.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
La mafia de los algoritmos Greedy

10 de Abril de 2025

Franco Guardia
109374

Mateo Requejo
110109

Santino
Regidor
112147

1. Introducción

Trabajamos para la mafia de los amigos Amarilla Pérez y el gringo Hinz y en estos momentos hay un problema: alguien les está robando dinero. No saben bien cómo, no saben exactamente cuándo, y por supuesto no saben quién. Evidentemente quien lo está haciendo es muy hábil.

La información con la que contamos son n transacciones sospechosas de las que tenemos un *timestamp* aproximado. Es decir tenemos n tiempos t_i con un posible error e_i. Por lo tanto sabemos que dichas transacciones fueron realizadas en el intervalo $[t_i - e_i; t_i + e_i]$

Un interrogado dio el nombre de alguien que podría ser *la rata* y El Gringo nos pidió revisar las transacciones de dicha persona que casualmente también eran n. Lo que queda saber es si dichas transacciones condicen con los *timestamps* aproximados que habíamos obtenido previamente

Se nos pide implementar un algoritmo Greedy que determinen si las transacciones coindicen con los intervalos o no y por lo tanto se pueda saber si el sospechoso es *la rata* o no

2. Análisis del problema

En este trabajo realizaremos el análisis teórico y empírico de un algoritmo Greedy propuesto para indicar si las n transacciones del sospechoso condicen con los intervalos aproximados o no y poder determinar así si es *la rata* o no

En la situación planteada dada una cantidad n de transacciones que se asumen están ordenadas y una cantidad n de intervalos aproximados t_i cada uno con su correspondiente error e_i lo que se busca es que cada transacción esté asociada a uno de los intervalos, para que de esta forma un total de n transacciones estén cubiertas por los n intervalos aproximados

Puede ocurrir que una transacción sea cubierta por más intervalos, pero necesariamente debemos ser capaces de asociarle uno a cada una de ellas. En caso de que haya una transacción que no sea cubierta se concluye que el sospechoso no es *la rata*, o también si dada una cierta asignación de intervalos a transacciones queda alguna que no puede asignarse a ningún otro (donde se concluye lo mismo que el caso anterior)

Considerando esto es esperable que busquemos aprovechar lo máximo posible la longitud de los intervalos y para esto asignemos aquella transacción que haya ocurrido antes en el tiempo a aquel intervalo con centro en un tiempo menor, ya que este en general cubrirá una franja de tiempo que otros no y al hacer esto también se evita hacer una asignación que deje a una transacción sin asignar lo que puede causar un resultado obtenido erróneo en ciertos casos. De esta forma se garantiza que en caso de no poder hacer una asignación uno a uno de intervalos y transacciones es por la propia distribución de estos y nuestro algoritmo solo halló la respuesta más obvia en esa situación

3. Algoritmo

3.1. Explicación

Para la resolución del problema, se buscó un algoritmo que permita resolver de manera óptima todos los casos posibles que nos presenta el problema: Tanto los casos triviales, cuando se puede asociar una transacción de un sospechoso con un único timestamp aproximado, como los casos mas complejos, cuando hay intersección entre los timestamps aproximados y tenemos varias transacciones del sospechoso incluídas en la misma.

Para eso, pensamos en la siguiente idea: Ordenar los timestamps aproximados por tiempo de fin. Es decir, dado un tiempo t_i con un error e_i, ordenamos de menor a mayor los intervalos tomando a t_i+e_i. Luego, iterando por los timestamps del sospechoso, que ya vienen ordenados, los asociamos con el primer intervalo aproximado al que pertenezcan. Es decir, si pertenece a un único intervalo,

será asignado a ese. Pero si tenemos una intersección donde un mismo timestamp del sospechoso pertenece a más de un intervalo aproximado, se lo asociará al de menor tiempo de fin.

Cuando un intervalo queda asociado a una transacción del sospechoso, el mismo se deja de considerar para otras transacciones. Si no se logra asignar una transaccion a un intervalo, sabemos que esa persona no es "la rata"del problema. Esto podemos afirmarlo ya que tenemos n intervalos y n transacciones, por lo que cada transacción debe estar asociada a un único intervalo.

