

## Trabajo Práctico 2

[7529/9506] Teoría de Algoritmos I  
Segundo cuatrimestre de 2021

Grupo:	404
Repositorio:	<a href="https://github.com/lucashemmingsen/7529tp2">github.com/lucashemmingsen/7529tp2</a>
Entrega:	nº 1 (20/10/2021)

### Integrantes del grupo 404

Padrón	Apellido, Nombre	Email
76187	Xxxxxxxxxx, Xxxxx Xxxxx	xxxxxxxxxxxx@fi.uba.ar
97529	Xxxxxxxxx, Xxxxx Xxxxxxx	xxxxxxxxxx@fi.uba.ar
102593	Xxxxxx, Xxxxx Xxxxx	xxxxxxx@fi.uba.ar
102649	Xxxxxx, XXXXXXXX Xxxxx	xxxxxxx@fi.uba.ar
106713	Xxxx Xxxxx, Xxxxx	xxxxx@fi.uba.ar

## Índice

<b>1. Introducción</b>	<b>2</b>
1.1. Resumen . . . . .	2
1.2. Lineamientos básicos . . . . .	2
<b>2. Parte 1: Minimizando costos</b>	<b>3</b>
2.0.1. Formato de los archivos . . . . .	3
2.1. Algoritmo de Johnson: funcionamiento . . . . .	4
2.2. Comparación de algoritmos: Complejidad . . . . .	5
2.3. Comparación de algoritmos: Ventajas y desventajas . . . . .	6
2.4. Algoritmo de Johnson: resolución manual . . . . .	7
2.4.1. Convertir a grafo sin negativos (Bellman-Ford) . . . . .	7
2.4.2. Aplicar Dijkstra . . . . .	10
2.4.3. Restaurar valor de distancias . . . . .	10
2.4.4. Reconstrucción caminos mínimos . . . . .	10
2.5. Metodología Greedy . . . . .	12
2.6. Programación dinámica . . . . .	13
2.7. Programa con solución . . . . .	14
<b>3. Parte 2: Un poco de teoría</b>	<b>15</b>
3.1. Enunciado . . . . .	15
3.2. Formas de resolución de problemas . . . . .	16
3.3. Caso teórico puntual . . . . .	17

# 1. Introducción

## 1.1. Resumen

El presente informe documenta el enunciado y la solución del segundo trabajo práctico de la materia Teoría de Algoritmos I. El mismo comprende el análisis del problema planteado, la comparación de posibles algoritmos y la resolución manual del algoritmo de Johnson; así como también respuestas a preguntas teóricas.

## 1.2. 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

## 2. Parte 1: Minimizando costos

### Enunciado

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.

### 2.0.1. Formato de los archivos

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: "depositos.txt"

```
A,B,54
A,D,-3
B,C,8
...
```

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

## 2.1. Algoritmo de Johnson: funcionamiento

### Enunciado 1.1

Investigar el algoritmo de Johnson y explicar cómo funciona. ¿Es óptimo?

## 2.2. Comparación de algoritmos: Complejidad

### Enunciado 1.2

En una tabla comparar la complejidad temporal y espacial de las tres propuestas.

### 2.3. Comparación de algoritmos: Ventajas y desventajas

#### Enunciado 1.3

Analizar en qué situaciones una solución es mejor que otras.

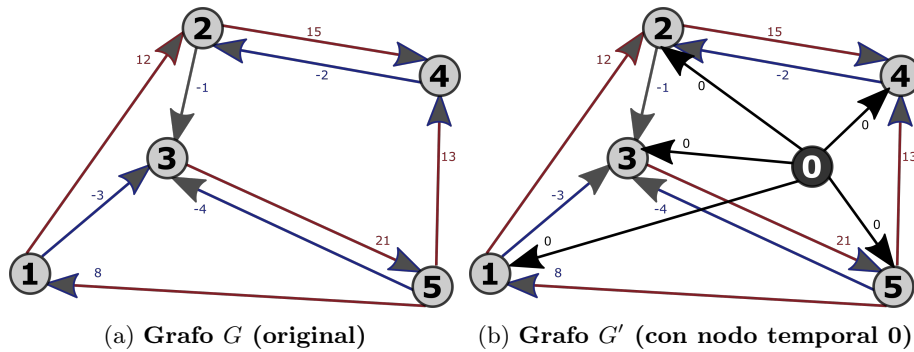
## 2.4. Algoritmo de Johnson: resolución manual

### Enunciado 1.4

Crear un ejemplo con 5 depósitos y mostrar paso a paso cómo lo resolvería el algoritmo de Johnson.

