Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 13

18/06/2024 – Escuela de Ingeniería de Software

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Modelará una red con grafos, explicará el algoritmo Dijkstra, Floyd-Warshall y resolverá problemas utilizando grafos.**

El algoritmo de Dijkstra

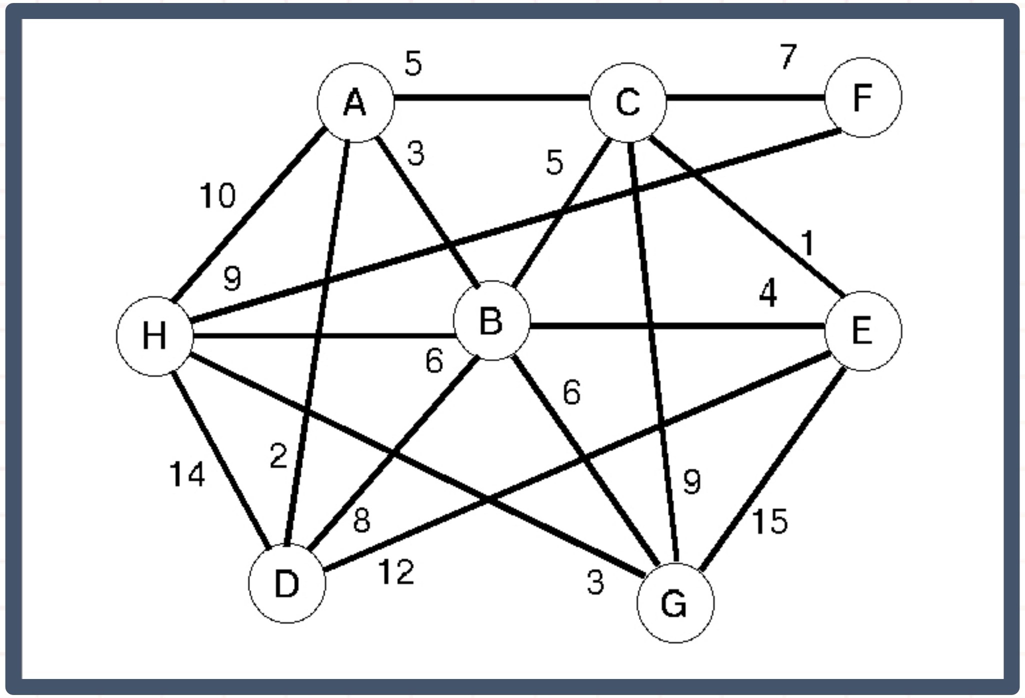
Grafos

01

Encontrado el camino más corto

En la teoría de grafos, el problema del camino más corto es el problema que consiste en encontrar un camino entre dos vértices o nodos, de tal manera que la suma de los pesos de las aristas que lo constituyen sea mínima.

Este problema no necesariamente tiene una única solución. Además, tiene diversas aplicaciones. Un ejemplo es encontrar el camino más rápido para ir de una ciudad a otra en un mapa. En este caso, los vértices representan las ciudades y las aristas las carreteras que las unen, cuya ponderación viene dada por el tiempo que se emplea en atravesarlas



Encontrado el camino más corto

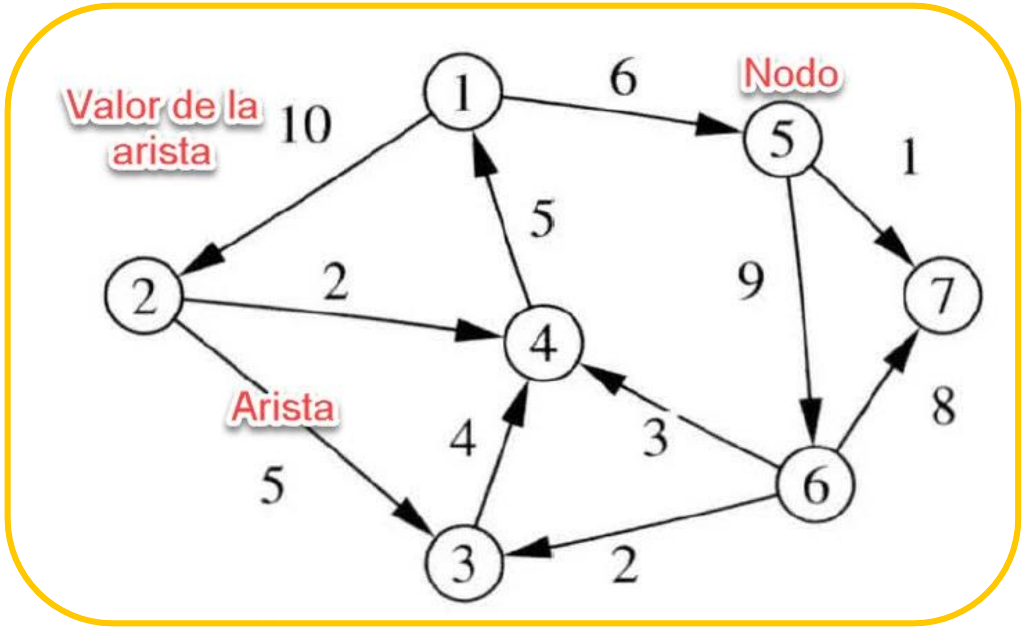
Los algoritmos más importantes para resolver este problema son:

- Algoritmo de Dijkstra, resuelve el problema de los caminos más cortos desde un único vértice origen hasta todos los otros vértices del grafo.
- Algoritmo de Bellman - Ford, resuelve el problema de los caminos más cortos desde un origen si la ponderación de las aristas es negativa.
- Algoritmo de Búsqueda A*, resuelve el problema de los caminos más cortos entre un par de vértices usando la heurística para intentar agilizar la búsqueda.
- Algoritmo de Floyd - Warshall, resuelve el problema de los caminos más cortos entre todos los vértices.

¿Qué es el algoritmo de Dijkstra?

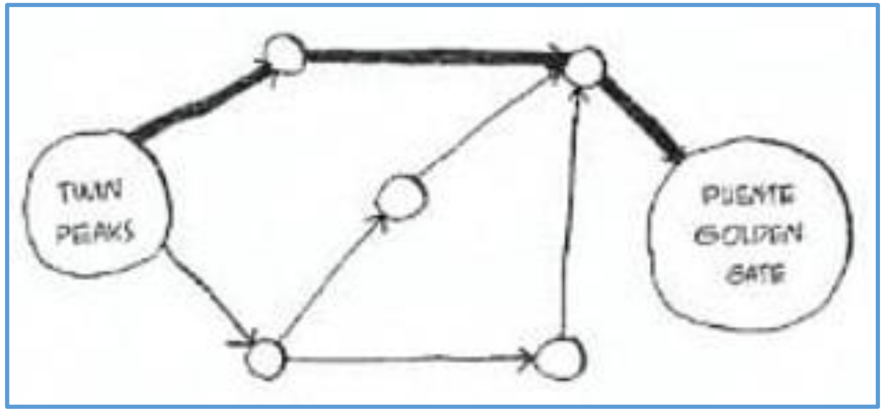
Este algoritmo de búsqueda es denominado tambien algoritmo de caminos mínimos.

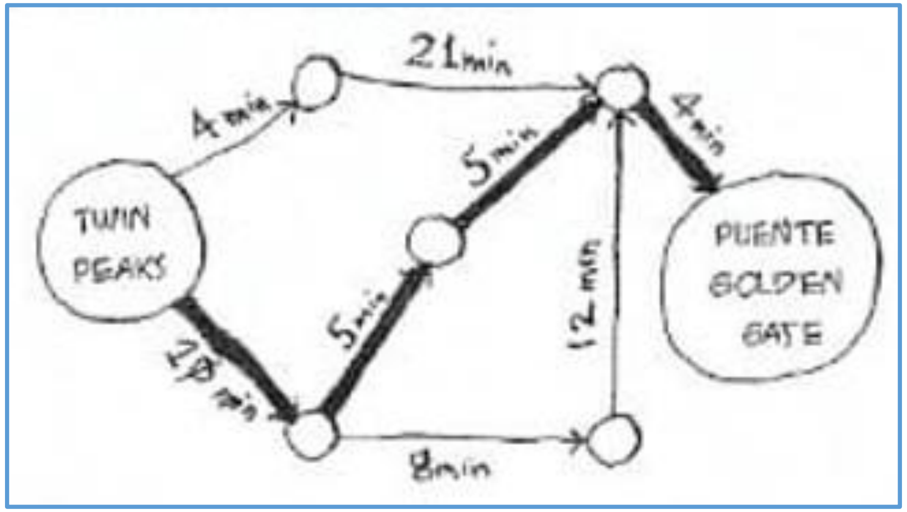
Con el principal objetivo de determinar la ruta más corta desde el nodo origen ,hasta cualquier nodo de la red los cuales tienen pesos en cada arista.



Algoritmo de Dijkstra

El camino con menor número de segmentos no siempre será el camino más rápido. (Búsqueda a lo ancho).



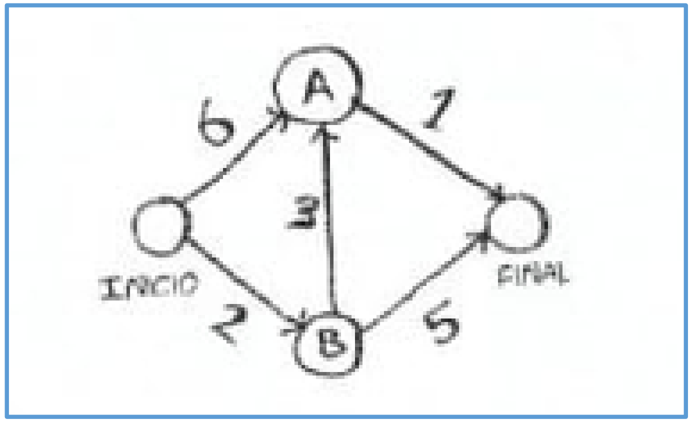


Para hallar el camino más rápido se necesitará el algoritmo de Dijkstra.

Para encontrar el camino más corto en un grafo no ponderado, utiliza búsqueda a lo ancho, para calcular el camino más corto en un grafo ponderado utiliza el algoritmo de Dijkstra. El Algoritmo de Dijkstra sólo funciona en un grafo dirigido acíclico.

Trabajando con el algoritmo de Dijkstra

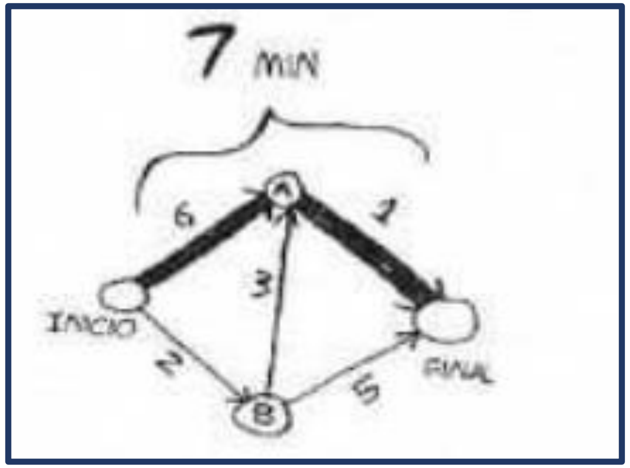
Veamos cómo funciona en este grafo:



Cada segmento muestra el tiempo de viaje en minutos.

Usaremos el algoritmo de Dijkstra para ir del inicio hasta el final en el menor tiempo posible.

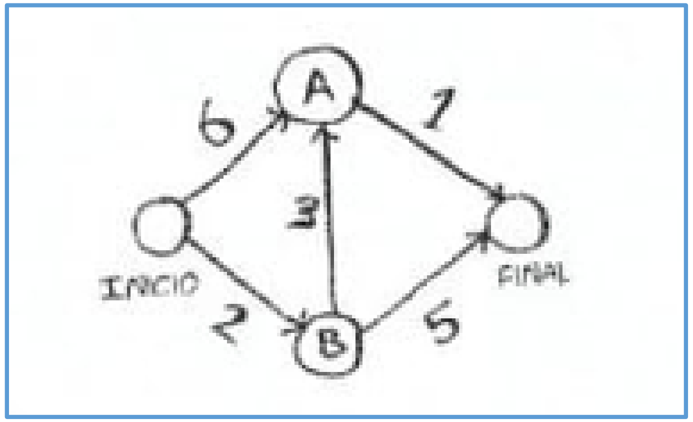
Si se ejecuta la búsqueda a lo ancho, obtenemos:



Pero este camino toma 7 minutos, veamos si podemos encontrar un camino que necesite menos tiempo

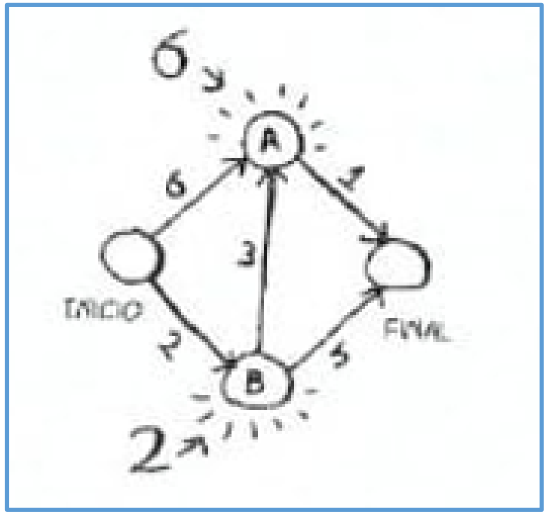
Trabajando con el algoritmo de Dijkstra: Ejercicio

Ir del nodo inicio al nodo final en el menor tiempo posible . Hay 4 pasos en el algoritmo de Dijkstra



PASO 1

Encontrar el nodo más barato. Este es el nodo al que puede llegar en el menor tiempo posible.



Padre
Inicio
Inicio
-

NODO	TIEMPO PARA LLEGAR AL NODO
A	6
B	2
FINAL	∞

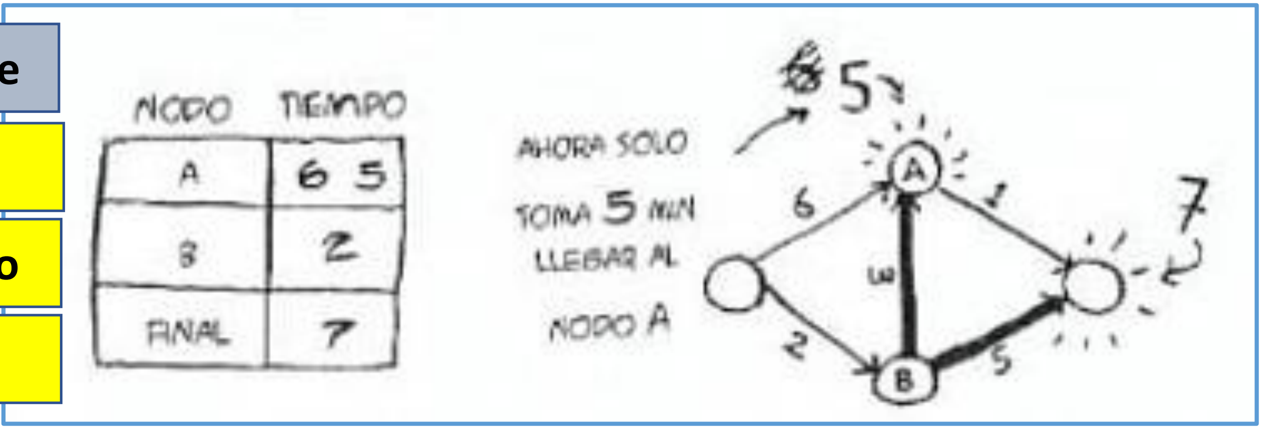
El nodo B es el más cercano está a solo 2 minutos

Trabajando con el algoritmo de Dijkstra: Ejercicio

PASO 2

Actualizar los costos de los nodos vecinos del nodo identificado en el paso 1

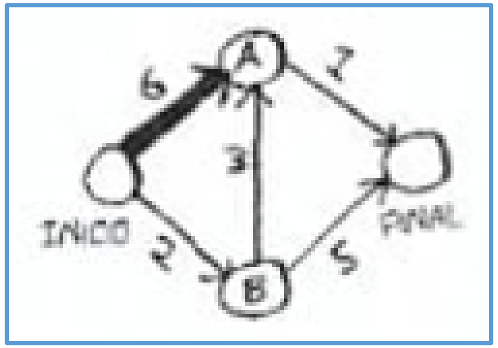
Padre
B
Inicio
B



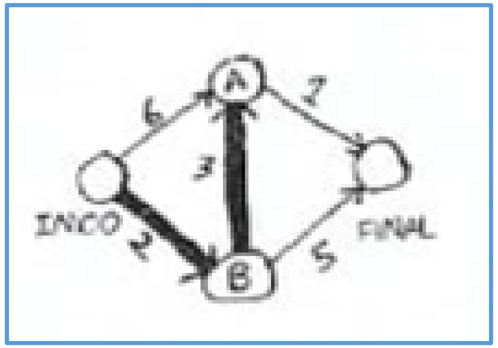
Se calcula cuánto demora en ir a los nodos vecinos de B.

