Grafos

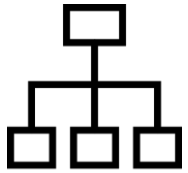
Jaime A. Riascos

Adaptado de:

https://ocw.mit.edu/courses/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/69b5b28067ecf1769a6143453d77eba1_MIT6_0002F16_lec3.pdf

https://github.com/mauriciotoro/ST0245-Eafit/tree/master

Estructura de Datos



Colección de objetos del mismo tipo con el menor consumo de recursos para un algoritmo

Ejemplos de recursos

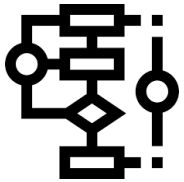
tiempo

memoria

Relación entre EDA y Algoritmo

Dependiendo de la estructura de datos

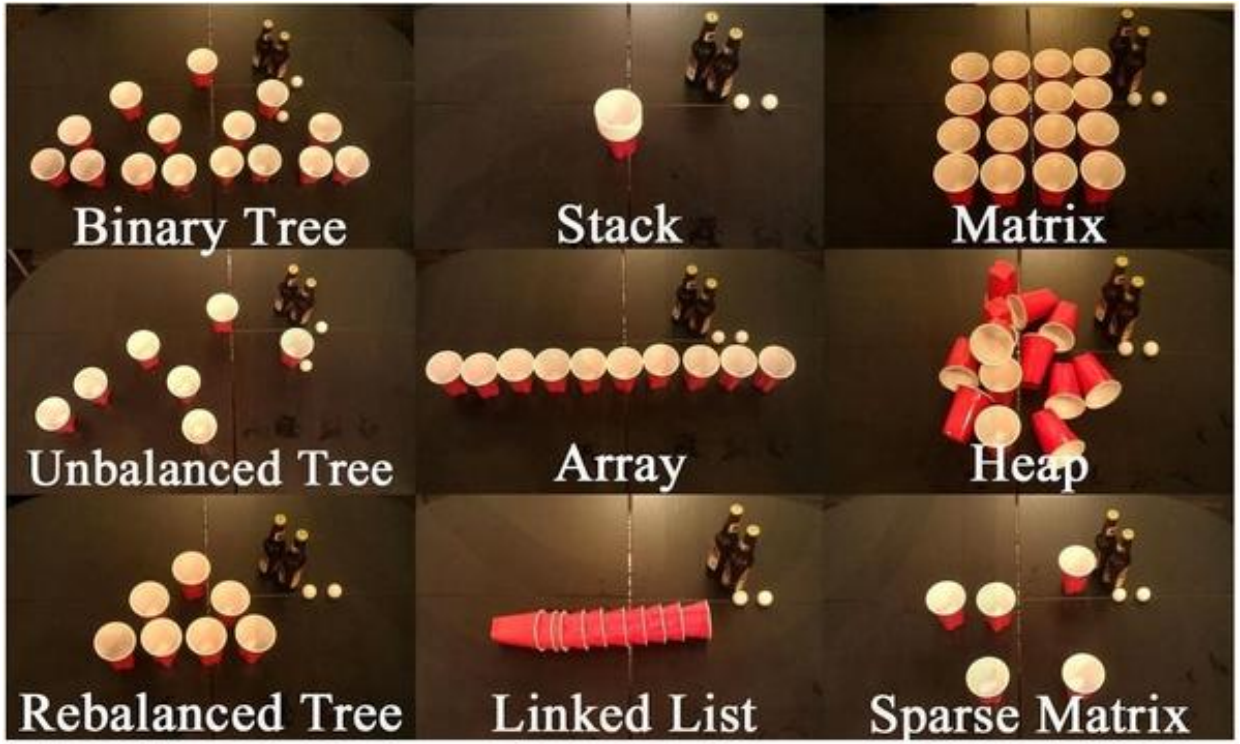
cambia la complejidad del



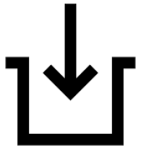
algoritmo

tiempo

memoria



Operaciones de una Estructura de Datos



Acceso

Búsqueda

Inserción

Borrado

Barbara Liskov
Inventó las estructuras de datos

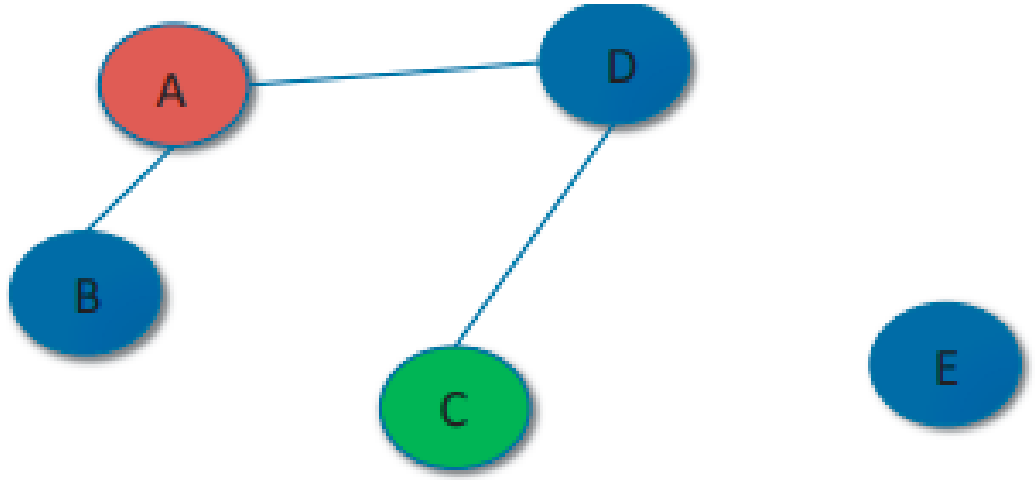
Definición de Grafo

Un **grafo es una tupla** de dos cosas. Primero, **una "bolsa" o conjunto de nodos** (llamados **vértices**) y, segundo, una "**bolsa" o conjunto de conexiones entre parejas de vértices** llamadas **aristas**.

Un **grafo** se usa para representar problemas en los que hay **objetos** y **conexiones entre los objetos.**

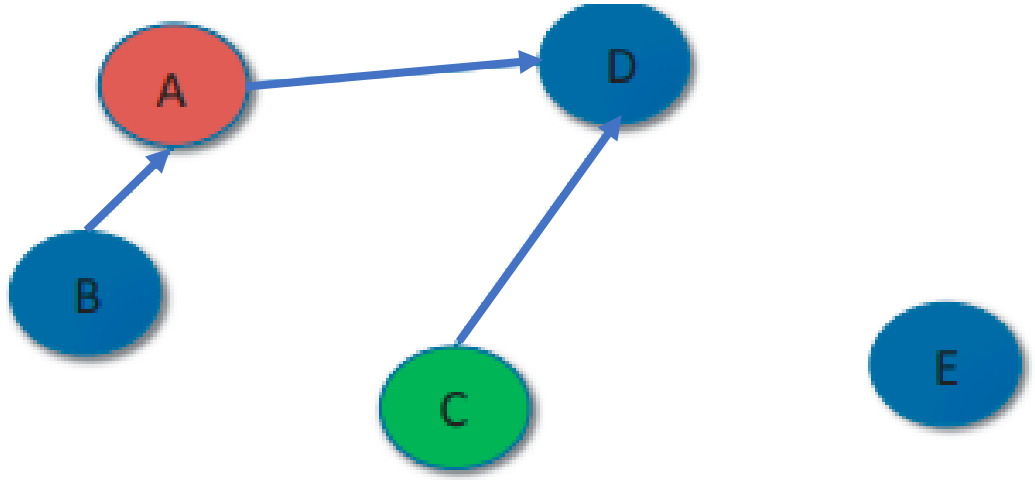
Grafos

- Conjunto de nodos (vértices) con algunas propiedades asociadas a ellos.
- Conjunto de arcos que consisten en un unir un par de nodos
  - Grafos direccionados (importa la secuencia o dirección), con definición de nodo padre y nodo hijo
  - Grafos no direccionados.
  - Grafos con pesos o sin pesos



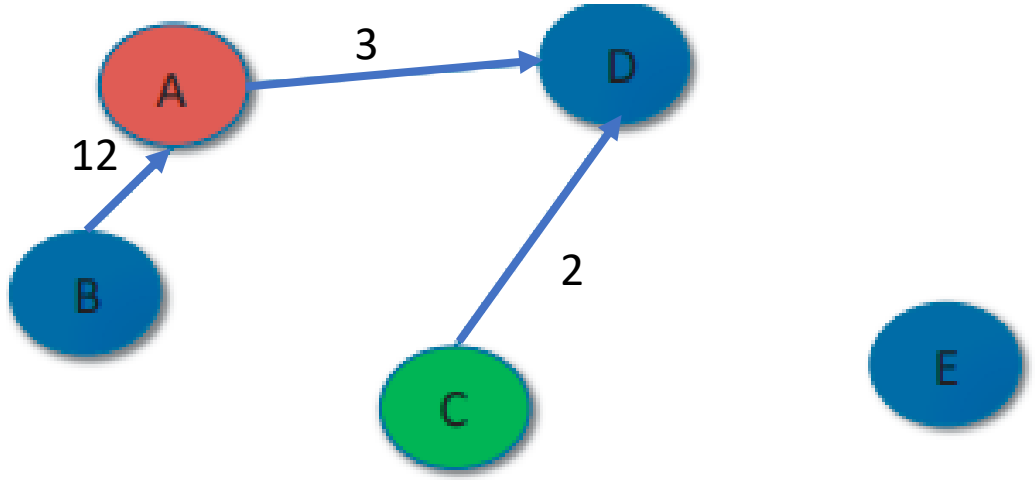
Grafos

- Conjunto de nodos (vértices) con algunas propiedades asociadas a ellos.
- Conjunto de arcos que consisten en un unir un par de nodos
  - Grafos direccionados (importa la secuencia o dirección), con definición de nodo padre y nodo hijo
  - Grafos no direccionados.
  - Grafos con pesos o sin pesos



Grafos

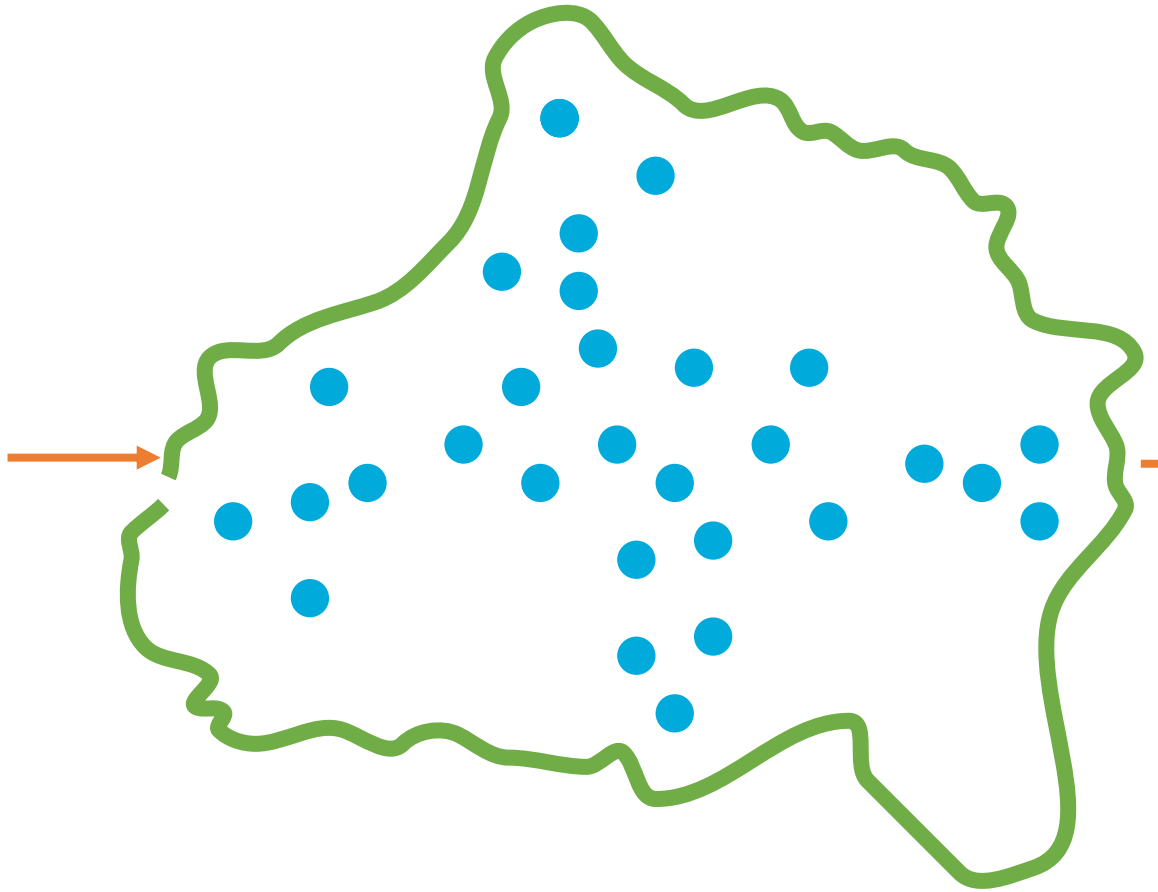
- Conjunto de nodos (vértices) con algunas propiedades asociadas a ellos.
- Conjunto de arcos que consisten en un unir un par de nodos
  - Grafos direccionados (importa la secuencia o dirección), con definición de nodo padre y nodo hijo
  - Grafos no direccionados.
  - Grafos con pesos o sin pesos



Grafos en videojuegos

Vértices

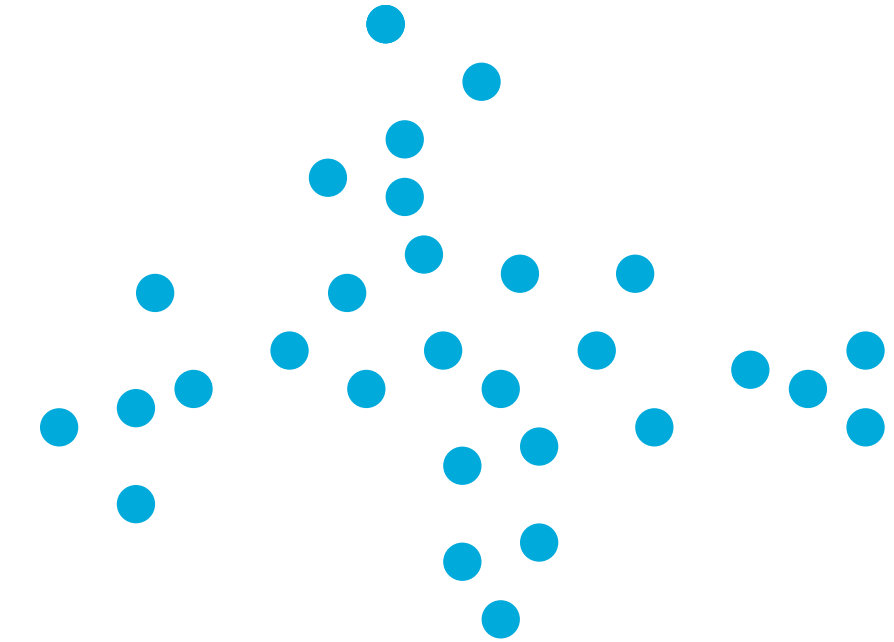
Una “bolsa” de números del 0 al n-1 sin repetir



