UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Informe Proyecto 1 ADA II

(fuerza bruta, programación dinámica y programación voraz)

Asignatura:

Análisis y diseño de algoritmos II

Programa:

Ingeniería de Sistemas

Estudiantes:

Victor Manuel Álzate Morales - 202313022
Juan Esteban López Aránzazu - 202313026
Daniel Meléndez Ramirez - 202313024
Diego Alejandro Tolosa Sanchez - 202313023

Semestre:

Tercer Semestre

Docente:

Carlos Delgado

Universidad del Valle – Sede Tuluá

UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Universidad del Valle

Informe del proyecto

Estructura de archivos del proyecto

1. Carpeta “app” contiene todos los archivos de la aplicación
2. Carpeta “data_inputs” contiene los archivos de texto con los datos de entrada para los problemas de la terminal inteligente y la subasta pública
3. Carpeta “public_auction” contiene los archivos de los algoritmos de fuerza bruta, programación dinámica y programación voraz del problema de la subasta pública
    a. brute_force_auction.py (algoritmo de fuerza bruta)
    b. dynamic_programming_auction.py (algoritmo de programacion dinamica)
    c. greedy_auction.py (algoritmo de programacion voraz)
4. Carpeta “smart_terminal” contiene los archivos de los algoritmos de fuerza bruta, programación dinámica y programación voraz del problema de la terminal inteligente
    a. brute_force_terminal.py (algoritmo de fuerza bruta)
    b. dynamic_programming_terminal.py (algoritmo de programacion dinamica)
    c. greedy_terminal.py (algoritmo de programacion voraz)
5. Carpeta “tests” contiene los archivos para correr las pruebas de cada algoritmo
    a. test_auction.py (pruebas del problema de la subasta publica)
    b. test_terminal.py (pruebas del problema de la terminal inteligente)
6. Carpeta “graphics” contiene los archivos para generar las gráficas de cada problema
    a. auction_graphics.py (graficas de tiempo para la subasta publica)
    b. terminal_graphics.py (graficas de tiempo para terminal inteligente)
7. Archivo main es el archivo con la interfaz gráfica de usuario

Comando para ejecutar el programa

python ./app/main.py

Comandos para ejecutar los tests

python -m unittest ./app/tests/test_terminal.py

python -m unittest ./app/tests/test_auction.py

UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Universidad del Valle

Problema Terminal Inteligente

Para este problema se tiene que buscar el mínimo costo para transformar la cadena A en la cadena B usando las operaciones advance, delete, replace, insert y kill, donde cada operación tiene un costo

Datos de entrada

Cadena A, Cadena B, Costo advance, Costo delete, Costo replace, Costo insert y Costo kill

Para los siguientes algoritmos se tienen en cuenta algunos caso base

1. Cuando no hay caracteres tanto en la cadena A como la cadena B
2. Cuando no hay caracteres en la cadena A pero si hay en la cadena B
3. Cuando hay caracteres en la cadena A pero no hay en la cadena B

Entendiendo el problema

Para el caso

Cadena A: ingenioso

Cadena B: ingeniero

Posible solución 1

Operaciones: advance, advance, advance, advance, advance, advance, replace, replace, advance

Costo: 6*a+r

Posible solución 2

Operaciones: kill, insert i, insert n, insert g, insert e, insert n, insert i, insert e, insert r, insert o

Costo: 9*i+k

Conclusión: Existen muchas soluciones para el problema, pero se debe tener en cuenta el costo de cada operación, en este caso se busca el costo mínimo para transformar la cadena A a la cadena B

Universidad del Valle

Fuerza Bruta

El algoritmo prueba todas las combinaciones posibles para transformar la cadena, cuenta con tres casos base, el primer caso base es cuando ya no hay caracteres en la cadena A y la cadena B entonces retorna costo cero, el segundo caso es cuando hay caracteres en la cadena A pero no hay en la cadena B entonces retorna el mínimo costo de usar la operación delete y la operación kill, el tercer caso es cuando no hay caracteres en la cadena A pero si hay en la cadena B entonces retorna el costo de usar la operación insert, para los casos recursivos del algoritmo cuando en la posición i de la cadena A y la posición j de la cadena B son iguales entonces se hace un llamado recursivo usando la operación advance, de lo contrario se hace un llamado recursivo para la operación replace, se hace un llamado recursivo para la operación delete y para la operación insert, finalmente retorna el mínimo costo de hacer estas operaciones