3.2. Código

A continuación, presentamos el código del algoritmo explicado.

```
def es_rata(intervalos: list[int, int], sospechosas: list[int]) -> list[list]:
    evidencia = []
    for sospechoso in sospechosas:
        asigne = False
        for i in range(len(intervalos)):
            hora = intervalos[i][0]
            error = intervalos[i][1]
            if hora == -1:
                continue
            if sospechoso > hora + error or sospechoso < hora - error:
                continue
            else:
                evidencia.append([sospechoso, hora, error])
                intervalos[i] = (-1, -1) #Asi marcamos como 'usado' al intervalo
                asigne = True
                break
            if not asigne:
                break
    return evidencia
```

Para saber si la persona es la rata o no, basta con comparar la cantidad de asociaciones entre transacciones e intervalos con la cantidad de intervalos totales iniciales (n)

```
def main():
    evidencia = es_rata(intervalos, transacciones_sospechoso)
    if len(evidencia) < len(intervalos):
        print("No es el sospechoso correcto")
        return
```

3.3. Visualización

Veamos algunos ejemplos de manera visual para entender el funcionamiento del algoritmo. Por facilidad, tomaremos $n = 2$

1. Caso trivial: Una única transacción perteneciente a un único timestamp aproximado. Empezaremos iterando por T1, que únicamente pertenece al intervalo A. Los asociamos entre sí y quitamos a A de los intervalos considerados. Seguimos por T2, también pertenece a un único intervalo, entonces lo asociaremos a ese mismo. Encontramos que cada transacción del sospechoso está asociada a un intervalo aproximado, podemos concluir que esa persona es la rata.

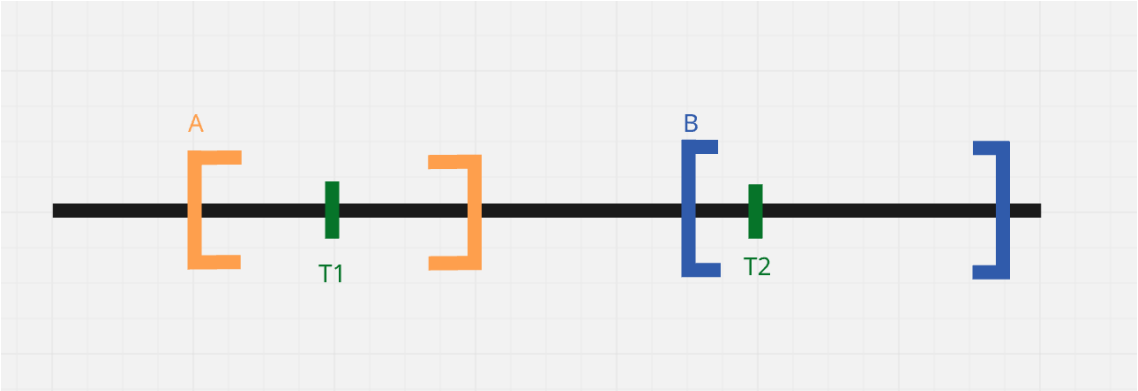

Figura 1: Caso trivial - Una única transacción perteneciente a un único timestamp aproximado

Notar que si T1 o T2 estuviesen fuera de A y de B, quedarían intervalos vacíos por lo que esta persona no sería la rata.

2. En el caso de la intersección entre A y B, si ninguna de las transacciones del sospechoso pertenece a la intersección, la lógica será la misma que el punto anterior. Empezaremos iterando por T1, únicamente vemos que pertenece a A, entonces los asociamos. Quitamos a A de los intervalos a considerar, y seguimos con T2. Vemos que solo pertenece a B, entonces los asociamos. No queda intervalo vacío, esta persona es la rata.

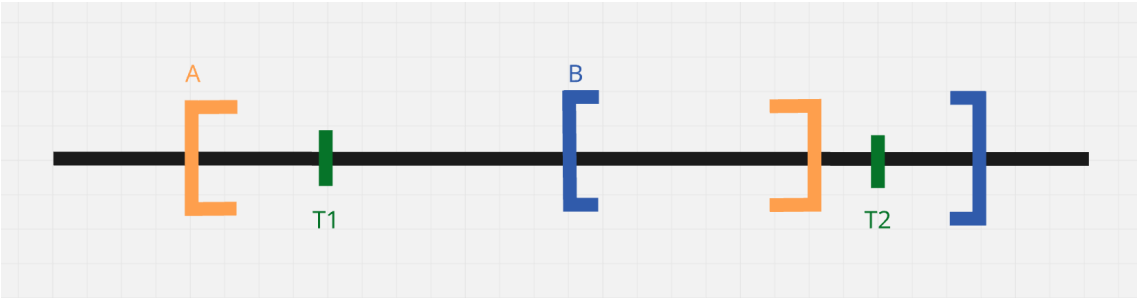

Figura 2: Intersección de A con B. T1 y T2 no pertenecen a la intersección

3. Si tenemos el mismo caso pero una de las transacciones del sospechoso pertenece a la intersección, nuestro algoritmo detecta esto sin problema. Empezamos iterando por T1 y vemos que pertenece a 2 intervalos. Como mencionamos, estos están ordenados por tiempo de fin, por lo tanto, el primer intervalo que evaluaremos será A. Como T1 pertenece a A, los asociamos entre sí y eliminamos a A.

