



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

Trabajo Práctico N°2

Nombre: Melina Lazzaro **Padrón:** 105931

Nombre: Nahuel Castro **Padrón:** 106551

Nombre: Ana Gutson **Padrón:** 105853

Nombre: Nicolás Pinto **Padrón:** 105064

Nombre: Iván Litteri **Padrón:** 106223

Fecha de Entrega: 18/10/2021

Grupo: liomessi30

Lineamientos básicos

- El trabajo se realizará en grupos de cinco personas.
- Se debe entregar el informe en formato pdf y código fuente en (.zip) en el aula virtual de la materia.
- El lenguaje de implementación es libre. Recomendamos utilizar C, C++ o Python. Sin embargo si se desea utilizar algún otro, se debe pactar con los docentes.
- Incluir en el informe los requisitos y procedimientos para su compilación y ejecución. La ausencia de esta información no permite probar el trabajo y deberá ser re-entregado con esta información.
- El informe debe presentar carátula con el nombre del grupo, datos de los integrantes y y fecha de entrega. Debe incluir número de hoja en cada página.
- En caso de re-entrega, entregar un apartado con las correcciones mencionadas

Índice general

1. Minimizando costos	3
1.1. Objetivos	3
1.1.1. Algoritmo de Johnson	4
1.1.2. Comparación de complejidad temporal y espacial entre los algoritmos propuestos . . .	5
1.1.3. Comparación entre situaciones	6
1.1.4. Ejemplo	7
1.1.5. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología greedy? . .	8
1.1.6. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología de programación dinámica?	8
2. Un poco de teoría	9
2.1. Objetivos	9
2.2. Resultados	9
2.3. Conclusiones	9

Capítulo 1

Minimizando costos

1.1. Objetivos

Una empresa productora de tecnología está planeando construir una fábrica para un producto nuevo. Un aspecto clave en esa decisión corresponde a determinar dónde la ubicarán para minimizar los gastos de logística y distribución. Cuenta con N depósitos distribuidos en diferentes ciudades. En alguna de estas ciudades es donde deberá instalar la nueva fábrica. Para los transportes utilizarán las rutas semanales con las que ya cuentan. Cada ruta une dos depósitos en un sentido. No todos los depósitos tienen rutas que los conecten. Por otro lado, los costos de utilizar una ruta tienen diferentes valores. Por ejemplo hay rutas que requieren contratar más personal o comprar nuevos vehículos. En otros casos son rutas subvencionadas y utilizarlas les da una ganancia a la empresa. Otros factores que influyen son gastos de combustibles y peajes. Para simplificar se ha desarrollado una tabla donde se indica para cada ruta existente el costo de utilizarla (valor negativo si da ganancia).

Los han contratado para resolver este problema.

Han averiguado que se puede resolver el problema utilizando Bellman-Ford para cada par de nodos o Floyd-Warshall en forma general. Un amigo les sugiere utilizar el algoritmo de Johnson.

Aclaración: No existen ciclos negativos!

Se pide:

1. Investigar el algoritmo de Johnson y explicar cómo funciona. ¿Es óptimo?
2. En una tabla comparar la complejidad temporal y espacial de las tres propuestas.
3. Analizar en qué situaciones una solución es mejor que otras
4. Crear un ejemplo con 5 depósitos y mostrar paso a paso cómo lo resolvería el algoritmo de Johnson.
5. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología greedy? Justifique
6. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología de programación dinámica? Justifique.
7. Programar la solución usando el algoritmo de Johnson.

Formato de los archivos:

El programa debe recibir por parámetro el path del archivo donde se encuentran los costos entre cada depósito. El archivo debe ser de tipo texto y presentar por renglón, separados por coma un par de depósitos con su distancia.

Ejemplo: "depósitos.txt"

Debe resolver el problema y retornar por pantalla la solución. Debe mostrar por consola en en que ciudad colocar el depósito. Además imprimir en forma de matriz los costos mínimos entre cada uno de los depósitos.

1.1.1. Algoritmo de Johnson

El algoritmo de Johnson es un algoritmo cuya finalidad es hallar el camino más corto entre todos los pares de vértices de un grafo dirigido disperso. Permite que las aristas tengan pesos negativos, pero no que los ciclos tengan peso negativo.

