Trabajo Práctico 2

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

Grupo:	404
Repositorio:	github.com/lucashemmingsen/7529tp2
Entrega:	nº 1 (20/10/2021)

Integrantes del grupo 404

Padrón	Apellido, Nombre	Email
76187	Xxxxxxxxxx, Xxxxx Xxxxx	xxxxxxxxxxx@fi.uba.ar
97529	Xxxxxxxx, Xxxxx Xxxxxxx	xxxxxxxxx@fi.uba.ar
102593	Xxxxxx, Xxxxx Xxxxx	xxxxxxx@fi.uba.ar
102649	Xxxxxx, Xxxxxxx Xxxxx	xxxxxxx@fi.uba.ar
106713	Xxxx Xxxxx, Xxxxx	xxxxx@fi.uba.ar

Índice

I. Introducción **2**
I.1. Resumen . . . 2
I.2. Lineamientos básicos . . . 2

P1. Parte 1: Minimizando costos **3**
a. Enunciado . . . 3
b. Formato de los archivos . . . 3
1. Algoritmo de Johnson: funcionamiento . . . 4
2. Comparación de algoritmos: Complejidad . . . 4
2.1. Algoritmo de Bellman-Ford . . . 4
2.2. Algoritmo de Floyd-Warshall . . . 4
2.3. Algoritmo de Johnson . . . 4
3. Comparación de algoritmos: Ventajas y desventajas . . . 5
4. Algoritmo de Johnson: resolución manual . . . 6
4.1. Convertir a grafo sin negativos (Bellman-Ford) . . . 6
4.2. Aplicar Dijkstra . . . 9
4.3. Restaurar valor de distancias . . . 9
4.4. Reconstrucción caminos mínimos . . . 9
5. Metodología Greedy . . . 11
6. Programación dinámica . . . 12
7. Programa con solución . . . 13

P2. Parte 2: Un poco de teoría **14**
a. Enunciado . . . 14
1. Fromas de resolución de problemas . . . 15
2. Caso teórico puntual . . . 16

I. Introducción

I.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 Johnsosn; así como también respuestas a preguntas teóricas.

I.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

P1. Parte 1: Minimizando costos

a. 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.

b. 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 en que ciudad colocar el depósito. Además imprimir en forma de matriz los costos mínimos entre cada uno de los depósitos.

1. Algoritmo de Johnson: funcionamiento

Enunciado 1.1

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

El algoritmo de Johnson determina los caminos minimales entre todos los pares de vértices de un grafo dirigido. Es preferible aplicarlo sobre grafos de escasas aristas, ya que los procesa con mayor rapidez que otros algoritmos de símil uso.

Su funcionamiento se basa en reponderaciones del grafo. Además, utiliza los algoritmos de Djikstra y Bellman-Ford como subrutinas.

La reponderación se ejecuta condicionalmente. Si todos los pesos de las aristas son positivos, no es necesaria una reponderación, basta con ejecutar Djikstra una vez por vértice y devolviendo el resultado final. En cambio, si el grafo (G) tiene aristas de peso negativo, pero no ciclos negativos, computamos un nuevo set de aristas con pesos no negativos que nos permita utilizar Djikstra, como en el caso anterior. A este proceso de cómputo de nuevas aristas se lo llama reponderación, o cambio de peso.

El set de aristas obtenido de la reponderación debe cumplir dos propiedades:

I Se preservan los caminos más cortos.

II No hay aristas de peso negativo.

El proceso de reponderación tiene un costo de $O(V \times E)$.

Dado que la reponderación asegura que no quedan aristas de peso negativo, se procesa por Djikstra como en el caso anterior. El último caso se da cuando el grafo tiene un ciclo negativo.

En instancias de este tipo, el algoritmo reporta que el grafo contiene dicho ciclo y termina.

2. Comparación de algoritmos: Complejidad

Enunciado 1.2

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

Algoritmo	Temporal	Espacial
Bellman-Ford	$O(V^2 \times E)$	
Floyd-Warshall	$\Theta(V^3)$	$O(V^2)$
Johnson	$O(V^2 lg(V) + V \times E)$	

Cuadro 1: Complejidades. V=vértices, E=aristas.

2.1. Algoritmo de Bellman-Ford

La complejidad temporal es $O(V \times E)$ por vértice de origen; sin embargo a los fines de comparación se debe tomar en cuenta la complejidad para analizar todo el grafo (cada vértice como origen), lo que lo convierte en $O(V^2 \times E)$.

