ALGORITMOS Y ESTRUCTURAS DE DATOS II, GRADO INGENIERÍA INFORMÁTICA, CURSO 24/25 PRÁCTICA DE DIVIDE Y VENCERÁS Y ANÁLISIS DE ALGORITMOS

Eduardo Terry Gavilá Grupo: 3.4

Aaron Atonga Ruiz Grupo: 3.4

Correos:

eduardo.terryg@um.es

a.atongaruiz@um.es

Profesor: JOSE RAMÓN HOYOS BARCELO

Introducción:

El problema que nos ha tocado ha sido el 4. Este problema consiste en encontrar la subcadena m, la cual, al sumar el valor absoluto del valor numérico dependiendo de la posición de dicho carácter en el abecedario sea mayor. Ejemplo: la cadena "bbcda" valdria 0+1+1+3 = 5. En esta ocasión lo resolveremos con la técnica divide y vencerás vista en clase dividiendo en dos subproblemas.

Diseño en pseudocódigo:

```
SolDirecta(canvay: string; inicio, ult, m: Entero) <Entero, Entero>
  var dif: Entero
  var maxdif: Entero
  var pos: Entero
  var A: string

  dif:= 0
  maxdif:=0
  pos:= inicio
  A:= extraer_subcadena(inicio, ult-inicio+1)
  for j:=0 to tam(A)-m do
     dif:=0
     for i:=1 to m do
        dif:= abs(A[j+i]-A[j+i-1])
     end
     si dif> maxdif
        maxdif:= dif
        pos:= inicio+j
     fin si
  end
  devolver maxdif, pos


Combinar (canvay: string; inicio, mid, ult, m: Entero) <Entero, Entero>
   var maxdif: Entero
   var pos: Entero
   var caso: <Entero, Entero>

   maxdif:= 0
   pos:= mid

   for i:=0 to m do
      si mid-i >= inicio Y mid-i+(m-1) <= ult
         caso:= SolDirecta(canvay, mid-i, mid-i+(m-1), m)
         si caso.first > maxdif
              maxdif:= caso.first
              pos:= caso.second
         fin si
      fin si
   end
   devolver maxdif, pos


DyV (canvay: string; inicio, mid, ult, m: Entero): <Entero, Entero>
    si  ult-inicio+1 <= m
        devolver SolDirecta (canvay, inicio, ult, m)
    fin si

    mid= (inicio+ult) / 2

    S1:= DyV(canvay, inicio, mid, m)
    S2:= DyV(canvay, mid+1, ult, m)
    S3:= Combinar(canvay, inicio, mid, ult, m)

    si  S1.first== max(S1.first, S2.first, S3.first)
        devolver S1
    fin si
    si  S2.first== max(S1.first, S2.first, S3.first)
        devolver S2
    fin si
    devolver S3
```

En el código visto tenemos la llamada principal DyV la cual comienza comprobando si la llamada se trata de un caso base. Se tratará de un caso base cuando el tamaño de la cadena sea menor o igual que m ya que al ser la cadena menor o igual que la subcadena no tiene mucho sentido dividirla porque esta sería la solución. En ese caso, devuelve el resultado de llamar a la función caso base la cual en el pseudocódigo se llama SolDirecta. Se explicará el funcionamiento de esta función más adelante. La función DyV continúa declarando una variable mid la cual determina la mitad de la cadena para poder dividir en subproblemas. A continuación, se definen las variables s1, s2 y s3 las cuales son de tipo pair<int, int>. La variable s1 será el resultado de llamar recursivamente a DyV con la mitad izquierda de la cadena mientras que s2 será el resultado de llamar a la mitad derecha de la cadena. La variable s3 será el resultado de analizar la parte central de la cadena ya que puede haber soluciones en el medio de la cadena que se divide las cuales no son contempladas lo cual nos impediría obtener la solución óptima. Finalmente, se comprueba cuál de las soluciones tiene el mayor valor y se devuelve ya que en la primera posición se guarda la solución y en la segunda la posición de esta.