Se basa en dos algoritmos:

- El algoritmo de Bellman-Ford, el cual realiza una transformación en el grafo inicial, eliminando todas las aristas de peso negativo, y detectando todos los ciclos negativos;
- El algoritmo de Dijkstra, usandolo en el grafo resultante de utilizar el algoritmo anterior.

¿Cómo funciona?

Para aplicar el algoritmo, se siguen los siguientes pasos:

1. Se agrega un nodo q al grafo, el cual está conectado a cada uno de los nodos del grafo por una arista de peso cero.
2. Se utiliza el algoritmo de Bellman-Ford, empezando por el nuevo vértice q , para determinar para cada vértice v el peso mínimo $h(v)$ del camino de q a v .
→ Si en este paso se detecta un ciclo negativo, el algoritmo concluye.
3. Se actualiza el peso de las aristas del grafo, usando lo calculado anteriormente: una arista de u a v con tamaño $w(u, v)$, da el nuevo tamaño $w(u, v) + h(u) - h(v)$
4. Para cada nodo s se usa el algoritmo de Dijkstra para determinar el camino más corto entre s y los otros nodos, usando el grafo actualizado anteriormente.

Como todas las aristas tienen ahora un peso actualizado con el mismo agregado de $h(u) - h(v)$, se asegura de que el camino más corto en el grafo original lo sea también en el grafo actualizado.

También, debido al modo en el que se computan los valores $h(v)$, se asegura de que todos los pesos de las aristas son no negativos.

¿Es óptimo?

Cuando analizamos algoritmos lo que nos interesa de ellos es analizar su optimalidad y su eficiencia. La optimalidad nos dice si, al utilizar un algoritmo en específico para resolver un problema, siempre obtengo el resultado correcto; y por otro lado la eficiencia (tanto en tiempo y espacio) tiene que ver con la cantidad de operaciones que se realizan para resolver el problema.

Para el problema que nos interesa resolver, este algoritmo sí es óptimo ya que se nos aclara que los grafos de nuestro problema no poseen ciclos negativos. Sin esa aclaración, se podría decir que el algoritmo no es óptimo ya que no devuelve el resultado correcto con grafos que contienen ciclos negativos.

Además, al lograr que los pesos de las aristas no sean negativas, aseguramos la optimalidad de los caminos encontrados por Dijkstra, y por ende, de Johnson.

1.1.2. Comparación de complejidad temporal y espacial entre los algoritmos propuestos

Los algoritmos propuestos para resolver el problema en cuestión son:

- El algoritmo de Bellman Ford.
- El algoritmo de Floyd Warshall.
- El algoritmo de Johnson.

Se procede entonces a medir el uso de recursos de estos algoritmos.

Al medir el uso de recursos de un algoritmo, uno lo puede hacer, entre otras formas, mediante su complejidad temporal, es decir, cuánto demora el algoritmo en terminar su ejecución dada una entrada v , y su complejidad espacial, midiendo el uso de memoria operativa requerida por el algoritmo (tanto el código como los datos con los que éste opera).

Se analizan las complejidades temporales y espaciales de los algoritmos propuestos.

Bellman-Ford

El algoritmo de Bellman-Ford es un algoritmo para encontrar el camino más corto en un grafo dirigido ponderado. Normalmente se utiliza cuando hay aristas con peso negativo.

Si el grafo contiene un ciclo de coste negativo, el algoritmo lo detectará, pero no encontrará el camino más corto que no repite ningún vértice.

El algoritmo consiste en hacer varias veces lo siguiente: recorrer todas las aristas. Si la arista (u, v) de longitud L es tal que la distancia que teníamos guardada desde el origen (s) hasta el nodo u más la longitud de la arista L , es menor a la distancia que teníamos guardada hasta el nodo v , actualizamos la información de la distancia hasta el nodo v como la suma mencionada.

Si el grafo tiene V nodos, entonces, alcanza con recorrer sus aristas $V-1$ veces. Por ende, la complejidad temporal del algoritmo es

$$\mathcal{O}(V.E)$$

con V : vértices E : aristas.