Complejidad Temporal Fuerza Bruta

La complejidad temporal del algoritmo es O(4^(n+m)) por que se hacen 4 llamados recursivos, ya que se hacen los llamados recursivos para las operación advance, replace, insert y delete, la operación kill es un caso base, para la complejidad temporal se tiene en cuenta que n es el tamaño de cadena A y m es el tamaño de la cadena B

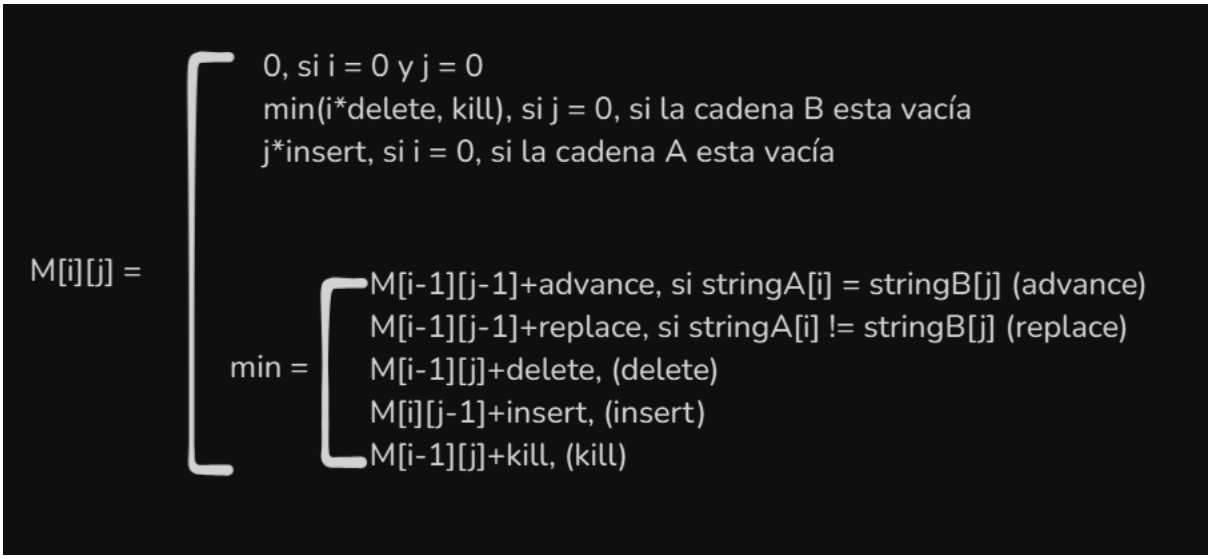
Programación Dinámica

Para este problema se usó como base el problema de la distancia de levenshtein, la diferencia es que usa solo 3 operaciones, delete, insert y replace, también se usó como base el problema de subsecuencias más larga en común entre dos cadenas LCS, el enfoque de este problema es crear matrices donde cada posición de la matriz se encuentra la solución de cada subproblema hasta encontrar la solución del problema en general, en el caso de LCS se entiende que para la posición i y j quiere decir que el valor almacenado en esa posición de la matriz indica que la subsecuencia común más larga entre las dos cadenas es [i,j], este enfoque también se busca para el problema de la terminal inteligente

Subestructura óptima

La solución óptima del problema más grande (convertir toda la cadena A en B) se construye a partir de las mejores soluciones de problemas más pequeños. Por ejemplo, para transformar las primeras i letras de la cadena A en las primeras j letras de la cadena B, usamos las soluciones ya calculadas para convertir cadenas más cortas.

Definición de recurrencia



Donde i representa el puntero de la cadena A y j el puntero de la cadena B, entonces M[i][j], representa el costo mínimo de transformar la cadena A a la cadena B

El algoritmo crea una matriz de tamaño (n+1)*(m+1), el +1 significa que se deben considerar los casos vacíos, inicialmente se llena la matriz con ceros, luego se recorre la matriz calculando cada posición [i,j] con el mínimo costo para transformar la cadena A en la cadena B usando las operaciones de advance, delete, replace, insert y kill, la solución óptima se encuentra en la posición [0,0] de la matriz, se encuentra en esta posición porque la matriz se llena de abajo hacia arriba (enfoque button up), hay casos bases cuando se recorrieron todos los caracteres de la cadena A y la cadena B entonces salta a la siguiente ejecución, cuando la cadena A está vacía entonces solo se puede usar insert, caso cuando la cadena B está vacía entonces se tiene que buscar el menor entre usar delete y kill, cuando los caracteres en la posición i y j son iguales entonces se puede usar advance, caso contrario se puede usar replace, delete, insert y kill entonces se tiene que buscar el costo menor entre estas operaciones