Unicamente nos queda T2. Solo pertenece al intervalo B, y a ese mismo será asociado. No nos quedó ningún intervalo vacío, esta persona es la rata.

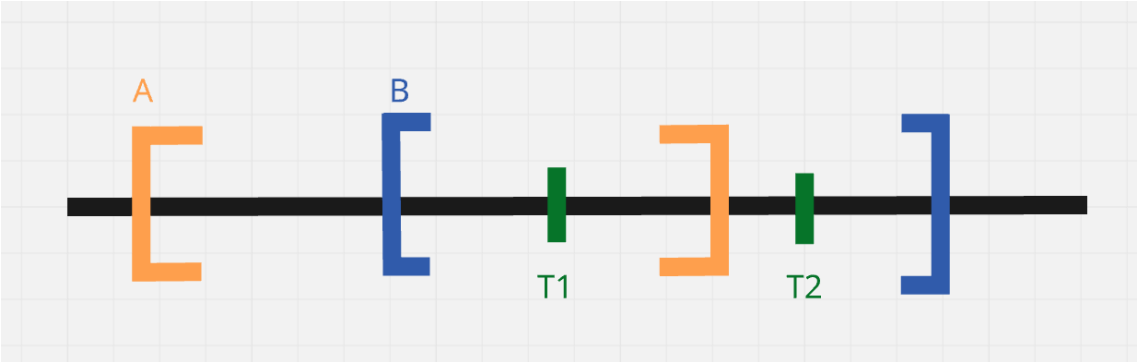

Figura 3: T1 pertenece a la intersección entre A y B

Podemos tener el mismo caso pero T2 siendo la primer transacción, tal como se ve en la siguiente figura:

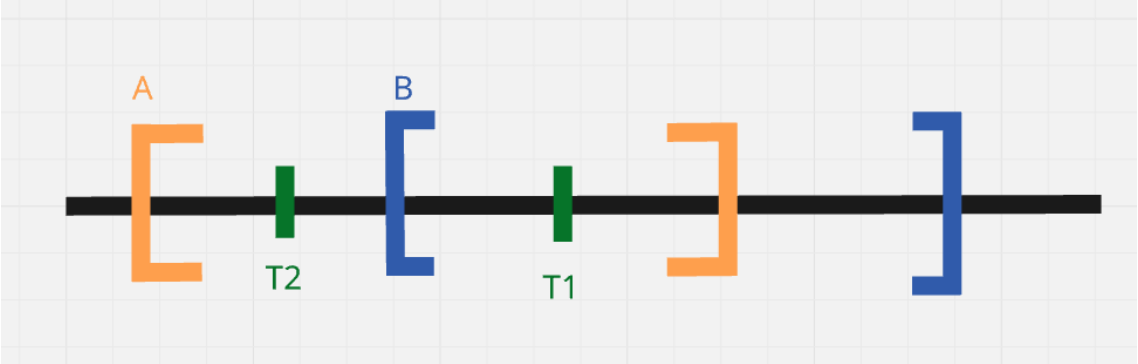

Figura 4: T2 no está en la intersección y aparece antes que T1

En este caso, empezaremos iterando por T2. Únicamente pertenece a A, los asociamos entre sí y eliminamos a A. Luego, seguimos iterando por T1 y únicamente corresponderá vincularlo a B ya que A fue eliminado.

4. Ambas transacciones pueden pertenecer a la intersección de A y B. En este caso, empezamos iterando por T1. Vemos que pertenece a ambos intervalos, y lo asociaremos a A ya que en nuestro ordenamiento es el que viene primero por tener menor tiempo de fin. Eliminamos a A de los intervalos a considerar. Seguimos iterando por T2 y vemos que únicamente pertenece a B. Ningún intervalo quedó vacío, esta persona es la rata.