2.2. Algoritmo de Floyd-Warshall

2.3. Algoritmo de Johnson

3. Comparación de algoritmos: Ventajas y desventajas

Enunciado 1.3

Analizar en qué situaciones una solución es mejor que otras.

Para grafos densos, el algoritmo de Johnson es mejor ya que su complejidad temporal depende de la cantidad de aristas.

Entonces, cuando el grafo es disperso el tiempo total del algoritmo puede ser menor que el algoritmo de Floyd-Warshall, que resuelve el mismo problema en un tiempo de $O(V^3)$. [1] Además, la complejidad temporal de Johnson es mejor que repetir Bellman-Ford por cada vértice, porque Johnson realiza una sola iteración de Bellman-Ford para elimina los pesos negativos, y luego aplica Dijkstra que tiene una complejidad menor.

[1] Wikipedia (autores varios). «Algoritmo de Johnson »

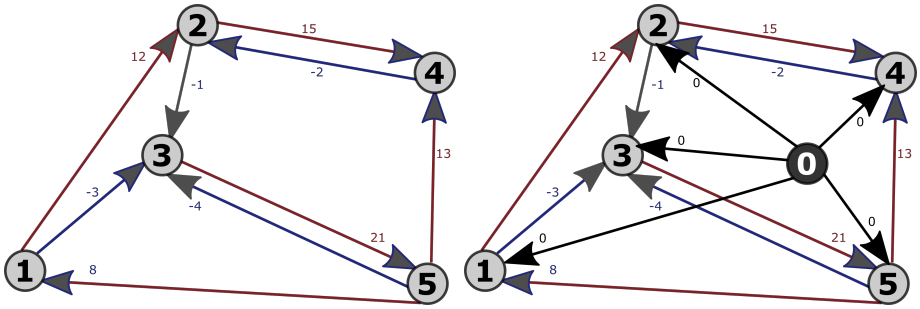
4. Algoritmo de Johnson: resolución manual

Enunciado 1.4

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

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) **Grafo G (original)** (b) **Grafo G' (con nodo temporal 0)**

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. [2]

[2] 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 2: Primera iteración de Bellman-Ford en el algoritmo de Johnson

Fórmula	1ª iteración			V_1		V_2		V_3		V_4		V_5	
posible nuevo	Nuevo	$\lessgtr$	Ant.	h	p	h	p	h	p	h	p	h	p
$0 \to 1 \ldots 0 \to 5$	$0+0$	$<$	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 3: Segunda iteración de Bellman-Ford en el algoritmo de Johnson

Fórmula	2ª iteración			Resultados
posible nuevo	Nuevo	$\lessgtr$	Ant.	
$0 \to 1 \ldots 0 \to 5$	$0+0$	$\geq$	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

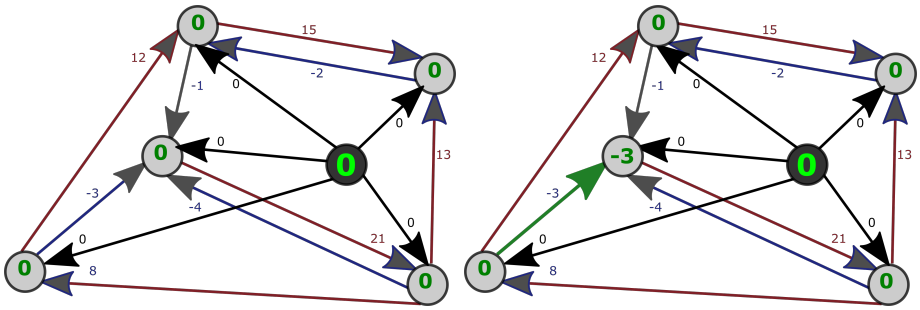
Como puede observarse en el Cuadro 2, en las primeras reducciones simplemente se actualiza la distancia estimada hasta cada vértices como `0`; esto siempre es así, por definición. [3] Por esto lo hemos marcado en una sola fila (véase figura 2a). En las siguientes reducciones, sólo hay tres en las que se realiza una actualización:

$V_1 \overset{-3}{\to} V_3$ (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 $w(1,3) = -3$).

$V_4 \overset{-2}{\to} V_2$ (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 $w(4,2) = -2$).

