Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos

Semestre: 2022-1

Grupo: 3CV11

Práctica 5

Fecha: 26 de noviembre de 2021

Práctica 5: Algoritmos Greedy

Brayan Ramirez Benitez.
*bramirezb*1901@*alumno.ipn.mx*

Resumen: En esta práctica realizamos el análisis a priori y posteriori para un algoritmo que ofrece una solución al problema planteado, grosso modo, el problema aborda lo siguiente, un granjero busca hacer el mínimo número de desplazamientos a un pueblo y para proponer una solución hacemos uso de estrategias Greedy. Para esto, el algoritmo se implementará mediante el uso del lenguaje de programación en C, posteriormente desarrollamos el análisis a priori para el algoritmo propuesto mediante secciones de código, obteniendo como resultado, una complejidad temporal lineal, además llevamos a cabo el análisis a posteriori haciendo uso de una Calculadora Grafica que ofrece el sitio Web Desmos, con el objetivo de corroborar los resultados obtenidos.

Palabras Clave: Greedy, Optimalidad, Mínimo, Desplazamientos.

1 Introducción

Cuando diseñamos un algoritmo no existe una solución milagrosa que funcione para todos los problemas de computación, diferentes problemas requieren diferentes tipos de soluciones, entonces como programadores utilizamos algunas técnicas en función del tipo de problema, algunas de las técnicas más conocidas son Divide y Vencerás, Algoritmos Greedy o Programación Dinámica, a pesar de ello, en esta práctica nos enfocaremos en la técnica de Algoritmos Greedy.

Un algoritmo Greedy, siempre busca tomar la decisión que parece ser la mejor en ese momento, esto significa que hace una elección óptima a nivel local con el objetivo de que esta elección nos conduzca a una solución óptima a nivel general. No obstante, en una gran cantidad de problemas, una estrategia Greedy no produce una solución óptima. Por ejemplo, un algoritmo Greedy que busca encontrar la ruta con la mayor suma, lo hace seleccionando el mayor número disponible en cada paso, sin embargo, el algoritmo no consigue encontrar la suma más grande puesto que toma decisiones con base en la información que tiene en cada paso, sin tener en cuenta el problema general.

Para algunos problemas resulta bastante sencillo idear un algoritmo Greedy o incluso varios algoritmos, además determinar la complejidad de los algoritmos Greedy generalmente será mucho más sencillo que para otras técnicas como Divide y Vencerás puesto que para esta técnica en cada nivel de recursividad el tamaño se reduce y aumenta el número de subproblemas. La parte complicada en esta técnica es encontrar y demostrar que una solución es óptima, esto implica mucha creatividad.

El problema planteado para esta práctica implica obtener el menor número de desplazamientos que debe hacer un granjero en donde se conocen los días en los cuales la tienda se encuentra abierta, si analizamos detenidamente el

planteamiento del problema podemos determinar que es un problema de optimización, por lo tanto, esta práctica tiene como objetivo analizar la complejidad temporal del algoritmo propuesto haciendo uso de estrategias Greedy.

2 Conceptos Básicos

En esta sección mostraremos todos los conceptos necesarios para el desarrollo de la práctica.

Algoritmo Greedy: Es un algoritmo simple e intuitivo que se utiliza en problemas de optimización, el cual toma la decisión óptima en cada paso, puesto que intenta encontrar la forma óptima general de resolver todo el problema. Los algoritmos Greedy tienen bastante éxito resolviendo algunos problemas, como la codificación de Huffman, que se utiliza para comprimir datos, o el algoritmo de Dijkstra, que se utiliza para encontrar la ruta más corta a través de un gráfico, este tipo de algoritmos se utilizan en problemas para los cuales en cada paso existe una opción que es óptima en ese momento, y después del último paso, el algoritmo produce la solución óptima del problema completo.

Dos condiciones definen el paradigma Greedy.

1. Cada solución paso a paso debe estructurar un problema hacia su solución mejor aceptada.
2. Es suficiente si la estructuración del problema puede detenerse en un número finito de pasos.

Complejidad temporal o eficiencia en tiempo: Es el número de operaciones para resolver un problema con una entrada de tamaño **n**. Se denota por $\mathbf{T(n)}$.

