Análisis y diseño de algoritmos

Tarea 04

Diego Guadalupe Rodríguez Prieto

Professor/Dr. Eduardo A. Rodriguez Tello

1. Dado el problema de intersección de segmentos de línea, desarrolle los siguientes puntos:

   a) Programe en C++ un algoritmo de fuerza bruta para resolver el problema. El programa debe recibir por redirección de la linea de comandos los casos de prueba (archivos texto) con el siguiente formato: primera línea un entero n que indica el número total de segmentos de línea del problema, después hay n líneas cada una con cuatro enteros separados por espacios que representan las coordenadas x y y de los dos puntos extremos de cada línea del problema. El programa regresa un archivo con el siguiente formato: primera línea con un entero k que representa cuántas intersecciones existen entre las línea del problema, después hay k lineas con dos enteros separados por espacio que indican las coordenadas de los puntos de intersección respectivos.

   b) Analice matemáticamente el algoritmo usando las metodologías vistas en clase (i.e., usando sumatorias o relaciones de recurrencia).

      - Decidir acerca del parámetro n que indica el tamaño de la entrada del algoritmo: n es el tamaño del arreglo.
      - Identificar la operación básica del algoritmo: La comparación.
      - Determinar los casos peor, promedio y mejor para una entrada de tamaño n:
        - Mejor caso: Ocurre cuando en la primera comparación los segmentos se intersectan.
        - Caso medio: Cuando los segmentos que se intersectan están distribuidos uniformemente.
        - Peor caso: Cuando se deben comparar todos los segmentos posibles.
      - Expresar como una sumatoria el número de veces que la operación básica es ejecutada por el algoritmo analizado.

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1$$

      - Simplificar la sumatoria empleando propiedades y reglas estándar.

$$\begin{aligned} C_{algoritmo}(n) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 \\ &= \sum_{i=1}^{n-1}[(n)-(i+1)+1] \\ &= \sum_{i=1}^{n-1}(n-i) \\ &= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= n(n-1) - \frac{(n-1)n}{2} \\ &= \frac{n(n-1)}{2} \end{aligned}$$

$$\frac{n^2-n}{2} \in \Theta(n^2)$$

   c) Efectúe al menos 10 pruebas con el algoritmo desarrollado generando para cada prueba instancias de tamaños 25, 50, 100, 200, 400, etc. Ejecute 10 corridas de su algoritmo con cada tamaño de instancia generado y registre sus resultados promedio.

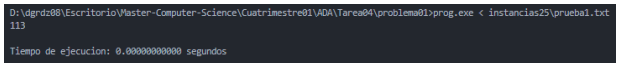
Cuadro 1. Promedio de tiempos de ejecución.

25	50	100	200	400
0.0	0.0005	0.0009	0.0044	0.008

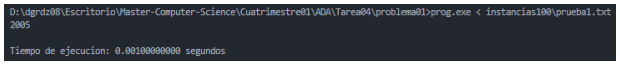
Los números de la fila superior reflejan el valor de k y los números de la fila inferior el tiempo promedio para cada valor de k.

   d) Presente capturas de pantalla para una corrida con cada uno de los tamaños de instancia siguientes: 25, 100, 400 y 1600.

      - Para el valor de $k = 25$

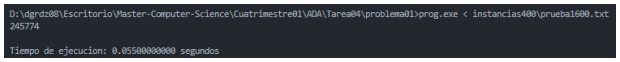


      - Para el valor de $k = 100$



      - Para el valor de $k = 400$

      - Para el valor de $k = 1600$



   e) Genere una tabla con información estadística de su programa: tamaño de la instancia, tiempo promedio para efectuar 10 ejecuciones, desviación estándar del tiempo de ejecución, y número de operaciones básicas promedio. Estas tablas deben permitir analizar cómo afecta el crecimiento del tamaño de los casos de prueba al desempeño del algoritmo implementado.

Cuadro 2. Promedio de tiempos de ejecución.

Tamaño	Promedio	Desv. Estándar	Núm. Op.
25	0.0	0.0	300.0
50	0.0005	0.0005	1225.0
100	0.0009	0.0003	4950.0
200	0.0044	0.005936	19900.0
400	0.008	0.0027568	79800.0

   f) A partir de la tabla anterior grafíque para el algoritmo analizado los siguientes resultados de sus ejecuciones: tamaño de entrada (eje X) contra el tiempo de cómputo consumido

(eje Y). Genere la línea de tendencia para los datos, utilizando la opción (exponencial, lineal, logarítmica, etc) que permita obtener el mejor ajuste. Reporte la ecuación de la línea de tendencia, así como el valor de R^2. (Un valor de R^2 más cercano a 1.0 es indicativo de un mejor ajuste).

