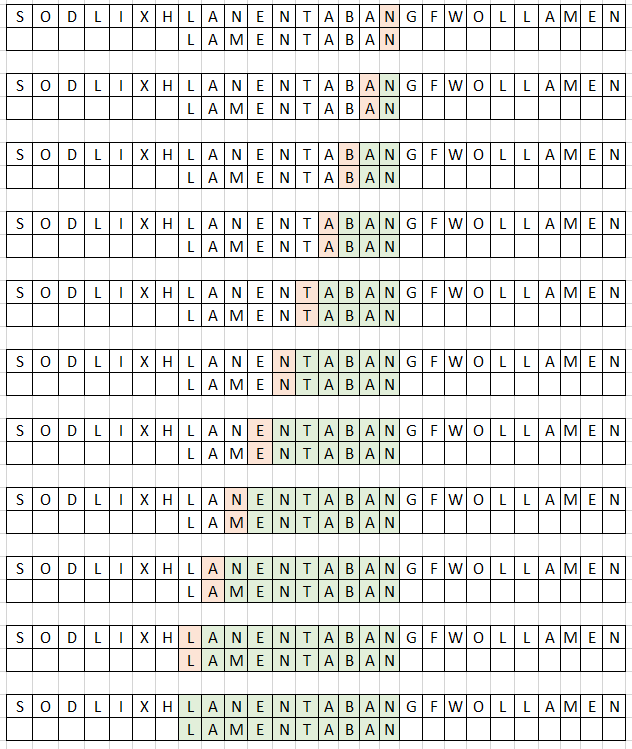
Buscamos hasta encontrar un fallo



Como no coincide la “B” con la “L”, nos vamos a nuestra tabla para mirar cuantos espacios corremos la cadena hacia la derecha. En este caso son 7 espacios.



Se verifica la ultima letra y como encuentra coincidencia recorre toda la cadena hasta que coincida, si no coincide repite el procedimiento hasta encontrar la palabra.



