



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

SEMESTRE 2023-1

PRACTICA 5

GRUPO: 3CV11

MATERIA: ANÁLISIS DE ALGORITMOS

ALUMNOS:

ISAAC SÁNCHEZ VERDIGUEL

ISANCHEZV1603@ALUMNO.IPN.MX

AXEL TREVIÑO PALACIOS

ATREVINOP1500@ALUMNO.IPN.MX

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

MAESTRO:

BENJAMIN LUNA BENOSO

21 Diciembre 2022

Índice general

1	Introducción	3
1.1	Resumen	3
1.2	Introducción	3
2	Desarrollo	4
2.1	Conceptos Básicos	4
2.2	Algoritmo Granjero Greedy	5
2.2.1	Algoritmo Granjero Greedy	5
3	Experimentación y Resultados	6
3.1	Algoritmo: Granjero Greedy	6
3.1.1	Análisis a Priori	6
3.1.2	Análisis a Posteriori	7
3.2	Pantallas de Ejecución del Algoritmo	8
4	Conclusiones	9
4.1	Conclusiones Generales	9
4.2	Isaac Sánchez - Conclusiones	10
4.3	Axel Trevino - Conclusiones	11
5	Anexo	12
5.1	Problemas y Ejercicios	12
5.1.1	Ejercicio 1	12
5.1.2	Ejercicio 2	13
5.1.3	Ejercicio 3	13
5.1.4	Ejercicio 4	13
5.1.5	Ejercicio 5	13
5.1.6	Ejercicio 6	14

Índice de figuras

3.1	Análisis a Priori: Granjero Greedy	6
3.2	Análisis a Posteriori: Granjero Greedy	7
3.3	Ejecución de Granjero Greedy	8
4.1	Isaac Sánchez	10
4.2	Axel Treviño	11
5.1	Codificación cadena Ciencias de la Tierra	13

1 | Introducción

1.1. Resumen

La práctica consta de implementar un algoritmo que resuelva un problema de un granjero que busca optimizar sus rutas de desplazamiento hacia el pueblo donde tiene que conseguir su fertilizante haciendo uso del **algoritmo de Greedy**. Dicha práctica será desarrollada en un ambiente de programación con **Python** y **Linux**.

Palabras Clave: Python, Algoritmo, Greedy.

1.2. Introducción

Los algoritmos son una parte fundamental de la ciencia de la computación, ya que estos al ser computables pueden dar solución o una idea más concreta acerca de la solución de un problema.

Un algoritmo no siempre dará una solución correcta, lo cual jamás será malo, porque esto nos ayudará a poder minimizar su radio de error. Una característica casi obligatoria para el buen funcionamiento de un algoritmo es su **rendimiento y eficacia**. El rendimiento adecuado se encuentra en la solución más rápida y menos costosa [Cormen, 2009].

Una solución a problemas que tengan que ver con encontrar una solución óptima, es hacer el planteamiento del Algoritmo de Greedy. Este algoritmo es voraz y trabaja para solucionar problemas de optimización. Su fuerte es que es un algoritmo sencillo de implementar, siendo rápido a la hora de brindar soluciones óptima [Wikipedia, 2022]. La práctica hace uso del algoritmo para solucionar el problema del granjero que necesita saber qué días tiene que abastecer su suministro de fertilizante para que no se agote.

2 | Desarrollo

2.1. Conceptos Básicos

La **complejidad temporal**, dentro del análisis de algoritmos, es el número de operaciones que ejecuta un algoritmo en cierto tiempo. Su denotación es $T(n)$ y puede ser analizada mediante dos tipos de análisis:

- Análisis de priori: entrega una función que muestra el tiempo de cálculo de un algoritmo.
- Análisis a posteriori: es la prueba en tiempo real del algoritmo, midiendo su costo mediante valores de entrada.