Valor de $R^2 = 0{,}98102986$

Tomando valores hasta 1600, siendo la ultima instancia empleada, para poder ver mas claramente la manera en que se comporta la función.

g) Responder la siguiente pregunta: Con base en los experimentos realizados y considerando un tiempo máximo de ejecución sobre su computadora de 7 días, ¿Cuál es el tamaño máximo de entrada que puede resolver el algoritmo analizado (con una sola ejecución independiente)? Justifique su respuesta.

Para poder determinar algo así, tenemos que pensar una ecuación que nos permita calcular esto sin necesidad de correr nuestro código por tanto tiempo, es por ello que, tomando n = 100, siendo n el tamaño de la instancia, tardando un t_p (tiempo promedio) de 0.0009.

$$Tiempo_N \approx t_p \times \left(\frac{N}{n}\right)^2$$

Donde podemos ver al $Tiempo_N$ como los 7 días, que su equivalencia en segundos es 604800 y el cuadrado reflejando al comportamiento no lineal del algoritmo.

$$604800 \approx 0{,}0009 \times \left(\frac{N}{100}\right)^2$$

$$N \approx 100 \times \sqrt{\frac{604,800}{0{,}0009}} \approx 2,5923 \times 10^6$$

Siendo entonces que si ejecutamos el código por 7 días, podríamos resolver una instancia de tamaño $2,5923 \times 10^6$.

h) Analice y discuta sus observaciones de la información contenida en la tabla y gráfica. Trate de correlacionar sus observaciones con los resultados del análisis matemático de la complejidad del algoritmo que implementó.

Lo que podemos ver en la gráfica es un comportamiento que se atribuye a un crecimiento exponencial, de tal modo que mientras aumentemos el tamaño de la instancia a resolver, el tiempo crecerá acorde a este tipo de complejidad, tal como lo que se encuentra al resolver nuestra complejidad haciendo uso de sumatorias.

2. La empresa Transportes Eficientes dedicada a la logística tiene almacenes situados en diversos estados del país. Dado que se conocen las distancias de las carreteras comunican ciertos pares de almacenes, entonces es posible modelar esta situación como un grafo ponderado no dirigido. El gerente de la empresa desea conocer la ruta mas larga entre cualquier par de almacenes ya que esto le permitirá tomar decisiones estratégicas para hacer mas eficiente la operación de su empresa. Dado este problema práctico, desarrolle los siguientes puntos:

   a) Diseñe e implemente en C++ un algoritmo recursivo basado en DFS para encontrar la ruta más larga entre cualquier par de almacenes de la empresa. El programa recibe por redirección de la línea de comandos el nombre de un archivo de texto que contiene un grafo ponderado dirigido con el formato que se utilizó en clase (sesión 9). El resultado que se imprimirá en pantalla es la distancia total de la ruta más larga entre cualquier par de almacenes de la empresa, la lista de almacenes que forman dicha ruta y el tiempo total de ejecución. Asegúrese de utilizar una lista de adyacencia para representar el grafo.
   b) Analice matemáticamente la complejidad temporal y espacial de su algoritmo

   Partiendo de la siguiente parte de código

```
    for(int i = 0; i < E; i++) {
    std::cin >> x >> y >> w;
    ady[x].push_back({y, w});
}
```

   Dado lo anterior, se puede observar que una parte de la complejidad se define por $O(E)$.

   Tomando en cuenta el siguiente fragmento de código.

```
for(int i = 0; i < V; i++) {
    for(int j = 0; j < MAX; j++) {
        visitado[j] = false;
    }
    currentDist = 0;
    dfs_longest(i, 0);
}
```

   En la parte anterior nos damos cuenta de que tenemos un ciclo for anidado, donde el exterior nos recorre los vértices y hace una llamada recursiva a $dfs_longest$, donde contiene un ciclo dentro y un bucle interno para cambiar el estado del vértice inicial.
   Habiendo considerado todo lo contrario, se puede esperar que:
   - Complejidad temporal: $O(V(V+E))$
   - Complejidad espacial: $O(V+E))$

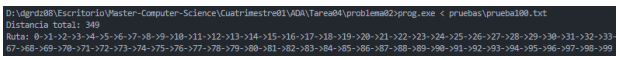
   En donde la complejidad temporal puede elevarse a un orden mayor
   c) Presente capturas de pantalla con ejemplos de sus corridas para los tamaños de instancia $|V| = n = 25, 50, 100$ generados aleatoriamente. Para cada tamaño presente una corrida.
   - Para el valor de $k = 25$

```
D:\dgrdz08\Escritorio\Master-Computer-Science\Cuatrimestre01\ADA\Tarea04\problema02>prog.exe < pruebas\prueba25.txt
Distancia total: 89
Ruta: 0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18->19->20->21->22->23->24
```

