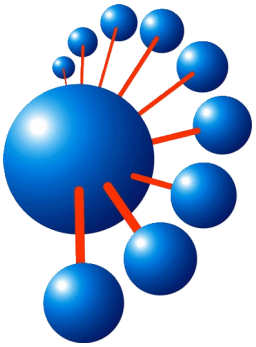


Tarea 3
Análisis de Algoritmos

Profesora: Andrea Rodríguez.
Alumnos: Diego Varas.
Jeremías Torres.

Lunes 25, junio 2018

Introducción

Para esta nueva tarea del curso se nos pide extraer, de una matriz de tamaño NxM, una submatriz cuya suma de sus números debe ser máxima. Para ello se pide diseñar e implementar el código que resuelva el problema dado.

La solución a este problema se aplica mediante programación dinámica.

Algoritmo de Solución

Para esta tarea se ha utilizado el algoritmo de kadane, el cual entrega la subsecuencia de suma máxima en un arreglo.

Definición de Variables y Funciones:

- **int FILA;** Variable que almacena el tamaño de la fila.
- **int COL;** Variable que almacena el tamaño de la columna.
- **int maxSum, finalizq, finalder, finalArriba, finalAbajo;** Variables para almacenar el resultado final
- **int **M;** Matriz dinámica principal
- **int *temp;** Arreglo dinámico que almacenará partes de la matriz principal M.
- **int* arr;** Arreglo copia de temp.

Es por ello que transformamos primeramente la matriz a un arreglo, en la siguiente función.

```
1 for (izq = 0; izq < COL; ++izq){
2        temp = (int*)calloc(FILA,sizeof(int));
3        for (der = izq; der < COL; ++der){
4            for (i = 0; i < FILA; ++i){// Calcula la suma entre la
izquierda y la derecha actual para cada fila 'i'
5                temp[i] += M[i][der];
6            }
7
8            sum = kadane(temp, &inicio, &final, FILA);

            // Compara sum con la suma máxima hasta el momento.
9            if (sum > maxSum){
10                maxSum = sum;
11                finalizq = izq;
12                finalder = der;
13                finalArriba = inicio;
14                finalAbajo = final;
15               }
16           }
17    }
```

Se le entrega la matriz M[][] que está llena de valores reales, en dicha función periódicamente se está almacenando en el vector temp[], la idea es arreglar las columnas izquierda y derecha una por una y encontrar la suma máxima de filas contiguas para cada par de columnas izquierda y derecha.

Básicamente, encontramos números de filas superiores e inferiores (que tienen la suma máxima) para cada par de columnas fijo izquierdo y derecho. Para encontrar los números de las filas superior e inferior, calcule el sol de los elementos en cada fila de izquierda a derecha y almacene estas sumas en una matriz, temp []. Entonces, temp [i] indica la suma de elementos de izquierda a derecha en la fila i.

Se crea ese vector según los valores mencionados previamente, luego en la línea 5 realiza dicha separación y suma, la cual se almacena en temp[ ] luego dicho vector se utilizará en la función de kadane:

```
1 for (i = 0; i < n; ++i){
2       sum += arr[i];
3       if (sum < 0){
4            sum = 0;
5            local_inicio = i+1;
          }
6        else if (sum > maxSum){
7            maxSum = sum;
8            *inicio = local_inicio;
9            *final = i;
          }
   }
```

Esta función recibe temp[ ] y suma los valores de la fila hasta encontrar una suma máxima la cual se comparará con la suma máxima anterior.

Cabe destacar que las columnas se suman por fila, es decir si tenemos esta matriz:

|1 2 3 4|
|5 6 7 8|

Se entrega a temp[ ] = {1,5}, el cual pasa por kadane y entrega un resultado (en este caso 6), luego la siguiente iteración o la siguiente columna a entregar será:

temp[ ] = {3, 11}

el cual entregará 14 y así sucesivamente.

Ahora si tenemos este vector temp[ ] = {2, -25, 7, 25} el algoritmo se comporta de esta manera (previamente se explicó con números positivos, esta vez se vuelve negativa la suma en algún momento).

Entonces comenzamos con sum = 0 y maxSum = 0, vamos de elemento en elemento:

suma + temp[0] (= 0+2 = 2), nuestra suma es mayor que maxSum así que actualizamos maxSum a 2.

Luego sum + temp[1] (= 2 + [-25] = -23), como nuestra suma actual no es mayor que maxSum no actualizamos, pero sum se actualiza a 0, porque nos dio un valor negativo y comenzamos nuevamente a sumar.

sum + temp[2] (= 0 + 7 = 7), luego sum es mayor que maxSum así que actualizamos maxSum = 7.