Vértices

Vértices

Una “bolsa” de números del 0 al n-1 sin repetir

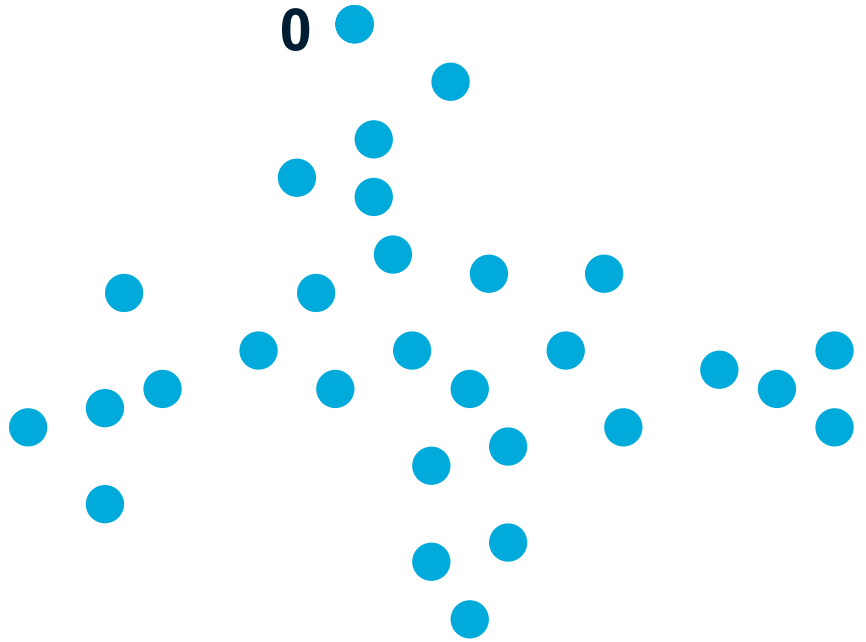


Vértices

$$V \subseteq \mathbb{N} \cup \{0\}$$

Vértices

Una “bolsa” de números del 0 al n-1 sin repetir

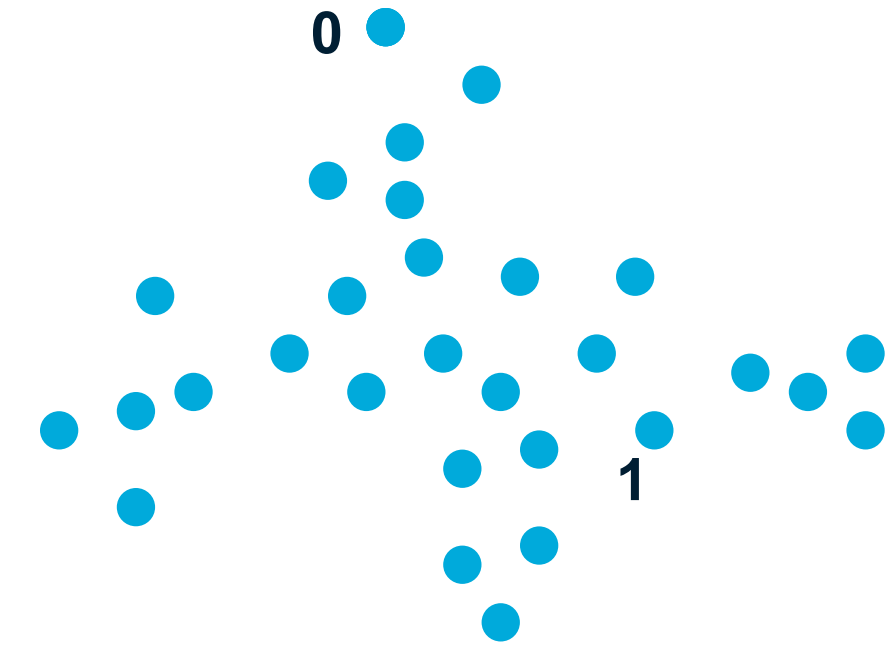


Vértices

$$V \subseteq \mathbb{N} \cup \{0\}$$

Vértices

Una “bolsa” de números del 0 al n-1 sin repetir

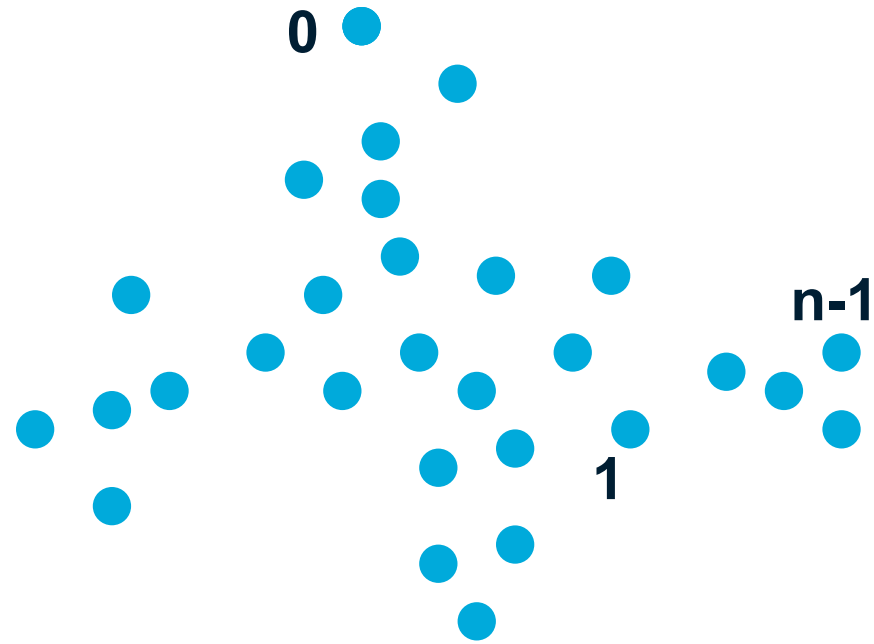


Vértices

$$V \subseteq \mathbb{N} \cup \{0\}$$

Vértices

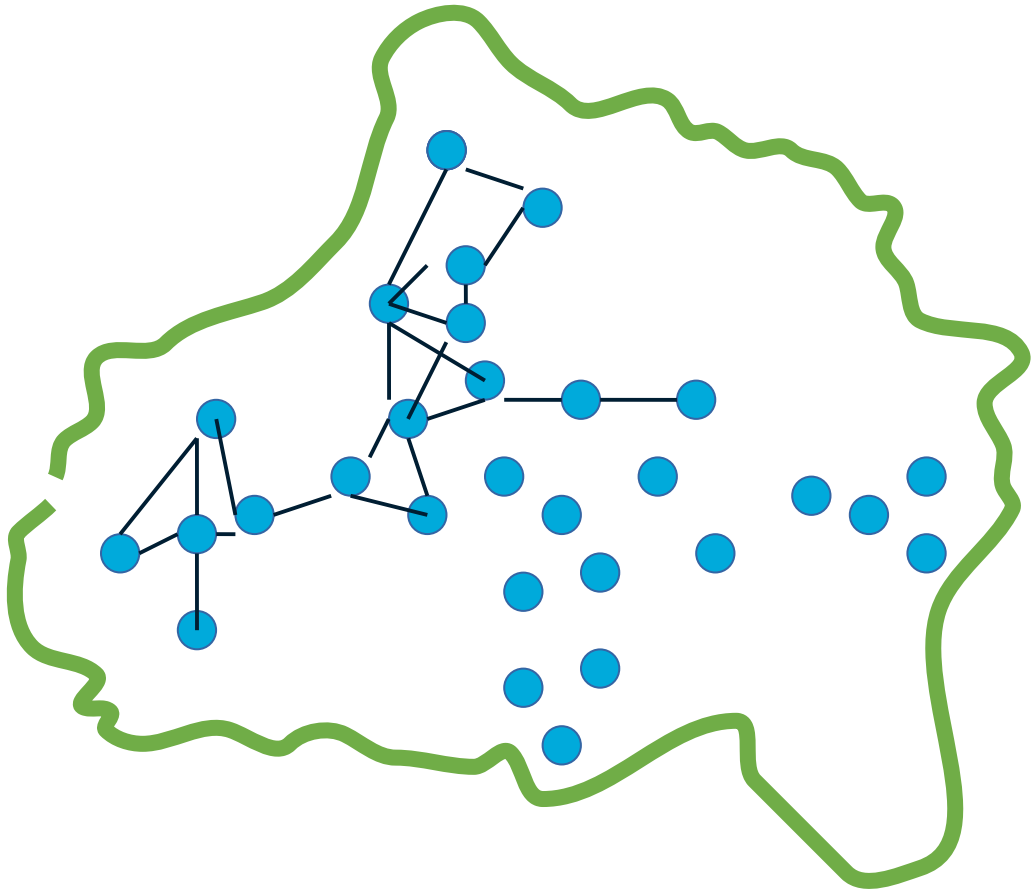
Una “bolsa” de números del 0 al n-1 sin repetir



Vértices

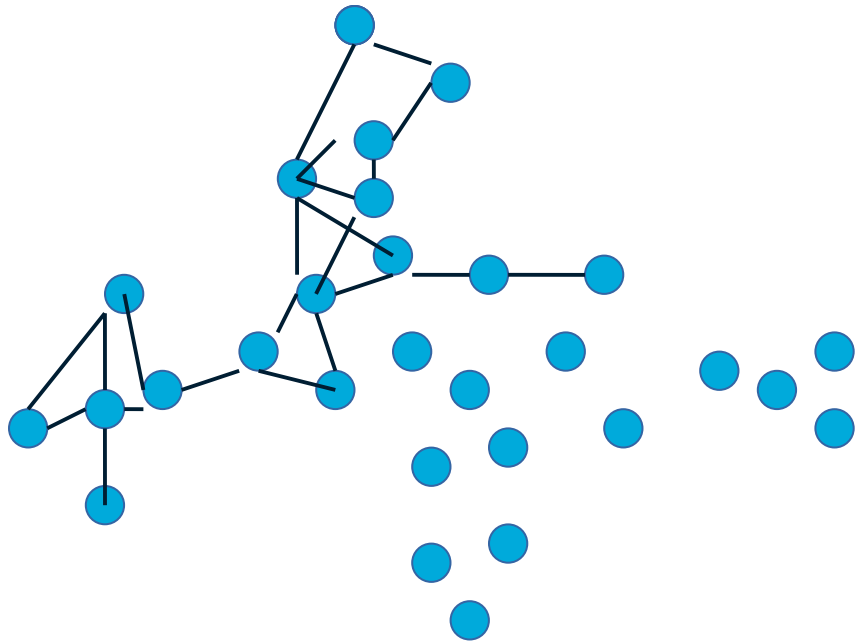
$$V \subseteq \mathbb{N} \cup \{0\}$$

Aristas



Aristas

Una “bolsa” de conexiones entre los vértices sin repetir

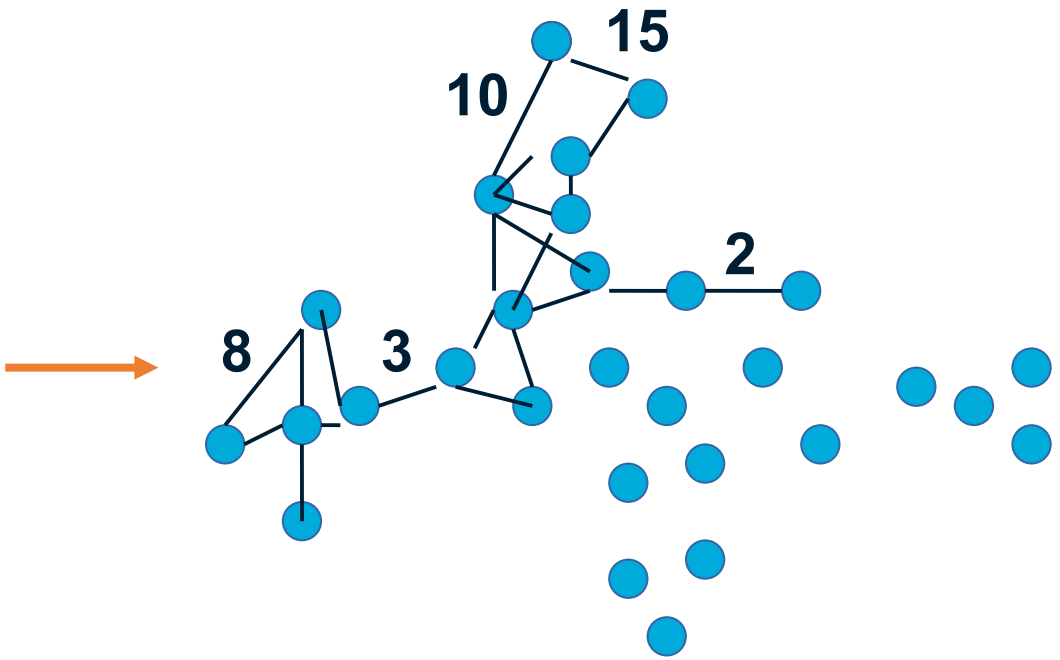


Aristas

$$g = (V, E), E \subseteq V \times V$$

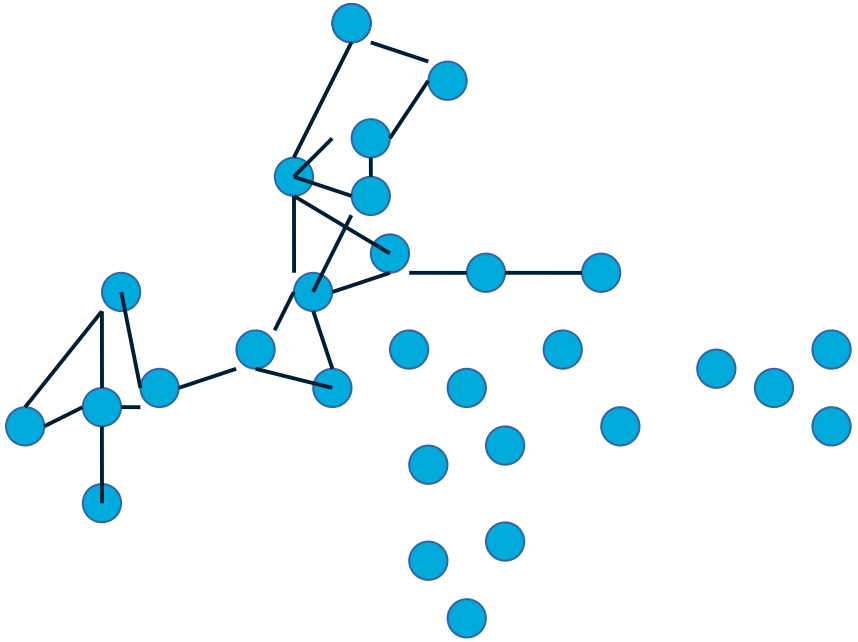
Aristas con pesos

Las aristas pueden tener pesos



Grafo no dirigido

En las conexiones
NO importa el
sentido

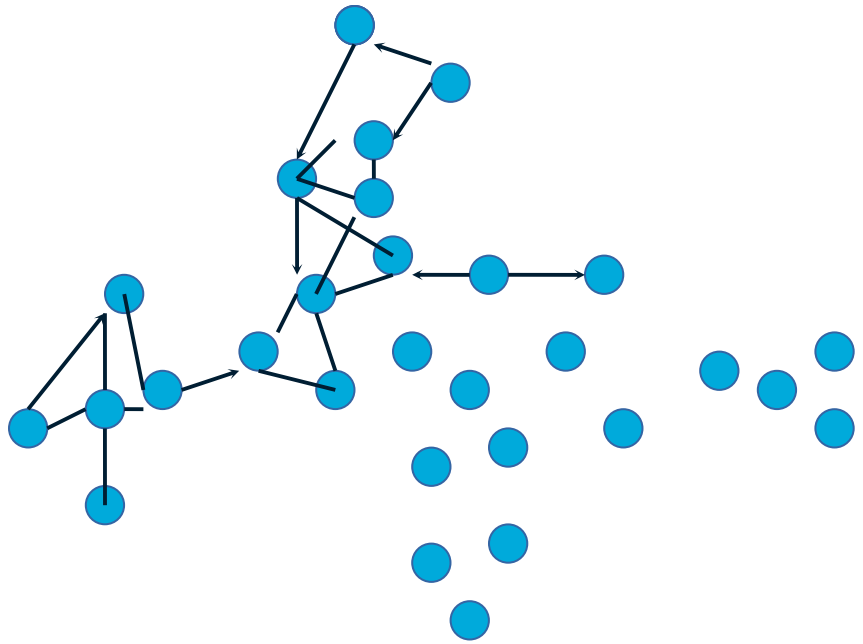


Grafo
NO dirigido

$$\forall a \forall b (a, b) \in E \rightarrow (b, a) \in E$$

Grafo dirigido

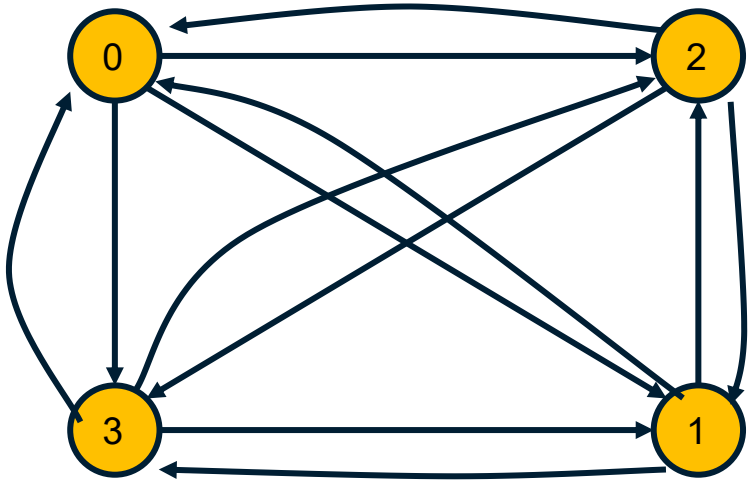
En las conexiones
SÍ importa el
sentido



Grafo
dirigido

Grafo completo

Todos están conectados con todos



Grafo completo

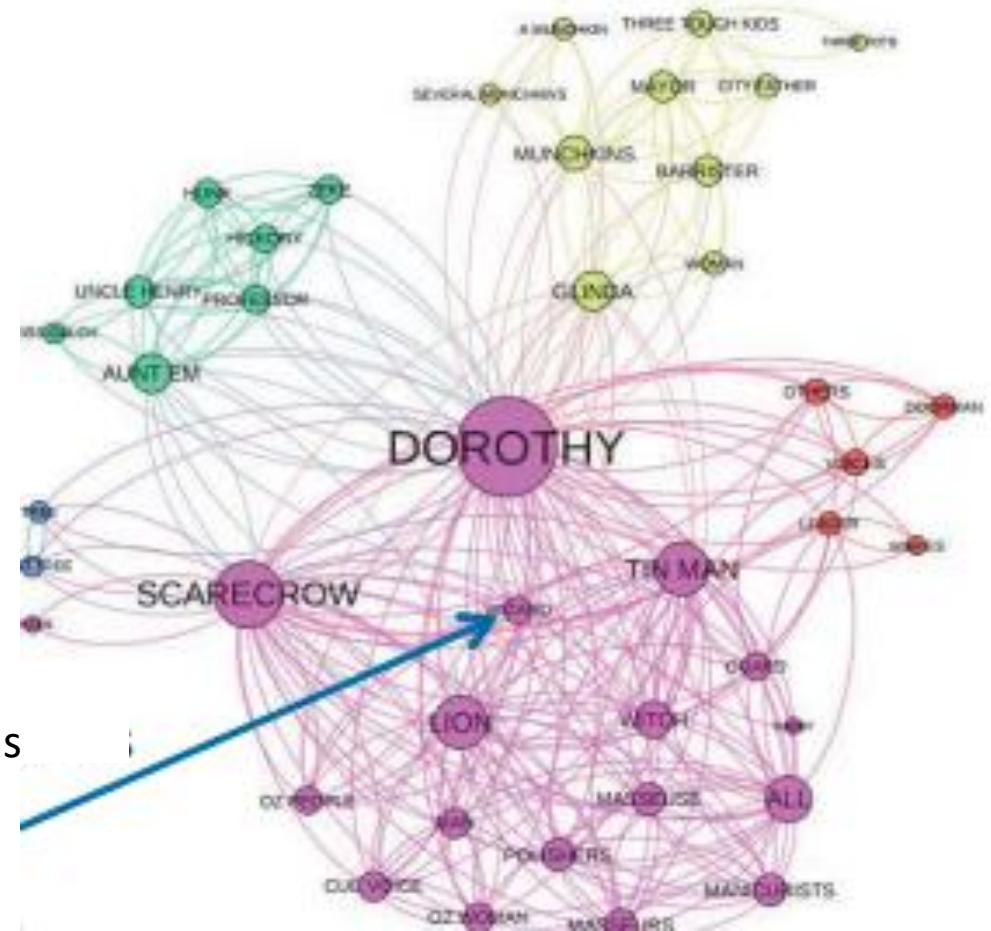
$$E = V \times V, |E| = |V \times V| = |V|^2$$

¿Por qué usamos Grafos?

- Capturar relaciones útiles entre entidades
  - Metro que une Bello con Itagüí
  - Cómo los átomos de una molécula se relacionan entre sí
  - Relaciones ancestrales (familiares)

¿Para qué son útiles los grafos?

- Nuestro mundo está explicado por un sinnúmero de redes basadas en relaciones:
  - Redes de computadora
  - Redes de transporte
  - Redes financieras
  - Redes de agua o alcantarillado
  - Redes políticas
  - Redes criminales
  - Redes sociales



Análisis de las interacciones de los personajes en las escenas (tamaño) e interacciones naturales entre personajes (color) – Novela Mago de Oz

¿Para qué son útiles los grafos?

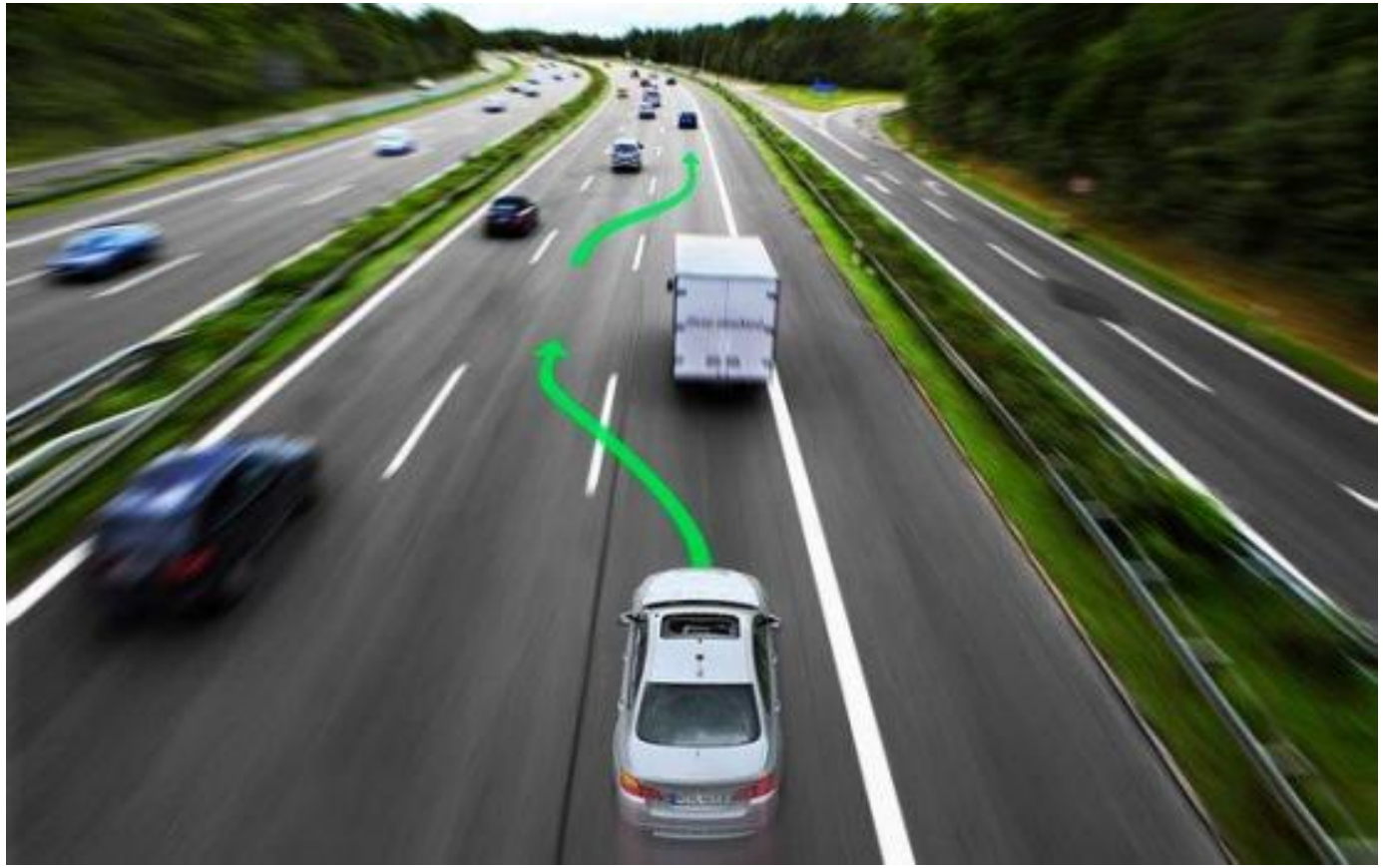
- Pero además de capturar las relaciones que existan entre instancias de un problema, los grafos nos permiten crear inferencias sobre esas estructuras:
  - Encontrar si existe o no un camino o secuencia entre A y B
  - Encontrar el camino más corto entre A y B (shortest path problem)
  - Partir el grafo entre un subconjunto de elementos conectados (graph partition problem)
  - Encontrar la forma más eficiente de dividir un conjunto de elementos conectados (min cut-max/max-flow problem)

¿Para qué son útiles los grafos?

