Portada

**String Patter Matching**

Instituto de Informática, Universidad Austral de Chile.

Profesores. Mauricio Ruiz-Tagle y Héctor Ferrada

Alumnos. Fernanda Salgado {fernanda.salgado@alumnos.uach.cl}, Fernando Inzulza {fernanda.inzulza@alumnos.uach.cl} , Vicente Alves {vicente.alves@alumnos.uach.cl}

Resumen

Realizar una búsqueda de patrones en un texto parece no ser una tarea muy compleja, sin embargo, existen diversas maneras de realizar esto, a su vez el trabajo que realizará el programa será dependiente del algoritmo que se usará para resolver el problema. Por ello evaluaremos 3 diferentes algoritmos **String Pattern Matching**. Los algoritmos que se verán involucrados para esta problemática serán: Búsqueda Exhaustiva, Búsqueda mediante lista enlazada de sufijos y Búsqueda mediante el arreglo de sufijos. Con este trabajo abordaremos la efectividad de cada uno y su simplicidad; plantearemos la creación de cada algoritmo e iremos desarrollando cada uno a la par que vamos evaluándolos.

Índices:

**1**.**Introduccion.**

**2**.**Hipótesis.**

2.1. Búsqueda Exhaustiva

2.2. Búsqueda Mediante la Lista Enlazada de Sufijos

2.3. Búsqueda Mediante el Arreglo de Sufijos

**3.Estructuras de los algoritmos.**

3.1. Pseudocódigos

3.2. Tablas de Ejecución

4.Analisis de los algoritmos.

4.1. Búsqueda exhaustiva

1.Introducción

El problema a resolver es una búsqueda de patrones en un texto, **String Pattern Matching.** El algoritmo requiere 2 datos textuales para poder realizar la búsqueda (T = ⟨t0t1t2 . . . tn−1⟩; P = ⟨p0p1p2 . . . pm−1⟩; m<=n), el primero siendo el texto a examinar y el segundo un patrón de búsqueda que se le aplicará. Los pasos que seguiremos para resolver esta problemática será la creación, desarrollo e implementaron de 3 diferentes algoritmos(Los cuales serán hechos de forma premeditada), con los cuales evaluaremos: El comportamiento de cada programa, su simplicidad y a su vez efectividad.

Hipótesis

Respecto a los 3 diferentes algoritmos de búsqueda que nos han planteado. Hemos generado las siguientes expectativas respecto a cada uno.

1. **Búsqueda exhaustiva:** Este algoritmo irá recorriendo el texto hasta el final, por cada iteración que hace recorrerá el patrón y buscará alguna coincidencia. Analizando la estructura de lo que será este algoritmo estimamos que será el menos efectivo, debido que al usar dos ciclos For tendrá un comportamiento de , mientras más iteraciones realice esta tenderá al infinito.
2. **Búsqueda mediante la Lista Enlazada de Sufijos:** Este algoritmo usará una lista doblemente enlazada para almacenar lexicográficamente los N sufijos del texto, luego irá recorriendo la lista enlazada buscando el patrón SOLAMENTE en el inicio del sufijo. Lógicamente al ser un algoritmo con una estructura de ordenamiento hará que queden juntos en un lugar de la lista todos los sufijos que tengan el patrón al principio, tenemos la premisa de que será un código de orden y mucho más efectivo que el de búsqueda exhaustiva ya que no recorrerá la lista enlazada completa.
3. **Búsqueda mediante el Arreglo de Sufijos:** Este algoritmo usará un arreglo para almacenar lexicográficamente los N sufijos del texto, esto permite que se pueda implementar el algoritmo de búsqueda binaria para encontrar de una manera efectiva las veces que se repite el patrón en los N sufijos, a partir de aquí tenemos 2 posibles formas de continuar:

* La primera sería encontrar el índice del principio y el final donde se encuentran los patrones con 2 búsquedas binarias. Este método es efectivo, es más complejo de crear .
* La segunda forma es menos efectiva ya que ocupa una búsqueda binaria para encontrar una repetición del patrón en un sufijo y a partir de ese índice irá iterando hacia la izquierda y derecha para contar las veces que se repite el patrón en los demás sufijos, este algoritmo a pesar de ocupar búsqueda binaria tenemos la premisa de que sea

3.Estructuras de los algoritmos:

*3.1 Pseudocódigos:*

1. Búsqueda exhaustiva

Búsqueda\_Exhaustiva(T, P,indices)

j←0

N\_T←T.size()

N\_P←P.size()

Para i←0 Hasta N\_T-N\_P-1 Con Paso 1 Hacer

Mientras j<N\_P Y P[j]=T[i+j] Hacer

j← j+1

FinMientras

Si j = N\_P Entonces

indices.pushback(i)

FinSi

j←0

FinPara

FinProcedimiento

**Input:** Un arreglo T de tipo Char[0..n-1], P un arreglo de tipo Char[0..m-1] e indices que es una lista enlazada simple con valores que representan las repeticiones del patrón en el texto.

**Output:** Índices en los cuales se repite la ocurrencia de P en T

1. Búsqueda Mediante Lista Enlazada

Estructura De La lista enlazada

Celda de memoria

|  |  |  |
| --- | --- | --- |
| prev | IN | next |

prev: puntero al nodo anterior

IN: índices del texto ordenado lexicográficamente

next: puntero al siguiente nodo

**Input:** T un arreglo de tipo Char[0..n-1] con el texto, P un arreglo de tipo Char[0..m-1] que contiene el patrón a buscar, n entero que representa el tamaño de T, m entero que representa el tamaño de T, I un arreglo de enteros con los índices del texto ordenado lexicográficamente. .

**Output:** Indices unVector de enteros, que contiene los Índices en los cuales se repite la ocurrencia de P en T.

Procedimiento BúsquedaListaEnlazada (T, P, I, n, m)

New(p)

L ← p

p -> IN ← I[0]

Para i←1 Hasta n-1 Con Paso 1 Hacer

New(q)

q-> IN ←I[i]

p->next ← q

q->prev ← p

p←q

Fin Para

q←L

mini ← n-m

b\_existe ←Falso

Mientras ((p->IN != q->IN) Y (b\_existe = Falso)) Hacer

i\_patron ←0

i\_lista ← q->IN;

Mientras q->IN <= mini Y i\_patron < m Y T[i\_patron] = P[i\_lista] Hacer

i\_patron ← i\_patron + 1

i\_lista ← i\_lista +1

Fin Mientras

Si m = i\_patron Entonces

b\_existe ← Verdadero

b\_lugar ← Verdadero

Fin Si

Sino

i\_patron ← 0

i\_lista ← p->IN;

Mientras p->IN <= mini Y i\_patron < m Y T[i\_lista] = P[i\_patron] Hacer

i\_patron ← i\_patron + 1

i\_lista ← i\_lista +1

Fin Mientras

Si m= i\_patron Entonces

b\_existe ← Verdadero

b\_lugar\_← Falso

Fin Si

SiNo

p ← (p->prev)

q ← (q->next)

Fin SiNo

Fin SiNo

Fin Mientras

Si b\_existe=Verdadero Entonces

Si b\_lugar=Verdadero Entonces

Indices.push\_back(q->IN)

Mientras b\_existe = Verdadero Hacer

epataron ← 0

i\_lista ← p->IN

Mientras p->IN <= mini Y i\_patron < m Y T[i\_lista] = P[i\_patron] Hacer

i\_patron ← i\_patron + 1

Fin Mientras

Si m= i\_patron Entonces

b\_existe ← Falso

Fin Si

Sino

p ← (p->prev)

Fin SiNo

Fin Mientras

i\_patron ← 1

Mientras (q->IN != p->IN) Hacer

Indices.push\_back(p->IN)

i\_patron ← i\_patron+1

p ← (p->prev)