Por último sum + temp[3] (=7 + 25 = 32), actualizamos maxSum a 32 y termina. Este algoritmo es O(n) y probaremos que es correcta en la siguiente sección.

Análisis de Correctitud

Se analizará el algoritmo de Kadane:

Este algoritmo recibe un vector con números de la matriz (una submatriz) y analiza si la suma de ella es el número máximo, esta es la subestructura de solución, ya que se va entregando pequeños problemas que irá buscando la suma máxima.

Ahora demostremos que Kadane es correcto:

Denotamos temp(i,j) a la subsecuencia que comienza en i y termina en j y denotamos sum(i,j) a la suma de los elementos de temp(i,j).

Entonces si sum(i,j) < 0 entonces denotamos para todo k>j sum(i,q) no puede ser máximo:

sum(i,j) < 0 /+sum(j+1,q)
sum(i,j) + sum(j+1,q) < 0 + sum(j+1,q)

pero sum(i,j) + sum(j+1,q) = sum(i,q)
luego sum(i,q) < sum(j+1,q)

Por lo tanto, sum(i,q) no puede ser máximo, ya que existe otra subsecuencia temp(j+1,q) con una suma más grande.

Evaluación Experimental

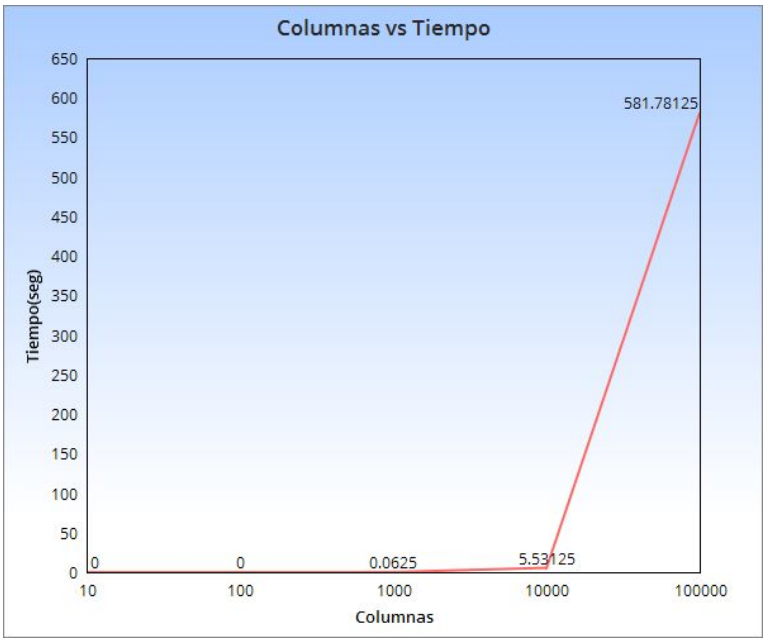
Se realizaron 2 pruebas a la solución en donde la primera se deja un número fijo de Filas y se hace variar el número de las columnas, para la segunda prueba se deja el número de columnas fijas y se hace variar el número de filas. Las pruebas se realizaron en una laptop con procesador Intel Core i3-5005U CPU @ 2.00GHz y 8gb de memoria Ram.

Prueba 1

Se probó para un valor fijo de filas, generando 4 gráficos.

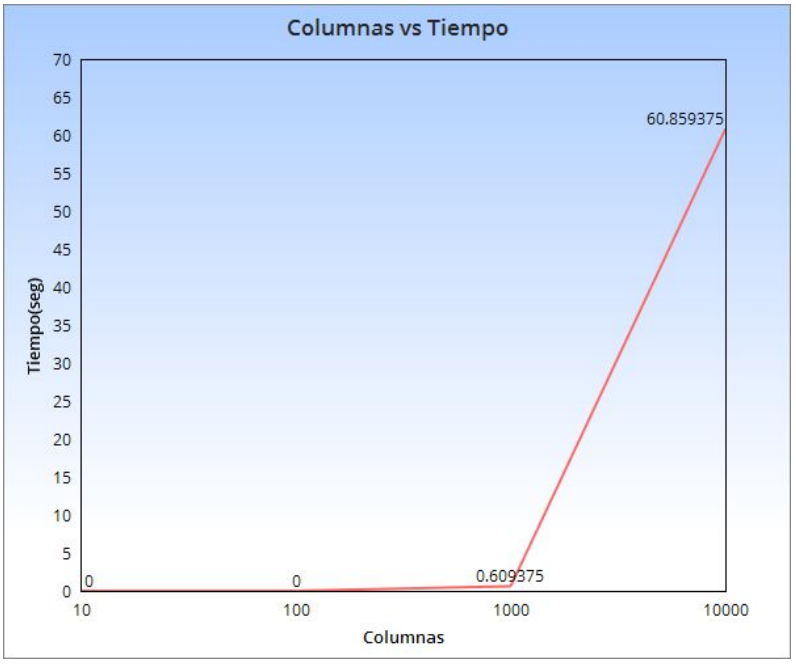
Número de Filas: 10

Fila	Columna	Tiempo
10	10	0
10	100	0
10	1000	0.0625
10	10000	5.53125
10	100000	581.78125



