Algorítmica

Curso 2023-2024

Grupo Viterbi

PRÁCTICA 2-DIVIDE Y VENCERÁS

Integrantes:

Miguel Ángel De la Vega Rodríguez	miguevrod@correo.ugr.es
Alberto De la Vera Sánchez	joaquin724@correo.ugr.es
Joaquín Avilés De la Fuente	adelaveras01@correo.ugr.es
Manuel Gomez Rubio	e.manuelgmez@go.ugr.es
Pablo Linari Perez	e.pablolinari@go.ugr.es

Facultad de Ciencias UGR
Escuela Técnica Ingeniería Informática UGR
Granada
2023-2024

Índice general

1 Autores **3**

2 Objetivos **4**

3 Definicion Problema **5**

4 Algoritmo Especifico **6**
- 4.1 Problema 1: Subsecuencia de suma máxima. 6
  - 4.1.1 Estudio teórico 6
  - 4.1.2 Estudio empírico 7
  - 4.1.3 Estudio híbrido 7
- 4.2 Problema 2: Enlosar un espacio 8

5 Algoritmo Divide y Vencerás **9**
- 5.1 Problema 1: Subsecuencia de suma máxima. 9
  - 5.1.1 Estudio teórico 9
  - 5.1.2 Estudio empírico 10
  - 5.1.3 Estudio híbrido 11
- 5.2 Problema 2: Enlosar un espacio 11
- 5.3 Problema 3: Problema del viajante de comercio. 11
  - 5.3.1 Estudio Empírico 11

6 Conclusiones **13**

Apartado 1

Autores

- **Miguel Ángel De la Vega Rodríguez:** 20%
  - Plantilla y estructura del documento LaTeX
  - Programación Viajante
  - Programación SumaMax (DyV)
  - Tests de eficiencia
- **Joaquín Avilés De la Fuente:** 20%
  - Programacion SumaMax (DyV)
  - Tests de eficiencia
- **Alberto De la Vera Sánchez:** 20%
  - Redacción LaTeX
  - Graficas y ajustes
- **Manuel Gomez Rubio** 20%
  - Programacion SumaMax (Kadane)
  - Programacion Losetas
- **Pablo Linari Pérez:** 20%
  - Programacion SumaMax (Kadane)
  - Programacion Losetas

Apartado 2

Objetivos

En esta práctica, se pretende resolver problemas de forma eficiente aplicando la técnica de Divide y Vencerás. Para ello, se han planteado varios problemas cuya solución es conocida (excepto para el problema del viajante), y se han implementado algoritmos que los resuelven mediante el método convencional y mediante la técnica de Divide y Vencerás. Posteriormente, se ha buscado un umbral en el cual ambos tengan el mismo tiempo de ejecución, finalmente, se ha buscado el umbral óptimo para cada problema.

Apartado 3

Definicion Problema

Apartado 4

Algoritmo Especifico

En este apartado, estudiaremos la eficiencia teórica, empírica e híbrida de los algoritmos especificos de cada uno de los problemas.

4.1 Problema 1: Subsecuencia de suma máxima.

Para el primer problema, el algoritmo específico que empleamos es el algoritmo de Kadene.

4.1.1 Estudio teórico

```
1     int kadane(int *a, int size){
2         int max_global = a[0];
3         int max_current = a[0];
4 
5         for (int i = 1; i < size; i++) {
6             max_current = max(a[i], max_current + a[i]);
7             if (max_current > max_global) {
8                 max_global = max_current;
9             }
10        }
11        return max_global;
12    }
```

Como podemos observar la eficiencia del codigo en las líneas 6-8, tienen eficiencia O(1). Por tanto, su tiempo de ejecución es constante y notaremos por a. Luego, el bucle for se ejecutará $(size-1)-i+1$ veces, es decir, $size-i$ veces. Sabiendo que el resto de líneas del código tienen eficienciaa O(1), tenemos el siguiente resultado

$$\sum_{i=inicial}^{size-1} a$$

Tomaremos $size = n$ e $inicial = 1$ para simplificar el cálculo y veamos que obtenemos ahora

$$\sum_{i=1}^{n-1} a = a \cdot \sum_{i=1}^{n-1} 1 = a \cdot (n-1)$$

Es claro que $a \cdot (n-1) \in O(n)$ y por tanto la eficiencia teórica del algoritmo de kadane es $O(n)$.

