Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos, Sem.: 2022-1, 3CV11, Práctica 4, 12 noviembre 2021

Práctica 5: Algoritmo Greedy

Vanegas García Andrés, Vaquera Aguilera Ethan Emiliano.

*avanegasg*1601@*alumno.ipn.mx*, *evaqueraa*@*alumno.ipn.mx*

Resumen: Para la practica en cuestión se realizo el análisis de un problema que incumbía el encontrar los posibles resultados de la búsqueda mas efectiva de los días, en las que un hombre le es o era mas eficiente ir a comprar cierto fertilizante a la tienda del pueblo donde los horarios de apertura son irregulares, es por ello que se deseaba sacar una solución por medio de un algoritmo greedy desarrollado en C.

Palabras Clave

C: C es un lenguaje de programación estructurado, usado por ya poco más de 52 años, esto brindándole un amplio repertorio de bibliotecas, además de un gran soporte de la comunidad que lo utiliza. C al ser un lenguaje de programación estructurado comparte similitudes con otros lenguajes desarrollados en la época además de que es un lenguaje de alto nivel, que de igual manera pude ser utilizado a bajo nivel para programar en ensamblador y rutinas de memoria.

Complejidad de un Algoritmo: Este es el punto en el que se mide la complejidad de resolución de un algoritmo por medio de una computadora, poniendo énfasis en el uso de la memoria de este sistema además de, el tiempo que este toma para desarrollarlo. Esta desde el inicio de la computación es una problemática que ha ido evolucionando a modo de que hoy en día se utiliza el análisis de algoritmos para so resolución y determinar la complejidad de un algoritmo en especifico.

Análisis a priori y posterior: Según lo comprendido en clase. El análisis a priori en computación es analizar un algoritmo, de manera formalmente teórica y matemáticamente, primero haciendo un análisis de sus posibles pseudocódigos y después en base a esto realizar una formulación del comportamiento de este en base a una función matemática.

Por otro lado el análisis a posterior, es la recolección de datos por medio de la experimentación, que en el caso de un algoritmo es la programación de este para así tomar una serie de pruebas en base a las veces ejecutadas y la cantidad de veces que este algoritmo tarda para terminar.

Iteración: Una iteración es la forma de hacer una serie de pasos repetidas veces hasta llegar al resultado esperado en un rango esperado sin sobrepasar este, las iteraciones pueden ir de cualquier manera en la vida real siendo como realizar un ejercicio matemático de una raíz, una potencia numérica o un logaritmo, y en computación serian las sentencias de while, for, do-while, etc.

Recursividad: La recursividad consiste mas que nada en la realización iterativa de un mismo lenguaje descriptivo, pero sin tener que mandar a llamar a una función iterativa. En palabras mas simples se trata de el desarrollo de una función que se llama a si misma, y después de llegar a un resultado esta regresa de vez en vez al inicio y finalmente otorgar el resultado final.

Divide y vencerás: Es una técnica para diseñar algoritmos que consiste en dividir el problema general en otros subproblemas de menor tamaño y del mismo tipo. Si un subproblema sigue siendo lo demasiado grande, esta técnica se emplea repetidas veces hasta que cada subproblema sea considerablemente pequeño para ser resuelto de una forma directa.

Algoritmo de Greedy: El algoritmo greedy es aquel algoritmo que se usa la para la búsqueda de posibles soluciones de una problemática planteada, siendo que esta problemática puede tener una cantidad n de soluciones, el algoritmo greedy no siempre encuentra la solución mas óptima para este, aveces puede que el algoritmo simplemente no encuentre la solución al problema.