La función Combinar analiza la parte central como se ha explicado anteriormente. Para ello se define una variable maxdif la cual tendrá el valor más grande y una variable pos la cual será crucial para obtener la posición de la solución. Ahora se deberá llamar a SolDirecta m-1 veces ya que son el número de posiciones en las cuales va a empezar una cadena perjudicada por la división. Aun en el bucle, se comprueba en cada iteración si SolDirecta ha devuelto un valor mayor a los anteriores y si es así se guarda su valor en maxdif y su posición en pos. Al terminar el bucle se devuelven los resultados.

La función SolDirecta es la encargada de calcular las soluciones. Se declaran las variables maxdif, dif y pos. Las variables maxdif y pos son muy parecidas a las anteriores solo que ahora pos se inicializa con inicio. Dif y maxdif a cero. Ahora vamos a guardar en una variable A la subcadena a analizar de manera que sea más fácil de analizar. Se inicia un bucle que se repite int(A.length()) - m veces ya que si lo ejecutásemos int(A.length()) veces accedería a posiciones de memoria inválidas. Nada más empezar el bucle pondremos la variable dif a cero de manera que podamos llevar la cuenta de manera correcta. A continuación, iniciaremos un bucle anidado al anterior el cual recorrerá desde una posición concreta sus m-1 posiciones vecinas y calculará la diferencia introduciéndola en dif. Al terminar de calcular y sumar las diferencias comprobamos si la diferencia de la subcadena m analizada es mayor que la diferencia máxima. En caso afirmativo, se guardará el valor en maxdif y la posición en pos. Una vez termine el bucle exterior se devolverán las soluciones.

Análisis teórico:

```
SolDirecta(array: String; inicio, ult, m: Entero) <Entero, Entero>
  var dif: Entero
  var maxdif: Entero
  var pos: Entero
  var A: String

  dif:= 0          +1
  maxdif:= 0       +1
  pos:= inicio     +1
  A:= extraer_subcadena(inicio, ult-inicio+1)   +1
  for j:= 0 to tam(A)-m do
     dif:=0
     for i:=1 to m do
        dif:= abs(A[j+i] - A[j+i-1])
     end
     si dif > maxdif
        maxdif:= dif
        pos:= inicio+j
     fin si
  end
  devolver maxdif, pos
```

$t(tam(A), m) \in \Theta(tam(A) \cdot m)$

```
Combinar(array: String; inicio, mid, ult, m: Entero) <Entero, Entero>
  var maxdif: Entero
  var pos: Entero
  var caso: <Entero, Entero>

  maxdif:= 0   +1
  pos:= mid    +1
  for i:=0 to m do  -> m
     si mid-i >= inicio y mid-i+(m-1) <= ult
        caso:= SolDirecta(array, mid-i, mid-i+(m-1), m)  -> tam(A)·m
        si caso.first > maxdif
           maxdif:= caso.first
           pos:= caso.second
        fin si
     fin si
  end
  devolver maxdif, pos
```

$t_m(tam(A), m) \in \Theta(m)$

$t_n(tam(A), m) \in \Theta(tam(A) \cdot m^2)$

```
DyV(array: String; inicio, mid, ult, m: Entero): <Entero, Entero>
  si ult - inicio + 1 <= m   +1
     devolver SolDirecta(array, inicio, ult, m)
  fin si
  mid = (inicio + ult) / 2   +1
  S1:= DyV(array, inicio, mid, m)   -> [a=2=b   c: log_2^2 = 1
  S2:= DyV(array, mid+1, ult, m)    -> [t(n) ∈ O(tam(A))
  S3:= Combinar(array, inicio, mid, ult, m)
  si S1.first == max(S1.first, S2.first, S3.first)   +1
     devolver S1                                     +1
  fin si
  si S2.first == max(S1.first, S2.first, S3.first)   +1
     devolver S2
  fin si                                             +1
  devolver S3                                        +1
```

En la siguiente imagen se puede observar el cálculo del orden de cada función en el mejor y peor caso (si no tiene orden exacto).

