Grafos: camino más corto

AED II

2020

UBP UNIVERSIDAD BLAS PASCAL

Repaso
algoritmos voraces

algoritmos voraces (greedy)

Los **algoritmos voraces** o **glotones** (en inglés **greedy** algorithms), deben su nombre a su comportamiento: en cada etapa toman lo que pueden sin analizar las consecuencias

Adoptan un enfoque **miope**: las decisiones se toman utilizando únicamente la información disponible en cada paso, sin tener en cuenta los efectos que estas decisiones puedan tener en el futuro.

Los algoritmos voraces resultan **fáciles** de diseñar, fáciles de implementar y, cuando funcionan, son eficientes.

algoritmos voraces

Normalmente se trata de algoritmos que resuelven problemas de optimización:

- Dado un problema con n entradas, se trata de obtener un subconjunto de éstas que satisfaga una determinada **restricción** definida para el problema.
- Cada uno de los subconjuntos que cumplan las restricciones son **soluciones factibles o prometedoras**.
- Una solución factible que maximice o minimice una función objetivo se denominará **solución óptima**.

- Ejemplos:
  - el camino más corto que une dos ciudades,
  - el valor máximo alcanzable entre ciertos objetos,
  - el costo mínimo para proveer un cierto servicio,
  - el menor número de billetes para pagar un cierto importe,
  - el menor tiempo necesario para realizar un trabajo, etc.

algoritmo voraz

El funcionamiento de los algoritmos voraces puede resumirse como sigue:

1. Para resolver el problema, un algoritmo voraz tratará de tomar el conjunto de decisiones (escoger el conjunto de candidatos) que, cumpliendo las restricciones del problema, conduzca a la solución óptima.

2. Para ello trabajará por etapas, tomando en cada una de ellas la decisión que le parezca mejor, sin considerar las consecuencias futuras.

3. De acuerdo a esta estrategia, tomará aquella decisión (escogerá a aquel candidato) que produzca un óptimo local en cada etapa, suponiendo que ello conducirá, a su vez, a un óptimo global para el problema.

4. Antes de tomar una decisión, el algoritmo voraz comprobará si es prometedora. En caso afirmativo, la decisión se ejecuta (es decir, se da un nuevo paso hacia la solución) sin posibilidad de retroceder; en caso negativo, la decisión será descartada para siempre.

5. Cada vez que se toma una nueva decisión, el algoritmo voraz comprobará si la solución construida hasta ese punto cumple las restricciones de una solución al problema.

algoritmos voraces: elementos

Conjunto de candidatos: elementos seleccionables
Solución parcial: candidatos seleccionados
Función de selección: determina el mejor candidato del conjunto de candidatos seleccionables
Función de factibilidad: determina si es posible completar la solución parcial para alcanzar una solución del problema
Criterio que define lo que es una solución: indica si la solución parcial obtenida resuelve el problema
Función objetivo: valor de la solución alcanzada

algoritmo voraz: esquema

- Se parte de un conjunto vacío: S = ø .
- De la lista de candidatos, se elige el mejor (de acuerdo con la función de selección).
- Comprobamos si se puede llegar a una solución con el candidato seleccionado (función de factibilidad).
- Si no es así, lo eliminamos de la lista de candidatos posibles y nunca más lo consideraremos.
- Si aún no hemos llegado a una solución, seleccionamos otro candidato y repetimos el proceso hasta llegar a una solución [o quedarnos sin posibles candidatos].

algoritmo voraz: esquema

```
Greedy (conjunto de candidatos C): solución S {
     S = ∅
     while (S no sea una solución y C ≠ ∅) {
          x = selección(C)
          C = C – {x}
          if (S ∪ {x} es factible)
               S = S ∪ {x}
     }
     if (S es una solución)
          return S;
     else
          return "No se encontró una solución";
}
```

Fin del Repaso

camino más corto: algoritmo de Dijkstra

Problema
Dado un grafo G(V,A) y un vértice de origen v, encontrar el camino más corto desde el vértice v a los demás vértices del grafo

La solución consiste en generar un **árbol de caminos mínimos** (SPT - Shortest Path Tree) que inicialmente contendrá solamente el vértice de origen. Mantenemos dos conjuntos: uno el de los vértices incluidos en el árbol de caminos mínimos, el otro conjunto contiene aquellos vértices que todavía no han sido incluidos.
En cada paso encontramos un nuevo vértice con distancia mínima desde el origen y lo agregamos al primer conjunto

camino más corto: algoritmo de Dijkstra

Algoritmo

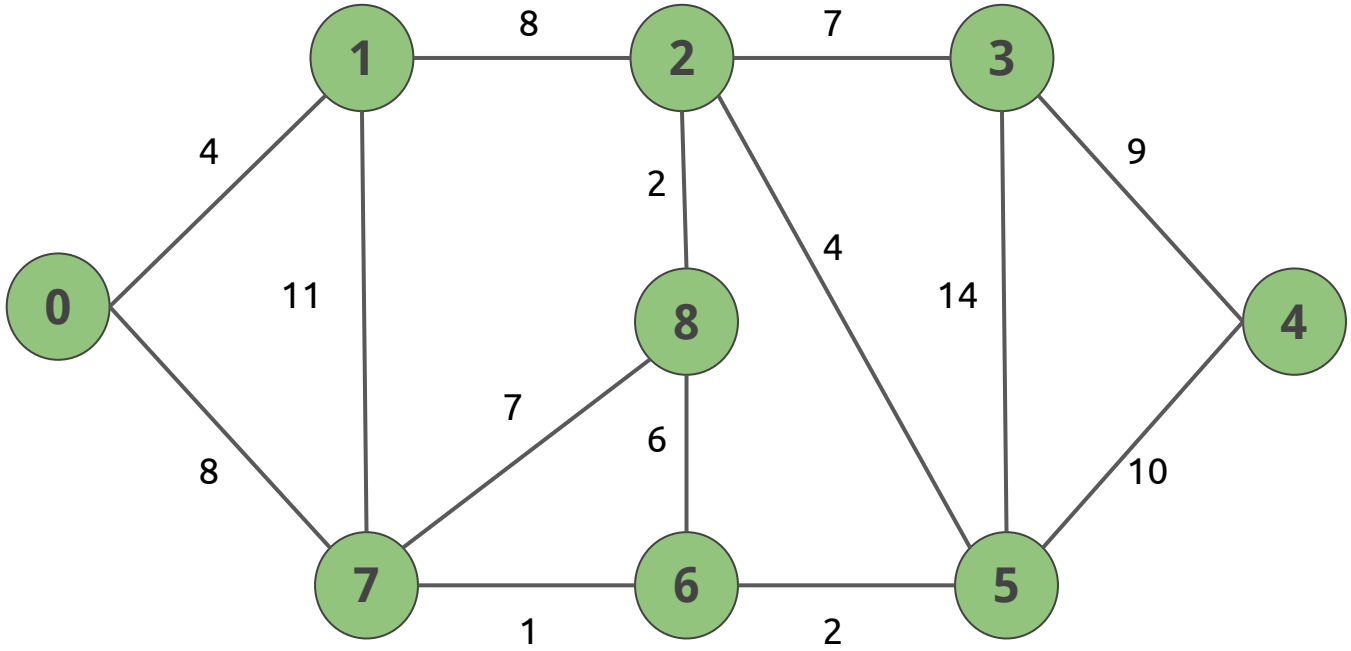
1) Crear un conjunto U que contiene los vértices incluidos en el árbol de caminos mínimos (aquellos cuya distancia al origen está calculada y finalizada). Este conjunto está vacío inicialmente

2) Asignamos una distancia a todos los vértices del grafo. Inicialmente esta distancia es 0 para el vértice de origen e infinito para el resto De modo que el vértice de origen es el primero que se selecciona

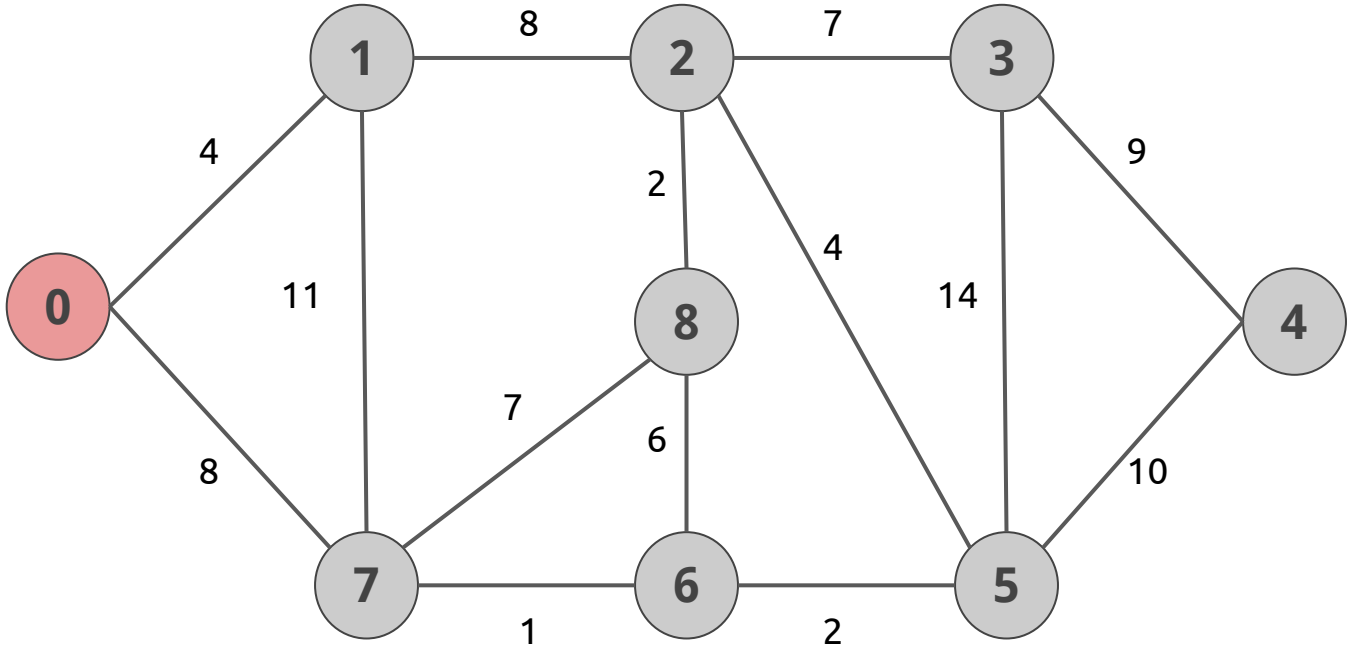
3) Mientras el conjunto U no contenga todos los vértices:

- **a)** Seleccionar un vértice u que no esté contenido en U y tiene distancia mínima
- **b)** Incluir u en U.
- **c)** Actualizar la distancia de los vértices adyacentes a u. Para cada vértice adyacente v, si la suma de la distancia de u (desde el origen) y el peso de la arista u-v es menor que la distancia de v, se actualiza la distancia de v.

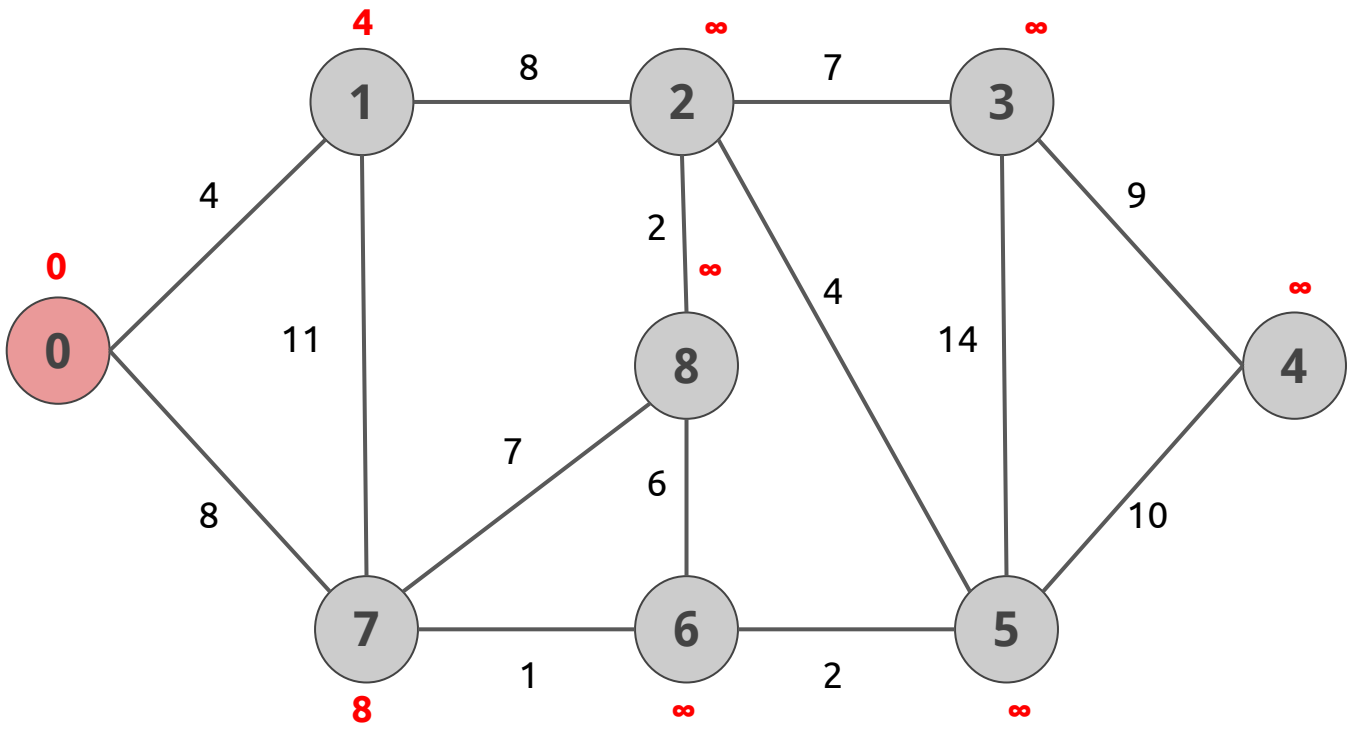
camino más corto: algoritmo de Dijkstra



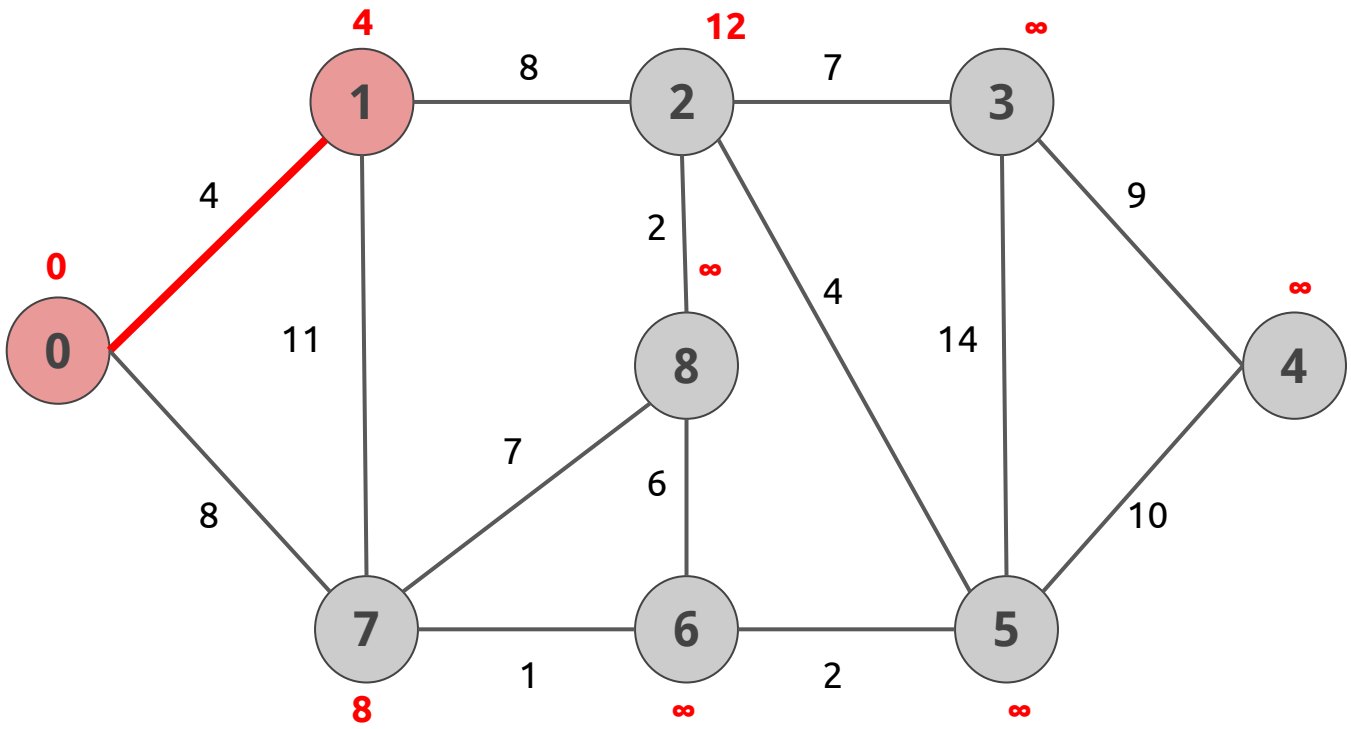
camino más corto: algoritmo de Dijkstra



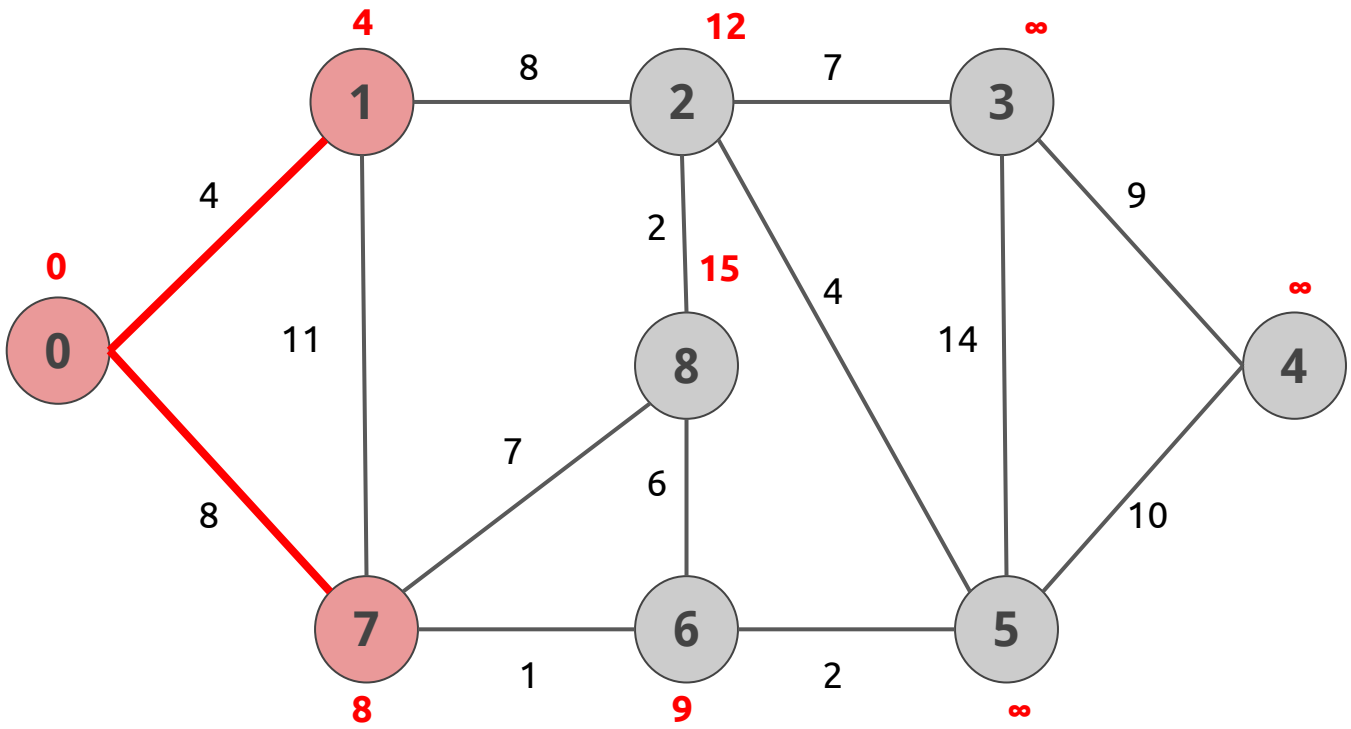
camino más corto: algoritmo de Dijkstra



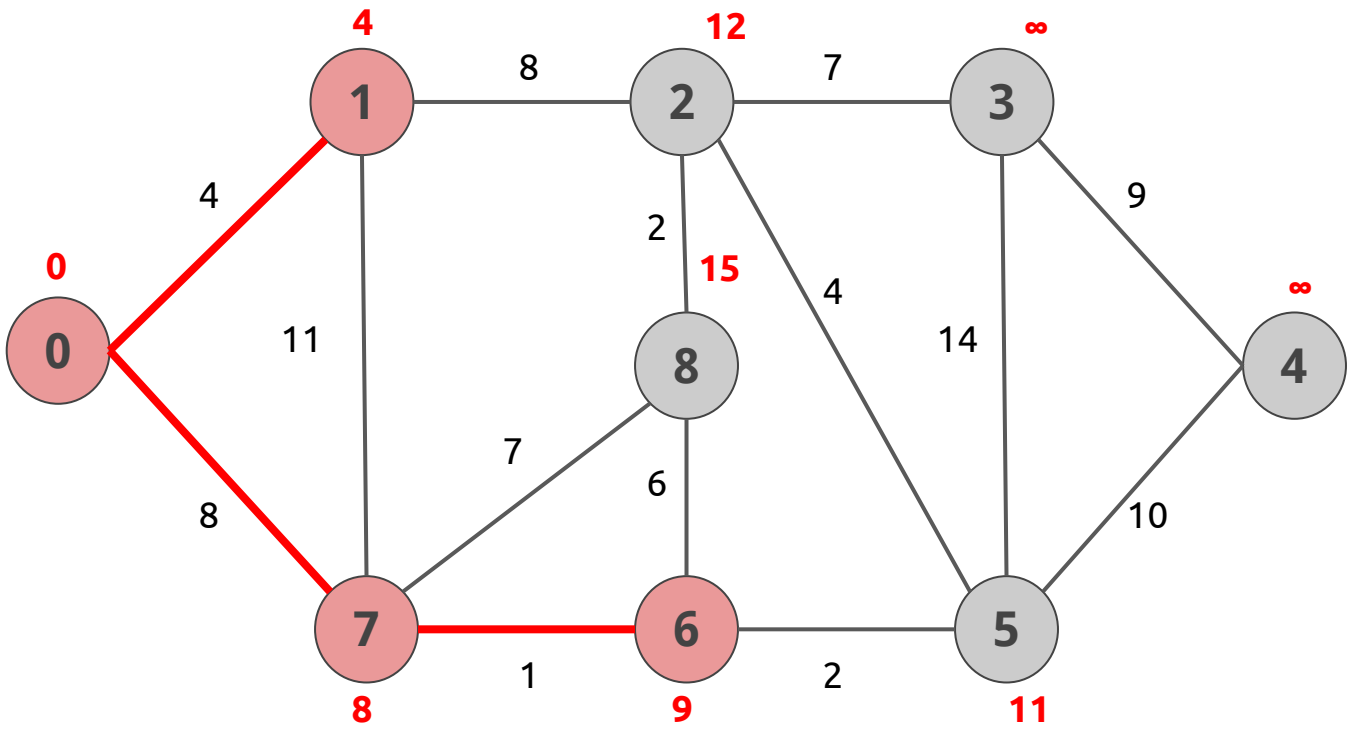
camino más corto: algoritmo de Dijkstra



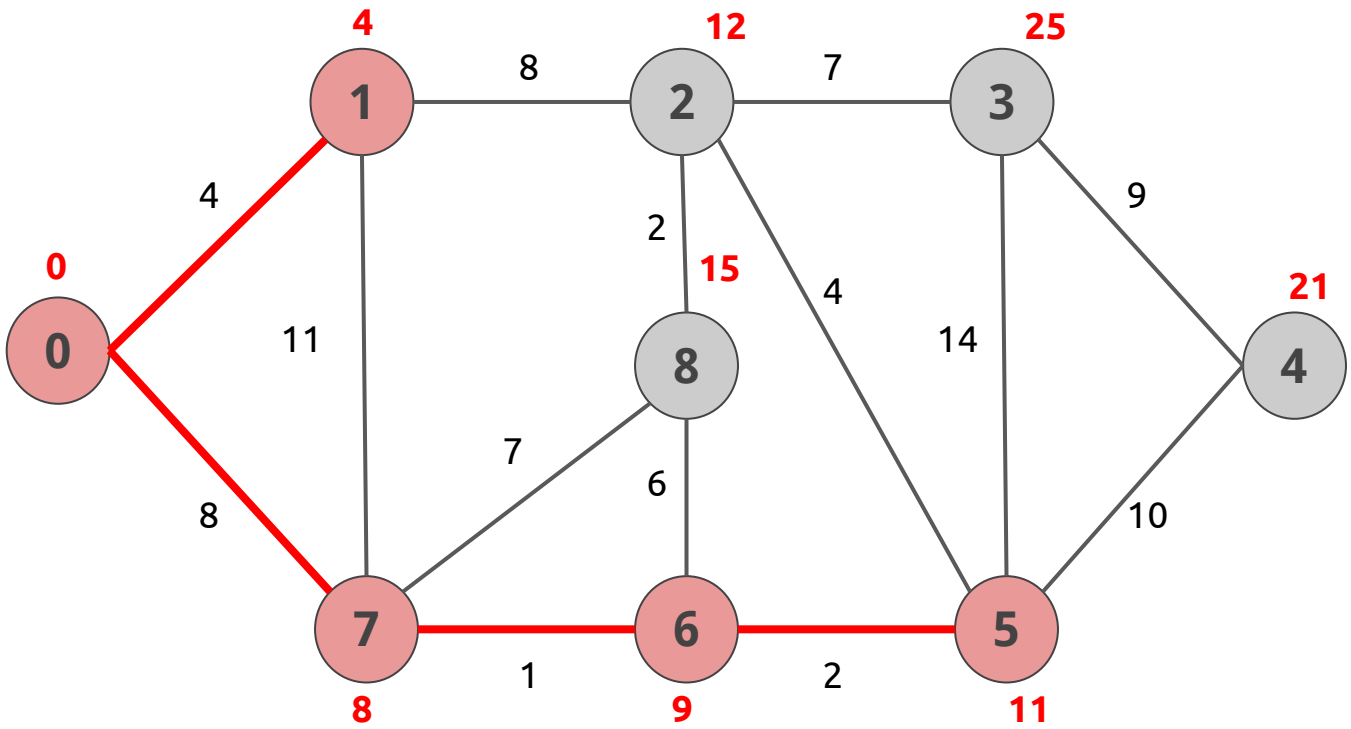
camino más corto: algoritmo de Dijkstra



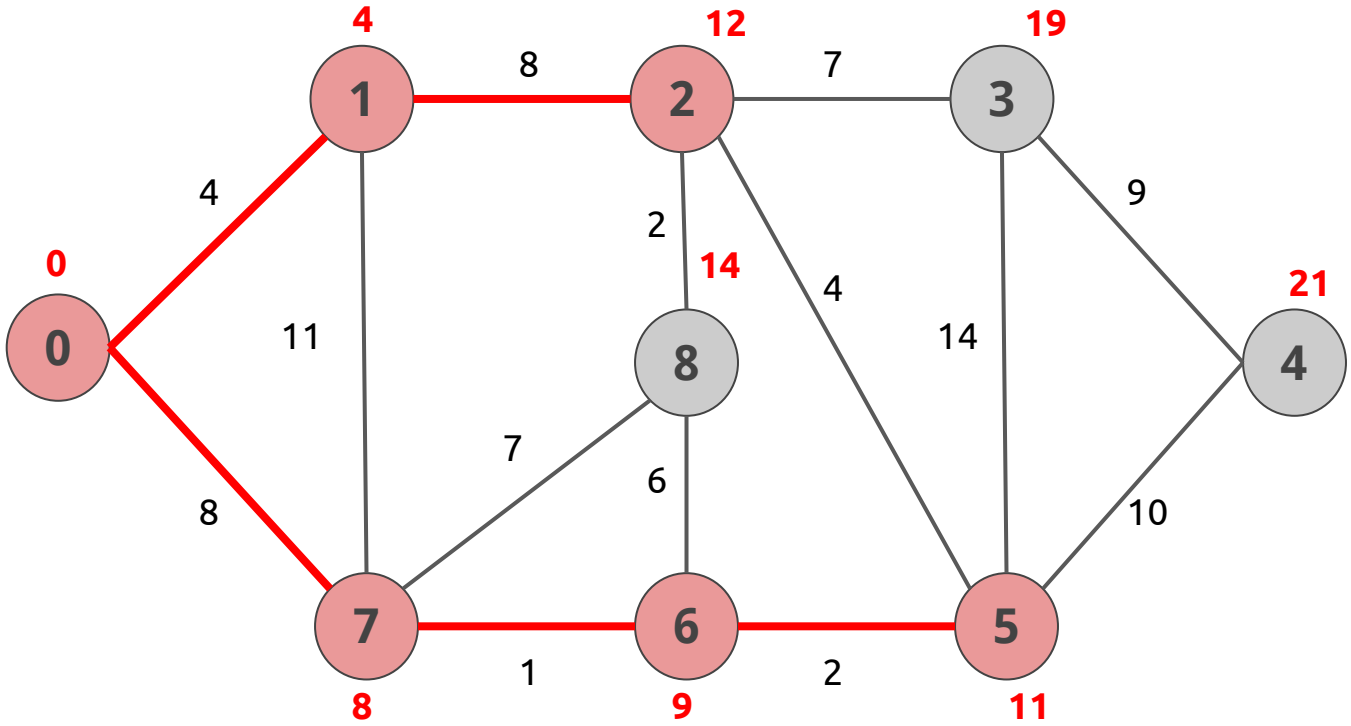
camino más corto: algoritmo de Dijkstra



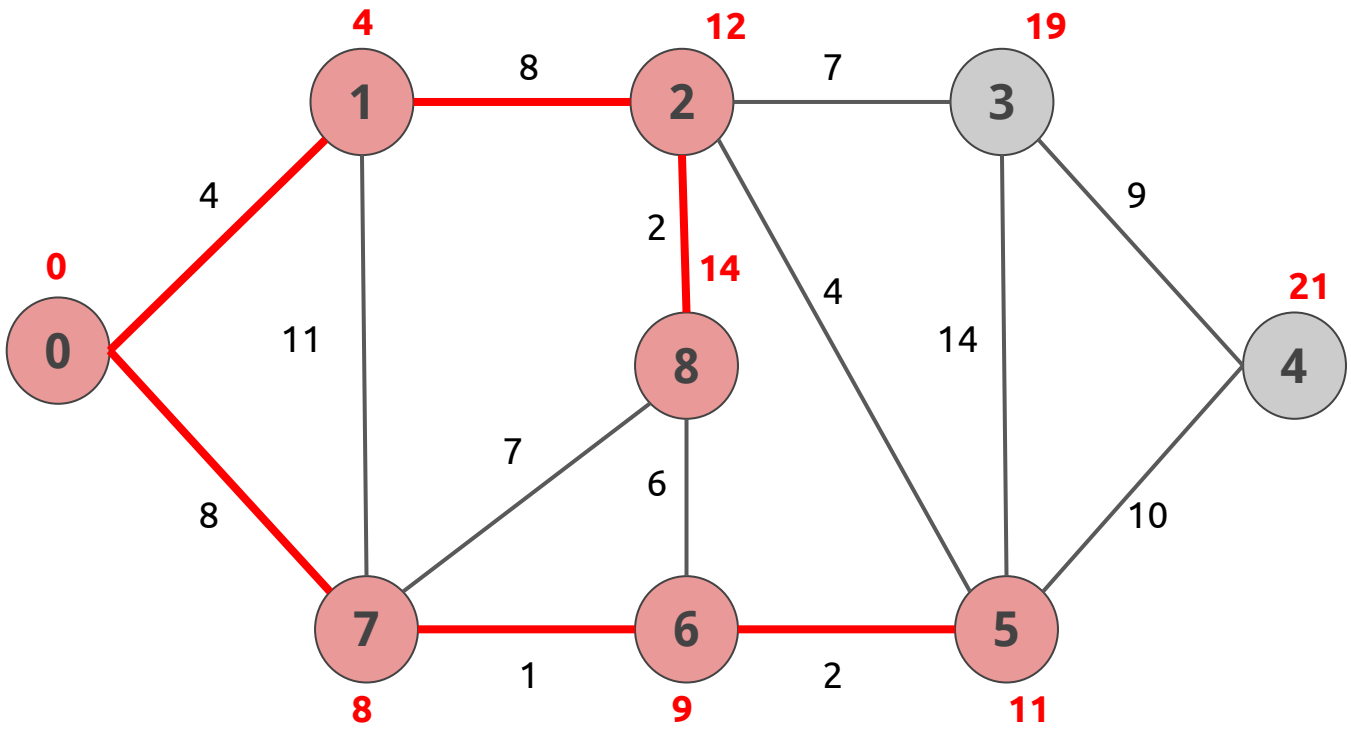
camino más corto: algoritmo de Dijkstra



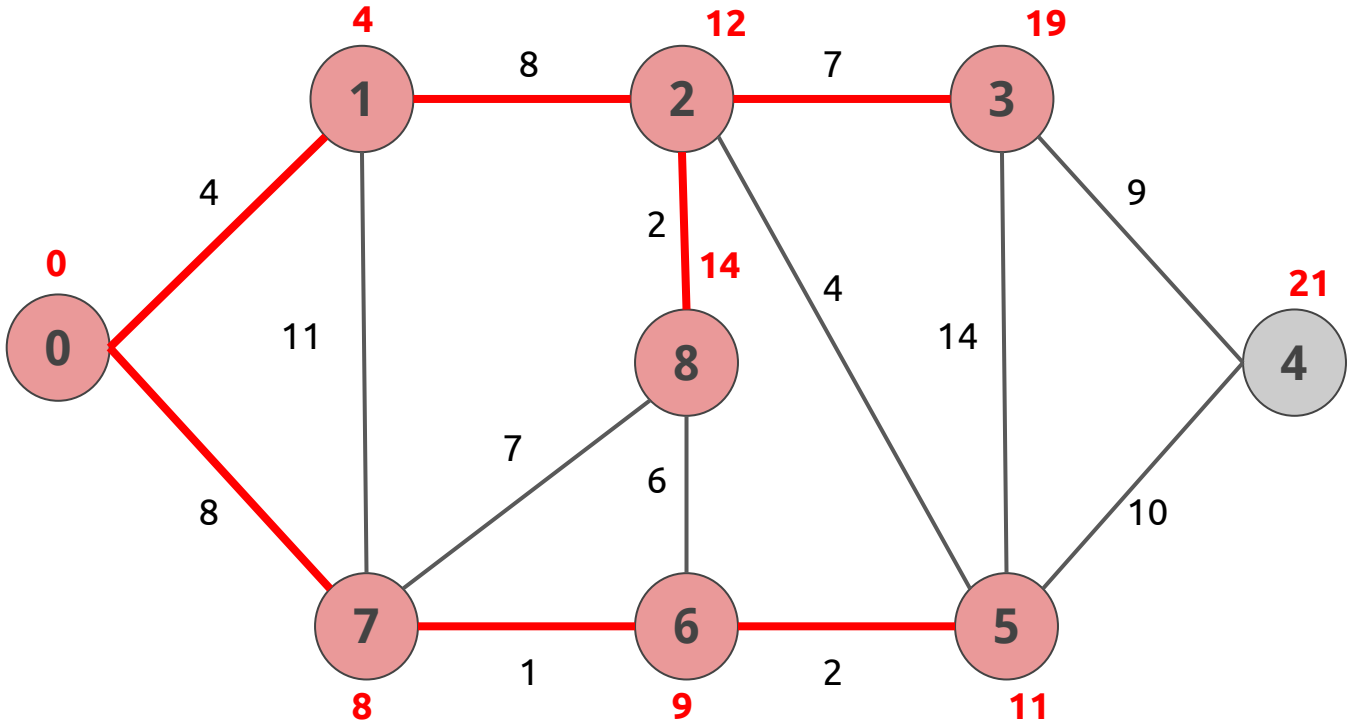
camino más corto: algoritmo de Dijkstra



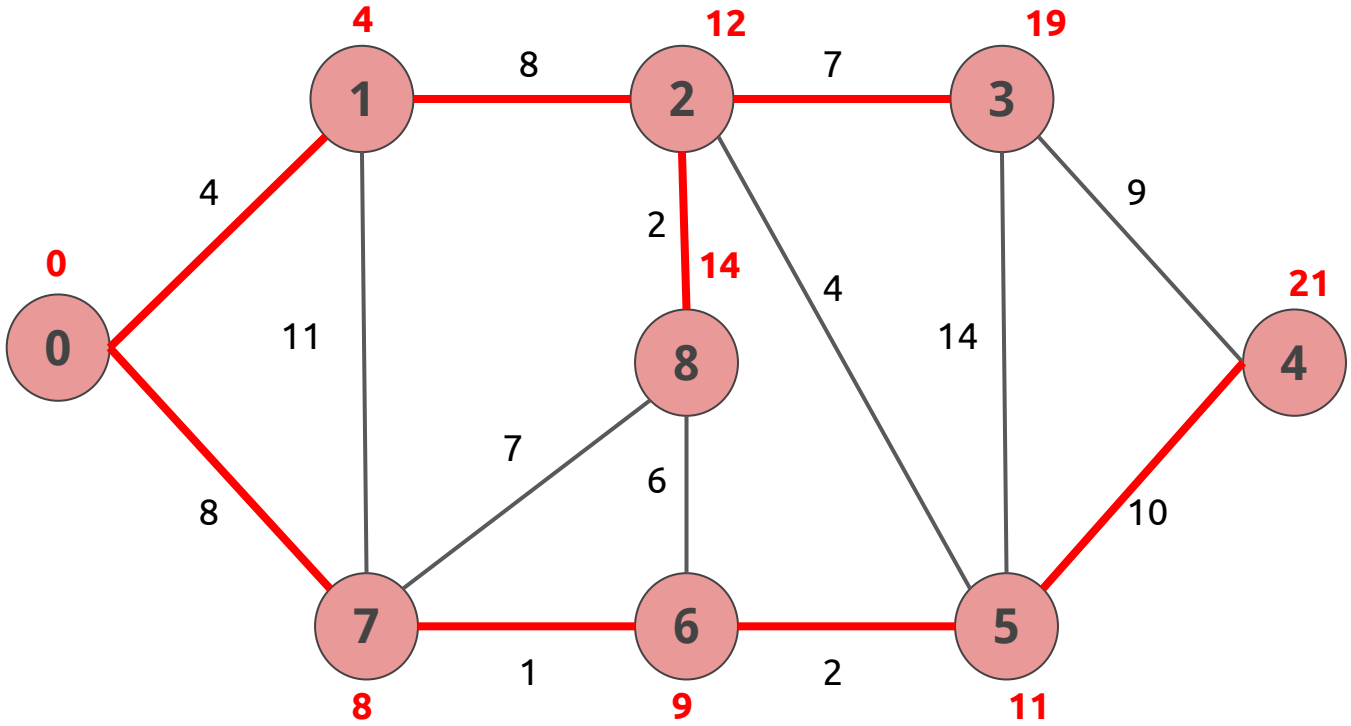
camino más corto: algoritmo de Dijkstra



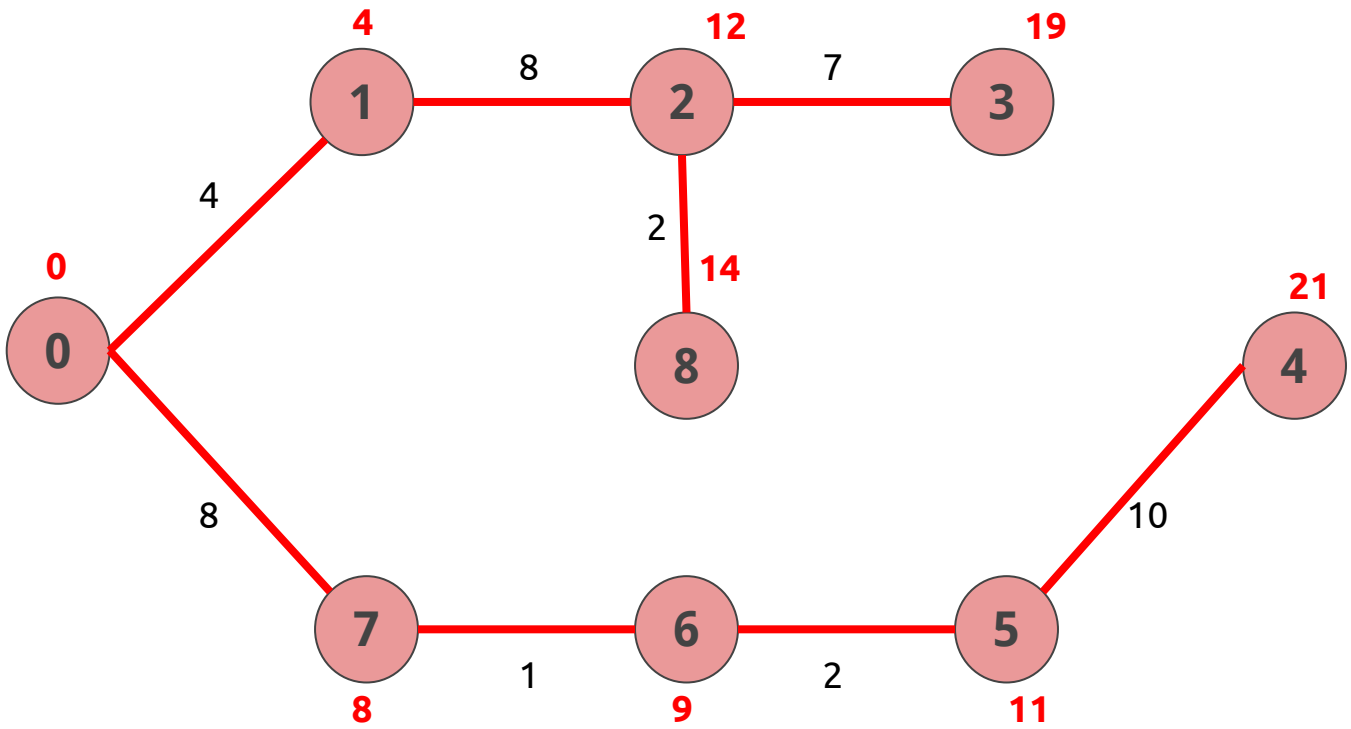
camino más corto: algoritmo de Dijkstra



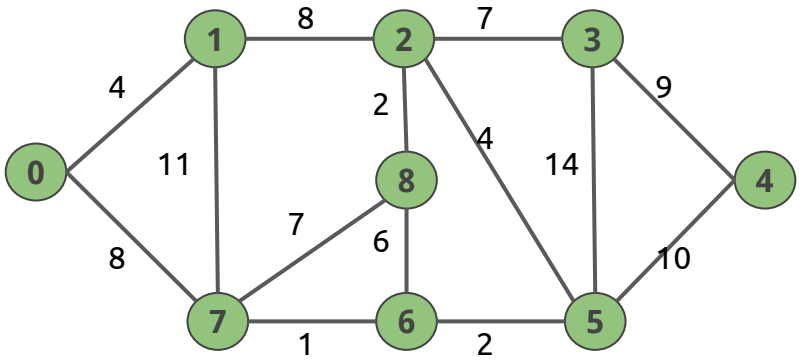
camino más corto: algoritmo de Dijkstra



camino más corto: algoritmo de Dijkstra



camino más corto: algoritmo de Dijkstra



	0	1	2	3	4	5	6	7	8
0	0_0	4_0	∞	∞	∞	∞	∞	8_0	∞
1		4_0	12_1	∞	∞	∞	∞	8_0	∞
7			12_1	∞	∞	∞	9_6	8_0	15_7
6			12_1	∞	∞	11_6	9_6		15_7
5			12_1	25_5	21_5	11_6			15_7
2			12_1	25_5	21_5				14_2
8				19_2	21_5				14_2
3				19_2	21_5				
4					21_5				