#### 2.4.1. Convertir a grafo sin negativos (Bellman-Ford)

Para el ejemplo tomaremos un grafo  $G$  arbitrario (figura 1a). A partir de éste, creamos un grafo  $G'$  (figura 1b) en principio idéntico pero al que le agregamos temporalmente un vértice  $V_0$  con aristas de peso 0 hacia cada otro nodo preexistente.



A partir de este grafo, aplicamos el algoritmo de Bellman-Ford tomando como origen el vértice temporal  $V_0$ . Para el mismo, en cada iteración, recorreremos todas las aristas haciendo una reducción; en términos generales:

Siendo  $h(x)$  la distancia al origen estimada para el vértice  $x$ , y  $w(u, v)$  el peso original de la arista que va desde el vértice  $u$  hasta  $v$ , comparamos  $h(u) + w(u, v)$  contra  $h(v)$ . Si el primer valor es inferior al segundo, actualizamos  $h(v) := h(u) + w(u, v)$  y anotamos  $u$  como padre de  $v$ :  $\pi(v) := u$ ; esto es, qué otro vértice se tomó para calcular  $h(v)$ .

Iteramos estas reducciones sobre todas las aristas, un total de  $|V| - 1$  veces, y luego una vez más para comprobar que no haya algún ciclo negativo (que no haya cambios). Adicionalmente, es posible finalizar con un resultado correcto (es decir, no hay ciclos) cuando durante una iteración externa no se produjo cambio alguno.<sup>1</sup>

<sup>1</sup>Si no hay cambios en la iteración, los datos de entrada de la siguiente iteración serán los mismos; por lo que el resultado también será el mismo (y seguirá sin haber cambios). Si bien no modifica big O, puede reducir el tiempo real de cómputo.



Cuadro 1: Primera iteración de Bellman-Ford en el algoritmo de Johnson

Fórmula posible nuevo	1ª iteración			$V_1$		$V_2$		$V_3$		$V_4$		$V_5$	
	Nuevo	$\leq$ $\geq$	Ant.	h	p	h	p	h	p	h	p	h	p
$0 \rightarrow 1 \dots 0 \rightarrow 5$	$0 + 0$	$<$	0	0	$V_0$	0	$V_0$	0	$V_0$	0	$V_0$	0	$V_0$
$h(1) + w(1, 2)$	$0 + 12$	$<$	0	0	$V_0$	0	$V_0$	0	$V_0$	0	$V_0$	0	$V_0$
$h(1) + w(1, 3)$	$0 + (-3)$	$<$	0	0	$V_0$	0	$V_0$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(2) + w(2, 3)$	$0 + -1$	$>$	-3	0	$V_0$	0	$V_0$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(2) + w(2, 4)$	$0 + 15$	$>$	0	0	$V_0$	0	$V_0$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(3) + w(3, 5)$	$-3 + 21$	$>$	0	0	$V_0$	0	$V_0$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(4) + w(4, 2)$	$0 + (-2)$	$<$	0	0	$V_0$	-2	$V_4$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(5) + w(5, 1)$	$0 + 8$	$>$	0	0	$V_0$	-2	$V_4$	-3	$V_1$	0	$V_0$	0	$V_0$
$h(5) + w(5, 3)$	$0 + (-4)$	$<$	-3	0	$V_0$	-2	$V_4$	-4	$V_5$	0	$V_0$	0	$V_0$
$h(5) + w(5, 4)$	$0 + 13$	$>$	0	0	$V_0$	-2	$V_4$	-4	$V_5$	0	$V_0$	0	$V_0$

Cuadro 2: Segunda iteración de Bellman-Ford en el algoritmo de Johnson

Fórmula posible nuevo	2ª iteración			Resultados
	Nuevo	$\leq$ $\geq$	Ant.	
$0 \rightarrow 1 \dots 0 \rightarrow 5$	$0 + 0$	$\geq$	$h(x)$	Sin cambios
$h(1) + w(1, 2)$	$0 + 12$	$>$	-2	Sin cambios
$h(1) + w(1, 3)$	$0 + (-3)$	$>$	-4	Sin cambios
$h(2) + w(2, 3)$	$-2 + (-1)$	$>$	-4	Sin cambios
$h(2) + w(2, 4)$	$-2 + 15$	$>$	0	Sin cambios
$h(3) + w(3, 5)$	$-4 + 21$	$>$	0	Sin cambios
$h(4) + w(4, 2)$	$0 + (-2)$	$>$	-2	Sin cambios
$h(5) + w(5, 1)$	$0 + 8$	$>$	0	Sin cambios
$h(5) + w(5, 3)$	$0 + (-4)$	$>$	-4	Sin cambios
$h(5) + w(5, 4)$	$0 + 13$	$>$	0	Sin cambios