Al implementar el algoritmo, se utiliza una matriz que dependerá de la cantidad de vértices que tenga el grafo, por lo que entonces la complejidad espacial del mismo será de

$$\mathcal{O}(V)$$

con V : vertices.

Floyd-Warshall

El algoritmo de Floyd-Warshall es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. Este algoritmo es un claro ejemplo de programación dinámica, ya que desglosa el problema en subproblemas más pequeños, luego combina las respuestas a esos subproblemas para resolver el gran problema inicial.

Este algoritmo compara todas las rutas posibles a través del gráfico entre cada par de vértices. Es capaz de hacer esto con solo V^3 comparaciones. Esto es notable considerando que puede haber hasta V^2 aristas en el gráfico y se prueba cada combinación de aristas. Lo hace mejorando gradualmente una estimación en la ruta más corta entre dos vértices, hasta que se sabe que la estimación es óptima.

Una forma de representar el algoritmo es mediante una matriz cuadrada de $V \times V$, dando lugar a una complejidad espacial de

$$\mathcal{O}(V^2)$$

con V: vertices.

Es debido a todo lo anteriormente mencionado que su complejidad temporal es de

$$\mathcal{O}(V^3)$$

con V: vertices,

dependiendo completamente del número de vértices del gráfico.

Como se verá más adelante, esto lo hace especialmente útil para un cierto tipo de gráfico y no tanto para otros tipos.

Johnson

Como se dijo previamente, Johnson está comprendido por dos algoritmos, cuyas complejidades ya conocemos. Se procede a analizar entonces cada uno de los pasos.

- el primer paso del algoritmo es simple: consiste en añadir un vértice, con una arista por cada vértice existente. Por ende, al complejidad es de $\mathcal{O}(V)$
- el segundo y tercer paso se ejecutan con la complejidad del algoritmo Bellman-Ford, es decir, $\mathcal{O}(VE)$.
- el paso final se ejecuta con la complejidad del algoritmo de Dijkstra, el cual es implementado por cada vértice del grafo. Si se decide implementar el algoritmo de Dijkstra por montículos de Fibonacci, entonces por cada iteración, se requiere de $\mathcal{O}(V \log V + E)$ tiempo para completar para una complejidad total de

$$\mathcal{O}(V^2 \log V + VE)$$

con V: vertices E: aristas.

Ahora, se analiza su complejidad espacial. *falta*

Complejidad	Bellman-Ford	Johnson	Floyd-Warshall
Temporal	$\mathcal{O}(VE)$	$\mathcal{O}(V^2 \log V + VE)$	$\mathcal{O}(V^3)$
Espacial	$\mathcal{O}(V)$	$\mathcal{O}((V + 1)(E + V))$	$\mathcal{O}(V^2)$

1.1.3. Comparación entre situaciones

Con base en lo analizado anteriormente, concluimos que:

El algoritmo Bellman-Ford es un algoritmo que calcula las rutas más cortas desde un único vértice de origen a todos los demás vértices en un dígrafo ponderado, detectando los ciclos negativos.

El algoritmo Floyd-Warshall calcula las rutas más cortas de cada nodo a todos los demás. Se utiliza cuando cualquiera de todos los nodos puede ser una fuente, por lo que desea que la distancia más corta para llegar a cualquier nodo de destino desde cualquier nodo de origen. Esto solo falla cuando hay ciclos negativos.

Su complejidad temporal depende solamente de los vertices del grafo.

El algoritmo de Johnson calcula las rutas más cortas de cada nodo a todos los demás.

Su complejidad temporal depende tanto de los vertices como de la cantidad de aristas del grafo.

Entonces, se pueden dividir los algoritmos sugeridos en dos grupos: los algoritmos para calcular las rutas más cortas de una sola fuente (single-source), y los algoritmos para calcular las rutas más cortas de todos los pares (all-pairs).

En el primer grupo se encuentra el algoritmo de Bellman-Ford, mientras que en el segundo grupo están los algoritmos de Floyd-Warshall y Johnson.