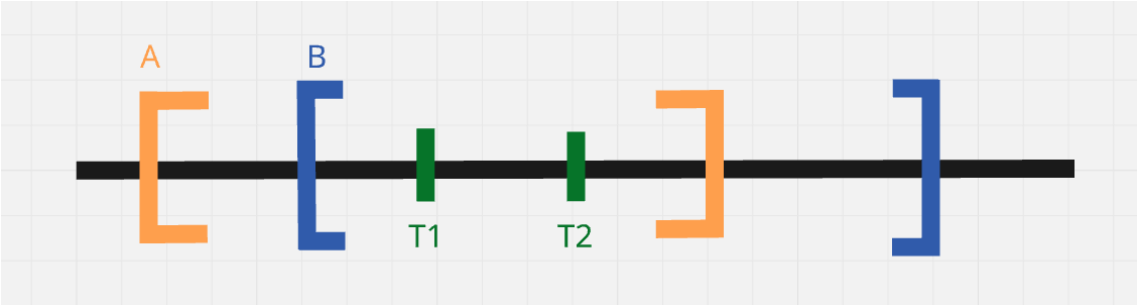

Figura 5: T1 y T2 pertenecen a la intersección de A y B

3.4. ¿Por qué es un algoritmo Greedy?

El algoritmo es Greedy porque en cada paso se toma la decisión local de asignar un timestamp sospechoso al primer intervalo (ordenado por tiempo de fin) en el que encaja, sin retroceder ni reconsiderar asignaciones previas. Es decir, consideramos nuestro estado local que se va actualizando constantemente con cada paso. Esta elección se basa en el razonamiento de que, al usar el intervalo que termina antes, se "liberan"(o se preservan) los intervalos que terminan más tarde para poder asignarles los timestamps restantes, maximizando así las posibilidades de asignar correctamente todas las transacciones. Esta es nuestra regla Greedy que nos encuentra nuestro óptimo local en cada iteración.

Para cada timestamp del sospechoso (que ya vienen ordenados), se recorre la lista de intervalos hasta encontrar aquel en el que encaja. Esa elección es localmente óptima porque:

- **Minimiza el desperdicio:** Al usar el intervalo que finaliza más pronto, se evita robarün intervalo con mayor flexibilidad, que podría acomodar un timestamp que llega más tarde.
- **Reducción de la solución parcial:** Una vez asignado, el intervalo se marca como usado, lo que reduce el espacio de búsqueda para los timestamps restantes.

Paso a paso nuestro estado local es único, y todo lo que "quedó atrás"podemos considerar que fue asignado correctamente: solo nos interesa ver nuestro estado actual y los intervalos restantes, y cuál es la mejor decisión que puedo tomar con ellos. Para cada estado local aplicaremos nuestra regla Greedy hasta alcanzar el óptimo global: Que todos los intervalos queden con una transacción del sospechoso asignada, o no, para así poder decidir si la persona es la rata o no lo es.

4. Demostración correctitud

El algoritmo es_rata nunca produce falsos positivos (es decir, no asigna una transacción a un intervalo que no pertenece) ni falsos negativos(si una transacción pertenece a un intervalo, entonces siempre será asignada) Esto se demostrará por contradicción. suponemos:

1. Sea $I = \{[hiei, hi + ei]\}$ ninguno de los intervalos se intersecan.
2. Sea $T = \{t1, t2, \dots\}$ las transacciones sospechosas.

Supongamos, que ocurre un **falso positivo** que se asigna t a un intervalo $[h, he]$ y $t[he, h + e]$, esto jamás puede pasar porque contradice esta condición

```
            if sospechoso > hora + error or sospechoso < hora - error:
                continue
            else:
                evidencia.append([sospechoso, hora, error])
                intervalos[i] = (-1, -1)
                asigne = True
                break
```

tampoco se puede asignar dos transacciones a un mismo intervalo porque como ya se explico al principio del informe, los intervalos ya asignados se sobreescriben para que no se vuelvan a utilizar. Por lo tanto no puede ocurrir un falso positivo.

Ahora para los **falsos negativos** supongamos que una transacción cumple $t[he, h + e]$ pero no fue asignada. El algoritmo evalúa los intervalos en orden, y si encuentra uno donde (t[he,h+e], lo asigna inmediatamente. Dado que no hay intersección de intervalos y como cada transacción se asigna a un único intervalo una única vez, y los intervalos son únicos y disjuntos, lo único que puede pasar para que de un falso negativo en esta situación es que no lo asigne cosa que contradice el código mostrado anteriormente.