Definición: Dada una función $\mathbf{g(n)}$. $O(\mathbf{g(n)})$ denota el conjunto de funciones definidas como:
$O(\mathbf{g(n)}) = \{\ \mathbf{f(n)}$: $\exists$n $C > 0$ y $n_0 > 0$ constantes Tal que $0 \leq \mathbf{f(n)} \leq C\ \mathbf{g(n)}\ \forall$ n $\geq n_0\ \}$

Análisis a priori: Consiste en determinar una expresión que nos indique el comportamiento para un algoritmo particular en función de los parámetros que contiene, es decir, determina una función que limita el tiempo de cálculo para un algoritmo en específico, por lo tanto, es un factor fundamental para el diseño de algoritmos.

Análisis a posteriori: Consiste en obtener estadśticas de tiempo utilizadas por el algoritmo mientras se ejecuta.

Planteamiento del problema: Un granjero necesita un determinado fertilizante que se vende en una tienda del pueblo. La tienda tiene un horario irregular, pero el granjero sabe que días abre la tienda. La cantidad máxima de fertilizante que puede usar el granjero le dura r días. 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, de modo que si fuera al siguiente día de apertura ya se le habría acabado el fertilizante. Consideremos el conjunto de días ordenados que abre la tienda dentro del periodo de interés $< d_1, d_2, ..., d_n >$ y el número de días r en el que el granjero tiene fertilizante. Para llevar a cabo este algoritmo tomamos como d_i al día de visita en el que nos encontramos, la siguiente visita sería el día d_j^* donde j^* es el máximo valor de j tal que $d_j - d_i \leq r$ y j ¿ i. Así conseguiríamos el día de apertura de la tienda tal que $d_j - d_i \leq r$ y $d_j + 1 > r$.

Optimalidad de la situaciíon

Sea la solución de nuestro problema del conjunto $d_{p0}, d_{p1}, ..., d_{pm}$ y suponemos que hay otra solución factible $d_{q0}, d_{q1}, ..., d_{qk}$ que realiza **k** visitas, menos 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 $\mathbf{j = 1}$, podemos ver que obviamente $d_{p1} > d_{q1}$ ya que d_{p1} es el último día de apertura que puede estar el granjero sin fertilizante ($d_{p1} \leq r$). Entonces obviamente cualquier día de apertura siguiente a ese ya no tendría fertilizante ($d_q > d_{p1+1} > r$ y **q** no seía solución factible)

- Suponemos que $d_{pi-1} \geq d_{qi-1}$, entonces $d_{qi} - d_{pi-1} \leq d_{qi} - d_{qi-1}$ y como **q** es factible tambieén se cumple que $d_{qi} - d_{qi-1} \leq r$ y por lo tanto $d_{qi} - d_{pi-1} \leq r$. Esto nos indica que d_{qi} está en el rango de d_{qi} es el día más 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 más alejado dentro del rango. Entonces se demuestre que $d_{pj} \geq d_{qj}$ para **j** entre 1 y k.

- Finalmente como sabemos que $d_{pk} \geq d_{qk}$, entonces de $d_{n+1} - d_{pk} \geq d_{n+1} - d_{qk}$ y como **q** es factible $d_{n+1} - d_{qk} \leq r$, entonces $d_{n+1} - d_{pk} \leq r$. Por lo que d_{pk} está dentro del periodo de interés y no tiene sentido que vayamos más días.

Algoritmo propuesto: Este algoritmo consiste en encontrar el mínimo número de desplazamientos al pueblo que debe hacer un granjero para comprar

fertilizante si la cantidad máxima que puede usar le dura r días y para ello, como ya lo mencionamos mediante un algoritmo Greedy. La función recibe dos arreglos, el arreglo S contiene el conjunto de días solución y el arreglo C contiene el conjunto de días de apertura, la variable r representa los días que dura el fertilizante y n representa el tamaño del arreglo C, la función devuelve el tamaño del conjunto solución S. Comenzamos declarando tres variables, la variable f guardara el valor del día en que se acabara el fertilizante, la variable i es el índice para el conjunto C y la variable j es el índice para el conjunto S. Ahora asignamos el primer día al conjunto solución puesto que es la primera vez que el granjero va a comprar fertilizante, luego asignamos a la variable f el día en que se acabara el fertilizante tomando el valor del conjunto solución más los r días que le dura el fertilizante, después de esto iniciaremos un ciclo for para recorrer el conjunto C, recordemos que el arreglo tiene un tamaño n, entonces n será el limite para el for, dentro del ciclo preguntamos si el i-ésimo día del conjunto C es mayor a f que es el valor donde se acabará el fertilizante, si esto se cumple agregaremos el día anterior al i-ésimo al conjunto solución, luego modificamos el valor de f asignándole el valor del nuevo día en donde se acabara el fertilizante e incrementamos j puesto que agregamos un elemento al conjunto solución, en caso contrario, si la condición no se cumple esto significa que podría haber otro día en el que todavía no debemos comprar fertilizante, entonces no hacemos nada, simplemente incrementamos i, por último, agregamos al conjunto solución el ultimo día del conjunto C y regresamos el valor de j que si recordemos es el tamaño del conjunto C.