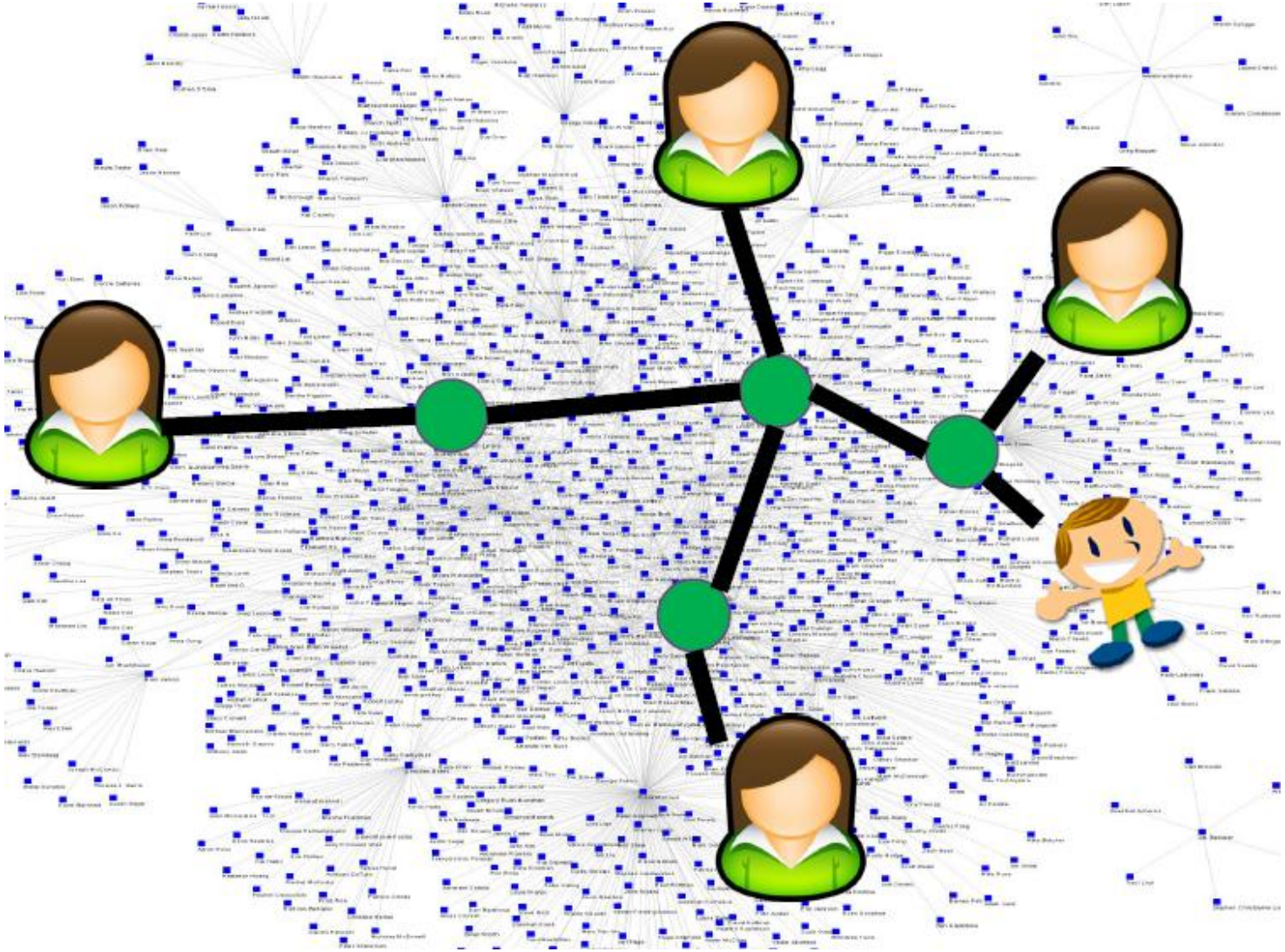
¿Para qué son útiles los grafos?

¿Para qué son útiles los grafos?

¿Para qué son útiles los grafos?



¿Para qué son útiles los grafos?



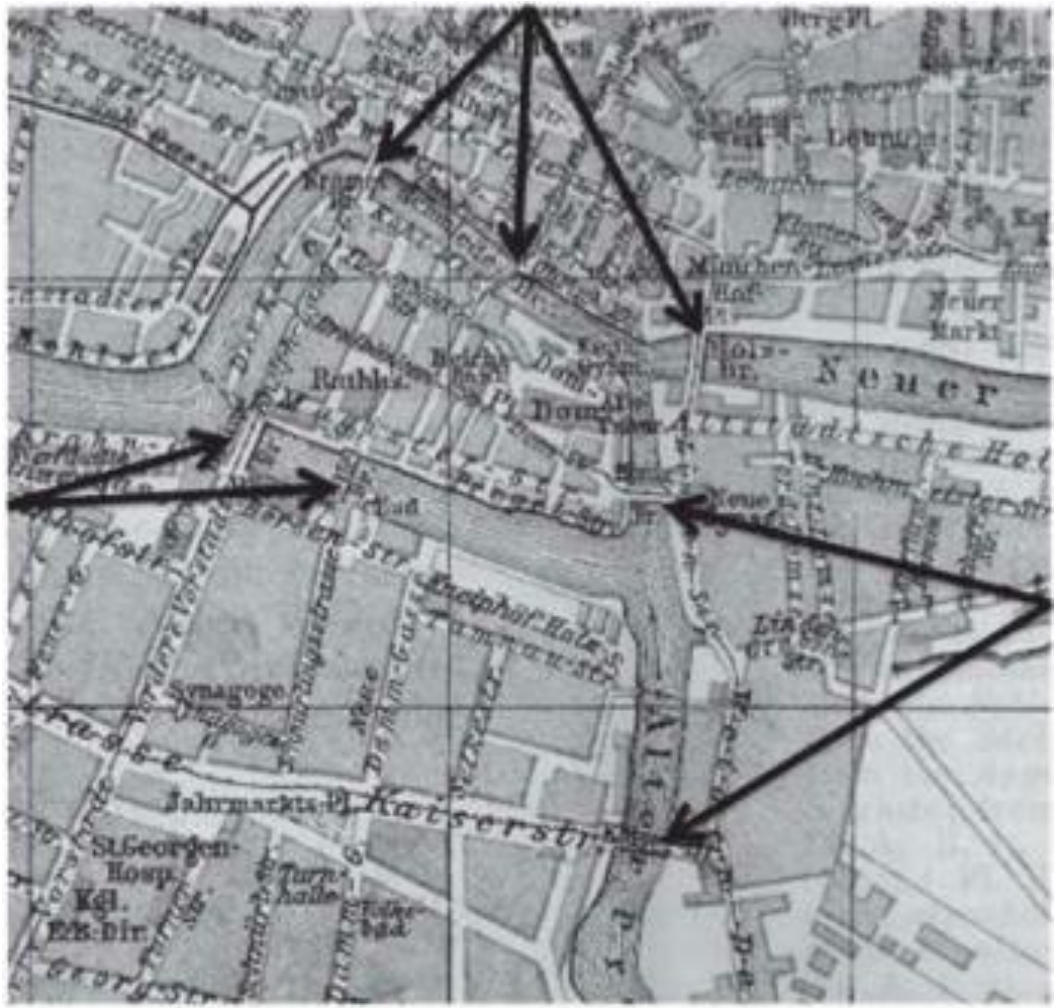
¿Para qué son útiles los grafos?

- Modelando el sistema de transporte usando un digraph
  - Nodos: puntos donde los caminos se unen o terminan
  - Arcos: conexión entre puntos
  - Cada arco tiene un peso donde:
    - Tiempo estimado en ir desde un punto (origen) a otro (destino)
    - Distancia entre origen y destino
    - Velocidad promedio de viaje entre origen y destino
- Resolviendo un problema de optimización
  - Camino más corto entre mi casa y la universidad, considerando los pesos:

¿Desde cuándo se usan los grafos?

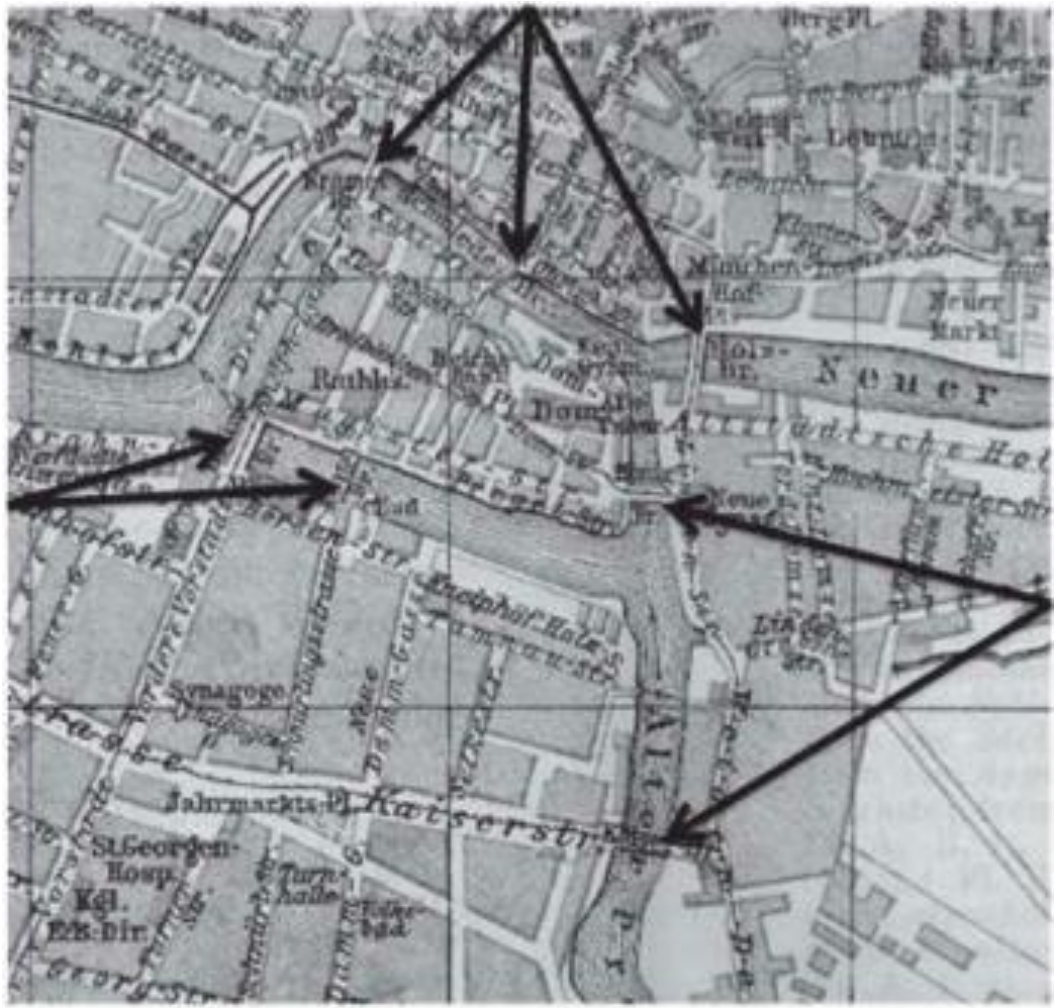
- Puentes de Konigsberg (1735)

- ¿Es posible dar un paseo atravesando una sola vez cada uno de los puentes?

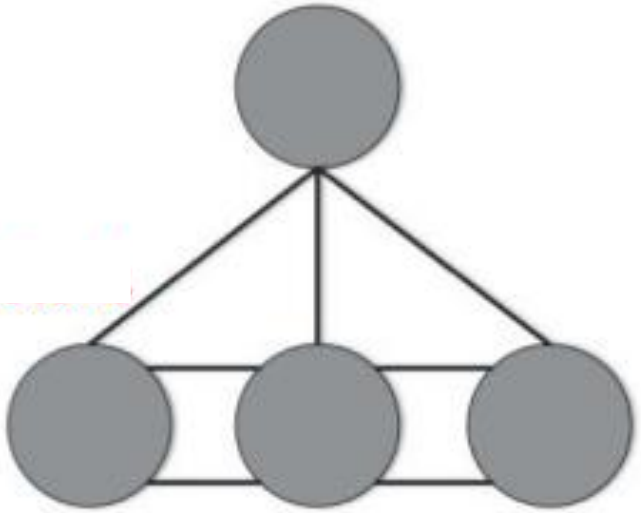


¿Desde cuándo se usan los grafos?

- Puentes de Konigsberg (1735)
- ¿Es posible dar un paseo atravesando una sola vez cada uno de los puentes?



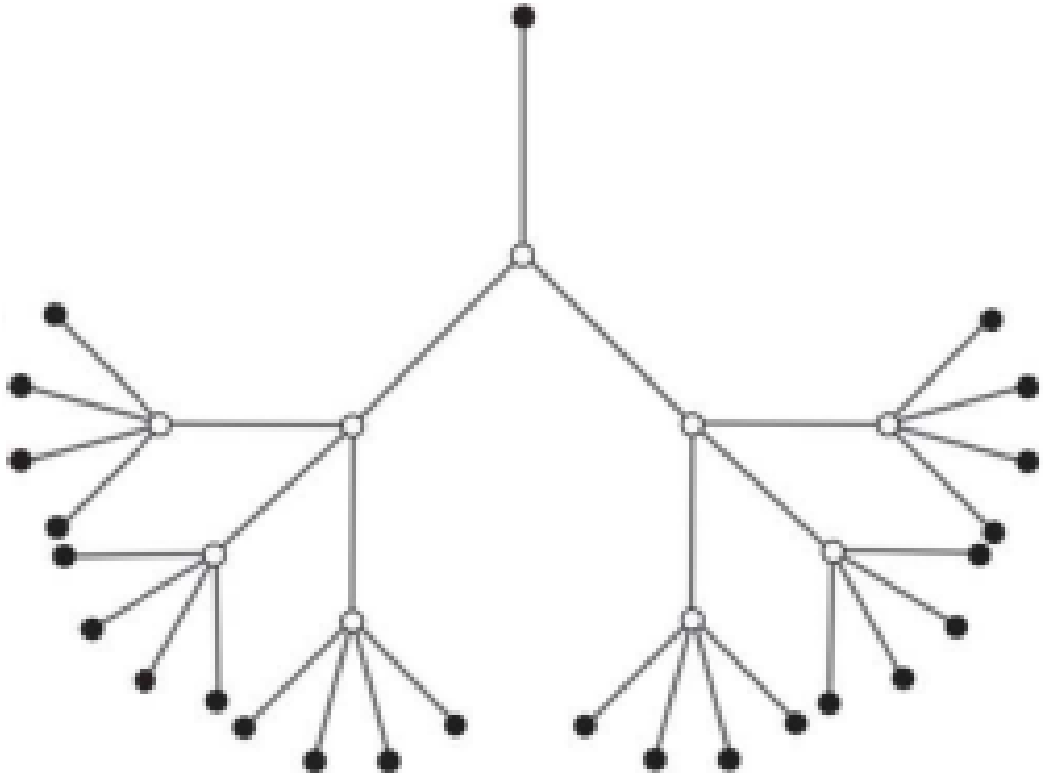
Modelo de Leonhard Euler



¿Son los árboles grafos?

¿Son los árboles grafos?

- Si, un tipo especial de grafo direccionado donde cada par de nodos está conectado por un camino único

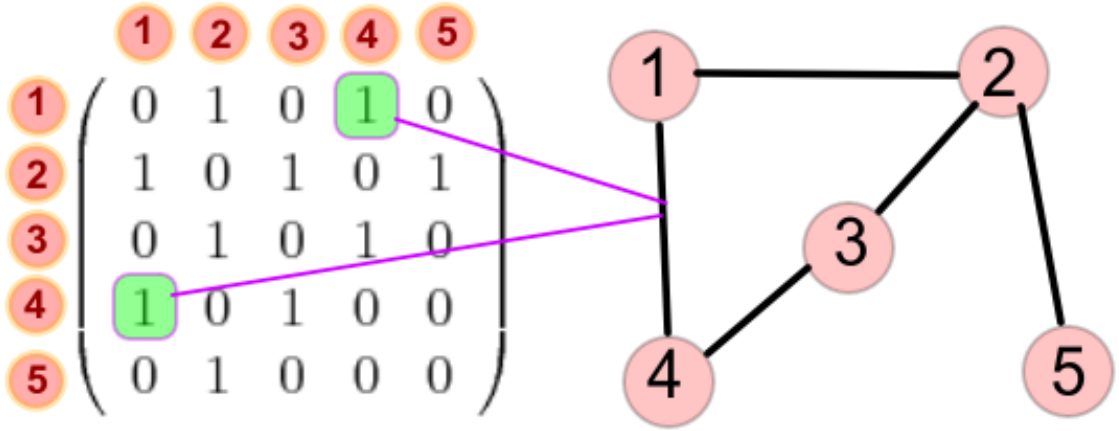


Representación de grafos direccionados (digraphs)

Grafo direccionado = Arcos pasan en una sola dirección

Matriz de adyacencia:

- Filas: nodos origen
- Columnas: nodos destino
- Celda[i,j] = 1 si hay un arco entre el origen y destino
  = 0 si no.
- Matriz asimétrica
- Complejidad de consulta: O(1)



https://matematicasies.com/Matriz-de-adyacencia-de-un-grafo

Graph Representation

Vertex List

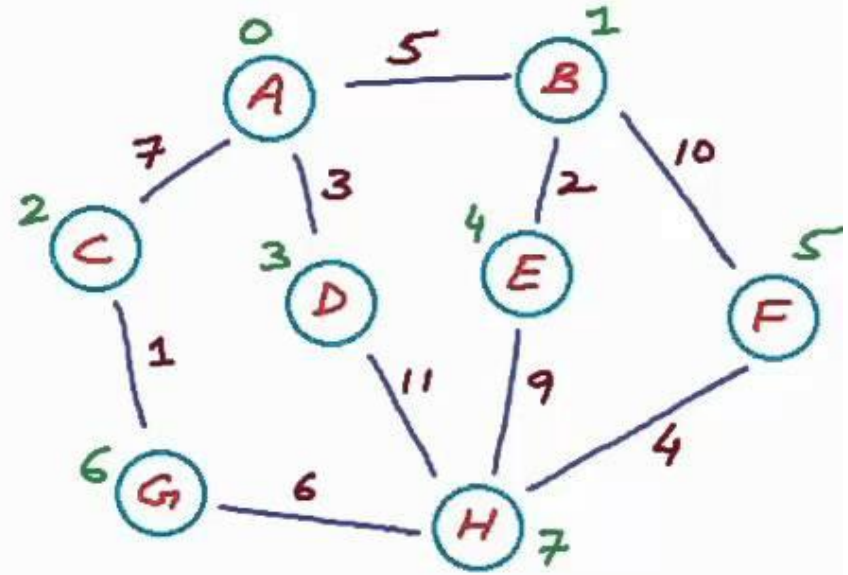
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

Adjacency Matrix

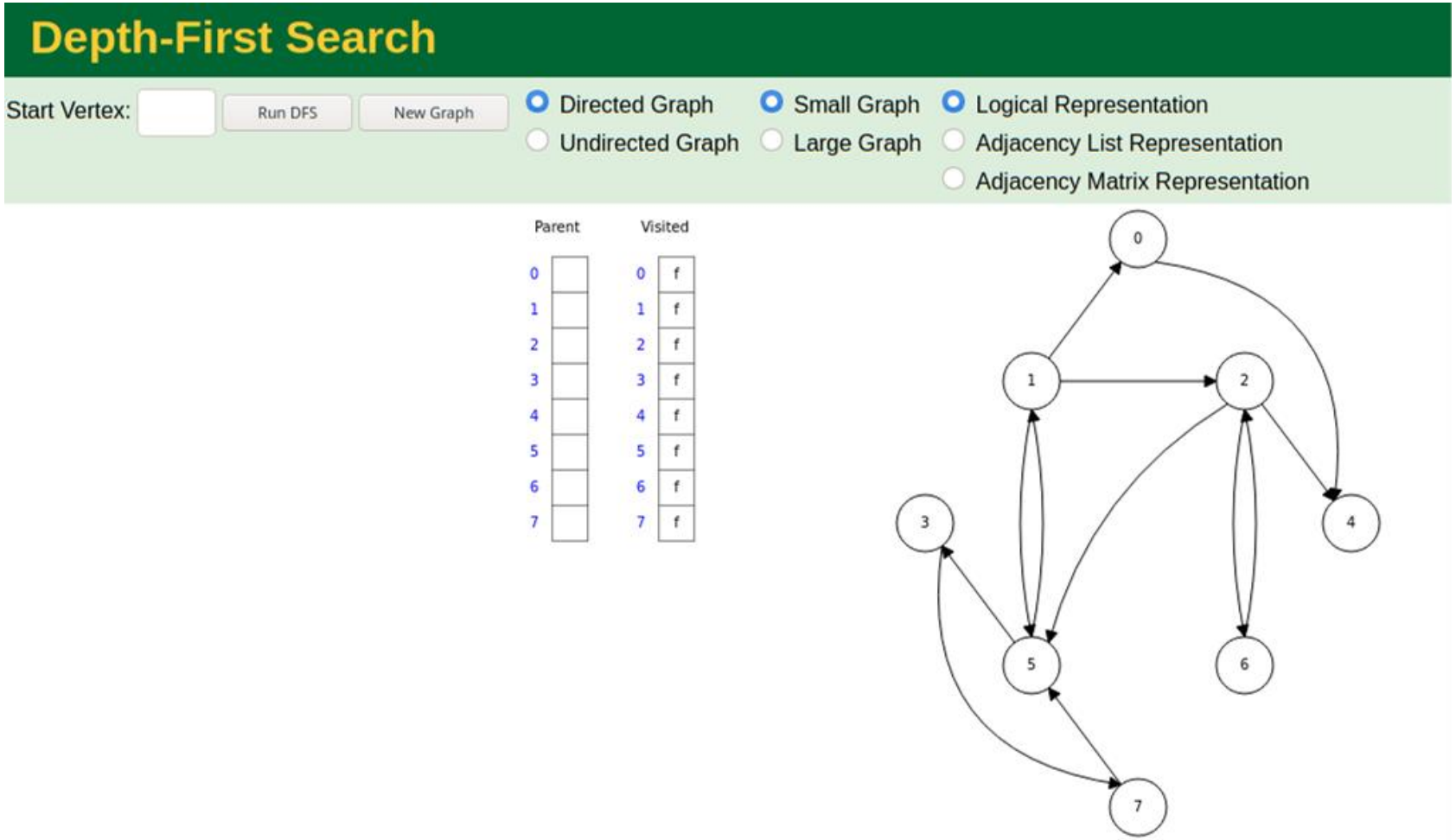
	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

A

$|V| = v$

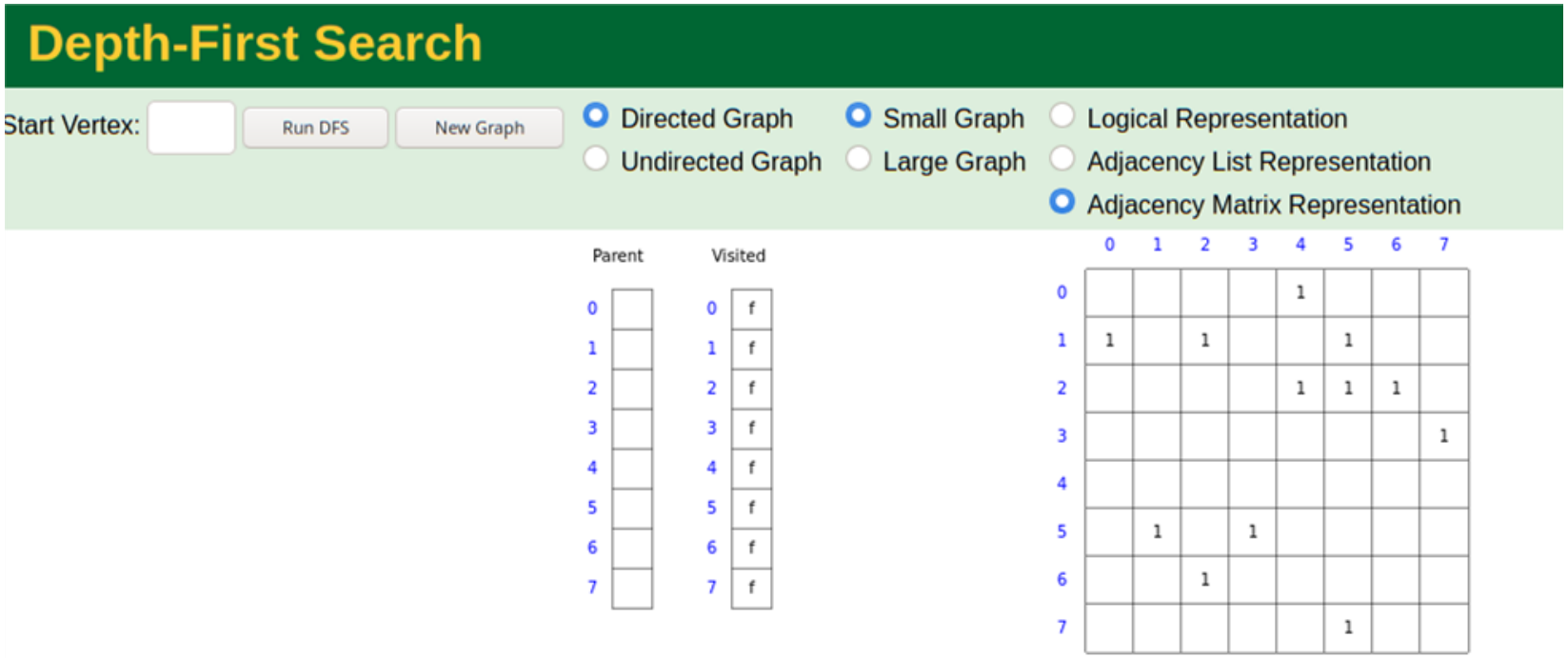


Matrices de adyacencia



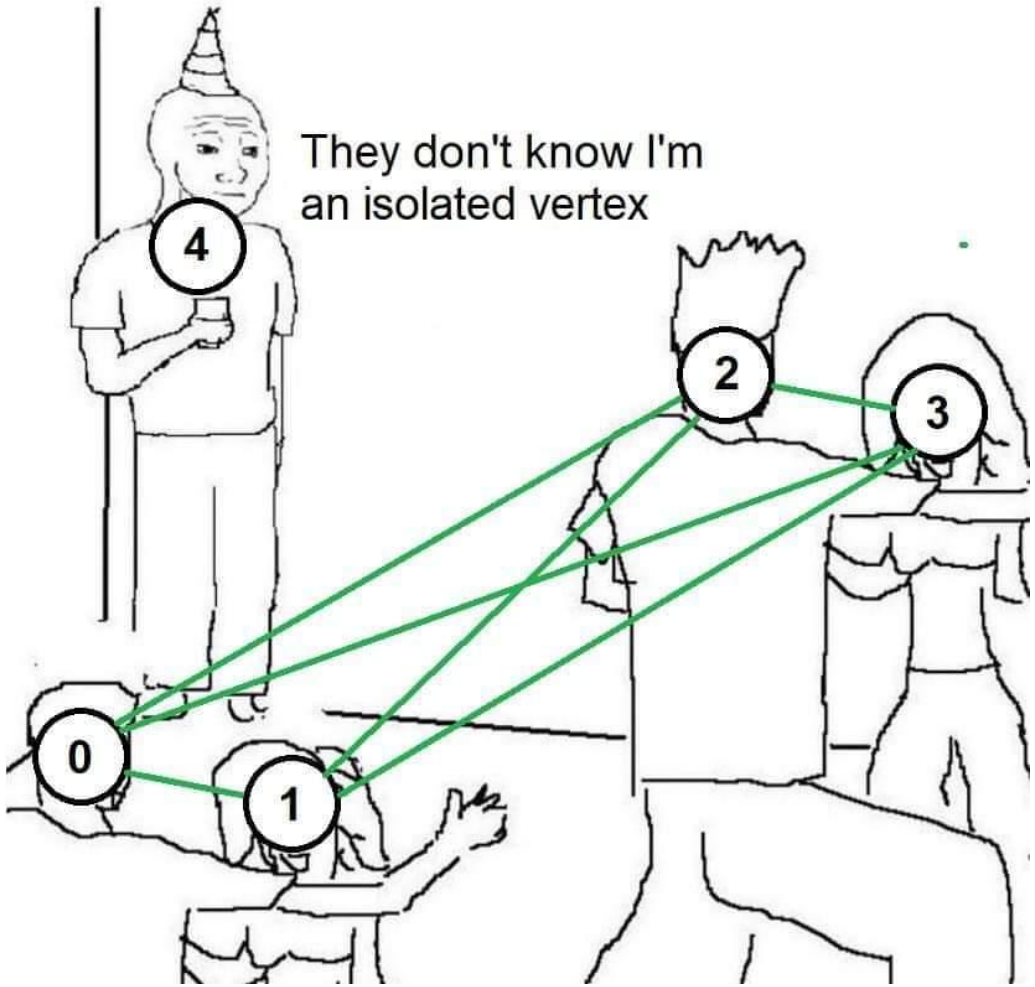
https://www.cs.usfca.edu/~galles/visualization/DFS.html

Matrices de adyacencia



https://www.cs.usfca.edu/~galles/visualization/DFS.html

Pueden haber vértices aislados en un grafo



¿Cómo se representa un vértice aislado?

Fila de ceros

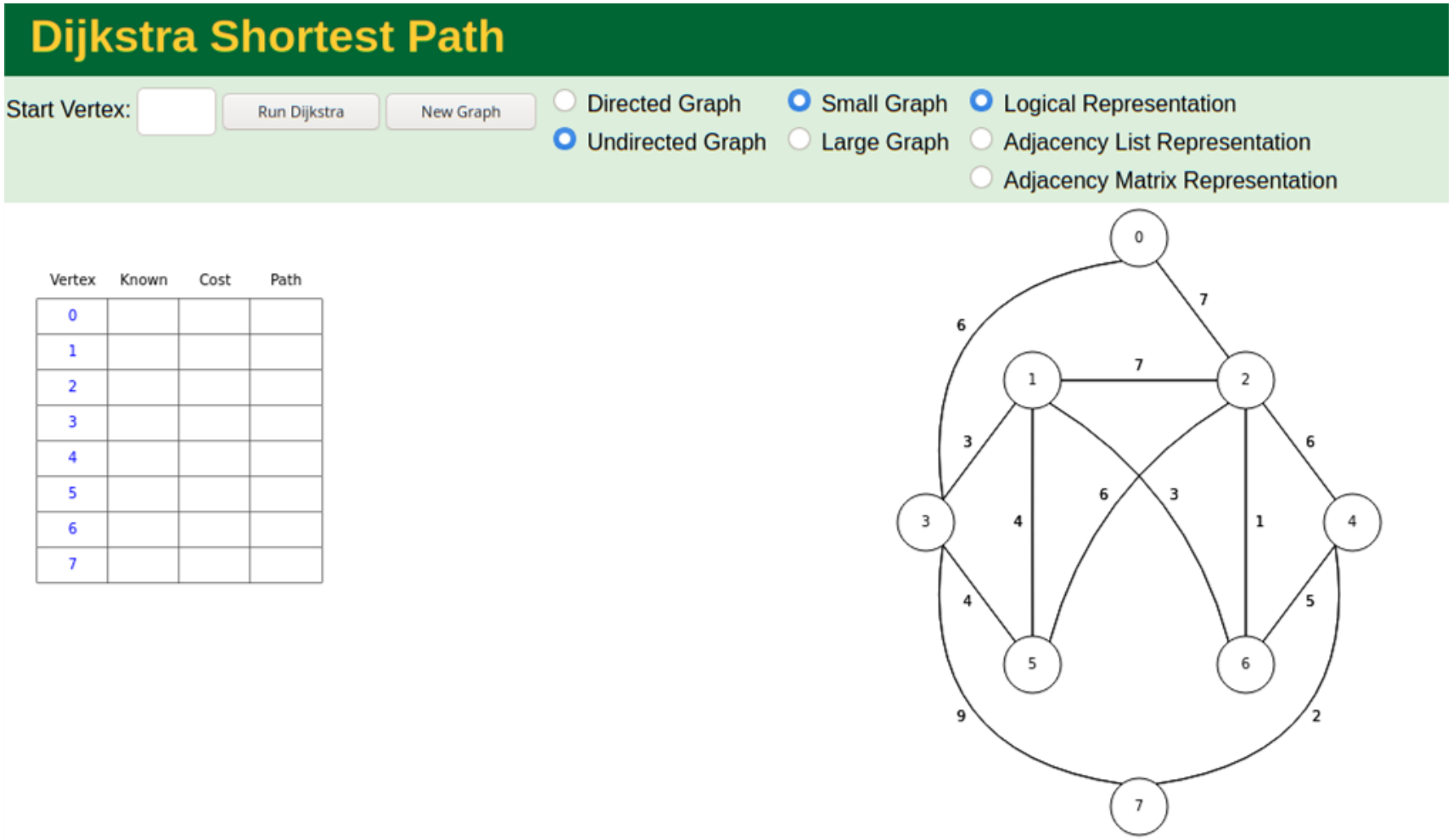
Columna de ceros

Fila y columna de ceros

No se puede

En Matrices de Adyacencia

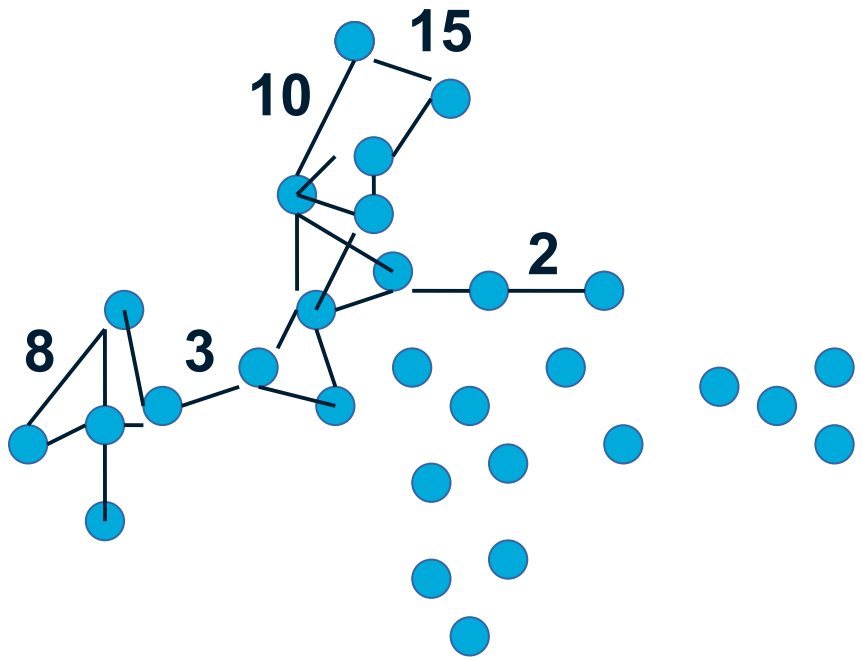
Aristas con pesos



https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Aristas con pesos

Las aristas pueden tener pesos



¿Cuál es la complejidad de modificar el peso de un arco?

O(1)

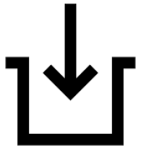
O(V log E)

$O(V^2)$

O(V * E)

En Matrices de Adyacencia

Complejidad en el tiempo



Obtener un peso → O(1)

Cambiar un peso → O(1)

Insertar un vértice →

Borrar un vértice →

En Matrices de Adyacencia

¿Cuál es la complejidad de insertar un nuevo vértice?

O(1)

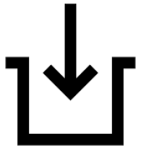
$O(V^2)$

O(V log E)

O(V * E)

En Matrices de Adyacencia

Complejidad en el tiempo



Obtener un peso → O(1)

Cambiar un peso → O(1)

Insertar un vértice → $O(V^2)$

Borrar un vértice → $O(V^2)$

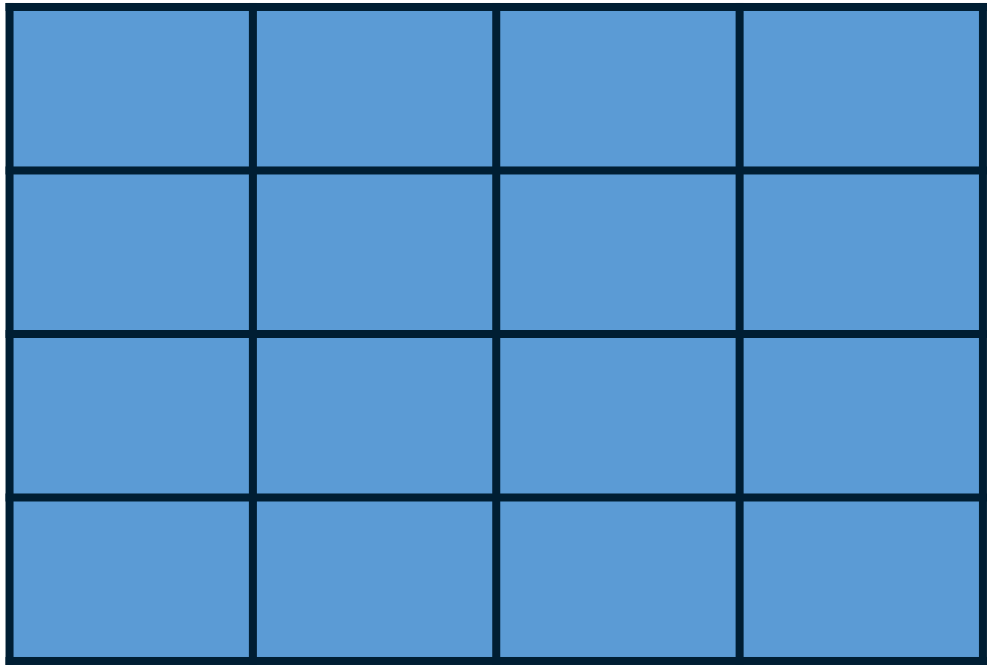
En Matrices de Adyacencia

¿Por qué insertar es $O(V^2)$?

1	2	3
4	5	6
7	8	9

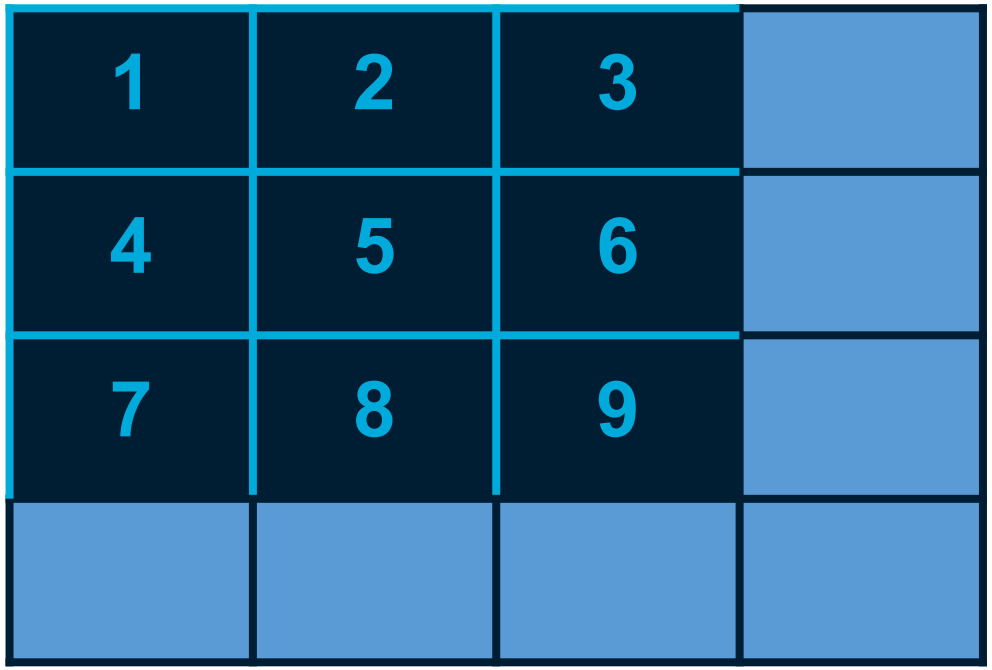
En Matrices de Adyacencia

¿Por qué insertar es $O(V^2)$?