Complejidad Temporal Programación Dinámica

La complejidad temporal del algoritmo es de O(n*m), donde n es el tamaño de la cadena A y m es el tamaño de la cadena B

Universidad del Valle

Programación Voraz

Se toman decisiones locales, es decir en cada iteración se usa la operación con el mínimo costo, para este algoritmo se usa un bucle while y se detiene cuando la posición i sea mayor al tamaño de la cadena A o la posición j sea mayor al tamaño de la cadena B, se tienen los casos cuando el carácter de la posición i de la cadena A es igual al carácter de la posición j de la cadena B entonces la posición i y la posición j incrementa uno, si los caracteres son diferentes entonces se puede usar replace y las posiciones i y j incrementan uno, sino también se puede usar delete entonces la posición i aumenta uno, por último se puede usar insert donde la posición j incrementa uno, finalmente se escoge entre la operación que tenga el menor costo

Complejidad Temporal Programación Voraz

La complejidad temporal del algoritmo es el mínimo de O(min(n, m)), donde n el tamaño de la cadena A y m es el tamaño de la cadena B

Problema Subasta pública

Para este problema se tiene que buscar la máxima ganancia que se puede obtener para la compra de las acciones de las ofertas y del gobierno, se debe retornar un arreglo con cada xi: número de acciones que compraron y donde xi=0 o mi<=x<=Mi, donde xi = <x1,x2…> donde la sumatoria de xi*pi es la ganancia máxima obtenida

Datos de entrada

A: número de acciones, B: precio que paga el gobierno, n: número de ofertas, cada oferta se representa como <pi,mi,Mi>, donde pi: precio de la oferta, mi: número mínimo de acciones que pueden comprar y Mi: número máximo de acciones que pueden comprar

Para los siguientes algoritmos se debe considerar algunos caso base

1. Cuando no hay ofertas y el número de acciones es igual a cero
2. Cuando no hay ofertas y el número de acciones es mayor a cero
    a. Las acciones restantes las compra el gobierno

Entendiendo el problema

Para el caso

A: 1000

B: 100

n: 2

Ofertas: [[500, 100, 600], [450, 400, 800]]

La solución óptima es:

Ganancia máxima: 480.000

Asignación: [600, 400, 0], esto indica que el oferente 1 compró 600 acciones a un precio de 500, el oferente 2 compró 400 acciones a un precio de 450 y que el gobierno no compro ninguna acción

Teniendo que el vr(x): 600*500+400*450+0*100 = 480.000

Otra solución no óptima es:

Ganancia: 340.000

Asignación: [600, 0, 400], esto indica que el oferente 1 compró 600 acciones a un precio de 500, el oferente 2 no compro ninguna acción y que el gobierno compró 400 acciones a un precio de 100

Teniendo que el vr(x): 600*500+0*450+400*100 = 340.000

Conclusión: Existen muchas combinaciones para asignar el número de acciones de cada oferente y el gobierno, pero se debe garantizar la solución óptima teniendo en cuenta el precio (pi) y el rango de acciones que puede comprar cada oferente [mi, Mi]

Universidad del Valle

Una primera aproximación

Considere el algoritmo que lista las posibles asignaciones:

$$< y_1, y_2, ..., \sum_{i=1}^{n-1} y_i = A >$$

donde $y_i \in \{0, M_i\}$, luego calcula el vr para cada asignación y selecciona la mejor.

Este algoritmo tiene el enfoque de fuerza bruta, ya que lista todas las posibles asignaciones que puede comprar cada oferente y luego selecciona la asignación de mejor ganancia, es decir se debe asignar la cantidad de acciones que compra cada oferente y luego calcular el valor vr(x) que mejor ganancia ofrezca, es decir la sumatoria de pi*xi, donde pi es el precio del oferente y xi la cantidad asignada al oferente que cumpla que mi <= xi <= Mi

Este algoritmo garantiza la solución óptima, pero es el más costoso en complejidad temporal