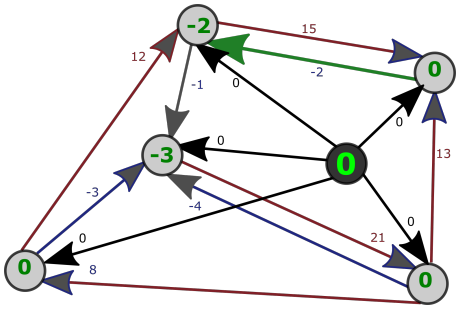
$V_5 \overset{-4}{\to} V_3$ (figura 2d) Nos encontramos con otra arista que llega al nodo V_3. Su distancia estimada ahora se actualiza a -4, ya que $h(5) + w(5,4) = 0 + (-4) = -4$ es menor a la distancia estimada la momento (-3).

[3] 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.

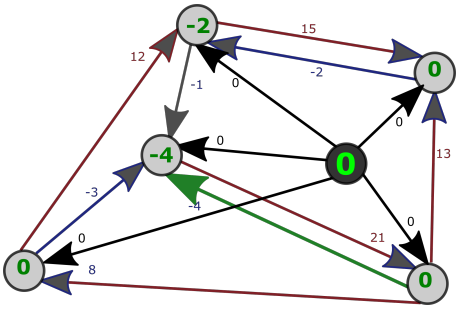


(a) **Estimación inicial**

(b) **Estimación para vértice 3 es -3.**



(c) **Estimación para vértice 2 es -2.**



(d) **Vuelve a cambiar la estimación para vértice 3.**

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 siguientesa, 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:

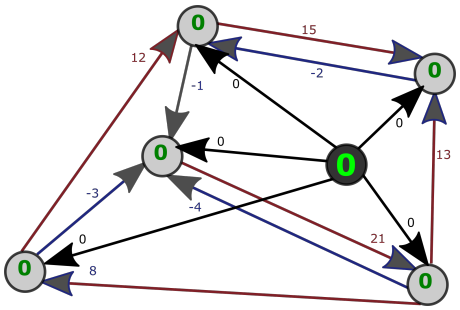
$$\hat{w}(u, v) \ := \ w(u, v) + h(u) - h(v)$$

Donde:

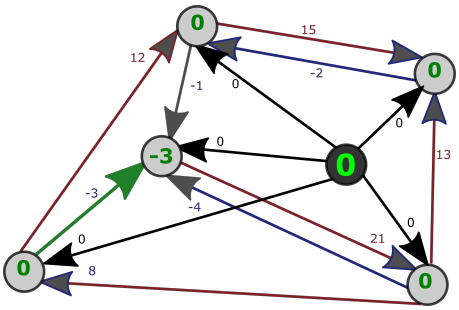
$w(u, v)$ es el peso original de la arista, en el grafo G, desde el vértice u hasta el vértice v.

$h(x)$ 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.



(a) **Cálculo de cada arista para sus pesos.**



(b) **Grafo H con los valores calculados.**

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

4.2. Aplicar Dijkstra

origen	V_1	V_2	V_3	V_4	V_5
V_1	0	14	1	27	18
V_2	26	0	1	13	18
V_3	25	30	0	30	17
V_4	26	0	1	0	18
V_5	8	13	0	13	0

(a) Distancia mínima

origen	V_1	V_2	V_3	V_4	V_5
V_1	-	V_1	V_1	V_2	V_3
V_2	V_5	-	V_2	V_2	V_3
V_3	V_5	V_4	-	V_5	V_3
V_4	V_5	V_4	V_2	-	V_3
V_5	V_5	V_4	V_5	V_5	-

(b) Nodo antecesor a cada destino

Cuadro 4: Resultados al aplicar Dijkstra

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 5) cada distancia mínima de cuadro 4a para obtener la tabla cuadro 6 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 $\not{+0-0}$	18$\not{+0-0}$
$h(V_2)=-2$	$26 + 0 \not{-} (\not{-} 2)$	0 $\not{+(-2)-(-2)}$	$1 + (-4) \not{-} (\not{-} 2)$	$13 + 0 \not{-} (\not{-} 2)$	$18 + 0 \not{-} (\not{-} 2)$
$h(V_3)=-4$	$25 + 0 \not{-} (\not{-} 4)$	$30 + (-2) \not{-} (\not{-} 4)$	0 $\not{+(-4)-(-4)}$	$30 + 0 \not{-} (\not{-} 4)$	$17 + 0 \not{-} (\not{-} 4)$
$h(V_4)= 0$	26 $\not{+0-0}$	0 +(-2)-0	1+(-4)-0	0 $\not{+0-0}$	18 $\not{+0-0}$
$h(V_5)= 0$	8 $\not{+0-0}$	13+(-2)-0	0+(-4)-0	13 $\not{+0-0}$	0 $\not{+0-0}$