Algorithm 1 *PFertilizante*

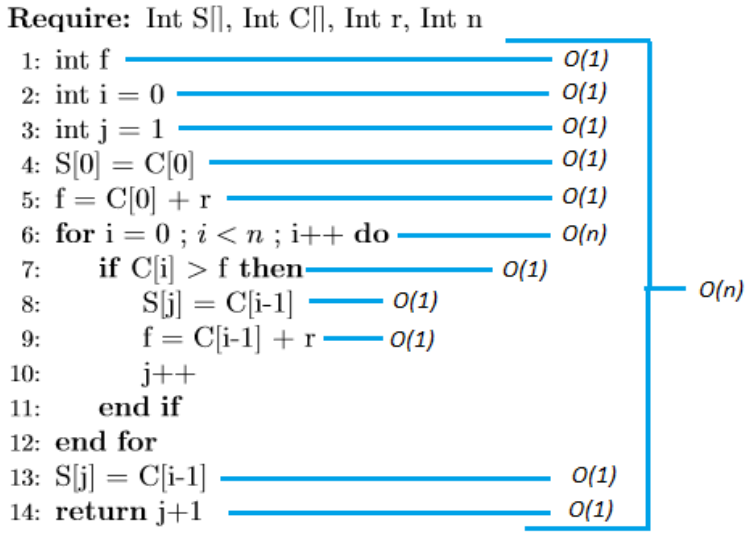
Require: Int S[], Int C[], Int r, Int n
Ensure: Menor número de desplazamientos

```
1: int f
2: int i = 0
3: int j = 1
4: S[0] = C[0]
5: f = C[0] + r
6: for i = 0 ; i < n ; i++ do
7:     if C[i] > f then
8:         S[j] = C[i-1]
9:         f = C[i-1] + r
10:        j++
11:    end if
12: end for
13: S[j] = C[i-1]
14: return j+1
```

3 Experimentación y Resultados

Análisis a priori para el algoritmo propuesto: Para el algoritmo **PFertilizante** llevaremos a cabo el análisis mediante segmentos de código. Si observamos el pseudo-código podemos determinar que el peor caso ocurre cuando tiene que guardar todos los días del conjunto C en el conjunto S, es decir, el granjero tiene que acudir al pueblo cada vez que la tienda está abierta.

Se tiene que



De lo anterior tenemos que

$\therefore$ **PFertilizante** $\in O(n)$

Análisis a posteriori para el algoritmo propuesto:

Comenzamos implementando el algoritmo como una función llamada **"PFertilizante"** con base en el pseudo-código del **Algorithm 1 PFertilizante**, mediante el Lenguaje de Programación en C.

```c
int PFertilizante(int *S, int *C, int r, int n){
    int f, i = 0, j = 1;//iniciamos dos contadores para los conjuntos

    S[0] = C[0];//El primer dia siempre se agrega al conjunto solucion
    f = C[0] + r;//Asignamos el valor de f

    for(i = 0; i < n; i++){
        if(C[i] > f){//preguntamos si el dia es mayor a f
            S[j] = C[i-1];//Lo agregamos al conjunto solucion
            f = C[i-1] + r;//Volvemos a asignar el valor de f
            j++;//incrementamos el contador para el conjunto solucion
        }
    }
    S[j] = C[i-1];//agregamos al conjunto solucion el ultimo
    return j+1;//regresamos el tamaño del conjunto solucion
}
```

Figura 1. Código para el **algoritmo propuesto** en lenguaje C.

Además, utilizaremos la función **Principal** la cual nos servirá para ejecutar el algoritmo y obtener los datos para el análisis, donde enviaremos los resultados a un archivo.