4.1.2 Estudio empírico

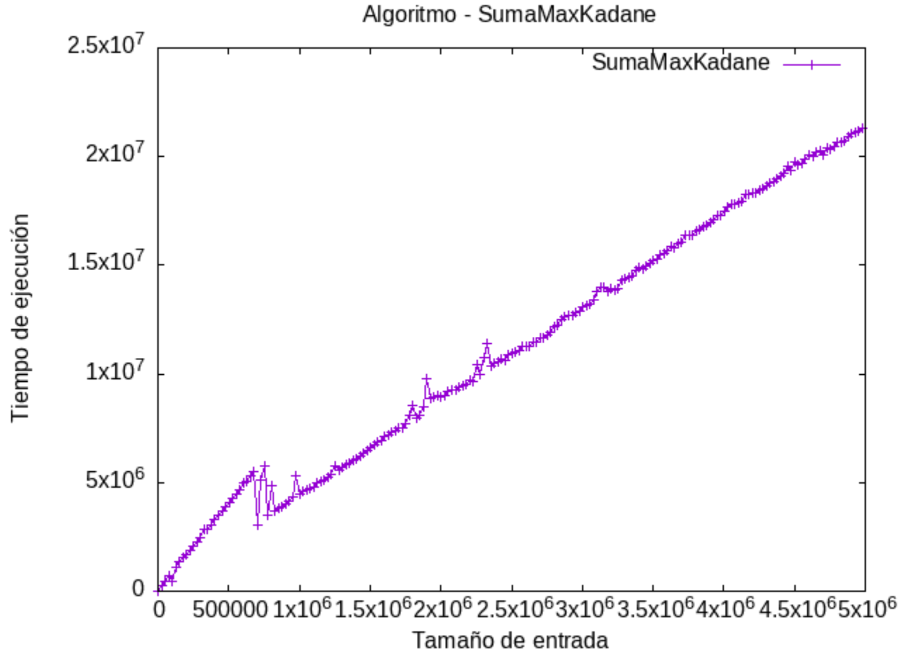

Figura 4.1: Ejecución algoritmo SumaMaxKadane

4.1.3 Estudio híbrido

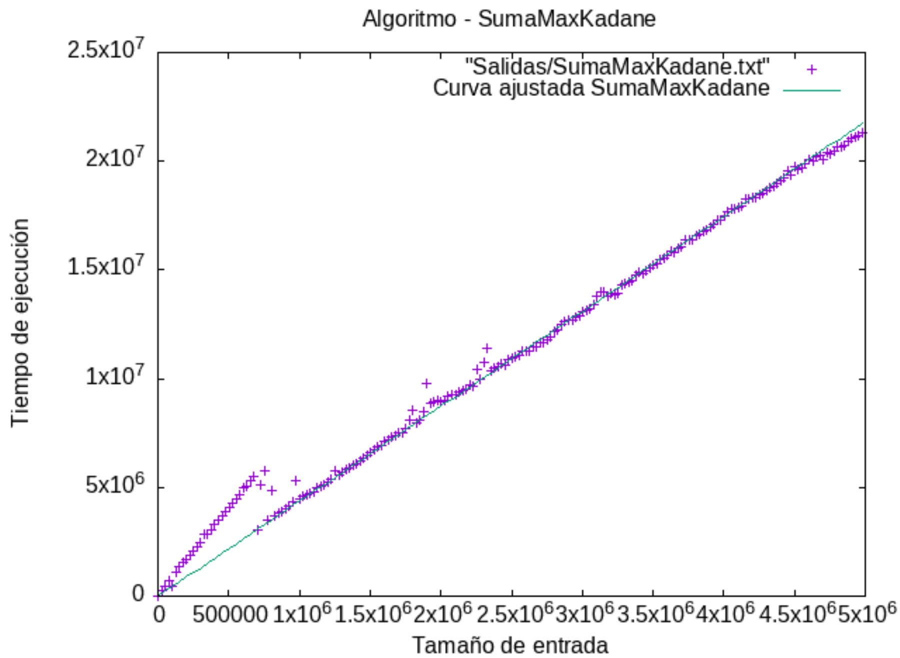

Figura 4.2: Ajuste hibrido

Tras la interpretación de los datos empíricos en gnuplot y de la formula teorica del métoco, obtenemos que las constantes ocultas son:

