Clase 27 - Grafos

IIC1001 - Algoritmos y Sistemas Computacionales

Cristian Ruz – cruz@uc.cl

Jueves 18-Junio-2026

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Contenidos

Contacto

Temas

Grafos

Contenidos

Contacto

Temas

Grafos

Contacto

🐐

vicente.cabra@uc.cl

Vicente Cabra

Ayudante jefe



🐸

alejandro.tapia@uc.cl

Alejandro Tapia

Ayudante jefe



🛼

gabriel.sandoval@uc.cl

Gabriel Sandoval

Ayudante jefe



Contenidos

Contacto

Temas

Grafos

Temas del curso

Sistemas computacionales

- Representación datos, números y codificación
- Funcionamiento hardware, procesadores y memoria.
- Funcionamiento de sistemas operativos
- Funcionamiento de Internet
- Herramientas computacionales: github + latex

Algoritmos

- Algoritmos y resolución de problemas
- Eficiencia algorítmica
- Estructuras secuenciales y ordenamiento
- Grafos y árboles

Próximos eventos

Viernes 5-Junio: Recorrección presencial I1 y ACs

- 13:30-15:40
- Sala H1

Martes 9-Junio: Algoritmos de ordenamiento

Jueves 11-Junio: Actividad 6

- En clase

Viernes 12-Junio: Notas I2

Contenidos

Contacto

Temas

Grafos

Motivación

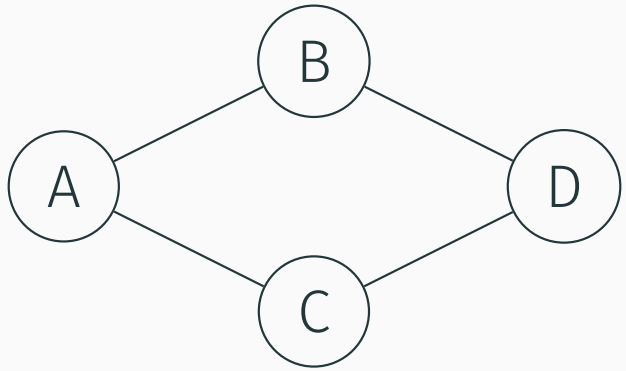
¿Qué tienen en común Google Maps, Instagram y una red de computadores?

Todos pueden modelarse mediante una misma estructura:

Grafo

Ejemplo 1: Red Social

- Cada persona es un nodo.
- Una amistad es una conexión.
- Podemos estudiar:
  - Influencia.
  - Comunidades.
  - Recomendaciones de contactos.