Fuerza Bruta

El algoritmo prueba todas las combinaciones posibles para asignar las acciones a cada oferente hasta que encuentra la mejor asignación la cual tiene la mayor ganancia posible, cuenta con un caso base, cuando ya no hay ofertas entonces se comprueba si el número de acciones es igual a cero, en ese caso se calcula la ganancia total dado el arreglo de asignaciones, en otro caso si el número de acciones es mayor a cero significa que quedaron acciones por vender entonces estas acciones las compra el gobierno a un precio B, el caso recursivo consiste en un bucle for que itera desde el número de acciones mínimas hasta el número de acciones máxima incluyendo el cero del oferente actual y si cumple la condición que el número de acciones A es mayor o igual a cantidad del oferente entonces se hace un llamado recursivo con el siguiente oferente y así de manera sucesiva probando las combinaciones

Complejidad Temporal Fuerza Bruta

La complejidad temporal del algoritmo es de O((max_qty+1)^n), donde n es el número de oferentes y max_qty es la cantidad máxima de acciones de cada oferente

Universidad del Valle

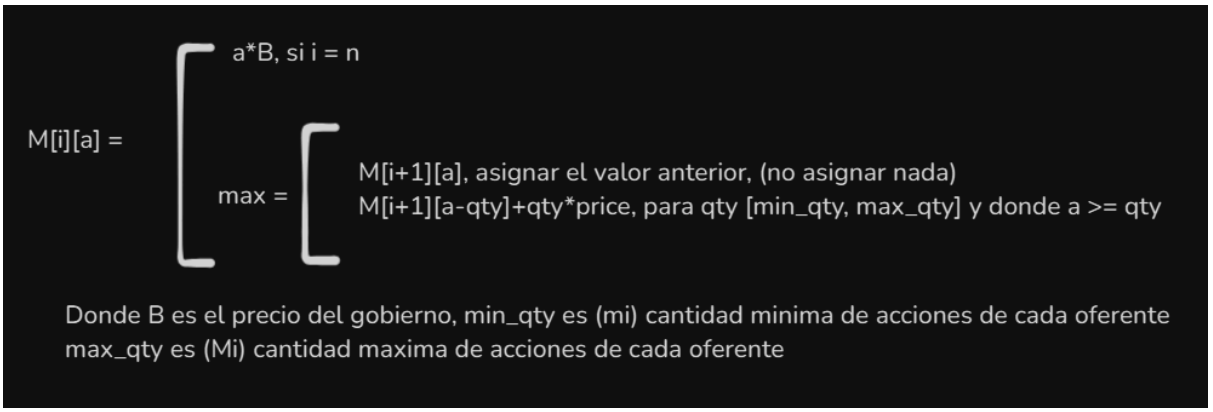
Programación Dinámica

Para este algoritmo se usó como caso base el problema de la mochila 0/1, ya que el enfoque de este algoritmo es maximizar el beneficio de llevar o no llevar el objeto, en este caso maximizar la ganancia al vender las acciones a cada oferente y al gobierno, la solución del problema de la mochila se representa como un arreglo, por ejemplo <1,1,0>, para el caso de la subasta pública se busca asignar el número de acciones que compra cada oferente, donde la solución se representa como un arreglo, por ejemplo <450,0,310>

Subestructura óptima

La solución óptima para el problema completo (asignar todas las acciones A a los oferentes) se construye a partir de las soluciones óptimas para problemas más pequeños. Por ejemplo, la mejor forma de asignar acciones a los i primeros oferentes depende de las mejores decisiones tomadas para los i−1 oferentes y las acciones restantes.

Definición de recurrencia



Donde i representa la oferta y a las acciones disponibles, entonces M[i][a], representa el máximo beneficio o la ganancia máxima de la oferta i y la cantidad de acciones a

El algoritmo crea una matriz de tamaño (n+1)*(A+1), +1 significa que se debe considerar los casos vacíos, inicialmente se llena la matriz con -infinito, porque se encontrará la ganancia máxima, luego se recorre la matriz y se calcula cada posición [i,j] con la ganancia máxima, la solución óptima se encuentra en la posición [0,A] de la matriz, es decir la ganancia máxima para n ofertas y A acciones, la solución se encuentra en la fila cero porque el algoritmo llena la matriz de abajo hacia arriba (enfoque button up), para encontrar el valor máximo que se puede asignar para cada oferta, se recorre el número de acciones mínimo hasta el número de acciones máximo para cada oferta

