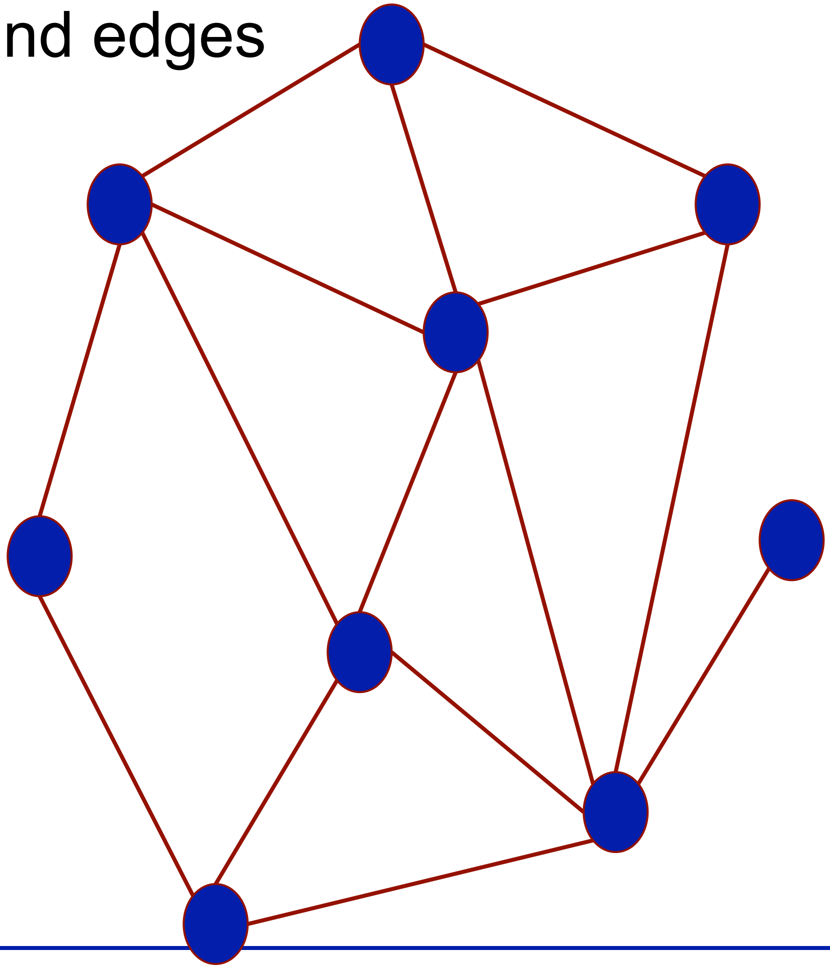
CS200: Graphs

Prichard Ch. 14

Rosen Ch. 10

Graphs

A collection of nodes and edges

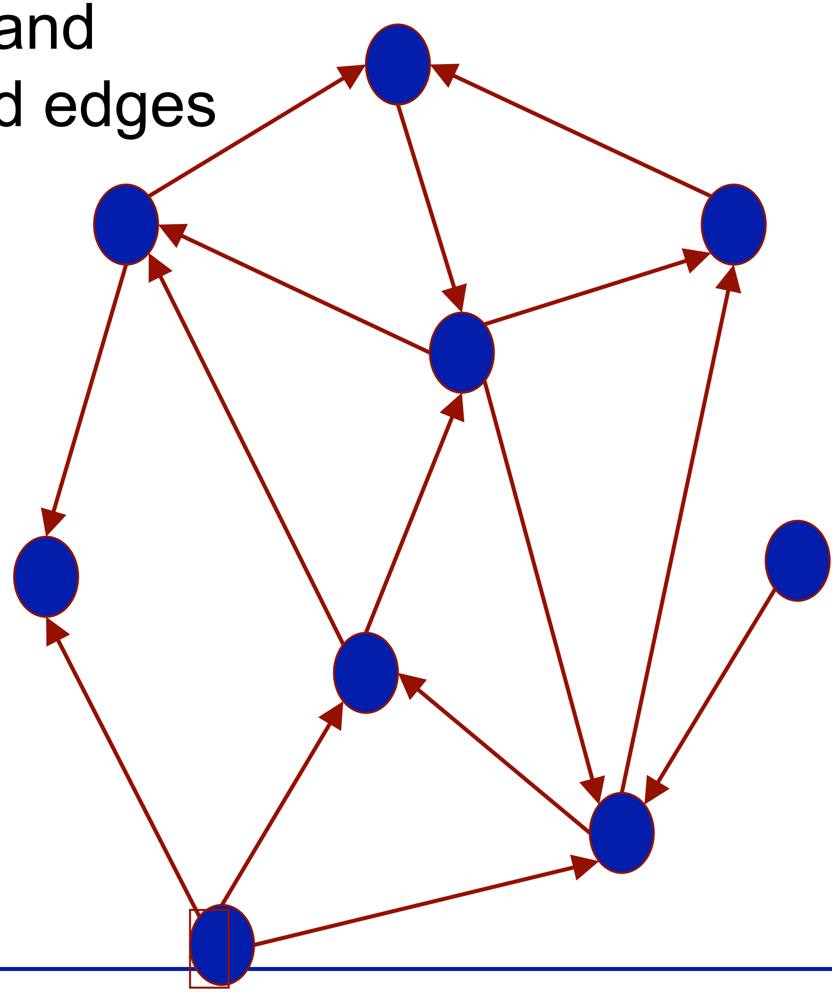


What can this represent?