Se puede comprobar que la función SolDirecta tiene un orden exacto de tam(A) * m ya que los bucles siempre se iteran completamente y el primero se realiza tam(A) - m veces y el siguiente m veces.

La función combinar pertenece al tiempo mejor de m ya que si nunca se cumple el if de dentro del bucle nunca hay que tener en cuenta el tiempo de la llamada SolDirecta. Pertenece al tiempo peor de tam(A) * m^2 ya que como hemos dicho anteriormente SolDirecta tiene un orden exacto de tam(A) * m y se llama a esta función m veces.

La función DyV tiene un tiempo perteneciente al orden de tam(A) ya que sería el resultado de aplicar el teorema maestro y despreciando m en todos los órdenes de otras funciones ya que es necesario para representar los tiempos y simplificar ya que m es una constante.

Implementación:

```cpp
string GeneradorCadenasSolucionAlInicio(int longitud, int m){   //Crea la cadena con la solucion al principio

    string resultado;
    for (int i = 0; i < m; ++i) {
        if (i%2 == 0){
            resultado += 'z';
        }
        else{
            resultado += 'a';
        }
    }

    resultado += GeneradorCadenas(longitud - m);

    return resultado;
}
string GeneradorCadenasSolucionAlFinal(int longitud, int m){    //Crea la cadena con la solucion al final

    string resultado;
    resultado += GeneradorCadenas(longitud - m);

    for (int i = 0; i < m; ++i) {
        if (i%2 == 0){
            resultado += 'z';
        }
        else{
            resultado += 'a';
        }
    }

    return resultado;
}

string GeneradorCadenasSolucionEnMedio(int longitud, int m){    //Crea la cadena con la solucion en el medio

    int generados = longitud - m;
    int izquierda   = generados / 2;
    int derecha = generados - izquierda;

    string parteIzquierda = GeneradorCadenas(izquierda);
    string parteDerecha  = GeneradorCadenas(derecha);

    string resultado;

    for (int i = 0; i < m; ++i)
        resultado += (i % 2 == 0 ? 'z' : 'a');

    return parteIzquierda + resultado + parteDerecha;
}

```

```cpp
#include <iostream>
#include <cstdlib>
#include <algorithm>
#include <random>
#include <string>
#include <vector>
#include <array>
#include <ctime>
#include <chrono>
#include <sys/time.h>

using namespace std;

int  const tam = 5;

void QueEjercicioMeToca(){

    int Integrante1 = 54961208;
    int Integrante2 = 24420735;

    int problema = ((Integrante1+Integrante2)%10) + 1;

    cout << "Se ha realizado el problema " << problema << endl;
}

string GeneradorCadenas(int longitud) {
    string caracteres = "abcdefghijklmnopqrstuvwxyz";  // Establece qué caracteres pueden aparecer
    int numCaracteres = caracteres.length();  // Calcula el número de caracteres disponibles

    unsigned int seed = chrono::system_clock::now().time_since_epoch().count(); // Genera una semilla usando el tiempo en milisegundos
    srand(seed);  // Usa la semilla dinámica para rand()

    string cadena;  // Cadena a devolver

    for (int i = 0; i < longitud; ++i) {
        // Usa rand() para seleccionar un índice aleatorio en el rango de caracteres
        cadena += caracteres[rand() % numCaracteres];
    }

    return cadena;
}

