**MEMORIA**PRÁCTICA DE DIVIDE Y VENCERÁS

**Profesor:** Francisco José Montoya Dato

**Nombre:** David Sergio Anishchenko Halkina   
**E-mail:** [ds.anishchenkohalkina@um.es](mailto:ds.anishchenkohalkina@um.es)  
**Grupo:** 2.1

**Nombre:** Irene Beatriz López Jiménez   
**E-mail:** [ibeatriz.lopez@um.es](mailto:ibeatriz.lopez@um.es)   
**Grupo:** 2.1

**TABLA DE CONTENIDO**

[1. Introducción 2](#_Toc193908446)

[2. Diseño algoritmo 2](#_Toc193908447)

[3. Análisis teórico 3](#_Toc193908448)

[4. Implementación y documentación 3](#_Toc193908449)

[5. Validación 5](#_Toc193908450)

[6. Estudio experimental 5](#_Toc193908451)

[7. Contraste entre el análisis teórico y experimental 5](#_Toc193908452)

[8. Conclusiones 5](#_Toc193908453)

# Introducción

En esta memoria, se van a documentar los diferentes pasos que se han seguido para desarrollar y validar un algoritmo basado en divide y vencerás que soluciona el problema con el siguiente enunciado:

*(2) Dadas dos cadenas A y B de la misma longitud n, y un natural m ≤ n, encontrar C, la subcadena de A de tamaño m con más diferencia total en valor absoluto entre los caracteres en cada posición de A y B (suma de los m valores ⎪A[i]-B[i]⎪ entre las posiciones incluidas en C). Devolver el índice p de comienzo de la solución y el valor de la mayor diferencia total. En caso de empate, será válida cualquiera de las soluciones óptimas.*

*Ejemplo: n=10, m=5  
A= c d d a b c d a c c   
B= c a c d d b c a d c   
dife: 0 3 1 3 2 1 1 0 1 0*

*Solución: C, posición de inicio igual a 2, y diferencia total igual a 10.*

Además, se realizará un estudio experimental, con el objetivo de compararlo con el análisis teórico que se habrá llevado a cabo con anterioridad.

# Diseño algoritmo

Una forma de resolver este problema por divide y vencerás sería dividir el array de diferencias en dos mitades recursivamente hasta llegar al caso base, donde el tamaño de los subproblemas es menor que el de la subcadena buscada (o que el. Llegados a ese punto, teniendo en cuenta la posición de inicio y de fin de las subsoluciones, realizamos la búsqueda de las posibles soluciones que se encuentren entre ambas mitades a través de una ventana deslizante de tamaño .

Un ejemplo de cómo funcionaría el algoritmo para y sería el siguiente:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | 2 | 5 | 7 | 1 | 2 | 1 |

Este es el array de diferencias inicial.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | 2 | 5 | 7 | 1 | 2 | 1 |

Se divide en dos mitades.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | 2 | 5 | 7 | 1 | 2 | 1 |

Como al dividirlo, quedarían subproblemas de tamaño , se procede a buscar la solución de tamaño de cada mitad (coloreadas de azul y naranja).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 3 | **2** | **5** | **7** | 1 | 2 | 1 |

Al volver al problema original, sabemos que la solución final está entre los índices 1 y 6 (empezando a contar desde 0), por lo que descartamos el primer y último elemento del array e, igual que antes, buscamos una solución entre dichas posiciones con una ventana deslizante (resaltado en negrita).

Este algoritmo, implementado en pseudocódigo, quedaría de la siguiente manera:

DivideYVencerás(dif: array [1..n] de entero; inicio, fin, m: entero)

ALGORITMO

medio := (inicio + fin) / 2;

s.suma := 0;

s.indice := inicio;

p1.suma := 0;

p1.indice := inicio;

p2.suma := 0;

p2.indice := medio;  
SI medio – inicio > m ENTONCES  
 p1 := DivideYVencerás(dif, inicio, medio, m);

p2 := DivideYVencerás(dif, medio, fin, m);   
FIN\_SI  
DEVOLVER Combinar (dif, p1, p2, fin, m);  
FIN\_ALGORITMO

Por otro lado, la resolución cuando se llega al caso base se basa en tomar una ventana de tamaño para ir desplazándola por el array de diferencias e ir guardando aquella cuya suma sea máxima.

Combinar (dif: array [1..n] de entero; p1, p2: Solucion; fin, m: entero)

ALGORITMO  
aux := 0;

PARA i DESDE p1.indice HASTA p1.indice + m HACER

aux := aux + dif[i];

FIN\_PARA

s.suma := aux;

s.indice := p1.indice;

i := p1.indice;

MIENTRAS i < MINIMO (p2.indice, fin - m) HACER

aux := aux – dif[i] + dif[i + m];

SI aux > s.suma ENTONCES

s.suma := aux;

s.indice := i + 1;

FIN\_SI

i++;

FIN\_MIENTRAS  
DEVOLVER s;  
FIN\_ALGORITMO

Además, con el objetivo de facilitar el retorno de la solución de los algoritmos diseñados, conviene declarar una estructura de datos donde se vayan almacenando el índice donde comienza la subcadena pedida y la suma total de las diferencias.

# Análisis teórico

# Implementación y documentación

Tras diseñar un algoritmo en pseudocódigo para resolver el problema aplicando divide y vencerás, y analizar su complejidad teórica, lo implementamos en C++:

1. #include <iostream>

2. #include <time.h>

3. using namespace std;

4.

5. // Declaramos una estructura de datos para almacenar el índice y la suma

6. // de la solución calculada.

7. struct Solucion {

8. int indice;

9. int suma;

10. };

11.