En Matrices de Adyacencia

¿Por qué insertar es $O(V^2)$?



En Matrices de Adyacencia

Complejidad en memoria

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

...Tanto **tiempo** pasó desde el día que te fuiste…

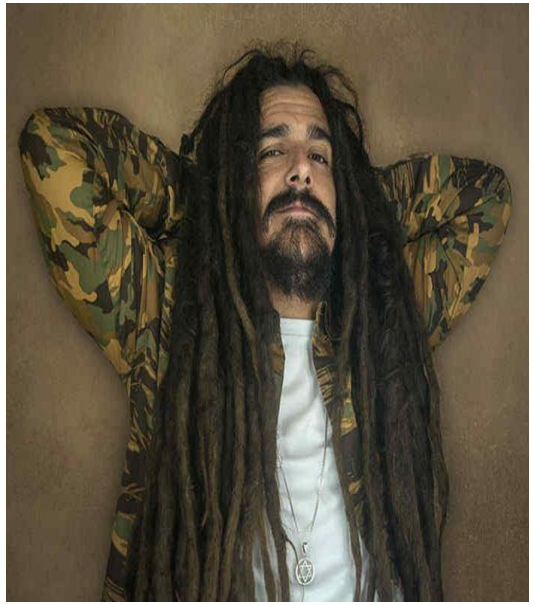
Y allí supe que las despedidas son muy tristes

En Matrices de Adyacencia

Complejidad en memoria

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

¡Pero necesitamos Memoria y NO Tiempo!



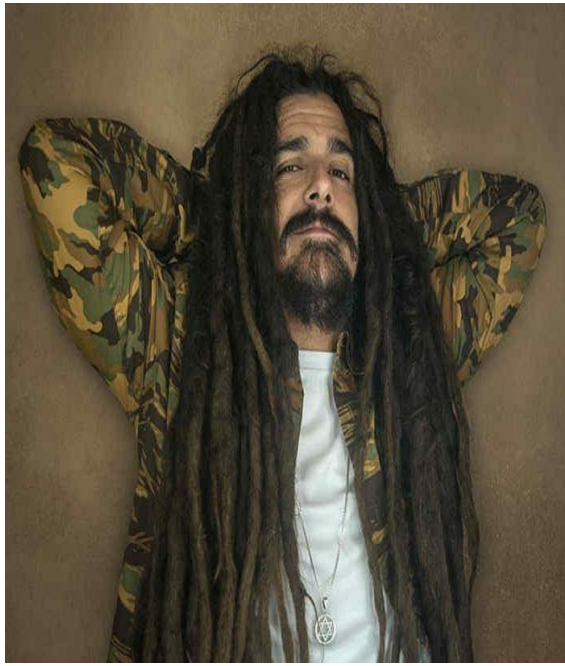
En Matrices de Adyacencia

Complejidad en memoria

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

...Tanta **memoria** pasó desde el día que te fuiste

Y allí supe que las despedidas son muy tristes



En Matrices de Adyacencia

¿Cuánta memoria consume una matriz de adyacencia?

O(V)

$O(V^2)$

O(V log E)

O(V * E)

En Matrices de Adyacencia

Complejidad en memoria

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

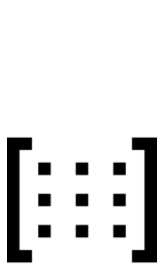
Consume
$O(V*V) = O(V^2)$
$= O(V+V+V+V\ldots$
V veces)

En Matrices de Adyacencia

Complejidad en memoria en la vida real

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

¿Cuánto ocupa la matriz de adyacencia de Facebook Colombia?



En Matrices de Adyacencia

¿Cuánta memoria consume una matriz de adyacencia con V = 40 M?

1387 TB

1387 MB

1387 KB

1387 GB

En Matrices de Adyacencia

Complejidad en memoria en la vida real

	0	1	2	3	4	5	6	7
0	∞	5	7	3	∞	∞	∞	∞
1	5	∞	∞	∞	2	10	∞	∞
2	7	∞	∞	∞	∞	∞	1	∞
3	3	∞	∞	∞	∞	∞	∞	11
4	∞	2	∞	∞	∞	∞	∞	9
5	∞	10	∞	∞	∞	∞	∞	4
6	∞	∞	1	∞	∞	∞	∞	6
7	∞	∞	∞	6	11	9	4	∞

1387 TB !!!

1387 discos duros

https://web2.0calc.com/

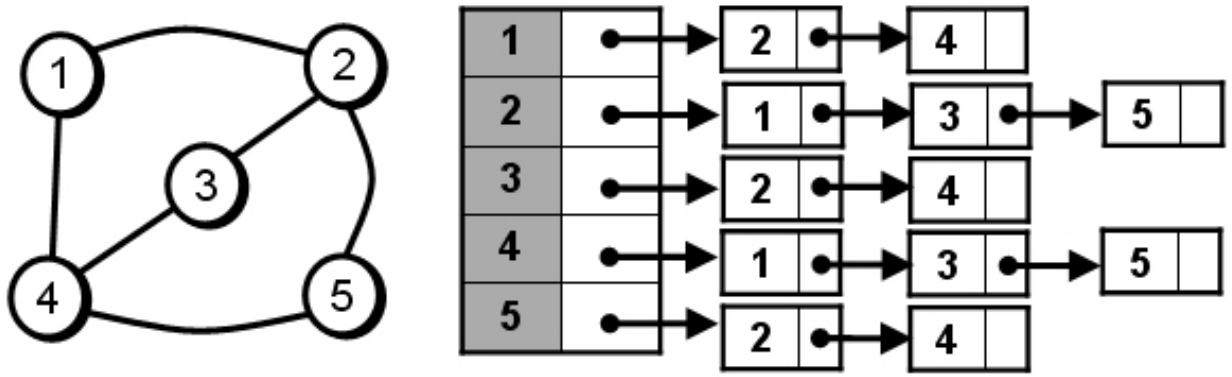

UN MAPA CON MATRICES DE ADYACENCIA...
¡USA LISTAS DE ADYACENCIA!
imgflip.com

Representación de grafos direccionados (digraphs)

Grafo direccionado = Arcos pasan en una sola dirección

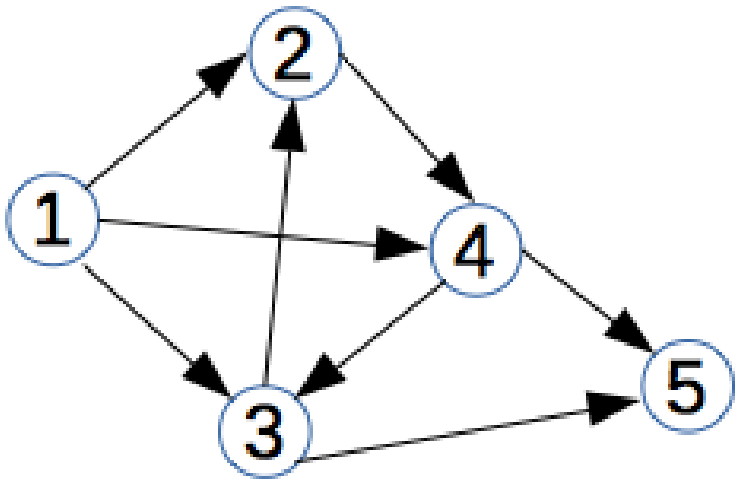
Lista de adyacencia:

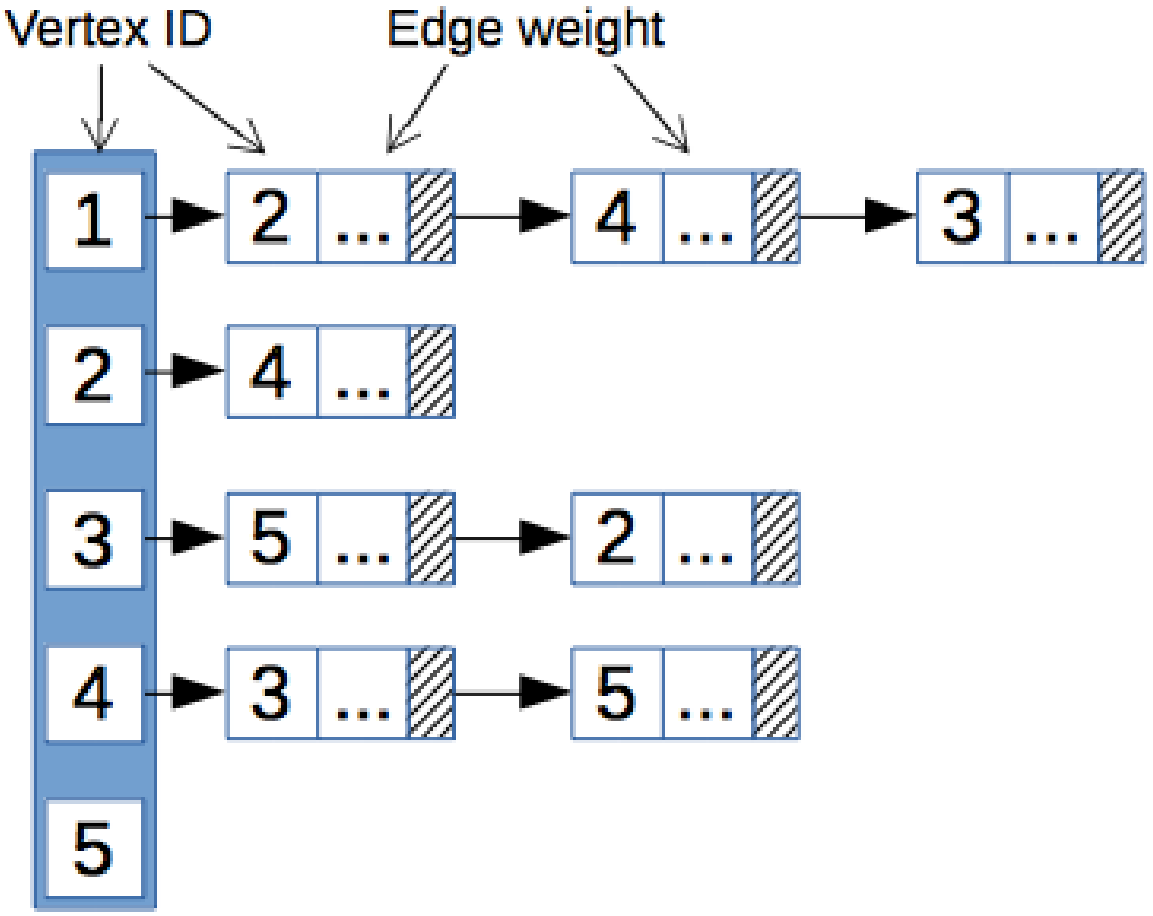
- Asociar cada nodo a una lista de nodos de destino
- Complejidad de consulta O(n)



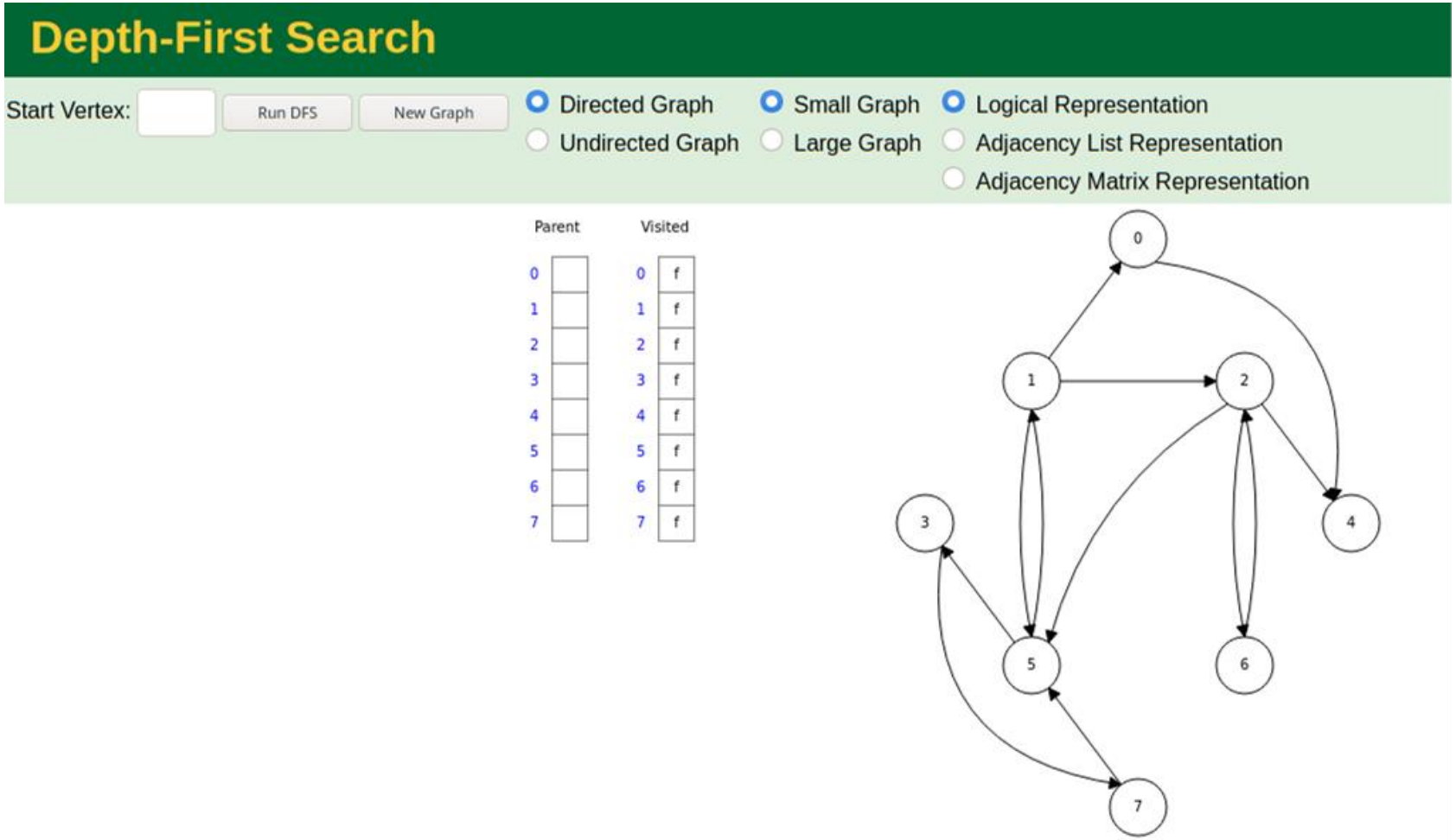
https://es.m.wikipedia.org/wiki/Archivo:Listas_de_adyacencia.jpg

Listas de adyacencia



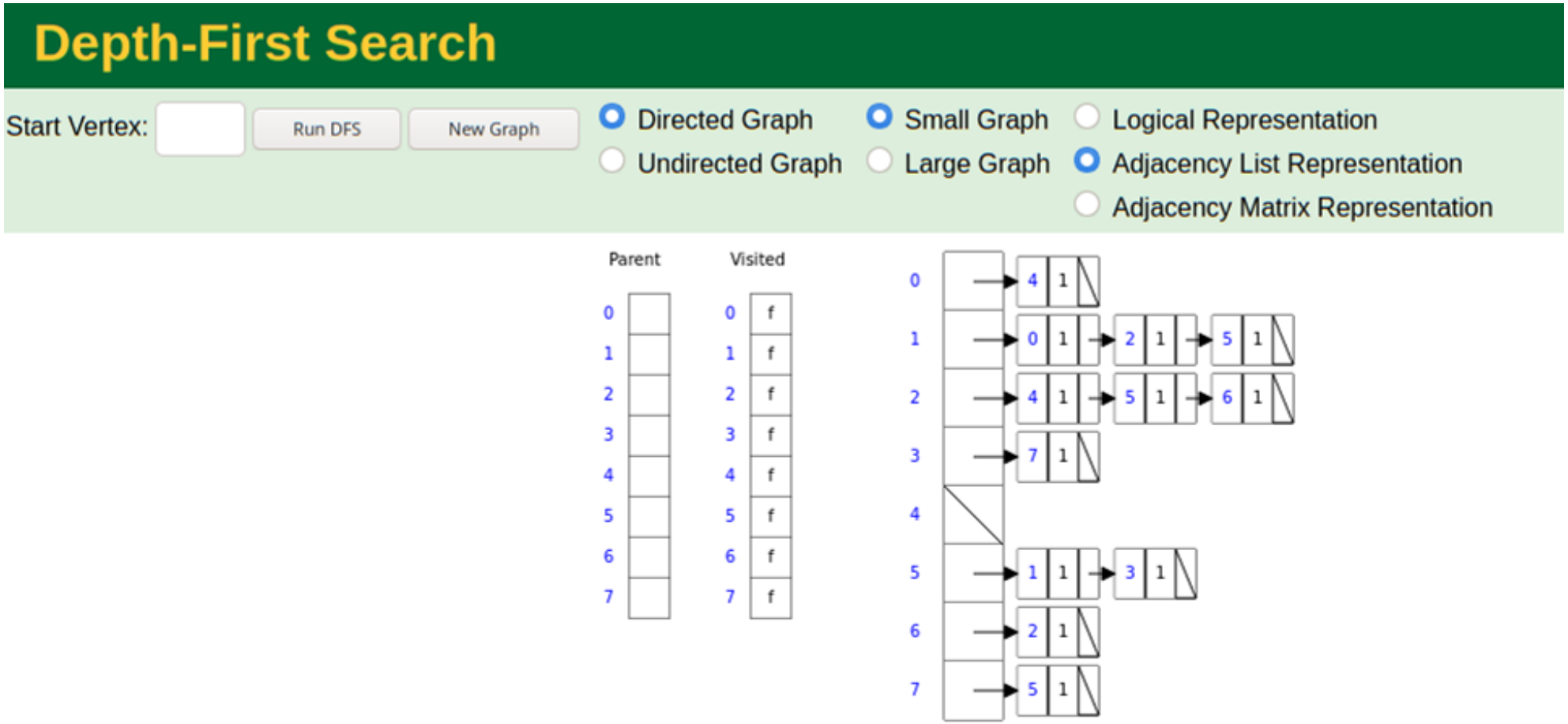


Listas de adyacencia



https://www.cs.usfca.edu/~galles/visualization/DFS.html

Listas de adyacencia



https://www.cs.usfca.edu/~galles/visualization/DFS.html

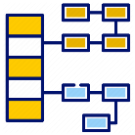
¿Cuál es la complejidad de buscar el peso de un arco?

O(1)

O(V log E)

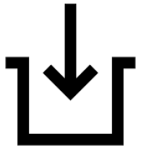
$O(V^2)$

O(V)



En Listas de Adyacencia

Complejidad en el tiempo

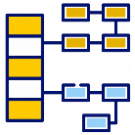


Obtener un peso → O(V)

Cambiar un peso → O(V)

Insertar un vértice →

Borrar un vértice →



En Listas de Adyacencia

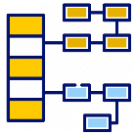
¿Cuál es la complejidad de insertar un nuevo vértice?

O(1)

$O(V^2)$

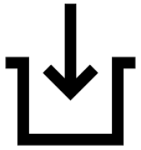
O(V log E)

O(V * E)



En Listas de Adyacencia

Complejidad en el tiempo

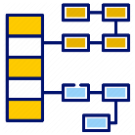


Obtener un peso → O(V)

Cambiar un peso → O(V)

Insertar un vértice → O(V)

Borrar un vértice → O(V)



En Listas de Adyacencia

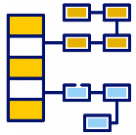
¿Cómo se representa un vértice aislado?