```

```cpp
array<string, tam*4> GeneradorCasos(int m){

    int casos[] = {600000, 700000, 800000, 900000, 1000000};    //Longitud de cadenas a probar
    array<string, tam*4> cadenas;

    for(int i = 0; i < tam; i++){
        cadenas[i] = GeneradorCadenas(casos[i]);        //Se generan las cadenas aleatorias
        cadenas[i+5] = GeneradorCadenasSolucionAlInicio(casos[i], m);   //Se generan las cadenas con solucion en
        cadenas[i+10] = GeneradorCadenasSolucionEnMedio(casos[i], m);   //Se generan las cadenas con solucion en
        cadenas[i+15] = GeneradorCadenasSolucionAlFinal(casos[i], m);   //Se generan las cadenas con solucion en
    }
    return cadenas;
}

pair<int, int> CasoBase(string array, int inicio, int ult, int m) {
    int dif = 0;
    int maxDif = 0;
    int pos = inicio;

    string A = array.substr(inicio, ult - inicio + 1); //Extrae la subcadena a analizar

    for (int j = 0; j <= int(A.length()) - m; j++) {    //Algoritmo de aplicacion directa iterativo
        dif = 0;
        for (int i = 1; i < m; i++) {           //Se recorre cada caracter y sus m consecutivos y se calcula la d
            dif += abs(A[j + i] - A[j + i - 1]);
        }

        if (dif > maxDif) {     //Se comprueba si la diferencia obtenida es la mayor
            maxDif = dif;
            pos = inicio + j;
        }
    }
    return {maxDif, pos};   //Devolvemos el par diferencia maxima y posicion
}

pair<int, int> Combinar(string array, int inicio, int mid, int ult, int m) {
    int maxDif = 0;
    int pos = mid;

    for (int i = 0; i < m - 1; i++) {   //Este bucle examina las posiciones concretas las cuales se ven perjudica
        if (mid - i >= inicio && mid - i + (m - 1) <= ult) {    //Se comprueba que no se accede a posiciones de m
            pair<int, int> caso = CasoBase(array, mid - i, mid - i + (m - 1), m);   //Se llama al caso base con p

            if (caso.first > maxDif) {  //Se comprueba si el valor obtenido es el maximo y en ese caso se guarda
                maxDif = caso.first;
                pos = caso.second;
            }
        }
    }
    return {maxDif, pos};   //Devolvemos el par diferencia maxima y posicion
}
```

```cpp
pair<int, int> DyV(string array, int inicio, int ult, int m) {

    if (ult - inicio + 1 <= m) {    //Se comprueba si el problema es suficientemente pequeño como para llamar a la resolucion directa
        return CasoBase(array, inicio, ult, m);
    }

    int mid = (inicio + ult) / 2;
    //Dividimos el problema en subproblemas
    pair<int, int> s1 = DyV(array, inicio, mid, m); //Parte izquierda
    pair<int, int> s2 = DyV(array, mid + 1, ult, m); //Parte derecha
    pair<int, int> s3 = Combinar(array, inicio, mid, ult, m);   //Analisis de la parte central

    if (s1.first >= s2.first && s1.first >= s3.first){  //Se comprueba cual es la mejor solucion en funcion del valor de la diferencia de posiciones y se devuelve
        return s1;
    }

    if (s2.first >= s1.first && s2.first >= s3.first){
         return s2;
    }

    return s3;
}
```

```cpp
int main(){

    QueEjercicioMeToca();

    int m = 5;
    int n = 0;
    struct timeval ti, tf;  //Para medir el tiempo con precision
    double tiempo;


    array<string, tam*4> cadenasGeneradas = GeneradorCasos(m);  //Se crea un array capaz de guardar 10 cadenas y lo rellenamos llamando al generador de casos

    cout << endl;
    cout << "Resolucion por Divide y Venceras: " << endl;
    cout << endl;

    for(int i = 0; i < tam*4; i++) {
        n = cadenasGeneradas[i].size();
        gettimeofday(&ti, NULL); // Instante inicial
        pair<int,int> solucion = DyV(string(cadenasGeneradas[i]), 0, n-1, m);   //Se llama a la funcion principal, divide y venceras
        gettimeofday(&tf, NULL); // Instante final
        tiempo = ((tf.tv_sec - ti.tv_sec)*1000 + (tf.tv_usec - ti.tv_usec)/1000.0)/1000.0;  //Calculo de los segundos
        cout << "Posicion de inicio es " << solucion.second << " diferencia total maxima igual a " << solucion.first << endl;   //Se imprime el resultado
        cout << "Tamaño cadena: " << n << " " << "Tiempo en segundos: " << tiempo << endl;
        cout << endl;
    }

    cout << endl;
    cout << "Resolucion directa: " << endl;
    cout << endl;

    for(int i = 0; i < tam*4; i++) {
        n = cadenasGeneradas[i].size();
        pair<int,int> solucion2 = CasoBase(string(cadenasGeneradas[i]), 0, n-1, m); //Se llama a la funcion principal, divide y venceras
        cout << "Posicion de inicio es " << solucion2.second << " diferencia total maxima igual a " << solucion2.first << endl; //Se imprime el resultado
    }

    return 0;
}
```