- A computer network
- Abstraction of a map
- Social network

Directed Graphs

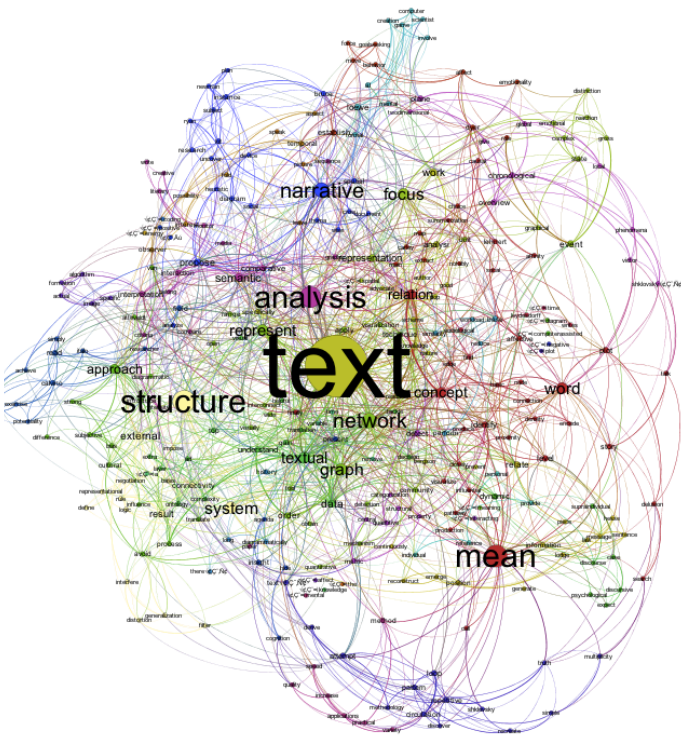
A collection of nodes and directed edges



Sometimes we want to represent directionality:

- Unidirectional network connections
- One way streets
- The web

Graphs/Networks Around Us



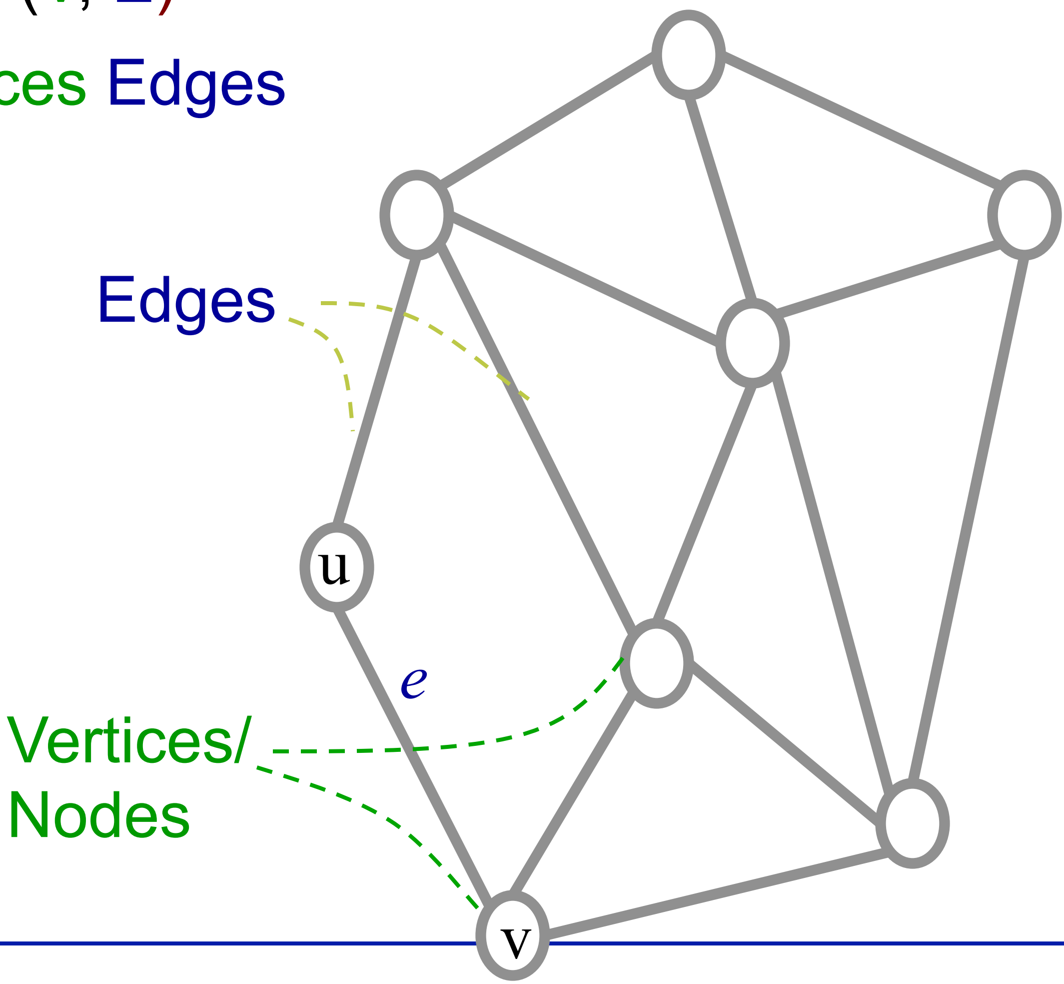
http://noduslabs.com/wp-content/uploads/2011/12/
figure-5-meaning-circulation.png

http://lin-ear-th-inking.blogspot.com/2010/12/
visualizing-geodetic-information-with.html

Graph Terminology

G=(V, E)

Vertices Edges

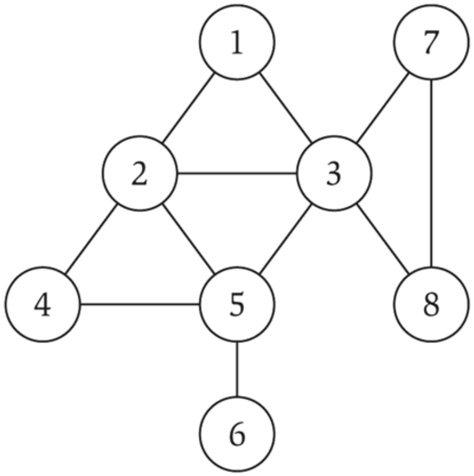


Two vertices (or nodes) are adjacent if they are connected by an edge.
An edge is incident on two vertices, an edge e can be represented by two vertices (u,v)
Degree of a vertex or node: number of edges incident on it

Graph terminology:
14.1 in Prichard,
10.1 in Rosen

Undirected Graphs

- Undirected graph. G = (V, E)
  - V = set of nodes.
  - E = set of edges between pairs of nodes.
  - Captures pairwise relationship between objects.
  - Graph size parameters: n = |V|, m = |E|.