   - Para el valor de $k = 50$

```
D:\dgrdz08\Escritorio\Master-Computer-Science\Cuatrimestre01\ADA\Tarea04\problema02>prog.exe < pruebas\prueba50.txt
Distancia total: 174
Ruta: 0->1->2->3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18->19->20->21->22->23->24->25->26->27->28->29->30->
```

   - Para el valor de $k = 100$



d) Sin realizar modificaciones a su programa, analice cómo se vería afectada la complejidad temporal y espacial de su algoritmo en el caso de que se utilizara una matriz de adyacencia para representar el grafo. Presente su análisis y justifique sus afirmaciones.

Dado que en un principio al hacer uso de una lista de adyacencia, tenemos una complejidad de $O(V+E)$, debido que solamente es necesario almacenar información de vértices y aristas ya existentes, ahora deberemos contemplar el espacio de memoria para una matriz en la cual puede o no haber coincidencias en una arista, afectando directamente a la **complejidad espacial**, en el caso de la **complejidad temporal**, sabemos que en el momento de nosotros recorrer una matriz, debemos movernos en las columnas y filas de los vértices, por tanto la complejidad crecería en $O(V^2)$

3. Un grafo es bipartita [1] si todos sus vértices pueden ser divididos en dos subconjuntos disjuntos X y Y de forma tal que cada arco conecte un vértice en X con uno en Y. Dado el problema de identificación de grafos bipartitas, desarrolle los siguientes puntos:

a) Diseñe e implemente en C++ un algoritmo basado en BFS para verificar si un grafo es bipartita o no. Este recibe por redirección de la línea de comandos el nombre de un archivo de texto que contiene el grafo a verificar con el formato que se utilizó en clase (sesión 9). El resultado que despliega en pantalla es “Bipartita” o “No bipartita” y el tiempo total de ejecución. Asegúrese de utilizar una lista de adyacencia para representar el grafo.

b) Analice matemáticamente la complejidad temporal y espacial de su algoritmo.

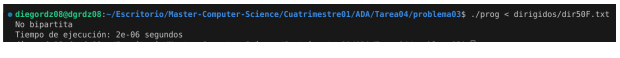
De modo que en en la función booleana que se declara en un principio se presenta un ciclo en el cual contenemos el recorrido de nuestra lista de adyacencia, dependemos de el en el tiempo para poder cumplir con el cometido del algoritmo, en el entendimiento que se tiene que recorrer cada vértice por lo menos una vez. Dado que a la par dependemos del tamaño de las aristas, se suman ambas complejidades tanto para lo temporal como lo espacial.

- Complejidad Temporal: $O(V+E)$
- Complejidad Espacial: $O(V+E)$

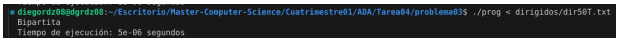
c) Presente capturas de pantalla con ejemplos de sus corridas para los tamaños de instancia $n = 50, 100, 150$. Para cada tamaño presente dos corridas una con un grafo bipartita y otra con un grafo no bipartita. El conjunto de sus instancias de prueba debe contener grafos dirigidos y no dirigidos.

En caso de grafos dirigidos

- Para el valor de $n = 50$ dirigido (No bipartita)



- Para el valor de $n = 50$ dirigido (Bipartita)



- Para el valor de $n = 100$ dirigido (No bipartita)

- Para el valor de $n = 100$ dirigido (Bipartita)

- Para el valor de $n = 150$ dirigido (No bipartita)

- Para el valor de $n = 150$ dirigido (Bipartita)

d) Sin realizar modificaciones a su programa, analice cómo se vería afectada la complejidad temporal y espacial de su algoritmo en el caso de que se utilizara una matriz de adyacencia para representar el grafo. Presente su análisis y justifique sus afirmaciones.

Se sabe que al hacer uso de una matriz de adyacencia, nuestro algoritmo se tardaria mas el tiempo de ejecucion de modo que como se requiere asegurarse que el vector se mueve en cada uno de los indices, es necesario hacer un recorrido de $V \times V$ veces, afectando de igual manera a la complejidad espacial, siendo causante de que las complejidades se definan como:

- Complejidad temporal: $O(V^2)$
- Complejidad espacial: $O(V^2)$

Referencias

[1] INAOE, *Capítulo 4: Grafos*, Consultado el 8 de noviembre de 2024, 2024. dirección: https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Matematicas_Discretas/Capitulo_4_Grafos.pdf.