Validación:

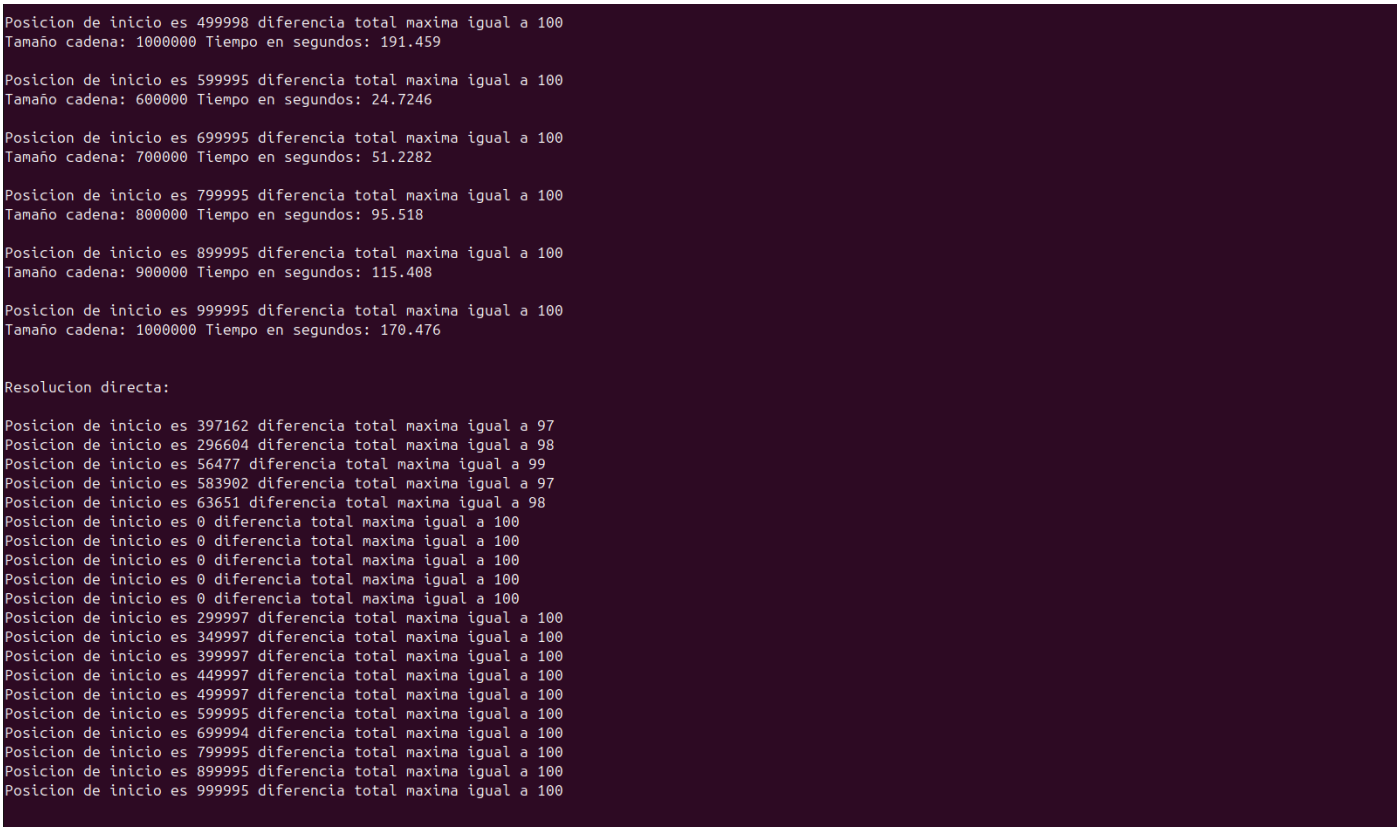
Hemos creado un algoritmo el cual genera cadenas del siguiente tamaño:

{600000, 700000, 800000, 900000, 1000000}

De esta manera evaluamos los tamaños más relevantes del rango establecido por el enunciado. El programa calcula el resultado usando DyV y utilizando resolución directa iterativa y muestra por pantalla los resultados los cuales deben coincidir.

Ejemplo de salida:

```
Se ha realizado el problema 4

Resolucion por Divide y Venceras:

Posicion de inicio es 397162 diferencia total maxima igual a 97
Tamaño cadena: 600000 Tiempo en segundos: 24.6228

Posicion de inicio es 296604 diferencia total maxima igual a 98
Tamaño cadena: 700000 Tiempo en segundos: 51.8921

Posicion de inicio es 56477 diferencia total maxima igual a 99
Tamaño cadena: 800000 Tiempo en segundos: 92.9312

Posicion de inicio es 583902 diferencia total maxima igual a 97
Tamaño cadena: 900000 Tiempo en segundos: 97.4589

Posicion de inicio es 63651 diferencia total maxima igual a 98
Tamaño cadena: 1000000 Tiempo en segundos: 159.524

Posicion de inicio es 0 diferencia total maxima igual a 100
Tamaño cadena: 600000 Tiempo en segundos: 24.3395

Posicion de inicio es 0 diferencia total maxima igual a 100
Tamaño cadena: 700000 Tiempo en segundos: 50.6253

Posicion de inicio es 0 diferencia total maxima igual a 100
Tamaño cadena: 800000 Tiempo en segundos: 92.1919

Posicion de inicio es 0 diferencia total maxima igual a 100
Tamaño cadena: 900000 Tiempo en segundos: 111.69

Posicion de inicio es 0 diferencia total maxima igual a 100
Tamaño cadena: 1000000 Tiempo en segundos: 176.63

Posicion de inicio es 299997 diferencia total maxima igual a 100
Tamaño cadena: 600000 Tiempo en segundos: 25.0795

Posicion de inicio es 349997 diferencia total maxima igual a 100
Tamaño cadena: 700000 Tiempo en segundos: 53.3217

Posicion de inicio es 399997 diferencia total maxima igual a 100
Tamaño cadena: 800000 Tiempo en segundos: 100.418

Posicion de inicio es 449997 diferencia total maxima igual a 100
Tamaño cadena: 900000 Tiempo en segundos: 122.697
```



Como se puede observar se ejecuta 4 veces el algoritmo para el mismo tamaño debido a que en cada uno utiliza una de las funciones distintas para generar cadenas. Estas son la función de cadenas aleatorias, la de solución al principio, solución al final y solución en el centro.

Estudio experimental:

Para realizar este estudio hemos utilizado 4 generadores distintos para realizar todos los casos interesantes. Hemos usado una función que genera cadenas totalmente aleatorias, una que pone la solución al principio de la cadena, otra que pone la solución en el medio de la cadena y otra que pone la solución al final de la cadena.