Para realizar el análisis la función **PFertilizante (Figura 1)** debe recibir un parámetro por referencia **(ct)** que obtendrá el número ejecuciones de cada línea.

```
int PFertilizante(int *S, int *C, int r, int n, int *ct){
    int f, i = 0, j = 1;(*ct)++;(*ct)++;(*ct)++;

    S[0] = C[0];(*ct)++;
    f = C[0] + r;(*ct)++;
    (*ct)++;
    for(i = 0; i < n; i++){
        (*ct)++;
        if(C[i] > f){
            (*ct)++;
            S[j] = C[i-1];(*ct)++;
            f = C[i-1] + r;(*ct)++;
            j++;(*ct)++;
        }
        (*ct)++;
    }
    (*ct)++;
    S[j] = C[i-1];(*ct)++;
    return j+1; (*ct)++;
}
```

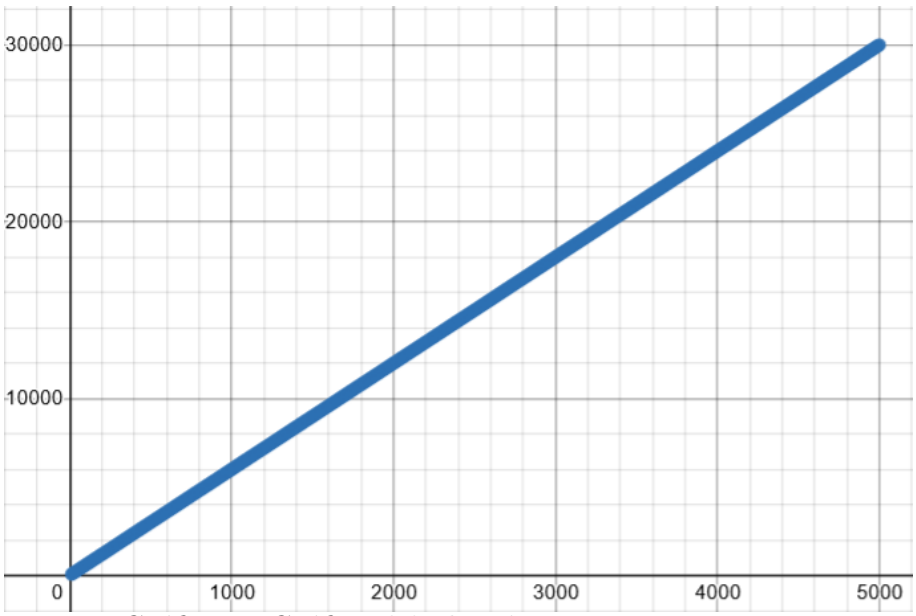
Figura 2. Código para el **algoritmo propuesto** con el parámetro **ct**.

Del análisis a priori sabemos qel peor caso ocurre cuando $d_i > f$ para todos los valores de i. Para obtener los puntos de muestra alimentamos el algoritmo con valores de $d_i = d_{i-1} + r + 1$ para el conjunto C, por ejemplo con C = 0, 31, 62, 93, 124, etc. El resultado de esta ejecución arroja los siguientes valores.

n	Ct
10	64
20	124
30	184
40	244
50	304
60	364
70	424
4970	29 824
4980	29 884
4990	29 944
5000	30 004

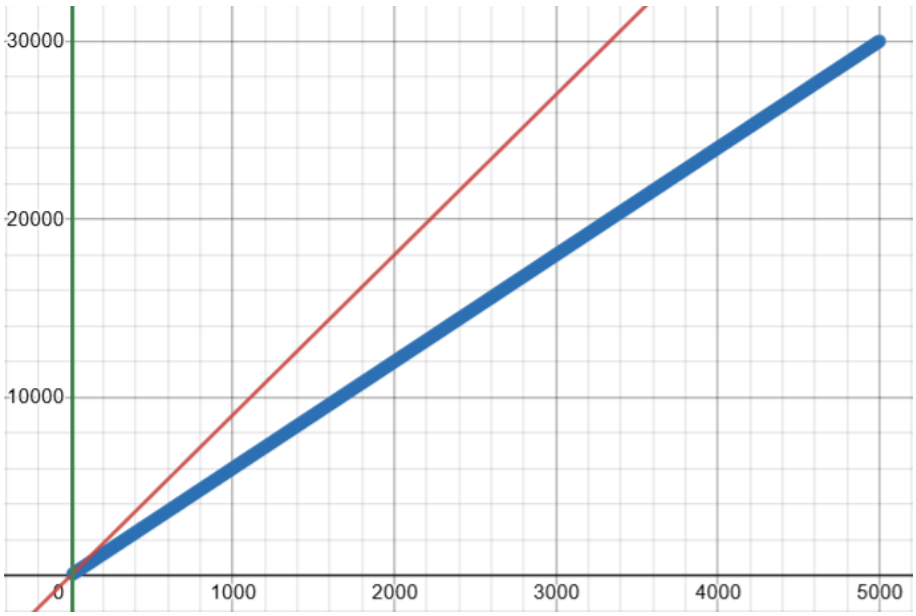
Tabla 1. Valores de la ejecución para el **algoritmo propuesto**.