Lista vacía

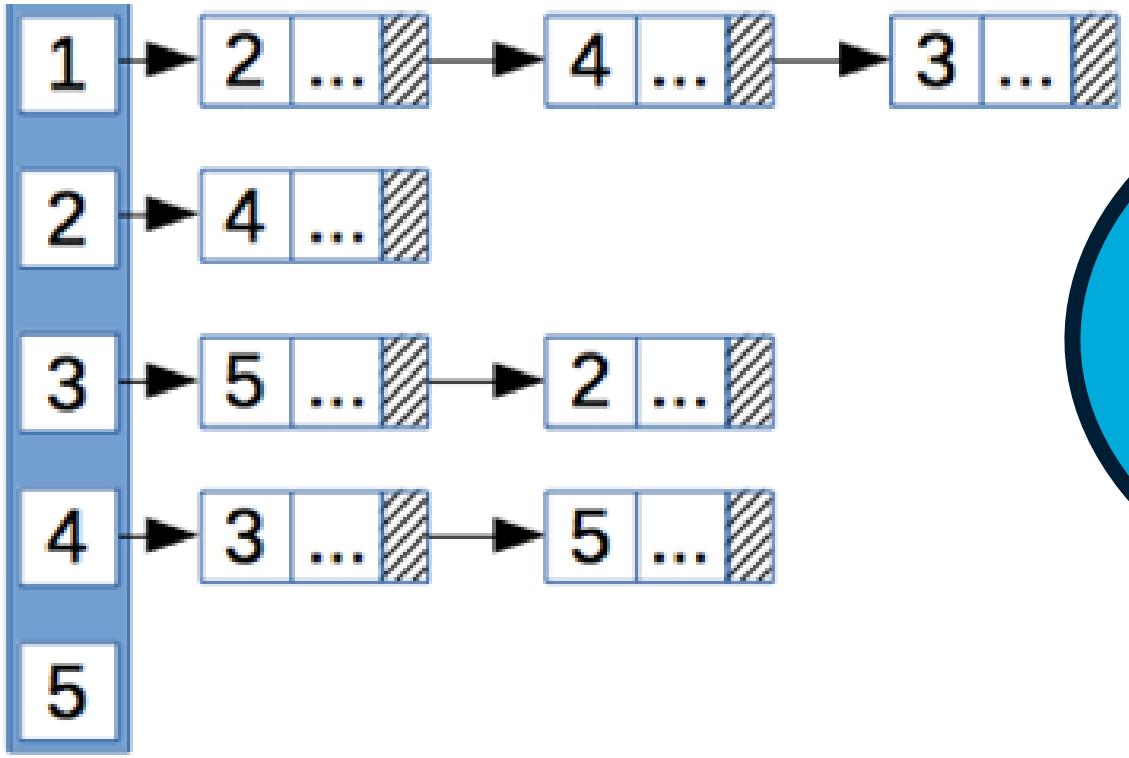
Lista de ceros

No se puede

Lista vacía y
no aparece en otras listas



En Listas de Adyacencia



¡Necesitamos es Memoria y NO Tiempo!

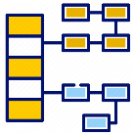
¿Cuánta memoria consume una lista de adyacencias?

O(V)

$O(V^2)$

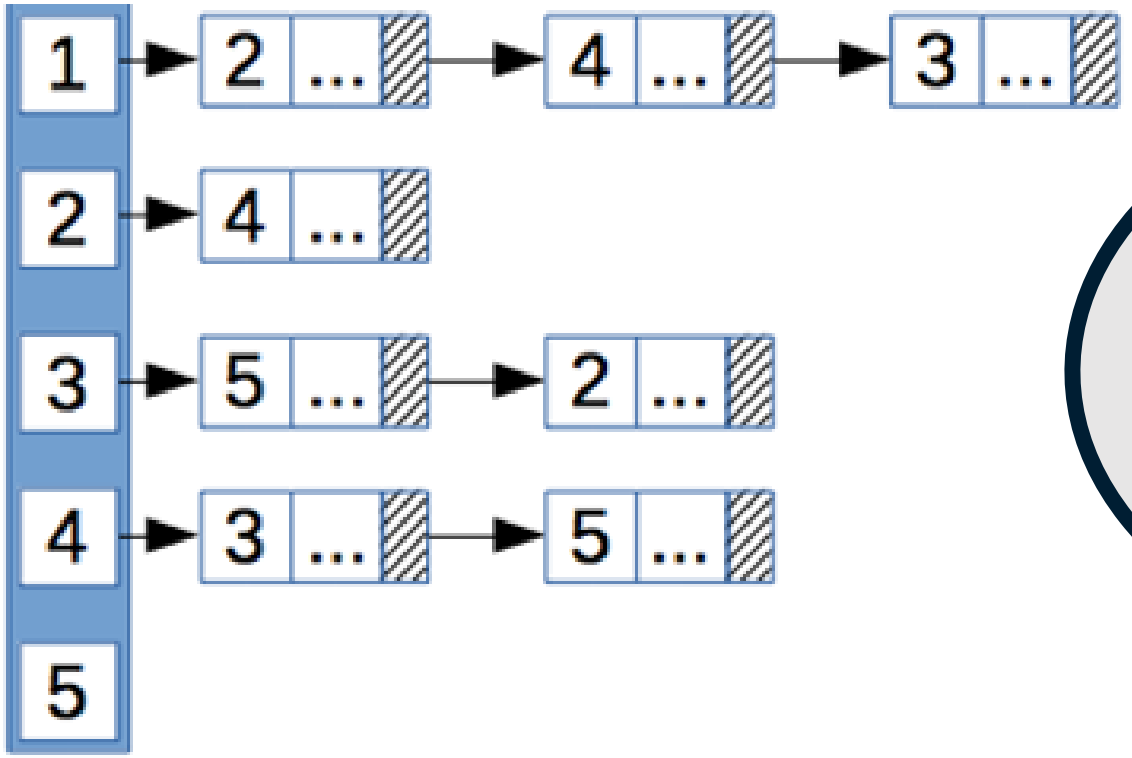
O(V log E)

$O(V * E) = O(V^3)$

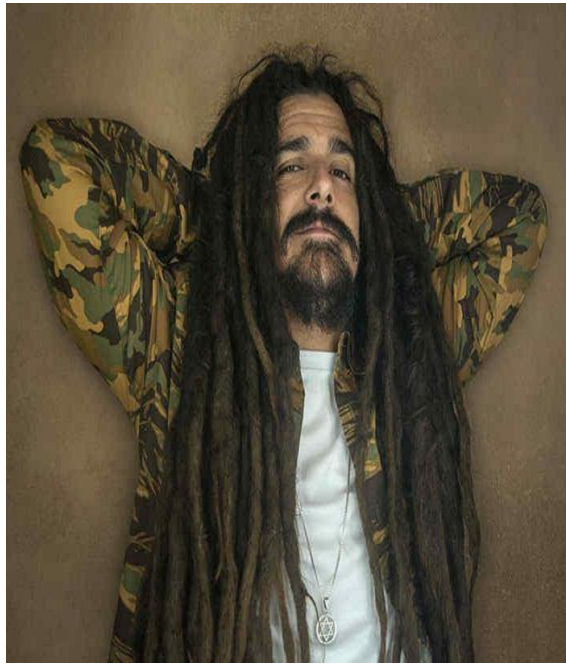


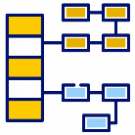
En Listas de Adyacencia

Complejidad en memoria



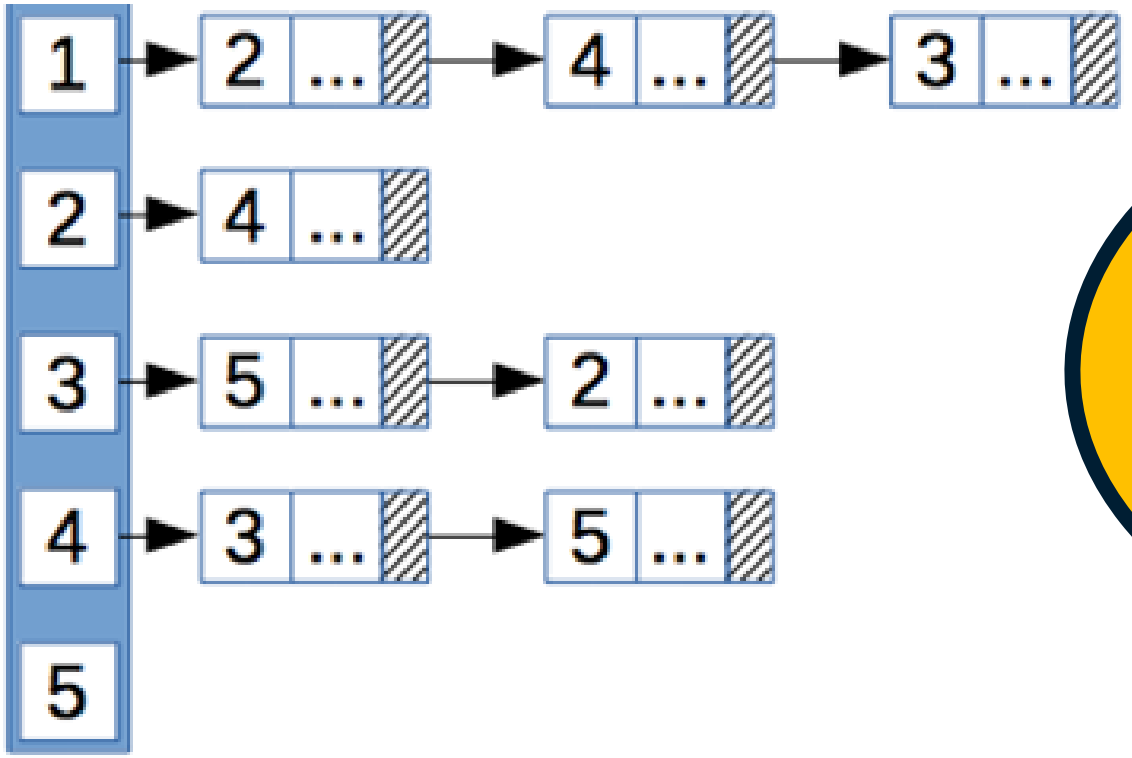
Consume
$O(V*V) = O(V^2)$





En Listas de Adyacencia

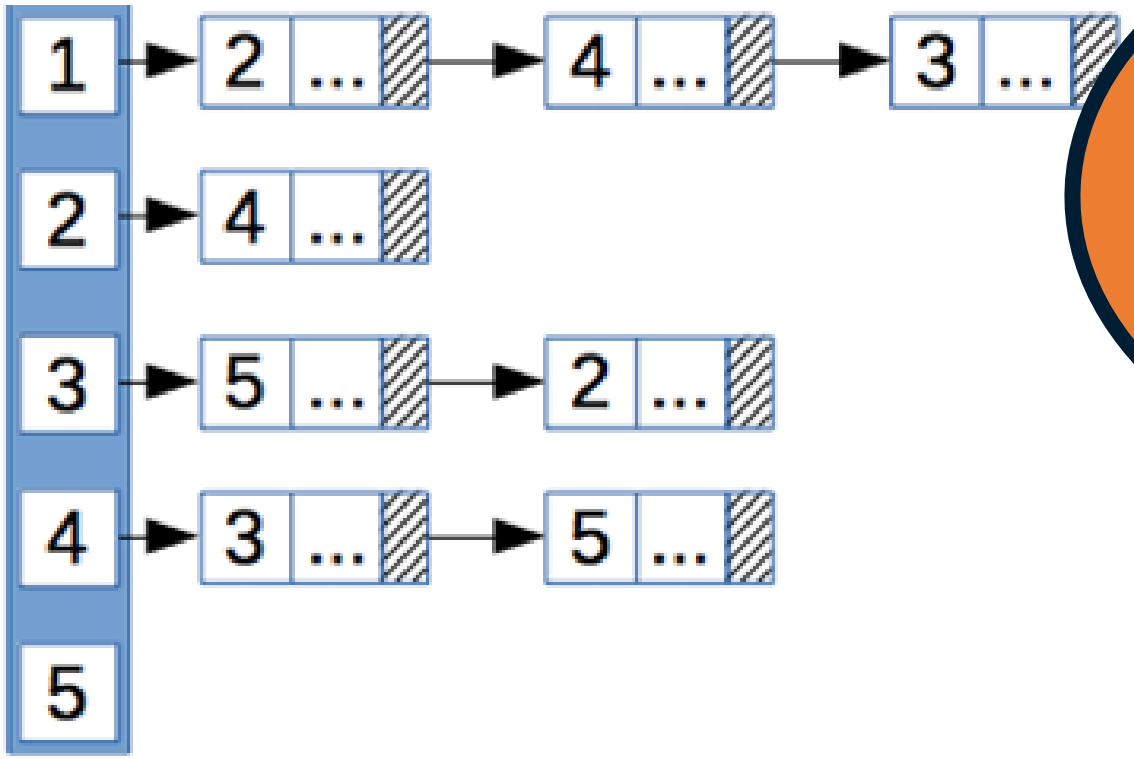
Complejidad en memoria



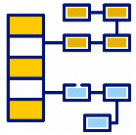
$O(V*V) = O(V^2)$
…en el peor caso, cuando
$E = V^2$

En Listas de Adyacencia

Complejidad en memoria en la vida real



¿Cuánto ocupa la lista de adyacencia de Facebook Colombia?



En Listas de Adyacencia

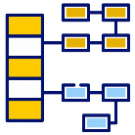
¿Cuánta memoria consume una lista de adyacencia con V = 40 M?

