Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos

Semestre: 2022-1

Grupo: 3CV11

Práctica 6

Fecha: 3 de diciembre de 2021

Práctica 6: Programación Dinámica.

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

Resumen: En esta práctica realizamos el análisis a priori para un algoritmo que ofrece una solución al problema planteado, grosso modo, el problema consiste en lo siguiente, tenemos dos archivos de texto que contienen código fuente, entonces debemos determinar el porcentaje de parecido, mediante el uso de programación dinámica. 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 segmentos de código, obteniendo como resultado, una complejidad temporal **m x n**.

Palabras Clave: Programación, Dinámica, Comparar archivos, Código fuente.

1 Introducción

De manera general, la programación dinámica consiste principalmente en tomar un algoritmo recursivo y determinar los subproblemas superpuestos, es decir, llamadas repetidas. Posteriormente, almacenar estos resultados para las futuras llamadas recursivas. También, puede estudiar el patrón de las llamadas recursivas con el objetivo de implementar algo iterativo. Entonces, la programación dinámica principalmente es una optimización para la recursividad simple.

Normalmente cuando tenemos una solución recursiva que tiene llamadas repetidas para las mismas entradas, es posible optimizarla utilizando esta técnica, lo cual reduce la complejidad temporal de exponencial a polinomio. Para poder hacer uso de programación dinámica, debemos poder dividir un determinado problema en subproblemas más simples, luego, debemos verificar si el problema cumple algunas propiedades como tener subproblemas superpuestos y una subestructura óptima, entonces, si un problema se puede dividir en subproblemas más sencillos y cumple ambas propiedades, podemos aplicar programación dinámica para resolver este problema. Existen dos maneras diferentes para almacenar los valores obtenidos para poder reutilizarlos. Estas son tabulación o enfoque de abajo hacia arriba y memorización o el enfoque de arriba hacia abajo, más adelante describiremos mejor estas maneras de almacenar lo valores.

Por otro lado, la subsecuencia común más larga (LCS) está definida como la subsecuencia más larga que es común en todas las secuencias originales, ésto siempre y cuando no se requiera que los elementos de la subsecuencia se encuentren en posiciones consecutivas dentro de las secuencias originales. El problema anterior es bastante clásico en la informática, puesto que es la base de los programas para comparación de datos y tiene algunas aplicaciones en la bioinformática. Además, es ampliamente utilizado por los sistemas de control

de versiones, como SVN y Git, para conciliar múltiples cambios realizados en una colección de archivos controlados por versiones.

El problema planteado para esta práctica tiene como objetivo determinar el porcentaje de parecido para dos archivos de texto que contienen código fuente haciendo uso del algoritmo de la subsecuencia común más larga, por lo tanto, esta práctica tiene como objetivo analizar la complejidad temporal del algoritmo propuesto mediante programación dinámica.

2 Conceptos Básicos

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

Programación dinámica : Es una técnica que se utiliza para dar solución a un problema de optimización, dividiendo este problema en subproblemas más sencillos, en donde la solución óptima al problema general depende estrictamente de la solución óptima a sus subproblemas.

Esto se puede lograr de dos formas:

Enfoque de arriba hacia abajo: Es la consecuencia directa de la formulación recursiva de cualquier problema. Si la solución a cualquier problema se puede formular de manera recursiva haciendo uso de la solución a sus subproblemas, y si sus subproblemas se superponen, entonces se pueden almacenar fácilmente estas soluciones a los subproblemas mediante una tabla. Cuando intentamos resolver un nuevo subproblema, primero observamos la tabla para determinar si ya se encuentra resuelto, puesto que, si se ha registrado una solución, podemos utilizarla directamente o en caso contrario resolvemos el subproblema y agregamos la solución a la tabla.

Enfoque de abajo hacia arriba: Cuando formulamos una solución a un problema de forma recursiva, podemos intentar replantearlo de una manera de abajo hacia arriba, intentando resolver los subproblemas primero y utilizar sus soluciones para construir las soluciones a subproblemas mayores. Lo anterior también se suele hacer mediante una tabla generando iterativamente soluciones para subproblemas cada vez más grandes utilizando las soluciones para subproblemas pequeños.