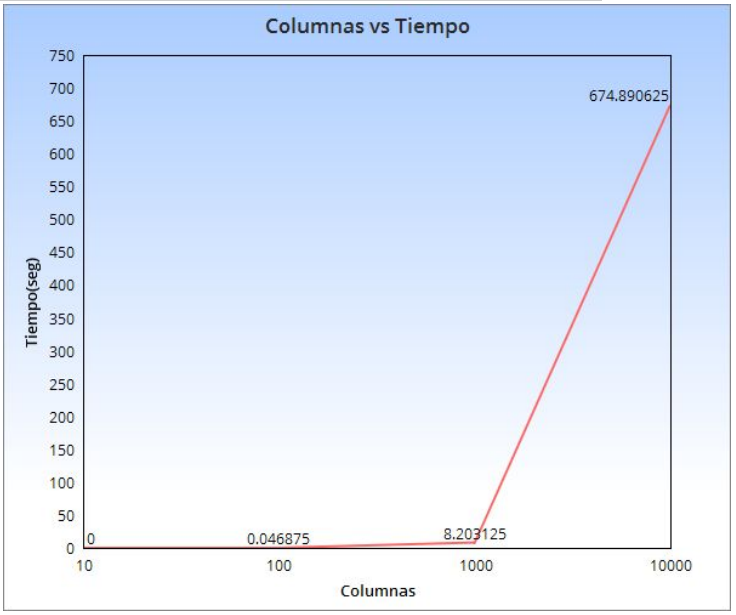
Número de Filas: 100

Fila	Columna	Tiempo
100	10	0
100	100	0
100	1000	0.609375
100	10000	60.859375



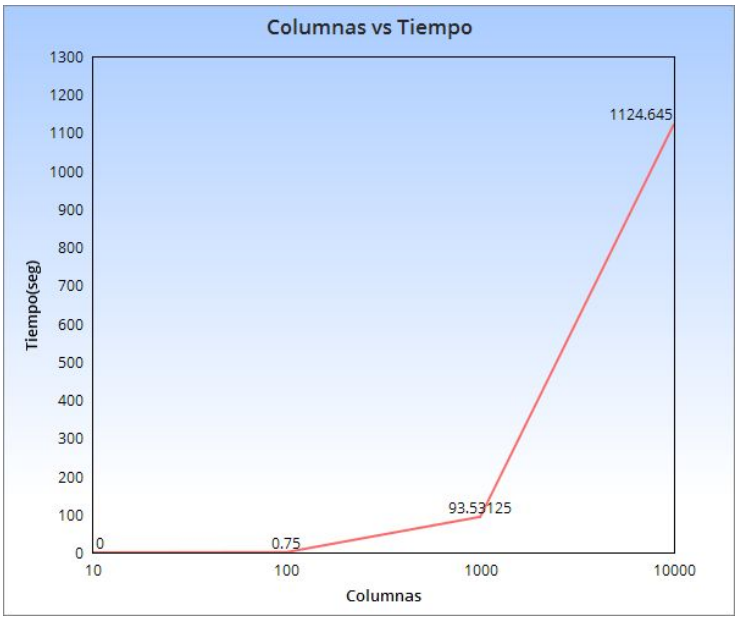
Número de Filas: 1000

Fila	Columna	Tiempo
1000	10	0
1000	100	0.046875
1000	1000	8.203125
1000	10000	674.890625



Número de Filas: 10000

Fila	Columna	Tiempo
10000	10	0
10000	100	0.75
10000	1000	93.53125
10000	10000	1124.645

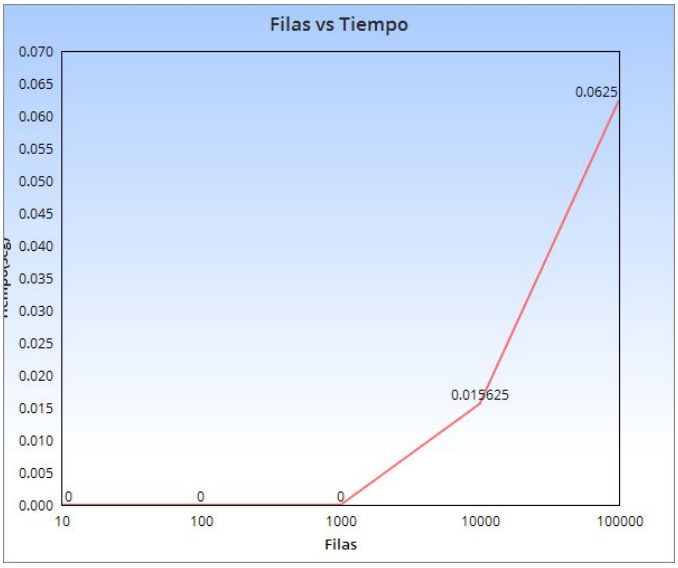


Prueba 2

Se probó para un valor fijo de columnas, generando 4 gráficos.