Como puede observarse en el Cuadro 1, en las primeras reducciones simplemente se actualiza la distancia estimada hasta cada vértices como 0; esto siempre es así, por definición. <sup>2</sup> Por esto lo hemos marcado en una sola fila (véase figura 2a). En las siguientes reducciones, sólo hay tres en las que se realiza una actualización:

$V_1 \xrightarrow{-3} V_3$  (figura 2b) El vértice  $V_3$  tenía una distancia estimada de 0, que es actualizada a -3 ( que resulta de la suma de la distancia estimada de  $h(V_1) = 0$  y el peso original de la arista  $w(1, 3) = -3$ ).

$V_4 \xrightarrow{-2} V_2$  (figura 2c) La distancia estimada al vértice  $V_2$  era 0, y es actualizada a -2 (por la suma de la  $h(V_4)$  y el peso original de la arista  $w(4, 2) = -2$ ).

$V_5 \xrightarrow{-4} V_3$  (figura 2d) Nos encontramos con otra arista que llega al nodo  $V_3$ . Su distancia estimada ahora se actualiza a -4, ya que  $h(5) + w(5, 4) = 0 + (-4) = -4$  es menor a la distancia estimada la momento (-3).

<sup>2</sup>Como se agregó una arista de peso 0 desde el origen hasta cada nodo, y las mejores distancias se inicializan en infinito excepto el origen mismo.

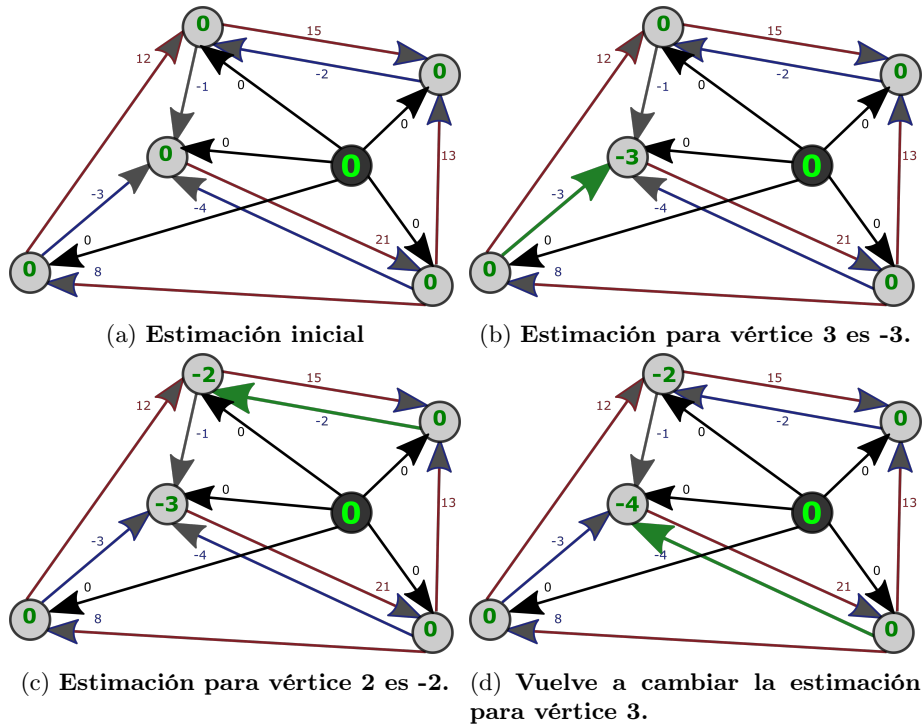


Figura 2: Grafo con cambios mediante Bellman-Ford.

Mientras que en la segunda iteración, si bien los valores de  $h(x)$  son distintos, el resultado de la comparación es el mismo por lo que no hay ningún cambio en ésta ni las siguientes, finalizando exitosamente el algoritmo de Bellman-Ford. En este punto, se crea un nuevo grafo  $H$  posteriormente empleado para calcular los caminos mediante el algoritmo de Dijkstra. Se toma como base el grafo  $G$ , actualizando los pesos de las aristas según la fórmula:

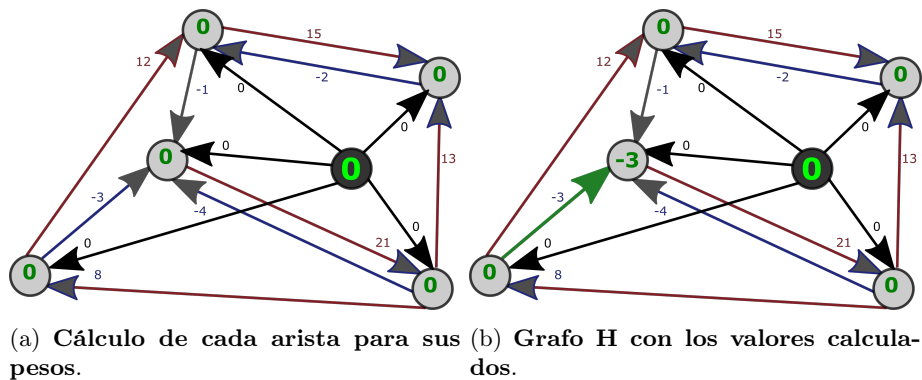
$$\hat{w}(u, v) := w(u, v) + h(u) - h(v)$$

Donde:

$w(u, v)$  es el peso original de la arista, en el grafo  $G$ , desde el vértice  $u$  hasta el vértice  $v$ .

$h(x)$  es la distancia estimada, en el grafo  $G'$ , desde  $V_0$  hasta  $x$ .

$\hat{w}$  es el nuevo peso, en el grafo  $H$ , desde el vértice  $u$  hasta el vértice  $v$ .

Figura 3: Grafo  $H$  con cambios de pesos con  $\hat{w}$ .

### 2.4.2. Aplicar Dijkstra

origen	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>
V <sub>1</sub>	0	14	1	27	18
V <sub>2</sub>	26	0	1	13	18
V <sub>3</sub>	25	30	0	30	17
V <sub>4</sub>	26	0	1	0	18
V <sub>5</sub>	8	13	0	13	0

(a) Distancia mínima

origen	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>
V <sub>1</sub>	-	V <sub>1</sub>	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>
V <sub>2</sub>	V <sub>5</sub>	-	V <sub>2</sub>	V <sub>2</sub>	V <sub>3</sub>
V <sub>3</sub>	V <sub>5</sub>	V <sub>4</sub>	-	V <sub>5</sub>	V <sub>3</sub>
V <sub>4</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>2</sub>	-	V <sub>3</sub>
V <sub>5</sub>	V <sub>5</sub>	V <sub>4</sub>	V <sub>5</sub>	V <sub>5</sub>	-

(b) Nodo antecesor a cada destino

Cuadro 3: Resultados al aplicar Dijkstra

### 2.4.3. Restaurar valor de distancias

Finalmente, es importante tener en cuenta que el resultado de Dijkstra nos muestra los caminos mínimos, pero con las distancias alteradas  $\hat{\delta}(u, v)$  (basadas en los pesos  $\hat{w}$ ). Entonces un último paso, para cada iteración de Dijkstra (por cada vértice de origen), es restaurar la verdadera distancia hasta cada vértice destino  $v$ :

$$d(u, v) := \hat{w}(u, v) + h(v) - h(u)$$

Tomando entonces  $h(x)$  que calculamos de Bellman-Ford, actualizamos (cuadro 4) cada distancia mínima de cuadro 3a para obtener la tabla cuadro 5 de distancias mínimas reales.

origen	$h(V_1)=0$	$h(V_2)=-2$	$h(V_3)=-4$	$h(V_4)=0$	$h(V_5)=0$
$h(V_1)=0$	0 +0-0	14+(-2)-0	1+(-4)-0	27 <del>+0-0</del>	18 <del>+0-0</del>
$h(V_2)=-2$	26 +0 <del>/(-2)</del>	0 <del>+(-2)-(-2)</del>	1 + (-4) <del>/(-2)</del>	13 +0 <del>/(-2)</del>	18 +0 <del>/(-2)</del>
$h(V_3)=-4$	25 +0 <del>/(-4)</del>	30 + (-2) <del>/(-4)</del>	0 <del>+(-4)-(-4)</del>	30 +0 <del>/(-4)</del>	17 +0 <del>/(-4)</del>
$h(V_4)=0$	26 <del>+0-0</del>	0 +(-2)-0	1+(-4)-0	0 <del>+0-0</del>	18 <del>+0-0</del>
$h(V_5)=0$	8 <del>+0-0</del>	13+(-2)-0	0+(-4)-0	13 <del>+0-0</del>	0 <del>+0-0</del>

Cuadro 4: Cálculos de ajuste de distancias reales

origen	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>
V <sub>1</sub>	0	12	-3	27	18
V <sub>2</sub>	28	0	-1	15	20
V <sub>3</sub>	29	32	0	34	21
V <sub>4</sub>	26	-2	-3	0	18
V <sub>5</sub>	8	11	-4	13	0