$$T_{Kadane} = 1.64263x + 0.996451$$

4.2 Problema 2: Enlosar un espacio

Para el segundo problema el algoritmo específico que empleamos para resolver el problema es el siguiente :

```

      void resuelve2(int i,int j,int n, vector<vector<int>> & mat){
        losas++;
        for(int l = 0; l< n; l++){
            for(int k = 0; k<n; k++){
                if(mat[i+l][j+k] == 0){
                    mat[i+l][j+k] = losas;
                }
            }
        }

}
```

El código consiste en rellenar las posiciones de una matriz 2x2 con un valor entero en las posiciones donde el valor sea 0. Podemos observar que hay dos bucles for anidados los cuales son $O(n)$ y el interior del bucle más profundo es $O(1)$ por tanto la eficiencia de este código es de $O(n^2)$

Apartado 5

Algoritmo Divide y Vencerás

En este apartado, estudiaremos la eficiencia teórica, empírica e híbrida de los algoritmos divide y vencerás de cada uno de los problemas.

5.1 Problema 1: Subsecuencia de suma máxima.

Para el primer problema, el algoritmo específico que empleamos es

5.1.1 Estudio teórico

```
SumaData SumaMax (int *v, int inicio, int final){
        SumaData result, d1, d2;
        if (inicio==final){
                result.max_izq = v[inicio];
                result.max_dch = v[inicio];
                result.sum = v[inicio];
                result.max_sub = v[inicio];
                return (result);
        }

        int mid = (final+inicio)/2;
        (d1)=SumaMax(v, inicio, mid);
        (d2)=SumaMax(v, mid+1, final);

        result.max_izq = max(d1.max_izq, d1.sum+d2.max_izq);
        result.max_dch = max(d2.max_dch, d2.sum+d1.max_dch);
        result.sum = d1.sum + d2.sum;
        int max_cross = d1.max_dch + d2.max_izq;
        result.max_sub = max(max(max_cross, d1.max_sub), d2.max_sub);
        return (result);
}
```

Este algoritmo devuelve un tipo de dato Suma data, un struct definido por cuatro datos de tipo int. En cuanto a la eficiencai teórica, podemos ver una llamada recursiva a la función SumaMax con vectores de tamaño $\frac{n}{2}$. Teniendo en cuenta que el resto de líneas de código son asignaciones, comparaciones y operaciones arimtéticas, que son O(1), obtenemos la siguiente ecuación

$$T(n) = 2T(\frac{n}{2}) + 1$$

Al realizar un cambio de variable $n = 2^m$(luego $m = log_2(n)$), obtenemos:

$$T(2^m) = 2T(2^{m-1}) + 1$$
$$T(2^m) - 2T(2^{m-1}) = 1$$

Ahora calculamos por un lado la parte homogénea y por otro la no homogénea. En primer lugar, la homogénea:

$$T(2^m) - 2T(2^{m-1}) = 0 \implies p_H(x) = x - 2$$

En cuanto a la parte no homogénea

$$1 = b_1^m q_1(m) \Longrightarrow b_1 = 1 \wedge q_1(m) = 1 \text{ con grado } d_1 = 0$$

Tenemos entonces el siguiente polinómio característico

$$p(x) = (x-2)(x-b_1)^{d_1+1} = (x-2)(x-1)$$

Por tanto la solución general es

$$t_m = c_{10}2^m m^0 + c_{20}1^m m^0 \overset{*}{\Longrightarrow} t_n = c_{10}n + c_{20} \Longrightarrow T(n) = c_{10}n + c_{20}$$

donde en $(*)$ hemos deshecho el cambio de variable
Por lo que obtenemos como resultado que $T(n) \in O(n)$