Al realizar el estudio para valores suficientemente grandes obtenemos estos resultados:

Tamaño - Tiempo(s)

Aleatorio:
600000 24.1068
700000 51.289
800000 99.8589
900000 121.501
1000000 191.852

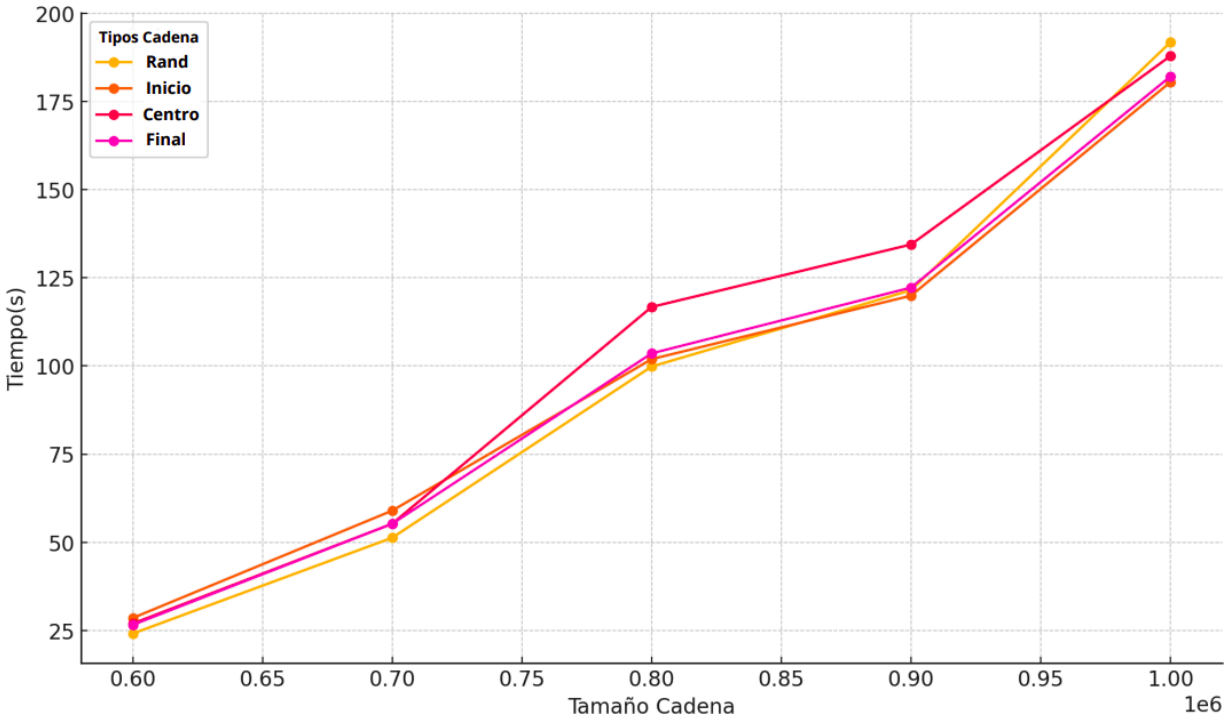
Solución inicio:
600000 28.5262
700000 58.9704
800000 101.951
900000 119.967
1000000 180.574

Solución medio:
600000 26.9462
700000 55.2534
800000 116.753
900000 134.478
1000000 187.952

Solución final:
600000 26.5704
700000 55.2401
800000 103.583
900000 122.256
1000000 182.155

Contraste:

Se puede apreciar que ambos resultados coinciden debido a que el crecimiento de las funciones prácticamente coinciden.
A continuación, se pueden ver las dos imágenes juntas para apreciar la similitud.



La primera imagen es el estudio experimental y la segunda el teórico. Se ve que el crecimiento de ambas es prácticamente el mismo por lo que podemos dar por hecho que los análisis han sido un éxito.

Conclusión:

El algoritmo de DyV es bastante malo en este caso ya que el iterativo tarda mucho menos o al menos eso hemos comprobado al pedirle al programa cadenas de un tamaño aproximado a 10^6. Sin embargo, en otros casos puede ser mucho más útil que un algoritmo iterativo.

Hemos tardado aproximadamente 30 horas (15 cada uno) entre las clases prácticas, tiempo autónomo del código y desarrollo de la memoria.