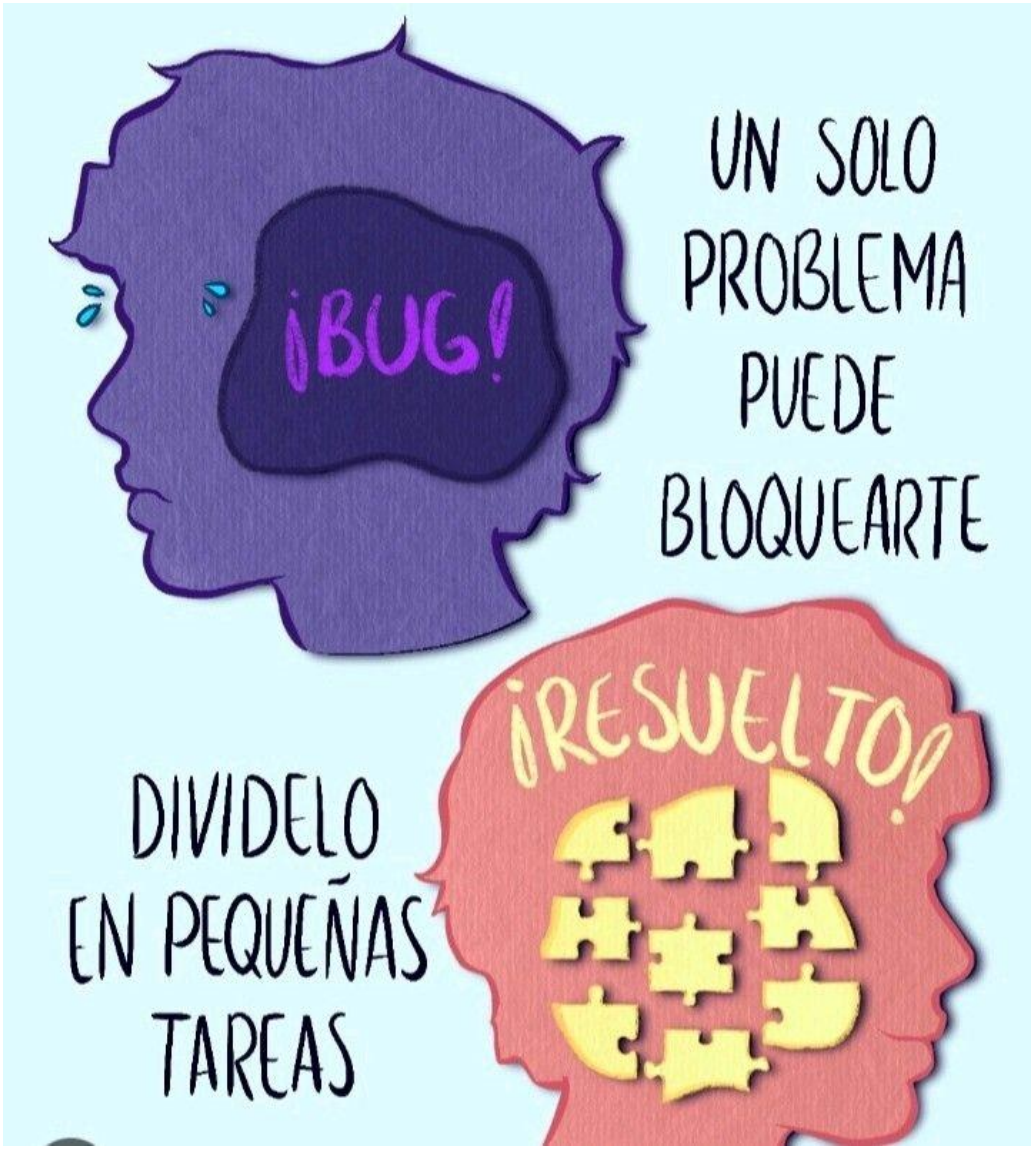
PRÁCTICA 1 ALGORITMOS Y ESTRUCTURAS DE DATOS II - DIVIDE Y VENCERAS



Subgrupo 3.3 – Facultad de Ingeniería Informática – Universidad de Murcia

Autores:

José Ángel García Marín – DNI: 49478872F

Juan Rondán Ramos – DNI: 48854653P

Profesor:

Francisco José Montoya Dato

Índice

I) Introducción y selección del ejercicio a resolver

A lo largo de este documento se dará respuesta a las preguntas planteadas en el enunciado de la práctica 1. Para ello se calculará el número de ejercicio asignado, en función de los DNI's de los integrantes, mediante la siguiente fórmula indicada en las instrucciones.

$$R = ((X+Y) \text{ módulo } 10) + 1 = (49478872+48854653) \text{ módulo } 10 + 1 = 6$$

Por tanto, nos corresponde el ejercicio número 6.