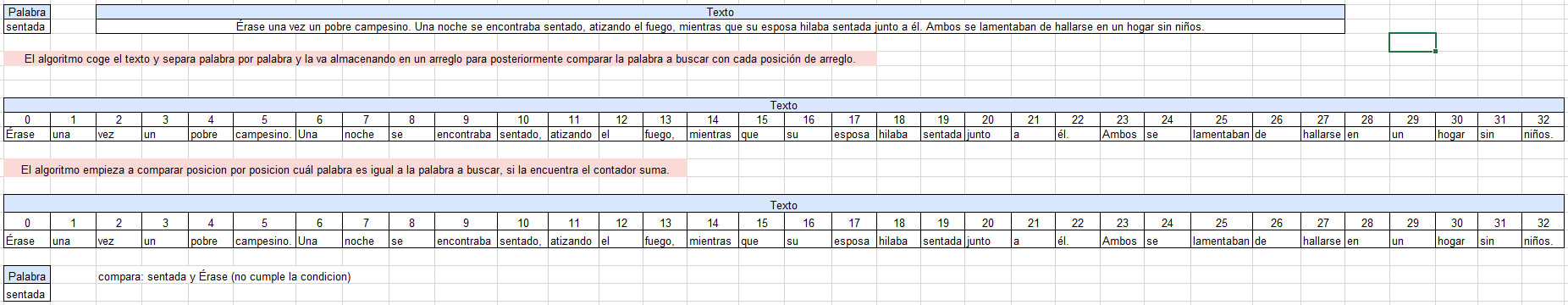
## Conclusiones

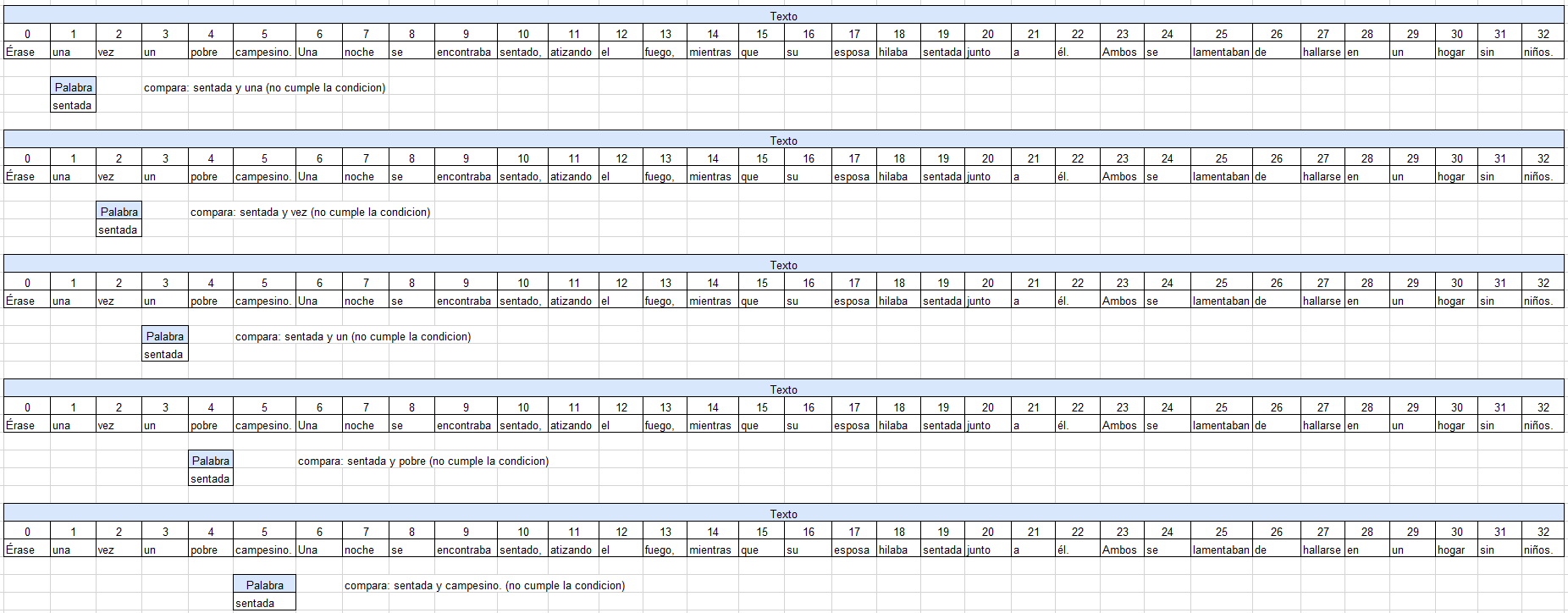
1. Podemos decir que el algoritmo de Boyer Moore tiene aplicaciones variadas como la búsqueda de una palabra en un archivo de texto, este algoritmo es extremadamente rápido, aun cuando exista un gran número de letras
2. En grandes series de documentos en un dominio particular, el algoritmo de Boyer Moore Horspool alcanza los mejores resultados al encontrar una secuencia de texto o palabra, este algoritmo realiza la búsqueda por lo menos dos veces más rápido que otros algoritmos.