Fin Mientras

Retorna Indices

Fin Si

Sino

Indices.push\_back(p->IN)

Mientras b\_existe = Verdadero Hacer

i\_patron ← 0

i\_lista ← q->IN

Mientras q->IN <= mini Y i\_patron < m Y T[i\_lista] = P[i\_patron] Hacer

i\_lista ← i\_lista + 1

i\_patron ←i\_patron + 1

Fin Mientras

Si m = i\_patron Entonces

b\_existe ← Falso

Fin Si

Sino

q ← (q->next)

Fin SiNo

Fin Mientras

i\_patron ← 1

Mientras q->IN != p->IN Hacer

Indices.push\_back(q->IN)

I\_patron ← i\_patron+1

q ← (q->next)

Fin Mientras

Retorna Indices

Fin SiNo

Fin Si

Sino

Retorna Indices

Fin SiNo

Fin Procedimiento

1. Búsqueda mediante Arreglo de Sufijos

**Input:** Recibe 4 vectores como parámetros:

* **T:** es de tipo char[0..n-1] y almacena el texto
* **inSuf:** es de tipo entero[0..n-1] y almacena los índices de los sufijos ordenados lexicográficamente
* **P:** es de tipo char[0..z-1] y es el que contiene el patrón a buscar
* **indices:** es de tipo entero[0..w-1] y almacena los índices en donde se encontraros coincidencias con el patrón.

**Output:** No tiene,ocupa direcciones de memoria por lo que modifica directamente el vector indices

Procedimiento Búsqueda\_ArregloSufijos(T, inSuf, P, indices)

n ← inSuf.size()

L ← 0

R ← n-1

M ← R/2

Mientras L<=R hacer

Si esPatron(inSuf[M], T, P) entonces

Si M = 0 o ­­­­­ ¬ esPatron(inSuf[M-1], T, P) entonces

L← M

Break

FinSi

R ← M-1

SiNo

Si menorLex(P, T, inSuf[M]) entonces

R ← M-1

SiNo

L ← M+1

FinSi

FinSi

M ← (L+R)/2

FinMientras

Si R<L entonces

return //AQUÍ LO QUE HACE ES SALIR DE LA FUNCION

R ← n-1

L\_aux ← L

M ← (L+R)/2

Mientras L\_aux<=R hacer

Si esPatron(inSuf[M], T, P) entonces

Si M >= (n-1) o ­­­­­ ¬ esPatron(inSuf[M+1], T, P) entonces

R← M

Break

FinSi

L\_aux ← M+1

SiNo

R ← M-1

FinSi

M ← (L\_aux+R)/2

FinMientras

Para j ← L Hasta R Con Paso 1 Hacer

indices.push\_back(inSuf[ j ])

FinPara

FinProcedimiento

**esPatron(int t\_i, vector<char> T, vector<char> P):** Esta función recibe un entero y 2 vectores, el entero servirá para iterar a partir de ese índice en el primer vector pedido (que contiene el texto) y el segundo vector es el patrón que se buscará al principio del primer vector a partir del índice entregado. Retorna True si tiene el Patrón, False si no.

**menorLex(vector<char> P, vector<char> T, int t\_i):** La función menorLex recibe 2 vectores, uno es un patrón (Primer vector P) y el otro es el que tiene el texto almacenado (Segundo vector T), además de recibir también un entero que se utiliza para iterar a partir de ese índice en el vector T. Retorna True si el primer vector es menor lexicográficamente que el segundo, False si no.

*3.2. Tablas de ejecución:*

El texto que se leerá para probar los algoritmos es “AABAACAADAABAABA”

Búsqueda exhaustiva:

Imagen que contiene edificio, persiana, biombo

Descripción generada automáticamente

Búsqueda mediante Lista Enlazada

**Tabla

Descripción generada automáticamente con confianza media**

-Representación gráfica de la lista enlazada

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Búsqueda mediante Arreglo de sufijos



4.Análisis de los algoritmos:

A continuación, realizaremos un análisis de nuestros algoritmos, con este quedará remarcado de forma empírica el rendimiento que tiene cada método de búsqueda para situaciones específicas.

Para ello utilizaremos principalmente notaciones del trabajo, comportamiento y sobre todo gráficos BIG O con tal de dejar en claro cual es la tendencia del programa.

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Referencia obtenida de: <https://www.netmentor.es/entrada/notacion-big-o>

4.1Análisis Búsqueda Exhaustiva:

Para el análisis del siguiente algoritmo hemos decidido evaluar el trabajo que genera éste, su comportamiento y darle valores en diferentes situaciones.

En vista de lo anterior. Lo primero que haremos será evaluar la problemática del problema y cuáles son las operaciones básicas de este.

***Trabajo del Algoritmo de búsqueda exhaustiva***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problema | Operaciones Básicas | Average: A(n) | | Worst: W(n) |
| Encontrar todos los  índices de un texto. | | Comparar 2 Chars entre sí (Texto, Patrón) | A |  | |
| Comportamiento |  |  | |  |

Imagen que contiene Gráfico

Descripción generada automáticamenteDado esto, podemos observar que este algoritmo tiene un orden creciente considerablemente mayor a los diferentes algoritmos que se verán a continuación; si bien esto no puede percibirse en un principio, esto tiene un mayor impacto en cuanto más datos de entrada recibimos.

Por lo que mientras más datos de entrada tengamos, este tendrá un comportamiento que tenderá al infinito.

Haciendo que este algoritmo llegará a un punto en el que su trabajo será incalculable.

De igual manera, no podemos comparar este método de búsqueda con los demás. Esto pues el algoritmo con el que estamos operando funciona mediante la búsqueda de patrones en un vector de tipo *Char* sin un orden fijo como los programas que se verán a continuación. De igual manera, se mostrará una representación de los trabajos que realizó el algoritmo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  patrón | Largo del texto | Numero de Coincidencias | Tiempo de ejecución |
| AABA |  |  | 16 |  |  |
| AABA |  |  | 96 |  |  |
| G |  |  | 160.283 |  |  |
| GA |  |  | 160.283 |  |  |
| GAA |  |  | 160.283 |  |  |
| in |  |  | 51.421.152 |  |  |
| the |  |  | 51.421.152 |  |  |
| dog |  |  | 51.421.152 |  |  |

Dado lo anterior concluimos que, para ser un algoritmo de búsqueda sencilla, este logra cumplir su función. De igual manera consideramos que existen soluciones más eficientes que la Búsqueda exhaustiva.

4.2 Análisis de Búsqueda por lista Enlazada

Para analizar el comportamiento del algoritmo, se debió de examinar detenidamente el código que lo compone, con esto se pudo deducir que el algoritmo (en el peor caso) es , pero en la gran mayoría de ocasiones no sobrepasa el comportamiento lineal ().

Además, se realizaron una serie de pruebas para corroborar las deducciones antes mencionadas

Estas pruebas se hicieron en 2 documentos (dna.50mb y english.50mb) y a su vez estas se dividieron en dos, se vio el comportamiento del algoritmo con los mismos patrones con entradas diferentes.

La primera prueba fue para 3 patrones de largo 3, con .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| GTA |  |  | 279771 |  |  |
| GTC |  |  | 279771 |  |  |
| TTT |  |  | 279771 |  |  |

Con

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| GTA |  |  | 743264 |  |  |
| GTC |  |  | 743264 |  |  |
| TTT |  |  | 743264 |  |  |

Se puede observar que los tiempos de ejecución entre diferentes varían por centésimas, y que entre diferentes patrones con el mismo largo no se observan variaciones significativas, se visualiza un crecimiento lineal en el tiempo de ejecución tal como se había esperado.

Esta misma prueba se realizó en otro texto

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| Who |  |  | 475767 |  |  |
| The |  |  | 475767 |  |  |
| was |  |  | 475767 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| Who |  |  | 937769 |  |  |
| The |  |  | 937769 |  |  |
| was |  |  | 937769 |  |  |

De la misma forma que el anterior, corrobora nuestra hipótesis, los tiempos de ejecución crecen linealmente.

En la segunda prueba se cambian el tamaño de los patrones, para observar que importancia tienen estos en el comportamiento.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| ATCCCTCCCTGCCATG |  |  | 279771 |  |  |
| GGATCACTTGAGTTCA |  |  | 279771 |  |  |
| TGAGTCATCATAGGCA |  |  | 279771 |  |  |

A primera vista no parece haber un cambio significativo, se deduce que el largo del patrón no tiene efecto alguno en el tiempo de ejecución

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| ATCCCTCCCTGCCATG |  |  | 743264 |  |  |
| GGATCACTTGAGTTCA |  |  | 743264 |  |  |
| TGAGTCATCATAGGCA |  |  | 743264 |  |  |

En este ultimo parece haber un pequeño aumento, con respecto a la prueba anterior, se espera que el siguiente experimento aclare las ideas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| Revolutionaries |  |  | 475767 |  |  |
| proportionally |  |  | 475767 |  |  |
| transportation |  |  | 475767 |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón a buscar | Texto en el que se busca | Largo del  Patrón | Largo del texto | Numero de coincidencias | Tiempo de ejecución |
| Revolutionaries |  |  | 937769 |  |  |
| proportionally |  |  | 937769 |  |  |
| transportation |  |  | 937769 |  |  |

Con los resultados anteriores no podemos declarar que el patrón tenga un efecto significativo en el trabajo realizado por el algoritmo.

Tomando los diferentes experimentos realizados, se puede afirmar que el algoritmo tiene un comportamiento promedio de

Imagen que contiene Forma

Descripción generada automáticamente

Los resultados nos llevan a concluir que este algoritmo es mas eficiente que el anterior de búsqueda exhaustiva, pero claramente existen mejores algoritmos para resolver este problema.

4.3 Análisis Búsqueda Mediante el Arreglo de Sufijos:

Para examinar este algoritmo se analizará el comportamiento y se probarán diferentes búsquedas de patrones en diferentes archivos.

*Siendo* ***M*** *el largo del Patrón y* ***N*** *el largo del Texto entonces:*

El algoritmo de búsqueda funciona buscando los índices de los bordes en donde se encuentran los sufijos con el patrón al principio, para encontrarlos se ocupará la búsqueda binaria, 2 en este caso (Borde Izquierdo y Borde Derecho). Por lo que con esta información se puede estimar que se comportara de manera similar a un .

Pero también hay que tener en cuenta el largo del patrón y como este se compara cuando se ocupa la búsqueda binaria, haciendo que al final este termine multiplicado por **M**

*Y así en base a este análisis se concluye que el comportamiento del algoritmo es*

**A continuación, hay imágenes de como crece la función dependiendo del patrón:**

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Con **M** = 1

Gráfico

Descripción generada automáticamente

Con **M** = 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Patrón | Texto en el que se busca | Largo del  patrón | Largo del texto | Numero de Coincidencias | Tiempo de ejecución |
| the |  |  | 475767 |  |  |
| i |  |  | 475767 |  |  |
| and |  |  | 937769 |  |  |
| i |  |  | 937769 |  |  |
| TTT |  |  | 279771 |  |  |
| AAAAA |  |  | 279771 |  |  |
| TTT |  |  | 743264 |  |  |
| AAAAA |  |  | 743264 |  |  |

Y en base a estos datos podemos concluir que, este algoritmo pese a tener una complejidad más alta al implementarse en el código, es capaz de cumplir con su función de una manera muy eficiente (Menos comparaciones hechas y poco tiempo requerido en cada búsqueda de un patrón).

Comparado con las otras 2 soluciones anteriores, esta puede ser la mejor opción si se quiere un algoritmo de búsqueda eficiente.