1 Introducción

Las cuestiones que abarcamos en esta practica son algo sencillas de decir pero nada fácil de implementar, ya que la búsqueda de soluciones, el análisis de un problema, o solamente el planta miento del problema puede resultar en una tarea de cuestiones mas complicadas de lo que imaginamos, en la practica de que se redacta aquí, veremos el planteamiento de un problema que tiene cierto hombre para ir a la compra de fertilizante para su granja, siendo que la problemática planteada es que el hombre no puede quedarse sin fertilizante, y además de que el fertilizante le dura una N cantidad de días, por lo que lo mas óptimo para el problema es que compre en la fecha cercana al termino de los días en que se le termina el fertilizante, pero sin que este llegue

nunca a quedarse sin este insumo para su granja. Igualmente debemos de encontrar un análisis para este problema, el problema en cuestión puede ser analizado de mil maneras distintas, pero nosotros como programadores es que debemos de encontrar un análisis para el problema en cual, ser nos sea posible la implementación del programa. Por ultimo después del análisis vamos a la implementación del programa, solo si es que se encontró un análisis donde podamos implementar la solución, ya que hay veces en las que le problema no puede tener una solución, y de ser que tiene una solución nosotros trataremos de encontrar la solución las óptima en algunos casos con el algoritmo greedy, ya que como se describe en la parte del Algoritmo Greedy en la sección de palabras clave este a pesar de todo no siempre va a encontrar la solución mas óptima en la mayoría de los casos.

El algoritmo Greedy es muy efectivo para la búsqueda de soluciones, pero de esa manera como es efectivo no podemos imaginar un mundo perfecto donde todas las soluciones que encuentre sean las mas óptimas, ya que por si solo encontrar una es mas que mágico para poder hacer que la vida de cierto granjero sea mas sencilla para no tener que gastar dinero de mas o tiempo de viajes, además de que las aplicaciones son enormes si es que el problema permite que este pueda ser computable o no.

Situación mas óptima

Sea la solución de nuestro problema el conjunto $d_{p0y}, d_{p1y}, ..., d_{pmy}$ y suponemos que hay otra solución factible $d_{q0y}, d_{q1y}, ..., d_{qk}$ que realiza k vistas, menes que la solución anterior ($k < m$). Vamos a demostrar por inducción que para todo valor de j entre 1 y k, $d_{pj} \geq d_{qj}$.

Para j=1, podemos ver que obviamente $d_{p1} > d_{q1}$ ya que d_{p1} es le ultimo día de apertura que puede estar el granjero son fertilizante ($d_{p1} \leq r$). Entonces obviamente cualquier día de apertura siguiente a ese ya no tendría fertilizante ($d_{p1-1} > r$, combinado: $d_q > d_{p1-1} > r$ y q no tendrán solución factible).

Suponemos que $d_{pi-1} \geq d_{qi-1}$, entonces $d_{qi} - d_{pi-1} \leq d_{q1} - d_{qi-1}$ y como q es factible también se cumple que $d_{q1} - d_{q1-1} \leq r$ y por lo tanto $d_{qi} - d_{pi-1} \leq r$. Esto nos indica que d_{qi} esta en el rango de d_{p-1} lo que significa que d_{qi} es el día mas alejado en el que puede ir el granjero al pueblo por lo que el algoritmo greedy lo habría seleccionado. Por lo tanto no puede ser el día mas alejado dentro del rango. Entonces se demuestra que $d_{pj} \geq d_{qj}$ para j entre 1 y k.

Finalmente como sabemos que $d_{pk} \geq d_{qk}$, entonces $d_{n+1} - d_{pk} \leq d_{qi} - d_{q1-1}$ y como q es factible $d_{n+1} - d_{dk} \leq r$, entonces $d_{n+1} - d_{pk} \leq r$. Por lo que d_{pk}

esta dentro del periodo de interés y no tiene sentido que vayamos mas días.

2 Conceptos Básicos

Θ: La sigla griega Teta (Θ) es la denotación que describe de manera clara, el comportamiento similar de las funciones matemáticas del peor y mejor caso, esto siendo una referencia clara para hacer distinguir de Ω y O, que tiene definiciones diferentes.

Definición formal: $\Theta(g(n))= \{f(n) : existe\, n, c_1, c_2, > 0\, y\, n_0 > 0\, || \, 0 \leq c_1 g(n \leq f(n) \leq c_2 g(n)\, \forall\, n \geq n_0\}$

***O*:** Big O es la denotación usada para el conjunto de funciones que se acotan por encima de f(n), describiendo de manera clara el comportamiento da la función del peor caso, y denota los puntos experimentales de esta también.