Ahora pasaremos a la demostración en los casos donde haya una intersección siendo x las transacciones del sospechoso. Mientras que A y B son los intervalos y con la definición de intersección:

$$A \cap B = \{x \mid x \in A \wedge x \in B\} \tag{1}$$

Demostraremos que el algoritmo siempre produce una solución correcta a través de una demostración por inversiones. Primero vamos a definir que es una inversión, el schedule tiene una inversión cuando se tiene una transacción del sospechoso i, asignada a un intervalo f_i y otra transacción del sospechoso j, asignada a un intervalo de fin f_j. $i, j \in F_i \cap F_j$

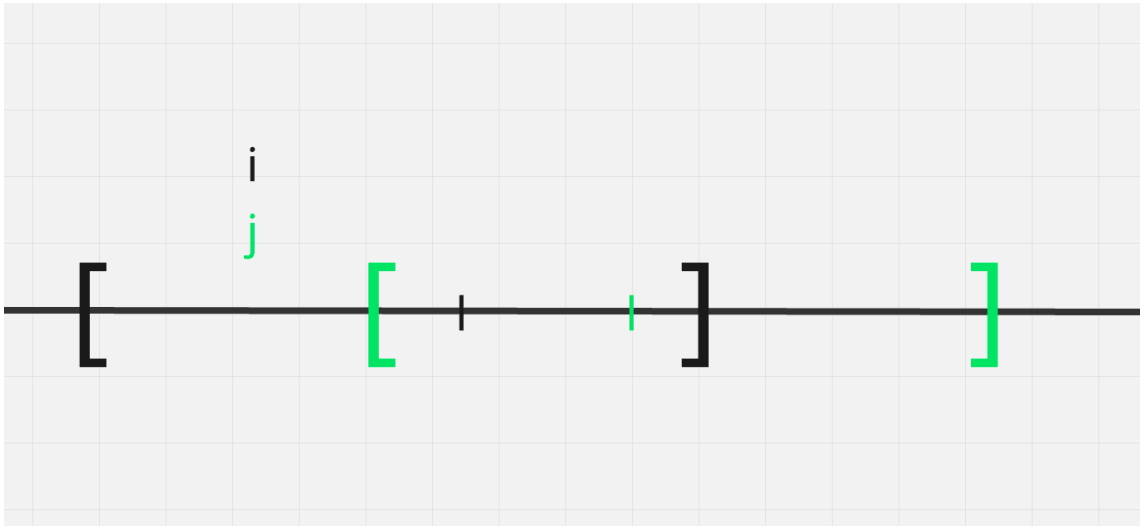

Figura 6: Transacciones inversibles

Postulado: Todos los schedules sin inversiones posibles (y donde la cantidad de intervalos es igual a la cantidad de transacciones) solo tienen dos soluciones correctas.

Demostración: Por no tener inversiones, sólo puede ocurrir que asigne todas las transacciones a su único intervalo posible (es rata) o que no asigne por lo menos una (no es rata).

Un ejemplo de inversión tomando la figura anterior sería cambiar el intervalo asignado a i por el asignado a j.

Ahora supongamos que hay una solución correcta con inversiones, donde el sospechoso es rata. En este caso por la definición de intersección de conjuntos, invertir las asignaciones dos a dos de transacciones que cumplan la condición de estar en una intersección siempre es válida porque las transacciones van a seguir perteneciendo al nuevo intervalo que se les asignó.

Para demostrar esto vamos a primero recordar la definición de intersección de conjuntos:

$$A \cap B = \{x \mid x \in A \text{ y } x \in B\}.$$

Luego supongamos que tenemos dos transacciones $t_1, t_2 \in I_i \cap I_j$ con t_1 asignadas al intervalo I_i y t_2 asignada al intervalo I_j. Entonces, por la definición anterior, podemos decir que:

$$t_1 \in I_i \quad \text{y} \quad t_1 \in I_j, \quad t_2 \in I_i \quad \text{y} \quad t_2 \in I_j$$

tengo estas asignaciones:

$t_1 \mapsto I_i, \quad t_2 \mapsto I_j$

por lo dicho anteriormente es válido invertirlas y quedan:

$t_1 \mapsto I_j, \quad t_2 \mapsto I_i$

como no se alteran a los intervalos ni a las transacciones se sigue cumpliendo la intersección y las pertenencias lo que hace que haciendo sucesivas inversiones desde la hipotética solución correcta