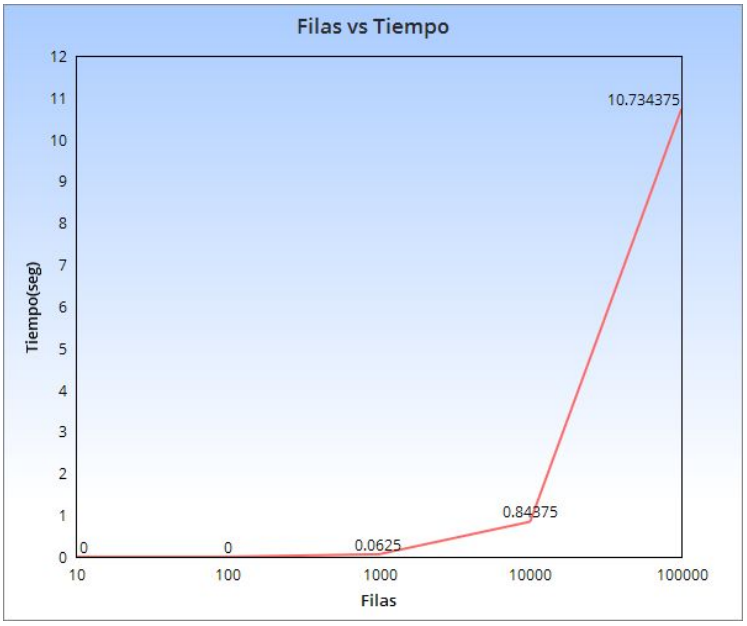
Número de Columnas: 10

Fila	Columna	Tiempo
10	10	0
100	10	0
1000	10	0
10000	10	0.015625
100000	10	0.0625



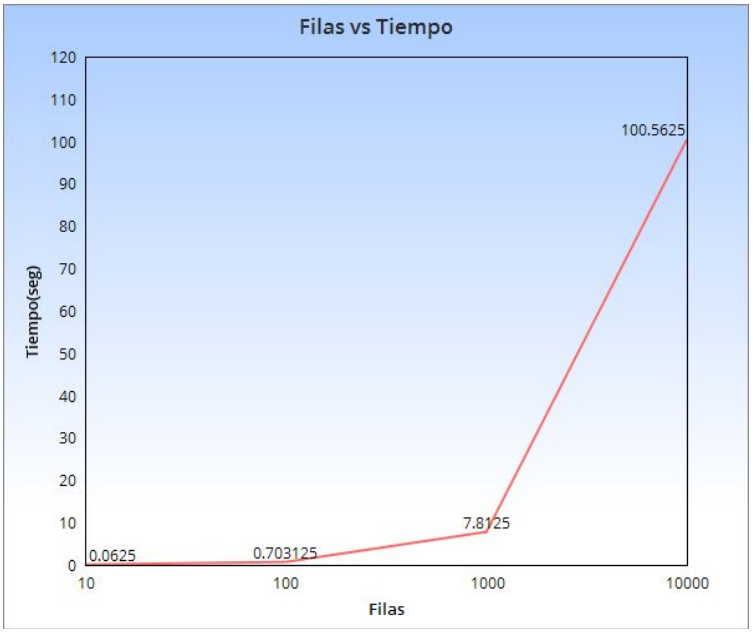
Número de Columnas:100

Fila	Columna	Tiempo
10	100	0
100	100	0
1000	100	0.0625
10000	100	0.84375
100000	100	10.734375



Número de Columnas: 1000

Fila	Columna	Tiempo
10	1000	0.0625
100	1000	0.703125
1000	1000	7.8125
10000	1000	



Dado todos los gráficos anteriores se puede visualizar que la curva de complejidad se asemeja a O(n^3), esto es dado porque en nuestro algoritmo existen 3 bucles for anidados, además la función que realiza el trabajo duro que es kadane(), esta es de complejidad O(n), por lo que la complejidad total de nuestra solución sería O(n^3 + n) lo que queda finalmente, dado todas las pruebas realizadas como O(n^3).