Universidad del Valle

Complejidad Temporal Programación Dinámica

La complejidad temporal del algoritmo es de O(n*A*(max_qty-min_qty)), donde n es el número de ofertas, A es el número de acciones, min_qty la cantidad mínima de acciones de cada oferente y max_qty es la máxima cantidad de acciones de cada oferente

Programación Voraz

Para este algoritmo ya que se toman decisiones locales, primero se ordena la lista de oferentes de mayor a menor precio que comprara las acciones, se usa un bucle for que itera cada oferente, buscando la mayor cantidad posible de acciones para asignar, es decir desde el rango de número máximo de acciones hasta el número mínimo de acciones del oferente actual hasta que se cumpla la condición del número de acciones A mayor o igual a la cantidad de acciones, entonces se resta el número de acciones A con la cantidad máxima de acciones que se puede asignar, por último se tiene en cuenta las acciones restantes y en ese caso las compra el gobierno por un precio B

Complejidad Temporal Programación Voraz

La complejidad temporal del algoritmo es de O(n), donde n es el número de oferentes

Pruebas y comparación de tiempo del programa

El algoritmo con menor tiempo de ejecución es el algoritmo de programación voraz, seguido de algoritmo de programación dinámica y finalmente fuerza bruta. Sin embargo el algoritmo de programación voraz no garantiza la solución óptima en todos los casos, mientras que el algoritmo de programación dinámica y fuerza bruta garantizan la solución óptima del problema

Para cada problema es posible que existan casos donde los algoritmos encuentran la solución óptima pero las combinaciones que se hicieron sea diferente entre los algoritmos, es decir existen diferentes soluciones y la primera que encuentra cada algoritmo es la que al final retornan

Terminal inteligente

Caso1
stringA: francesa
stringB: ancestro

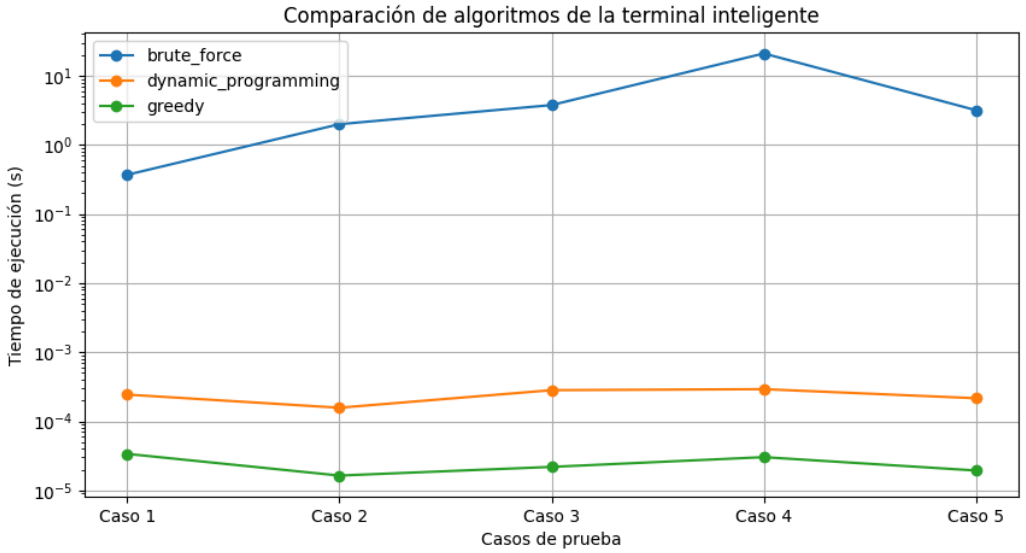
operaciones: advance: 1, delete: 2, replace: 3, insert: 2, kill: 1

Caso2
stringA: ingeniero
stringB: ingenioso
operaciones: advance: 1, delete: 2, replace: 3, insert: 2, kill: 1

Caso3
stringA: algorithm
stringB: altruistic
operaciones: advance: 1, delete: 2, replace: 3, insert: 2, kill: 1

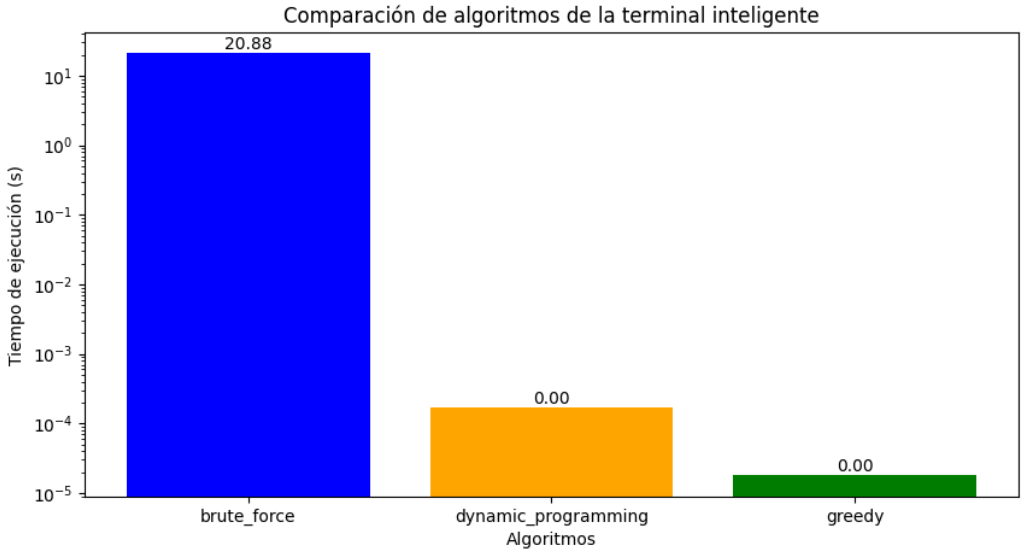
Caso4
stringA: programming
stringB: programmer
operaciones: advance: 8, delete: 2, replace: 3, insert: 2, kill: 1

Caso5
stringA: computer
stringB: computation
operaciones: advance: 1, delete: 21, replace: 3, insert: 6, kill: 1

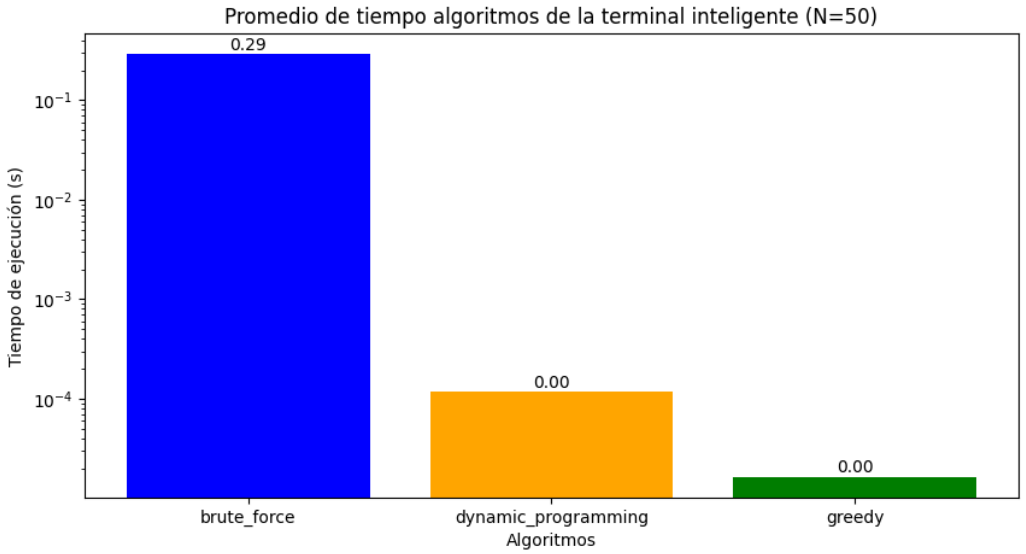


Caso individual
stringA: programming

stringB: programmer
operaciones: advance: 8, delete: 2, replace: 3, insert: 2, kill: 1



Caso individual promedio 50 ejecuciones
stringA: francesa
stringB: ancestro
operaciones: advance: 8, delete: 2, replace: 3, insert: 2, kill: 1



UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Universidad del Valle

Subasta pública

Caso1
A: 1000
B: 100
Oferentes: [450, 100, 400], [400, 100, 400], [500, 400, 550]

Caso2
A: 1000
B: 100
Oferentes: [150, 375, 385], [200, 250, 265], [300, 250, 345]

