Trabajo Práctico 1: La mafia de los algortimos *greedy*

El presente trabajo busca evaluar el desarrollo y análisis de un algoritmo *greedy*. La fecha de entrega del mismo es el 10/04.

Introducción

Trabajamos para la mafia de los amigos Amarilla Pérez y el Gringo Hinz. En estos momentos hay un problema: alguien les está robando dinero. No saben bien cómo, no saben exactamente cuándo, y por supuesto que no saben quién. Evidentemente quien lo está haciendo es muy hábil (probablemente haya aprendido de sus mentores).

La única 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 $[t_i - e_i; t_i + e_i]$.

Por medio de métodos de los cuales es mejor no estar al tanto, un *interrogado* dio el nombre de alguien que podría ser *la rata*. El Gringo nos pidió revisar las transacciones realizadas por dicha persona... en efecto, eran n transacciones. Pero falta saber si, en efecto, condicen con los *timestamps* aproximados que habíamos obtenido previamente.

El Gringo nos dio la orden de implementar un algoritmo que determine si, en efecto, las transacciones coinciden. Amarilla Perez nos sugirió que nos apuremos, si es que no queremos ser nosotros los siguientes sospechosos...

Consigna

1. Hacer un análisis del problema, y proponer un algoritmo *greedy* que obtenga la solución al problema planteado: Dados los n valores de los *timestamps* aproximados t_i y sus correspondientes errores e_i, así como los *timestamps* de las n operaciones s_i del sospechoso (pueden asumir que estos últimos vienen ordenados), indicar si el sospechoso es en efecto *la rata* y, si lo es, mostrar cuál *timestamp* coincide con cuál *timestamp* aproximado y error. Es importante notar que los intervalos de los *timestamps* aproximados pueden solaparse parcial o totalmente.
2. Demostrar que el algoritmo planteado determina correctamente siempre si los *timestamps* del sospechoso corresponden a los intervalos sospechosos, o no. Es decir, si condicen, que encuentra la asignación, y si no condicen, que el algoritmo detecta esto, en todos los casos.
3. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de los diferentes valores a los tiempos del algoritmo planteado.
4. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares para que puedan usar de prueba.
5. Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Esto, por supuesto, implica que deben generar sus *sets* de datos. Agregar los casos de prueba necesarios para dicha corroboración. Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos. Recomendamos tomar más de una medición de la misma muestra y quedarse con el promedio para reducir el ruido en la medición.
6. Agregar cualquier conclusión que les parezca relevante.

Entrega

Completar el formulario de entrega con los integrantes y el link al repositorio donde se encuentre el código fuente, y donde debe encontrarse el informe en formato PDF.

La forma de ejecutar el programa debe ser:

```
./tp1 ruta/a/entrada.txt
```

O bien:

```
python3 tp1.py ruta/a/entrada.txt
```

El formato a ser aceptado debe ser como los ejemplos dado por el curso. El formato de salida debe ser parecido a los ejemplos de salidas brindados. No es necesario que sea exactamente igual, y pueden modificar algo en función de lo que analicen, pero debe mantenerse similar.

Importante: en caso que lo descripto no se cumpla, el trabajo será enviado automáticamente para reentrega sin otra corrección mediante, ya que lo primero que hacemos es verificar que el programa funcione.

El informe debe ser:

- Autocontenido: es decir, no debe ser necesario ponernos a buscar el código por diferentes lugares. El código que debe ser incluido es el código de los algoritmos a desarrollar (no es de interés un `main`, o el procesamiento de archivos, sino los algoritmos principales), y estos deben estar incluidos donde consideren correcto dado el desarrollo del informe, para su entendimiento.
- Tener todo el análisis correspondiente.
- Ser realizado en un formato profesional. Para esto, les brindamos un template en LaTeX para que puedan utilizar (también se encuentra en la *home* del sitio web del curso). No es necesario que lo sigan al pie de la letra, es simplemente un ejemplo que tiene varias cosas que pueden llegar a utilizar de LaTeX. Si ya conocen LaTeX no es necesario que lo utilicen, o mismo si utilizan algún otro formato (e.g. Markdown con Pandoc), pero recomendamos su revisión para que vean cosas que no deben de faltar. Por supuesto, pueden trabajar localmente como usar Overleaf o cualquier otra herramienta. El objetivo de darles el template no es la de limitar la creatividad, sino de asegurarnos que se cumplan lineamientos básicos sobre lo que se espera de una entrega de un informe en la facultad.
- En caso de ser necesarias reentregas, por favor agregar las modificaciónes en un Anexo al final del informe. No modificar lo hecho anteriormente. La excepción a esto sería si hay que rehacer una enorme mayoría de lo escrito.

La nota del trabajo práctico tendrá en cuenta tanto la presentación y calidad de lo presentado, como también el desarrollo del trabajo. No será lo mismo un trabajo realizado con lo mínimo indispensable, que uno bien presentado, analizado, y probado con diferentes volúmenes, *set* de datos, o estrategias de generación de *sets*, en el caso que corresponda.