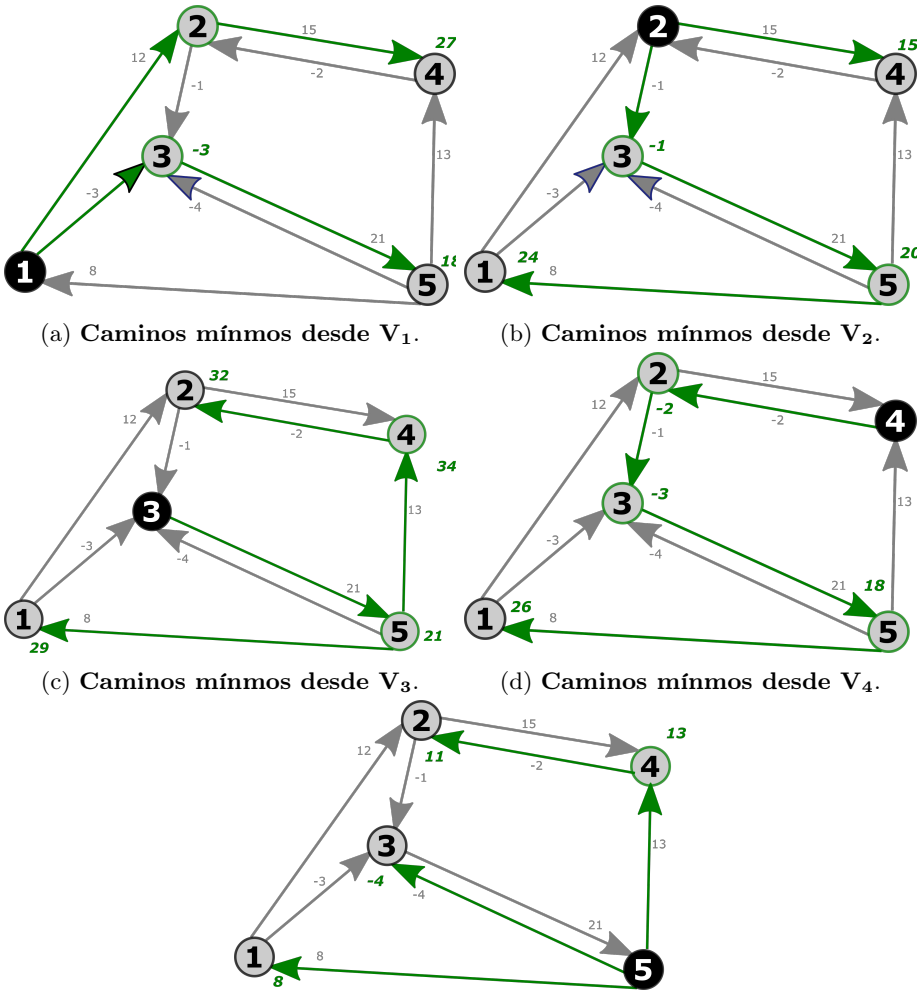
Cuadro 5: Cálculos de ajuste de distancias reales

origen	V_1	V_2	V_3	V_4	V_5
V_1	0	12	-3	27	18
V_2	28	0	-1	15	20
V_3	29	32	0	34	21
V_4	26	-2	-3	0	18
V_5	8	11	-4	13	0

Cuadro 6: Valores reales de distancias mínimas

4.4. Reconstrucción caminos mínimos

Para cualquier `(origen, destino)`, a partir del cuadro 4b podemos (recursivamenta desde `destino` al `origen`) reconstruir el camino mínimo, sabiendo que su peso es el calculado en el cuadro 6. A continuación en la figura figura 4, se muestran a modo de comprobación los caminos mínimos según el vértice de `origen`:



(a) **Caminos mínmos desde V_1.** (b) **Caminos mínmos desde V_2.**

(c) **Caminos mínmos desde V_3.** (d) **Caminos mínmos desde V_4.**

(e) **Caminos mínmos desde V_5.**

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

5. Metodología Greedy

Enunciado 1.5

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

6. Programación dinámica

Enunciado 1.6

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

7. Programa con solución

Enunciado 1.7

Programar la solución usando el algoritmo de Johnson.

P2. Parte 2: Un poco de teoría

a. 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.

1. 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?

2. 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?