Unidad 7 - Actividad 1

Materia: **Análisis de Algoritmos y Estructuras para Datos Masivos**

Alumno: **Luis Fernando Izquierdo Berdugo**

Fecha: **6 de Noviembre de 2024**

Introducción

En esta unidad se exploró el concepto de **búsqueda de patrones en cadenas**, el ejemplo más sencillo de esto es intentar encontrar una palabra específica en un documento.

La búsqueda de patrones en cadenas es el procedimiento donde se busca todas las apariciones de un patrón (en el ejemplo anterior, una palabra) dentro de un texto mayor (un documento en el ejemplo); para esto se utilizan diversos algoritmos como lo son el algoritmo "Shift-And" y el algoritmo "Naïve".

Explicación de métodos usados

Algoritmo Shift-And

Para explicar el algoritmo "Shift-And", se puede volver al ejemplo anterior y ahora buscar una frase en vez de una sola palabra en el documento; lo que el algoritmo haría es crear un mapa de la frase que se está buscando, el cual le permite saltar grandes secciones del libro si no se encuentra ninguna coincidencia.

El algoritmo se puede dividir en 3 procesos distintos, los cuales son:

1. Representación Binaria.

- Se crea una representación binaria de la frase a buscar y del documento, en esta, cada carácter se le asigna un número binario único.
- Se crea una máscara de bits para cada carácter de la frase, la cual tiene un 1 en la posición correspondiente al carácter y 0 en el resto

2. Preprocesamiento

- Se construye un autómata finito determinista (AFD) a partir de la frase, el cual representa todos los posibles prefijos de la frase.

3. Búsqueda

- Se inicia un desplazamiento en el documento, el cual irá dependiendo del tamaño de la frase.
- En cada posición del desplazamiento, se observa una operación “AND” entre la máscara del carácter actual de la ventana y el estado actual del autómata finito, lo cual actualiza el estado de este último.
- En caso de que el estado final del autómata finito se alcance, se está ante la presencia de una coincidencia del patrón lo cual indica que se encontró la frase en el documento

Algoritmo Naïve

El algoritmo “Naïve” explicado con el ejemplo de la frase y el documento sería como si se tomara el documento y se compare letra por letra la frase, en el caso de que la primera letra del documento coincida con la primera de la frase, se sigue con la siguiente letra de ambos; si en algún carácter no coincide se pasa a la siguiente letra del libro y se vuelve a comparar la frase completa.

Los pasos a seguir para la implementación de este algoritmo son:

1. Comparación carácter a carácter.

- Se compara el primer carácter de la frase con el primer carácter del documento.

2. Avance

- Si coincide, se pasa al siguiente carácter del patrón y del texto, volviendo a compararlos.

3. Reinicio

- Si los caracteres no coinciden, se vuelve al inicio de la frase y se compara desde el siguiente carácter del texto.

4. Ciclo

- Se repiten todos los pasos anteriores hasta llegar al final del documento.

Explicación de experimentos

Los experimentos a realizar en esta actividad son los siguientes:

- Se crearán los siguientes conjuntos de cadeanas aleatorias de caracteres:
  - A: 1000 cadenas de longitud 4, con números aleatorios (del 0 al 9)
  - B: 1000 cadenas de longitud 8, con números aleatorios (del 0 al 9)
  - C: 1000 cadenas de longitud 16, con números aleatorios (del 0 al 9)
  - D: 1000 cadenas de longitud 32, con números aleatorios (del 0 al 9)
  - E: 1000 cadenas de longitud 64, con números aleatorios (del 0 al 9)

- Se implementarán los algoritmos "Shift-And" y "Naïve".

- Se buscarán las cadenas de cada conjunto (A,B,C,D,E) en el documento `p1-m.txt` (el cual incluye el primer millón de dígitos de pi) con ambos algoritmos y se comparará por medio de un boxplot el tiempo acumulado de estas

Conjuntos de cadenas

Lo primero que se hizo fue ejecutar la apertura del documento que contiene el primer millón de carácteres de pi.