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 **g(n)**. Θ(**g(n)**) denota el conjunto de funciones definidas como:

Θ(**g(n)**) = { **f(n)**: $\exists$n C_1, $C_2 > 0$ y $n_0 > 0$ Tal que $0 \leq C_1$ **g(n)** $\leq$ **f(n)** $\leq$ C_2 **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.

Planteamiento del problema: Implementar un algoritmo que permita comparar dos archivos, los cuales contienen código fuente a través de programación dinámica, lo anterior haciendo uso del algoritmo de la Subsecuencia Común más larga.

Algoritmo de la Subsecuencia Común más larga: De manera general, la función recibe dos arreglos de cadenas. El algoritmo utiliza dos tablas B y C, en donde llenamos la primera fila y columna de ceros, luego nos movemos a través de la tabla mediante los dos ciclos for, cada uno con su respectivo índice (i y j respectivamente), entonces, mediante una condición preguntamos si el elemento i-ésimo de X es igual al elemento j-ésimo de Y, si esto se cumple añadimos a la posición de la tabla C y B con índice i y j, la suma de la posición C con fila igual i-1 y columna j-1 más 1, en B añadimos una flecha en diagonal. Si lo anterior no se cumple, significa que los elementos son diferentes, por lo tanto, preguntamos si la posición con fila igual a i-1 y columna j, es menor o igual a la posición C con fila i y columna j-1, si esto se cumple asignamos a la posición C con fila i y columna j, la posición C con fila i-1 y columna j, además, a la posición B con fila i y columna j, una flecha hacia arriba. En caso de no cumplirse ninguna de las condiciones, asignamos a la posición C con fila i y columna j, la posición C con fila i y columna j-1, también, a la posición B con fila i y columna j, una flecha horizontalmente. Por último, regresamos ambas tablas.

Algorithm 1 *LCS-LENGTH*

Require: X, Y
Ensure: Tablas C y B

```
1: m = X.length
2: n = Y.length
3: let B[1..m,1...n] and C[0..m,0...n] be new tables
4: for  i = 1 to m do
5:     C[i,0] = 0
6: end for
7: for j=0 to n do
8:     C[0,j] = 0
9: end for
10: for i = 1 to m do
11:     for j = 1 to n do
12:         if X_i == Y_j then
13:             C[i,j] = C[i-1,j-1] + 1
14:             B[i,j] = " ↖ "
15:         else if C[i-1,j] ≥ C[i, j-1] then
16:             C[i,j] = C[i-1,j]
17:             B[i,j] = " ↑"
18:         else
19:             C[i,j] = C[i,j-1]
20:             B[i,j] = " ← "
21:         end if
22:     end for
23: end for
24: return C and B
```

Algoritmo propuesto: Partiendo del algoritmo anterior, implementamos la función ComparaCadenas, la cual recibe los dos arreglos que contienen los caracteres de los archivos que deseamos comparar, además, recibe el tamaño de ambos arreglos m y n respectivamente y nos devuelve el porcentaje de parecido de ambos archivos. Comenzamos declarando una matriz para la tabla C con su respectivo tamaño (m+1,n+1) y declaramos dos iteradores i y j para recorrer esta matriz, luego, llenamos la primera fila y columna con ceros. Posteriormente iniciamos dos ciclos for anidados para poder acceder a cada una de las posiciones de la matriz C, inmediatamente después preguntamos si el elemento i-ésimo menos uno del arreglo X es igual al elemento j-ésimo menos uno del arreglo Y, si esto se cumple asignamos a la posición C con índices i y j el valor correspondiente, si esto no se cumple implica que los elementos son diferentes y tendríamos dos casos, entonces, preguntamos por el primer caso en donde el elemento de C[i-1][j] es mayor al elemento de C[i][j-1], si esto se

cumple asignamos a la posición C con índices i y j el valor correspondiente y en caso de no cumplirse lo anterior simplemente asignamos a la posición C con índices i y j el valor correspondiente al segundo caso para valores diferentes, por último, calculamos el porcentaje de parecido haciendo uso de la posición C[m][n], la cual multiplicamos por cien y dividimos por el tamaño máximo de los dos arreglos (X y Y respectivamente).