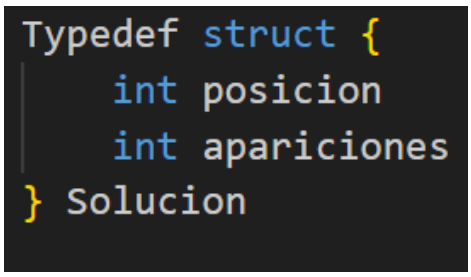
1. **Diseño de una solución divide y vencerás, incluyendo pseudocódigo y explicación del algoritmo, justificando las decisiones de diseño, las estructuras de datos y las funciones básicas del esquema algorítmico.**

Las funciones y estructura de datos que se ha implementado para realizar una solución Divide y Vencerás son las siguientes:

Estructura denominada Solucion para almacenar la solución de un subproblema.

- posicion almacena la posición de inicio de la mejor subcadena.

- apariciones almacena el número máximo de veces consecutivas que aparece el carácter C dentro de la subcadena.



Función solucionDirecta, encargada de encontrar la mejor subcadena de tamaño m de manera directa, sin recurrencia. Almacenan la posición de inicio de la cadena y el número máximo de caracteres C consecutivos.

Para ello se recorre la cadena de inicio a fin, contando cuántas veces aparece el carácter C de forma consecutiva. Si encuentra una mejor secuencia, actualiza la mejor posición y la cantidad máxima de apariciones, teniendo en cuenta que no se superen m apariciones consecutivas.

```
solucionDirecta(A: String; inicio, fin, m: entero; c: carácter) -> Solucion
    int mejorPosicion = inicio
    int maxApariciones = 0
    int apariciones = 0

    MIENTRAS (maxApariciones < m Y inicio <= fin) HACER:
        SI (A[inicio] == c) ENTONCES
            ++apariciones
            SI (apariciones > maxApariciones) ENTONCES
                maxApariciones = apariciones
                mejorPosicion = max(min(inicio - apariciones + 1, fin-m+1),0)
            FIN SI
        ELSE
            apariciones = 0
        FIN SI

        inicio++
    FIN MIENTRAS

    DEVOLVER (mejorPosicion, maxApariciones)
```

La función encontrarSubcadena, es la función principal recursiva en la que se implementa el algoritmo Divide y Vencerás para encontrar la mejor subcadena en A. Esta, divide la cadena en dos mitades y resuelve cada una de ellas recursivamente. Si alguna de ellas ya tiene m apariciones consecutivas de C, se devuelve. En caso contrario, combina ambas mitades para para encontrar la mejor solución. Cuando el tamaño del problema es menor que m, se usa la función solucionDirecta.

```
encontrarSubcadena(A: String; inicio, fin, m: entero; C: carácter) -> Solucion
    SI ((fin - inicio + 1) / 2 < m) ENTONCES
        DEVOLVER solucionDirecta(A, inicio, fin, m, C)
    FIN SI

    int medio = (inicio + fin) / 2
    Solucion izquierda = encontrarSubcadena(A, inicio, medio, m, C)
    Solucion derecha = encontrarSubcadena(A, medio + 1, fin, m, C)

    DEVOLVER combinar(izquierda, derecha, A, inicio, medio, fin, C, m)
```

La función combinar se encargada de fusionar las soluciones de la parte izquierda y derecha, seleccionando la mejor. Además, verifica si la mejor secuencia se encuentra en la frontera entre ambas mitades. Si es así, actualiza la mejor solución y la devuelve como resultado, asegurando una solución global óptima.

```
combinar(izq, der: Solucion; A: String; inicio, medio, fin: entero; C: carácter; m: entero) -> Solucion
    int mejorPosicion = izq.posicion
    int mejorApariciones = izq.apariciones

    SI (der.apariciones > mejorApariciones) ENTONCES
        mejorApariciones = der.apariciones
        mejorPosicion = der.posicion
    FIN SI

    int izquierdaMax = 0
    int derechaMax = 0

    PARA i = medio HASTA i >= inicio Y A[i] == C Y izquierdaMax < m HACER:
        izquierdaMax++
        i--
    FIN PARA
```