5.1.2 Estudio empírico

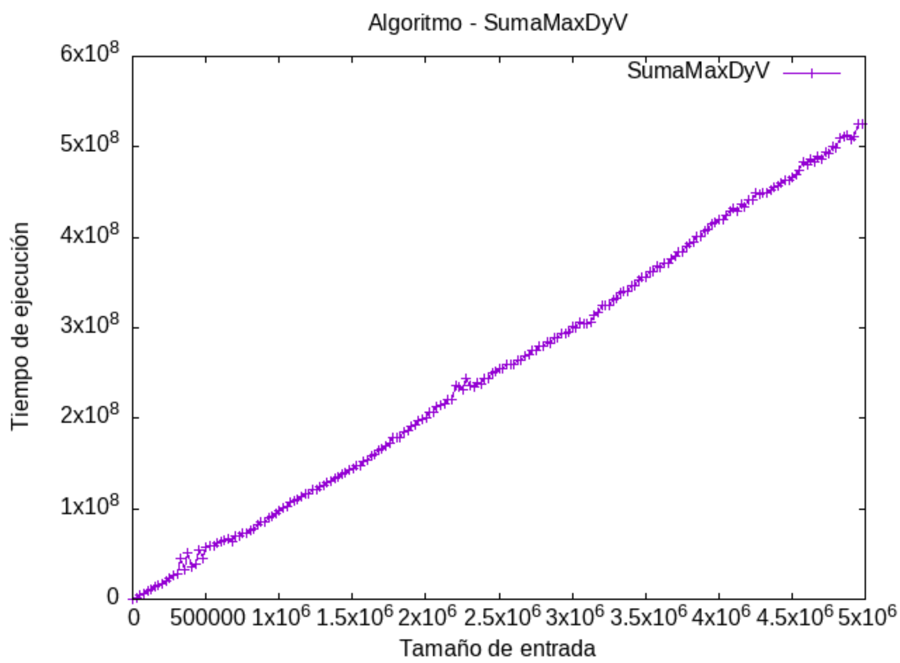

Figura 5.1: Ejecución algoritmo SumaMaxDyV

5.1.3 Estudio híbrido

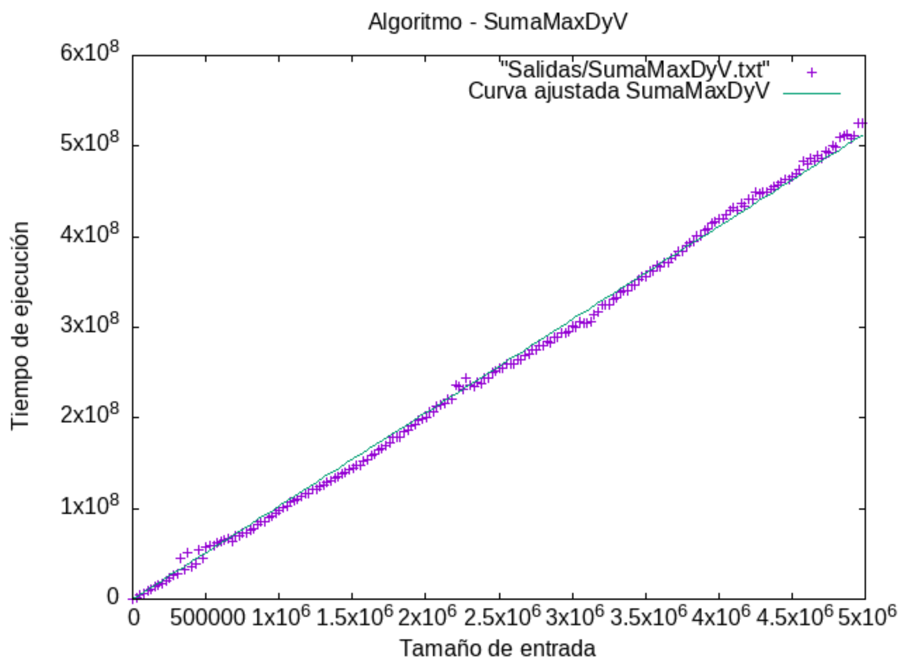

Figura 5.2: Ajuste híbrido algoritmo SumaMaxDyV

Tras la interpretación de los datos empíricos en gnuplot y de la formula teorica del métoco, obtenemos que las constantes ocultas son:

$$T_{SumaMaxDyV} = 1.65508x + 0.991967$$

5.2 Problema 2: Enlosar un espacio

5.3 Problema 3: Problema del viajante de comercio.

Para el tercer problema, nos encontramos ante un problema de complejidad NP-duro, por lo que no podemos encontrar una solución exacta en tiempo polinómico. Sin embargo, podemos encontrar una solución aproximada aplicando la técnica de Divide y Vencerás. Para ello, usamos el algoritmo de fuerza bruta para encontrar la solución exacta para subconjuntos del problema total y luego unimos las soluciones parciales para obtener una solución que se optimiza para asegurar un cierto mínimo de precision.

5.3.1 Estudio Empírico

Para determinar el umbral con el que obtenemos la mejor relación entre eficiencia y precisión, hemos realizado una serie de pruebas con diferentes valores de umbral, a partir de tamaño 10, el algoritmo de fuerza bruta se vuelve completamente inviable, de forma similar, para tamaño menor a 4, el algoritmo no nos produce ninguna mejora en cuanto a eficiencia debido a la cantidad de llamadas recursivas que saturan la pila, y la precision disminuye de forma considerable. Por ello, hemos decidido realizar las pruebas para tamaños de 4 a 10, con distintas ciudades con solución ya conocida, para comparar los tiempos de ejecución y la precisión de los resultados obtenidos:

Tiempo de ejecución

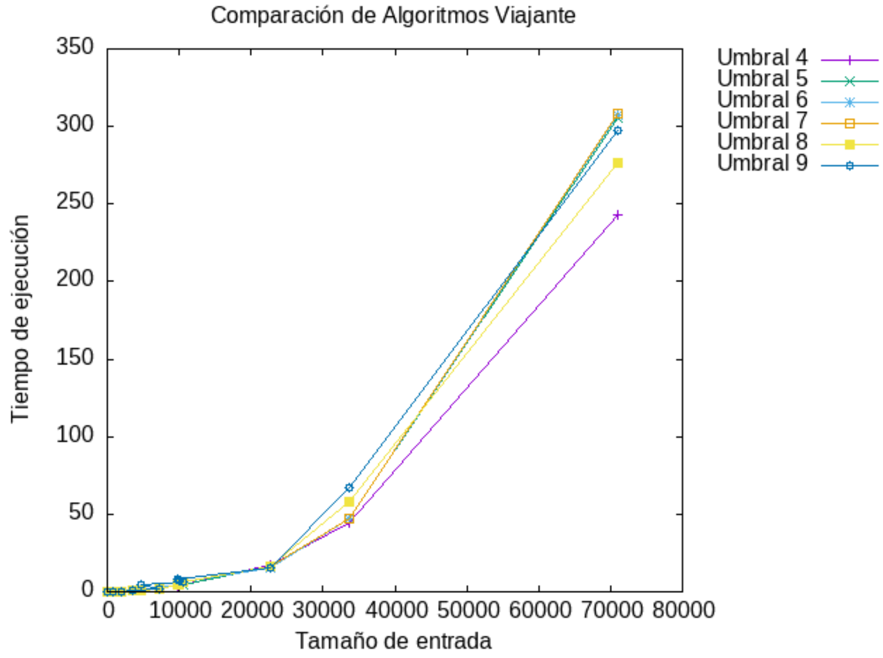

Figura 5.3: Ejecución algoritmo Viajante

Como se puede apreciar, los mejores resultados se han obtenido con los umbrales 4,8. Para decididir cual de los dos umbrales es el mejor, nos fijamos en la precisión que nos proporciona cada uno de ellos:

Precisión

Apartado 6

Conclusiones