Posteriormente se creó una función `generate_random_patterns` la cual sirve para generar las cadenas aleatorias que finalmente se añadirán al conjunto. Esta función toma como entrada el número de cadenas a generar y la longitud de esta.

Dentro de la función:

- Se declara la lista `patterns` que será la que contenga las cadenas aleatorias.
- Se entra a un ciclo que correrá la cantidad de cadenas a generar, en el cual se crea un `pattern` siendo este la union de elecciones aleatorias de entre el 0 y 9, con la longitud siendo la que se declaró en el parámetro de entrada, esto se insertará en la lista `patterns`.
- Se devuelve la lista de `patterns`.

Aquí se declaran los 5 conjuntos (`A` , `B` , `C` , `D` y `E`) con la cantidad de cadenas y la longitud de estas (4, 8, 16, 32 y 64, respectivamente).

In [20]:
```python
import random

# Leer el contenido del archivo 'pi-1m.txt'
with open('pi1-m.txt', 'r') as file:
    T = file.read().strip()

# Generar conjuntos A, B, C, D, E
def generate_random_patterns(num_patterns, pattern_length):
    patterns = []
    for _ in range(num_patterns):
        pattern = ''.join(random.choices('0123456789', k=pattern_length))
        patterns.append(pattern)
    return patterns

A = generate_random_patterns(1000, 4)
B = generate_random_patterns(1000, 8)
C = generate_random_patterns(1000, 16)
D = generate_random_patterns(1000, 32)
E = generate_random_patterns(1000, 64)
```

Implemetación Shift-And

La implementación de este algoritmo está traducida a python del código que se proporcionó en Julia.

Este algoritmo está dividido en dos funciones:

- La función `pattern` que toma una cadena `pat` y devuelve un diccionado con los patrones de bits
- La función `search` que toma el texto `text`, la cadena `pat` y una lista `L` opcional para almacenar los índices de coincidencias.

Dentro de la función `pattern` se ejecuta lo siguiente:

- Se inicializa un diccionario vacío.
- Se itera sobre los índice de la cadena `pat` y se actualiza el diccionario de bits.
- Se devuelve el diccionario de bits.

Dentro de la función `search` se ejecuta lo siguiente:

- Uso de la función `pattern` para obtener el diccionario de patrones.
- Se inicializa una variable `S`.
- Se guarda la longitud del patrón en la variable `plen`.
- Se itera sobre los índices del texto `text`, se actualiza `S` y se verifica si hay coincidencias.
- Si hay coincidencias, se añade el índice a la lista `L`.
- Se devuelve la lista `L`.

In [21]:
```
def pattern(pat):
    D = {}
    for i, c in enumerate(pat):
        d = D.get(c, 0)
        d |= 1 << i
        D[c] = d
    return D

def search(text, pat, L=None):
    if L is None:
        L = []
    D = pattern(pat)
    S = 0
    plen = len(pat)
    m = 1 << (plen - 1)
    for i, c in enumerate(text):
        d = D.get(c, 0)
        S = ((S << 1) | 1) & d
        if S & m > 0:
            L.append(i - plen + 1)
    return L
```

Para probar la implementación del algoritmo se ejecutaron algunas pruebas, las cuales devolvieron resultados correctos.

In [22]:
```
search('abracadabra', 'abr')
```

Out[22]:
```
[0, 7]
```

In [23]:
```
search('mississippi', 'ss')
```

Out[23]:
```
[2, 5]
```

In [24]:
```
search('mississippi', 'i')
```

Out[24]:
```
[1, 4, 7, 10]
```

Implementación Naïve

Para implementa el algoritmo `Naïve` se creo la función `naive_search` que toma el texto `text` y el patrón `pat` como argumentos.

Dentro de la función:

- Se inicializa una lista L para almacenar los índices de coincidencias.
- Se obtiene la longitud del texto y del patrón.
- Se itera sobre el texto desde el inicio hasta la posición donde el patrón encaja completamente.
- Por cada iteración, se verifica si el segmento del texto coincide con el patrón.
- Si es una coincidencia, se agrega el índice a la lista `L`.
- Se devuelve la lista `L`.