El análisis de complejidad temporal define que un algoritmo alcanza su máximo potencial cuando los valores de entrada son mayores al tiempo estimado de ejecución, siendo que es factible poder completar sus ejecuciones en menor tiempo posible.

Algoritmo Greedy Es un algoritmo que encuentra una solución globalmente óptima a un problema a base de hacer elecciones localmente óptimas. Es decir: el algoritmo siempre hace lo que “parece” mejor en cada momento, sin tener nunca que reconsiderar sus decisiones, y acaba llegando directamente a la mejor solución posible. [Aprende-Olimpiada-Informatica, 2022].

2.2. Algoritmo Granjero Greedy

2.2.1. Algoritmo Granjero Greedy

Pseudocódigo Algoritmo Granjero Greedy

Lo que busca el granjero es hacer el mínimo número de desplazamientos al pueblo y para ello, hace uso de un algoritmo Greedy. El algoritmo Greedy busca ir al pueblo el último día de apertura antes de que se acabe el fertilizante (contabilizado por r días que le dura el fertilizante), de modo que si fuera al siguiente día de apertura ya se le habría acabado el fertilizante. Considerando el periodo de interés $\langle d_1, d_2, \dots, d_n \rangle$, para encontrar la solución óptima se toma d_i como el día en el que nos encontramos y d_j como el día posterior tal que el día de apertura de la tienda sería $d_j - d_i \leq r$ y $j > i$. El algoritmo lo realiza de la siguiente manera:

Algorithm 1: Greedy Granjero

Result: *imagenRotada*

```
 $d = 0;$ 
 $res = [];$ 
for  $i \leftarrow 0$  to  $len(diasTienda)$  do
    if  $r + d < diasTienda[i]$  then
         $d = diasTienda[i-1];$ 
         $res.append(d);$ 
    else
        continue;
if  $len(res) > 0$  then
    if  $res[0] \neq 0$  then
         $res.insert(0, 0);$ 
    else
        continue;
else
     $res.insert(0, 0);$ 
 $res.append(diasTienda[-1]);$ 
return  $res;$ 
```

3 | Experimentación y Resultados

Aquí se presentaran los resultados del **Análisis a Priori y Posteriori** del algoritmo del granjero con Greedy

3.1. Algoritmo: Granjero Greedy

El algoritmo fue ejecutado en el lenguaje de programación **Python** en un entorno de **Linux**. A continuación se muestra el análisis de priori y posteriori.

3.1.1. Análisis a Priori

La figura 3.1 presenta el análisis a priori realizado sobre el pseudocódigo del algoritmo de Granjero de Greedy. Concluyendo que el algoritmo presenta $T(n) = \theta(n)$

Algorithm 1: Greedy Granjero

Result: *imagenRotada*

```
d = 0;
res = [];
for i ← 0 to len(diasTienda) do
    if r + d < diasTienda[i] then
        d = diasTienda[i-1];
        res.append(d);
    else
        continue;
if len(res) > 0 then
    if res[0] != 0 then
        res.insert(0, 0);
    else
        continue;
else
    res.insert(0, 0);
res.append(diasTienda[-1]);
return res;
```

$O(n)$

$O(1)$

Figura 3.1: Análisis a Priori: Granjero Greedy

3.1.2. Análisis a Posteriori

En el análisis posteriori se verifica que el análisis a priori demostró que la complejidad del peor caso es $T(n) = \theta(n)$. En la figura 3.2 se muestra la función que acota al peor caso (línea verde) junto con los puntos que demuestran la complejidad del algoritmo funcionando de manera aleatoria.

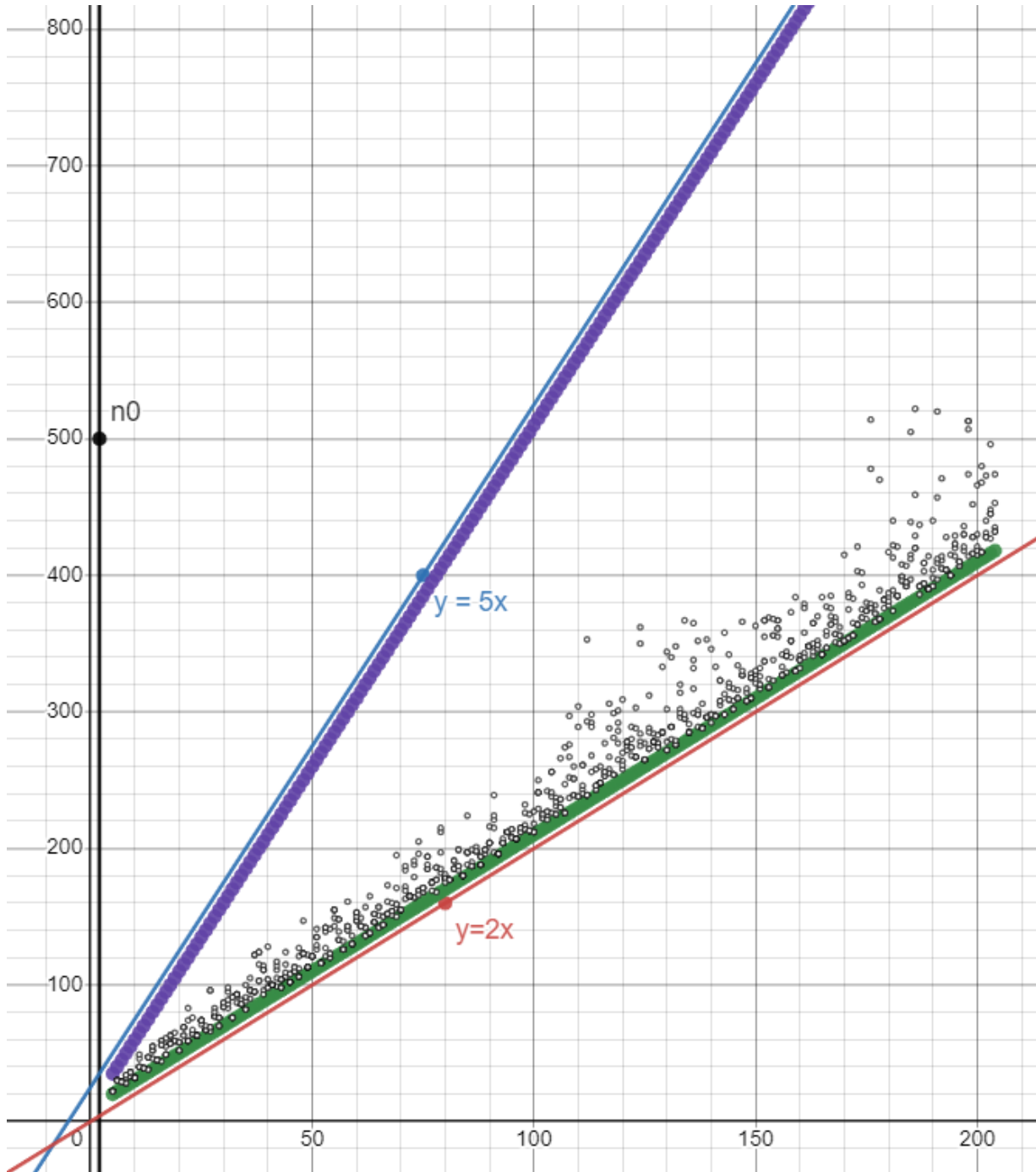
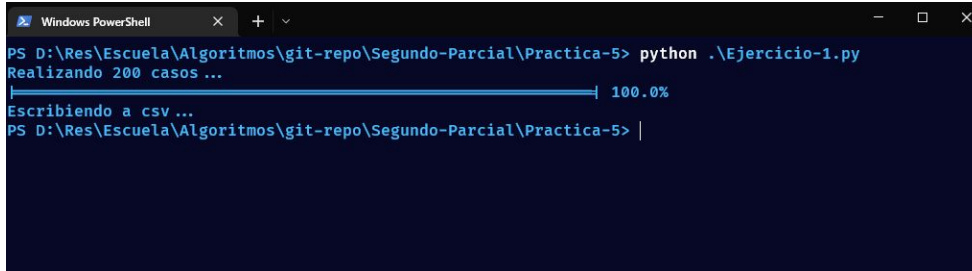


Figura 3.2: Análisis a Posteriori: Granjero Greedy

3.2. Pantallas de Ejecución del Algoritmo

Se muestra en la figura 3.3 la ejecución del algoritmo, demostrando la velocidad del algoritmo al ejecutar 200 casos.



```
Windows PowerShell
PS D:\Res\Escuela\Algoritmos\git-repo\Segundo-Parcial\Practica-5> python .\Ejercicio-1.py
Realizando 200 casos ...
|-----| 100.0%
Escribiendo a csv ...
PS D:\Res\Escuela\Algoritmos\git-repo\Segundo-Parcial\Practica-5> |
```

Figura 3.3: Ejecución de Granjero Greedy

4 | Conclusiones

4.1. Conclusiones Generales

Consideramos que la práctica ha sido la más sencilla de realizar, ya que el algoritmo de Greedy, si bien nos presentó una confusión en su explicación pero en su implementación conseguimos entender el algoritmo. El problema de la práctica nos dio una vista clara a las aplicaciones de diferentes algoritmos para mejorar la vida diaria. Aunque este algoritmo no presente una solución óptima adecuada, ayuda y eso es lo que importa.

4.2. Isaac Sánchez - Conclusiones

El algoritmo de Greedy fue de los temas que más tuve que poner atención, no conseguí entender a la primera y tuve que conseguir ayuda extra de ciertas bibliografías. Al finalizar la práctica y hacer el análisis a priori, descubrí que hay algoritmos que pueden dar soluciones a problemas tan simples como saber los días que se tienen que comprar fertilizante sin requerir una complejidad tan complicada.



Figura 4.1: Isaac Sánchez

4.3. Axel Trevino - Conclusiones

La práctica fué un buen ejemplo de el algoritmo greedy, porque es un uso en la vida real que le podrías dar. Lo único malo del greedy es que no te puede ayudar si no conoces bien los días en los que abre la tienda, o si puedes comprar cuanto fertilizante como quieras; como siempre, un algoritmo no resuelve vidas, sólo las hace más fáciles.



Figura 4.2: Axel Treviño

5 | Anexo

5.1. Problemas y Ejercicios

En esta sección se aborda la solución de los problemas solicitados durante las clases previas a la práctica.

5.1.1. Ejercicio 1

Descripción: Contestar las siguientes preguntas.

1. Documentar el orden de complejidad de la mochila fraccionaria. **Respuesta:** El orden de complejidad de la mochila fraccionaria es $\theta(n \log n)$
2. Mostrar mediante un contraejemplo que en el caso de elegir objetos enteros, el algoritmo voraz propuesta para el caso fraccionario puede no generar soluciones óptimas. **Respuesta:** No funcionaría por el simple hecho de que un objeto entero no puede generar una solución de un valor entero.
3. ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo valor? **Respuesta:** Se basaría en los valores más pequeños, al ser todos iguales algunos se queden fuera si la mochila queda sin espacio.
4. ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo peso? **Respuesta:** No afectaría ya que se mira por el valor y no por el peso.

5.1.2. Ejercicio 2

Descripción: Construir la codificación de Huffman para la cadena ciencias de la tierra. En la figura 5.1 se muestra el árbol construido por medio de una página web de la cadena Ciencias de la Tierra.

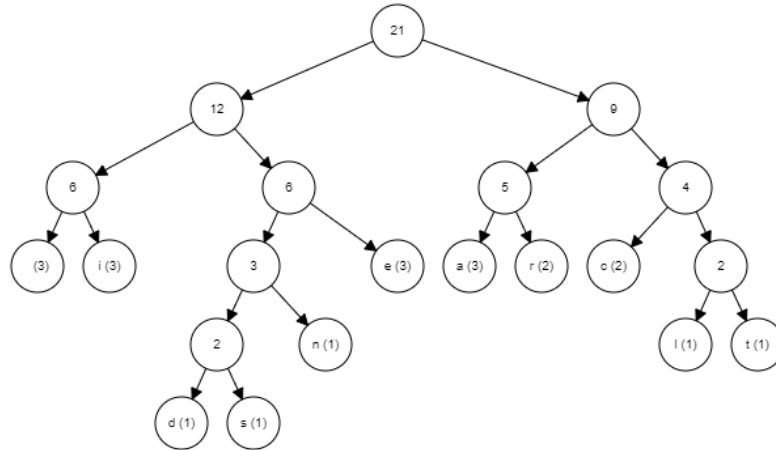


Figura 5.1: Codificación cadena Ciencias de la Tierra

5.1.3. Ejercicio 3

Descripción: Determinar el orden de complejidad el algoritmo de Huffman
La extracción de una cola de prioridad y al ser iterativo tiene una complejidad de $O = n \log n$

5.1.4. Ejercicio 4

Descripción: Documentar el orden de complejidad del algoritmo de Kruskal
Tiene una complejidad de $O = n \log n$ Siendo n el número de vértices y a el número de aristas del grafo. Éste orden de complejidad es el obtenido al realizar la ordenación de las aristas de menor a mayor peso [Complejidad Algoritmica, 2022a].

5.1.5. Ejercicio 5

Descripción: Investigar el algoritmo de Prim
El algoritmo de Prim, dado un grafo conexo, no dirigido y ponderado, encuentra un árbol de expansión mínima. Es decir, es capaz de encontrar un subconjunto de las aristas que formen un árbol que incluya todos los vértices del grafo inicial, donde el peso total de las aristas del árbol es el mínimo posible. Después de realizar el análisis del código, tal y como muestran los costes indicados en el apartado anterior, se puede decir que, para el algoritmo de Prim, el orden de complejidad computacional temporal es de $O(n^2)$. Siendo n el número de vértices del grafo [Complejidad Algoritmica, 2022b].

5.1.6. Ejercicio 6

Descripción: Documentar el orden de complejidad del algoritmo de Dijkstra.

El algoritmo de Dijkstra es un algoritmo eficiente de complejidad de $O(n^2)$ que sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo.

Bibliografía

- [Aprende-Olimpiada-Informatica, 2022] Aprende-Olimpiada-Informatica (2022). Algoritmos voraces. <https://aprende.olimpiada-informatica.org/algoritmia-voraz>. [Online; accessed 20-December-2022].
- [Complejidad Algoritmica, 2022a] Complejidad Algoritmica (2022a). Algoritmo de kruskal. <https://sites.google.com/site/complejidadalgoritmicaes/kruskal>. [Online; accessed 21-December-2022].
- [Complejidad Algoritmica, 2022b] Complejidad Algoritmica (2022b). Algoritmo de prim. <https://sites.google.com/site/complejidadalgoritmicaes/prim>. [Online; accessed 21-December-2022].
- [Cormen, 2009] Cormen, T. H. (2009). *Introduction to Algorithms*. The MIT Press, London.
- [Wikipedia, 2022] Wikipedia (2022). Algoritmo voraz. https://es.wikipedia.org/wiki/Algoritmo_voraz#:~:text=En%20ciencias%20de%20la%20computaci%C3%B3n,a%20una%20soluci%C3%B3n%20general%20%C3%B3ptima. [Online; accessed 20-December-2022].