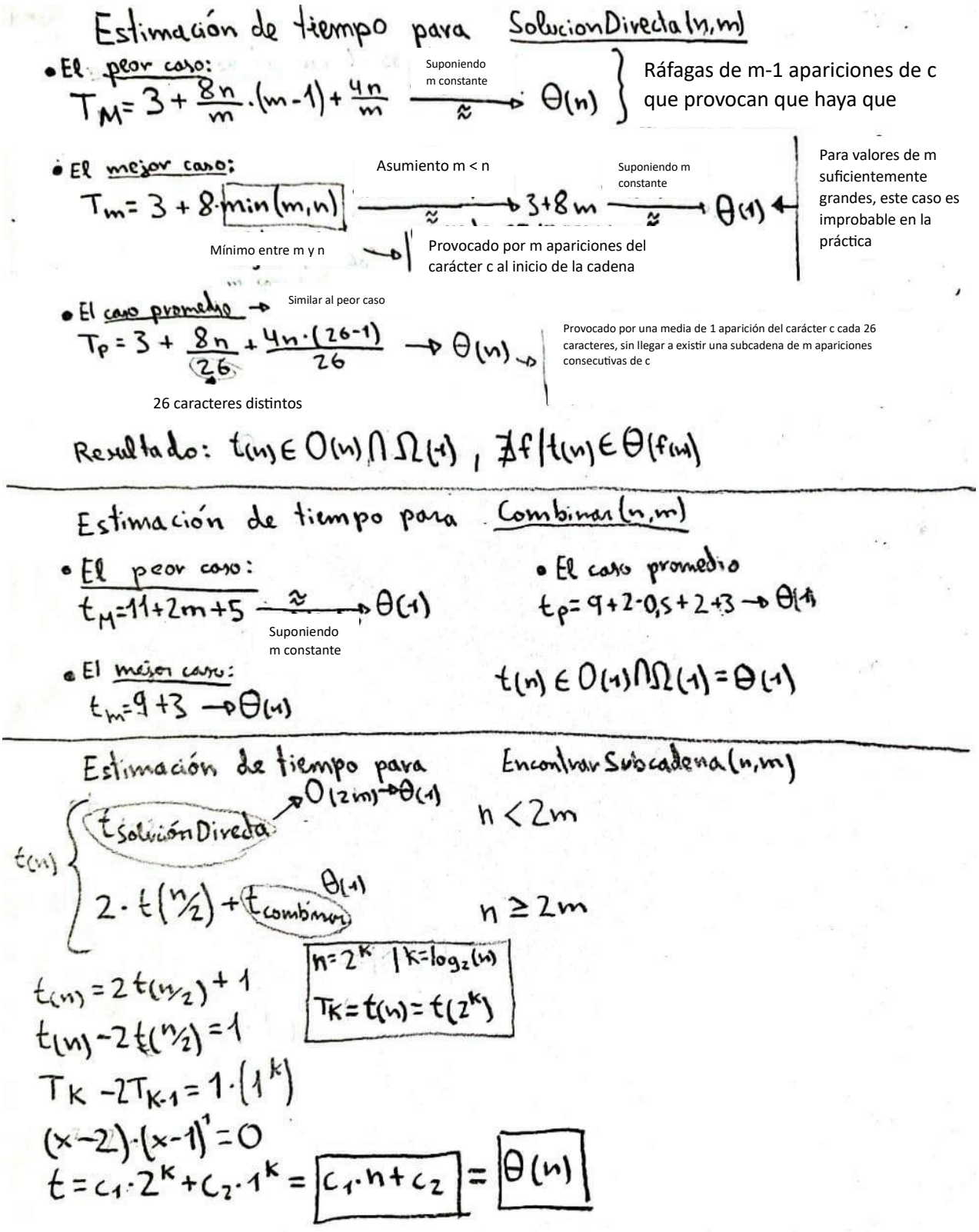
```
    PARA i = medio + 1 HASTA i <= fin Y A[i] == C Y (izquierdaMax + derechaMax) < m HACER:
        derechaMax++
        i++
    FIN PARA

    int fronteraApariciones = izquierdaMax + derechaMax

    SI (fronteraApariciones > mejorApariciones) ENTONCES
        mejorApariciones = fronteraApariciones
        mejorPosicion = medio - izquierdaMax + 1
    FIN SI

    DEVOLVER (mejorPosicion, mejorApariciones)
```

2. Análisis teórico del tiempo de ejecución, en los casos mejor y peor (tm y tM), y del orden de ejecución del algoritmo obtenido.



3. **Implementación: Programación del algoritmo (lo normal es hacer el programa tras haber diseñado y estudiado teóricamente el algoritmo). El código fuente debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente.**

   El algoritmo divide y vencerás del ejercicio 6 se ha implementado en C++. La documentación del algoritmo se ha incluido en el mismo fichero de C++ mediante diversos comentarios en los que se explica el funcionamiento del programa, así como de las diferentes funciones que se han implementado.