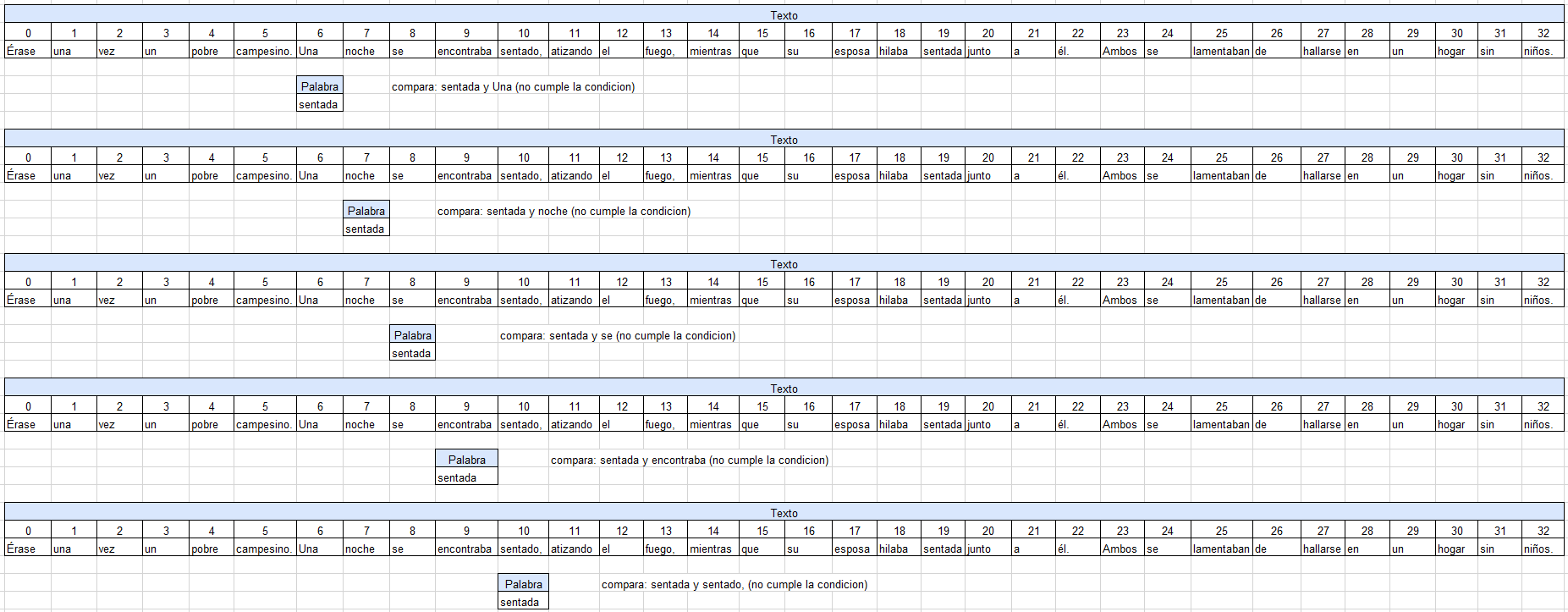
# Fuerza bruta

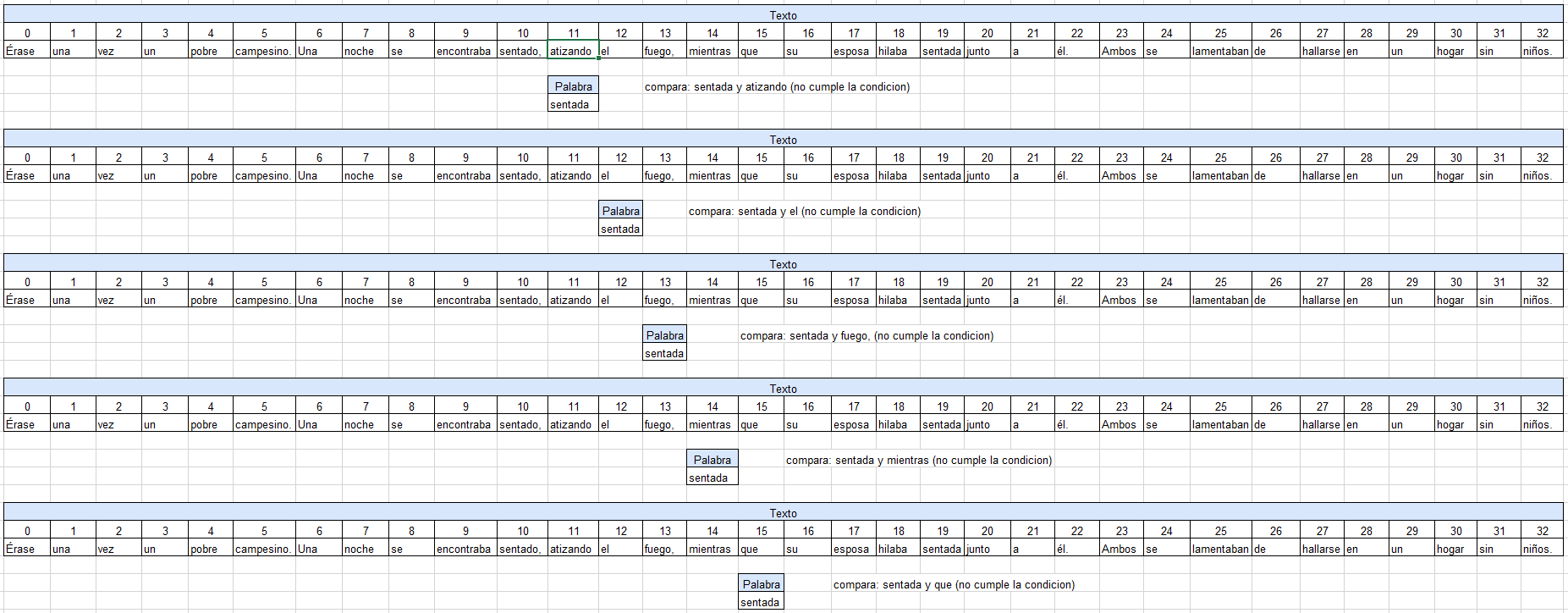
Este algoritmo se encarga de buscar una palabra en específico dentro de un texto, vamos a buscar la palabra “sentada” dentro del párrafo que tomamos del libro “Pulgarcito, Hermanos Grim”.