Cuadro 5: Valores reales de distancias mínimas

### 2.4.4. Reconstrucción caminos mínimos

Para cualquier (origen, destino), a partir del cuadro 3b podemos (recursivamente desde destino al origen) reconstruir el camino mínimo, sabiendo que su peso es el calculado en el cuadro 5. A continuación en la figura figura 4, se muestran a modo de comprobación los caminos mínimos según el vértice de origen:

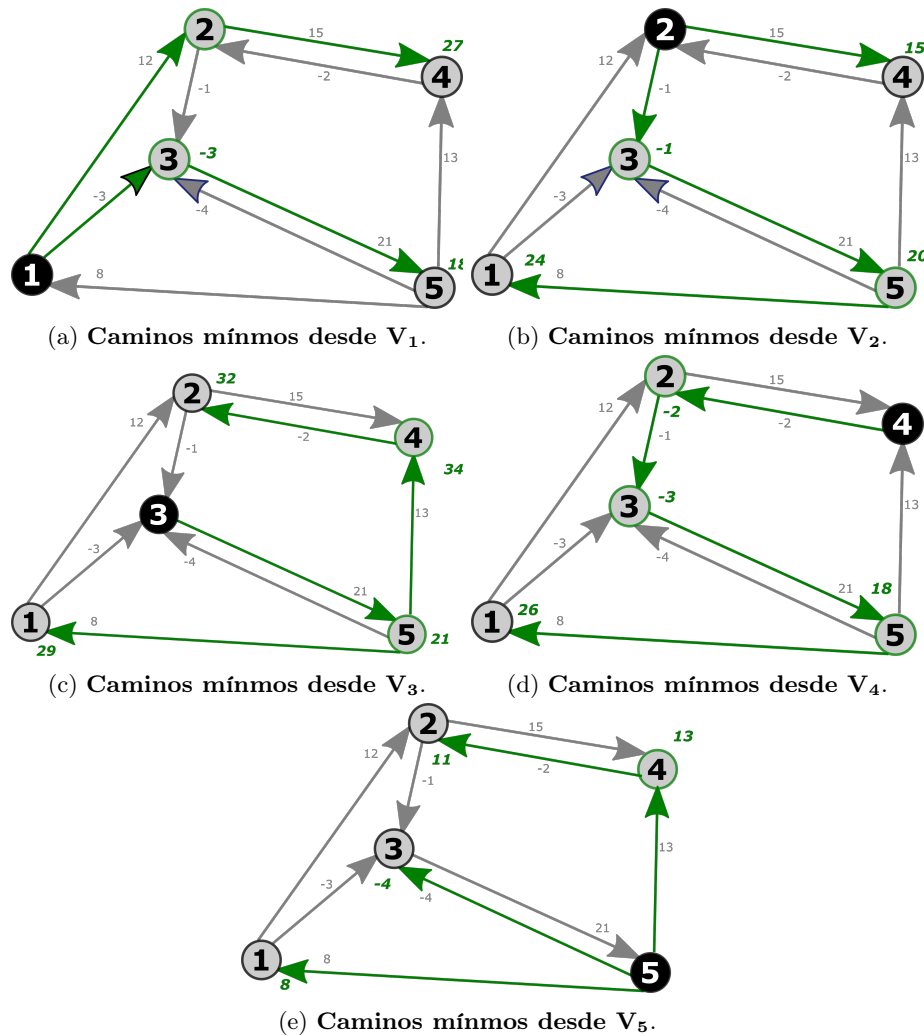


Figura 4: **Grafo**  $H$  con caminos mínimos (en verde, junto con la distancia) según el punto de origen.

## 2.5. Metodología Greedy

### Enunciado 1.5

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

## 2.6. Programación dinámica

### Enunciado 1.6

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

## 2.7. Programa con solución

Enunciado 1.7

Programar la solución usando el algoritmo de Johnson.

### 3. Parte 2: Un poco de teoría

#### 3.1. Enunciado

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  $O(PD) < O(G)$ . 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.

---



### 3.2. Fromas de resolución de problemas

#### Enunciado 2.1

Hasta el momento hemos visto 3 formas distintas de resolver problemas. Greedy, división y conquista y programación dinámica.

1. Describa brevemente en qué consiste cada una de ellas.
2. Identifique similitudes, diferencias, ventajas y desventajas entre las mismas. ¿Podría elegir una técnica sobre las otras?

### 3.3. Caso teórico puntual

#### Enunciado 2.2

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  $O(PD) < O(G)$ . Sabemos que tenemos limitaciones en nuestros recursos computacionales (CPU y principalmente memoria). ¿Qué algoritmo elegiría para resolver el problema?