Definición Formal: $O(g(n)) = \{f(n) : existen, c > 0\, \&\, n_0 > 0\, constantes\, || \, 0 \leq f(n) \leq cg(n)\, \forall\, n \geq n_0\}$

Ω: La denotación Omega (Ω) es la denotación usada para acotar por abajo a la función f(n), siendo que es usada para describir al conjunto de funciones que están por debajo de la función experimental f(n), y de igual manera sirve para describir los puntos experimentales de igual manera.

Definición Formal: $\Omega(g(n)) = \{f(n) : existen, c > 0\, \&\, n_0 > 0\, constantes\, || \, cg(n) \leq f(n)\, \forall\, n \geq n_0\}$

Recursividad: La recursividad es la serie de procesos definidos, en la cual se quiere la sucesión de varios factores y en vez de usar una función de la clase iterativa, se usan los mismos procesos ya definidos para crear esa sucesión de procesos, Imaginemos una muñeca de Matryoshkas (tipo de muñecas rusas), cada que abrimos una para sacar una nueva estas son mas pequeñas y además, todas y cada una de de ellas son iguales, usando ese ejemplo podemos definir un poco mejor la función recursiva, donde cada vez que mandamos a llamar a la función los valores de esta pueden ir disminuyendo o aumentando, dependiendo de lo que se quiera solucionar, hasta llegar a una condición de frontera, y de igual manera a pesar de que los valores cambien, lo que nunca se modifica son los procesos de la función recursiva que se mantiene de la misma manera.

Algorithm 1 Algoritmo Ejemplo Recursivo

```
function RECURSIVO(n)
    a ← n
    if a ≠ 0 then
        return Recursivo(a ← a − 1)
    else
        return a
    end if
end function
```

3 Experimentación y Resultados

Para el pseudocódigo tenemos una una función solución la cual implementa la técnica greedy, esta función tiene como entrada un arreglo y el tamaño del arreglo.

Algorithm 2 Algoritmo Ejemplo Recursivo

```
function SOLUCIÓN(Días, NumDias)
    S ← días en que se debe ir a comprar
    f ← día en el que se acaba fertilizante
    while Haya dias que abre la tienda do
        if dia < f then
            S ← Dia anterior que abre la tienda
            f ← nuevo día en que se acaba fertilizante
        end if
    end while
end function
```

Este pseudocódigo tiene un ciclo while el cual va a recorrer el arreglo de los días que abre la tienda, mientras haya días en lo que abre la tienda se compara si el día en que se acaba el fertilizante (Variable f) es menor al día en que abre la tienda, si esto ocurre se agregara al arreglo de soluciones (Arreglo S) el día anterior en que abre la tienda. Así el arreglo S tiene todos los dias en que el granjero debe ir a comprar fertilizante antes de que se le acabe.

3.1 Análisis a priori

Para el análisis a priori se eléboro ya el algoritmo el cual va a resolver nuestro problema. En este algoritmo la función recibe igual mente el array C (dias) y el tamaño de este. En el algoritmo se agregaron algunos contadores como

el tamaño del arreglo de soluciones (tamSol) y llamadas a la función imprimirArreglo que como su nombre lo indica imprime en consola el arreglo solución y los días en que abre la tienda.