305 TB

305 MB

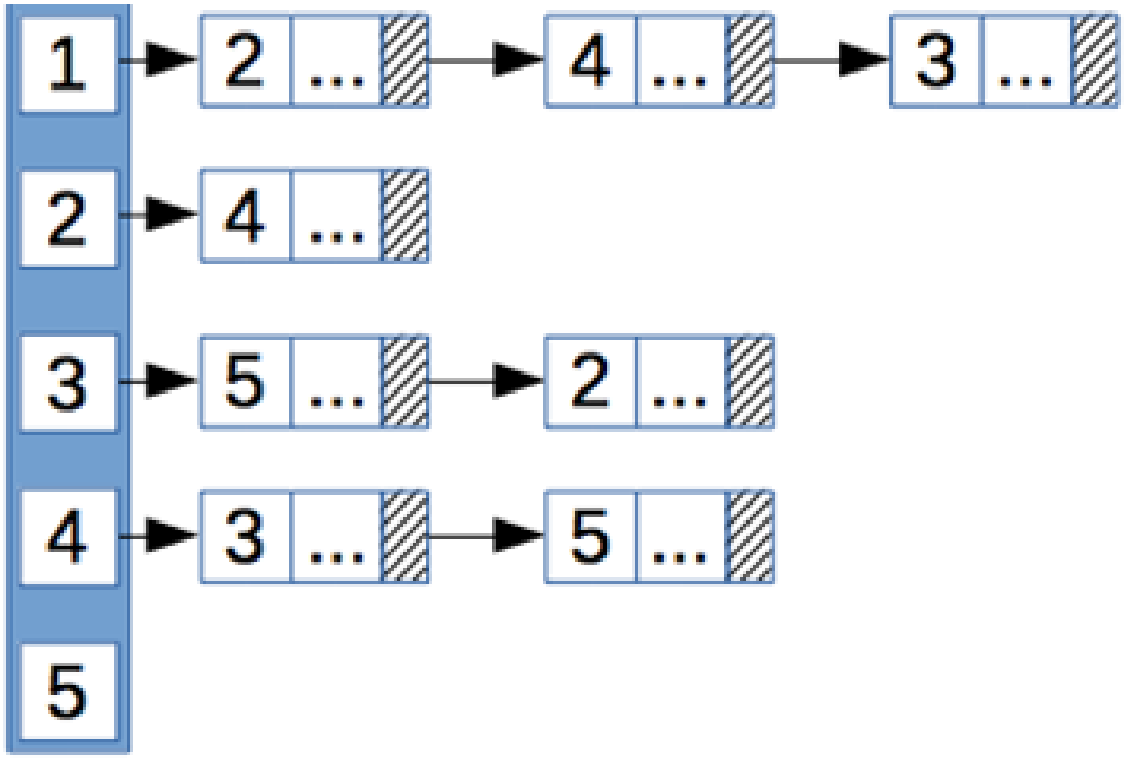
305 KB

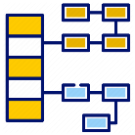
305 GB



En Listas de Adyacencia

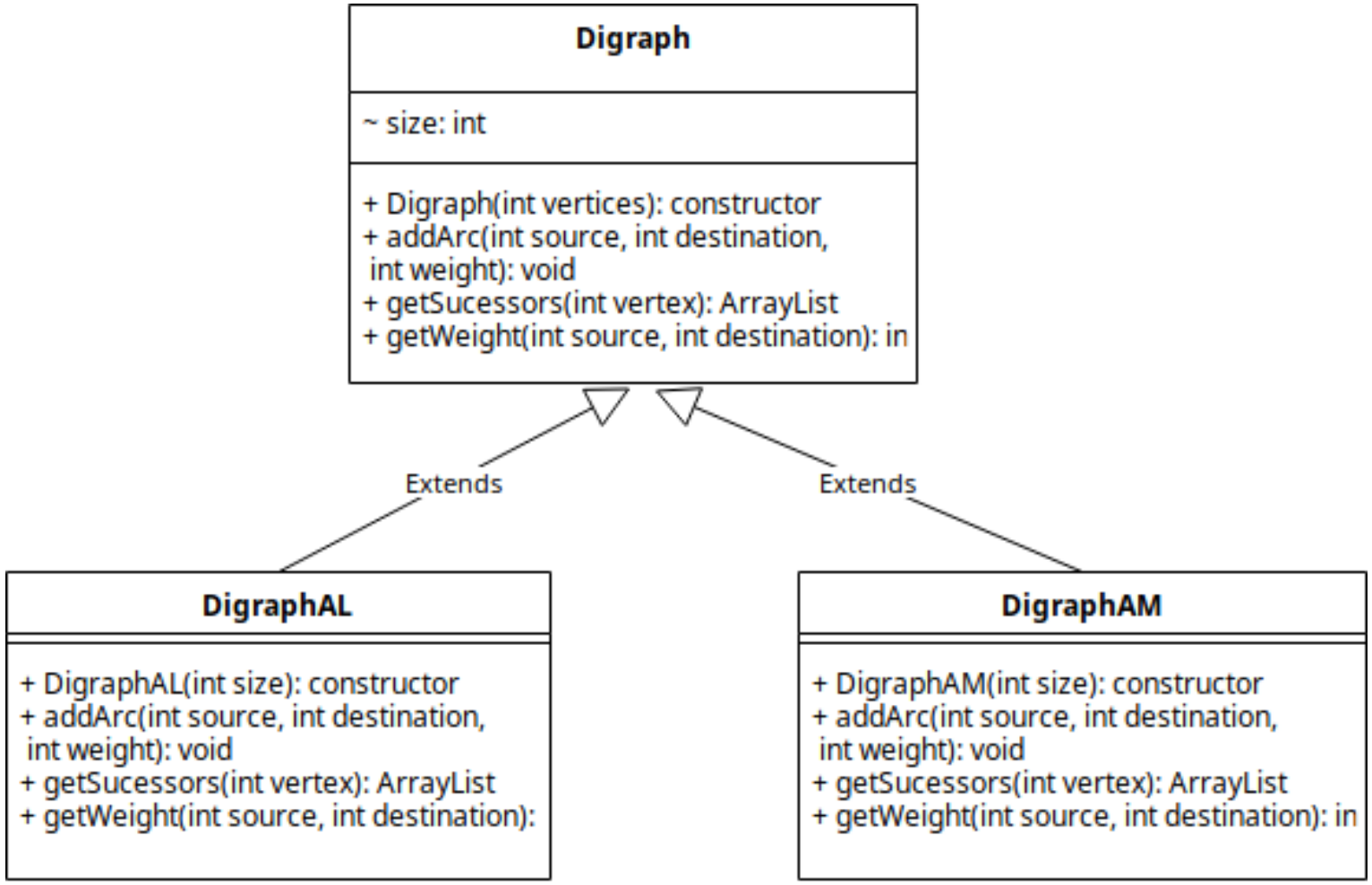
Complejidad en memoria en la vida real



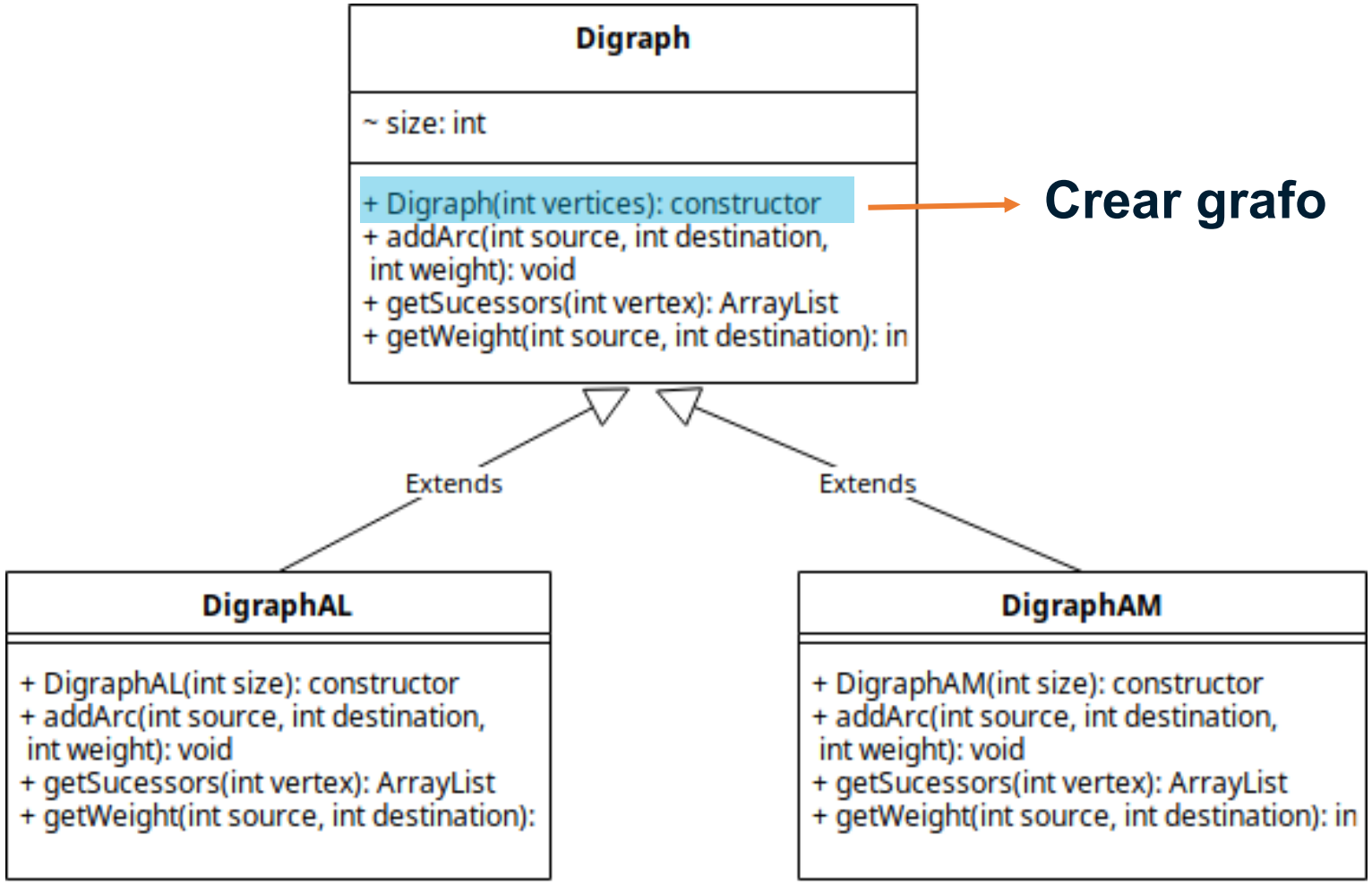


https://web2.0calc.com/

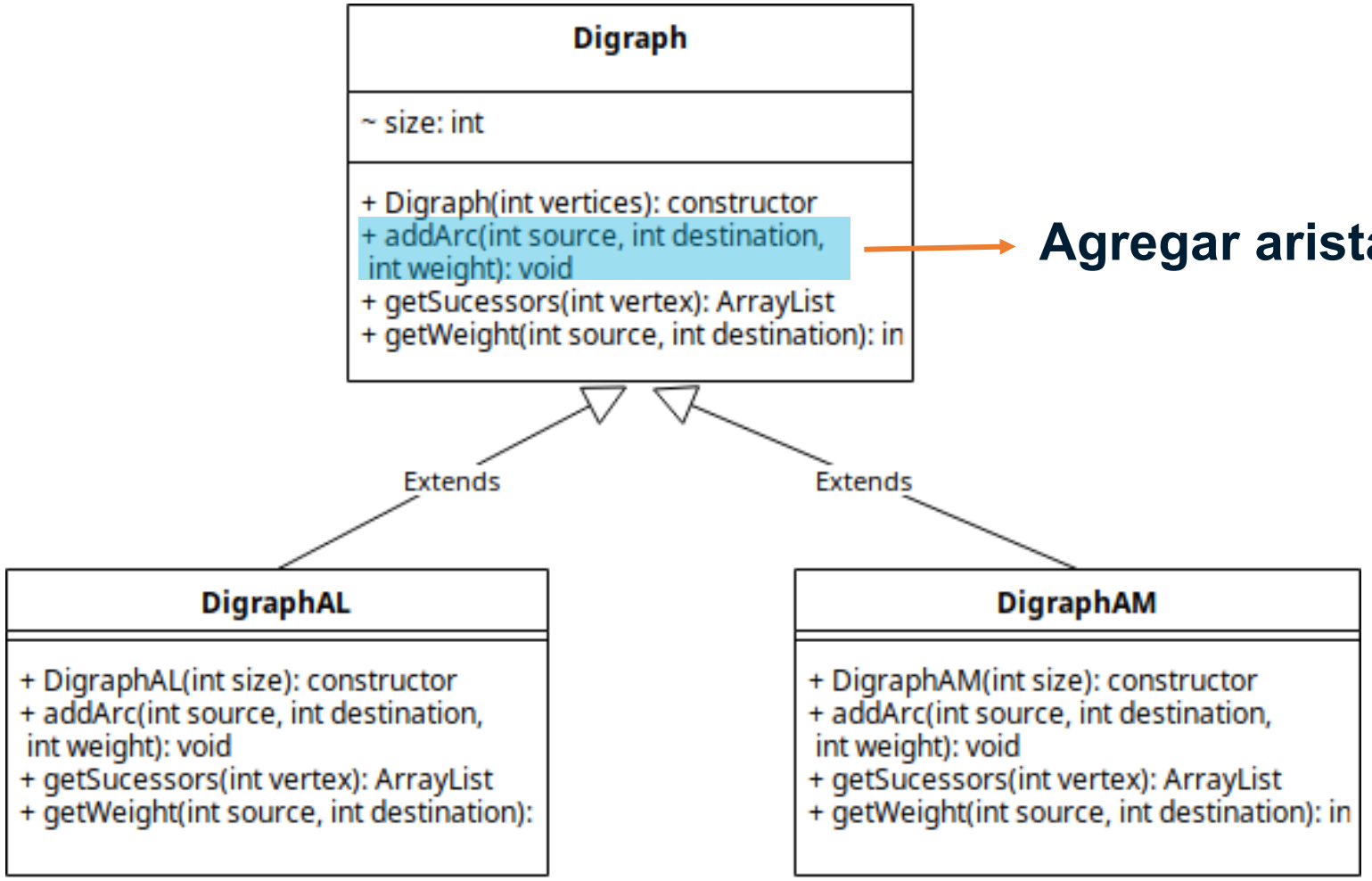
Implementación de un grafo dirigido (digrafo)



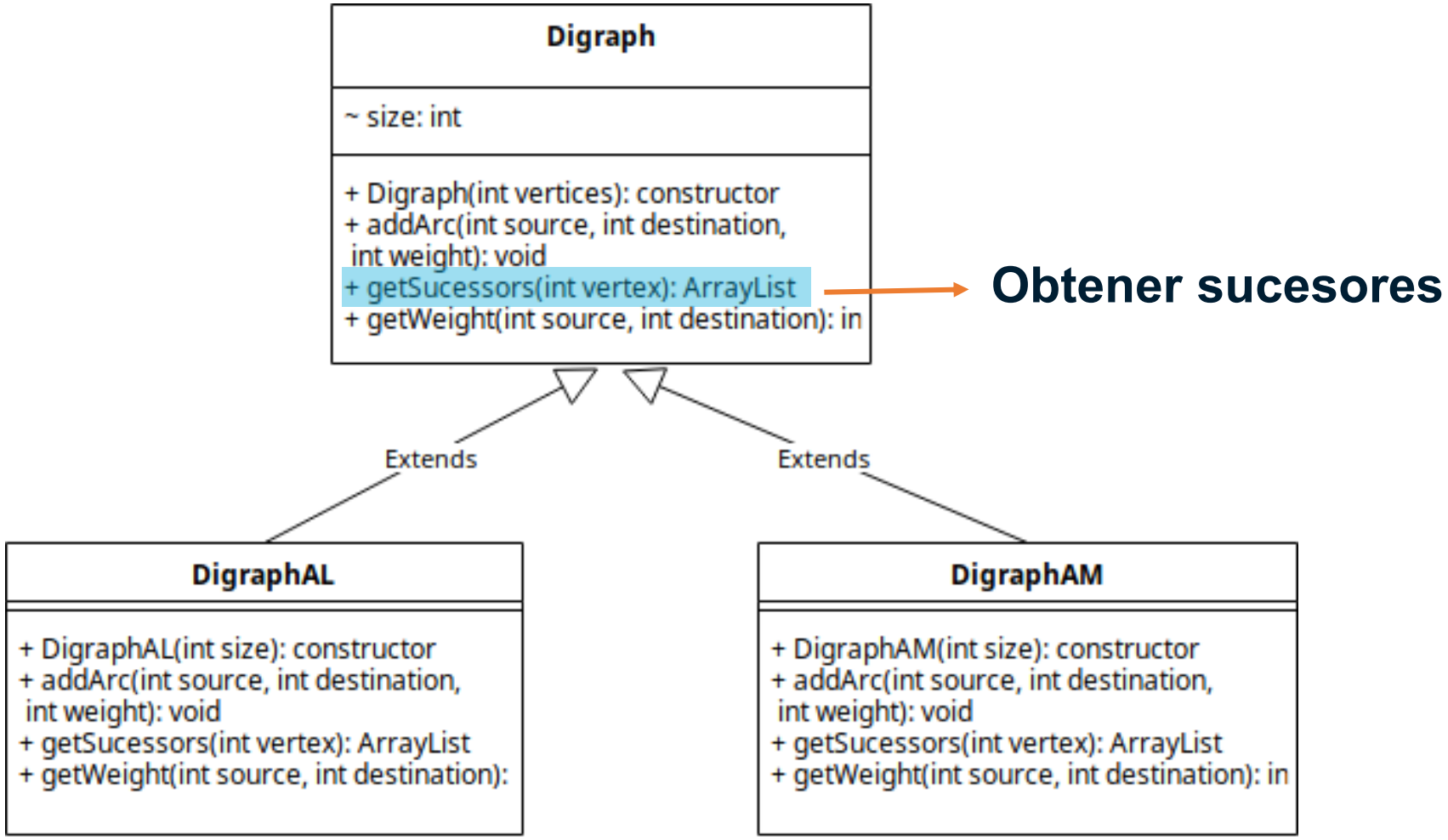
Operaciones de un grafo dirigido (digrafo)



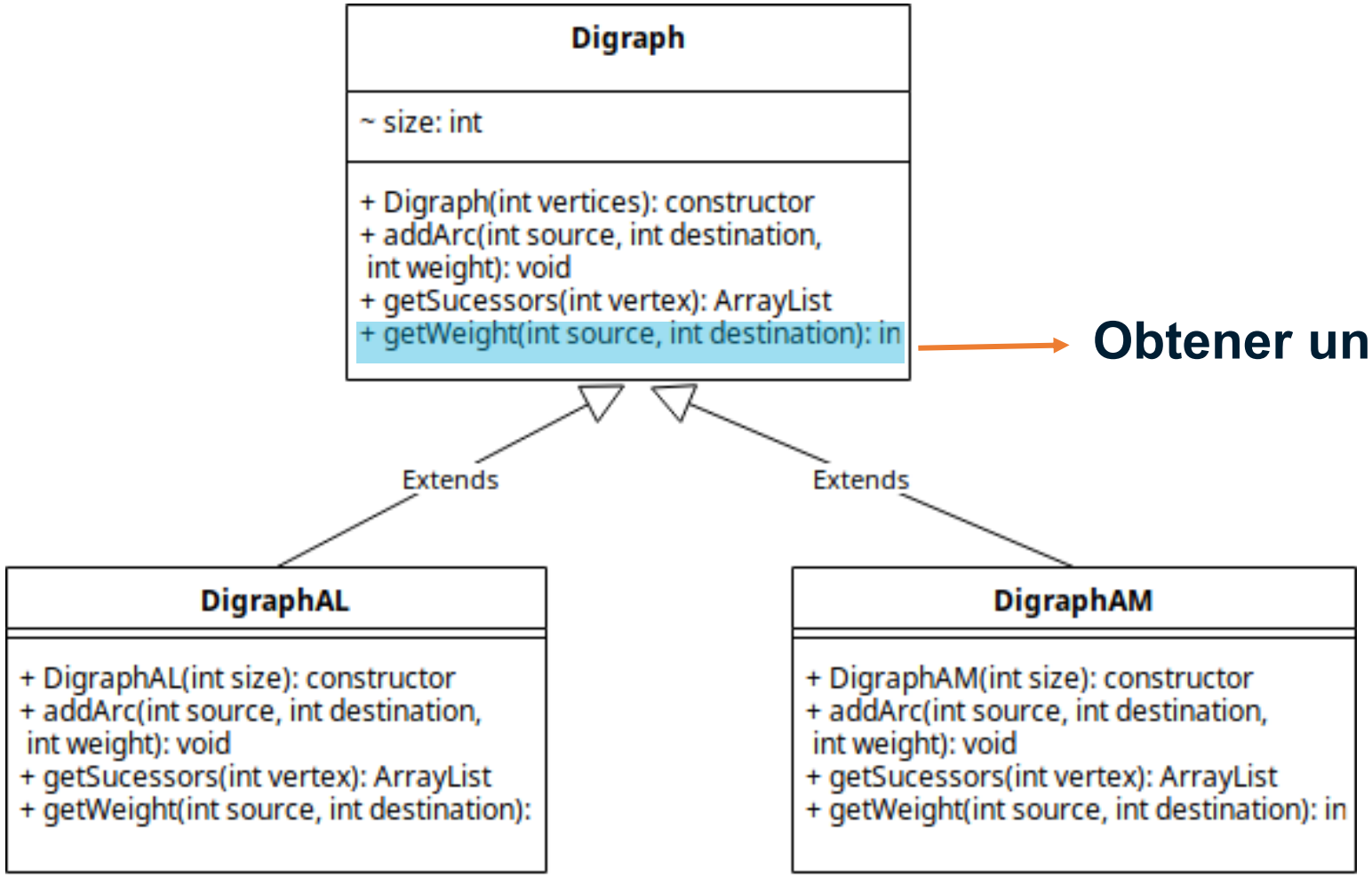
Operaciones de un grafo dirigido (digrafo)



Operaciones de un grafo dirigido (digrafo)



Operaciones de un grafo dirigido (digrafo)



Implementación de grafos

- Para implementar y usar grafos necesitaríamos:
  - Nodos
  - Arcos
  - Juntarlos para crear caminos
- Una vez creados los caminos podemos:
  - Buscar el camino más corto
  - Buscar el camino más óptimo

Implementación de grafos – Clase Node

```
class Node(object):
    def __init__(self, name):
        """Assumes name is a string"""
        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name
```

Implementación de grafos – Clase Edge

```
class Edge(object):
    def __init__(self, src, dest):
        """Assumes src and dest are nodes"""
        self.src = src
        self.dest = dest
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def __str__(self):
        return self.src.getName() + '->'\
               + self.dest.getName()
```

Implementación de grafos – Clase Digraph

```
class Digraph(object):
    """edges is a dict mapping each node to a list of
    its children"""

    def __init__(self):
        self.edges = {}

    def addNode(self, node):
        if node in self.edges:
            raise ValueError('Duplicate node')
        else:
            self.edges[node] = []

    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not (src in self.edges and dest in self.edges):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
```

Nodes are represented as keys in dictionary

Edges are represented by destinations as values in list associated with a source key

Implementación de grafos – Clase Digraph

```
def childrenOf(self, node):
    return self.edges[node]

def hasNode(self, node):
    return node in self.edges

def getNode(self, name):
    for n in self.edges:
        if n.getName() == name:
            return n
    raise NameError(name)

def __str__(self):
    result = ''
    for src in self.edges:
        for dest in self.edges[src]:
            result = result + src.getName() + '->'\
                     + dest.getName() + '\n'
    return result[:-1] #omit final newline
```

Implementación de grafos – Clase Graph

```
class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
```

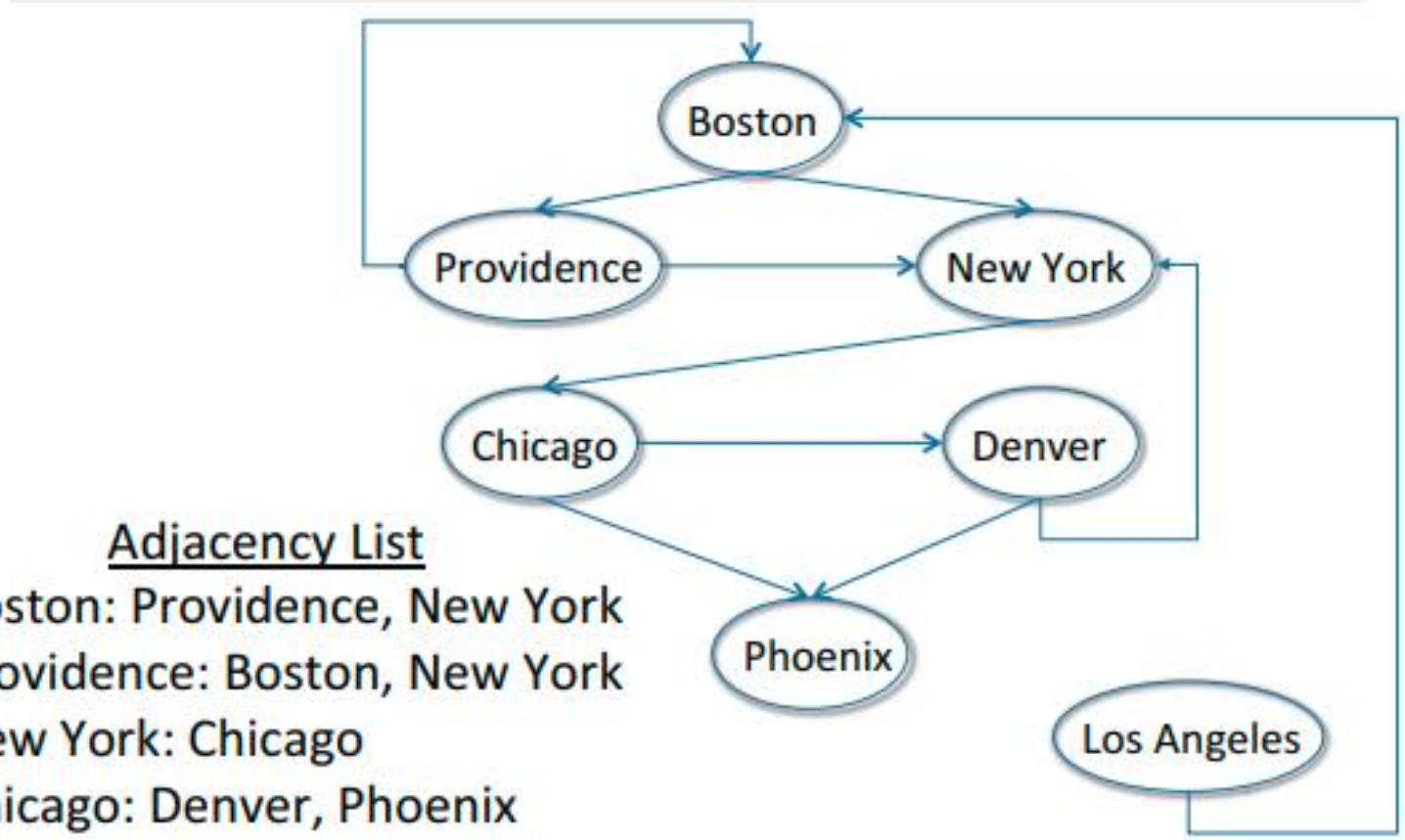
Grafo No direccionado: arcos conectan nodos en ambas direcciones

¿Por qué una subclase de Digraph?