Algorithm 2 *ComparaCadenas*

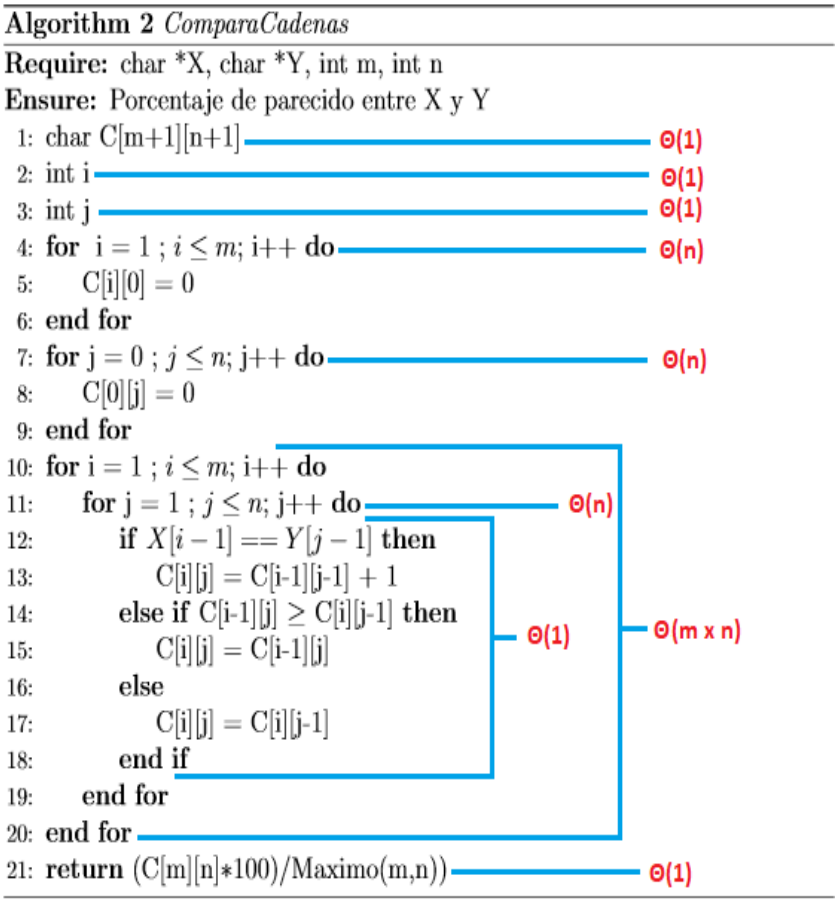
Require: char *X, char *Y, int m, int n
Ensure: Porcentaje de parecido entre X y Y

```
1: char C[m+1][n+1]
2: int i
3: int j
4: for  i = 1 ; i ≤ m; i++ do
5:     C[i][0] = 0
6: end for
7: for j = 0 ; j ≤ n; j++ do
8:     C[0][j] = 0
9: end for
10: for i = 1 ; i ≤ m; i++ do
11:     for j = 1 ; j ≤ n; j++ do
12:         if X[i − 1] == Y[j − 1] then
13:             C[i][j] = C[i-1][j-1] + 1
14:         else if C[i-1][j] ≥ C[i][j-1] then
15:             C[i][j] = C[i-1][j]
16:         else
17:             C[i][j] = C[i][j-1]
18:         end if
19:     end for
20: end for
21: return (C[m][n]∗100)/Maximo(m,n))
```

3 Experimentación y Resultados

Análisis a priori para el algoritmo propuesto: Para el algoritmo **ComparaCadenas** llevaremos a cabo el análisis mediante segmentos de código. Si observamos el pseudo-código podemos determinar que el peor caso y mejor caso son iguales, ya que siempre ejecutará todas las instrucciones para cualquier entrada y tamaño de los arreglos.