   En el código, se emplea un algoritmo de divide y vencerás para, dada una cadena de caracteres, encontrar una subcadena de tamaño m en la que se encuentren el máximo número de caracteres C consecutivos. Aquí, se encuentran las funciones solucionDirecta, combinar y encontrarSubcadena.

   Por otro lado, se han implementado dos funciones para imprimir los resultados: imprimir, que resalta en la propia cadena la subcadena resultado e imprimir_sin_remarcar, que hace exactamente lo mismo, pero no subraya la subcadena resultado, por si el programa se ejecuta en un equipo que no soporte el subrayado.

   Además, se ha creado un generador de cadenas aletorio con la función generarCadenaAleatoria, además de funciones para forzar tiempos mejores y peores con generarCadenaFavorable y generarCadenaDesfavorable.

   También se ha creado la función medirTiempo que mide el tiempo experimental del algoritmo con divide y vencerás y verifica el resultado producido usando como oráculo la soluciónDirecta (función sin aplicar divide y vencerás que sabemos que es correcta) . Cabe destacar que, con el objetivo de reducir la aleatoriedad del tiempo, se ejecuta varias veces el algoritmo y se divide el tiempo total de ejecución entre las veces que se ejecuta. Para medir el tiempo hemos utilizado la librería <sys/time.h>, cuyo uso se recomienda por parte del profesorado de la asignatura.

   Finalmente, con el fin de automatizar el análisis experimental, se ha desarrollado la función generarCSV, que ejecuta varios casos de prueba sin variar 'c' y 'm' (aunque m puede influir ligeramente en los tiempos no es determinante) y crea un fichero csv con los resultados obtenidos en el análisis experimental (usando ";" como delimitador y “.” para los números decimales”). Esta función llama a las la función medir tiempo con cadenas generadas automáticamente.

A continuación, se muestra en capturas de pantalla todo el código implementado:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

#include <fstream>

#include <stdio.h>
#include <sys/time.h>

#define TAM_MIN 99 //tamaño mínimo de la cadena generada
#define TAM_MAX 1050000 //tamaño máximo de la cadena generada
#define N_MUESTRAS 1000 //número de muestras para cada caso

//constantes para las mediciones de tiempo
#define M_PRUEBAS   100
#define C_PRUEBAS   'c'

#define S_CSV ';'// carácter usado como separador en el csv

using namespace std;

// Estructura para almacenar la solución de un subproblema
struct Solucion {
    int posicion;    // Índice de inicio de la subcadena
    int apariciones; // Número máximo de apariciones consecutivas de C
};
```

```cpp
//Imprime resultado subrayando en color la subcadena
void imprimir(Solucion resultado, const string& A, char c, int m) {
    cout << "Caso de prueba: A = \"" << A << "\", c = '" << c << "', m = " << m << endl;
    cout << "La subcadena comienza en el caracter " << resultado.posicion + 1 << " contando desde 1" << endl;
    cout << "Numero maximo de apariciones consecutivas de '" << c << "': " << resultado.apariciones << endl;
    for (size_t i = 0; i < A.size(); ++i) {
        if (i == static_cast<size_t>(resultado.posicion)) cout << "\033[4;96m";
        cout << A[i];
        if (i == static_cast<size_t>(resultado.posicion + m -1)) cout << "\033[0m";
    }
    cout << "\033[0m" << endl << endl;
}

//Imprime solución normal por si el equipo en el que se ejecuta no soporta el subrayado
void imprime_sin_remarcar(Solucion resultado, const string& A, char c, int m) {
    cout << "Caso de prueba: A = \"" << A << "\", c = '" << c << "', m = " << m << endl;
    cout << "La subcadena comienza en el caracter " << resultado.posicion + 1 << " contando desde 1" << endl;
    cout << "Numero maximo de apariciones consecutivas de '" << c << "': " << resultado.apariciones << endl << endl;

