Análisis de Algoritmos y Estructuras para Datos Masivos 2023

Unidad 7 - Tarea

Búsqueda de patrones en cadenas

David Aarón Ramírez Olmeda

Introducción

La búsqueda de patrones en cadenas de texto es una tarea muy común en el procesamiento de lenguaje natural, donde se busca encontrar una cadena de caracteres específica dentro de un texto más grande. Los algoritmos utilizados para esta tarea son fundamentales para una variedad de aplicaciones, desde motores de búsqueda en la web hasta la detección de virus informáticos.

El algoritmo Naive es un método simple y directo para encontrar patrones en una cadena de texto. Sin embargo, este algoritmo no es muy eficiente y puede ser muy lento cuando se trabaja con cadenas de texto más grandes.

El algoritmo de Shift-and es un método más eficiente para buscar patrones en cadenas de texto. Este algoritmo se basa en la utilización de operaciones lógicas y una tabla de preprocesamiento para reducir la complejidad de la búsqueda.

En esta tarea se compararon estos dos algoritmos en términos de tiempo de ejecución utilizando diferentes tamaños de texto y patrones utilizando la cadena pi como base.

Desarrollo

In [1]:
```
import time
import random
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
def shift_and(pattern, text):
    m = len(pattern)
    n = len(text)
    if m > n:
        return []
    masks = {c: 0 for c in alphabet}
    for i in range(m):
        masks[pattern[i]] |= 1 << (m - i - 1)
    matches = []
    R = ~(1 << (m - 1))
    S = 0
    for i in range(n):
        S = ((S << 1) | 1) & masks[text[i]]
        if S & (1 << (m - 1)):
            if i - m + 1 > R:
                R = i - m + 1
                matches.append(R)
    return matches
```

In [3]:
```
def naive(pattern, text):
    matches = []
    m = len(pattern)
    n = len(text)
    for i in range(n - m + 1):
        if text[i:i + m] == pattern:
            matches.append(i)
    return matches
```

In [4]:
```
A = ["".join([str(random.randint(0, 9)) for X in range(4)])  for X in ran
B = ["".join([str(random.randint(0, 9)) for X in range(8)])  for X in ran
C = ["".join([str(random.randint(0, 9)) for X in range(16)]) for X in ran
D = ["".join([str(random.randint(0, 9)) for X in range(32)]) for X in ran
E = ["".join([str(random.randint(0, 9)) for X in range(64)]) for X in ran
```

Tomamos nuestro alfabeto de símbolos (0 - 9) y generamos las cadenas aleatorias concatenando símbolos.

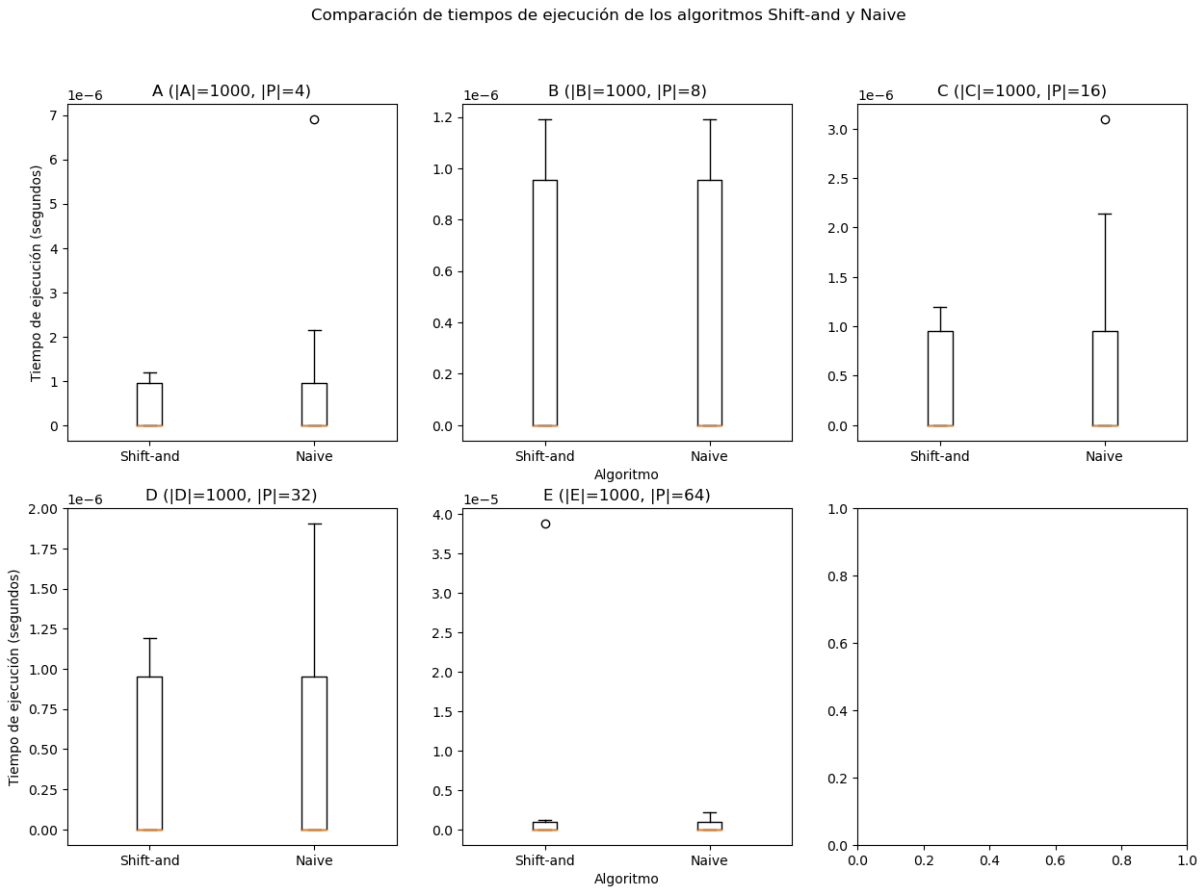
In [5]:
```
with open("pi-1m.txt", "r") as f:
    pi = f.read()
```