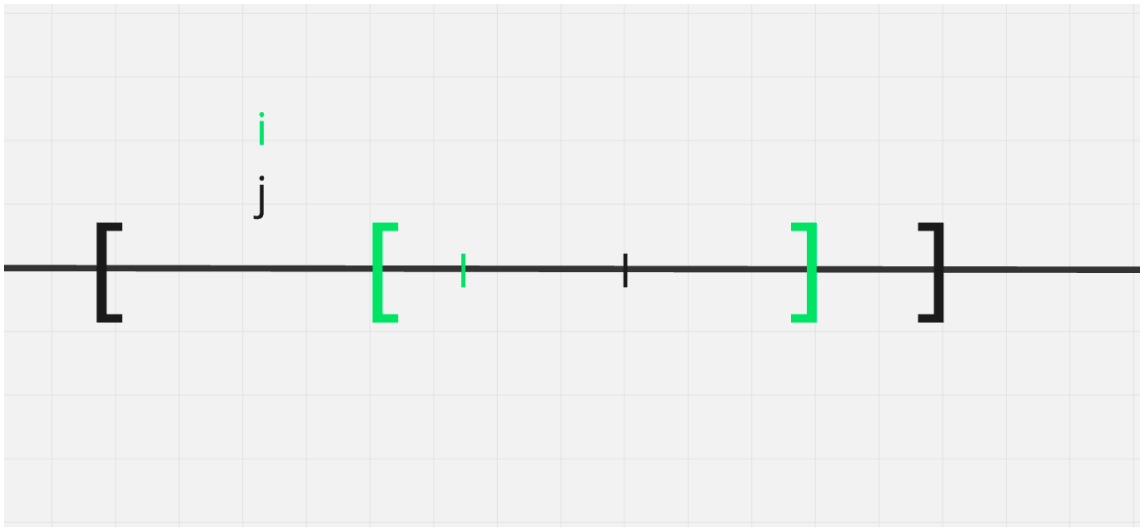

Figura 7: Enter Caption

se puede llegar a nuestra solución. Con ello, demostramos que nuestra solución es igual de correcta que cualquier otra.

Con esto demostramos: 1- Que en las intersecciones de dos intervalos cualquier asignación es valida (siempre que ambas transacciones pertenezcan a la intersección). 2- Que cualquier schedule sin inversiones solo tiene dos soluciones. 3- Nuestro algoritmo siempre da la solución correcta en ambos casos (no da falsos negativos ni falsos positivos).

5. Complejidad

5.1. Análisis

Tenemos 2 listas iniciales de n elementos: La que contiene a los timestamps del sospechoso, y la que contiene a los intervalos aproximados. Desestimamos la complejidad de lectura de los archivos de entrada para armar la lista ya que es lineal.

En primer lugar, ordenamos la lista de intervalos usando el ordenamiento de Pyhton, lo que tiene una complejidad de $O(nlog(n))$. En nuestro algoritmo, la iteración principal es de todas las transacciones del sospechoso (n). Para cada una de ellas, recorremos todos los intervalos para ver a cuáles pertenecen. En el peor de los casos, debemos recorrer la lista de intervalos completamente. Por lo tanto, el ciclo principal del algoritmo tiene una complejidad de $O(n^2)$.

Para cada intervalo, verificamos si la transacción pertenece al mismo ($O(1)$). Si pertenece, marcamos al intervalo como üsado"guardando -1 en sus valores ($O(1)$). Verificar si la persona es o no la rata es simplemente una comparación de tamaños, lo que es $O(1)$, e imprimir los resultados en caso de que la persona sea la rata es $O(n)$.

Entonces, en general, nuestro algoritmo tiene una complejidad de $O(n^2)$.

5.2. Variabilidad en los valores de entrada

La variabilidad en los valores de entrada no afecta en la complejidad del algoritmo. Esto porque sin importar cómo sean los intervalos, si son más grandes o más chicos o cómo estén dispuestos entre sí, siempre que haya que asignar una transacción a uno de ellos se los recorre de principio a fin buscando el primero que esté libre para que se la asigne a ese. En este caso el "primero"se refiere a aquel que tenga menor fin por el ordenamiento que se hace. El tiempo que tarde el algoritmo

puede llegar a ser mayor si este tiene que hacer un mayor recorrido hasta asignar un intervalo a una transacción (por ejemplo porque se trata de intervalos que estén más concentrados en el final del rango de tiempo, en lugar de que estén más distribuidos a lo largo de todo el tiempo).

Lo que se tiene es una lista ordenada de intervalos según tiempo de fin. Sin importar si los intervalos son muy pequeños, muy grandes, muy juntos entre sí, muy separados, siempre vamos a iterar N intervalos para verificar que la transacción del sospechoso pertenezca a alguno de ellos. Las comparaciones se realizan en tiempo O(1) ya que se verifica si la transacción del sospechoso está entre el inicio y el fin del intervalo. Por otro lado, lo que sí podría modificar la complejidad sería el ordenamiento de Python, ya que si los intervalos vienen muy ordenados o desordenados el desempeño del sort podría ser diferente, pero eso es algo externo a nuestro algoritmo.