}
```

```cpp
// Función para calcular la mejor subcadena de tamaño m de manera directa
Solucion solucionDirecta(const string& A, int inicio, int fin, int m, char c) {
    int mejorPosicion = inicio;
    int maxApariciones = 0;
    int apariciones = 0;
    while (maxApariciones < m && inicio <= fin) {
        if (A[inicio] == c) {
            ++apariciones;
            if(apariciones > maxApariciones) {
                maxApariciones = apariciones;
                // Encontramos la posicion de inicio de una cadena de tamaño m
                // (o menor si la cadena tiene tamaño menor) que contiene a la subsecuencia más larga del caracter c
                mejorPosicion = max(min(inicio - apariciones + 1, fin-m+1),0);
            }
        } else {
            apariciones = 0;
        }
        inicio++;
    }

    return {mejorPosicion, maxApariciones};
}
```

```cpp
// Función que combina las soluciones de los subproblemas contando C consecutivos en la frontera
Solucion combinar(const Solucion& izq, const Solucion& der, const string& A, int inicio, int medio, int fin, char C, int m) {
    int mejorPosicion = izq.posicion;
    int mejorApariciones = izq.apariciones;

    if (der.apariciones > mejorApariciones) {
        mejorApariciones = der.apariciones;
        mejorPosicion = der.posicion;
    }

    // Contar C consecutivos en la frontera
    int izquierdaMax = 0;
    int derechaMax = 0;

    // Contamos caracteres C en la parte izquierda desde la mitad hacia atrás
    for (int i = medio; i >= inicio && A[i] == C && izquierdaMax < m; --i) {
        izquierdaMax++;
    }

    // Contamos caracteres C en la parte derecha desde la mitad hacia adelante
    for (int i = medio + 1; i <= fin && A[i] == C && (izquierdaMax+derechaMax) < m; ++i) {
        derechaMax++;
    }

    int fronteraApariciones = izquierdaMax + derechaMax;
```

```
    // Comprobar si la mejor solución está en la frontera
    if (fronteraApariciones > mejorApariciones) {
        mejorApariciones = fronteraApariciones;
        mejorPosicion = medio - izquierdaMax + 1;
    }
    return {mejorPosicion, mejorApariciones};
}

// Función recursiva para aplicar Divide y Vencerás
Solucion encontrarSubcadena(const string& A, int inicio, int fin, int m, char C) {
    if ((fin - inicio + 1)/2 < m) {
        return solucionDirecta(A, inicio, fin, m, C);
    }

    int medio = (inicio + fin) / 2;
    Solucion izquierda = encontrarSubcadena(A, inicio, medio, m, C);
    Solucion derecha = encontrarSubcadena(A, medio + 1, fin, m, C);

    return combinar(izquierda, derecha, A, inicio, medio, fin, C, m);
}
```

```
// Genera una cadena aleatoria de tamaño n con caracteres en [a, b],
// así simulamos el caso promedio, que tendrá un tiempo de ejecución similar al caso mejor
string generarCadenaAleatoria(int n, char a, char b) {
    string cadena;
    for (int i = 0; i < n; ++i) {
        cadena += a + rand() % (b - a + 1);
    }
    return cadena;
}

// Encontrar caracter diferente a C
char getCaracterDistinto(char C) {
    char a = 'a';
    if (C == a) {
        a++;
    }
    return a;
}

// Genera una cadena con ningun caracter c, esto hace que la funcion combinar tenga coste mínimo
string generarCadenaFavorable(int n, int m, char C) {
    char caracterDiferente = getCaracterDistinto(C);
    string cadena(n, caracterDiferente);
    return cadena;
}

// Genera una cadena con solo el carácter C, esto provoca que el tiempo de combinar se máximo
string generarCadenaDesfavorable(int n, int m, char C) {
    string cadena(n, C);
    return cadena;
}
```

```cpp
// Funcion para calcular el tiempo promedio de un caso usando n muestras y verificar que el resultado sea correcto.
// Para vefícar el resultado usamos la solucion sin divide y vencerás como oráculo
// Siempre se usan m y C constantes para simplificar el análisis, aunque m puede influir en el tiempo de ejecucion
double medirTiempo(const string& A) {
    struct timeval ti, tf;
    double tiempo_promedio;
    Solucion resultadoDyV;
    gettimeofday(&ti, NULL);
    for (int i = 0; i < N_MUESTRAS; i++) {
        resultadoDyV = encontrarSubcadena(A, 0, A.size() - 1, M_PRUEBAS, C_PRUEBAS);
    }
    gettimeofday(&tf, NULL); // Instante final