Para ir al nodo A al comienzo se necesitaban 6 minutos pero ahora solo seran 5 minutos. Hay un camino más corto hasta el final (de infinito a 7)

ANTES



DESPUÉS

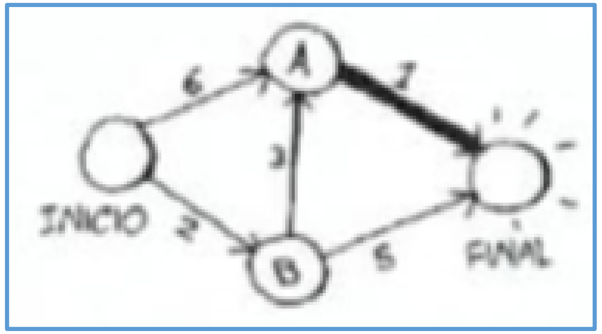


Trabajando con el algoritmo de Dijkstra: Ejercicio

PASO 3

Repetir hasta que hayas analizado cada nodo del grafo

Actualizar vecinos del nodo A



Valores para llegar al nodo A y B

Padre	NODO	TIEMPO
B	A	5
Inicio	B	2
A	FINAL	6

Paso 1 nuevamente: Encuentra el nodo al cual se llega en el menor tiempo posible. Es el nodo A

Paso 2 nuevamente: Actualiza los costos de los vecinos de A. Ahora sólo necesitas 6’ para llegar al final

Trabajando con el algoritmo de Dijkstra: Ejercicio

PASO 4

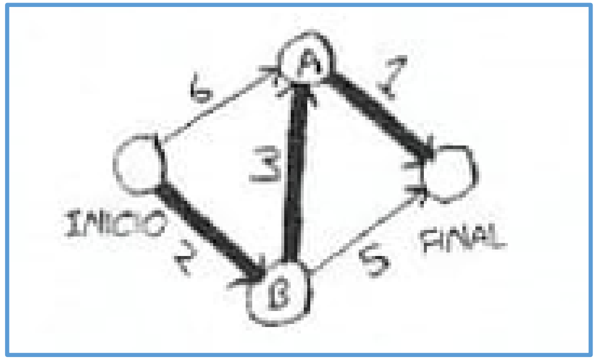
Calcular el camino final

Se guarda el camino final y se hace un conteo del tiempo en llegar al nodo final.

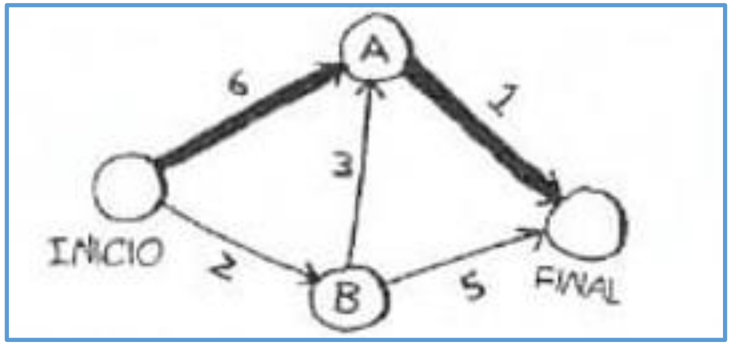
Padre	NODO	TIEMPO
B	A	5
Inicio	B	2
A	FINAL	6

Inicio - B - A - Final

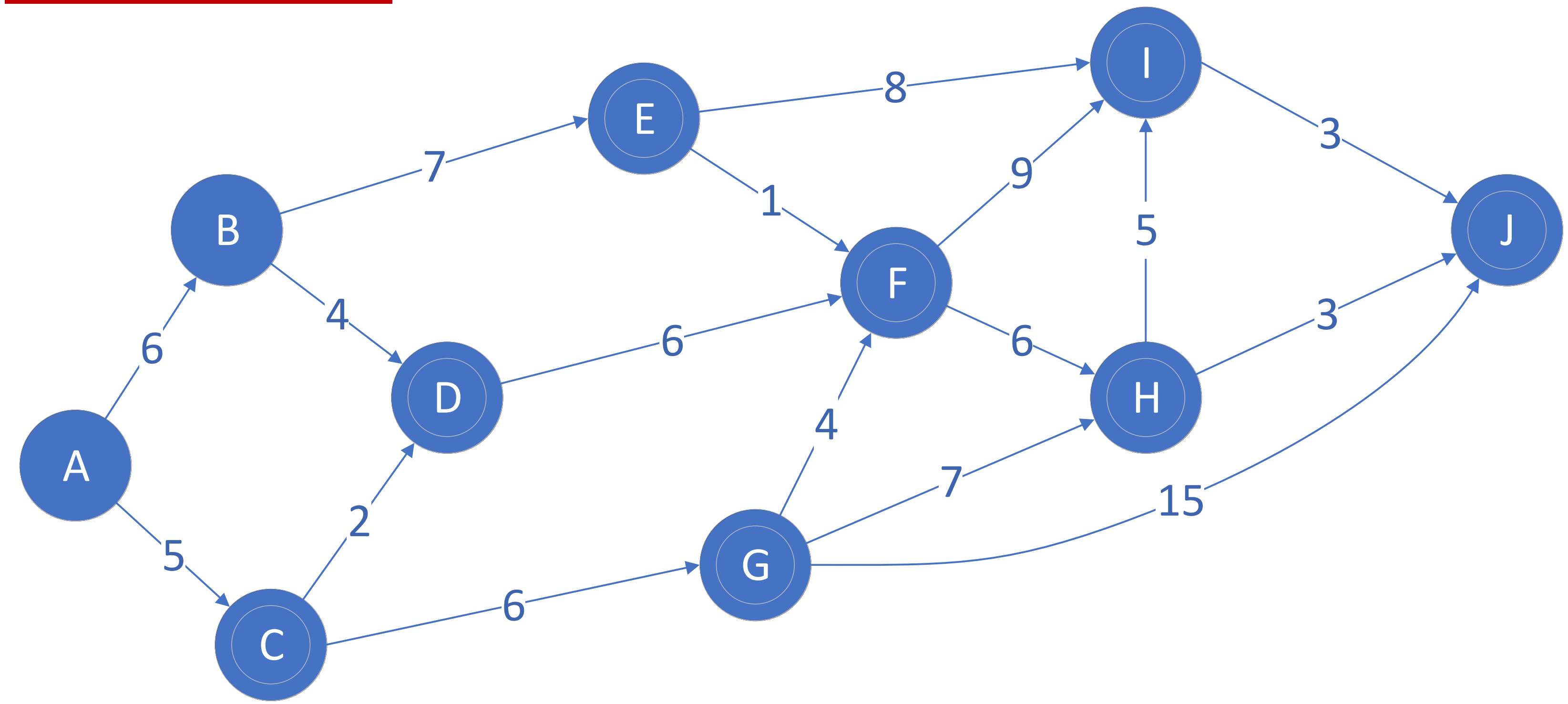
Algoritmo Dijkstra



Algoritmo Búsqueda a lo ancho



Algoritmo Dijkstra – Otra manera



Vértice	Distancia	V. Anterior
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-
G	∞	-
H	∞	-
I	∞	-
J	∞	-

V=[]

NV=[A,B,C,D,E,F,G,H,I,J]

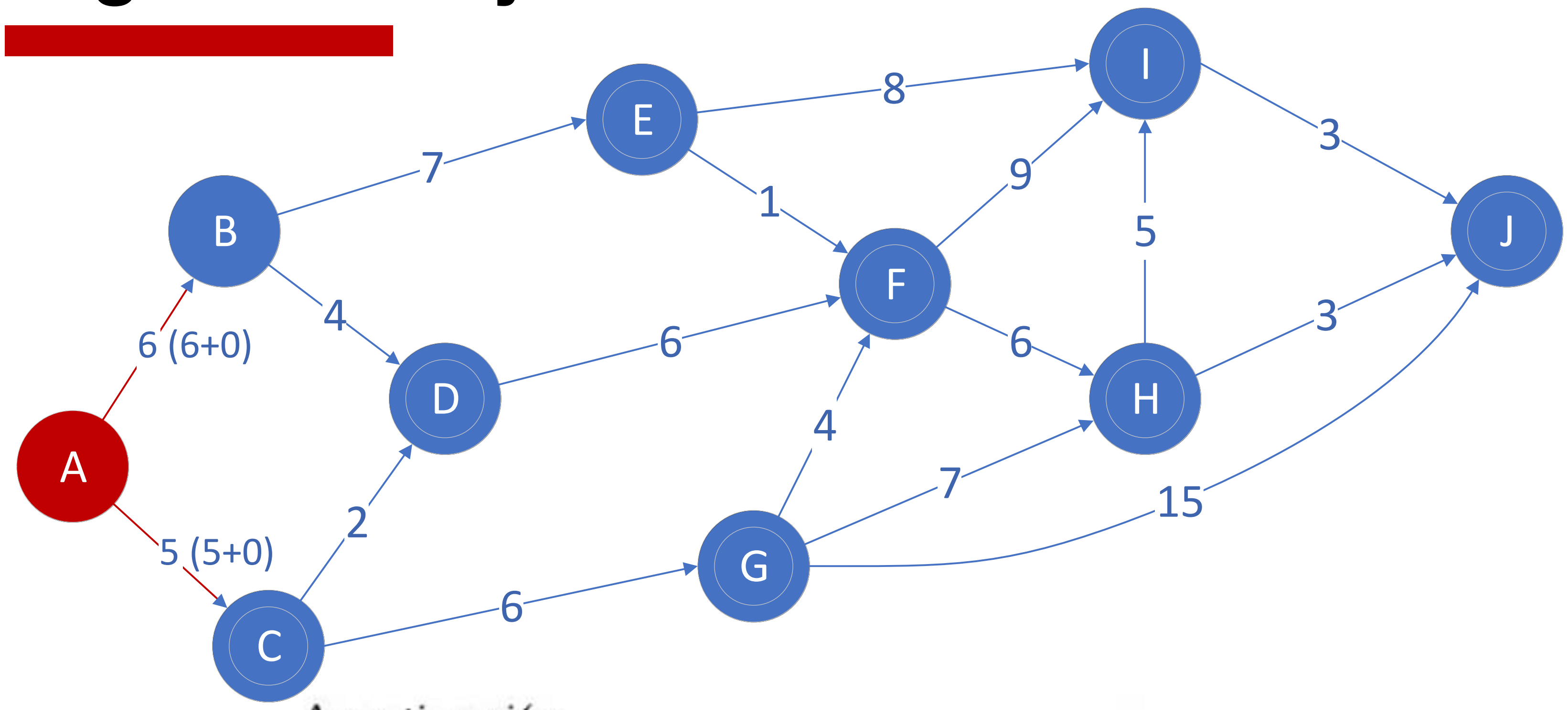
El algoritmo de Dijkstra toma un grafo y un nodo del mismo y calcula el camino mínimo de ese nodo a todos los demás nodos que componen el grafo. Para ello:

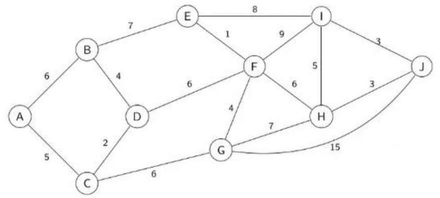
1. se elije el Vértice V sobre el cual se quiera aplicar el algoritmo,
2. se crean dos lista de nodos, una lista de nodos Visitados y otra listas de nodos NO Visitados, que contiene a todos los nodos del grafo.
3. se crea una tabla con 3 columnas, Vértice, Distancia mínima V y el nodo anterior por el cual se llego.
4. Se toma el Vértice V como vertice inicial y se calcula su distancia a sí mismo, que es 0.
5. se actualiza la tabla, en la cual todas las distancias de los demás vértices a V se marcan como infinito.

V: Visitados

NV: No visitados

Algoritmo Dijkstra – Otra manera





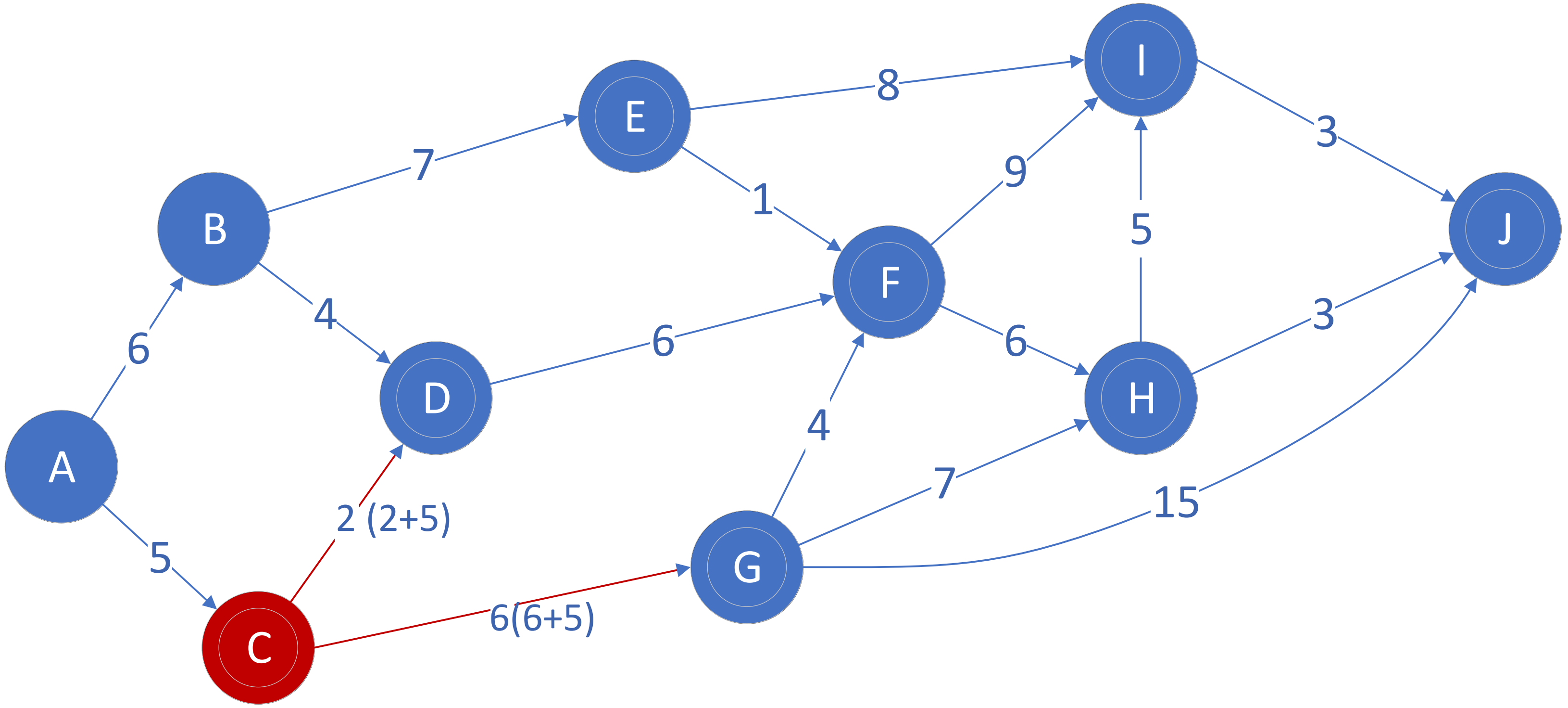
Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	∞	-
E	∞	-
F	∞	-
G	∞	-
H	∞	-
I	∞	-
J	∞	-

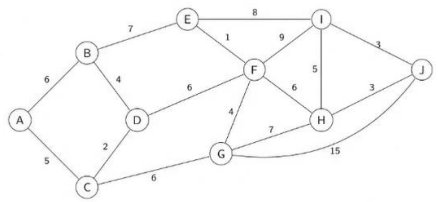
V=[A]
NV=[B,C,D,E,F,G,H,I,J]

A continuación:

- Se visita el vértice NO VISITADO con menor distancia conocida desde el primer vértice, que es A, ya que la distancia de A a A es 0 y las demás infinito.
- Se calcula la distancia entre los vertices sumando los pesos de cada uno con la distancia de A.
- Si la distancia calculada de los vértices conocidos es menor a la que está en la tabla se actualiza y tambien los vértices desde donde se llegó.
- Se pasa el Vertice A a la lista de Vertices visitados.
- Se continua con el vértice no visitado con menor distancia que es C