Ejemplo 2: Google Maps

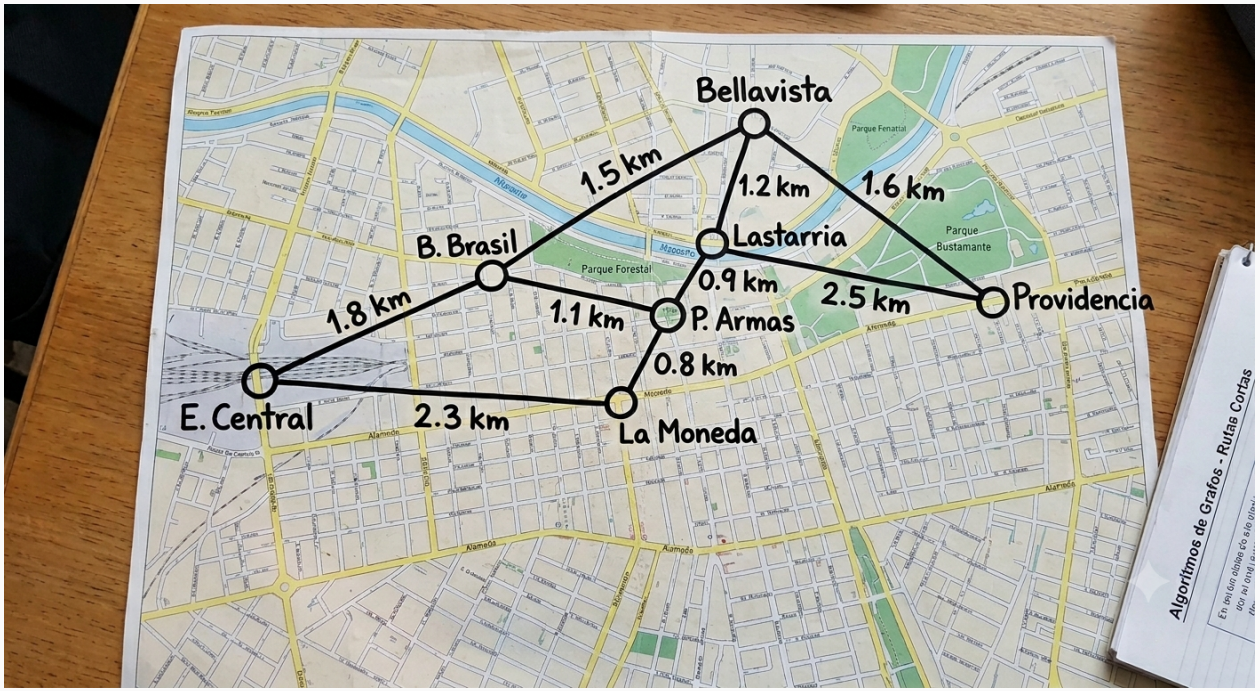
- Intersecciones $\rightarrow$ nodos.
- Calles $\rightarrow$ aristas.
- Distancia o tiempo $\rightarrow$ peso.

Problema típico:

Encontrar la ruta más corta.

Ejemplo 2: Google Maps

- Intersecciones $\rightarrow$ nodos.
- Calles $\rightarrow$ aristas.
- Distancia o tiempo $\rightarrow$ peso.



Ejemplo 3: Red de Computadores

- Routers como nodos.
- Enlaces de comunicación como aristas.
- Permiten estudiar conectividad y enrutamiento.

Definición Formal

Grafo
Un grafo es un par:

$$G = (V, E)$$

donde:

- V: conjunto de vértices (nodos).
- E: conjunto de aristas (conexiones).

Tipos de Grafos

No dirigido

- Las conexiones son bidireccionales.

$$(A, B) = (B, A)$$

Ejemplos:

- Facebook: no dirigido.
- Twitter/X/Instagram: dirigido.

Dirigido

- Las conexiones tienen dirección.

$$A \to B$$

Grafos Ponderados

Las aristas pueden tener pesos.

Ejemplos:

- Distancia.
- Tiempo.
- Costo.
- Capacidad.

$$(A, B) = 10$$

indica que el costo de viajar entre A y B es 10.

Lista de adyacencia

$$A : B, C$$

$$B : A, D$$

$$C : A, D$$

Ventajas:

- Poco espacio.
- Ideal para grafos dispersos.

Matriz de adyacencia

	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

Ventaja:

- Consulta rápida de conexiones.

Problemas clásicos sobre grafos

Dado un grafo, queremos responder preguntas como:

- ¿Existe un camino entre dos nodos?
- ¿Cuál es el camino más corto?
- ¿Están todos los nodos conectados?
- ¿Existen ciclos?
- ¿Cómo recorrer toda la red?

Cada una de estas preguntas motiva algoritmos específicos.

Problema Fundamental

Dado un grafo:

¿Cómo lo recorremos?

Aplicaciones:

- Búsqueda.
- Navegación.
- Descubrimiento de componentes.
- Planificación.

Breadth First Search (BFS)

Idea:

Explorar primero los vecinos más cercanos.