    Solucion resultadoEsperado = solucionDirecta(A, 0, A.size() - 1, M_PRUEBAS, C_PRUEBAS); // usamos como oráculo la funcion simple
    if ((resultadoDyV.posicion != resultadoEsperado.posicion) || (resultadoDyV.apariciones != resultadoEsperado.apariciones)) { //verificamos
        cout << "Error: El resultado del siguiente caso de estudio es incorrecto." << endl;
        cout << "Solución obtenida:" << endl;
        imprimir(resultadoDyV, A, C_PRUEBAS, M_PRUEBAS);
        cout << "Solución esperada:" << endl;
        imprimir(resultadoEsperado, A, C_PRUEBAS, M_PRUEBAS);
        tiempo_promedio = -1.0;
    } else {
        tiempo_promedio = ((tf.tv_sec - ti.tv_sec)*1000 + (tf.tv_usec - ti.tv_usec)/1000.0)/static_cast<double>(N_MUESTRAS);
    }

    return tiempo_promedio;
}
```

```cpp
// Funcion para generar un csv con las medias de las muestras de tiempo para cada caso
void generarCSV(const std::string& nombreArchivo) {
    // Crear y abrir el archivo
    ofstream archivo(nombreArchivo);

    if (!archivo.is_open()) {
        cerr << "Error al crear el archivo csv" << endl;
        return;
    }

    // Escribir encabezados
    archivo << "TamanoCadena" << S_CSV << "T promedio mejor caso(ms)" << S_CSV << "T promedio caso promedio(ms)" << S_CSV << "T promedio peor caso(ms)" << endl;

    // Escribir datos
    double t_mejor;
    double t_promedio;
    double t_peor;
    for (int i = TAM_MIN; i<TAM_MAX; i*=2) {
        t_mejor = medirTiempo(generarCadenaFavorable(i, M_PRUEBAS, C_PRUEBAS));
        t_promedio = medirTiempo(generarCadenaAleatoria(i,'a','b'));
        t_peor = medirTiempo(generarCadenaDesfavorable(i, M_PRUEBAS, C_PRUEBAS));
        archivo << i << S_CSV << t_mejor << S_CSV << t_promedio << S_CSV << t_peor << endl;

        if (t_mejor < 0 || t_promedio < 0 || t_peor < 0) {   //asegurar que los resultados son correctos
            cerr << "Se han detectado inconsistencias en las muestras de los tiempos" << endl;
            archivo.close();
            return;
        }
    }

    archivo.close();
    cout << "Archivo CSV generado correctamente: " << nombreArchivo << endl;
}
```

```cpp
int main() {

    char c = 'c';
    int m;
    string A;
    // Pruebas manuales
    // testing unitario de caja negra con clases de equivalencia
    // clase de equivalencia 1 : m > longitud de cadena
    m = 100; A = "hcc";
    Solucion resultado1 = encontrarSubcadena(A, 0, A.size() - 1, m, c);
    imprimir(resultado1, A, c, m);
    // clase de equivalencia 2 : m << longitud de cadena
    m = 3; A = "ckcckkccckkkccccckcckkccckkkcccc";
    Solucion resultado2 = encontrarSubcadena(A, 0, A.size() - 1, m, c);
    imprimir(resultado2, A, c, m);
    // clase de equivalencia 3 : m = longitud cadena y C no aparece
    m = 6; A = "ultima";
    Solucion resultado3 = encontrarSubcadena(A, 0, A.size() - 1, m, c);
    imprimir(resultado3, A, c, m);