Algoritmo Dijkstra – Otra manera

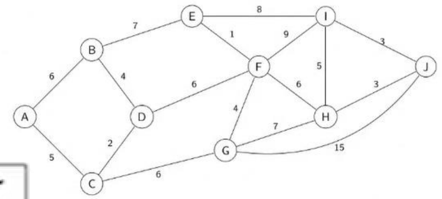


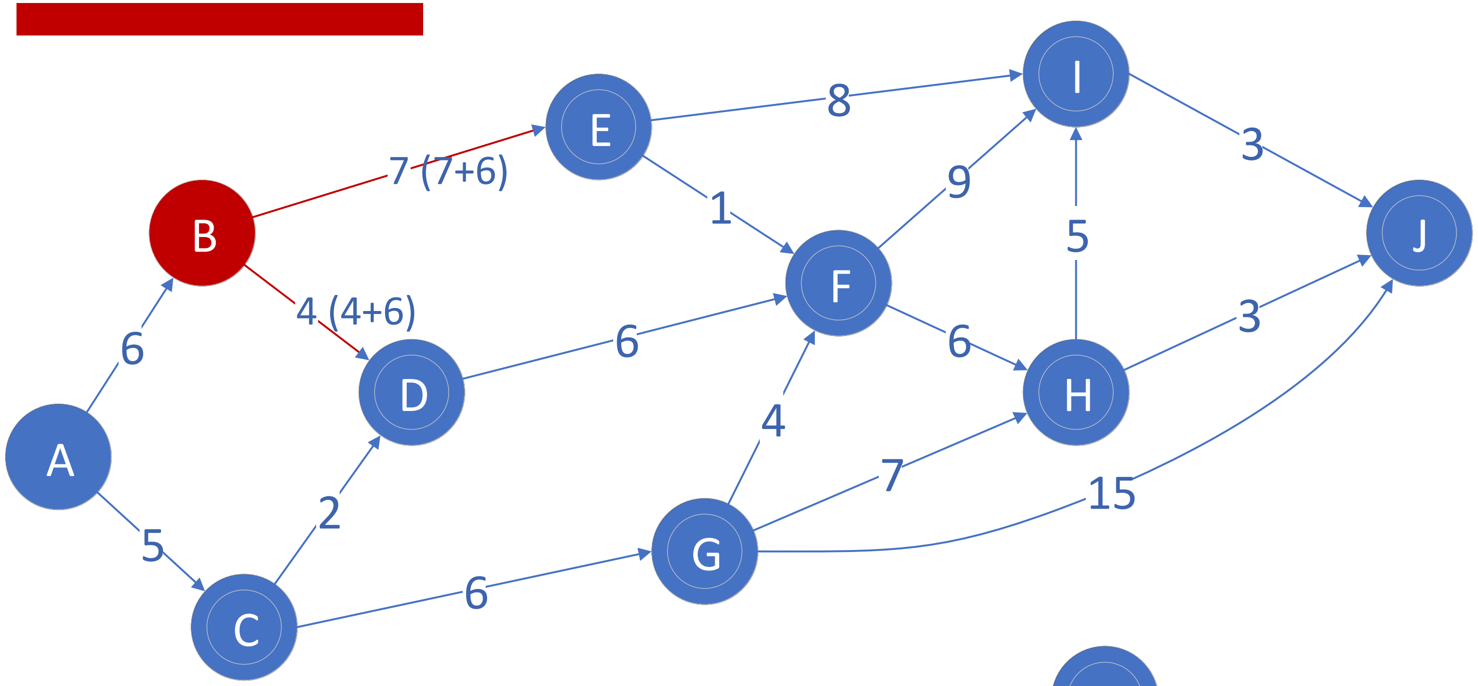


Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	∞	-
F	∞	-
G	11	C
H	∞	-
I	∞	-
J	∞	-

V=[A,C] NV=[B,D,E,F,G,H,I,J]

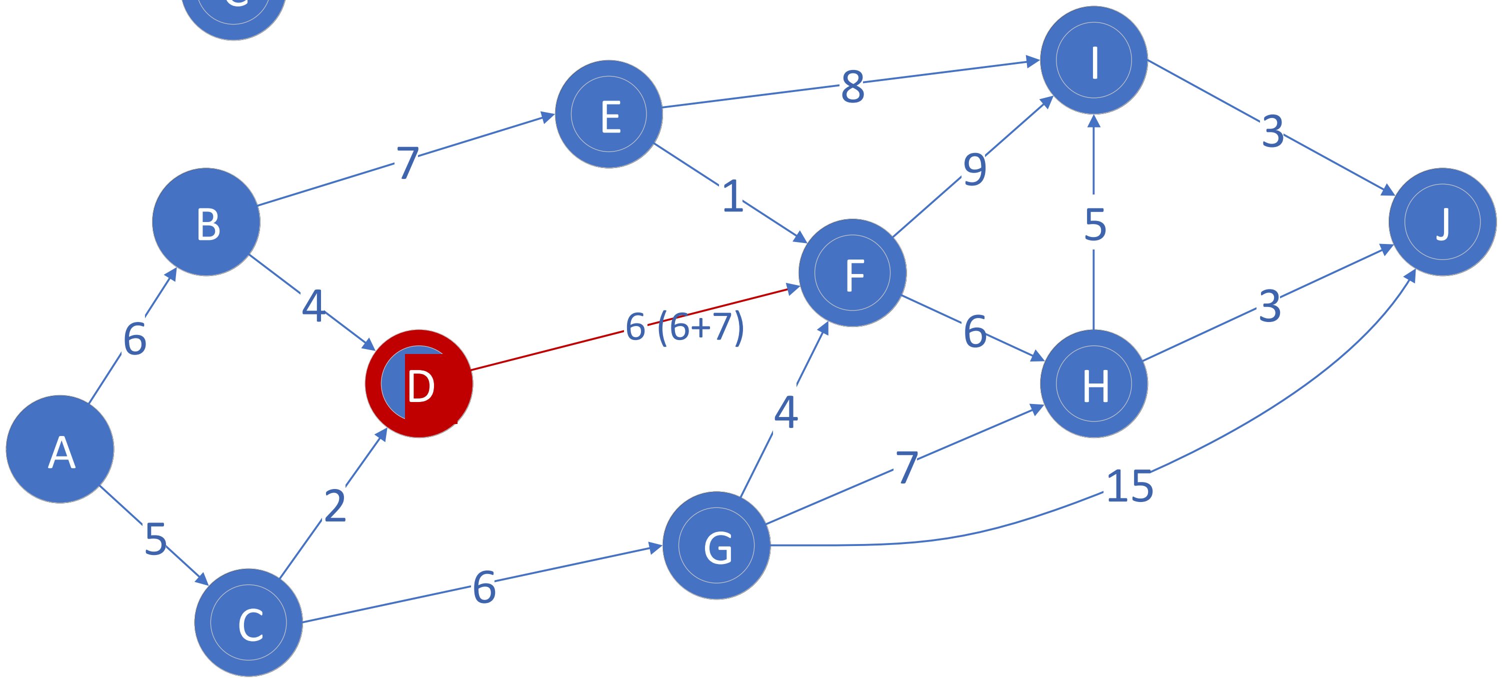
Algoritmo Dijkstra – Otra manera





Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	∞	-
G	11	C
H	∞	-
I	∞	-
J	∞	-

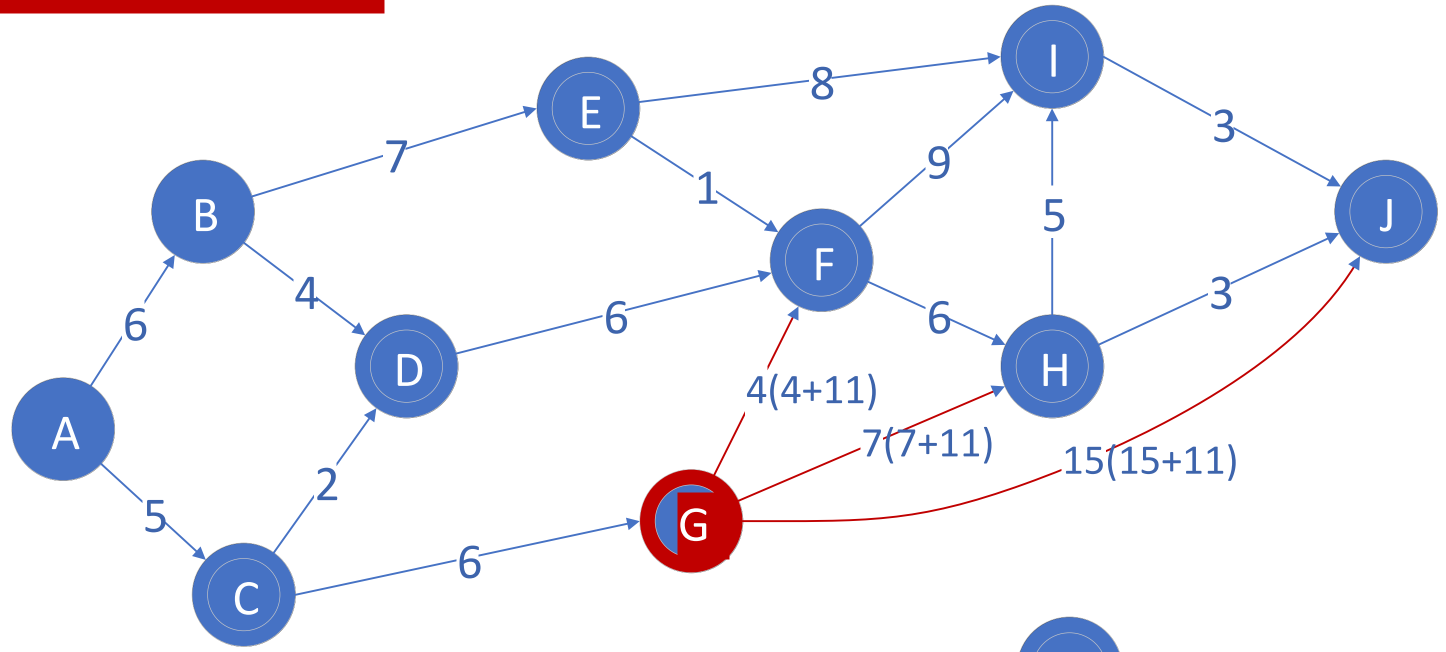
V=[A,C,B] NV=[D,E,F,G,H,I,J]

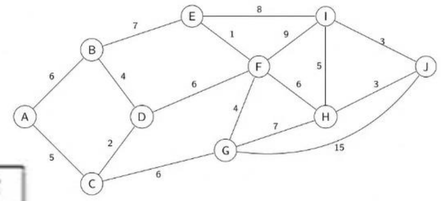


Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	∞	-
I	∞	-
J	∞	-

V=[A,C,B,D] NV=[E,F,G,H,I,J]

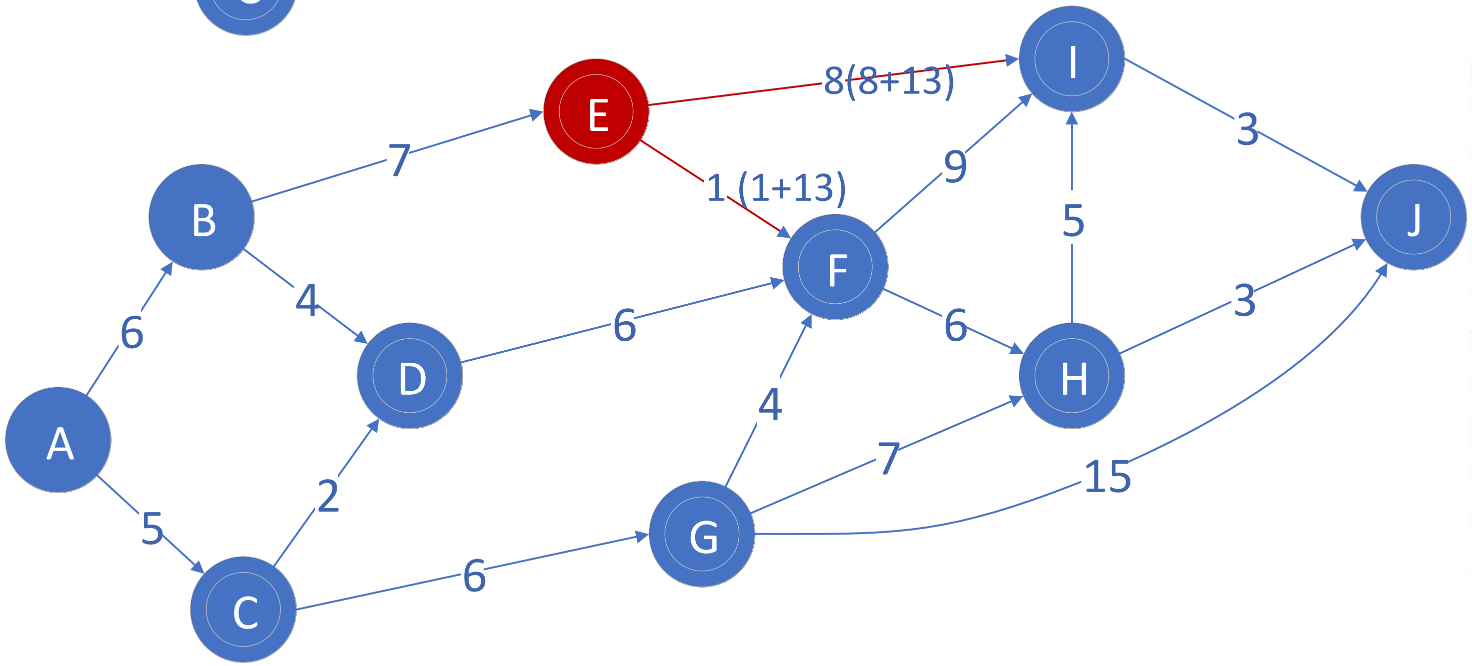
Algoritmo Dijkstra – Otra manera





Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	∞	-
J	26	G

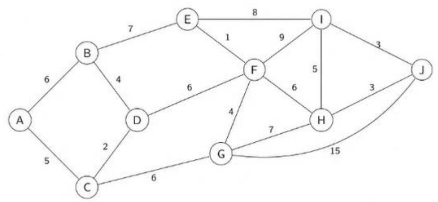
V=[A,C,B,D,G] NV=[E,F,H,I,J]

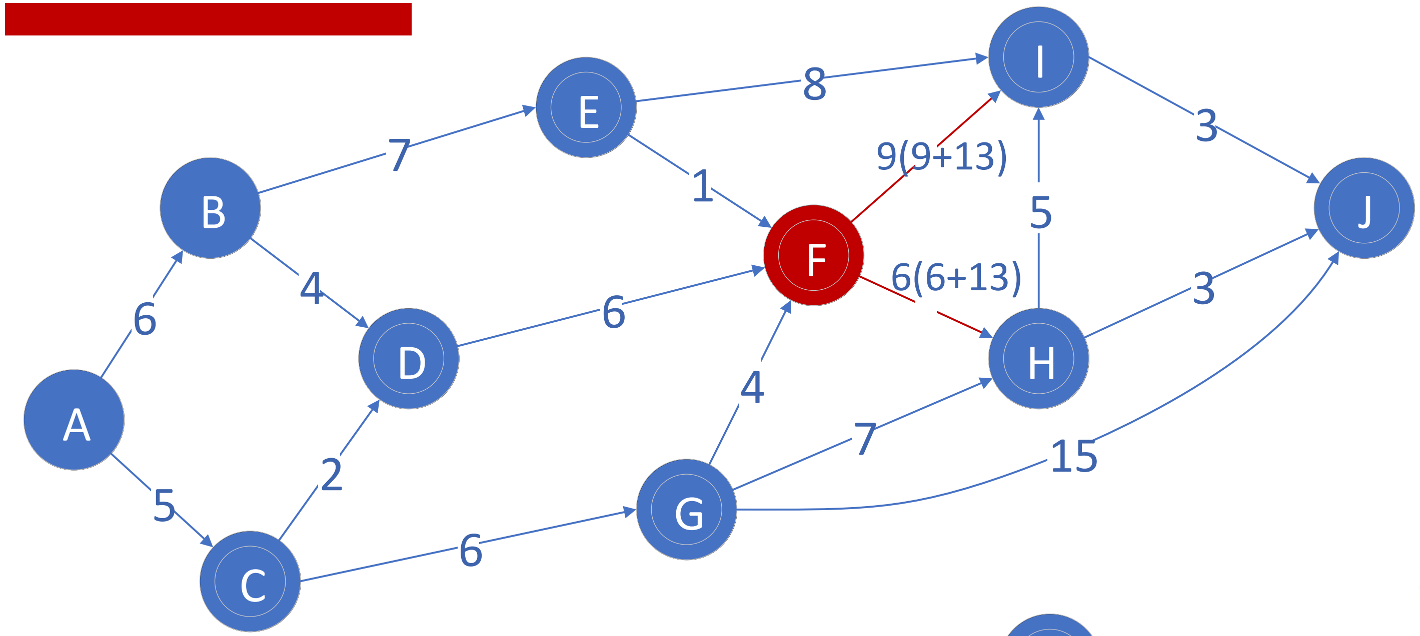


Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	26	G

V=[A,C,B,D,G,E] NV=[F,H,I,J]