Copiamos los valores que resultaron de la ejecución del programa y los graficamos mediante la calculadora Gráfica Desmos, observamos que es lineal $O(n)$.



Gráfica 1. Gráfica del **algoritmo propuesto**.

Ahora proponemos la función $\mathbf{g(n)} = 9(n) \in O(n)$ con $n_o = 10$ para la cota superior.



Gráfica 2. Gráfica del **algoritmo propuesto** con la función propuesta.

Ejemplos del funcionamiento del algoritmo propuesto:

```
r = 30
Conjunto C = 0 0 22 42 81 93 93 95 98 104 106 111
Conjunto S = 0 22 42 81 111
```

```
r = 25
Conjunto C = 0 11 15 17 31 40 61 62 73 104 107 110
Conjunto S = 0 17 40 62 73 104 110
```

```
r = 15
Conjunto C = 0 6 6 9 18 25 53 72 79 79 95 107
Conjunto S = 0 9 18 25 53 72 79 95 107
```

Características del equipo de cómputo:

- Procesador: Ryzen 5 1400
- Tarjeta gráfica: GTX 1050 ti de la marca PNY
- Tarjeta Madre: Asus A320M-K
- Memoria RAM: 8 Gb
- Disco duro: Disco Duro Interno 1tb Seagate

4 Conclusiones

Conclusiones generales: Sin duda alguna Greedy es una estrategia muy sencilla para resolver problemas, sin embargo, no siempre podemos obtener un buen resultado, puesto que principalmente resuelven problemas de optimización, en donde buscan obtener una solución óptima paso a paso, eligiendo en cada uno de estos pasos la solución que parece más apropiada en ese momento. Es importante mencionar que esta clase de algoritmos no revisan una solución, puesto que en teoría deberían haber elegido la mejor opción en cada paso, esta es la razón por la que no siempre dan resultado, ya que existen problemas para los cuales es conveniente realizar un paso aparentemente menos apropiado en ese momento pero que al final resulta en la mejor solución, no obstante, el algoritmo propuesto para dar solución al problema planteado nos ofrece una complejidad temporal bastante aceptable, obtuvimos los resultados esperados, es decir, para todos los experimentos obtuvimos una solución. Por ultimo, para mejorar el algoritmo propuesto es recomendable agregar una condición para evitar que los elementos del conjunto solución no se dupliquen, con el objetivo de mejorar su funcionamiento.

Conclusiones individuales (Brayan Ramirez Benitez): De manera general en esta práctica no ocurrieron problemas, no fue complicado implementar el algoritmo, unicamente me resultó un poco confuso encontrar cuando ocurre el peor caso. Por otro lado, me paració que fue más complicado dar solución a los problemas incluidos en el anexo puesto que requieren una muy buena comprensión del planteamiento para cada uno de estos problemas.

Brayan Ramirez Benitez.

5 Anexo

Problema 1: Mostrar mediante un contraejemplo que en el caso de elegir objetos enteros, el algoritmo voraz propuesto para el caso fraccionario puede no generar soluciones óptimas.

Solución: Tenemos los siguientes objetos con su beneficio, donde el peso máximo es de 8 kg.

Peso	4	3	5	2
Beneficio	10	40	30	20

si elegimos objetos enteros para el algoritmo voraz propuesto tenemos que

Peso	4	3	5	2
Beneficio	10	40	30	20
ben/Peso	0	1	0	1

tienen un beneficio: $40 + 20 = 60$ y peso: $3 + 2 = 5$ kg

Pero esta no es una solución óptima, ya que si elegimos

Peso	4	3	5	2
Beneficio	10	40	30	20
ben/Peso	0	1	1	0

lo que resulta en un beneficio: $40 + 30 = 70$ y peso: $3 + 5 = 8$ kg, la cual es una solución óptima.

por lo tanto, el algoritmo voraz propuesto para el caso fraccionario no siempre va a generar una solución óptima para objetos enteros.