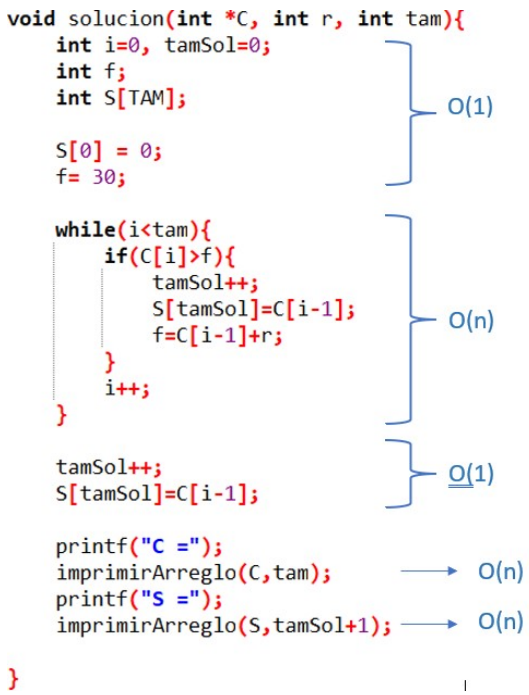


Figura 1. Análisis a priori

La **Figura 1** muestra el análisis del orden de complejidad del algoritmo, donde observamos que el while tiene orden $O(n)$ y las llamada a la función imprimirArreglo igualmente tiene orden lineal, por lo tanto el orden de complejidad de nuestro algoritmo es $O(n)$, en otras palabras tiene un comportamiento lineal.

3.2 Análisis a posteriori

Para generar la gráfica agregamos al código un contador que nos indicara el numero de pasos ejecutados por el algoritmo, con esto se genero una gráfica que comprara el numero de pasos ejecutados contra el tamaño de arreglo para los días.

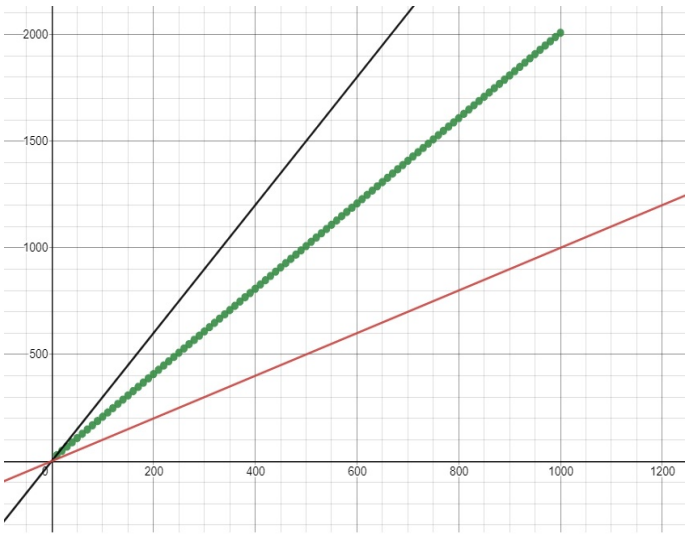


Figura 2. Análisis a posteriori

En la **Figura 2** en color verde se muestra la recta que forma los puntos que se obtuvieron y las otras dos rectas con las funciones propuestas con la cuales vamos a acotar nuestra recta generada por los datos de nuestro algoritmo. La recta $3x$ en color negro es la cota superior propuesta y la recta x en color naranja es nuestra cota inferior.

```
.:: Ejercicio con C de Tamaño = 12 ::.
{0, 29, 36, 50, 52, 66, 71, 85, 100, 117, 127, 129, 0, }
{0, 29, 52, 71, 100, 129, }
```