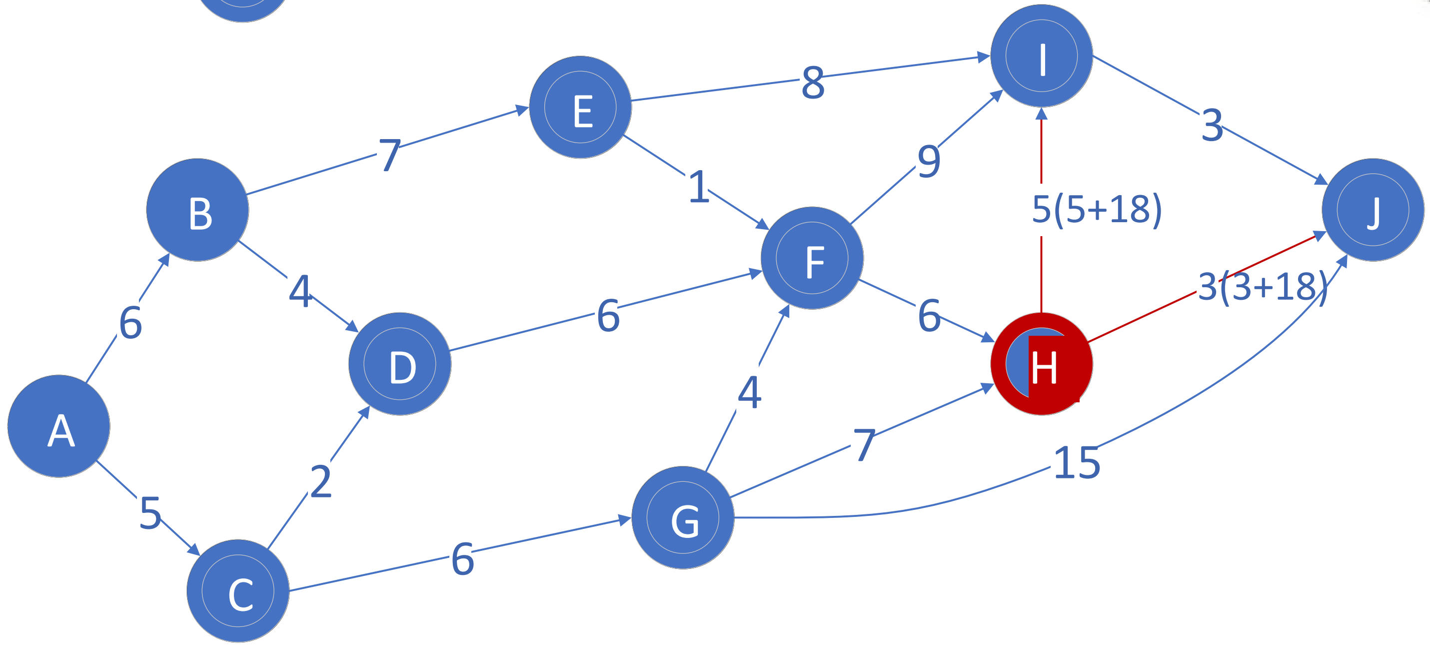
Algoritmo Dijkstra – Otra manera





Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	26	G

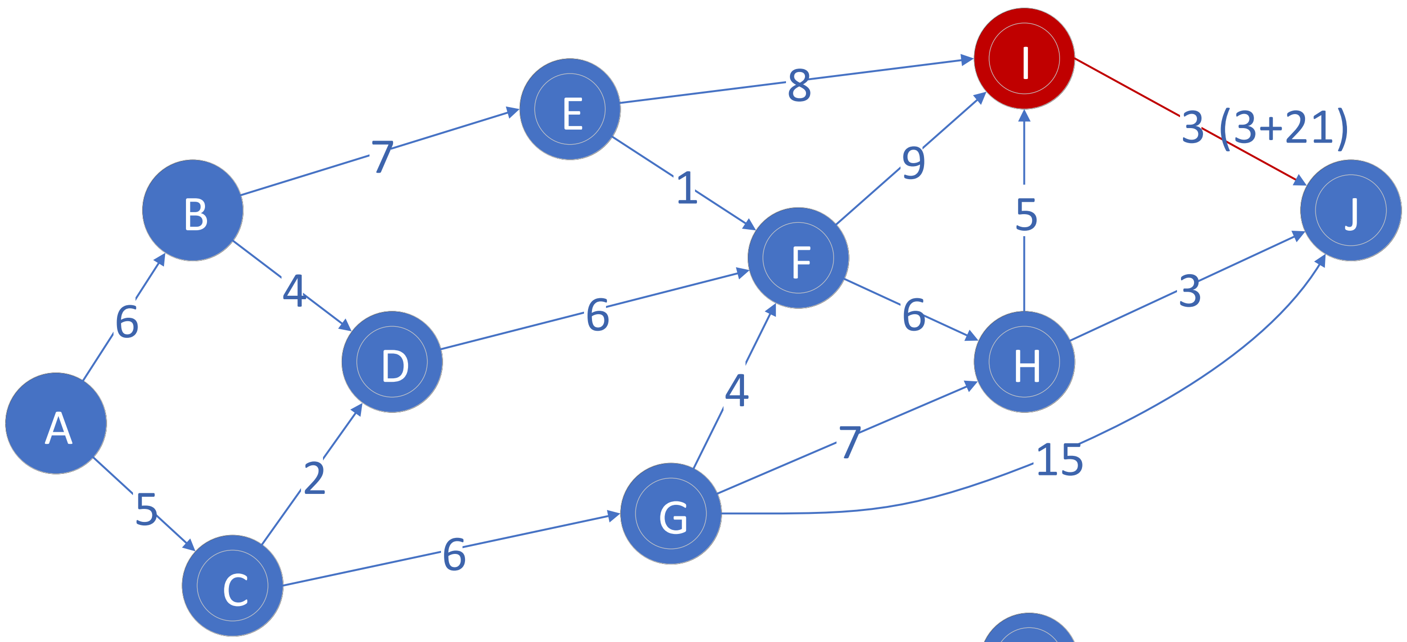
V=[A,C,B,D,G,E,F,] NV=[H,I,J]

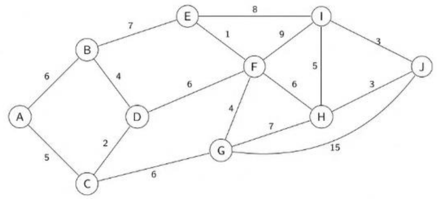


Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

V=[A,C,B,D,G,E,F,H] NV=[I,J]

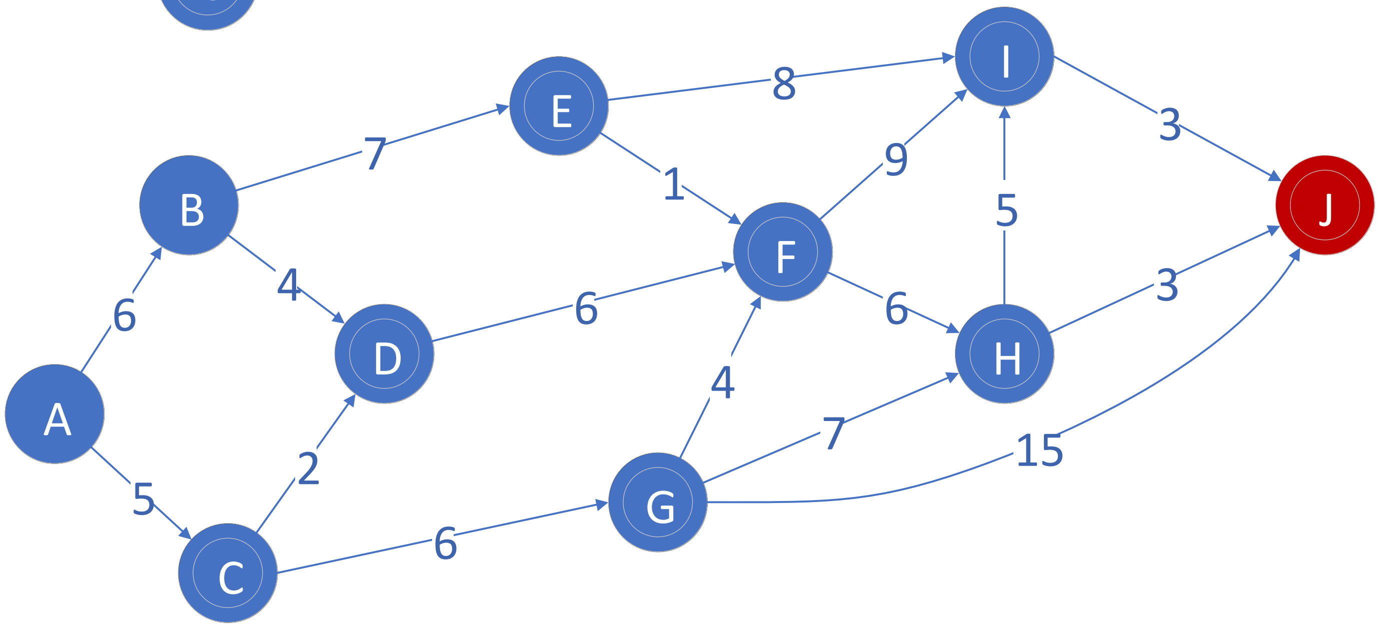
Algoritmo Dijkstra – Otra manera





Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

V=[A,C,B,D,G,E,F,I] NV=[J]



Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

V=[A,C,B,D,G,E,F,I,J] NV=[]

Algoritmo Dijkstra – Otra manera

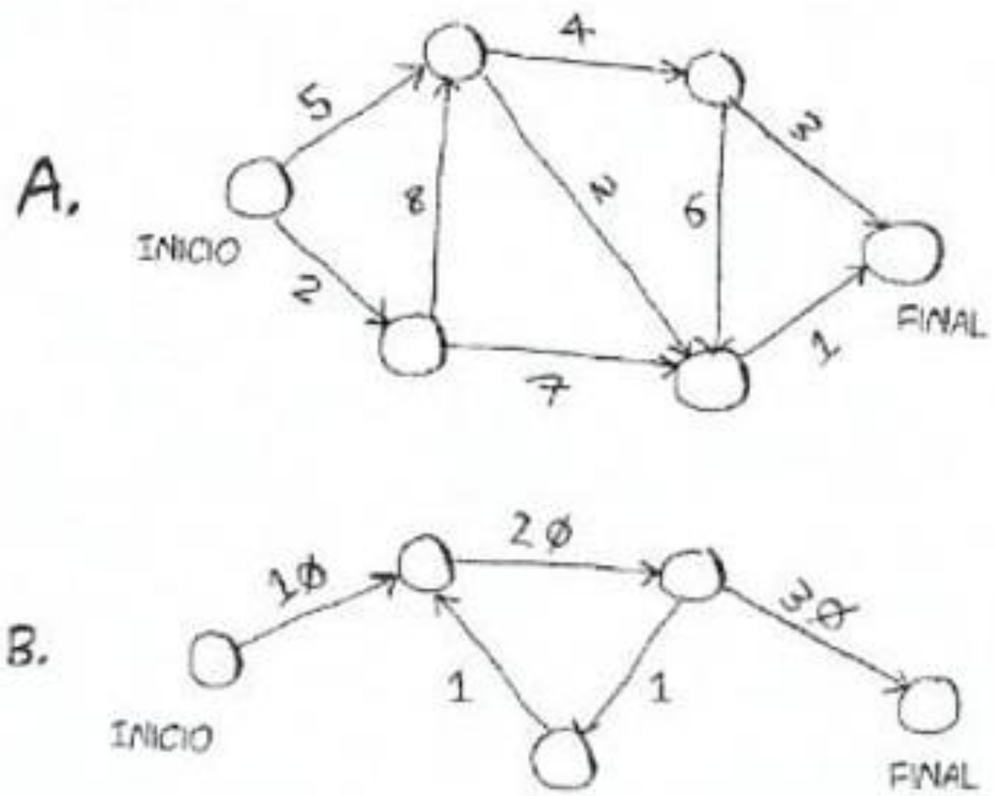
Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

V=[A,C,B,D,G,E,F,I,J] NV=[]

	Camino mínimo	
$A->B$	$A->B$	6
$A->C$	$A->C$	5
$A->D$	$A->C->D$	7
$A->E$	$A->B->E$	13
$A->F$	$A->C->D->F$	13
$A->G$	$A->C->G$	11
$A->H$	$A->C->G->H$	18
$A->I$	$A->B->E->I$	21
$A->J$	$A->C->G->H->J$	21

Ejercicio

- En cada uno de estos grafos ¿Cuál es el peso del camino más corto de principio a fin?



Trabajando con el algoritmo de Dijkstra: Ejercicio

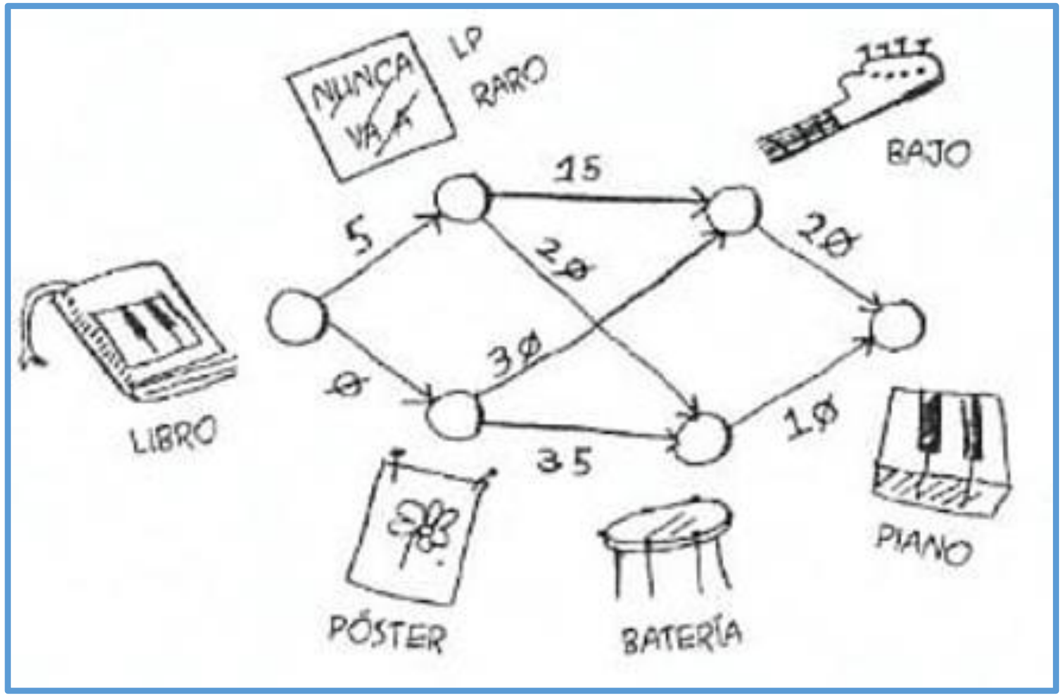
Intercambiando un piano

EJEMPLO

Rama quiere intercambiar su libro de música por un piano.

OFERTAS

- Alex a Rama
  - Poster
  - LP de un artista + $5
- Amy a Alex (Poster o LP)
  - Guitarra
  - Juego de batería
- Beethoven a Amy
  - Piano



Reto: ¿Cómo gastar la menor cantidad de dinero al hacer intercambios?

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

Preparándonos...

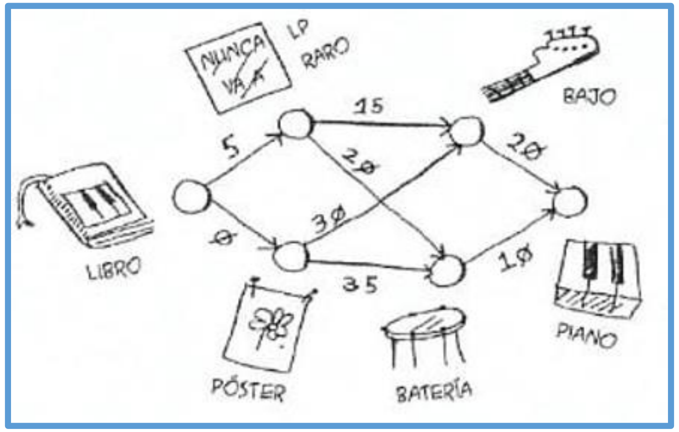


Tabla de costos para cada nodo

NODO	COSTO
LARGA DURACIÓN	5
PÓSTER	0
BAJO	∞
BATERÍA	∞
PIANO	∞

NO HEMOS LLEGADO A ESTOS NODOS DESDE EL PRINCIPIO

Tabla Nodo - Padre

NODO	PADRE
LARGA DURACIÓN	LIBRO
PÓSTER	LIBRO
BAJO	—
BATERÍA	—
PIANO	—

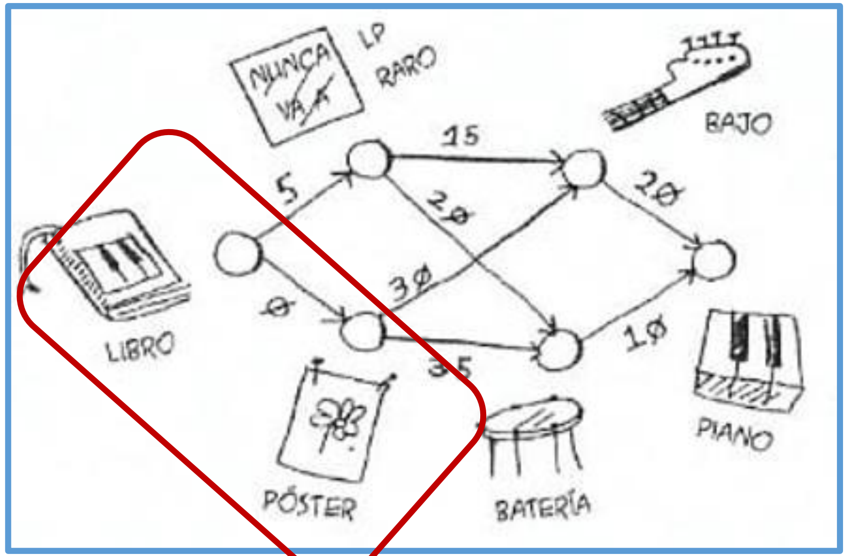
Trabajando con el algoritmo de Dijkstra: Ejercicio

Padre
Libro
Libro
-
-
-

Intercambiando un piano

PASO 1: Encuentra el nodo más barato

El póster es el intercambio más barato: $0



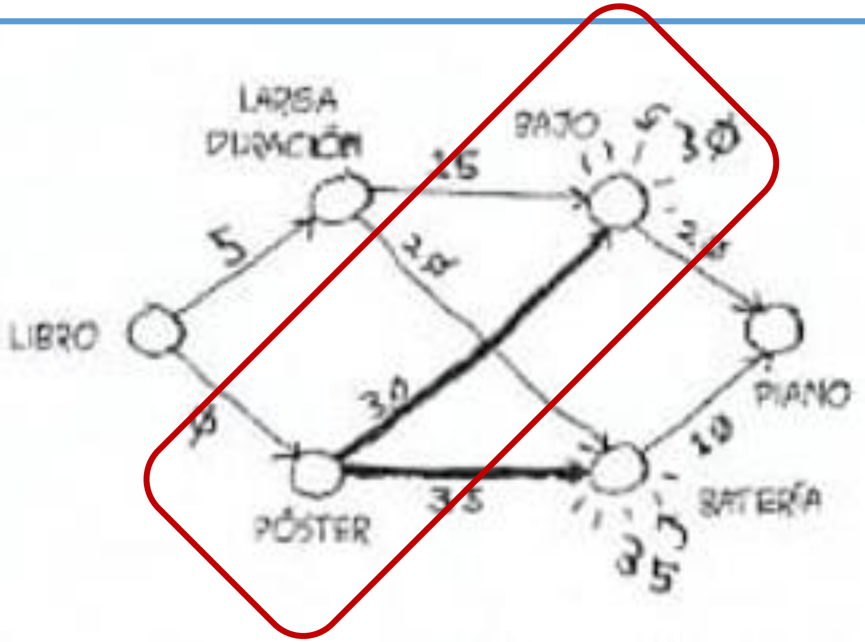
Idea del Alg. Dijkstra:

Mira al nodo más barato del grafo. ¡No existe una forma menos costosa de llegar a ese nodo!

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 2: Encuentra cuánto toma llegar a los vecinos (el costo)



PADRE	NODO	COSTO
LIBRO	LARGA DURACIÓN	5
LIBRO	PÓSTER	0
PÓSTER	BAJO	~~∞~~ 30
PÓSTER	BATERÍA	~~∞~~ 35
—	PIANO	∞

Tienes precios para el bajo y la batería en la tabla. Este valor se definió cuando fuiste a escoger el poster, así que el postre es el padre de ambos. Esto significa que para obtener el bajo, sigues una arista desde el poster, y lo mismo para la batería

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 2: Encuentra cuánto toma llegar a los vecinos (el costo)..continuación

PADRE	NODO	COSTO
LIBRO	LARGA DURACIÓN	5
LIBRO	PÓSTER	0
PÓSTER	BAJO	30
PÓSTER	BATERÍA	35
—	PIANO	∞

VAMOS DESDE "PÓSTER" PARA LLEGAR A ESTOS NODOS

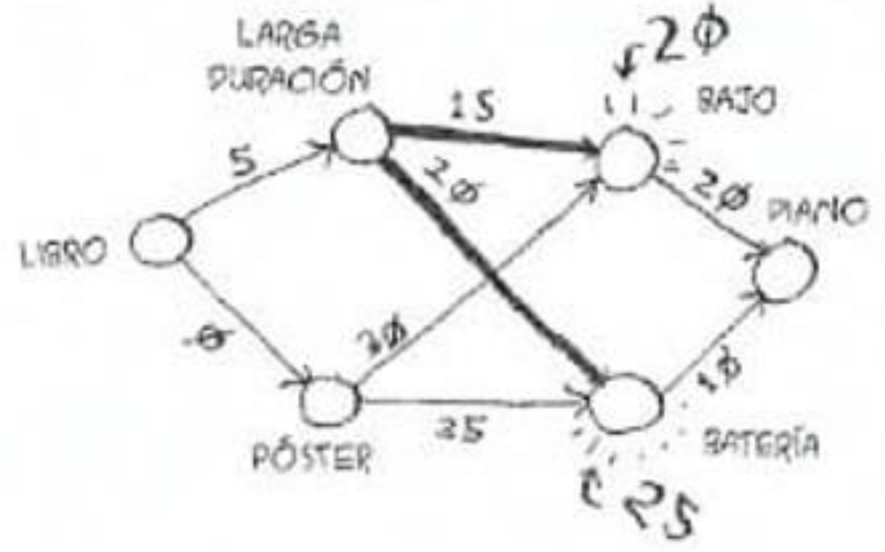
Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

Paso 3: Repites los pasos 1 y 2

PASO 1 NUEVAMENTE: El LP es el próximo nodo más barato, en $5

PASO 2 NUEVAMENTE: Actualiza los valores de sus vecinos



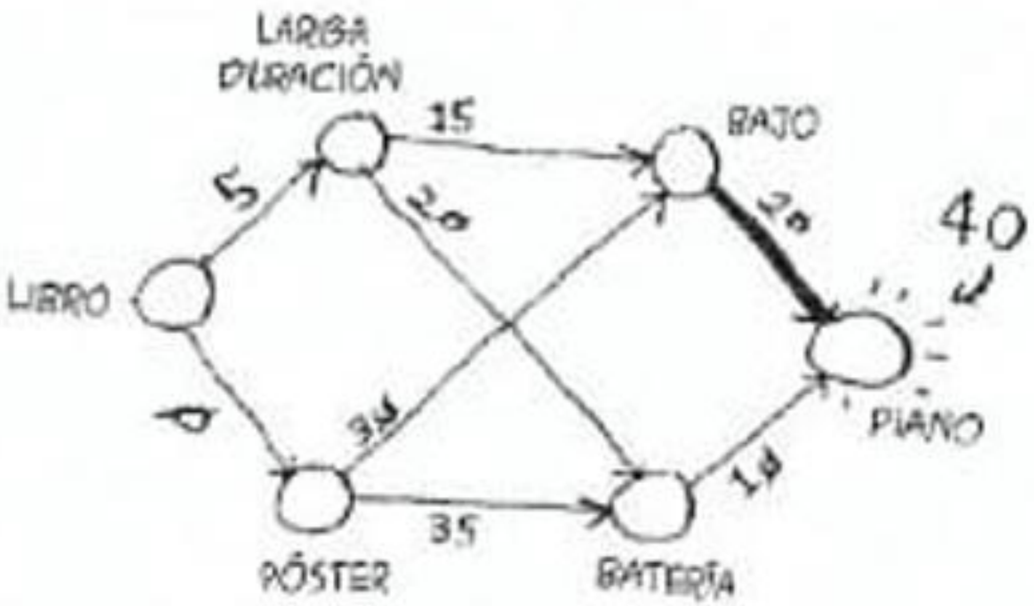
PADRE	NODO	COSTO
LIBRO	LARGA DURACIÓN	5
LIBRO	PÓSTER	0
LARGA DURACIÓN	BAJO	20
LARGA DURACIÓN	BATERÍA	25
—	PIANO	∞

Actualizaste el precio de la batería y el bajo. Eso significa que es más barato adquirir la batería y el bajo si sigues la arista desde el LP. Por tanto , asignas LP como padre de ambos instrumentos

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 3: Continuas repitiendo los pasos 1 y 2. El bajo es el próximo elemento. Actualiza sus vecinos



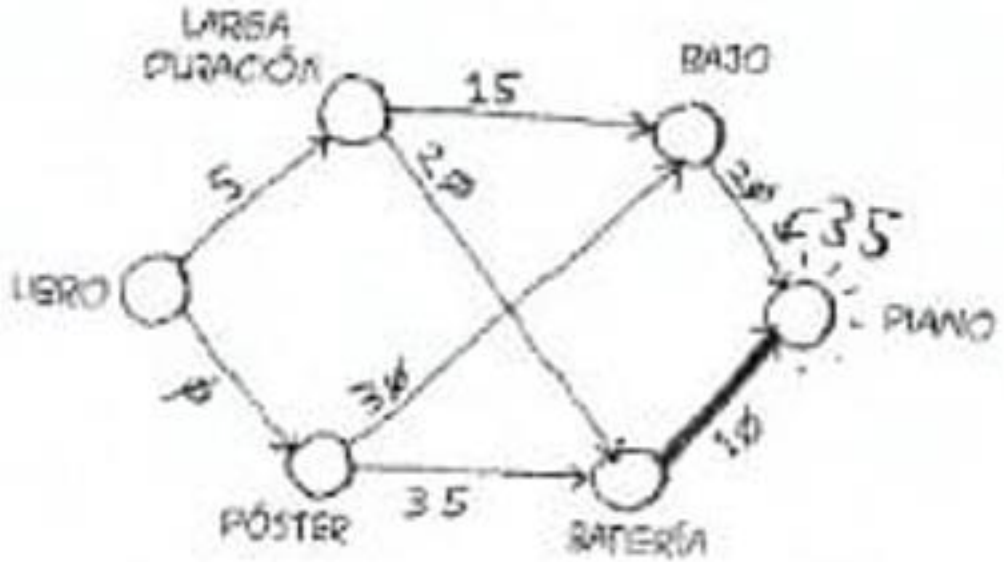
PADRE	NODO	COSTO
LIBRO	LARGA DURACIÓN	5
LIBRO	PÓSTER	0
LARGA DURACIÓN	BAJO	20
LARGA DURACIÓN	BATERÍA	25
BAJO	PIANO	40

Ok, finalmente tienes un precio para el piano, si intercambias el bajo por el piano. Así que marcas al bajo como su padre

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 3: Continuas repitiendo los pasos 1 y 2. Finalmente el último nodo es la batería



PADRE	NODO	COSTO
LIBRO	LARGA DURACIÓN	5
LIBRO	PÓSTER	0
LARGA DURACIÓN	BAJO	20
LARGA DURACIÓN	BATERÍA	25
BATERÍA	PIANO	35

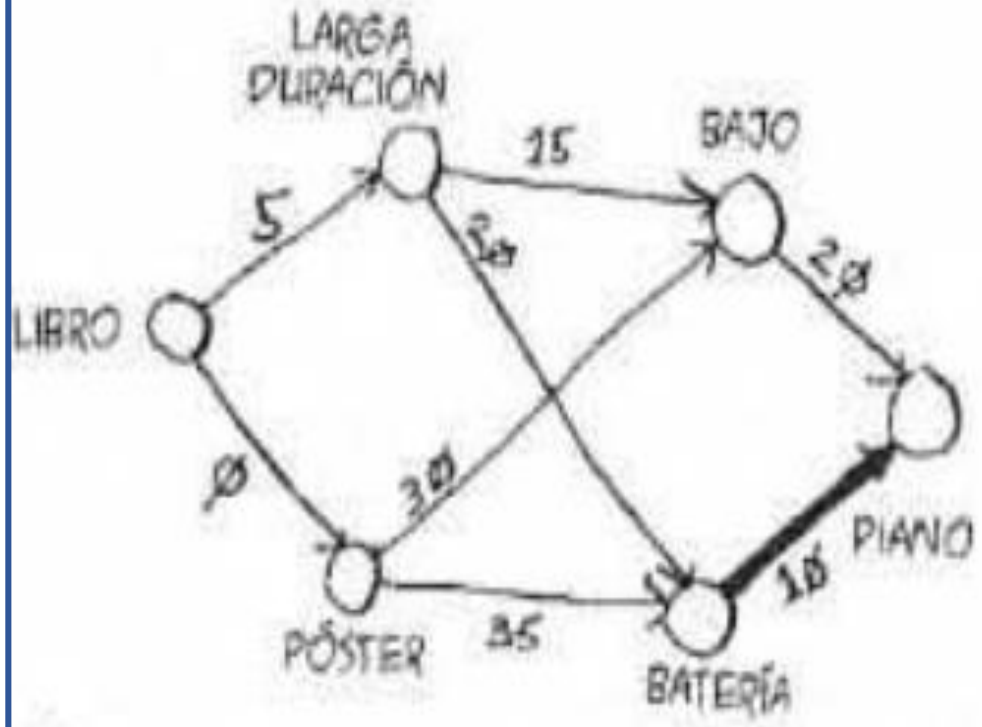
Rama puede obtener el piano aún más barato si intercambia la batería por el piano en su lugar. Así que el conjunto de intercambios más barato costará 35 $

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 4: Construir el camino

PADRE	NODO
LIBRO	LARGA DURACIÓN
LIBRO	PÓSTER
LARGA DURACIÓN	BAJO
LARGA DURACIÓN	BATERÍA
BATERÍA	PIANO



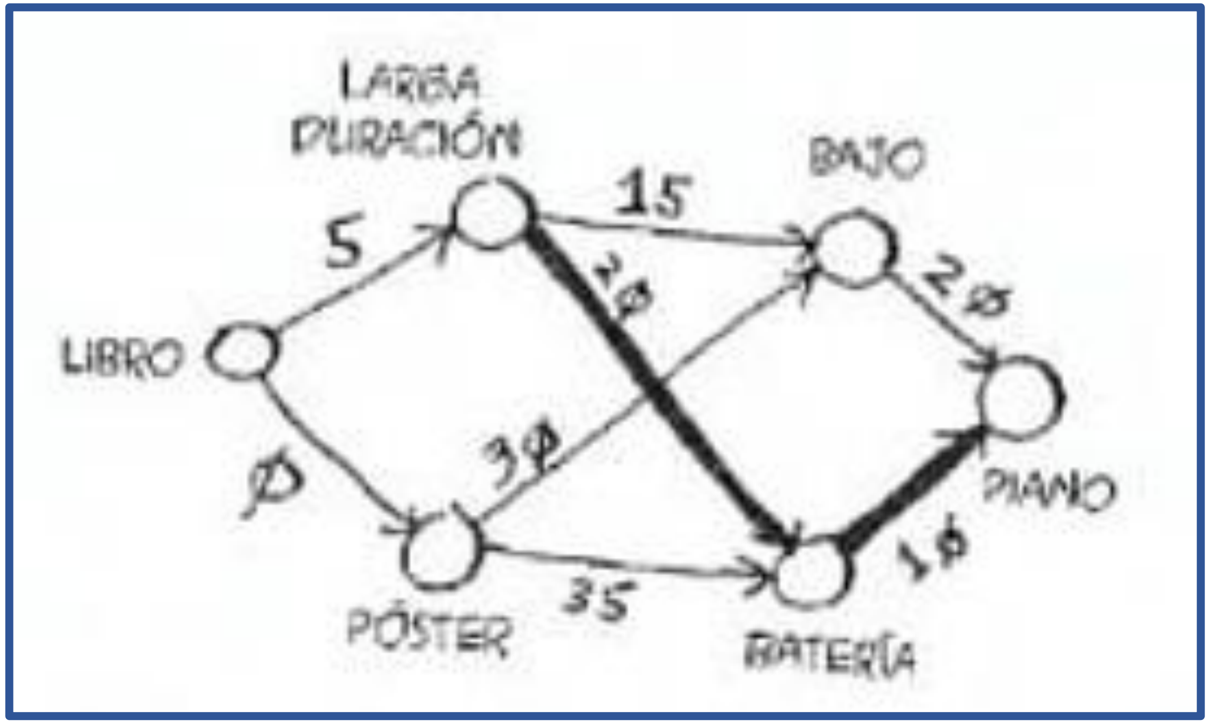
Para comenzar, fíjate en el valor de la columna padre para el piano. El piano tiene a la batería como su padre. Eso significa que Rama intercambia la batería por el piano, por lo tanto sigues esta arista

Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

PASO 4: Construir el camino

PADRE	NODO
LIBRO	LARGA DURACIÓN
LIBRO	PÓSTER
LARGA DURACIÓN	BAJO
LARGA DURACIÓN	BATERÍA
BATERÍA	PIANO

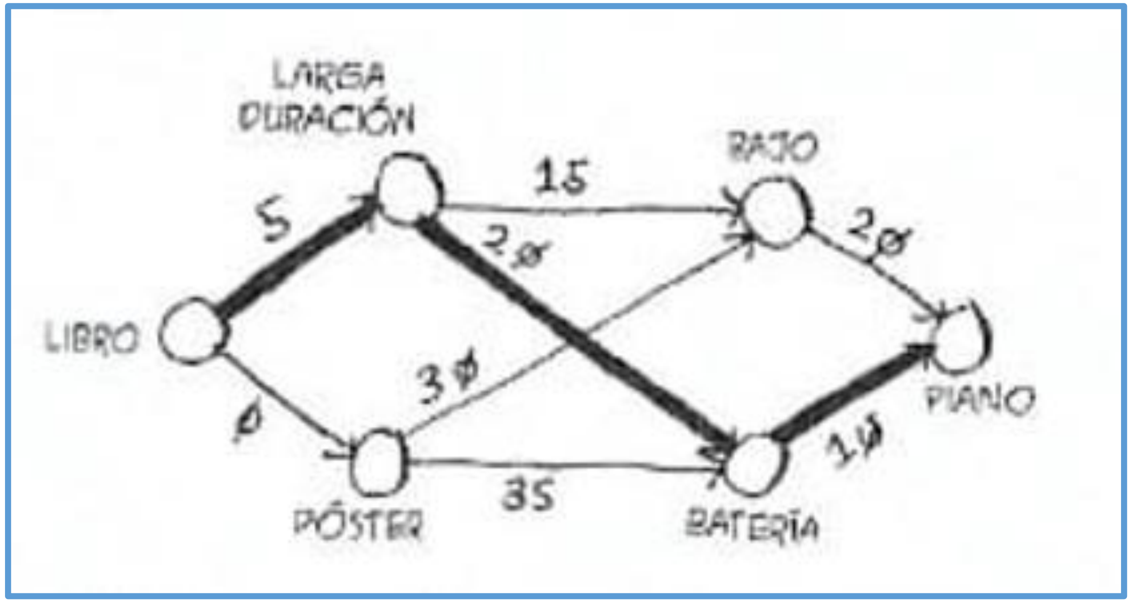


Y la batería tiene al LP como su padre, entonces Rama cambiaría el LP por la batería

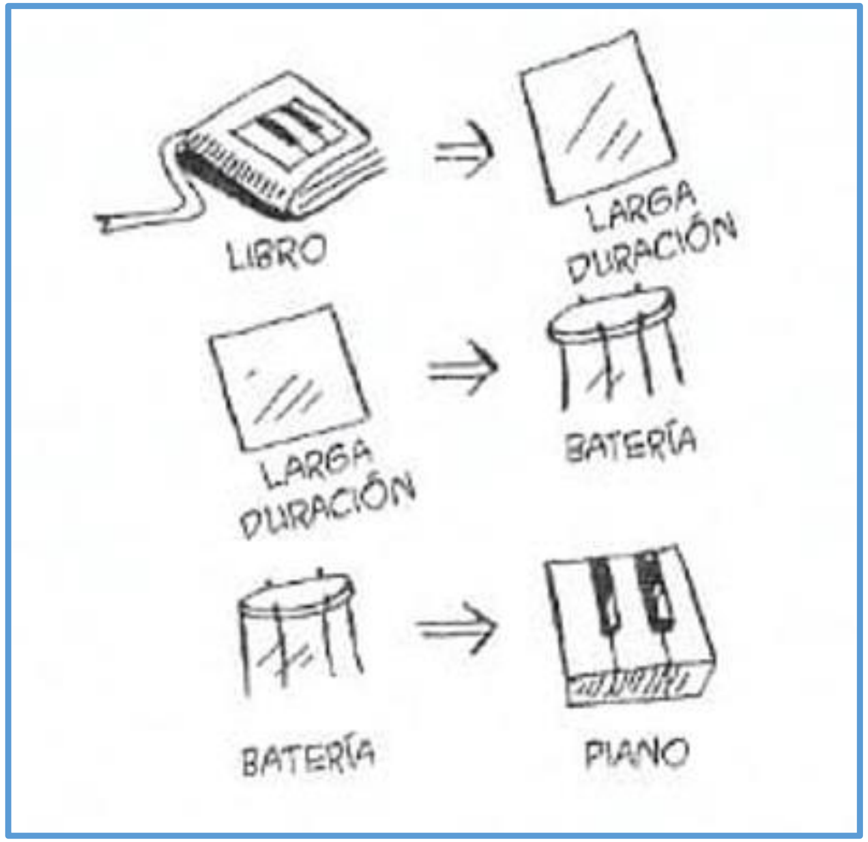
Trabajando con el algoritmo de Dijkstra: Ejercicio

Intercambiando un piano

Paso 4: Construir el camino



Y por supuesto, además cambiaría el libro por el LP. Al seguir a los padres en reversa construyes el camino completo



Aquí tienes la serie de intercambios que debe realizar Rama

Implementando el algoritmo de Dijkstra

Grafos

02

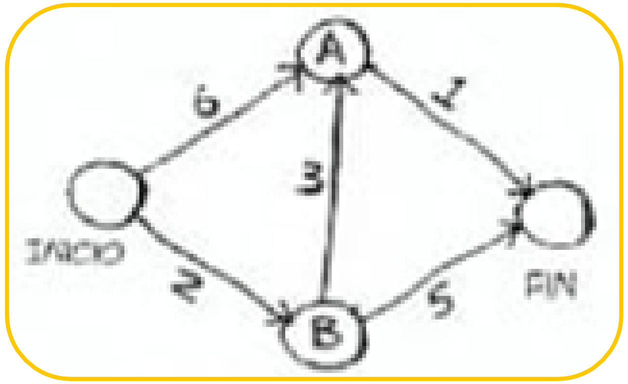
Implementación del algoritmo de Dijkstra

Implementar el algoritmo de Dijkstra en código.

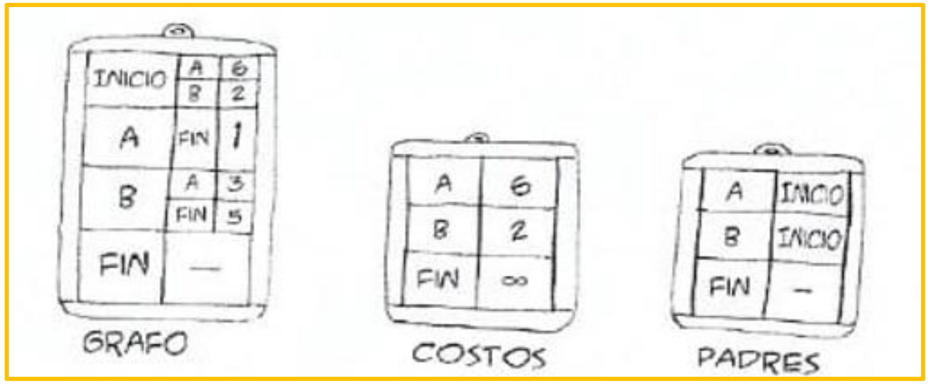
Grafo

Este es el grafo que usaremos como ejemplo

```
grafo={}
```

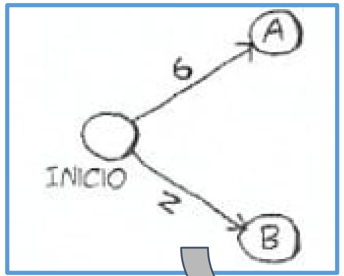


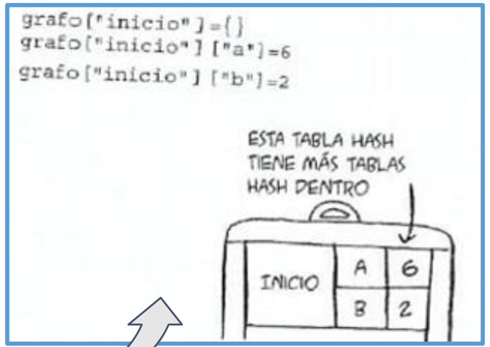
Se necesitarán 3 tablas hash



Actualizarás las tablas hash del costo y de los nodos padres a medida que el algoritmo progresa

Nodo inicial donde tiene dos vecinos A y B





Entonces grafo['inicio'] es una tabla hash. Puedes obtener todos los vecinos de "inicio" de la siguiente forma:

```
>>> print(grafo['inicio'].keys())
['a','b']
```

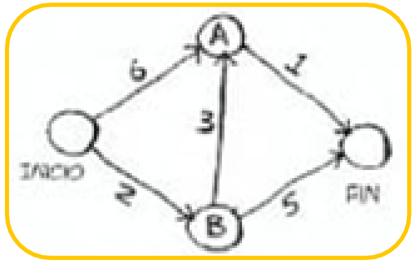
Existe una arista desde Inicio hacia A y otra arista desde Inicio hacia B

¿Qué harías para obtener los pesos de estas aristas?

```
>>>print (grafo['inicio']['a'])
6
>>> print (grafo['inicio']['b'])
2
```

Implementación del algoritmo de Dijkstra

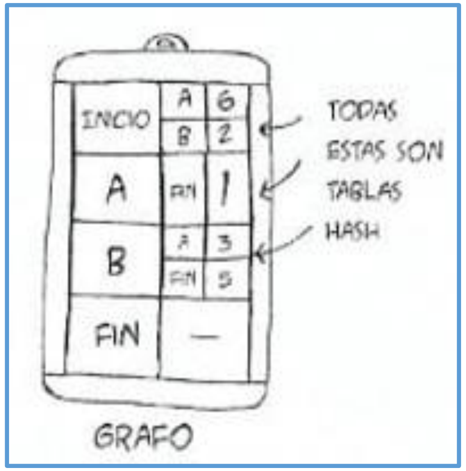
Incluyamos el resto de los nodos y aristas del grafo



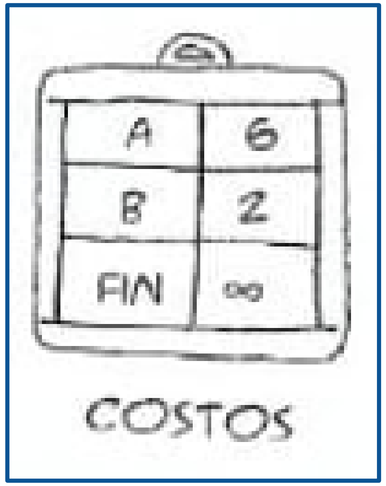

```
grafo["a"]={}
grafo["a"]["fin"]=1

grafo["b"]={}
grafo["b"]["a"]=3
grafo["b"]["fin"]=5

grafo["fin"]={}
```

Este nodo no tiene vecinos

La tabla hash completa se vé así



Ahora se necesita una tabla hash para guardar los costos de cada nodo.



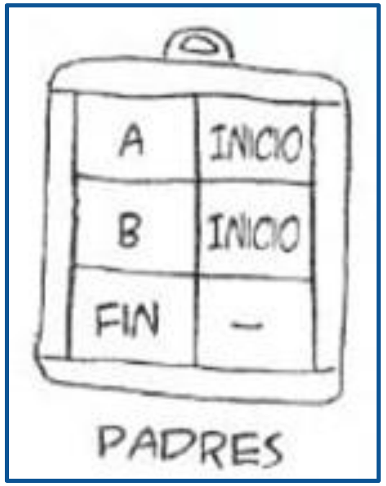
Tablas hash para guardar los costos

```
infinito=float("inf")
costos={}
costos["a"]=6
costos["b"]=2
costos["fin"]=infinito
```

Código para la tabla de costos

Implementación del algoritmo de Dijkstra

Además, necesitas otra tabla hash para los padres



Tablas hash para los padres

```
padres ={}
padres ["a"]="inicio"
padres ["b"]="inicio"
padres ["fin"]=None
```

Código para la tabla hash de los padres

Finalmente necesitas un arreglo para llevar cuenta de todos los nodos que ya has procesado, porque no necesitas procesar un nodo más de una vez

```
procesados=[]
```

Arreglo para tener todos los nodos que se han procesado

Implementación del algoritmo de Dijkstra

Lógica de Dijkstra

Implementación en Python – Función encuentra_nodo_de_menor_costo

Código

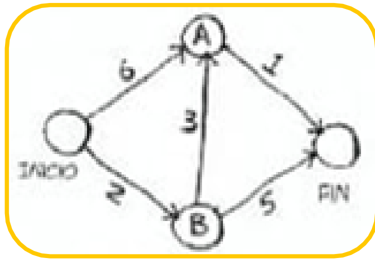
```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

- `nodo=encuentra_nodo_de_menor_costo(costos)` ← Encuentra el nodo de menor costo que aún no hayas procesado
- `while nodo is not None:` ← Si procesaste todos los nodos el ciclo termina
- `for n in vecinos.keys():` ← Itera sobre todos los vecinos de este nodo
- `if costos[n] > costo_nuevo:` ← Si es más barato llegar a este vecino a través de este nodo…
- `costos[n]=costo_nuevo` ← …actualiza el costo del nodo
- `padres[n]=nodo` ← Este nodo se convierte en el nuevo padre para este vecino
- `procesados.append(nodo)` ← Marca el nodo como procesado
- `nodo=encuentra_nodo_de_menor_costo(costos)` ← Encuentra el próximo nodo a procesar y continúa el ciclo

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

- `for nodo in costos:` ← Itera por cada nodo
- `if costo < menor_costo and nodo not in procesados:` ← Si es el nodo de menor costo hasta ahora y no ha sido procesado aún…
- `menor_costo=costo` ← …asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



Encontrar el nodo de menor costo


```
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo...