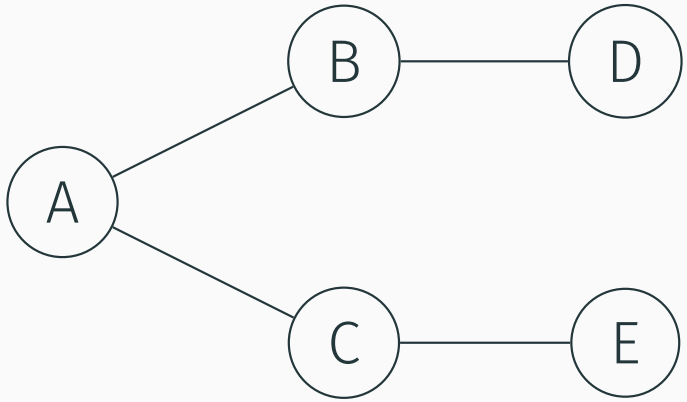
Utiliza una:

$$\boxed{\text{Cola (Queue)}}$$

Orden de exploración:

$$0 \to 1 \to 2 \to 3 \to \cdots$$

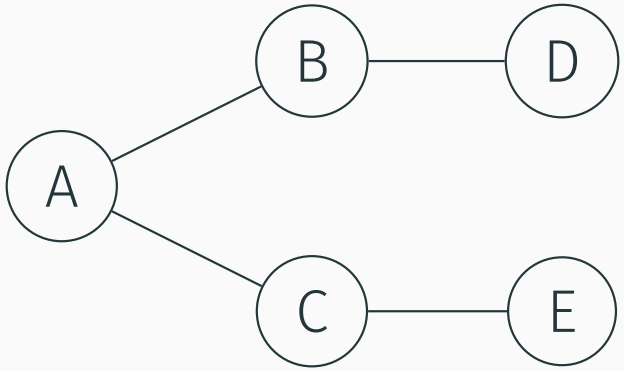
BFS: Ejemplo



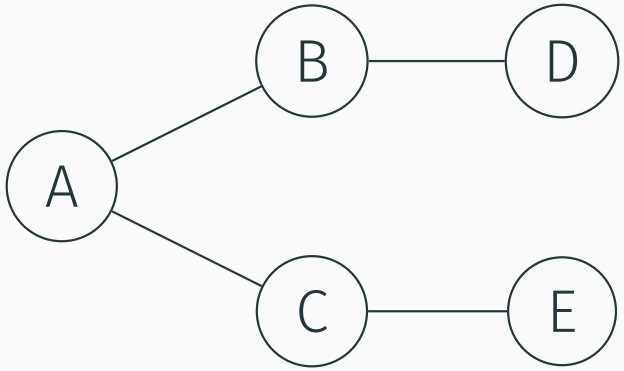
Comenzando en A:

$$A, B, C, D, E$$

BFS: Exploración por niveles



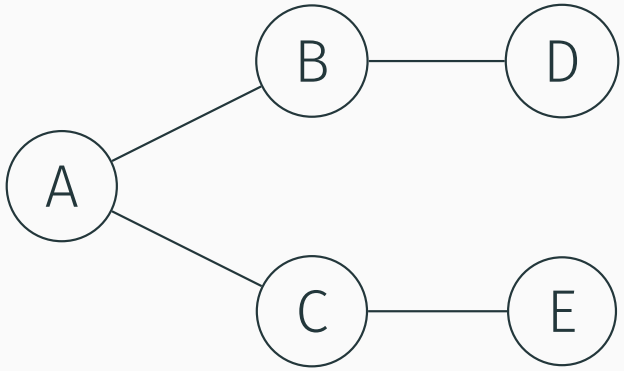
BFS: Exploración por niveles



Nivel 0:

A

BFS: Exploración por niveles



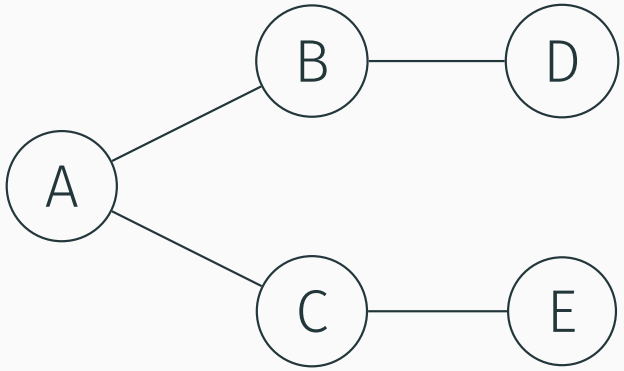
Nivel 0:

$$A$$

Nivel 1:

$$B, C$$

BFS: Exploración por niveles



Nivel 0:

A

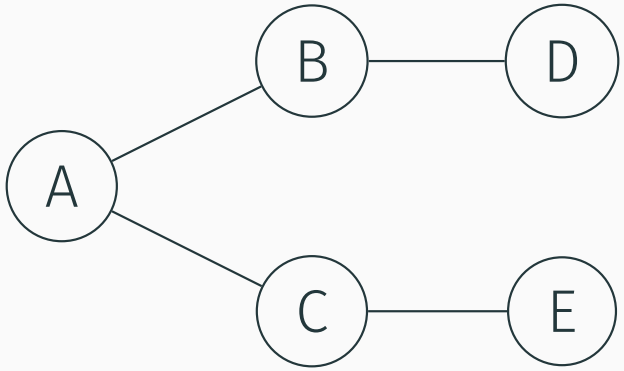
Nivel 1:

B, C

Nivel 2:

D, E

BFS: Exploración por niveles



Nivel 0:

$$A$$

Nivel 1:

$$B, C$$

Nivel 2:

$$D, E$$

Recorrido BFS: A, B, C, D, E

Pseudocódigo BFS

```
BFS(G, s)

crear cola Q
marcar s como visitado
encolar(Q, s)
mientras Q no esté vacía
    u = desencolar(Q)
    visitar(u)
    para cada vecino v de u
        si v no ha sido visitado
            marcar v como visitado
            encolar(Q, v)
```

Utiliza una cola (FIFO).

Aplicaciones de BFS

- Encontrar caminos mínimos en grafos sin pesos.
- Redes sociales:
  - amigos a distancia 1
  - amigos a distancia 2
- Propagación de información.
- Descubrir conectividad.

Complejidad:

$$O(V + E)$$

Depth First Search (DFS)

Idea:

Explorar lo más profundo posible antes de retroceder.

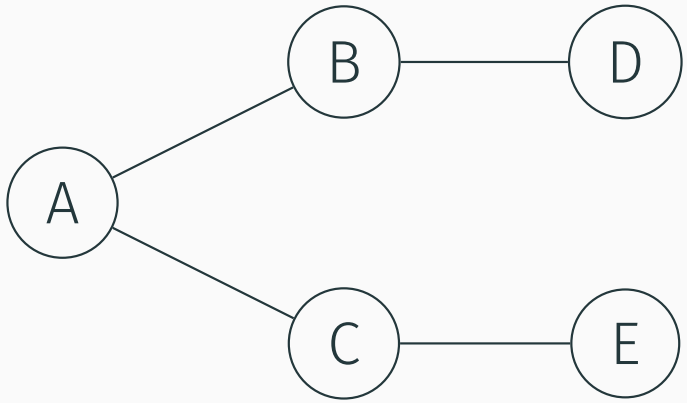
Utiliza:

Pila (Stack)

o recursión.

DFS: Ejemplo

Mismo grafo.



Posible recorrido:

$$A, B, D, C, E$$

La idea es seguir un camino hasta el fondo y luego regresar.

DFS: Exploración en profundidad

DFS: Exploración en profundidad

Comenzamos en A

DFS: Exploración en profundidad

Comenzamos en A

Seguimos un camino:

$$A \to B$$

DFS: Exploración en profundidad

Comenzamos en A

Seguimos un camino:

$$A \to B$$

Continuamos:

$$A \to B \to D$$

DFS: Exploración en profundidad

Comenzamos en A

Seguimos un camino:

$$A \to B$$

Continuamos:

$$A \to B \to D$$

Retrocedemos y exploramos:

$$C \to E$$

DFS: Exploración en profundidad

Comenzamos en A

Seguimos un camino:

$$A \to B$$

Continuamos:

$$A \to B \to D$$

Retrocedemos y exploramos:

$$C \to E$$

Recorrido posible: A, B, D, C, E

Pseudocódigo DFS

```
DFS(G, u)

marcar u como visitado
visitar(u)
para cada vecino v de u
    si v no ha sido visitado
        DFS(G, v)
```

Versión recursiva.