Se tiene que



De lo anterior podemos concluir que

$\therefore$ **ComparaCadenas** $\in \Theta(mxn)$

Para realizar las pruebas comenzamos implementando el algoritmo como una función llamada **"ComparaCadenas"** con base en el pseudo-código del **Algorithm 2 ComparaCadenas**, mediante el Lenguaje de Programación en C, es importante mencionar que la función Maximo unicamente regresa el mayor de dos números enteros.

```c
int ComparaCadenas(char *X, char *Y, int m, int n){//El algoritmo recibe los tamaños
    char C[m+1][n+1];// creamos una matriz para la tabla C
    int i, j;//Iniciamos los indices

    for(i = 1; i<=m; i++)//llenamos la primera fila con 0
        C[i][0] = 0;
    for(j = 0; j<=n; j++)//lenamos la primera columna con 0
        C[0][j] = 0;

    for(i=1; i<=m; i++){//recorremos filas
        for(j=1;j<=n;j++){//recorremos columnas
            if(X[i-1] == Y[j-1])//Los elementos en comun de ambos arreglos (X,Y)
                C[i][j] = C[i-1][j-1] + 1;
            else if(C[i-1][j]>=C[i][j-1])//Elementos diferentes caso 1
                C[i][j] = C[i-1][j];
            else//elementos diferentes caso 2
                C[i][j] = C[i][j-1];
        }
    }
    return (C[m][n]*100)/Maximo(m,n);//Calculamos y regresamos el % de parecido
}
```

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

Además, utilizaremos la función **Principal** la cual nos servirá para ejecutar el algoritmo y analizar los resultados obtenidos.

```c
void Principal(){
    FILE *f1 = fopen("c1.txt","rt");
    FILE *f2 = fopen("c6.txt","rt");
    char X[TAM], Y[TAM];

    int m = RecogeTxt(f1, X), n = RecogeTxt(f2, Y);//Recogemos los archivos de texto y obtenemos su tamaño
    printf("X: ");
    puts(X);//Imprimimos el 1er documento txt
    puts("");
    printf("Y: ");
    puts(Y);//Imprimimos el 2do documento txt
    puts("");
    printf("Tiene un parecido de %d porciento", ComparaCadenas(X,Y,m,n));//mostramos el porcentaje


    fclose(f1);fclose(f2);
}
```

Figura 2. Código para la función **Principal** del **algoritmo propuesto** en lenguaje C.

Ejemplos del funcionamiento del algoritmo propuesto: Para todas las pruebas se utilizaron archivos de texto, a los cuales se les retiran los siguientes caracteres: salto de línea, espacio, tabulación horizontal, tabulación vertical y el feed. Además, como ya mencionamos se guardarán en los arreglos X y Y respectivamente.

Primera prueba: En esta prueba comparamos el mismo archivo de texto, es decir, ambos arreglos son exactamente iguales. Como podemos observar en el resultado de la ejecución (**Figura 3**) X y Y son iguales, por lo tanto, el porcentaje de parecido es del 100 %.

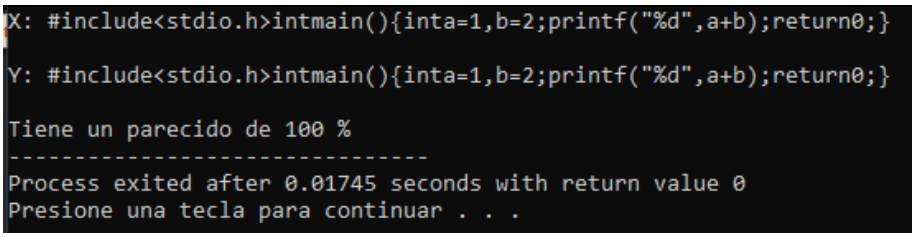


Figura 3. Resultado de la ejecución para la **primera prueba**.

Segunda prueba: En esta prueba comparamos dos archivo de texto casi iguales, es decir, ambos arreglos contienen bastantes caracteres en común. Como podemos observar en el resultado de la ejecución (**Figura 4**) X y Y tienen un porcentaje de parecido del 93 %.