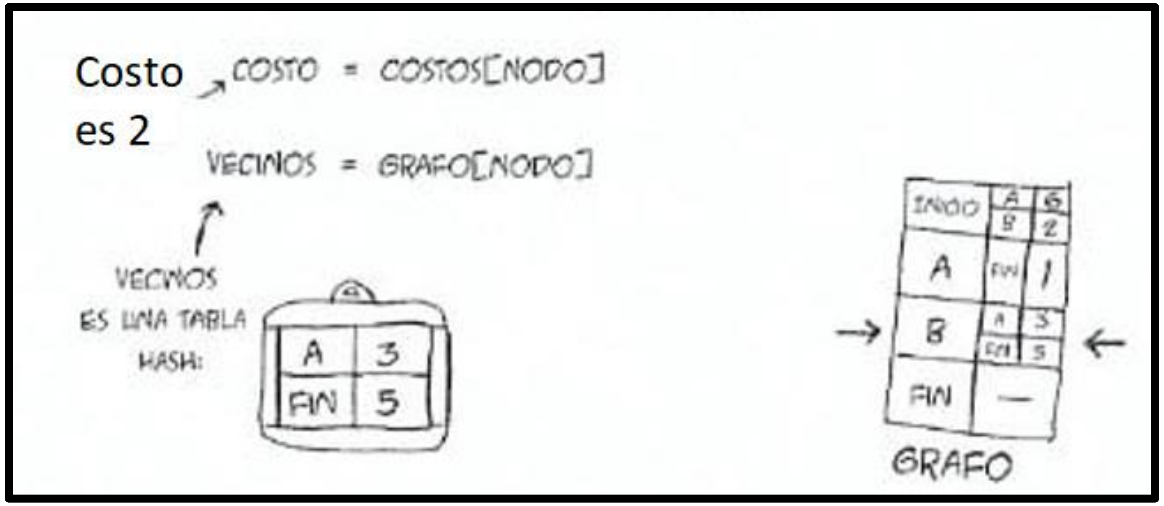
...actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

Obtén el costo y los vecinos de ese nodo



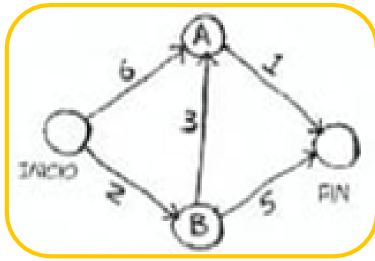

```
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

Si es el nodo de menor costo hasta ahora y no ha sido procesado aún...

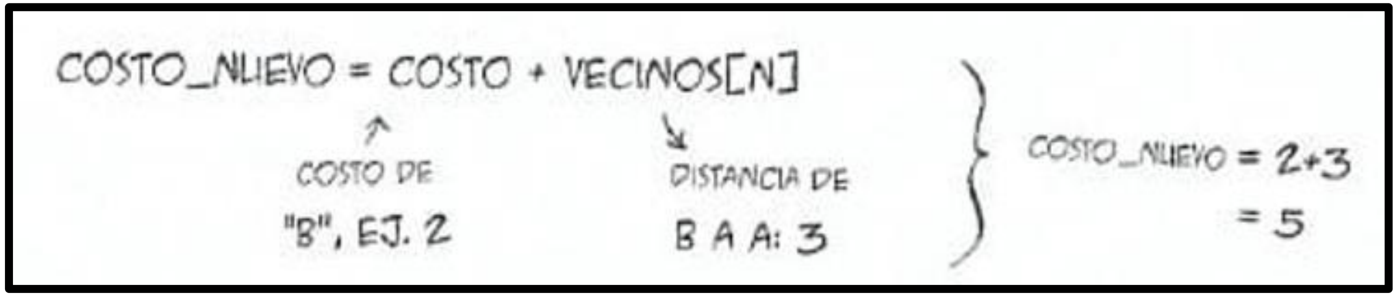
...asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



Iterar sobre los nodos vecinos

Cada nodo tiene un costo asociado. El costo definido por cuánto toma llegar a ese nodo desde el nodo inicio. Aquí, estamos calculando cuánto tomaría en llegar al nodo A si hicimos el siguiente camino Inicio > nodo B > nodo A, en lugar de ir directamente desde el inicio hacia el nodo A. Inicio > A




```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo…

…actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

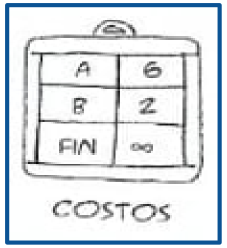
Encuentra el próximo nodo a procesar y continúa el ciclo

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

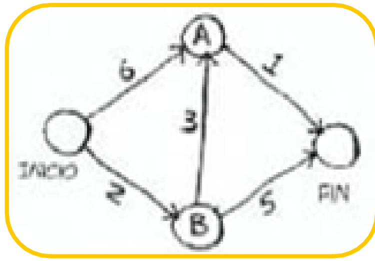
Itera por cada nodo

Si es el nodo de menor costo hasta ahora y no ha sido procesado aún…

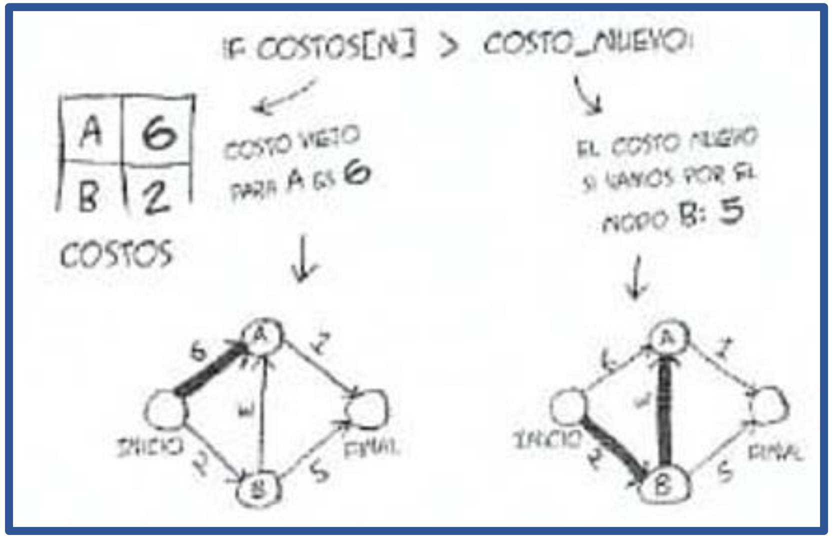
…asígnalo como el nuevo nodo de menor costo

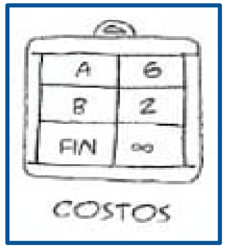


Ejecución del algoritmo de Dijkstra



Comparamos los costos






```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo…

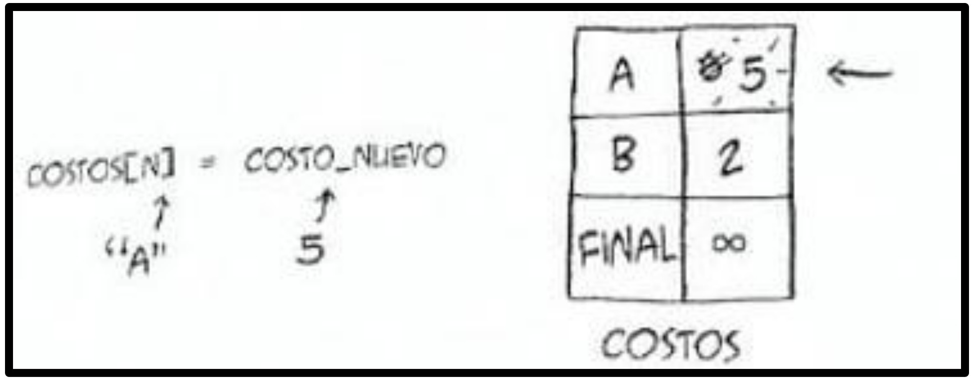
…actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

Al encontrar un camino más corto al nodo A. Se actualiza el costo en la tabla



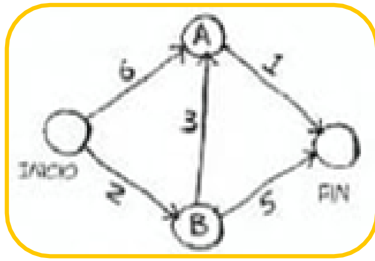

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

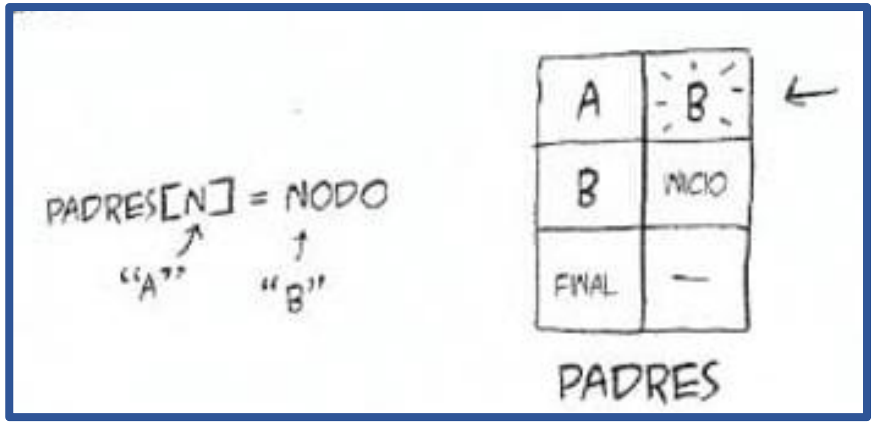
Si es el nodo de menor costo hasta ahora y no ha sido procesado aún…

…asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



El nuevo camino va a través del nodo B, así que se asigna al nodo B como el padre.




```
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo...

...actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

De vuelta al principio del ciclo. El próximo vecino es el nodo final.

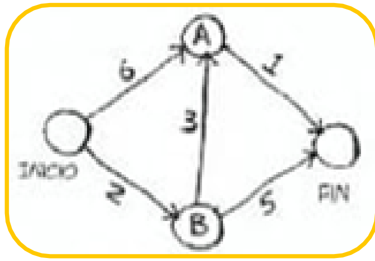

```
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

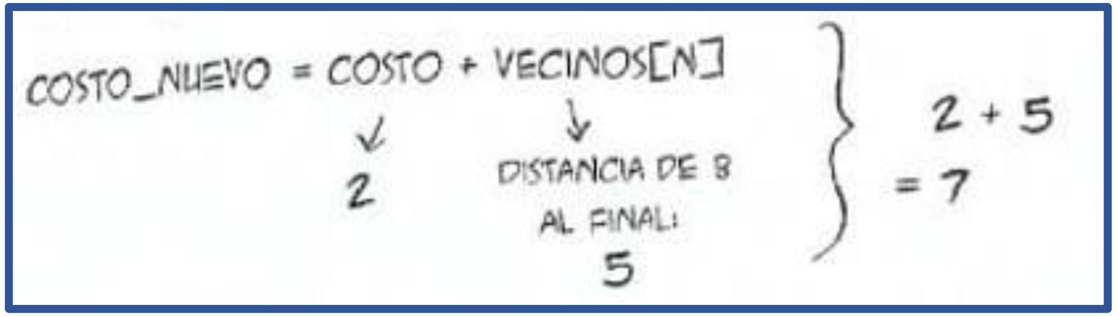
Si es el nodo de menor costo hasta ahora y no ha sido procesado aún...

...asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



Costo total para llegar al nodo final si se va a través del nodo B



Toma 7 minutos. El costo previo era infinito y 7 minutos es menor que eso


```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo…

…actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

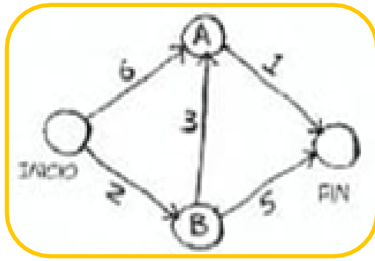
```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

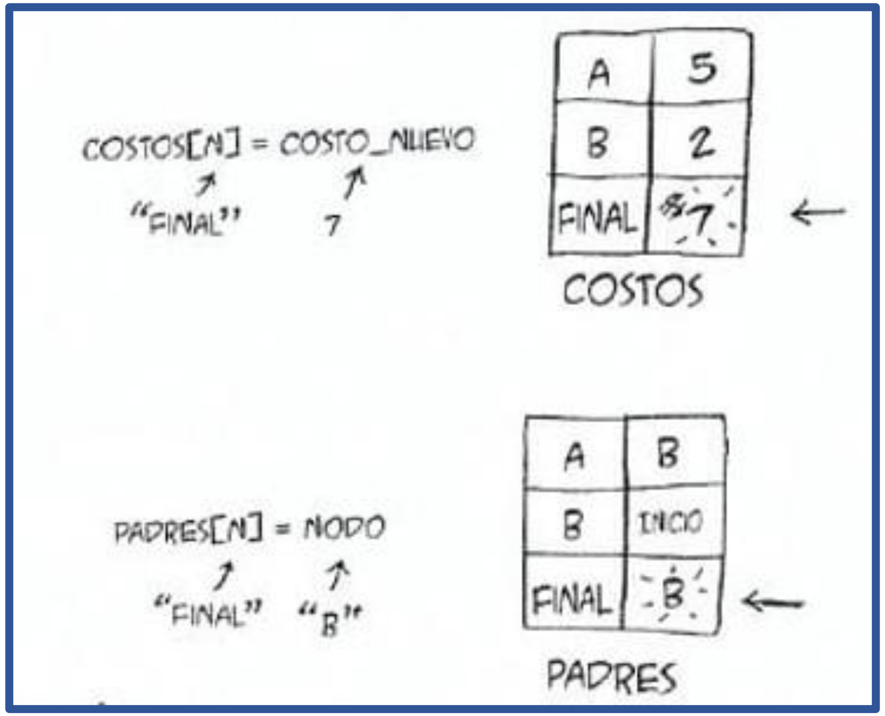
Si es el nodo de menor costo hasta ahora y no ha sido procesado aún…

…asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



Asignar el nuevo costo y el nuevo padre para el nodo final.




```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo...

...actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

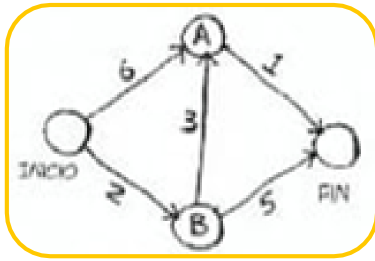
Itera por cada nodo

Si es el nodo de menor costo hasta ahora y no ha sido procesado aún...

...asígnalo como el nuevo nodo de menor costo

Al actualizar los costos de todos los nodos vecinos del nodo B. Se marca el nodo B como procesado.

Ejecución del algoritmo de Dijkstra



Busca un nuevo nodo a procesar.


```
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo…

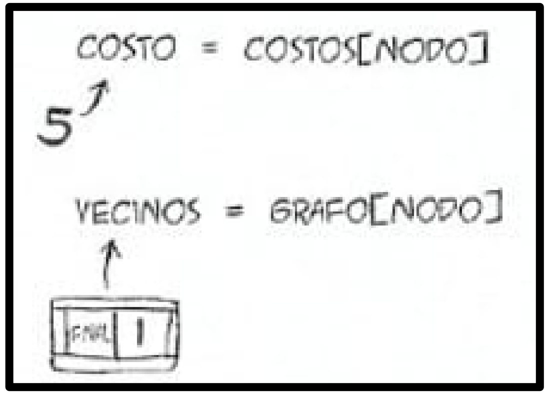
…actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

Una vez encontrado el nuevo nodo a procesar.
Se obtiene el costo y los vecinos de ese nodo.



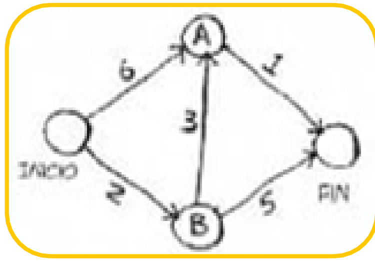

```
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

Si es el nodo de menor costo hasta ahora y no ha sido procesado aún…

…asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



El nodo elegido a procesar fue el nodo A, pero este nodo solo tiene un solo nodo vecino, en este caso, el nodo final.


```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo...

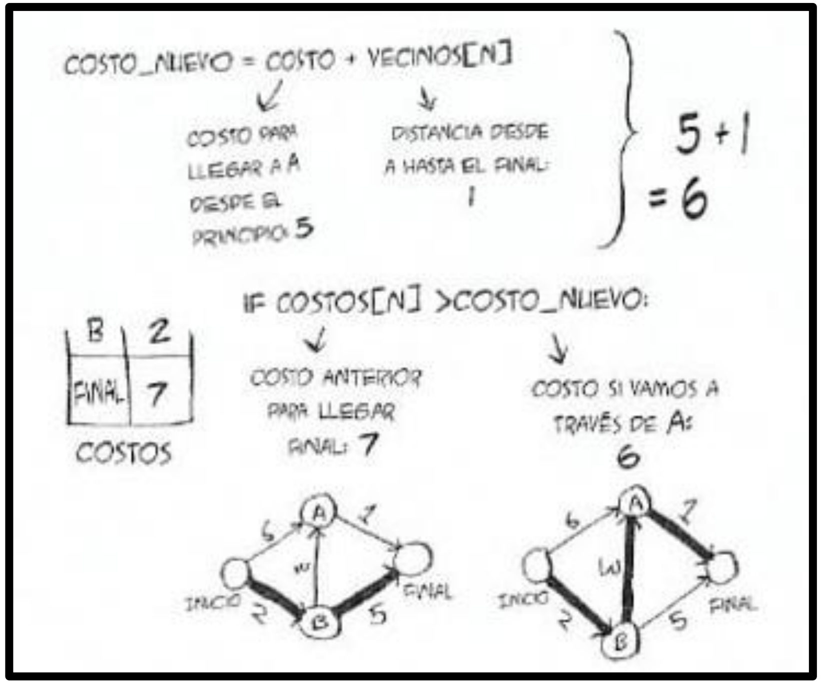
...actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

Hasta el momento, el costo para llenar al nodo final es de 7 minutos, a través del nodo B. ¿Ahora cuánto tomaría pasando a través del nodo A?



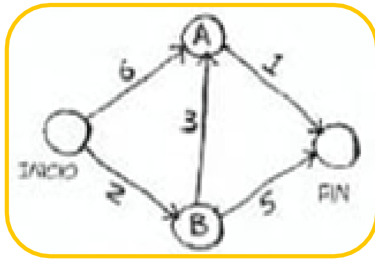

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

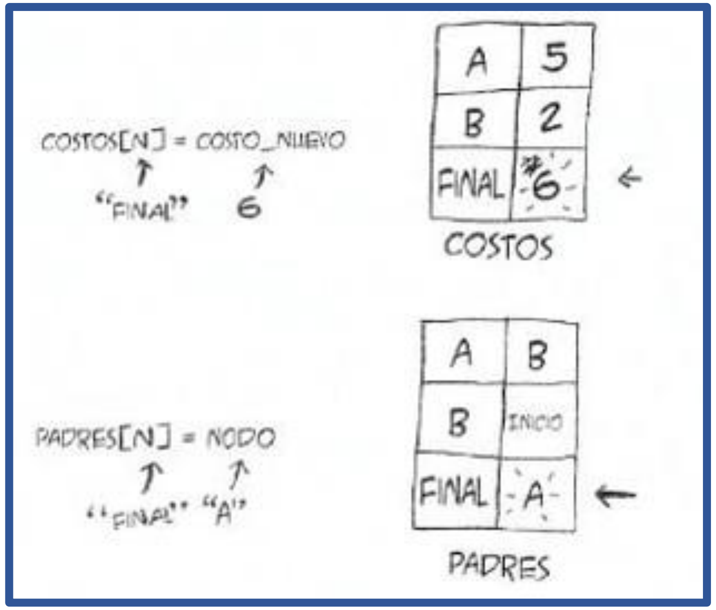
Si es el nodo de menor costo hasta ahora y no ha sido procesado aún...

...asígnalo como el nuevo nodo de menor costo

Ejecución del algoritmo de Dijkstra



Es más rápido llegar al nodo final desde el nodo A. Por lo que se procede a actualizar el costo y el padre.