V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6 }

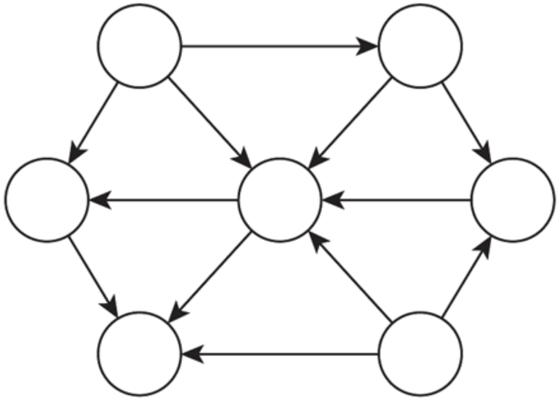
n = 8

m = 11

Directed Graphs

CS200

- Directed graph. G = (V, E)
  - Edge (u, v) goes from node u to node v.



- Example. Web graph - hyperlink points from one web page to another.
  - Modern web search engines exploit hyperlink structure to rank web pages by importance (pageRank).

Graph definitions

CS200

Graph **G = (V, E)** , V: set of **node**s or vertices,

E: set of **edge**s (pairs of nodes).

In an **undirected** graph, edges are unordered pairs (sets) of nodes. In a **directed** graph edges are ordered pairs (2-tuples) of nodes.

Path: sequence of nodes $(v_0..v_n)$ s.t. $\forall i: (v_i, v_{i+1})$ is an edge. **Path length**: number of edges in the path, or sum of weights. **Simple path:** all nodes distinct.

Cycle: path with first and last node equal. **Acyclic graph**: graph without cycles. **DAG**: directed acyclic graph.

Two nodes are **adjacent** if there is an edge between them. In a **complete graph** all nodes in the graph are adjacent.

More definitions

An undirected graph is **connected** if for all nodes v_i and v_j there is a path from v_i to v_j .

G'(V', E') is a **sub-graph** of G(V,E) if $V' \subseteq V$ and $E' \subseteq E$

The sub-graph of G **induced** by V' has all the edges $(u,v) \in E$ such that $u \in V'$ and $v \in V'$.

In a **weighted graph** the edges have a weight (cost, length,..) associated with them.

Question

- A Tree is a subtype (special type) of Graph.