El algoritmo empieza tomando el texto, separa palabra por palabra y la va almacenando en un arreglo para posteriormente comparar la palabra a buscar con cada posición de arreglo. Cada vez que se encuentre la palabra a buscar, se tiene un contador que va sumando para arrojar la información de cuantas veces se repite la palabra o si la palabra no existe en el texto.

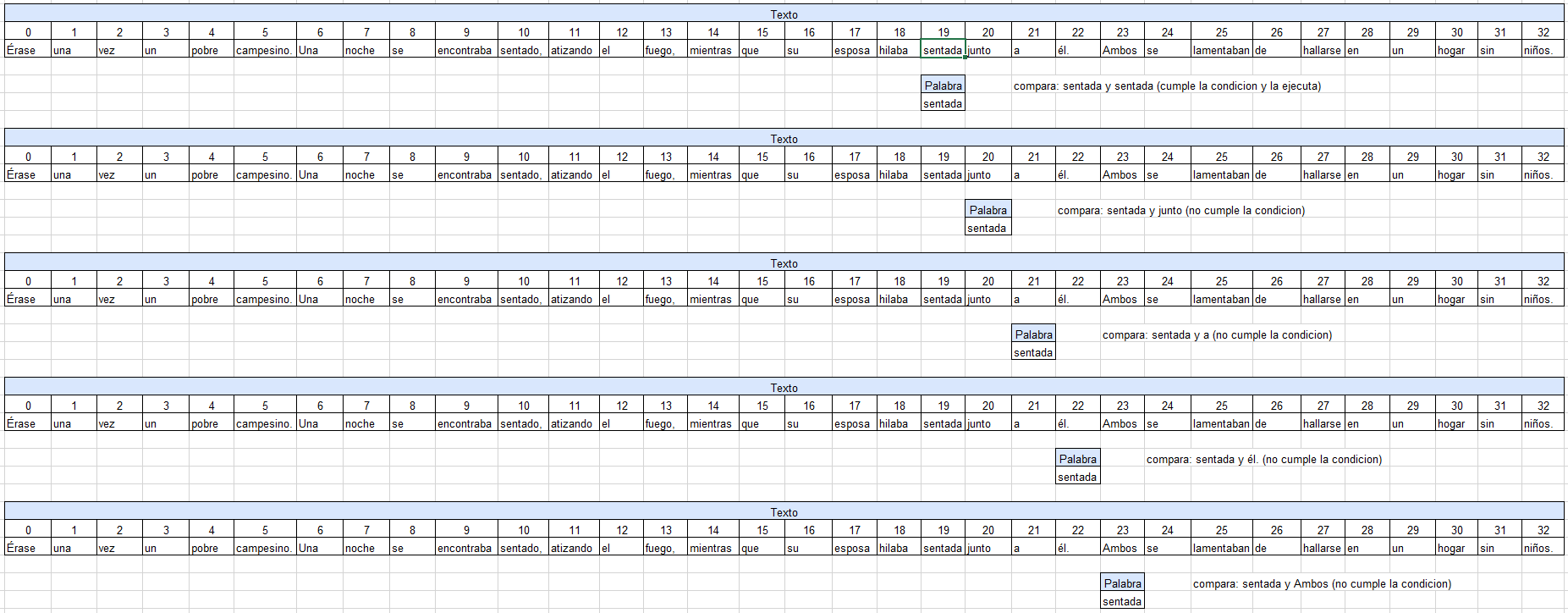


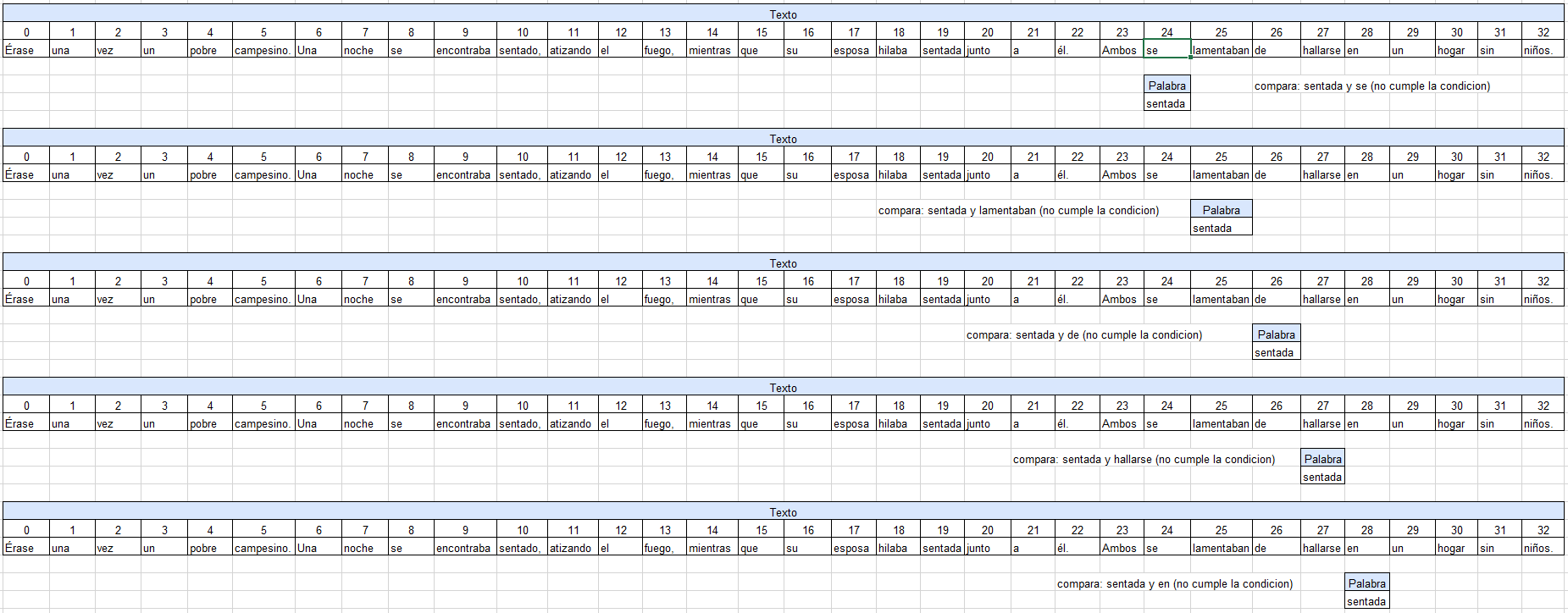


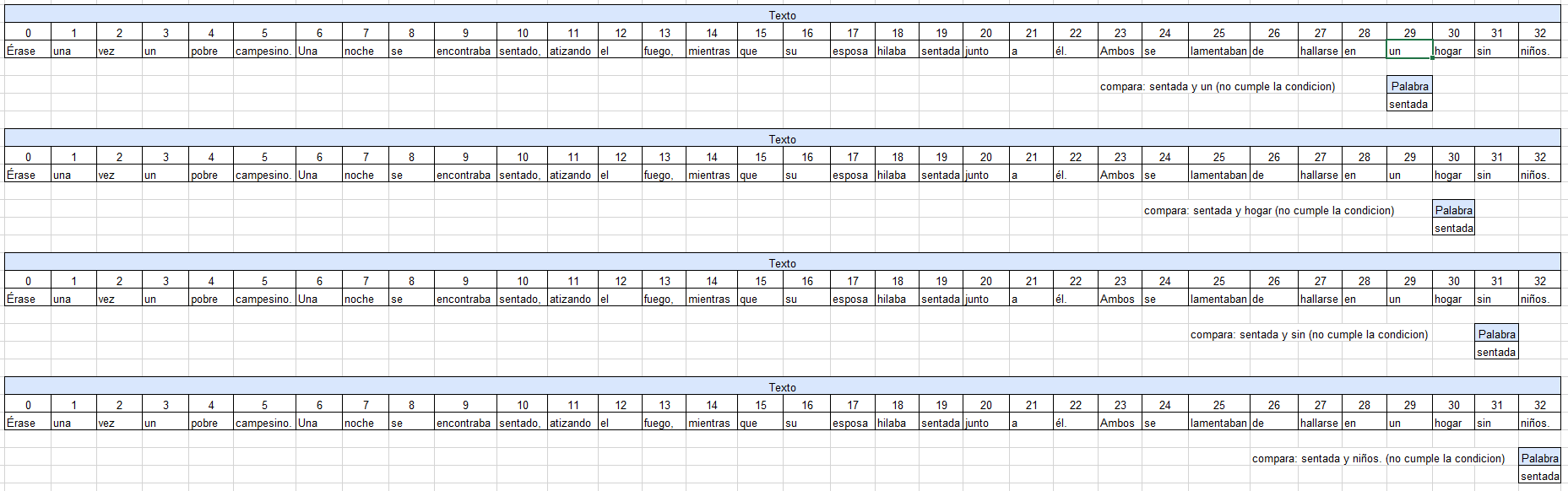












## Conclusiones

1. Usar un algoritmo de fuerza bruta para encontrar una palabra dentro de un texto no siempre es la mejor opción, aunque encuentre la solución, no siempre lo hará de la manera más optima consumiendo los mínimos recursos posibles.
2. Para buscar una palabra en un texto de una página no siempre será lo mismo que de un texto de mil páginas, por eso es importante implementar un algoritmo que realice las mínimas operaciones posibles a diferencia de uno de fuerza bruta.