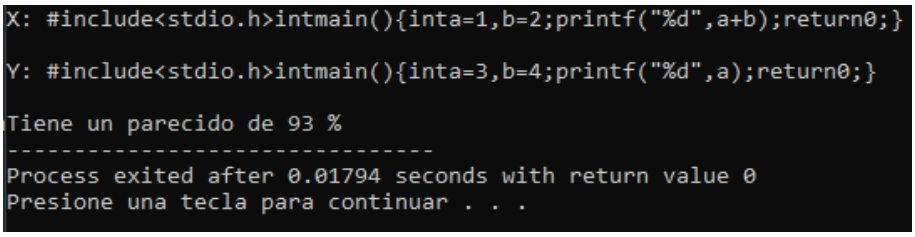


Figura 4. Resultado de la ejecución para la **segunda prueba**.

Tercera prueba: En esta prueba comparamos dos archivos de texto con algunas similitudes, es decir, ambos arreglos tiene un poco más de la mitad de caracteres en común. Como podemos observar en el resultado de la ejecución (**Figura 5**) X y Y tienen un porcentaje de parecido del 54 %.

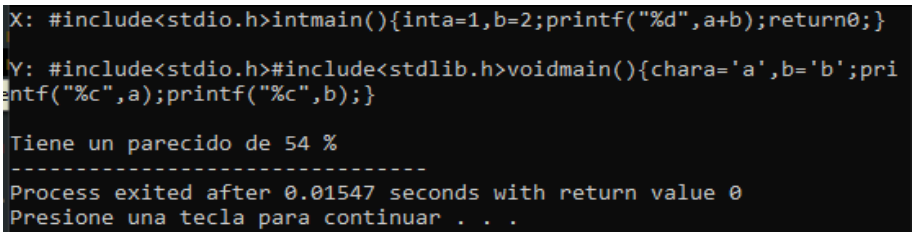


Figura 5. Resultado de la ejecución para la **tercera prueba**.

Cuarta prueba: En esta prueba comparamos dos archivos de texto totalmente diferentes, los cuales resuelven problemas totalmente distintos, sin embargo, puesto que ambos archivos contienen código fuente poseen algunas palabras reservadas en común. Podemos observar en el resultado de la ejecución (**Figura 6**) X y Y tienen un porcentaje de parecido del 13 %.

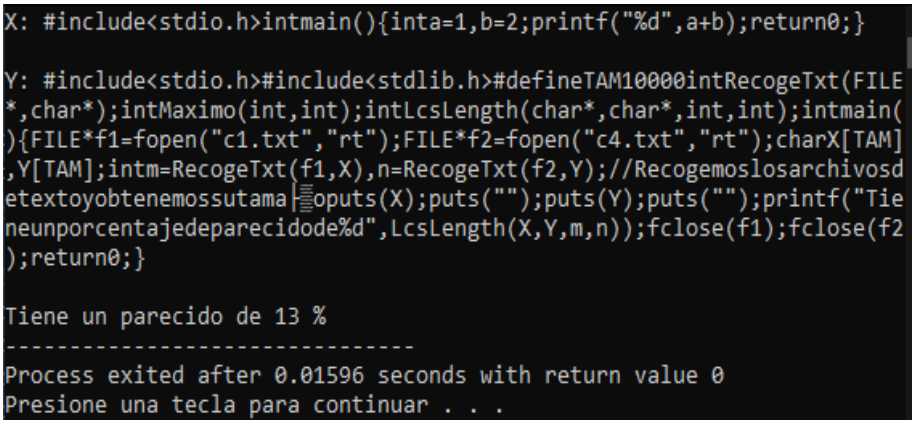


Figura 6. Resultado de la ejecución para la **cuarta prueba**.

Quinta prueba: En esta prueba comparamos dos archivos de texto distintos que contienen código fuente, los cuales desempeñan funciones totalmente distintas, de manera similar podemos observar en el resultado de la ejecución (**Figura 7**) X y Y tienen un porcentaje de parecido del 11 %, puesto que utilizan algunas palabras reservadas en común.