```python
nodo=encuentra_nodo_de_menor_costo(costos)
while nodo is not None:
    costo=costos[nodo]
    vecinos=grafo[nodo]
    for n in vecinos.keys():
        costo_nuevo=costo+vecinos[n]
        if costos[n] > costo_nuevo:
            costos[n]=costo_nuevo
            padres[n]=nodo
    procesados.append(nodo)
    nodo=encuentra_nodo_de_menor_costo(costos)
```

Encuentra el nodo de menor costo que aún no hayas procesado

Si procesaste todos los nodos el ciclo termina

Itera sobre todos los vecinos de este nodo

Si es más barato llegar a este vecino a través de este nodo...

...actualiza el costo del nodo

Este nodo se convierte en el nuevo padre para este vecino

Marca el nodo como procesado

Encuentra el próximo nodo a procesar y continúa el ciclo

```python
def encuentra_nodo_de_menor_costo(costos):
    menor_costo=float('inf')
    nodo_menor_costo=None
    for nodo in costos:
        costo=costos[nodo]
        if costo < menor_costo and nodo not in procesados:
            menor_costo=costo
            nodo_menor_costo=nodo
    return nodo_menor_costo
```

Itera por cada nodo

Si es el nodo de menor costo hasta ahora y no ha sido procesado aún...

...asígnalo como el nuevo nodo de menor costo

¡Una vez que hayas procesado todos los nodos el algoritmo llega a su fin!

Algoritmo de Dijkstra - Pseudocódigo

DIJKSTRA

Entrada: grafo dirigido $G = (V, E)$ representado como una lista de adyacencia, un vértice $s \in V$, una longitud $\ell_e \geq 0$ para cada $e \in E$.

Condición posterior: para cada vértice v, el valor $\text{len}(v)$ es igual a la verdadera distancia del camino más corto $\text{dist}(s, v)$.

// Inicialización

$X := \{s\}$

$\text{len}(s) := 0$, $\text{len}(v) := +\infty$ para todo $v \neq s$

// Bucle principal

mientras exista una arista (v, w) con $v \in X, w \notin X$ **hacer**

  $(a, b) :=$ dicha arista minimizando $\text{len}(v) + \ell_{vw}$

  añadir b a X

  $\text{len}(b) := \text{len}(a) + \ell_{ab}$

Conclusiones

- El algoritmo de Dijkstra se utiliza para calcular el camino de costo mínimo en un grafo ponderado.
- El algoritmo de Dijkstra funciona cuando todos los pesos son positivos.
- Si hay pesos negativos, utilizar el algoritmo Bellman-Ford.
- El algoritmo de Dijkstra es de los más populares para recorrer grafos.

Algoritmo Floyd Warshall

Grafos

02

Caminos mínimos entre todos los pares.

- **Problema:** Calcular los caminos mínimos entre todos los pares de nodos del grafo.

Posibilidades

- Aplicar el algoritmo de Dijkstra **n** veces, una por cada posible nodo origen:
  - Con matrices de adyacencia: **$O(n^3)$**
  - Con listas de adyacencia: **$O(a \cdot n \cdot \log n)$**
- Aplicar el algoritmo de Floyd:
  - Con listas o matrices: **$O(n^3)$**
  - Pero más sencillo de programar...

Caminos mínimos entre todos los pares.

- **Entrada:**

  C: array [1..n, 1..n] **de** real → Matriz de costes

- **Salida:**

  D: array [1..n, 1..n] **de** real → Costes caminos mínimos

Algoritmo de Floyd

```
D:= C
para k:= 1, ..., n hacer
    para i:= 1, ..., n hacer
        para j:= 1, ..., n hacer
            D[i, j]:= min ( D[i, j] , D[i, k] + D[k, j] )
```

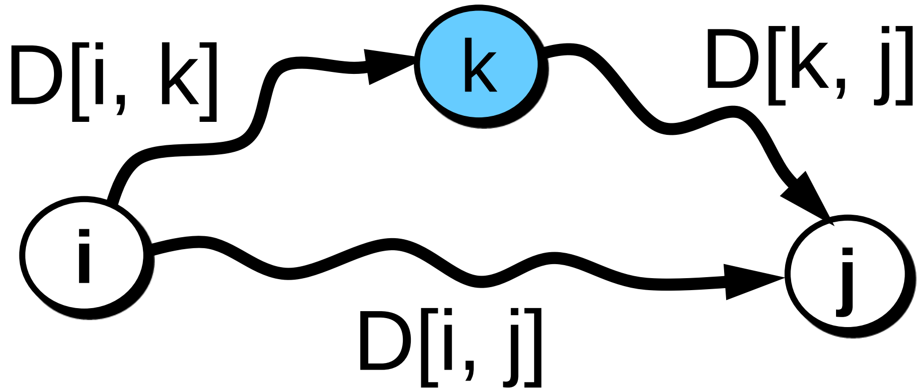
Caminos mínimos entre todos los pares.

- ¿En qué se basa el algoritmo de Floyd?

- En cada paso **k**, la matriz **D** almacena los caminos mínimos entre todos los pares pudiendo pasar por los **k** primeros nodos.
- **Inicialización: D** almacena los caminos directos.
- **Paso 1:** Caminos mínimos pudiendo pasar por el 1.
- ...
- **Paso n:** Caminos mínimos pudiendo pasar por cualquier nodo $\rightarrow$ Lo que buscamos.

- En el paso **k**, el nodo **k** actúa de pivote.

Caminos mínimos entre todos los pares.



- **Camino mínimo entre i y j, en el paso k:**
  - Sin pasar por **k**: D[i, j]
  - Pasando por **k**: D[i, k] + D[k, j]
  - Nos quedamos con el menor.
- **Ojo:** Falta indicar cuáles son los caminos mínimos.
- **P: array** [1..n, 1..n] **de** entero. **P[i, j]** indica un nodo intermedio en el camino de **i** a **j**.

$$i \rightarrow \ldots \rightarrow P[i, j] \rightarrow \ldots \rightarrow j$$

Caminos mínimos entre todos los pares.

Algoritmo de Floyd

```
D:= C
P:= 0
para k:= 1, ..., n hacer
    para i:= 1, ..., n hacer
        para j:= 1, ..., n hacer
            si D[i, k] + D[k, j] < D[i, j] entonces
                D[i, j]:= D[i, k] + D[k, j]
                P[i, j]:= k
            finsi
```

- ¿Cuánto es el orden de complejidad del algoritmo?

Caminos mínimos entre todos los pares.

- El algoritmo de Floyd se basa en una descomposición recurrente del problema:

$$D_k(i, j) := \begin{cases} C[i, j] & \text{Si } k=0 \\ \min(D_{k-1}(i, j),\ D_{k-1}(i, k) + D_{k-1}(k, j)) & \text{Si } k>0 \end{cases}$$

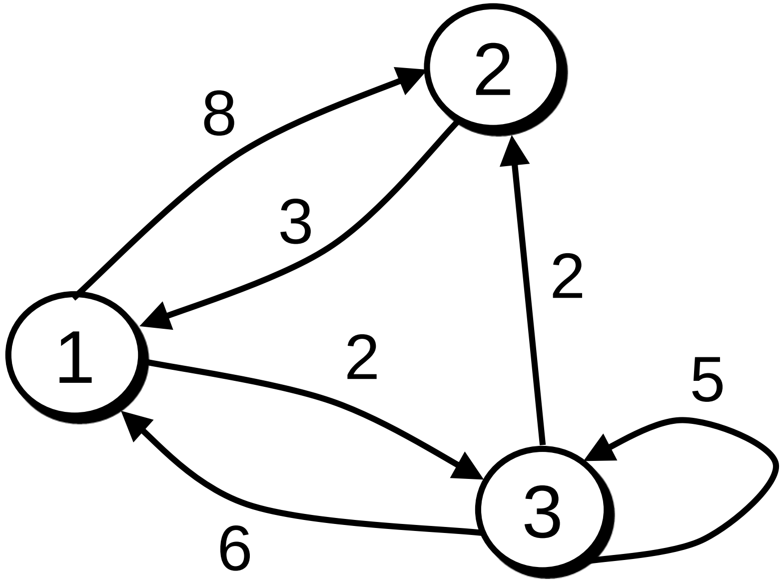
- Como la fila y columna **k** no cambian en el paso **k**, se usa una sola matriz **D**.
- ¿Cómo recuperar el camino?

```
operación camino (i, j: entero)
        k:= P[i, j]
        si k ≠ 0 entonces
            camino (i, k)
            escribe (k)
            camino (k, j)
        finsi
```

escribe (i)
camino (i, j) ←
escribe (j)

Caminos mínimos entre todos los pares.

- **Ejemplo:** Aplicar el algoritmo de Floyd al siguiente grafo dirigido.



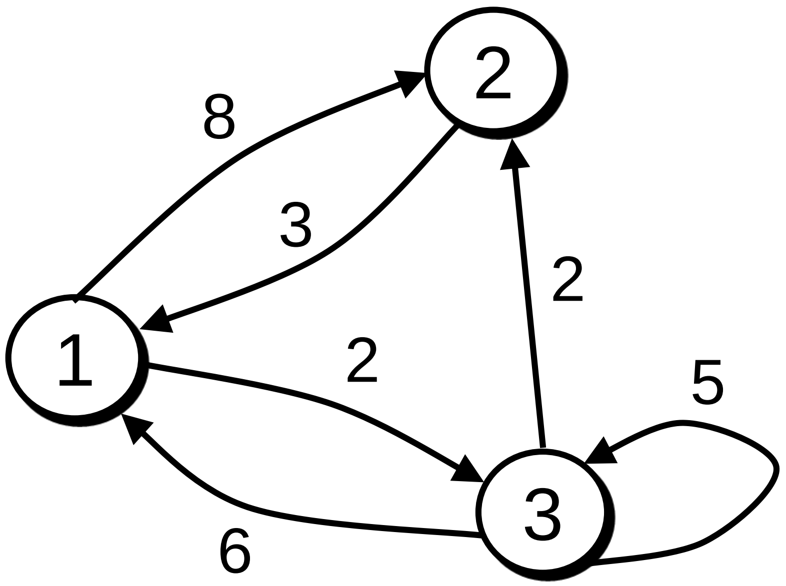
D	1	2	3
1	0	8	2
2	3	0	∞
3	6	2	0

P	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0

- Calcular el camino mínimo entre 1 y 2.

Caminos mínimos entre todos los pares.

- **Ejemplo:** Aplicar el algoritmo de Floyd al siguiente grafo dirigido.



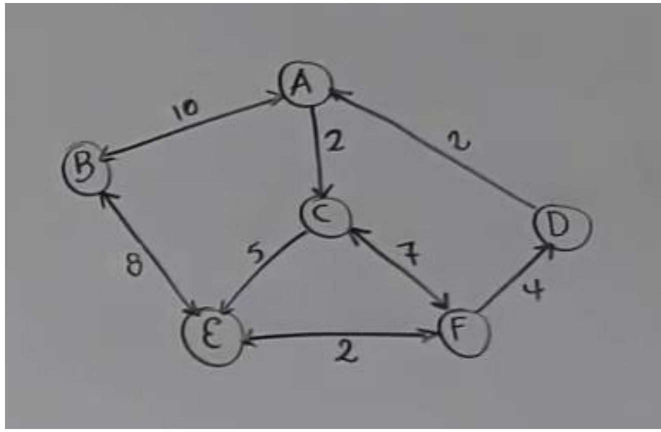
D	1	2	3
1	0	4	2
2	3	0	5
3	5	2	0

P	1	2	3
1	0	3	0
2	0	0	1
3	2	0	0

- Calcular el camino mínimo entre 1 y 2.

Ejercicio.

- **Ejemplo:** Aplicar el algoritmo de Floyd al siguiente grafo dirigido.

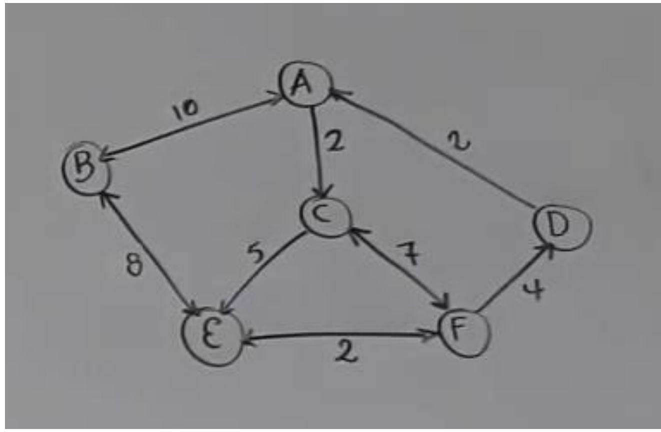


D	A	B	C	D	E	F
A	0	10	2	∞	∞	∞
B	10	0	∞	∞	8	∞
C	∞	∞	0	∞	5	7
D	2	∞	∞	0	∞	∞
E	∞	8	∞	∞	0	2
F	∞	∞	7	4	2	0

P	A	B	C	D	E	F
A	-	B	C	D	E	F
B	A	-	C	D	E	F
C	A	B	-	D	E	F
D	A	B	C	-	E	F
E	A	B	C	D	-	F
F	A	B	C	D	E	-

Ejercicio.

- **Ejemplo:** Aplicar el algoritmo de Floyd al siguiente grafo dirigido.



D	A	B	C	D	E	F
A	0	10	2	13	7	9
B	10	0	12	14	8	10
C	13	13	0	11	5	7
D	2	12	4	0	9	11
E	8	8	9	6	0	2
F	6	10	7	4	2	0

P	A	B	C	D	E	F
A	-	B	C	F	C	C
B	A	-	A	F	E	E
C	F	E	-	F	E	F
D	A	A	A	-	C	C
E	F	B	F	F	-	F
F	D	E	C	D	E	-

Cierre transitivo de un grafo.

- **Problema:** Dada una matriz de adyacencia **M** (de boolenos) encontrar otra matriz **A**, tal que **A[i, j]** es cierto si y sólo si existe un camino entre **i** y **j**.

Algoritmo de Warshall

- Es una simple adaptación del algoritmo de Floyd a valores booleanos.

A:= M

para k:= 1, ..., n **hacer**

  para i:= 1, ..., n **hacer**

    para j:= 1, ..., n **hacer**

      A[i, j]:= A[i, j] OR (A[i, k] AND A[k, j])

Problemas de caminos mínimos.

Conclusiones

- **Caminos mínimos:** Problema fundamental en grafos. Diferentes problemas, con diversas aplicaciones.
- Desde un origen hasta todos los demás nodos → Algoritmo de **Dijkstra**.
- **Idea:** Nodos escogidos y candidatos.

- Entre todos los pares → Algoritmo de **Floyd**.
- **Idea:** Pivotar sobre cada nodo.

- Ambos algoritmos pueden modificarse para resolver otros problemas relacionados.

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América