    // finalmente verificamos los resultados de casos aleatorios a la vez que medimos sus tiempos
    generarCSV("analisisExperimental.csv");
}
```

4. **Diseño y aplicación de un proceso de validación del algoritmo dyv implementado utilizando el método de resolución directa. Hay que indicar claramente los experimentos concretos realizados para asegurarnos de que el programa funciona correctamente, y en su caso programas utilizados para la validación.**

   Para comprobar el funcionamiento del algoritmo, lo primero que se hizo fue introducir el caso de prueba otorgado por el enunciado del problema, es decir:

   A = “c d d a b c d a c c” m = 5 C = ‘c’

   Donde se obtiene que la subcadena empieza en el carácter 6 y el máximo número de apariciones del carácter ‘c’ es 2.

   Tras comprobar que el resultado era el esperado, se introdujeron más casos de prueba, tanto casos estándar de pequeño y gran tamaño, como casos que podrían dar problemas, como cadenas cortas con un tamaño menor a m, cadenas donde el máximo número de caracteres C era mayor a m, cadenas con varias subcadenas solución posibles… Asegurándonos de que en todos los casos el resultado era el esperado. Algunos de estos casos son:

   A = "sccdnsccccndsjecccccccsdddc" m = 12 C = ‘c’

   Donde se obtiene que la subcadena comienza en el carácter 16 y el máximo número de apariciones del carácter ‘c’ es 7.

   A = “a c c d” m = 7 C = ‘c’

   Donde se obtiene que la subcadena comienza en el carácter 1 y el máximo número de apariciones del carácter ‘c’ es 2.

   A = “c c c a s c c c c c c f d c c” m = 4 C = ‘c’

   Donde se obtiene que la subcadena comienza en el carácter 6 y el máximo número de apariciones del carácter ‘c’ es 4.

   A = “d e c c c j k e c c c d s” m = 3 C = ‘c’

   Donde se obtiene que la subcadena comienza en el carácter 3 y el máximo número de apariciones es 3.

Tras haber realizado varias pruebas en casos estándar y en casos que podría haber controversia, se realizaron algunas pruebas de caja negra por clases de equivalencia

(incluyendo en el main actual una prueba por clase de equivalencia) y verificando todos los casos de prueba generados automáticamente usando como oráculo el método para de solución directa. Tras esto concluimos que el algoritmo funciona correctamente (aunque la ausencia de errores en un testing no garantiza que no los haya), por lo que se paso a realizar el estudio experimental del tiempo de ejecución con distintos casos de entrada.

5. **Estudio experimental del tiempo de ejecución para distintos tamaños de problema. Habrá que experimentar con tamaños suficientemente grandes para obtener resultados significativos (10^2 <= n <= 10^6). Será necesaria para ello la implementación de un generador de casos de prueba, que genere entradas para el programa para los casos más favorable, más desfavorable y promedio.**

Haciendo uso de la función generarCSV, (método cuya función e implementación se ha explicado anteriormente), se genera un csv con los tiempos de ejecución de nuestro algoritmo con casos de prueba generados automáticamente.

En la siguiente tabla se muestra los tiempos promedio para el mejor caso, caso promedio y peor caso, en función del tamaño de la cadena de entrada. Estos valores se han generado tomando para cada cadena generada 1000 muestras, con m = 100. Se han tomado como tamaños de cadena valores desde 100 hasta 3276800. Para ello, se ha duplicado el valor inicial hasta alcanzar el valor final, generando 3 cadenas por cada tamaño, una para cada caso.

Para generar una cadena para el mejor caso, se crea una cadena que contenga n veces un carácter diferente a c (debido a que en este caso la función Combinar no solo tiene un orden constante, sino que además no depende ni siquiera de m, ejecutando como máximo una vez cada línea de código de la función).

Para el caso promedio, se crea una cadena aleatoria con los caracteres del alfabeto indicado en el enunciado de la práctica.

Para el mejor caso, se crea una cadena que contenga n veces el carácter c (debido a que en este caso la función Combinar pertenece al orden de m, que suponemos como constante en el estudio).