# Floyd y Warshall

## Conclusiones

1. A
2. B

# Criterios de evaluación

CE1 Construye dos conclusiones propias acerca de la eficiencia de emplear tablas de hashing.

CE2 Construye dos conclusiones propias acerca de la eficiencia de las tres técnicas revisadas sobre el tema de caminos más cortos en un grafo y su uso.

CE3 Construye dos conclusiones propias acerca de la eficiencia de las técnicas de búsqueda de texto revisadas.

CE4 Guarda en la tabla de hash al menos 10 objetos diferentes del tema asignado, correctamente

CE5 Asigna correctamente una llave a cada información que almacena en la tabla.

CE6 Propone un mecanismo de conversión de llaves usando alguna de las funciones de hashing consultadas.

CE7 Soluciona las colisiones usando alguno de los mecanismos de resolución consultados.

CE8 Propone un grafo con mínimo 8 nodos y 16 enlaces que tenga sentido y relación directa con el tema asignado.

CE9 Hace la prueba del escritorio usando el grafo, para ilustrar el cálculo del camino más corto en un grafo sin pesos.

CE10 Hace la prueba del escritorio usando su grafo para ilustrar el cálculo del camino más corto empleando Dijsktra.

CE11 Hace la prueba del escritorio usando su grafo para ilustrar el cálculo del camino más corto empleando Floyd y Warshall.

CE12 Ilustra a través de una prueba de escritorio, el uso correcto de un algoritmo de fuerza bruta para buscar una palabra dentro de un texto.

CE13 Ilustra a través de una prueba de escritorio, el uso correcto del algoritmo de Knuth-Morris-Pratt para buscar una palabra dentro de un texto.

CE14 Ilustra a través de una prueba de escritorio, el uso correcto del algoritmo de Boyer-Moore para buscar una palabra dentro de un texto.

CE15 Concluye el tipo de características del texto para emplear cada algoritmo