La pila de llamadas actúa como un stack.

Aplicaciones de DFS

- Detección de ciclos.
- Componentes conexas.
- Ordenamiento topológico.
- Exploración exhaustiva.
- Backtracking.

Complejidad:

$$O(V + E)$$

Componentes conexas

Una componente conexa es un conjunto de nodos donde todos son alcanzables entre sí. Aplicaciones:

- Detectar comunidades.
- Analizar redes sociales.
- Verificar conectividad de una red.

DFS y BFS permiten encontrarlas eficientemente.

Detección de ciclos

Pregunta:

¿Podemos volver al punto de partida siguiendo las aristas?

Ejemplos:

- Dependencias circulares.
- Rutas de transporte.
- Sistemas de planificación.

DFS es una herramienta fundamental para detectarlos.

BFS vs DFS

	BFS	DFS
Estructura	Cola	Pila
Camino más corto	Sí	No
Exploración	Por niveles	En profundidad

Problema de Camino Más Corto

Supongamos que las aristas tienen pesos.

Ahora BFS ya no es suficiente.

Necesitamos un algoritmo especializado:

Dijkstra

Algoritmo de Dijkstra

Objetivo:

Encontrar la distancia mínima desde un nodo origen a todos los demás.

Requiere:

- Pesos no negativos.
- Cola de prioridad.

Algoritmo de Dijkstra

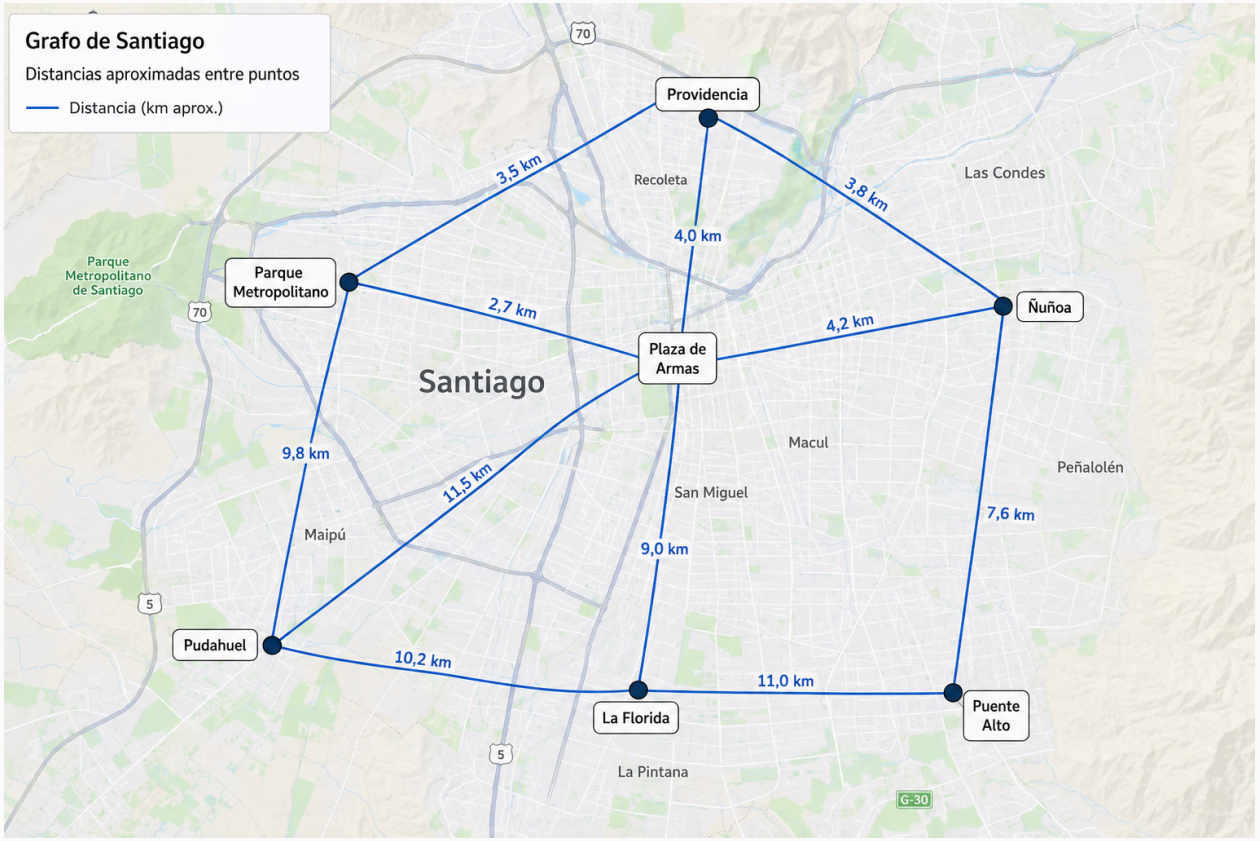
Objetivo:

Encontrar la distancia mínima desde un nodo origen a todos los demás.

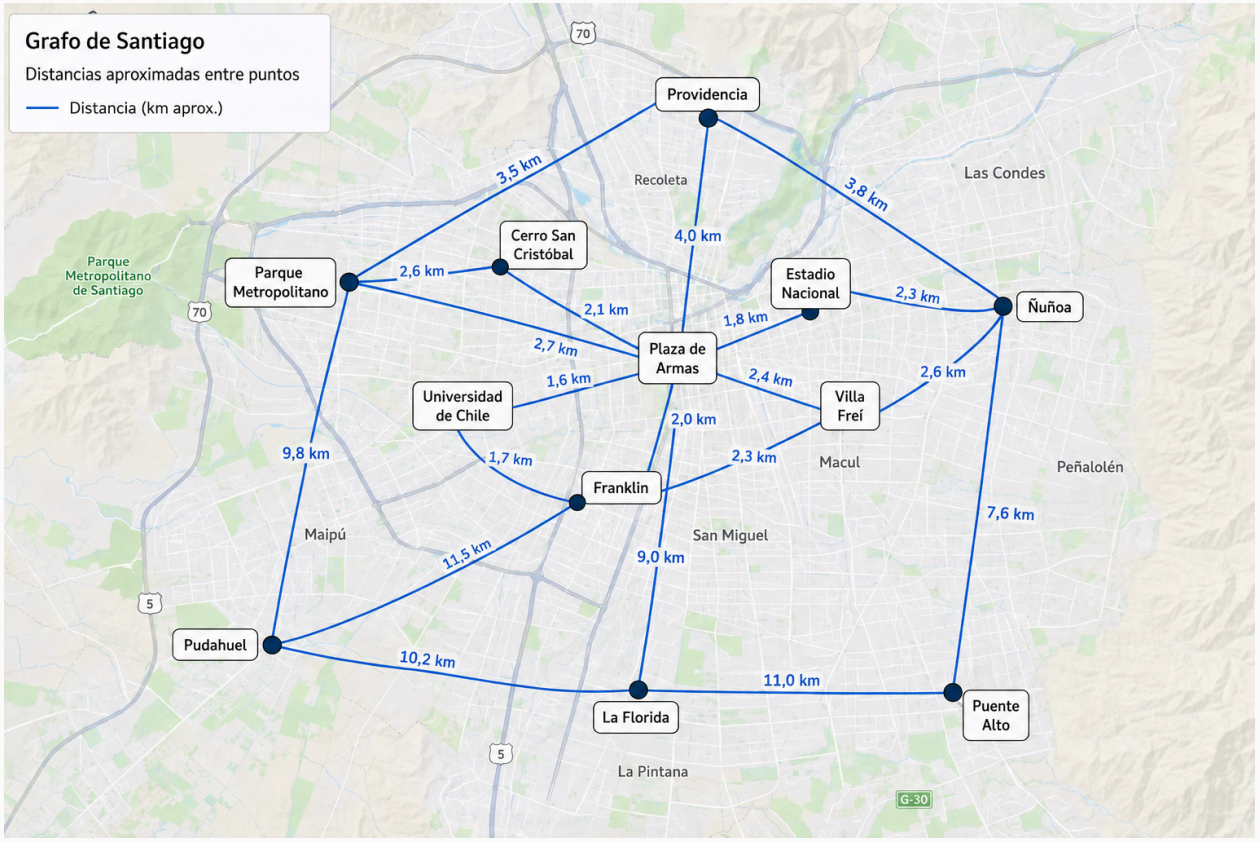
Requiere:

- Pesos no negativos.
- Cola de prioridad.

Idea de Dijkstra



Idea de Dijkstra



Ejemplo de Dijkstra

$$A \to B = 4$$

$$A \to C = 2$$

$$C \to B = 1$$

Ruta más corta de A a B:

$$A \to C \to B$$

Costo:

$$2 + 1 = 3$$

Dijkstra: Evolución de distancias

Nodo origen: A

$$d(A) = 0, d(B) = \infty, d(C) = \infty$$

Dijkstra: Evolución de distancias

Nodo origen: A

$$d(A) = 0, d(B) = \infty, d(C) = \infty$$

Después de visitar A:

$$d(B) = 4, d(C) = 2$$

Dijkstra: Evolución de distancias

Nodo origen: A

$$d(A) = 0, d(B) = \infty, d(C) = \infty$$

Después de visitar A:

$$d(B) = 4, d(C) = 2$$

Visitamos C:

$$d(B) = 3$$

Dijkstra: Evolución de distancias

Nodo origen: A

$$d(A) = 0, d(B) = \infty, d(C) = \infty$$

Después de visitar A:

$$d(B) = 4, d(C) = 2$$

Visitamos C:

$$d(B) = 3$$

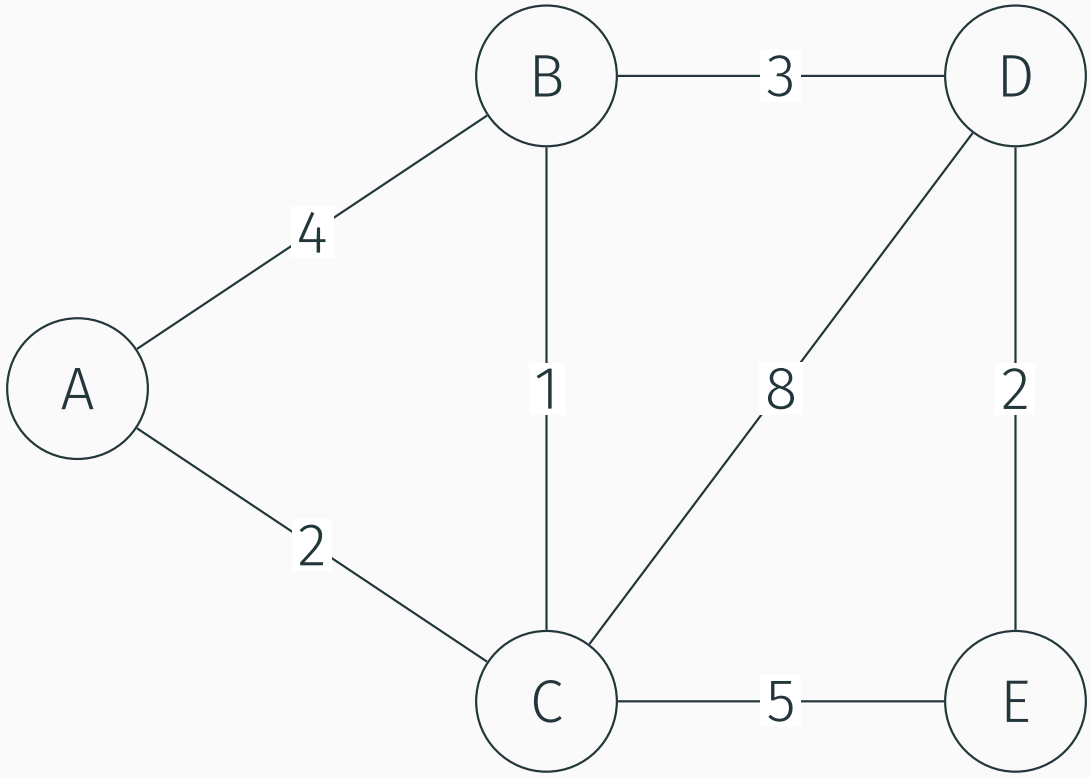
Resultado final:

$$A \to C \to B$$

Costo:

$$3$$

Ejemplo: Grafo para Dijkstra



Objetivo:

Encontrar las distancias mínimas desde el nodo A.

Dijkstra: Inicialización

Nodo origen:

A

Nodo	Distancia
A	0
B	∞
C	∞
D	∞
E	∞

Conjunto visitado: $\emptyset$

Paso 1: Visitar A

Seleccionamos el nodo con menor distancia: A

Actualizamos sus vecinos: $d(B) = 4, d(C) = 2$

Nodo	Distancia
A	0
B	4
C	2
D	∞
E	∞

Visitados:$\{A\}$

Paso 2: Visitar C

Entre los nodos no visitados: $\min(4, 2, \infty, \infty) = 2$

Seleccionamos:C

Revisamos sus vecinos. $A \to C \to B = 2 + 1 = 3$

Mejoramos:$d(B) = 3$

$$A \to C \to D = 2 + 8 = 10, d(D) = 10$$

$$A \to C \to E = 2 + 5 = 7, d(E) = 7$$

Nodo	Distancia
A	0
B	3
C	2
D	10
E	7

Paso 3: Visitar B

El nodo no visitado más cercano es:

$$B$$

con distancia:

$$3$$

Evaluamos:

$$A \to C \to B \to D = 2 + 1 + 3 = 6$$

Mejoramos:

$$d(D) = 6$$

Paso 4: Visitar D

Nodo seleccionado:

$$D$$

Distancia actual:

$$6$$

Evaluamos:

$$A \to C \to B \to D \to E = 6 + 2 = 8$$

Comparación:

$$8 > 7$$

Paso 5: Visitar E

Último nodo pendiente:

$$E$$

No hay mejoras posibles.

Distancias finales:

Nodo	Distancia mínima desde A
A	0
B	3
C	2
D	6
E	7

Resultado Final

Nodo	Distancia	Camino mínimo
A	0	A
B	3	A-C-B
C	2	A-C
D	6	A-C-B-D
E	7	A-C-E

Observación:

- El camino más corto no siempre utiliza la arista más directa.
- Dijkstra construye progresivamente una solución óptima.

Aplicaciones de Dijkstra

- GPS y navegación.
- Redes de computadores.
- Videojuegos.
- Logística y transporte.
- Planificación de rutas.

Complejidades

Algoritmo	Complejidad
BFS	$O(V + E)$
DFS	$O(V + E)$
Dijkstra (heap)	$O((V + E) \log V)$

Donde:

- V: número de vértices.
- E: número de aristas.

Pseudocódigo de Dijkstra

```
Dijkstra(G, s)

para cada vertice v
    distancia[v] = infinito
distancia[s] = 0

crear cola de prioridad PQ
insertar (0,s) en PQ

mientras PQ no esté vacía
    u = extraer_min(PQ)
    para cada vecino v de u
        nueva = distancia[u] + peso(u,v)
        si nueva < distancia[v]
            distancia[v] = nueva
            insertar (distancia[v], v) en PQ

retornar distancia
```


Utiliza una cola de prioridad (min-heap).

Resumen

- Los grafos modelan relaciones y redes.
- Formalmente: $G = (V, E)$.
- BFS explora por niveles.
- DFS explora en profundidad.
- Ambos recorren grafos en $O(V + E)$.
- Dijkstra encuentra caminos mínimos con pesos positivos.

¡Los grafos están en todas partes!