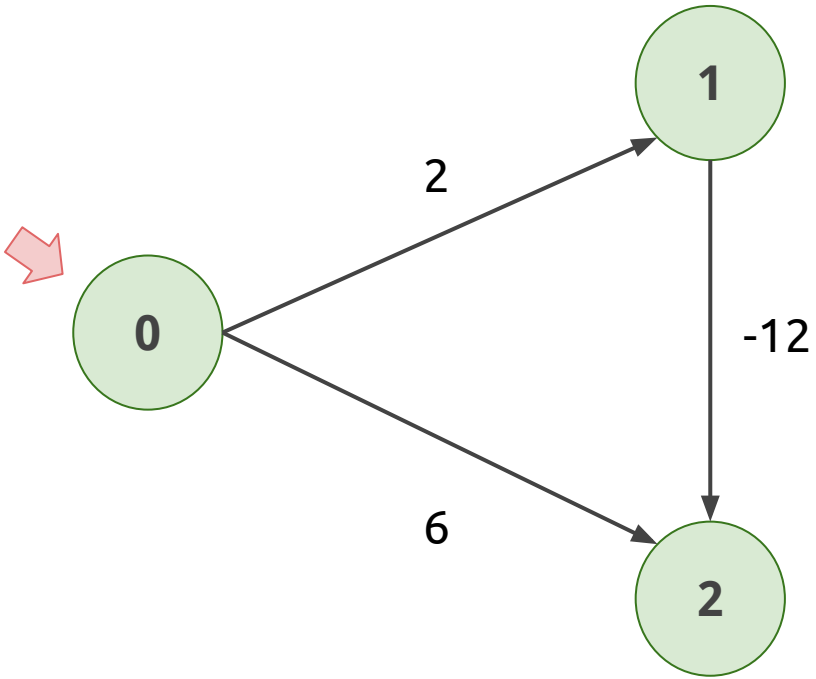
algoritmo de Dijkstra

```
DIJKSTRA (Grafo, origen)
     para u ∈ V[Grafo] hacer
         distancia[u] = INFINITO
         padre[u] = NULL
         spt[u] = false
     distancia[s] = 0
     agregar (cola, (s, distancia[s]))
     mientras que cola no es vacía hacer
         u = extraer_mínimo(cola)
         spt[u] = true
         para todos v ∈ adyacencia[u] hacer
             si distancia[v] > distancia[u] + peso (u, v) hacer
                 distancia[v] = distancia[u] + peso (u, v)
                 padre[v] = u
                 agregar (cola,(v, distancia[v]))
```

Ejercicio 1.16.

Implementá una función que calcule los caminos mínimos a todos los nodos desde un nodo de origen utilizando el algoritmo de Dijkstra.

Dijkstra ¿funciona con pesos negativos?



camino más corto: algoritmo de Floyd Warshall

El algoritmo de Floyd Warshall calcula los caminos mínimos entre **todos los pares** de vértices del grafo

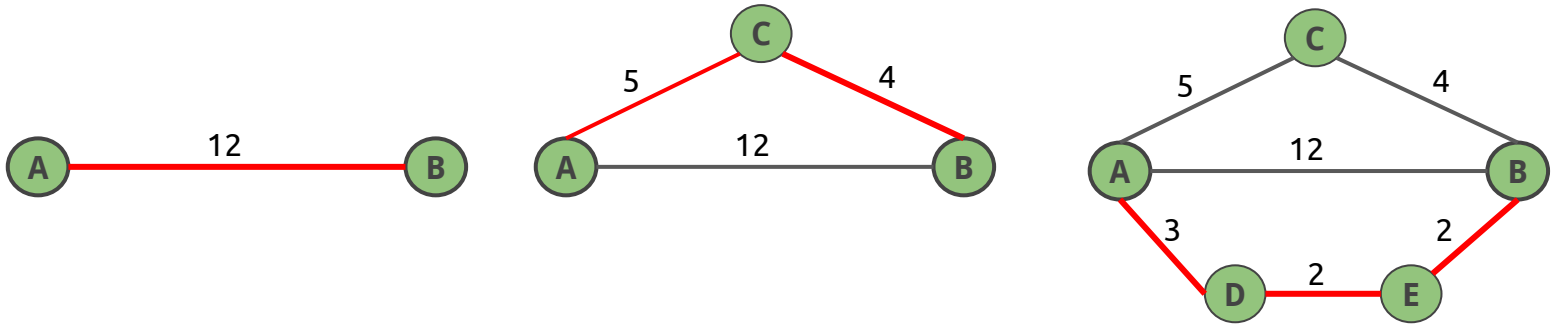
Funciona para pesos negativos

Es un ejemplo de programación dinámica

Floyd introduce una variación para grafos ponderados en el algoritmo de Warshall para hallar la matriz de clausura transitiva

camino más corto: algoritmo de Floyd Warshall

La idea es considerar progresivamente todos los caminos intermedios entre dos nodos i y j



Ejercicio 1.17.

Implementá una función que calcule los caminos mínimos de todos los nodos utilizando el algoritmo de Floyd - Warshall.

camino más corto: algoritmo de Bellman - Ford

Funciona para pesos negativos, aunque no para ciclos de peso negativo.

Es un ejemplo de programación dinámica, esto es: considera todas las posibles alternativas y escoge la mejor.

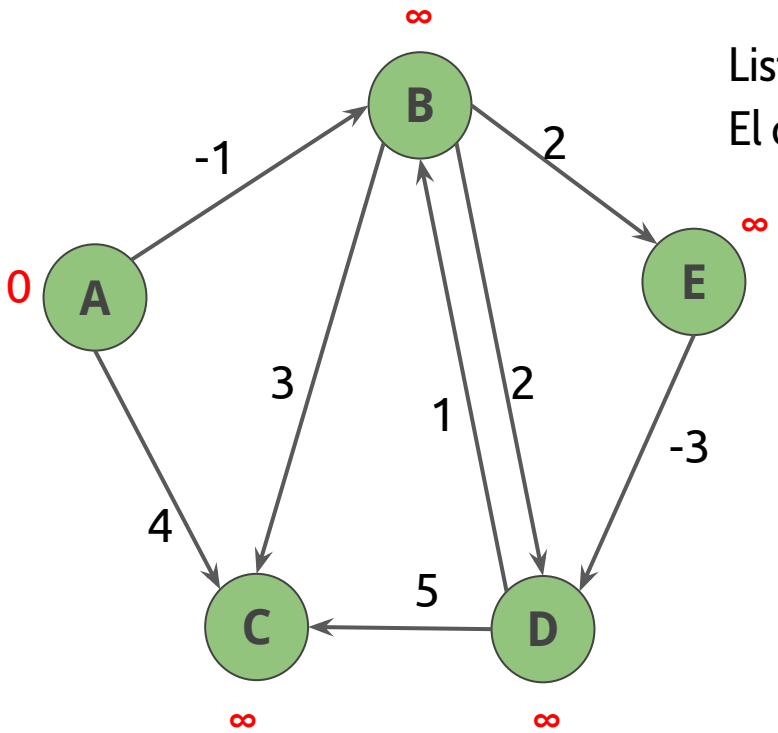
Como el camino al nodo más lejano del origen será de longitud V-1 como máximo, se relajan todas las aristas V-1 veces como máximo

algoritmo de Bellman - Ford

Relajación de una:

Si entre dos vértices (u,v) hay una arista
Si distancia[u] + costo(u,v) < distancia[v]
entonces distancia[v] = distancia[u] + costo(u,v)