Figura 3. Salida programa.

```
.:: Ejercicio con C de Tamaño = 10 ::.
C ={0, 27, 31, 47, 57, 61, 62, 82, 93, 99, }
S ={0, 27, 57, 82, 99, }

.:: Ejercicio con C de Tamaño = 20 ::.
C ={0, 27, 39, 69, 88, 95, 110, 131, 142, 158, 181, 206, 207, 235, 264, 288, 308, 323, 337, 352, }
S ={0, 27, 39, 69, 95, 110, 131, 158, 181, 207, 235, 264, 288, 308, 337, 352, }

.:: Ejercicio con C de Tamaño = 30 ::.
C ={0, 27, 42, 60, 84, 112, 131, 138, 165, 176, 206, 234, 243, 251, 274, 287, 289, 302, 326, 337, 343, 360, 386, 394, 40
2, 424, 428, 436, 461, 466, }
S ={0, 27, 42, 60, 84, 112, 138, 165, 176, 206, 234, 251, 274, 302, 326, 343, 360, 386, 402, 428, 436, 466, }

.:: Ejercicio con C de Tamaño = 40 ::.
C ={0, 27, 31, 40, 50, 58, 88, 118, 135, 162, 167, 170, 178, 208, 234, 252, 263, 275, 289, 307, 308, 332, 342, 360, 362,
 373, 379, 404, 415, 432, 449, 458, 484, 498, 507, 536, 563, 582, 602, 606, }
S ={0, 27, 50, 58, 88, 118, 135, 162, 178, 208, 234, 263, 289, 308, 332, 362, 379, 404, 432, 458, 484, 507, 536, 563, 58
2, 606, }

.:: Ejercicio con C de Tamaño = 50 ::.
C ={0, 27, 54, 55, 59, 60, 65, 87, 96, 120, 127, 145, 167, 178, 205, 211, 228, 232, 236, 253, 272, 273, 298, 304, 320, 3
31, 344, 349, 370, 392, 393, 415, 416, 427, 446, 456, 475, 495, 516, 544, 545, 550, 569, 597, 604, 624, 630, 631, 645, 6
49, }
S ={0, 27, 55, 65, 87, 96, 120, 145, 167, 178, 205, 232, 253, 273, 298, 320, 349, 370, 393, 416, 446, 475, 495, 516, 545
, 569, 597, 624, 649, }
```

Figura 4. Salida programa con arreglos aleatorios.

Para finalizar, la **Figura 3** muestra la salida del programa con un ejemplo concreto y en la **Figura 4** se observa la salida del programa con 4 arreglos de diferente tamaño con datos aleatorios. En las dos imprime como datos de salida el arreglo de los días y la solución.

Características PC Venegas Garcia Andres
-Procesador Intel i5-10400f a 2.9 Ghz six core
-16 Gb de Ram
-Nvidia Geforce GTX 970

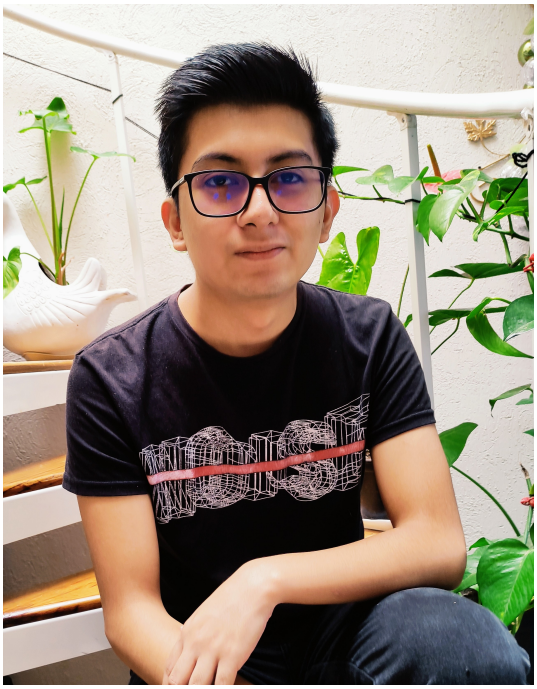
Características PC Vaquera Aguilera Ethan Emiliano
-Procesador ryzen 7-2700 a 3.2 Ghz octa core
-16 Gb de Ram
-AMD Radeon RX 5600XT

4 Conclusiones

Conclusión General: Por la practica realizada se observo de manera general que el planteamiento de un problema junto con el análisis proporcionado fue de gran ayuda, además de que se entiende de manera mas clara el uso del algoritmo greedy para la búsqueda de posibles soluciones que a pesar de que no siempre son buenas estas no dejan de ser soluciones óptimas dependiendo del caso que se busque solucionar.