5.3. Mediciones

A modo de complemento del análisis de complejidad se realizaron mediciones de tiempo del algoritmo de es_rata con sets de datos generados con el módulo random semi-aleatorios y el siguiente algoritmo.

```
MIN_ERROR = 5
# El maximo error es el minimo de las horas
# El salto del error es la mitad del MIN_HORA

# Estos par metros determinan el minimo y el salto de las transacciones sospechosas
MIN_HORA = 500
JUMP = 15
def generar_datos(size=50000):

  # Generamos las horas
    horas = [np.random.randint(MIN_HORA, MIN_HORA + 1)]
    for _ in range(size - 1):
        horas.append(horas[-1] + np.random.randint(0, JUMP))

  # Agregamos los errores
    intervalos = [(hora, np.random.randint(MIN_ERROR, MIN_HORA // 2)) for hora in horas[:size]]

    # Generamos sospechosos
    sospechosos = [np.random.randint(hora - error, hora + error) for hora, error in intervalos]
    np.random.shuffle(horas)

    return intervalos, sospechosos
```

Como se puede ver en el algoritmo, las transacciones del sospechoso siempre se generan dentro de un intervalo. Debido a esto ,en las mediciones de tiempo, el sospechoso siempre es rata. Los resultados de las mediciones de tiempo y el error de la aproximación por cuadrados mínimos a una curva de complejidad $O(n^2)$ fueron representados gráficamente con el uso de la biblioteca matplotlib de python.

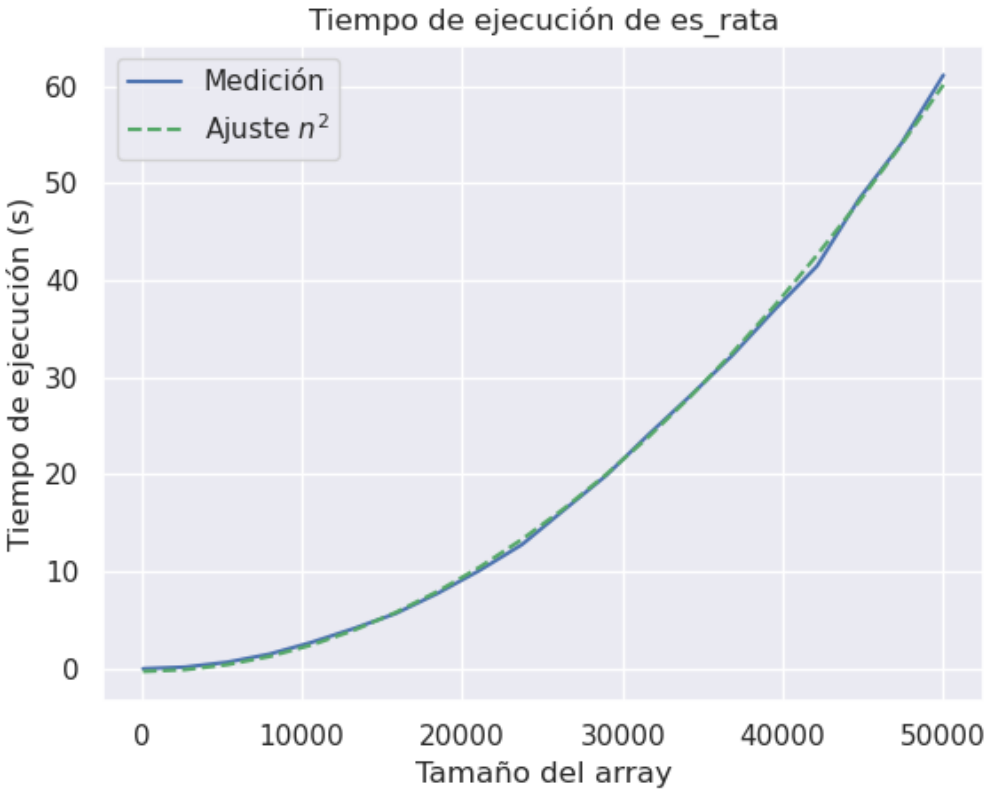

Figura 8: Medicion tiempo

Este gráfico sirve como respaldo empírico del análisis de complejidad que hicimos previamente ,ya que se puede observar como la medición es muy similar al ajuste n^2.