In [25]:
```
def naive_search(text, pat):
    L = []
    n = len(text)
    m = len(pat)
    for i in range(n - m + 1):
        if text[i:i + m] == pat:
            L.append(i)
    return L
```

Para comprobar, se ejecutaron las mismas pruebas que para el algoritmo Shift-And, obteniendo los mismos resultados correctos.

In [26]:
```
naive_search('abracadabra', 'abr')
```

Out[26]:
```
[0, 7]
```

In [27]:
```
naive_search('mississippi', 'ss')
```

Out[27]:
```
[2, 5]
```

In [28]:
```
naive_search('mississippi', 'i')
```

Out[28]:
```
[1, 4, 7, 10]
```

Busqueda en dígitos de pi

Se crea la función `measure_time` para medir los tiempos de ejecución para cada patrón en cada conjunto utilizando los dos algoritmos.

In [29]:
```
import time

# Medir tiempos
def measure_time(patterns, algorithm):
```

```
    times = []
    for pat in patterns:
        start_time = time.time()
        algorithm(T, pat)
        end_time = time.time()
        times.append(end_time - start_time)
    return times
```

Con la función ya definida, se crearon diccionarios que contienen las listas de tiempos y resultados de usar dicha función

In [30]:
```
# Medir tiempos para cada conjunto y algoritmo
times_shift_and = {
    'A': measure_time(A, search),
    'B': measure_time(B, search),
    'C': measure_time(C, search),
    'D': measure_time(D, search),
    'E': measure_time(E, search)
}

times_naive = {
    'A': measure_time(A, naive_search),
    'B': measure_time(B, naive_search),
    'C': measure_time(C, naive_search),
    'D': measure_time(D, naive_search),
    'E': measure_time(E, naive_search)
}
```

Boxplot

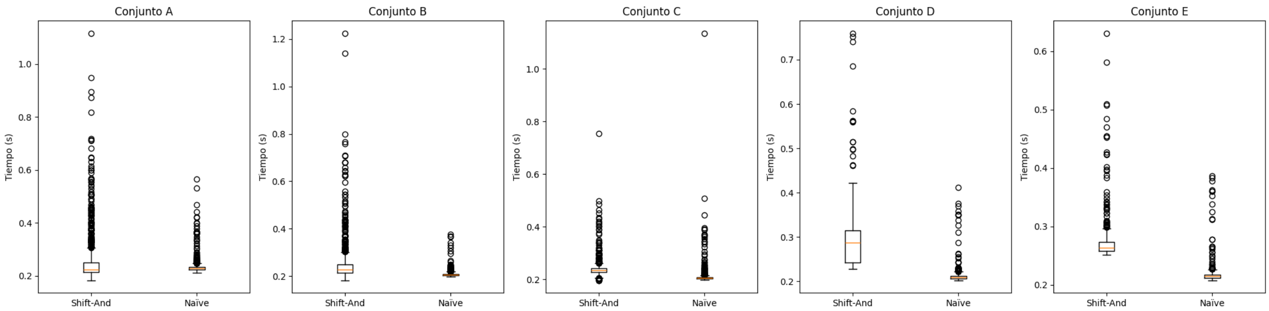
Se creó la función `plot_boxplots` para mostrar los resultados de todos los conjuntos y ambos algoritmos en el mismo gráfico