Conclusión Individual Vanegas García Andrés: Una vez realizada la practica, observe que esta técnica Greedy

Figura 15. Vanegas García Andrés



Conclusión Individual Vaquera Aguilera Ethan Emiliano: Por mi parte de la practica pude observar que el análisis de los ejercicios es muy importante, eso ya se sabia con anterioridad pero el detenerse y verdaderamente observar como se comporta y debe de comportarse un programa es esencial, siendo que esto se esta viendo mas desde practicas recientes. Otro punto es el algoritmo greedy que al se comprendió de mejor manera el como se usa para la búsqueda de soluciones mas no de la solución absoluta de todo el problema

algunas veces esto puede ser lo mas óptimo pero otras no lo es.

Figura 16. Vaquera Aguilera Ethan Emiliano

5 Anexo

5.1 Tarea 1:

Mostrar mediante un contra ejemplo que en el caso de elegir objetos enteros, el algoritmo voraz propuesto para el caso fraccionario puede no generar soluciones óptimas.

Peso Mochila: 49
Elementos: 0, 1, 2, 3
Peso: 14, 21, 28, 35
Beneficio: 7, 35, 49, 70
B/P: 0.5, 1.6, 1.7, 2

Por el Algoritmo propuesto tenemos que:
Greedy = (3,0), Beneficio = 77

Solución Óptima:
Sol. = (2, 1), Beneficio = 84

Así conseguimos un contra ejemplo de que el sistema no siempre encuentra la solución mas óptima.

5.2 Tarea 2:

¿Cuál seria la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo valor?
Para el caso planteado en cuestión se tomaran los siguientes valores:

Peso Mochila: 5
Elementos: 0, 1, 2, 3
Peso: 1, 1, 1, 1
Beneficio: 1, 1, 1, 1
B/P: 1, 1, 1, 1

Como podemos ver el caso es sencillo a simple vista peor a la consideración del equipo podemos ver que el mejor caso serio el siguiente:
Tomaremos el primer objeto que seria de un peso 1 con un beneficio 1 y un B/P 1 por lo tanto el beneficio de todos los K de la función tendrán el mismo valor pero como el recorrido es descriptivo se seleccionarían los objetos de manera consecutiva y no de manera repetida como se pensó en un inicio.

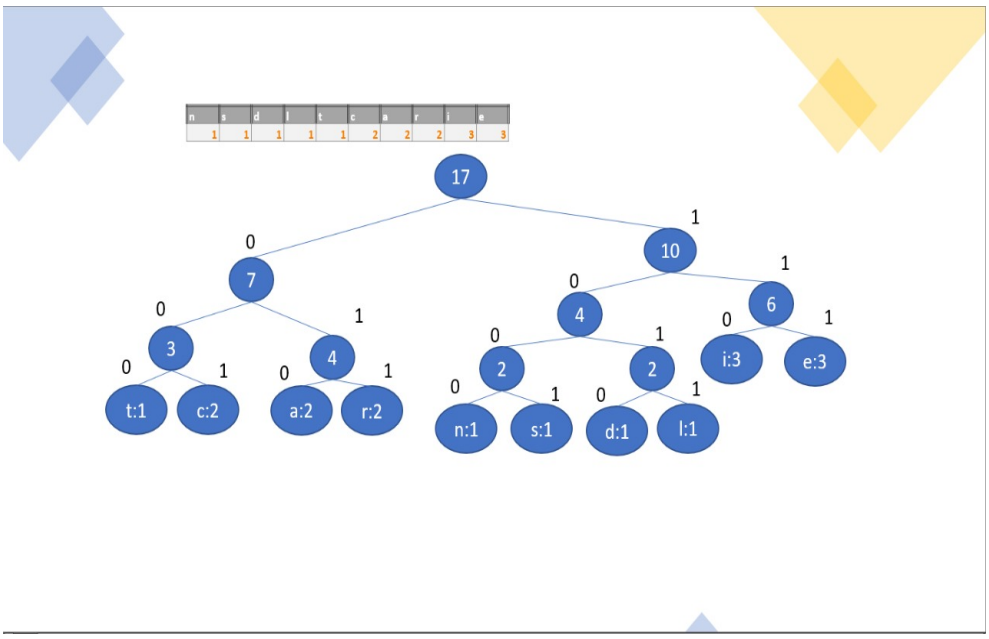
Greedy = (0, 1, 2, 3, 0) con un beneficio de 5 y esto hasta que le peso P sea completamente llenado.