Ahora bien, se comparan los algoritmos correspondientes al segundo grupo:

Si se mira la complejidad, como se menciona anteriormente, la complejidad temporal de Floyd Warshall depende totalmente de la cantidad de vértices del grafo, mientras que la Johnson depende tanto de la cantidad de aristas como la de vértices.

Es por esto que Floyd-Warshall es más efectivo para gráficos densos (muchos vértices), mientras que el algoritmo de Johnson es más efectivo para gráficos dispersos (pocos vértices).

1.1.4. Ejemplo

INSERTAR GRAFO ORIGINAL ACÁ

Paso 1. Se añade un nuevo vértice q al grafo, conectado a cada uno de los vértices del grafo por una arista de peso cero

INSERTAR GRAFO CON NUEVO VÉRTICE ACÁ

Paso 2. Se utiliza el algoritmo de Bellman-Ford, empezando por el nuevo vértice q , para determinar para cada vértice v el peso mínimo $h(v)$ del camino de q a v . Si en este paso se detecta un ciclo negativo, el algoritmo concluye

	q	v_0	v_1	v_2	v_3	v_4
0						
1						
2						
3						
4						

Paso 3.

Paso 4. Para cada nodo s se usa el algoritmo de Dijkstra para determinar el camino más corto entre s y los otros nodos, usando el grafo actualizado anteriormente.

	v_0	v_1	v_2	v_3	v_4
v_0					
v_1					
v_2					
v_3					
v_4					

	v_0	v_1	v_2	v_3	v_4
v_0					
v_1					
v_2					
v_3					
v_4					

	v_0	v_1	v_2	v_3	v_4
v_0					
v_1					
v_2					
v_3					
v_4					

	v_0	v_1	v_2	v_3	v_4
v_0					
v_1					
v_2					
v_3					
v_4					

	v_0	v_1	v_2	v_3	v_4
v_0					
v_1					
v_2					
v_3					
v_4					

1.1.5. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología greedy?

Se podría decir que una parte de su funcionamiento utiliza una metodología greedy ya que este algoritmo usa el algoritmo de Dijkstra y éste utiliza en su funcionamiento una metodología Greedy, porque siempre elige el vértice 'más liviano' o 'más cercano' en $V - S$ para agregar al conjunto S (siendo S el conjunto de los vértices cuyos pesos finales de la fuente s ya se han determinado). Decimos 'una parte' porque además el algoritmo de Johnson en una parte de su funcionamiento utiliza el algoritmo de Bellman-Ford (que no utiliza en su funcionamiento una metodología greedy).

1.1.6. ¿Puede decirse que Johnson utiliza en su funcionamiento una metodología de programación dinámica?

Se podría decir que una parte de su funcionamiento utiliza una metodología de programación dinámica ya que este algoritmo (como se dijo en el punto anterior) usa el algoritmo de Bellman-Ford. Decimos 'una parte' porque además el algoritmo de Johnson en una parte de su funcionamiento utiliza el algoritmo de Bellman-Ford (que no utiliza en su funcionamiento una metodología greedy).

Capítulo 2

Un poco de teoría

2.1. Objetivos

1. Hasta el momento hemos visto 3 formas distintas de resolver problemas. Greedy, división y conquista, y programación dinámica..
 - a) Describa brevemente en qué consiste cada una de ellas.
 - b) Identifique similitudes, diferencias, ventajas y desventajas entre las mismas. ¿Podría elegir una técnica sobre las otras?
2. Tenemos un problema que puede ser resuelto por un algoritmo Greedy (G) y por un algoritmo de Programación Dinámica (PD). G consiste en realizar múltiples iteraciones sobre un mismo arreglo, mientras que PD utiliza la información del arreglo en diferentes subproblemas a la vez que requiere almacenar dicha información calculada en cada uno de ellos, reduciendo así su complejidad; de tal forma logra que $O(PD) \ll O(G)$. Sabemos que tenemos limitaciones en nuestros recursos computacionales (CPU y principalmente memoria). ¿Qué algoritmo elegiría para resolver el problema?

Pista: probablemente no haya una respuesta correcta para este problema, solo justificaciones correctas.

2.2. Resultados

2.3. Conclusiones