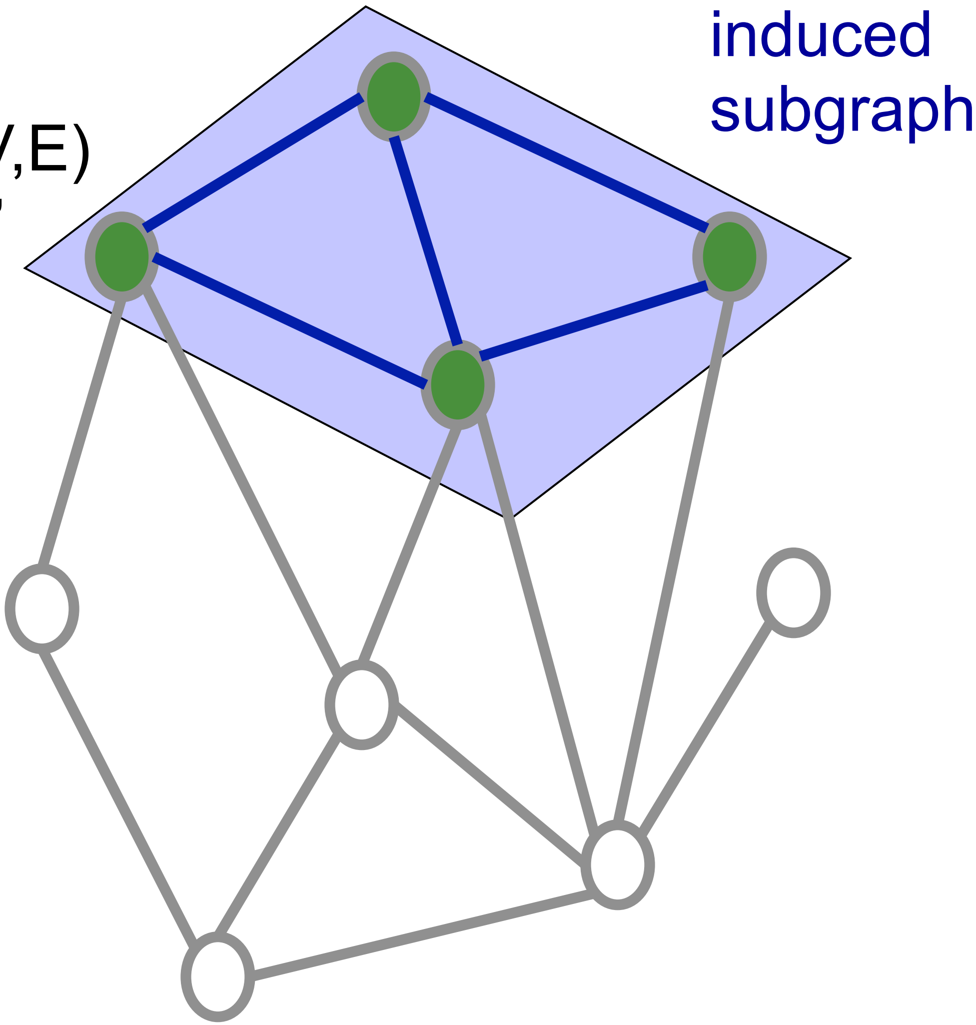
A. True

B. False

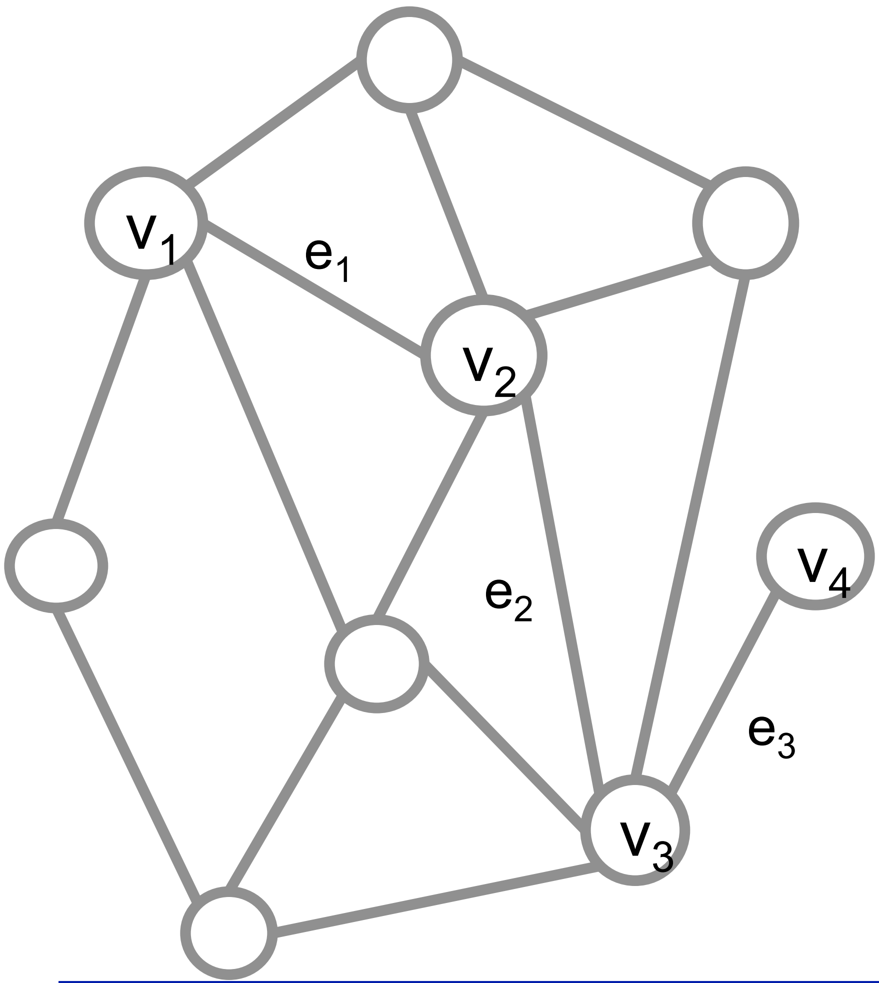
Graph Terminology

A subgraph of a graph G = (V,E) is a graph (V’,E’) such that V’ is a subset of V and, E’ is a subset of E

The sub-graph of G **induced** by V’ has all the edges (u,v) ∈ E such that u ∈ V’ and v ∈ V’.

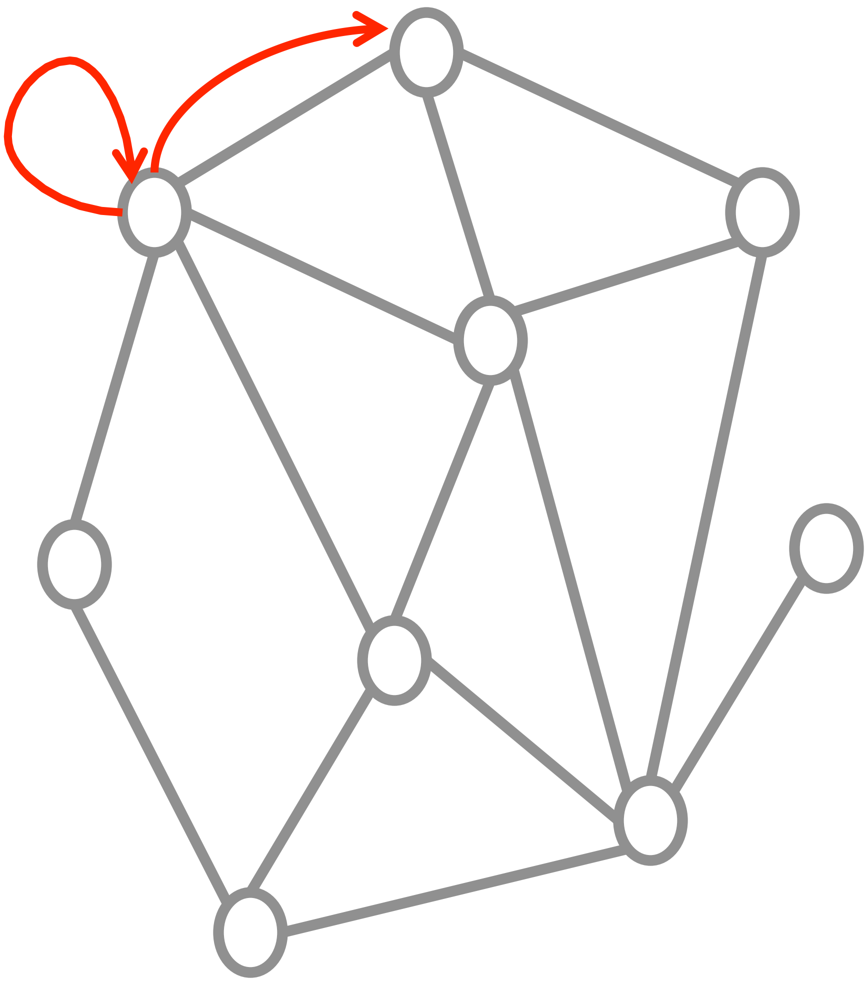


Paths



- **Path:** a sequence of edges, e.g. $((v_1,v_2), (v_2,v_3), (v_3,v_4))$ s.t. the first node in the next edge is the second node in the previous edge.
- A simple path passes **through** a vertex only once.
- (e_1, e_2, e_3) is a simple path of length 3 from v_1 to v_4
- A path can be represented by a sequence of vertices, here (v_1,v_2,v_3,v_4)

Graph Terminology



Self loop (loop): an edge that connects a vertex to itself

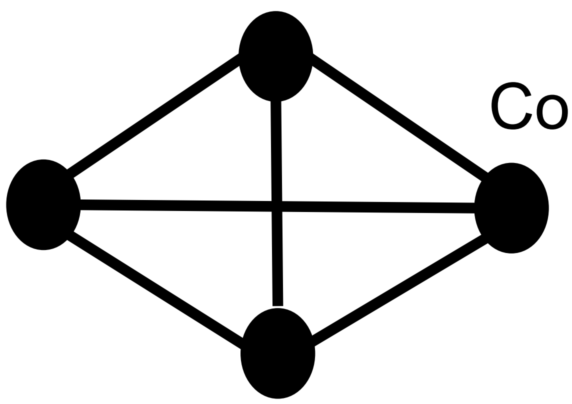
(Simple) Graph: no self loops and no two edges connect the same vertices (E is a set, so no multiples). We are mostly thinking about these.

Multigraph: may have multiple edges connecting the same vertices (not a graph: E is a set in graph)

Pseudograph: multigraph with self-loops

Complete Graphs

- Simple graph that contains exactly one edge between each pair of distinct vertices.



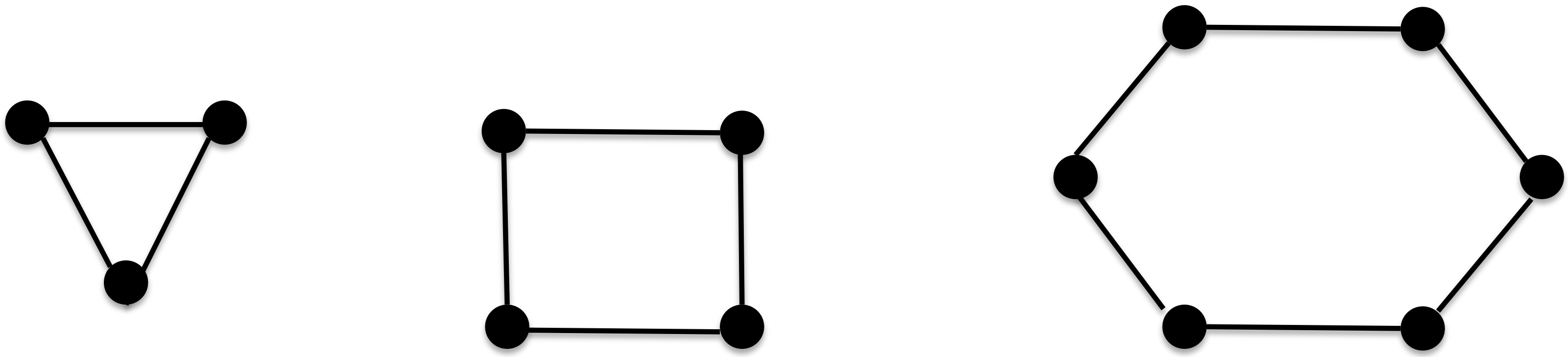
Complete Graph

Cycles

The cycle C_n, $n \geq 3$, consists of n vertices

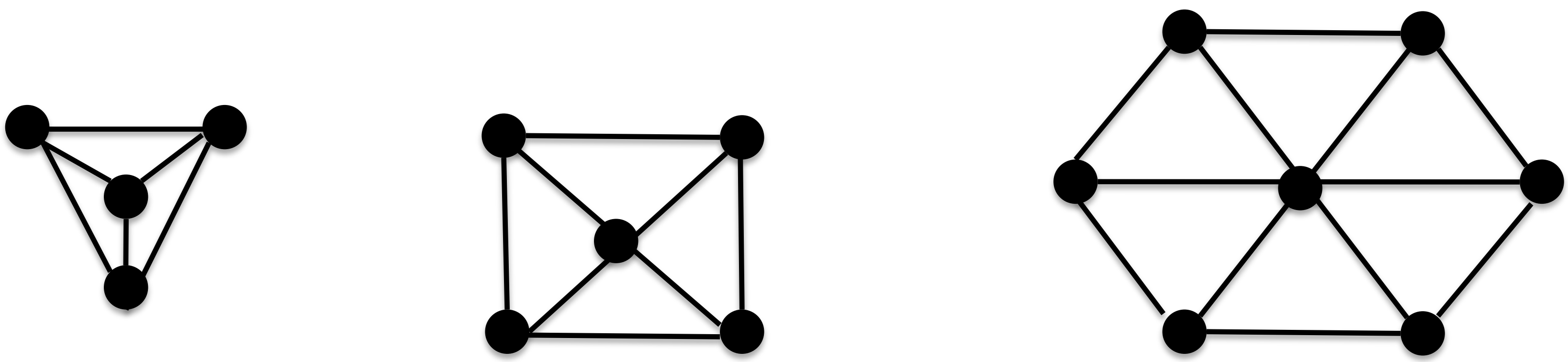
$v_1, v_2, \ldots, v_n$ and n edges

$\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$.



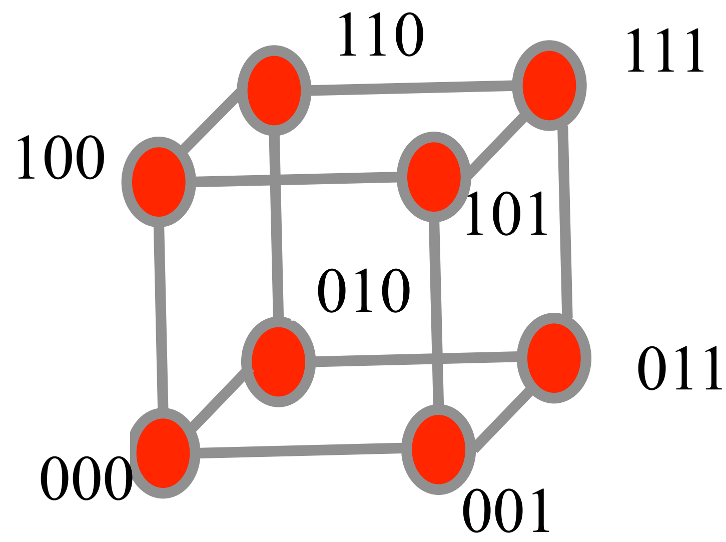
Wheels

- We obtain the wheel W_n when we add an additional vertex to the cycle C_n, for $n \leq 3$, and connect this new vertex to each of the n vertices in C_n, by new edges



n-Cube (*n*-dimensional hypercube)

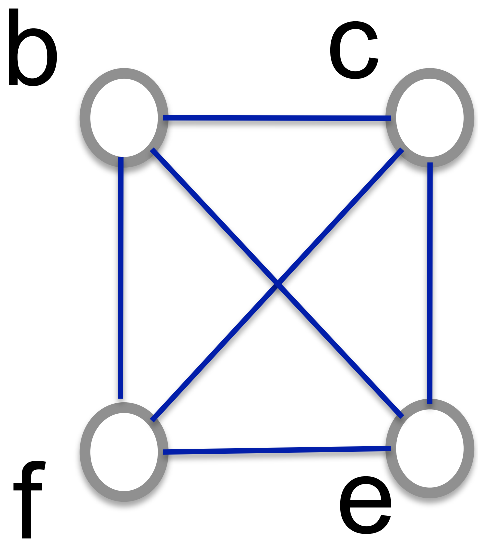
Hypercube



Question

Which describes this graph?

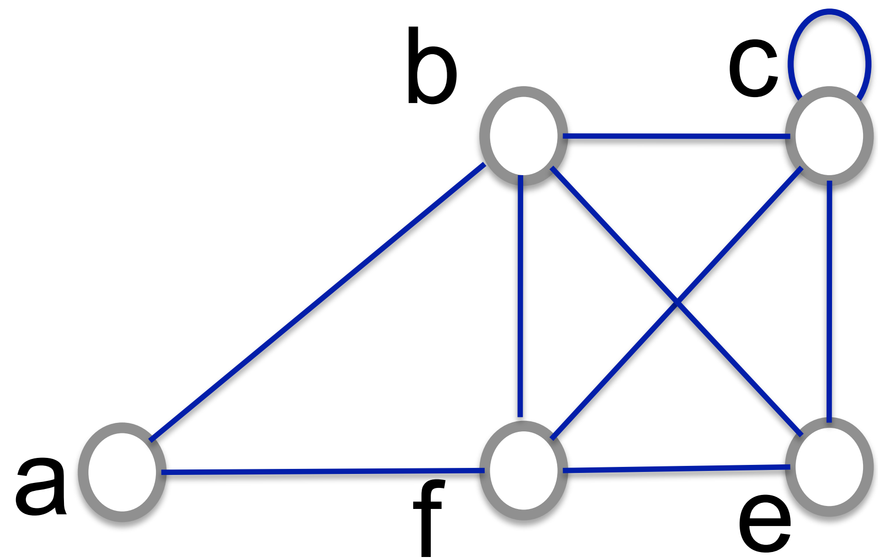
A. Simple
B. Pseudograph
C. Cycle
D. Complete



Question

Which describes this graph?

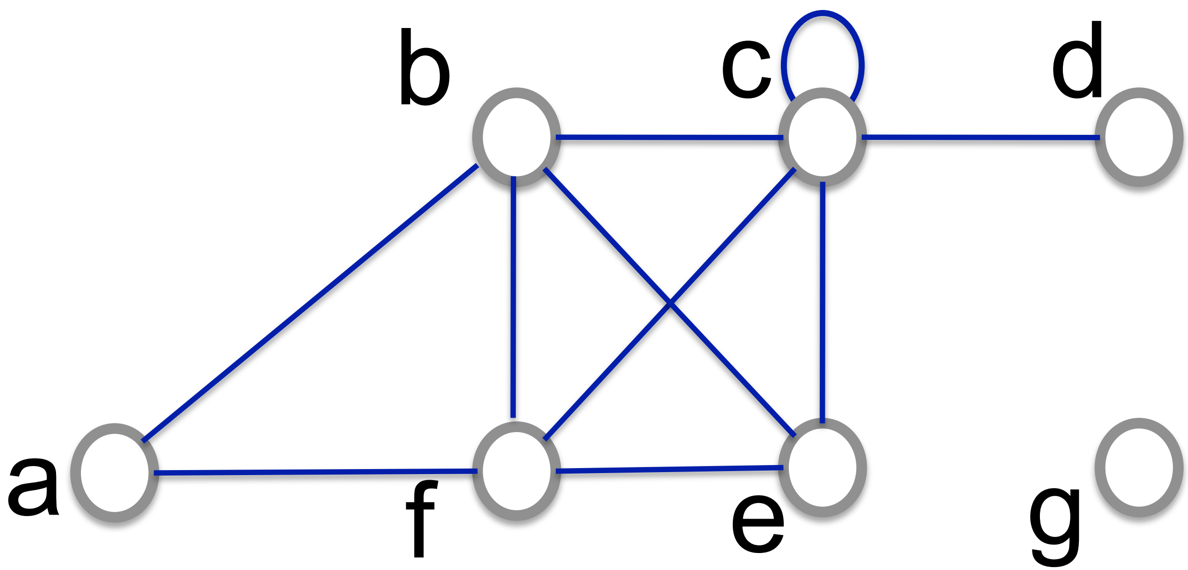
A. Simple
B. Pseudograph
C. Cycle
D. Complete



The degree of a vertex

- The degree of a vertex in an undirected graph
  - the number of edges incident with it
  - except that a loop at a vertex contributes twice to the degree of that vertex.

Example



d is "pendant"

g is "isolated"

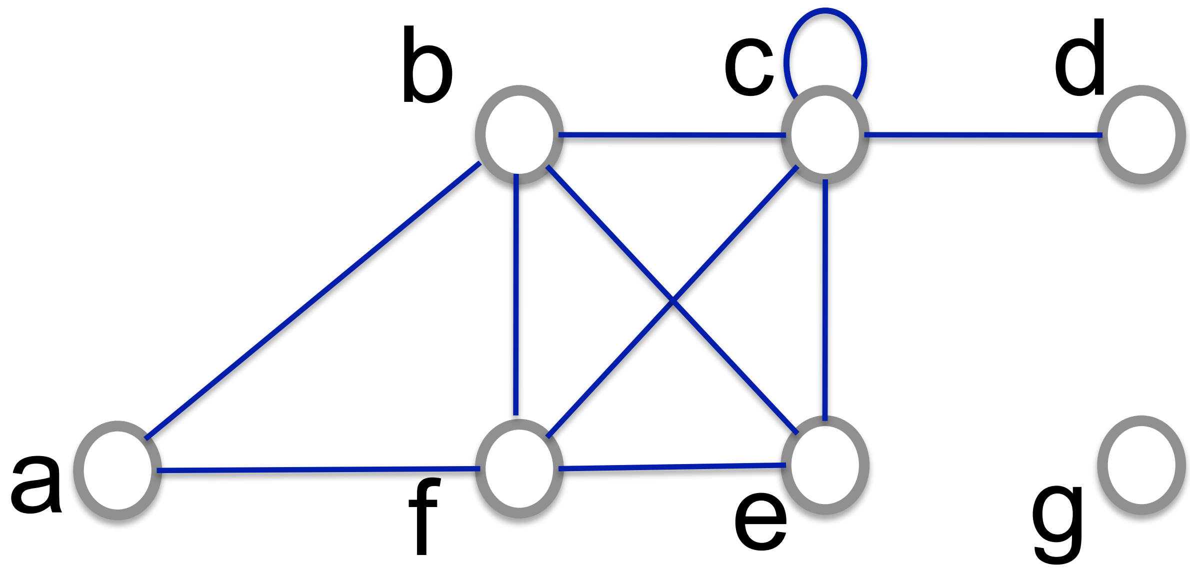
deg(a) = 2
deg(b) = deg(f) = 4

deg(d) = 1
deg(e) = 3
deg(g) = 0

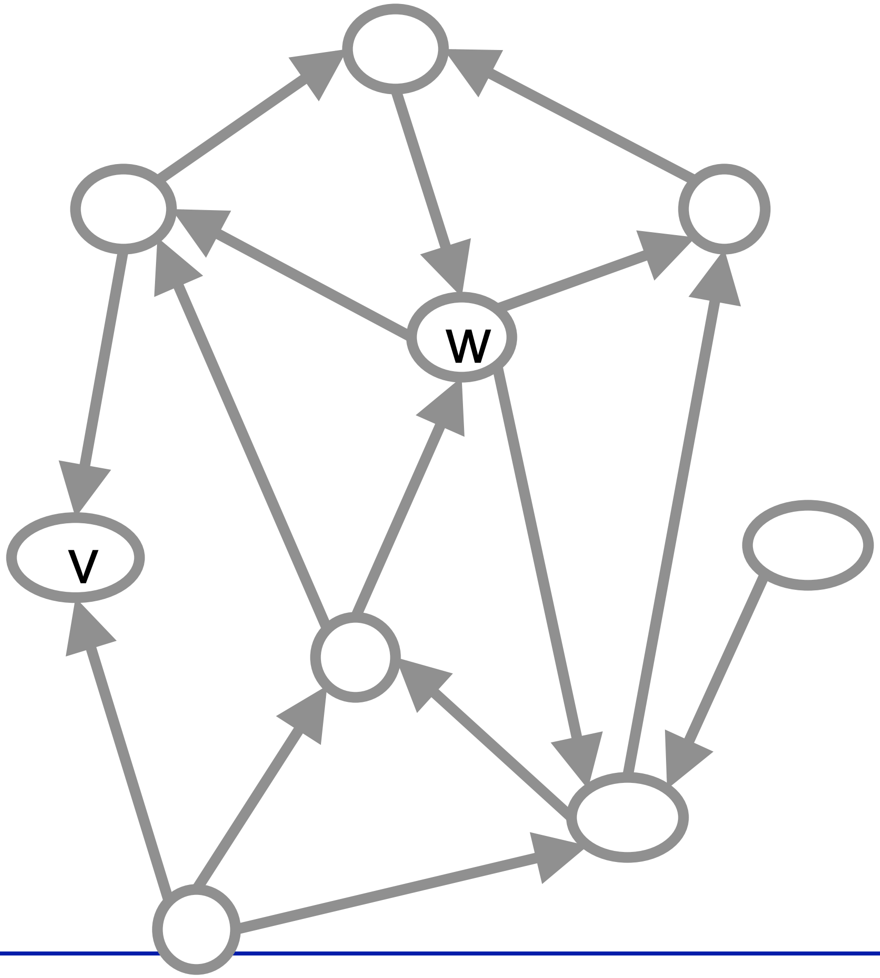
Clicker Q

What is the degree of c?

A. 4
B. 5
C. 6



Directed Graphs



Indegree: number of incoming edges

Outdegree: number of outgoing edges

Some Graph Theorems

- *Handshaking*: Let *G=(V,E)* be an undirected graph with *m* edges. Then

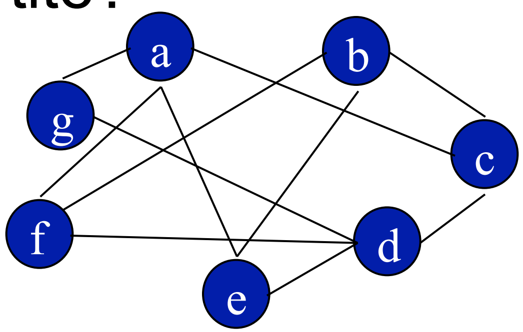
$$2m = \sum_{v \in V} \deg(v)$$

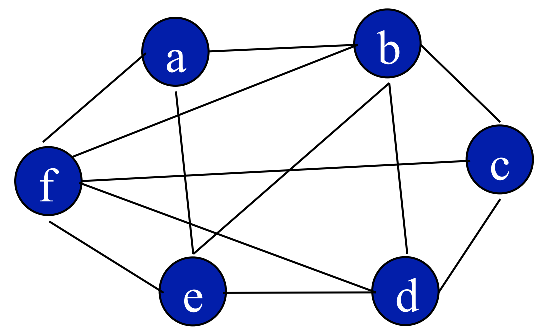
- An undirected graph has an even number of vertices of odd degree.
- Let *G=(V,E)* be a directed graph. Then

$$\sum_{v \in V} \deg^{-}(v) = \sum_{v \in V} \deg^{+}(v) = |E|$$

Bipartite Graphs

- A simple graph on which the vertex set V can be partitioned into two disjoint sets V1 and V2 such that every edge connects a vertex in V1 to one in V2.
- Bipartite?