In [8]:
```
shift_and_times = []
naive_times = []
patterns = [A, B, C, D, E]

for p_list in patterns:
    shift_and_times_p = []
    naive_times_p = []

    for p in p_list:
        start_time = time.time()
        shift_and(pi, p)
        shift_and_times_p.append(time.time() - start_time)

        start_time = time.time()
        naive(pi, p)
        naive_times_p.append(time.time() - start_time)

    shift_and_times.append(shift_and_times_p)
    naive_times.append(naive_times_p)
```

Medimos los tiempos de ejecución de ambos algoritmos para cada conjunto de datos de prueba.

In [14]:

```
tiempos_A = [shift_and_times[0], naive_times[0]]
tiempos_B = [shift_and_times[1], naive_times[1]]
tiempos_C = [shift_and_times[2], naive_times[2]]
tiempos_D = [shift_and_times[3], naive_times[3]]
tiempos_E = [shift_and_times[4], naive_times[4]]

etiquetas = ['Shift-and', 'Naive']
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15,10))
fig.suptitle('Comparación de tiempos de ejecución de los algoritmos Shift

axs[0,0].boxplot(tiempos_A, labels=etiquetas)
axs[0,0].set_title('A (|A|=1000, |P|=4)')
axs[0,0].set_ylabel('Tiempo de ejecución (segundos)')

axs[0,1].boxplot(tiempos_B, labels=etiquetas)
axs[0,1].set_title('B (|B|=1000, |P|=8)')
axs[0,1].set_xlabel('Algoritmo')

axs[0,2].boxplot(tiempos_C, labels=etiquetas)
axs[0,2].set_title('C (|C|=1000, |P|=16)')

axs[1,0].boxplot(tiempos_D, labels=etiquetas)
axs[1,0].set_title('D (|D|=1000, |P|=32)')
axs[1,0].set_ylabel('Tiempo de ejecución (segundos)')

axs[1,1].boxplot(tiempos_E, labels=etiquetas)
axs[1,1].set_title('E (|E|=1000, |P|=64)')
axs[1,1].set_xlabel('Algoritmo')

plt.show()
```



Conclusión

Hemos implementado dos algoritmos para buscar ocurrencias de un patrón en una cadena de texto: el algoritmo Naive y el algoritmo Shift-and.

Al comparar los tiempos de ejecución de ambos algoritmos, vimos que Shift-and es más rápido en la mayoría de los casos. Sin embargo, en algunos casos el tiempo de ejecución es similar y en otros casos Naive puede ser más rápido, especialmente cuando el patrón es muy pequeño, existe cierte variabilidad dependiendo del patrón.

Se puede concluir que el algoritmo Shift-and es una opción altamente efectiva para buscar ocurrencias de patrones en cadenas de texto, particularmente en casos donde los patrones y las cadenas son grandes. En algunos casos puede ser necesario evaluar diferentes algoritmos (Naive) y opciones para lograr un mejor rendimiento o adaptación a las necesidades particulares.

In [ ]: