Algoritmia y Estructura de Datos

2024

Profesores:
Cueva, R. | **Allasi**, D. | **Roncal**, A. | **Huamán**, F.
0581 | 0582 | 0583 | 0584

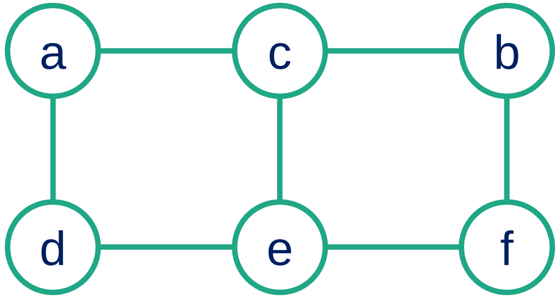
Capítulo 4

Grafos

PUCP

Definición

- Un **grafo** G = (V, E) está definido por el par de conjuntos:
  - **V:** conjunto finito no vacío de elementos llamados **vértices**
  - **E:** conjunto de pares de vértices llamados **aristas**



*Grafo **G***

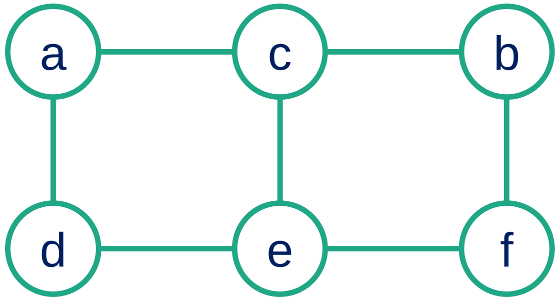
V *={a,b,c,d,e,f}*

E*={(a,c),(a,d),(b,c),(b,f),(c,e),(d,e),(e,f)}*

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición.
USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Grafos No Dirigidos

- Si las aristas *(u, v)* y *(v, u)* son las mismas, se dice que los vértices *u* y *v* son **adyacentes** y que están conectados por la **arista no dirigida** *(u, v)*
- Un grafo *G* es llamado **no dirigido** si cada una de sus aristas es no dirigida



Grafo no dirigido ***G***

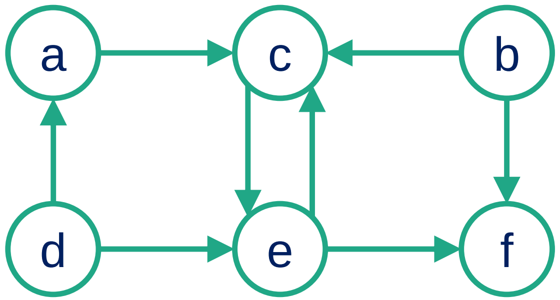
V *={a,b,c,d,e,f}*

E*={(a,c),(a,d),(b,c),(b,f),(c,e),(d,e),(e,f)}*

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Grafos Dirigidos o Digrafos

- Si las aristas *(u, v)* y *(v, u)* no son las mismas, se dice que la arista *(u, v)* está **dirigida** desde el vértice *u*
- Un grafo es llamado **dirigido** (o **digrafo**) si todas sus aristas son dirigidas



Grafo dirigido ***G***

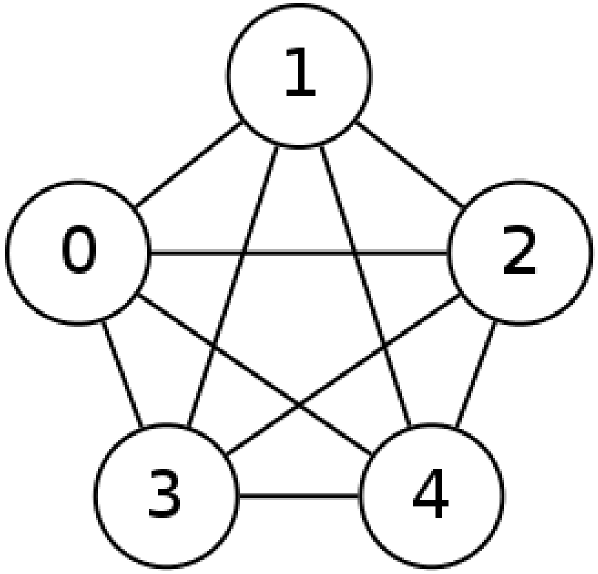
V *={a,b,c,d,e,f}*

E=*{*(a,c),(b,c),(b,f),(c,e),(d,a),(d,e),(e,c),(e,f)*}*

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Terminología

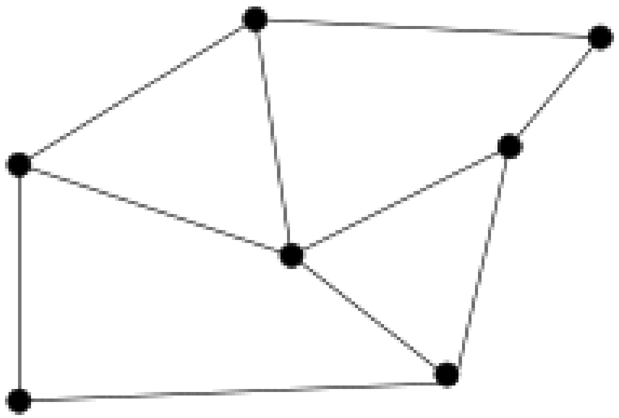
- Un grafo con todos sus pares de vértices conectados por una arista es llamado **completo**



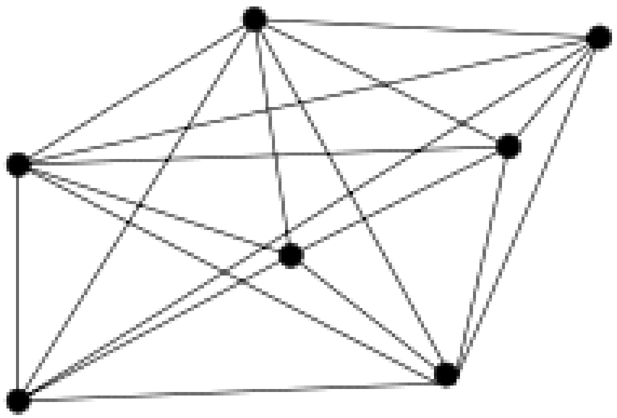
LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Terminología

- Un grafo con relativamente pocas aristas faltantes es llamado **denso**
- Un grafo con pocas aristas relativas al número de sus vértices es llamado **esparso**



esparso



denso

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Representación de Grafos

TAD Grafo: Operaciones

- Crear un grafo vacío
- Insertar una arista en un grafo
- Insertar un vértice en un grafo
- Verificar si existe determinada arista en el grafo
- Verificar si existe determinado vértice en el grafo
- Eliminar una arista del grafo
- Eliminar un vértice del grafo
- Imprimir un grafo
- Obtener el número de vértices del grafo

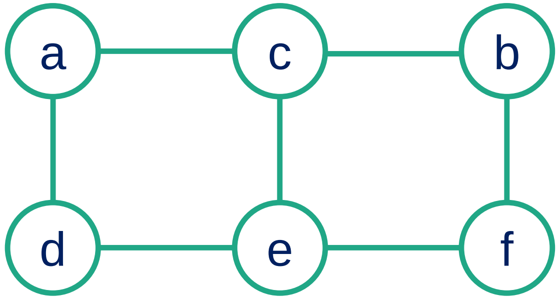
TAD Grafo: ¿Cómo se representa?

- La selección de la estructura de datos correcta para representar grafos tiene un impacto enorme en el desempeño de un algoritmo

- Representaciones usuales:
  - Matriz de adyacencia
  - Estructura de adyacencia (con listas de adyacencia)

Matriz de Adyacencia

- Dado un grafo *G = (V, E)*, la **matriz de adyacencia** *M* es una matriz de orden *n x n*, tal que:
  - **n** = número de vértices
  - **M[i, j] = 1**, si existe la arista del nodo *i* al nodo *j*
  - **M[i, j] = 0**, si <u>no</u> existe la arista del nodo *i* al nodo *j*



G = (V, E)

M =

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición.
USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Matriz de Adyacencia

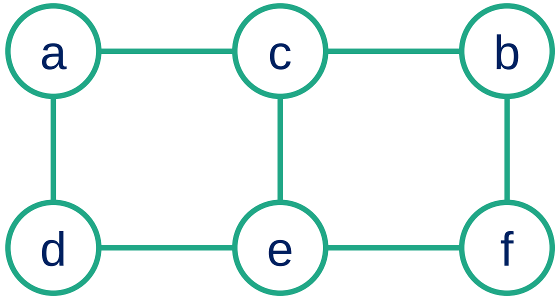
- Es la forma más simple de representación

- Propiedades:
  - Es **simétrica** para un grafo no dirigido
    - $M[i, j] = M[j, i]$, para todo $0 <= i, j <= n - 1$
  - Almacenamiento: $O(n^2)$
  - Prueba de si la arista (i,j) está en el grafo: $O(1)$

LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Estructura de Adyacencia

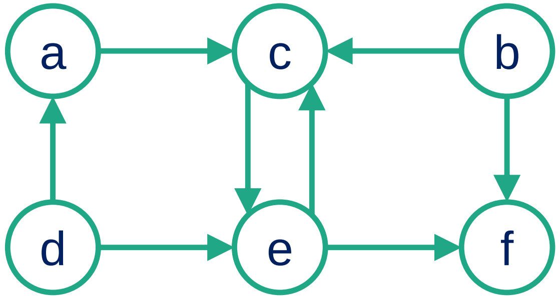
- Dado un grafo *G = (V, E)*, la **estructura de adyacencia** *A* es conjunto de *n* listas *A(v)*, una para cada vértice *v* que pertenece a *V*
- Cada lista *A(v)* es llamada **lista de adyacencia** del vértice *v* y contiene los vértices *w* adyacentes a *v* en *G*



G = (V, E)

$$A = \begin{array}{l} \boldsymbol{a} \rightarrow c \rightarrow d \\ \boldsymbol{b} \rightarrow c \rightarrow f \\ \boldsymbol{c} \rightarrow a \rightarrow b \rightarrow e \\ \boldsymbol{d} \rightarrow a \rightarrow e \\ \boldsymbol{e} \rightarrow c \rightarrow d \rightarrow f \\ \boldsymbol{f} \rightarrow b \rightarrow e \end{array}$$

¿Y para un grafo dirigido?



G = (V, E)

M =

	a	*b*	*c*	*d*	*e*	*f*
a	0	0	**1**	0	0	0
b	0	0	**1**	0	0	**1**
c	0	0	0	0	**1**	0
d	**1**	0	0	0	**1**	0
e	0	0	**1**	0	0	**1**
f	0	0	0	0	0	0

A =

a → *c*
b → *c* → *f*
c → *e*
d → *a* → *e*
e → *c* → *f*
f

Estructura de Adyacencia

- Representación más elaborada

- Propiedades:
  - Almacenamiento: O(|V| + |E|)
  - Prueba de si la arista *(i,j)* está en el grafo: $O(d_i)$, donde d_i es el grado del vértice *i*

Comparación de Representaciones

Característica	Matriz de Adyacencia	Lista de Adyacencia
Almacenamiento	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$
Insertar vértice	$O(\|V\|^2)$	O(1)
Insertar arista	O(1)	O(1)
Eliminar vértice	$O(\|V\|^2)$	$O(\|E\|)$
Eliminar arista	O(1)	$O(\|E\|)$
Consultar si existe arista *(u,v)*	O(1)	$O(\|V\|)$
Observación	Lento para insertar o eliminar vértices porque la matriz debe ser redimensionada o copiada	Cuando se eliminan aristas o vértices, se necesita encontrar todos los vértices o aristas

¿Cuándo usar cada representación?

- Si un grafo es **esparso**, la representación por listas de adyacencia podría usar menos espacio que su correspondiente matriz de adyacencia; la situación es exactamente opuesta para grafos **densos**

- En general, cuál de las dos representaciones es más conveniente depende de la naturaleza del problema, el algoritmo usado para resolverlo y, posiblemente, el tipo de grafo (esparso o denso)

Recorrido de Grafos

Recorriendo un Grafo

- Recorrer un grafo es un problema fundamental
- Se debe tener una forma sistemática de visitar las aristas y los vértices
- El algoritmo debe ser lo suficientemente flexible para adecuarse a una diversidad de grafos
  - **Eficacia:** No debe haber repeticiones (innecesarias) de visitas a un vértice y/o arista
  - **Correctitud**: Todos los vértices y/o aristas deben ser visitados

PUCP

Estrategia General

- Marcar los vértices como
  - No visitados
  - Visitados
  - Procesados
  - …
- Si se mantiene una lista de vértices *procesados* existen dos posibilidades de implementación:
  - Cola
  - Pila

BFS: Breadth-First Search

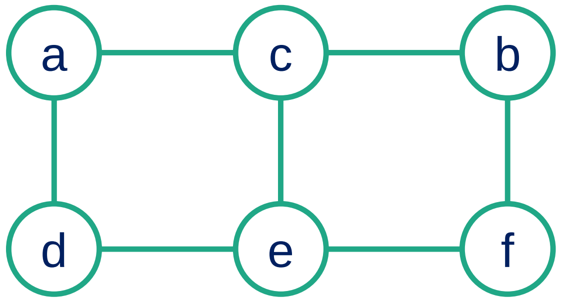
- Búsqueda en amplitud (o por niveles)
- Estructuras de datos y variables:
  - **Por Visitar:** Contiene la cola de vértices del grafo que van siendo procesados y aún faltan visitar
  - **Visitados:** Contiene toda la lista de nodos que han sido visitados a lo largo del algoritmo
  - **P:** Vértice del grafo que está siendo procesado en cada paso (iteración)
  - **Hijos_P**: Lista de nodos adyacentes a P
  - **Vértice Inicial y Vértice Final**
  - **Sentido** de lectura (horario/antihorario)

BFS: Breadth-First Search

```
Inicio BFS (Grafo, PorVisitar, VerticeIni, VerticeFin, Sentido)
    Cola_Inicializar(PorVisitar)
    Lista_Inicializar(Visitados)
    Cola_Enqueue(VerticeIni, PorVisitar)
    Mientras Cola_EsVacia(PorVisitar) = FALSO Y  P <>
VerticeFin hacer
        P = Cola_Dequeue (PorVisitar)
        Procesar P
        Lista_Insertar (Visitados, P)
        Hijos_P = Grafo_Adyacentes (Grafo, P, Sentido)
        Para cada u ∊ Hijos_P / u ∉ Visitados hacer
            Cola_Enqueue(PorVisitar, u)
    Fin Mientras
Fin BFS
```

Ejemplo BFS

- Recorra el siguiente grafo con BFS comenzando por el vértice a y buscando el vértice f:



DFS: Depth-First Search

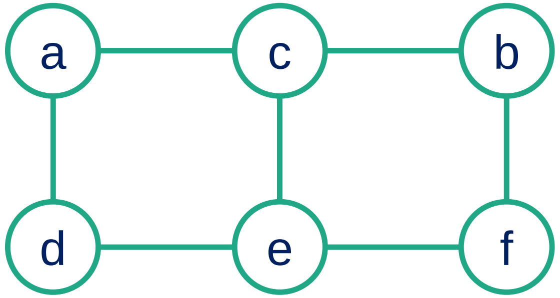
- Búsqueda en profundidad
- Estructuras de datos y variables:
  - **Por Visitar:** Contiene la pila de vértices del grafo que van siendo procesados y aún faltan visitar
  - **Visitados:** Contiene toda la lista de nodos que han sido visitados a lo largo del algoritmo
  - **P:** Vértice del grafo que está siendo procesado en cada paso (iteración)
  - **Hijos_P**: Lista de nodos adyacentes a P
  - **Vértice Inicial y Vértice Final**
  - **Sentido** de lectura (horario/antihorario)

DFS: Depth-First Search

```
Inicio DFS (Grafo, PorVisitar, VerticeIni, VerticeFin, Sentido)
    Pila_Inicializar(PorVisitar)
    Lista_Inicializar(Visitados)
    Pila_Push(VerticeIni, PorVisitar)
    Mientras Pila_EsVacia(PorVisitar) = FALSO Y  P <>
VerticeFin hacer
        P = Pila_Pop (PorVisitar)
        Procesar P
        Lista_Insertar (Visitados, P)
        Hijos_P = Grafo_Adyacentes (Grafo, P, Sentido)
        Para cada u ∈ Hijos_P / u ∉ Visitados hacer
            Pila_Push(PorVisitar, u)
    Fin Mientras
Fin DFS
```

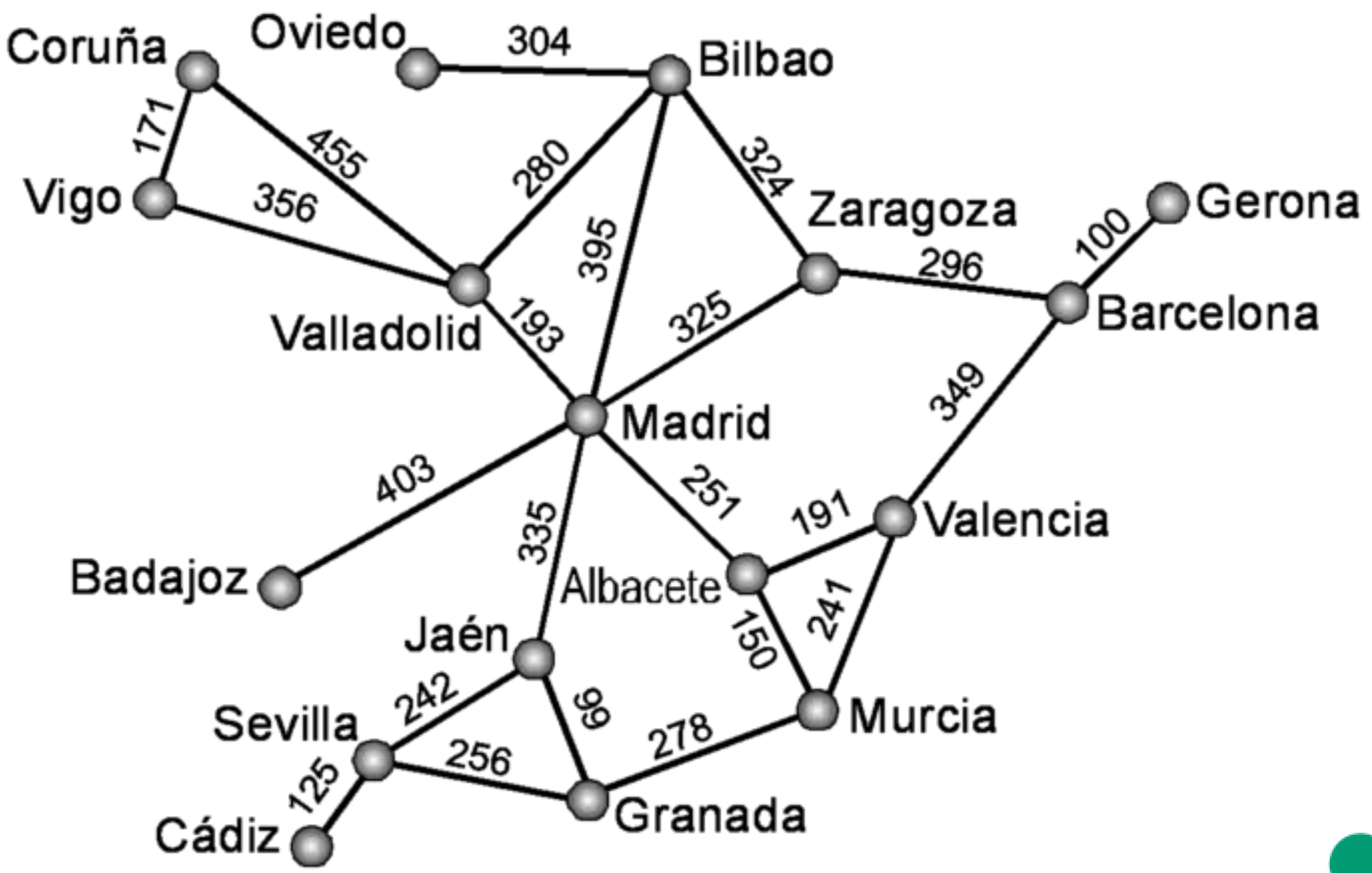
Ejemplo DFS

- Recorra el siguiente grafo con DFS comenzando por el vértice a y buscando el vértice f:



Caminos Más Cortos

Aplicación en Mapas



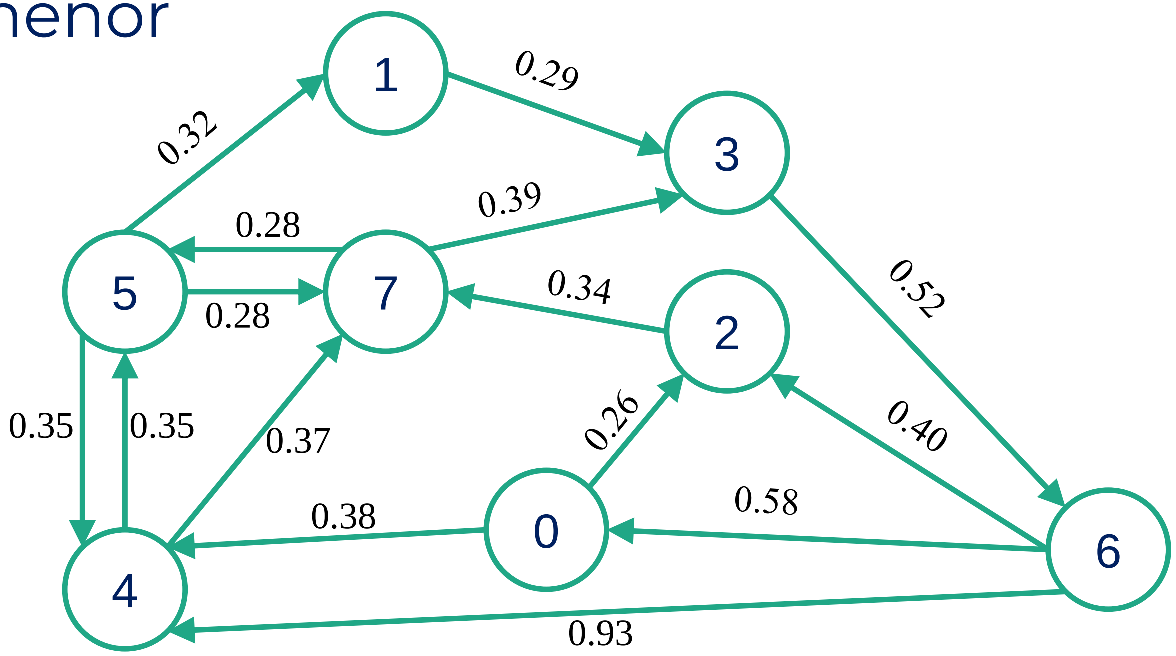
Aplicaciones Típicas

Aplicación	Vértice	Arista
Mapa	Intersección	Camino
Red	*Router*	Conexión
Programación de horarios	Tarea	Restricción de precedencia

SEDGEWICK, R. y WAYNE, K. **Algorithms**. 4ta edición. USA: Pearson, 2011.
ISBN-13 978-0-321-57351-3

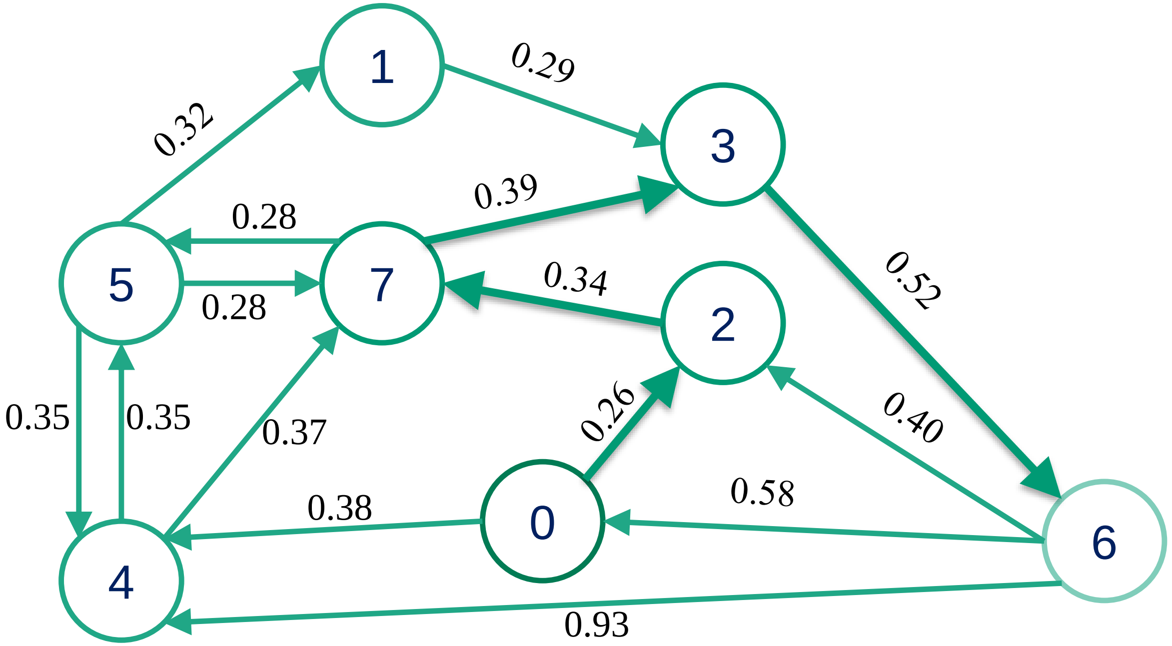
Definición

- El **camino más corto** desde el vértice *s* hasta el vértice *t* en un digrafo ponderado es un camino dirigido desde *s* hasta *t* con la propiedad de que no otro camino tiene un peso menor



SEDGEWICK, R. y WAYNE, K. **Algorithms**. 4ta edición. USA: Pearson, 2011.
ISBN-13 978-0-321-57351-3

¿Cuál es el camino más corto de 0 a 6?



Camino Más Corto de 0 a 6:

0 → 2 0.26
2 → 7 0.34
7 → 3 0.39
3 → 6 0.52

SEDGEWICK, R. y WAYNE, K. **Algorithms**. 4ta edición. USA: Pearson, 2011.
ISBN-13 978-0-321-57351-3

Variantes

- **Desde un único origen**
  - Encontrar los caminos más cortos desde un vértice origen *s* a todos los demás vértices del grafo
- **Con un único destino**
  - Encontrar los caminos más cortos desde todos los vértices del grafo a un único vértice destino
- **Entre un único par de vértices**
  - Encontrar el camino más corto entre los vértices *u* y *v*
- **Entre todos los pares de vértices**
  - Encontrar los caminos más cortos entre cada par de vértices *u* y *v* del grafo

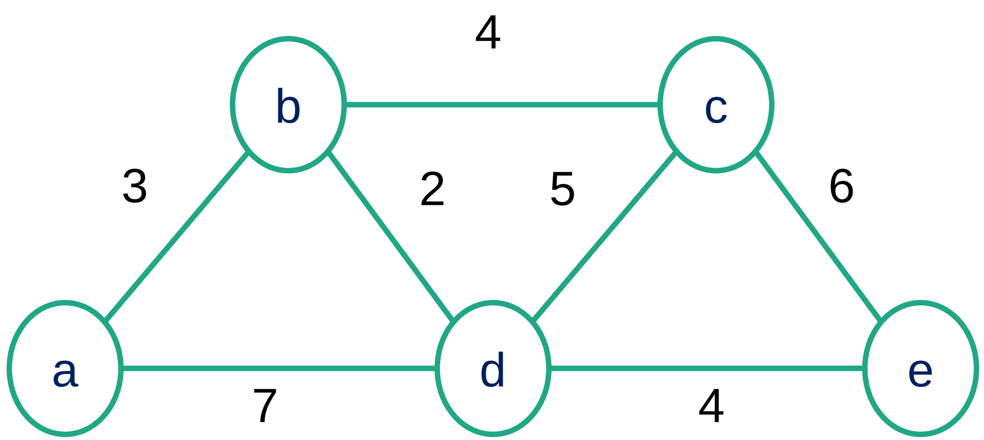
Algoritmo de Dijkstra

- Aplicable en grafos dirigidos y no dirigidos con pesos no negativos

- Idea básica:
  - Se inicia desde un vértice *s*
  - Se posee un conjunto de vértices para cuales conocemos sus caminos más cortos desde *s* (inicialmente vacío)
  - En cada iteración, se añade a este conjunto aquel vértice *u* adyacente al vértice actual *u** que esté más cerca de *s*
  - Se debe considerar la distancia desde *s* hasta el vértice actual *u** y el peso de la arista *(u, u*)*

Ejemplo 1

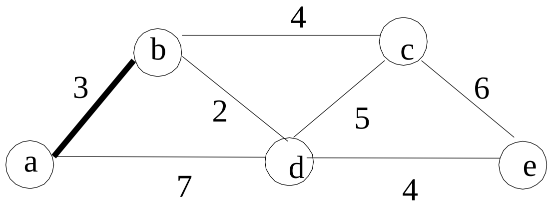
- Hallar los caminos más cortos desde el vértice ***a***

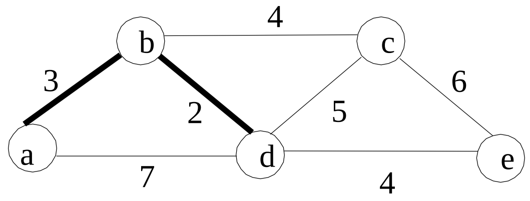


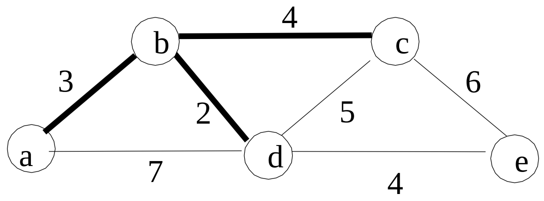
LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición.
USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

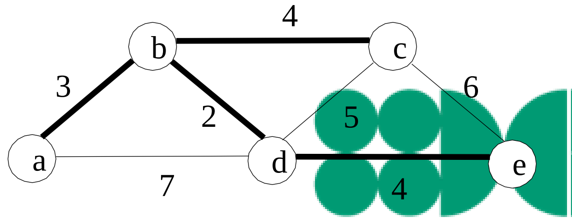
Ejemplo 1: Solución

Visitados	Por Visitar
a(-,0)	**b(a,3)** c(-,∞) d(a,7) e(-,∞)
b(a,3)	c(b,3+4) **d(b,3+2)** e(-,∞)
d(b,5)	**c(b,7)** e(d,5+4)
c(b,7)	**e(d,9)**
e(d,9)	





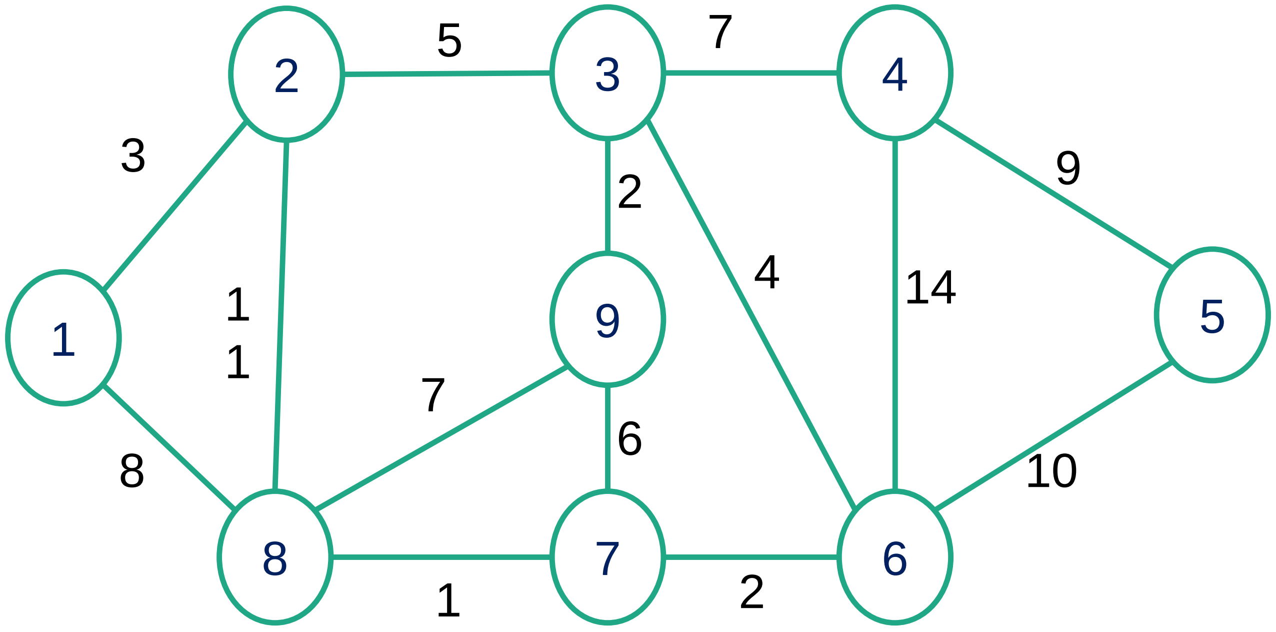




LEVITIN, A. **Introduction to The Design and Analysis of Algorithms**. 3ra edición. USA: Pearson, 2012. ISBN-13 978-0-13-231681-1

Ejemplo 2

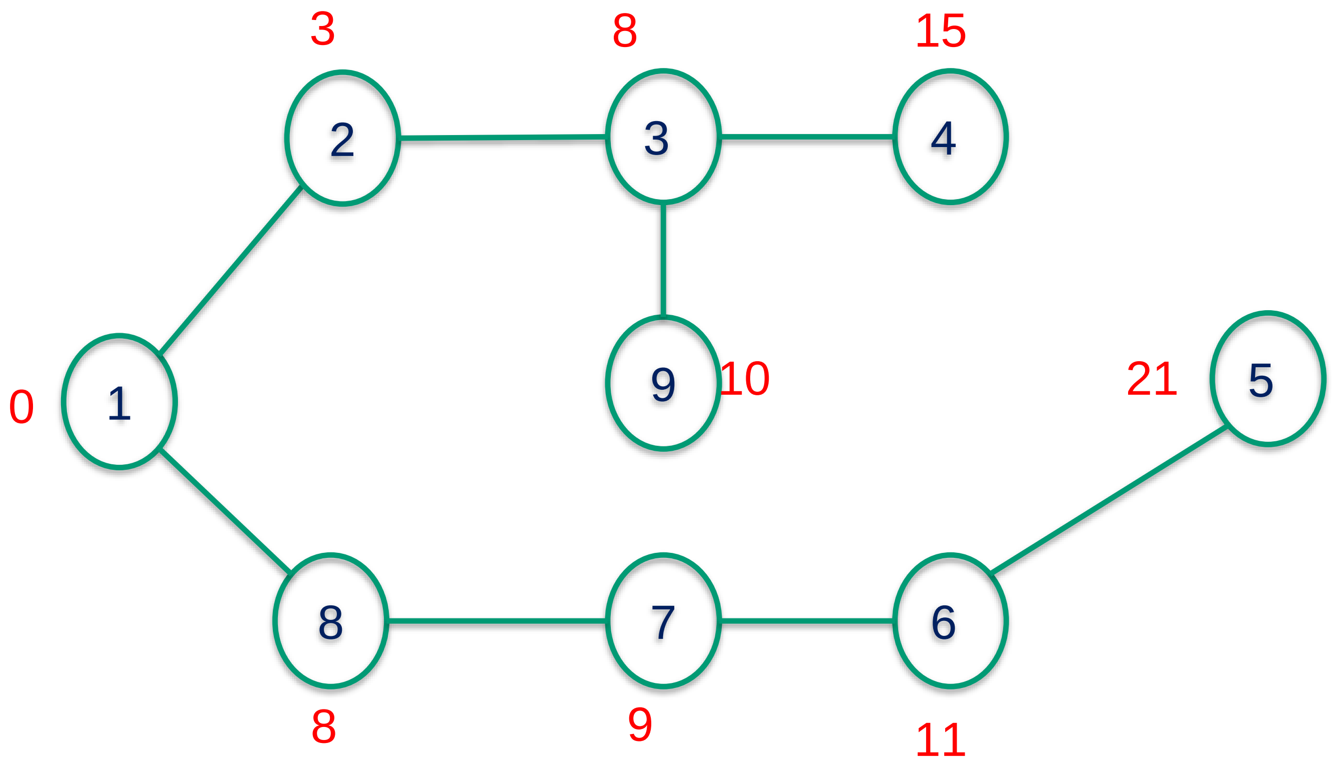
- Hallar los caminos más cortos desde el vértice **1**



Extraído de: http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

Ejemplo 2: Solución

- Hallar los caminos más cortos desde el vértice **1**



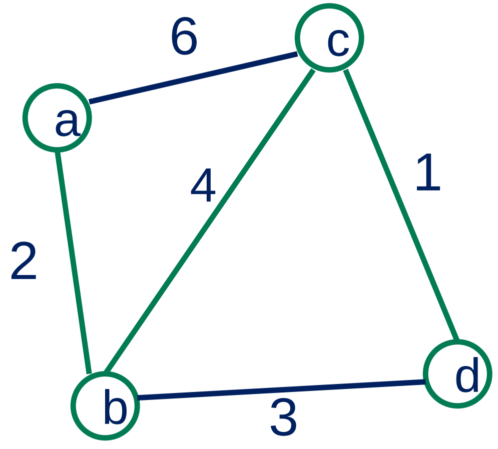
Extraído de: http://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

Eficiencia del Algoritmo de Dijkstra

- La implementación más simple:
  - El conjunto de vértices **NoVisitados** es una lista enlazada o un arreglo
  - Extraer el mínimo de **NoVisitados** es una búsqueda lineal
  - $O(|E|+|V|^2) = O(|V|^2)$
- Para grafos esparsos:
  - Grafo como lista de adyacencia
  - El conjunto de vértices **NoVisitados** es implementado usando:
    - ABBs auto-balanceables: 2-3, AVL, rojo-negro, etc.
    - Montículos (*heaps*): binarios, Fibonacci, etc.
  - $O(|E|+|V|\log|V|)$

Árbol de Expansión Mínimo

- El **árbol de expansión** de un grafo conexo *G* es un subgrafo conexo acíclico de *G* tal que incluye todo los vértices *G*
- El **árbol de expansión mínimo** de un grafo conexo ponderado *G* es un árbol de expansión de *G* de peso total mínimo



El algoritmo de Dijkstra genera un árbol de expansión, pero no es mínimo