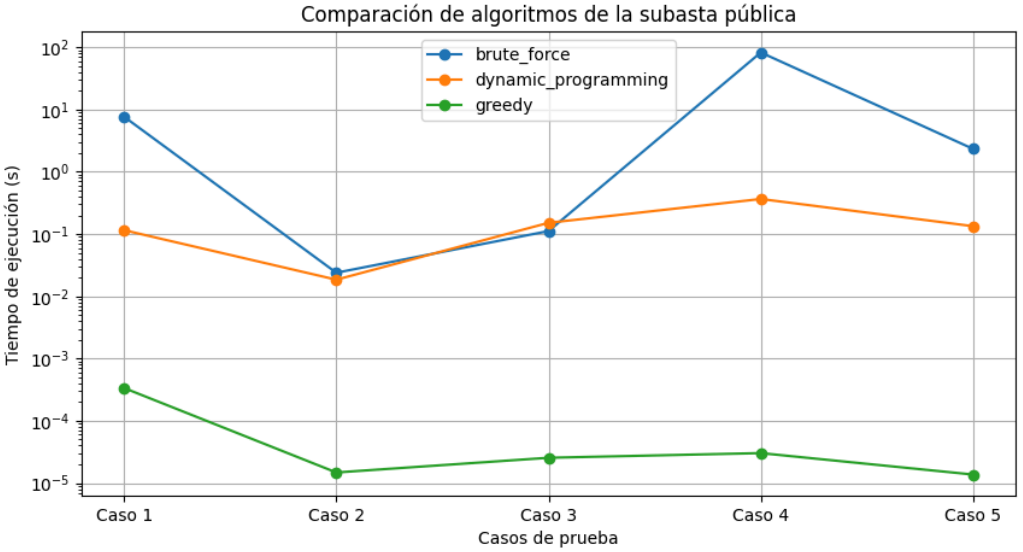
Caso3
A: 1000
B: 100
Oferentes: [500, 100, 600], [450, 400, 800]

Caso4
A: 2000
B: 150
Oferentes: [500, 100, 600], [450, 400, 800], [600, 500, 1000]

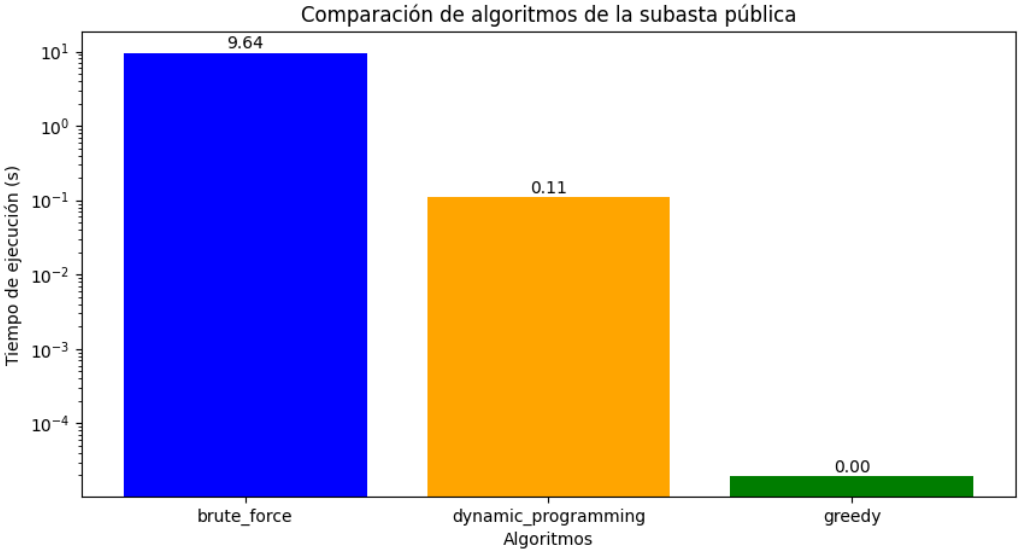
Caso5
A: 2000
B: 100
Oferentes: [450, 100, 300], [400, 280, 412], [500, 550, 618]

UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Universidad del Valle



Caso individual
A: 1000
B: 100
Oferentes: [450, 100, 400], [400, 100, 400], [500, 400, 550]



UNIVERSIDAD DEL VALLE - SEDE REGIONAL TULUÁ
PROGRAMA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ANÁLISIS Y DISEÑO DE ALGORITMOS II

Universidad del Valle

Caso individual promedio 50 ejecuciones
A: 1000
B: 100
Oferentes: [450, 100, 400], [400, 100, 400], [500, 400, 550]