Uso de grafos – Algunos problemas clásicos

- Encontrar la ruta más corta entre dos ciudades
- Diseño de redes de comunicación
- Encontrar un camino para una molécula en una estructura química
- …

Ejemplo



Adjacency List

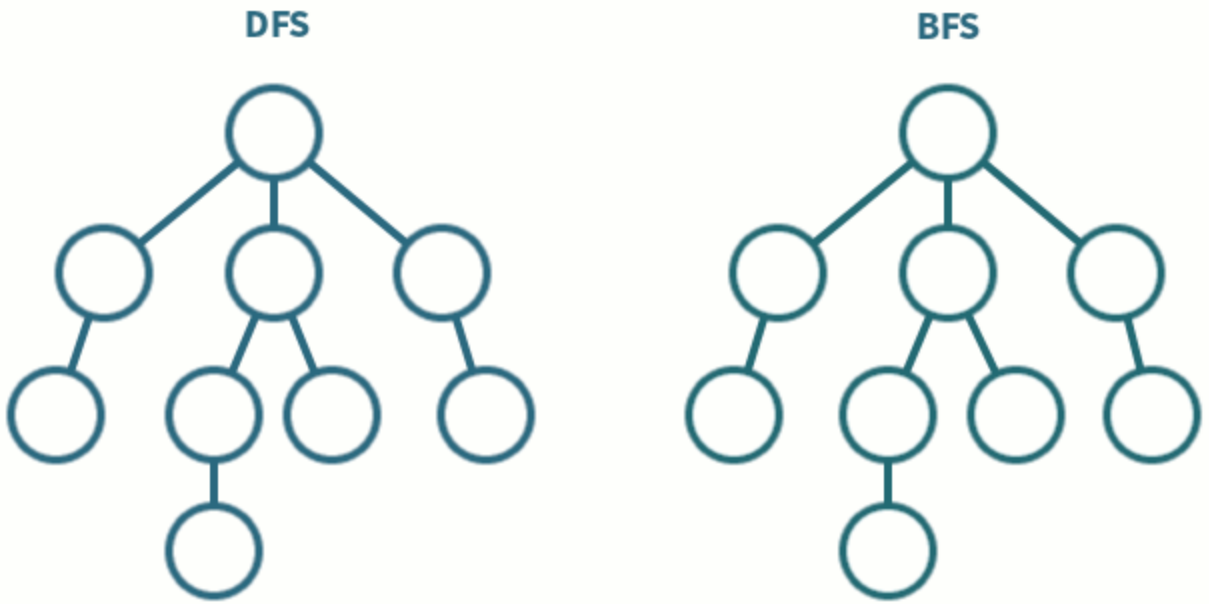
Boston: Providence, New York
Providence: Boston, New York
New York: Chicago
Chicago: Denver, Phoenix
Denver: Phoenix, New York
Los Angeles: Boston
Phoenix:

Ejemplo – Creando el grafo

```
def buildCityGraph(graphType):
    g = graphType()
    for name in ('Boston', 'Providence', 'New York', 'Chicago',
                 'Denver', 'Phoenix', 'Los Angeles'): #Create 7 nodes
        g.addNode(Node(name))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('Providence')))
    g.addEdge(Edge(g.getNode('Boston'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('Boston')))
    g.addEdge(Edge(g.getNode('Providence'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('New York'), g.getNode('Chicago')))
    g.addEdge(Edge(g.getNode('Chicago'), g.getNode('Denver')))
    g.addEdge(Edge(g.getNode('Chicago'), g.getNode('Phoenix')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('Phoenix')))
    g.addEdge(Edge(g.getNode('Denver'), g.getNode('New York')))
    g.addEdge(Edge(g.getNode('Los Angeles'), g.getNode('Boston')))
```

Ejemplo – Encontrando el camino más corto

- Existen dos alternativas clásicas y simples para esto:
  - Algoritmo de búsqueda por profundidad (Depth-First Search)
  - Algoritmo de búsqueda por anchura (Breadth-First Search)



Ejemplo – Depth-First Search

- Inicia un nodo origen
- Considera todos los arcos que tiene dicho nodo en algún orden
- Continuando con el primer arco, verifica si el nodo destino es el objetivo
- Si no, repita el proceso con el siguiente nodo
- Continúa hasta encontrar el nodo objetivo o quedarse sin opciones
  - Si se queda sin opciones, regresar al nodo previo e intentar con el siguiente arco y repetir el proceso.

Ejemplo – Depth-First Search

```
def DFS(graph, start, end, path, shortest, toPrint = False):
    path = path + [start]
    if toPrint:
        print('Current DFS path:', printPath(path))
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path: #avoid cycles
            if shortest == None or len(path) < len(shortest):
                newPath = DFS(graph, node, end, path, shortest, toPrint)
                if newPath != None:
                    shortest = newPath
        elif toPrint:
            print('Already visited', node)
    return shortest


def shortestPath(graph, start, end, toPrint = False):
    return DFS(graph, start, end, [], None, toPrint)
```

... returning to this point in the recursion to try next node

Note how will explore all paths through first node, before ...

DFS called from a wrapper function: shortestPath

Gets recursion started properly

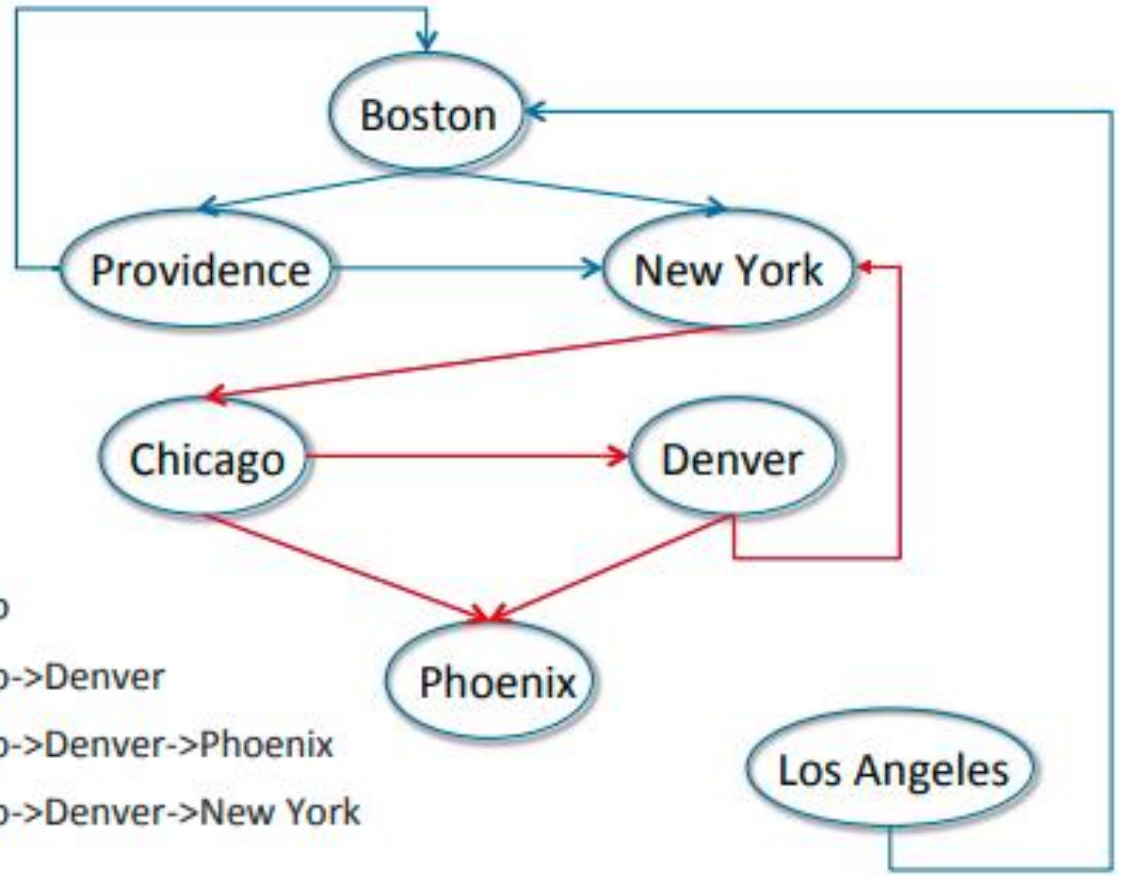
Provides appropriate abstraction

Ejemplo – Depth-First Search

```
def testSP(source, destination):
    g = buildCityGraph(DiGraph)
    sp = shortestPath(g, g.getNode(source), g.getNode(destination)
                      toPrint = True)
    if sp != None:
        print('Shortest path from', source, 'to',
              destination, 'is', printPath(sp))
    else:
        print('There is no path from', source, 'to', destination)

testSP('Boston', 'Chicago')
```

Ejemplo – Depth-First Search – Chicago a Boston



Current DFS path: Chicago

Current DFS path: Chicago->Denver

Current DFS path: Chicago->Denver->Phoenix

Current DFS path: Chicago->Denver->New York

Already visited Chicago

Current DFS path: Chicago->Phoenix

There is no path from Chicago to Boston

Ejemplo – Depth-First Search – Boston a Phoenix

Current DFS path: Boston

Current DFS path: Boston->Providence

Already visited Boston

Current DFS path: Boston->Providence->New York

Current DFS path: Boston->Providence->New York->Chicago

Current DFS path: Boston->Providence->New York->Chicago->Denver

Current DFS path: Boston->Providence->New York->Chicago->Denver->Phoenix Found path

Already visited New York

Current DFS path: Boston->Providence->New York->Chicago->Phoenix Found a shorter path

Current DFS path: Boston->New York

Current DFS path: Boston->New York->Chicago

Current DFS path: Boston->New York->Chicago->Denver

Current DFS path: Boston->New York->Chicago->Denver->Phoenix Found a "shorter" path

Already visited New York

Current DFS path: Boston->New York->Chicago->Phoenix Found a shorter path

Shortest path from Boston to Phoenix is Boston->New York->Chicago->Denver->Phoenix

Ejemplo – Breadth-First Search

- Inicia un nodo origen
- Considera todos los arcos que tiene dicho nodo en algún orden
- Continuando con el primer arco, verifica si el nodo destino es el objetivo
- Si no, repita el proceso con el **siguiente arco del nodo actual**
- Continúa hasta encontrar el nodo objetivo o quedarse sin opciones
  - Si se queda sin opciones, regresar al nodo anterior en el mismo nivel del origen y repita
  - Si se queda sin opciones, moverse al siguiente nivel del grafo y repetir.

Ejemplo – Breadth-First Search

```
def BFS(graph, start, end, toPrint = False):
    initPath = [start]
    pathQueue = [initPath]
    while len(pathQueue) != 0:
        #Get and remove oldest element in pathQueue
        tmpPath = pathQueue.pop(0)
        if toPrint:
            print('Current BFS path:', printPath(tmpPath))
        lastNode = tmpPath[-1]
        if lastNode == end:
            return tmpPath
        for nextNode in graph.childrenOf(lastNode):
            if nextNode not in tmpPath:
                newPath = tmpPath + [nextNode]
                pathQueue.append(newPath)
    return None
```

Ejemplo – Breadth-First Search – Boston a Phoenix

Current BFS path: Boston

Current BFS path: Boston->Providence

Current BFS path: Boston->New York

Current BFS path: Boston->Providence->New York

Current BFS path: Boston->New York->Chicago

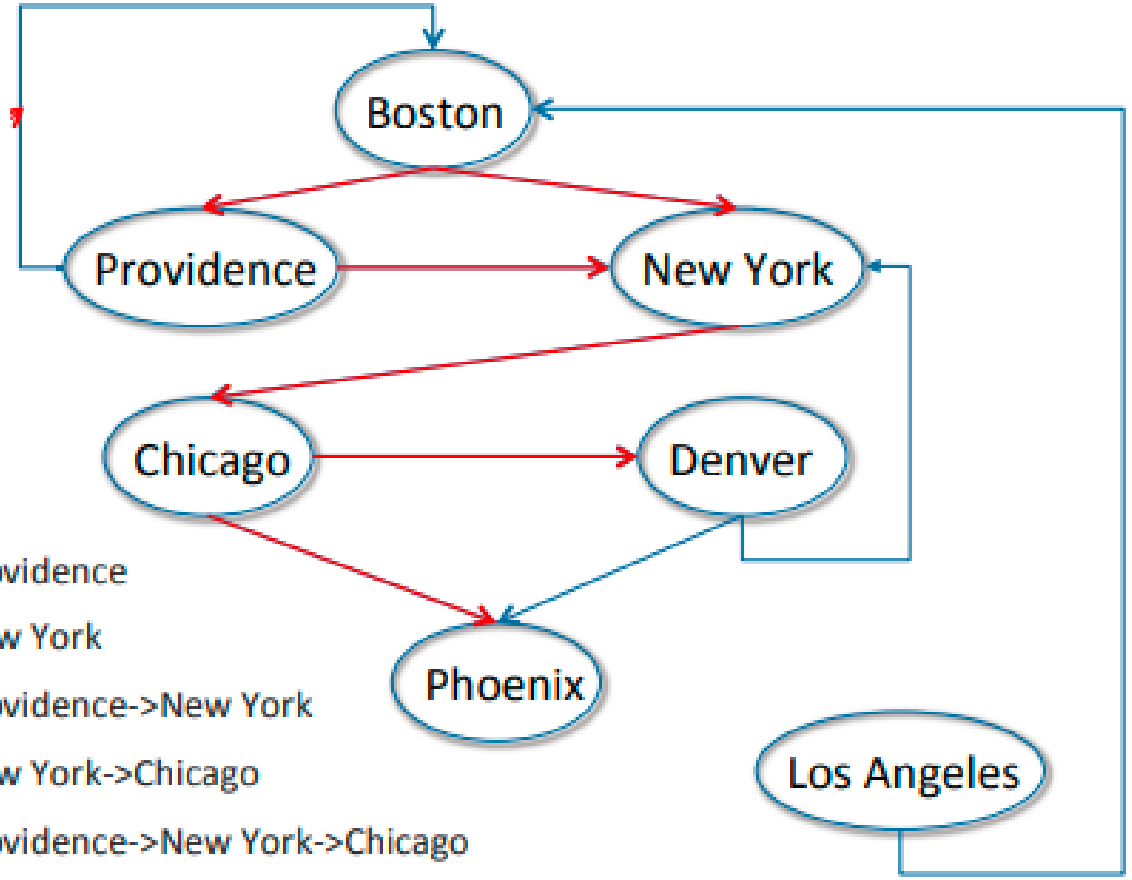
Current BFS path: Boston->Providence->New York->Chicago

Current BFS path: Boston->New York->Chicago->Denver

Current BFS path: Boston->New York->Chicago->Phoenix

Shortest path from Boston to Phoenix is Boston->New York->Chicago->Phoenix

Ejemplo – Breadth-First Search – Boston a Phoenix



Current BFS path: Boston

Current BFS path: Boston->Providence

Current BFS path: Boston->New York

Current BFS path: Boston->Providence->New York

Current BFS path: Boston->New York->Chicago

Current BFS path: Boston->Providence->New York->Chicago

Current BFS path: Boston->New York->Chicago->Denver

Current BFS path: Boston->New York->Chicago->Phoenix

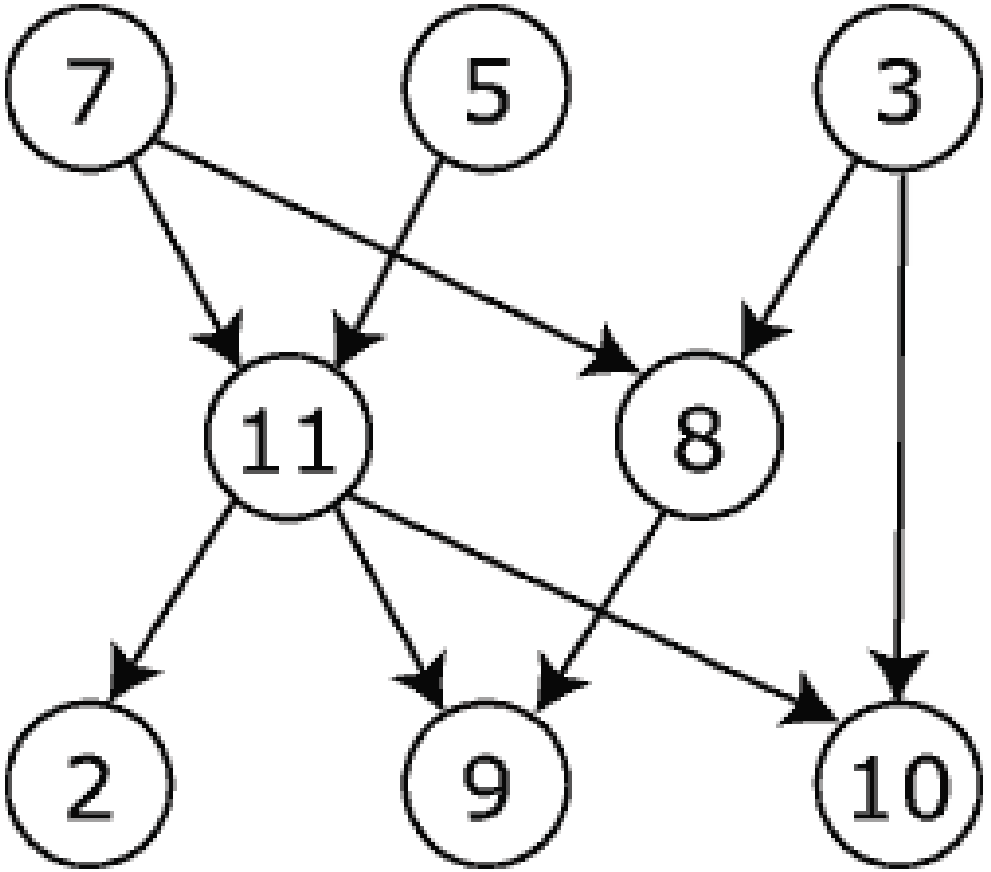
Shortest path from Boston to Phoenix is Boston->New York->Chicago->Phoenix

Codigo:

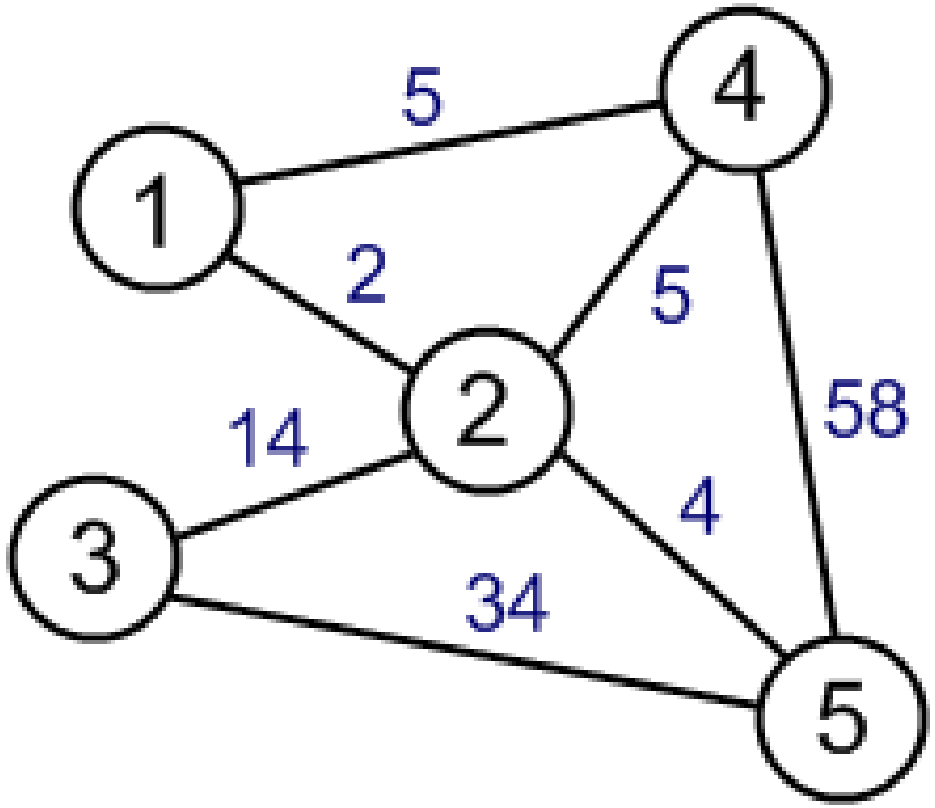
https://colab.research.google.com/drive/1dywndmekYWE80AJoOLgTk8F1alGj09Gx?usp=sharing

https://www.youtube.com/watch?v=1n5XPFcvxds
https://www.youtube.com/watch?v=cNAkUZaiDo4


How to draw adjacency matrices and lists???


7
5
3
11
8
2
9
10


How to draw adjacency matrices and lists???


1
2
3
4
5
5
2
14
5
58
4
34