In [ ]:
```
import matplotlib.pyplot as plt

# Generar boxplots
def plot_boxplots(times_shift_and, times_naive):
    fig, axs = plt.subplots(1, 5, figsize=(20, 5))
    for i, key in enumerate(times_shift_and.keys()):
        axs[i].boxplot([times_shift_and[key], times_naive[key]], tick_labels=['
        axs[i].set_title(f'Conjunto {key}')
        axs[i].set_ylabel('Tiempo (s)')
    plt.tight_layout()
    plt.show()

plot_boxplots(times_shift_and, times_naive)
```

```
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_22299/81641479.py:54:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renam
ed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  axs[i].boxplot([times_shift_and[key], times_naive[key]], labels=['Shift-And',
'Naïve'])
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_22299/81641479.py:54:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renam
ed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  axs[i].boxplot([times_shift_and[key], times_naive[key]], labels=['Shift-And',
'Naïve'])
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_22299/81641479.py:54:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renam
ed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  axs[i].boxplot([times_shift_and[key], times_naive[key]], labels=['Shift-And',
'Naïve'])
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_22299/81641479.py:54:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renam
ed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  axs[i].boxplot([times_shift_and[key], times_naive[key]], labels=['Shift-And',
'Naïve'])
/var/folders/v3/6n107fw10yb9ryc5t5mmqw5c0000gp/T/ipykernel_22299/81641479.py:54:
MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renam
ed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped
in 3.11.
  axs[i].boxplot([times_shift_and[key], times_naive[key]], labels=['Shift-And',
'Naïve'])
```



In [ ]:

Discusión de los resultados

En la gráfica del conjunto A se observa que el algoritmo Shift-And tiene tiempos de ejecución significativamente mejores en comparación con el Naïve, ya que, se observa la mediana en la parte inferior de la caja. El algoritmo Naïve parece ser más constante en sus tiempos de ejecución, sin embargo, estos son ligeramente más lentos para las busquedas de cadenas de 4 caracteres.

Para la gráfica del conjunto B se observa un comportamiento similar a la del conjunto A, sin embargo vemos que los tiempos del algoritmo Naïve cada vez se acerca más al límite inferior del algoritmo Shift-And; a pesar de que este último sigue presentando tiempos más veloces en algunos casos, se podría considerar mejor el algoritmo Naïve debido a su poca dispersión y gran constancia en tiempos.

La gráfica del conjunto C nos da resultados similares a la del conjunto B, con la diferencia de que el algoritmo Shift-And parece tener mas constancia con esta longitud de caracteres, siendo la comparación con el Naïve más dificil. Debido a que el Naïve tiene la mayoría de sus tiempos ligeramente mejores a la mayoría del Shift-And, se concluye que este es mejor.

En el conjunto C se observa en su gráfica ya una gran diferencia de tiempos entre los algoritmos. En este caso, el Naïve es el claro ganador. En el algoritmo Shift-And se observa una mayor dispersión en los tiempos y la mediana se ubica en la parte superior de la caja, así como se encuentra por encima de los tiempos del naïve.

Finalmente, en la gráfica del conjunto D observamos que el Naïve mantiene un comportamiento similar y el Shift-And, a pesar de ser más constante en sus tiempos, es bastante más lento que el Naïve.

Observaciones y conclusiones

Con lo discutido en los resultados, se puede observar que en la mayoría de los casos, el algoritmo Naïve es mejor en tiempos de procesamiento que el Shift-And, sin embargo, este último podría ser utilizado sin problemas en casos de patrones de caracteres de longitud pequeña, ya que suele tener los mejores tiempos.

Es digno resaltar los tiempos de procesamiento del algoritmo Naïve, ya que, sin importar la longitud del patrón a buscar, siempre mantuvo los tiempos y los casos extremos constantes en los 5 casos; a diferencia del algoritmo Shift-And que los tiempos más constantes fueron en el caso del conjunto C.

La principal observación de estos experimentos es que no se diferencían los tiempos de cuando se encuentra el caracter y cuando no, sin embargo, no se considera que esto pueda afectar los resultados del experimento. Si un patrón no se encuentra con el algoritmo Shift-And, tampoco se encontraría con el algoritmo Naïve, lo cual provocaría que el tiempo de procesamiento de ambos sea aquel que tarde en recorrer todos los caracteres de pi.

Otra observación sería que el documento de caracteres de pi inicia con "3 . ", los cuales no se eliminaron y podrían afectar el procesamiento, sin embargo, igual que en el ejemplo anterior esto afectaría a todos los casos, provocando una igualdad de condiciones para cada prueba.

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms, third edition. http://portal.acm.org/citation.cfm?id=1614191