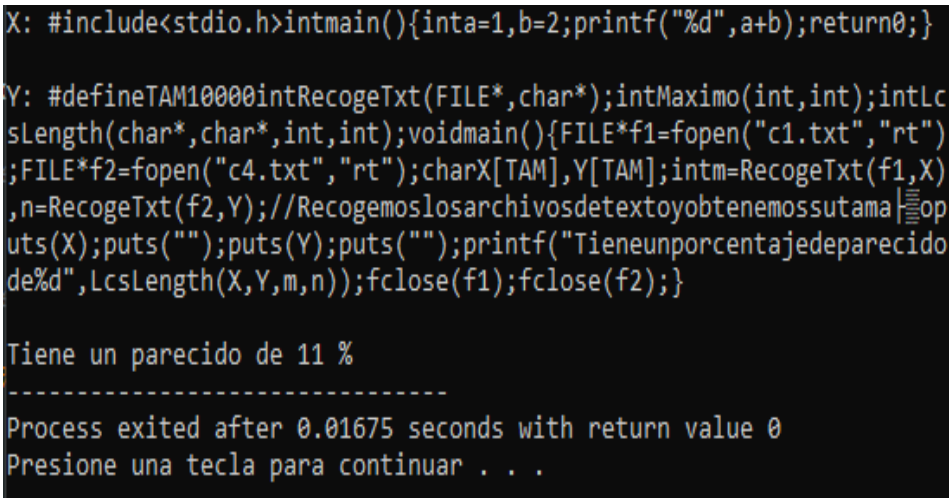


Figura 7. Resultado de la ejecución para la **quinta prueba**.

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: De los resultados obtenidos durante las pruebas realizadas para el algoritmo propuesto **(CompararCadenas)** es posible observar que el algoritmo ofrece una solución para el problema planteado, puesto que determina el porcentaje de parecido para los dos archivos de texto que contienen código fuente. Para la **primera prueba** en donde comparamos el mismo archivo, el porcentaje de parecido obviamente es el 100%, lo cual es completamente cierto. Para la **segunda prueba** los archivos son casi iguales, únicamente se modifican alguos caracteres, el porcentaje de parecido obtenido es de 93%, esté resultado también es cierto. Para la **tercera prueba** podemos observar que el porcentaje de parecido es de 54% pero si analizamos el código que contienen ambos archivos notamos que cumplen funciones distintas, a pesar de ésto comparten algunas instrucciones en común y es por esto el alto porcentaje de parecido. De manera similar para la **cuarta y quinta prueba** utilizamos códigos que cumplen funciones totalmente distintas, sin embargo, presentan un porcentaje de parecido considerable, para responder lo anterior, es importante mencionar que estos archivos siempre van a tener un porcentaje de parecido debido a que cuando programamos utilizamos palabras reservadas que no podemos alterar de ninguna manera, lo que provoca tener caracteres en común, además, esto implica que algunos archivos que tienen funciones totalmente distintas o que no sean de la misma naturaleza tengan cierto porcentaje de parecido pero ésto no quiere decir que sean iguales, simlemente comparten algunas instrucciones en común.

Conclusiones individuales (Brayan Ramirez Benitez): Durante el desarrollo de está práctica surgieron algunos problemas, principalmente debido a la matriz C que se utilizó para implementar el algoritmo propuesto para resolver el problema planteado, ya que ocurría un conflicto con los índices i y j que se utilizaron para llenar los campos de la matriz antes mencionada, sin embargo, analizando con más detalle como funciona el algoritmo de la subsecuencia común más larga, es posible dar solución a este problema de una manera relativamente sencilla de entender.

Brayan Ramirez Benitez.

5 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] Choudhary, V. (2018, 29 marzo). *). Introduction to Dynamic Programming.* Developer Insider. Recuperado 25 de noviembre de 2021, de https://developerinsider.co/introduction-to-dynamic-programming/

[4] Algorithms. (2016, 24 abril). *Introduction to Dynamic Programming 1 Tutorials & Notes* HackerEarth. Recuperado 25 de noviembre de 2021, de https://www.hackerearth.com/practice/algorithms/dynamic-programming/introduction-to-dynamic-programming-1/tutorial/