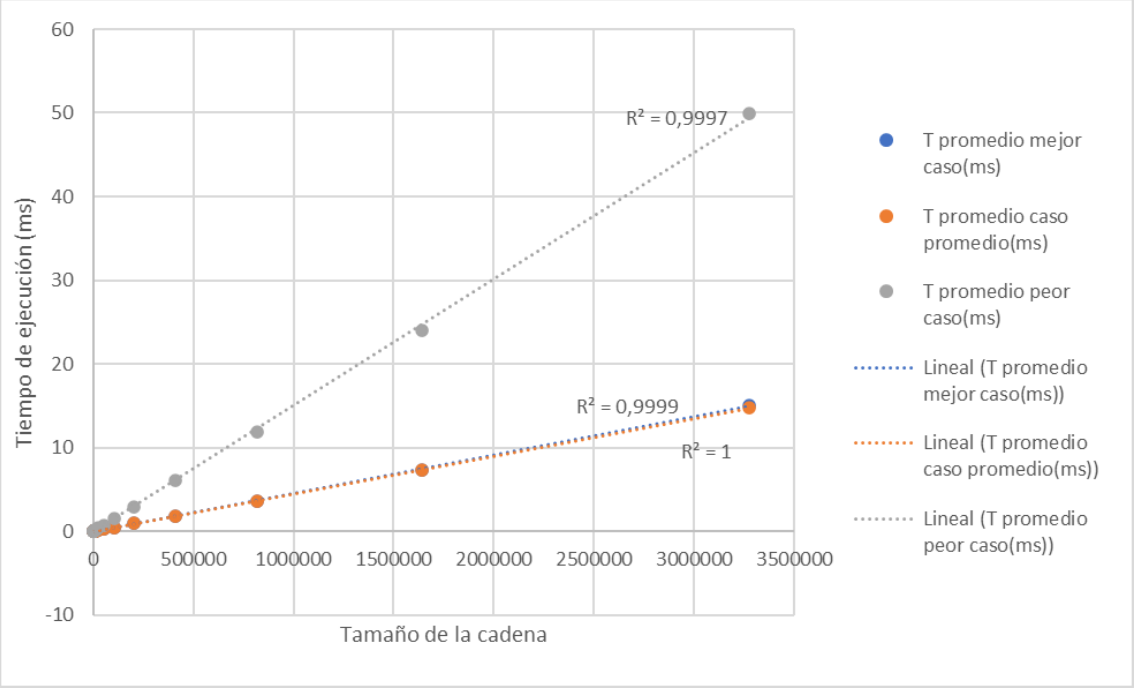
Tamaño Cadena	T promedio mejor caso(ms)	T promedio caso promedio(ms)	T promedio peor caso(ms)
100	0,000425	0,000465	0,000992
200	0,001269	0,000859	0,002452
400	0,002614	0,001826	0,005365
800	0,008312	0,003622	0,016763
1600	0,007035	0,00772	0,022745
3200	0,014677	0,01427	0,046006
6400	0,028084	0,028563	0,091642
12800	0,05665	0,05693	0,186409
25600	0,113345	0,113558	0,368669
51200	0,22805	0,249108	0,736493
102400	0,465328	0,459178	1,52465
204800	0,964401	0,916798	2,97376
409600	1,82908	1,8572	6,09636
819200	3,67098	3,64514	11,9
1638400	7,37297	7,32815	24,0731
3276800	15,0119	14,7111	49,8842

Por otro lado, en la siguiente figura se muestra gráficamente el aumento del tiempo de ejecución para cada caso en función del tamaño de la cadena de entrada.

Viendo la gráfica podemos realizar una serie de observaciones:

- En primer lugar, observamos que los resultados se adaptan perfectamente a una regresión lineal (Con R2 extremadamente cercanos a 1), lo que indica que el tiempo de ejecución de nuestro algoritmo pertenece a O(n) y, por lo tanto, presenta una evolución lineal.

- En segundo lugar, puede resultar curioso que el tiempo mejor sea tan parecido a tiempo promedio (e incluso darse el caso de que el tiempo promedio sea ligeramente inferior al tiempo mejor). No obstante, esto es completamente lo que esperábamos, pues el tiempo peor, que es producido por una cadena solo con el carácter c (debido a que en este caso la función Combinar pertenece al orden de m, que suponemos como constante en el estudio) es muy improbable en la práctica. Mientras tanto, el tiempo promedio será muy cercano al tiempo mejor, puesto que es improbable que se den muchas apariciones de c seguidas. Con respecto a que el tiempo promedio supere ligeramente al tiempo mejor en algunos casos es algo completamente normal puesto que el tiempo de ejecución real depende de muchos factores externos al algoritmo, siendo variaciones poco significativas que no son relevantes para nuestro estudio.



6. **Contraste del estudio teórico y el experimental, buscando justificación a las posibles discrepancias entre los dos estudios.**

Contrastando el estudio teórico con el análisis experimental, llegamos a la conclusión de que los resultados obtenidos en el análisis experimental se corresponden a los del análisis teórico realizado previamente a la implementación del algoritmo. Como hemos mencionado en el apartado anterior, el tiempo de ejecución de nuestro algoritmo es lineal, por lo que se ajusta al análisis teórico realizado como paso previo a la implementación.

7. **Conclusiones y valoraciones personales de la actividad, con una estimación del tiempo que se ha tardado en completarla (en horas, por cada miembro).**

   Con esta práctica hemos terminado de entender bien cómo funciona la técnica de resolución de problemas Divide y Vencerás, enfrentándonos a un problema en el que, aunque pensamos que hubiera sido una mejor elección una solución en la que únicamente se recorra la cadena de una pasada, hemos conseguido resolver de forma adecuada aplicando la técnica.

   Los aspectos de la práctica que más tiempo nos han llevado han sido sin duda, además de la propia elaboración del código, que se ha tenido que ir actualizando de forma constante para cubrir los casos más extremos de posibles cadenas de entrada, el estudio experimental. En este caso nos surgían problemas a la hora de medir el tiempo de ejecución, puesto que, la librería sys/time.h nos daba problemas en un inicio, ya que para cualquier cadena daba como resultado de tiempo 0. También tratamos de trabajar con otra librería para medir el tiempo, chrono, pero al ver que el resultado era el mismo decidimos volver a optar por sys/time.h, donde, tras hacer un par de ajustes, comenzó a funcionar.

   En resumen, está práctica nos ha llevado más tiempo del esperado (alrededor de 22 horas trabajando en conjunto, tanto presencial como en telemático), ya que nos ha surgido más de un contratiempo, pero hemos sido capaces de enfrentar estas adversidades y realizar un programa fiable y muy completo, por lo que estamos realmente contentos.