Problema 2: ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo valor?

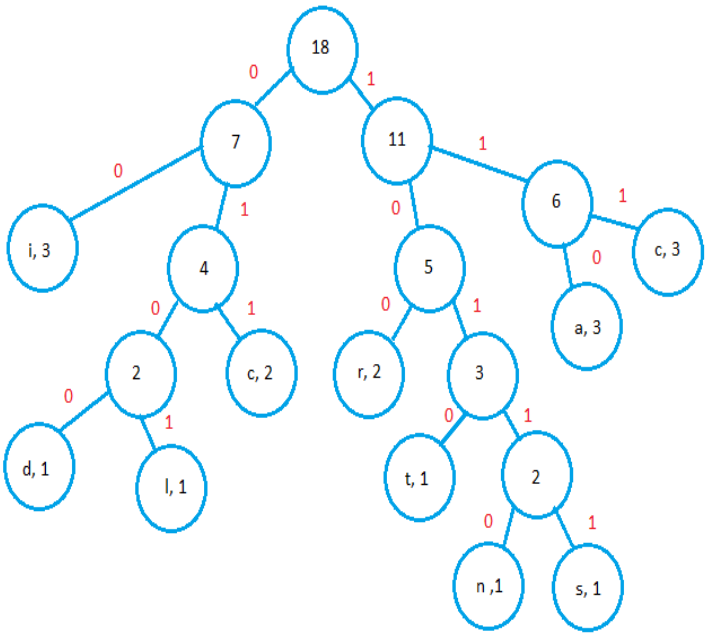
Solución: Si seleccionamos primero los objetos que tienen el menor peso , ya que de esta manera podremos almacenar la mayor cantidad de objetos.

Problema 3: ¿Cuál sería la mejor función de selección voraz en el caso en el que todos los objetos tuvieran el mismo peso?

Solución: Para este caso tendriamos que seleccionar primero los objetos más valiosos.

Problema 4: Construir la codificación de Huffman para la cadena: ciencias de la tierra.

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



i	00
d	0100
l	0101
c	011
r	100
t	1010
n	10110
s	10111
a	110
e	111

por lo tanto, 011001111011001100110101110 0100111 0101110 1010001111001001110

Problema 5: Documentar el orden de complejidad del algoritmo de Huffman

```
int main() {
    int n;
    while(~scanf("%d", &n)) {
        priority_queue<ll, vector<ll>, greater<ll> >q;
        ll res = 0;
        for (int i = 1; i <= n; ++i) {
            ll x;
            scanf("%lld", &x);
            q.push(x);
        }
        while (1) {
            ll a = q.top();
            q.pop();
            if (q.empty())break;
            ll b = q.top();
            q.pop();
            res += a + b;
            q.push(a + b);
        }
        printf("%lld\n", res);
    }
}
```

Figura. Código para la **codificación de Huffman**.

Para estudiar la complejidad del algoritmo que crea el árbol de Huffman, la extracción en una cola de prioridad tiene una complejidad O(n log n).

Por lo tanto, la complejidad de todo el algoritmo que es iterativo, será O(n log n).

6 Bibliografía

References

[1] Aguilar, I. (2019) *Introducción al análisis de algoritmos.* PDF. Recuperado de http://ri.uaemex.mx/bitstream /handle/20.500.11799/105198/Libro%20Complejidad.pdf? sequence=1&isAllowed=y

[2] Cormen, E. A. (2009) *Introduction to Algorithms.* 3Rd Ed. (3.a ed., Vol. 3). Phi.

[3] Basics of Greedy Algorithms Tutorials & Notes Algorithms. (2016, 27 abril) HackerEarth. Recuperado 20 de noviembre de 2021, de https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/

[4] Brilliant Math & Science Wiki. (s. f.). Recuperado 20 de noviembre de 2021, de https://brilliant.org/wiki/greedy-algorithm/

[5] Walker, A. (2021, 7 octubre) Greedy Algorithm with Examples: Greedy Method & Approach. Guru99. Recuperado 20 de noviembre de 2021, de https://www.guru99.com/greedy-algorithm.html