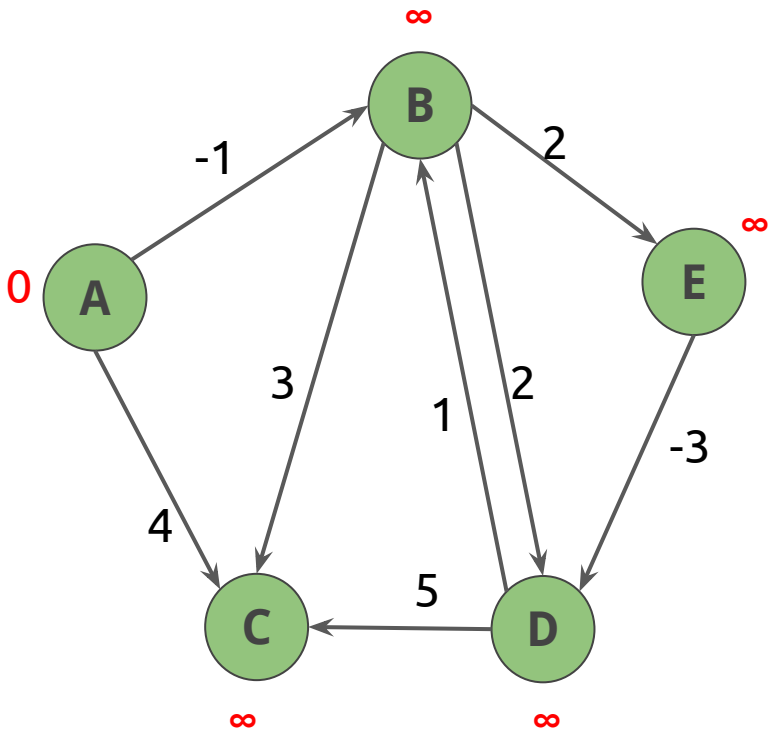
camino más corto: algoritmo de Bellman - Ford



Lista de aristas: (B,E), (D,B), (B,D) , (A,B), (A,C), (D,C), (B,C) (E,D)

El órden es arbitrario

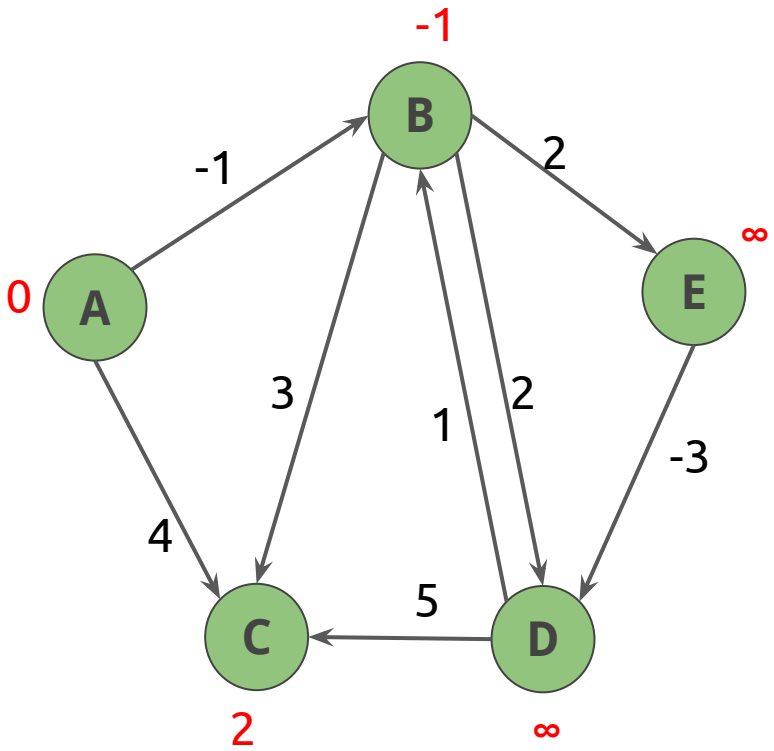
camino más corto: algoritmo de Bellman - Ford



A	B	C	D	E
0	∞	∞	∞	∞

Orden de procesamiento: (B,E), (D,B), (B,D) , (A,B), (A,C), (D,C), (B,C) (E,D)

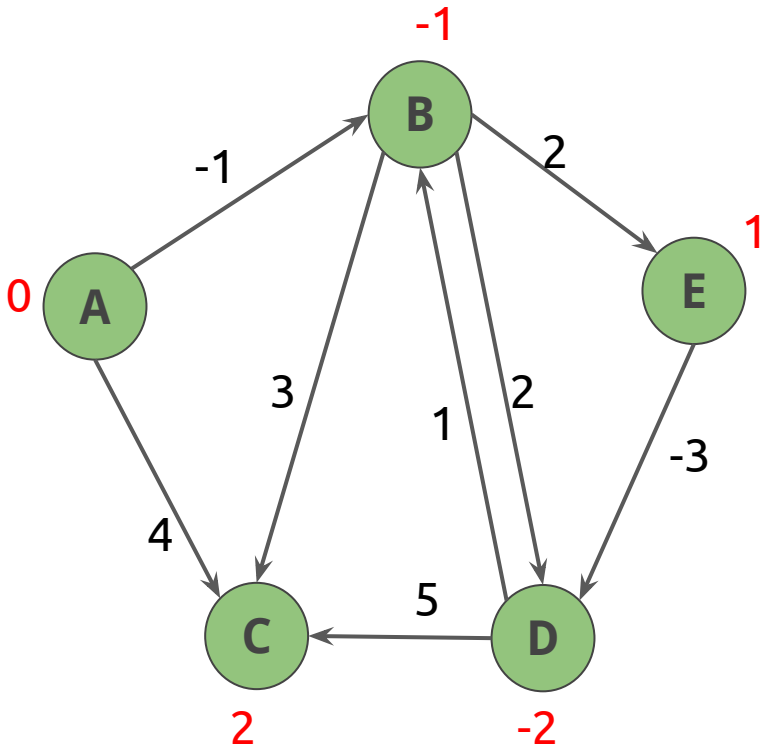
camino más corto: algoritmo de Bellman - Ford



A	B	C	D	E	
0	∞	∞	∞	∞	
0	-1	∞	∞	∞	(B,E), (D,B), (B,D) (A,B)
0	-1	4	∞	∞	(A,C),
0	-1	2	∞	∞	(D,C), (B,C) (E,D)

Orden de procesamiento: (B,E), (D,B), (B,D) , (A,B), (A,C), (D,C), (B,C) (E,D)

camino más corto: algoritmo de Bellman - Ford



A	B	C	D	E	
0	∞	∞	∞	∞	
0	-1	∞	∞	∞	(B,E), (D,B), (B,D) (A,B)
0	-1	4	∞	∞	(A,C),
0	-1	2	∞	∞	(D,C), (B,C) (E,D)
0	-1	2	∞	1	(B,E),
0	-1	2	1	1	(D,B), (B,D)
0	-1	2	-2	1	(A,B), (A,C), (D,C), (B,C) (E,D)

Orden de procesamiento: (B,E), (D,B), (B,D) , (A,B), (A,C), (D,C), (B,C) (E,D)

Ejercicio 1.18.

Implementá una función que calcule los caminos mínimos de todos los nodos utilizando el algoritmo de Bellman - Ford.

caminos mínimos: análisis de complejidad

- Dijkstra
  - La complejidad del algoritmo es de **$O(V^2)$** si no se usa colas de prioridad
  - Si se usan colas de prioridad la complejidad es **$O(A . \log V)$**
- Bellman - Ford
  - Tiene complejidad mayor que el algoritmo de Dijkstra **$O(A . V)$**
- Floyd Warshall
  - La complejidad del algoritmo es de **$O(V^3)$**