12. // Esta función aplica una ventana deslizante sobre el array de enteros

13. // 'dif' con el objetivo de buscar aquella ventana de tamaño 'm' que sume

14. // la mayor diferencia, es decir, para buscar una solución al problema dado.

15. Solucion combinar(int\* dif, Solucion p1, Solucion p2, int fin, int m) {

16. // Declaramos el entero 'aux', que irá almacenando los resultados

17. // intermedios de la ventana deslizante.

18. int aux = 0;

19.

20. // Calculamos la primera ventana.

21. for (int i = p1.indice; i < p1.indice + m; i++)

22. aux += dif[i];

23.

24. // Declaramos la variable de tipo Solucion 's', que irá almacenando

25. // información acerca de la ventanas con el campo 'suma' máximo, y

26. // lo inicializamos con el valor de 'aux' y el índice del primer

27. // subproblema

28. Solucion s;

29. s.suma = aux;

30. s.indice = p1.indice;

31.

32. // Declaramos e inicializamos el índice 'i' que utilizaremos para

33. // recorrer el array 'dif'.

34. int i = p1.indice;

35.

36. // Se desplaza la ventana deslizante por 'dif', restando su primer

37. // elemento y sumando el siguiente.

38. while (i < min(p2.indice, fin - m)) {

39. aux += dif[i + m] - dif[i];

40.

41. // Se va guardando el máximo en 's' y se actualiza su índice.

42. if (aux > s.suma) {

43. s.suma = aux;

44. s.indice = i + 1;

45. }

46. i++;

47. }

48. // Devolvemos la solución del problema.

49. return s;

50. }

51.

52. Solucion dyv(int\* dif, int inicio, int fin, int m) {

53. // Declaramos y calculamos el punto medio del subproblema.

54. int medio = (inicio + fin) / 2;

55.

56. // Declaramos las variables de tipo Solucion 's', 'p1' y 'p2', que

57. // utilizaremos para guardar la solución del problema, del subproblema

58. // derecho y del izquierdo, respectivamente.

59. Solucion s, p1, p2;

60. s.suma = 0;

61. s.indice = inicio;

62. p1.suma = 0;

63. p1.indice = inicio;

64. p2.suma = 0;

65. p2.indice = medio;

66.

67. // Se divide el problema en dos subproblemas si sus tamaños son mayores al

68. // de la subcadena buscada (con el fin de evitar dividir innecesariamente,

69. // pues luego se llamará a la función 'combinar' que buscará las ventanas

70. // de tamaño 'm' con suma máxima).

71. if (medio - inicio > m) {

72. p1 = dyv(dif, inicio, medio, m);

73. p2 = dyv(dif, medio, fin, m);

74. }

75.

76. // Combinamos las soluciones y devolvemos el resultado.

77. s = combinar(dif, p1, p2, fin, m);

78. return s;

79. }

80.

81. int main() {

82. // Declaramos las variables 'n' y 'm', y ejecutamos el programa

83. // mientras se puedan introducir por la consola.

84. int n, m;

85. while (cin >> n >> m) {

86. // Declaramos las cadenas 'A' y 'B', y las introducimos.

87. char A[n], B[n];

88. for (int i = 0; i < n; i++)

89. cin >> A[i];

90. for (int i = 0; i < n; i++)

91. cin >> B[i];

92.

93. // Declaramos el array de enteros 'dif' y almacenamos la diferencia

94. // entre las cadenas 'A' y 'B'.

95. int dif[n];

96. for (int i = 0; i < n; i++)

97. dif[i] = abs(A[i] - B[i]);

98.

99. // Llamamos a la función que aplica divide y vencerás, e imprimimos el

100. // resultado por la terminal.

101. Solucion s = dyv(dif, 0, n, m);

102. cout << s.indice + 1 << " " << s.suma << endl;

103.  }

104. }

# Validación

# Estudio experimental

1. void generadorCasosPromedio(char A[], char B[], int n) {

2. for (int i = 0; i < n; i++) {

3. A[i] = rand() % (26) + 'a';

4. B[i] = rand() % (26) + 'a';

5. }

6. }

1. void generadorCasosPeor(char A[], char B[], int n, int m) {

2. for (int i = 0; i < n; i++) {

3. if (i >= m / 2 || i <= n - m / 2) {

4. A[i] = 'a';

5. B[i] = 'a';

6. } else {

7. A[i] = rand() % (26) + 'a';

8. B[i] = rand() % (26) + 'a';

9. }

10. }

11. }

1. void generadorCasosMejor(char A[], char B[], int n, int m) {

2. for (int i = 0; i < n; i++) {

3. if (i < (n - m) / 2 || i >= (n + m) / 2) {

4. A[i] = 'a';

5. B[i] = 'a';

6. } else {

7. A[i] = rand() % (26) + 'a';

8. B[i] = rand() % (26) + 'a';

9. }

10. }

11. }

# Contraste entre el análisis teórico y experimental

# Conclusiones

A modo de conclusión, hemos de decir que esta práctica nos ha resultado bastante interesante, porque hemos podido estudiar más a fondo la eficiencia de los algoritmos impartidos en clases de teoría, en lugar de centrarnos únicamente en su desarrollo e implementación.

Por otro lado, nos parece que ha sido un acierto no dejar atrás los conocimientos de análisis de algoritmos del tema 1, pues, desde nuestro punto de vista, es un aspecto imprescindible que debe dominar un informático a la hora de desarrollar y optimizar cualquier tipo de código (ya que, por más que un programa realice su función correctamente, si al emplear entradas más grandes no es eficiente, no tiene mucha utilidad).

En general, para la realización de este proyecto, hemos empleado alrededor de 15-20 horas, de las cuales la mayoría (10-15 horas) han ido destinadas al diseño algorítmico y a la depuración del código.