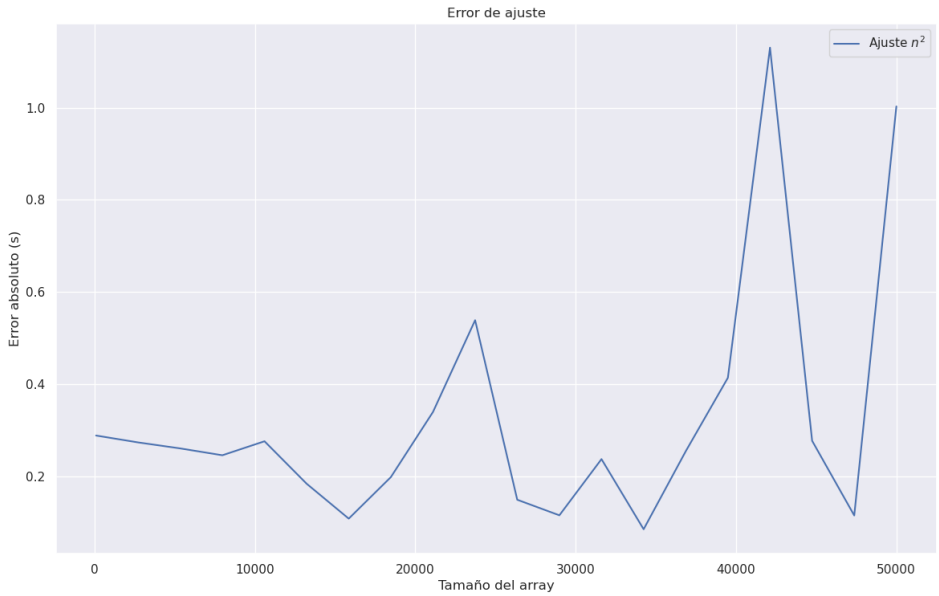

Figura 9: Error absoluto

El segundo gráfico nos muestra el error de la aproximación que hicimos respecto a los resultados obtenidos. Se puede observar que los valores obtenidos son bastante bajos lo que nos dice que el ajuste n^2 es bastante bueno respecto de la medición.

6. Ejemplos de ejecución

Para demostrar que el algoritmo siempre obtiene la solución óptima realizamos algunas pruebas

Como primer prueba utilizamos los datos provistos por la cátedra, de los cuales ya sabíamos cual era el resultado esperado

Para los 16 archivos de prueba, la respuesta de si el sospechoso es o no *la rata* dio el resultado esperado:

- 5-es.txt: Resultado obtenido: es la rata
- 5-no-es.txt: Resultado obtenido: no es la rata
- 10-es-bis.txt: Resultado obtenido: es la rata
- 10-es.txt: Resultado obtenido: es la rata
- 10-no-es-bis.txt: Resultado obtenido: no es la rata
- 10-no-es.txt: Resultado obtenido: no es la rata
- 50-es.txt: Resultado obtenido: es la rata
- 50-no-es.txt: Resultado obtenido: no es la rata
- 100-es.txt: Resultado obtenido: es la rata
- 100-no-es.txt: Resultado obtenido: no es la rata

- 500-es.txt: Resultado obtenido: es la rata
- 500-no-es.txt: Resultado obtenido: no es la rata
- 1000-es.txt: Resultado obtenido: es la rata
- 1000-no-es.txt: Resultado obtenido: es la rata
- 5000-es.txt: Resultado obtenido: es la rata
- 5000-no-es.txt: Resultado obtenido: no es la rata

Por otro lado también hicimos pruebas por nuestra cuenta con distintos sets de datos generados:

- 01-es.txt: Resultado obtenido: es la rata
- 02-no-es.txt: Resultado obtenido: no es la rata
- 03-no-es.txt: Resultado obtenido: no es la rata
- 04-no-es.txt: Resultado obtenido: no es la rata

Con estos tests la idea era probar el funcionamiento del algoritmo en casos borde para asegurarnos que la asignación de intervalos se siga haciendo de manera correcta y se siga obteniendo el resultado esperado en cada caso. Hay que destacar que no se podrían generar de forma random ya que tenemos que saber de antemano si hay una asignación posible o no y cómo podría llegar a ser esta

7. Conclusiones

A lo largo del desarrollo de este trabajo práctico pudimos ver qué tan aplicables son los algoritmos Greedy a problemas de la vida cotidiana en los que a partir de aplicar una regla sencilla e intuitiva de manera repetida se puede encontrar la mejor solución a estos. O no, muchas veces por las características del problema esto no es posible y solo queda conformarse con lo mejor que se pueda hacer, aun sabiendo que en determinados casos no se logrará obtener la solución óptima

Este no fue el caso para el problema presentado en este trabajo ya que el objetivo era poder hacer una asignación de cada una de las transacciones del sospechoso a cada uno de los intervalos aproximados y la forma en que decidimos realizar esa asignación garantiza que siempre que sea posible hacerla entonces el algoritmo la haga y devuelva la respuesta correcta. Solo no encuentra la asignación cuando ya de entrada no hay una asignación posible dadas las transacciones y los intervalos dados

De esta forma siempre se podrá determinar correctamente si el sospechoso es *la rata* o no, y el algoritmo Greedy propuesto siempre logrará encontrar la solución óptima