5.3 Tarea 3:

Construir la codificación de Huffman para la cadena: **ciencias de la tierra**

c: 2, i: 3, e: 3, n: 1, a: 2, s: 1, d: 1, l: 1, t: 1, r: 2.

Figura 4. Diagrama Huffman



t: 000
c: 001
a: 010
r: 011
n: 1000
s: 1001
d: 1010
l: 1011
i: 110
e: 111

ciencias de la tierra: 00111011110000011100101001 1010111 1011010 000110011011010

5.4 Tarea 4:

Documentar el orden de complejidad del algoritmo de Huffman

Para el análisis de complejidad del algoritmo de Huffman se toma en cuenta que los algoritmos de búsqueda son algoritmos de orden lineal junto con los algoritmos de inserción de datos dentro del árbol binario. Así tenemos lo siguiente.

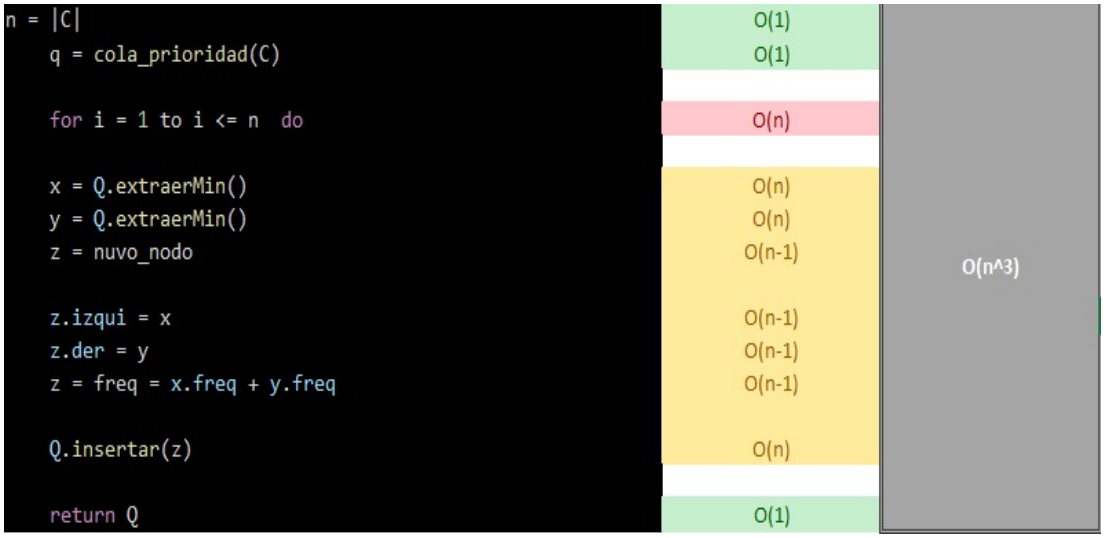


Figura 5. Análisis Algoritmo de Huffman

6 Bibliografía

programmerclick.com. (2021). Complejidad temporal y complejidad espacial. 2021, de programmerclick.com Sitio web: https://programmerclick.com/article/89671544092/
Agrawal, Manindra; Kayal, Neeraj; Saxena, Nitin: "PRIMES is in P". Annals of Mathematics 160 (2004), no. 2, pp. 781–793.
J.M.+Gimeno+y+J.L.+González. (2021). Recursividad. 2021, de web Sitio web: http://ocw.udl.cat/enginyeria-i-arquitectura/programacio-2/continguts-1/2-recursividad.pdf
-. (2021). Algoritmo Greedy. 2021, de Universidad de Granada Sitio web: https://elvex.ugr.es/decsai/algorithms/slides/4