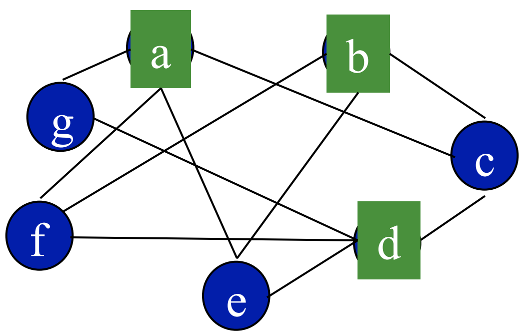


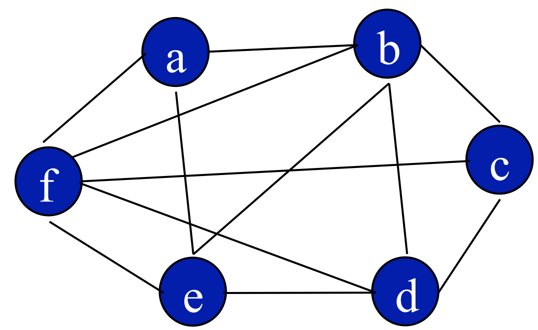


- Theorem: A simple graph is bipartite iff it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Bipartite Graphs

- Assign colors





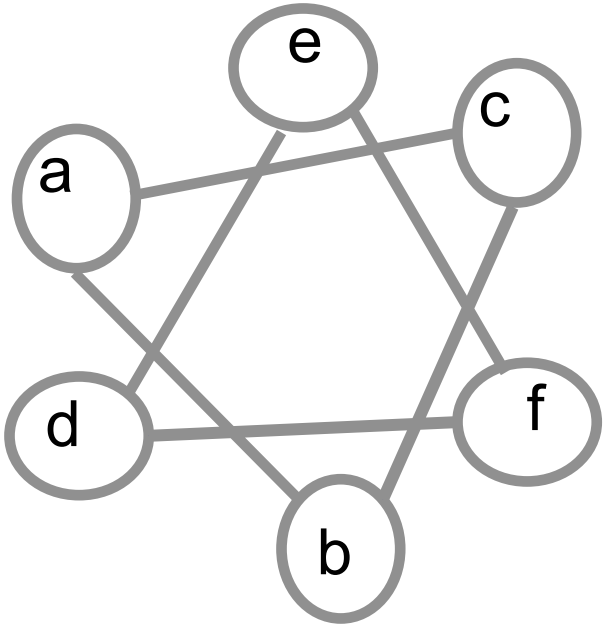
Theorem:
A graph G is bipartite iff
it contains no odd cycle

Question

- Is this graph bipartite?

A. Yes

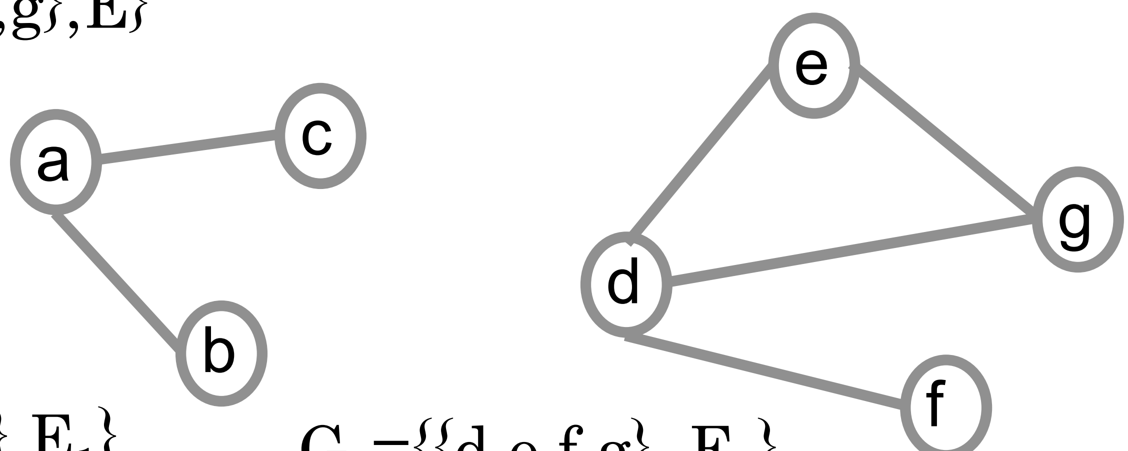
B. No



Connected Components

- An undirected graph is called **connected** if there is a path between every pair of vertices of the graph.
- A **connected component** of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G.

$G=\{\{a,b,c,d,e,f,g\},E\}$



$G_1=\{\{a,b,c\},E_1\}$ $G_2=\{\{d,e,f,g\}, E_2\}$

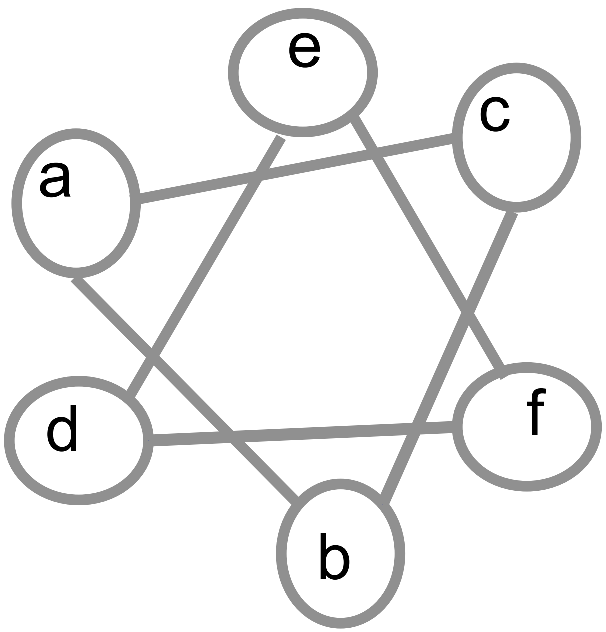
Question

- How many connected components does it have?

A. 0

B. 1

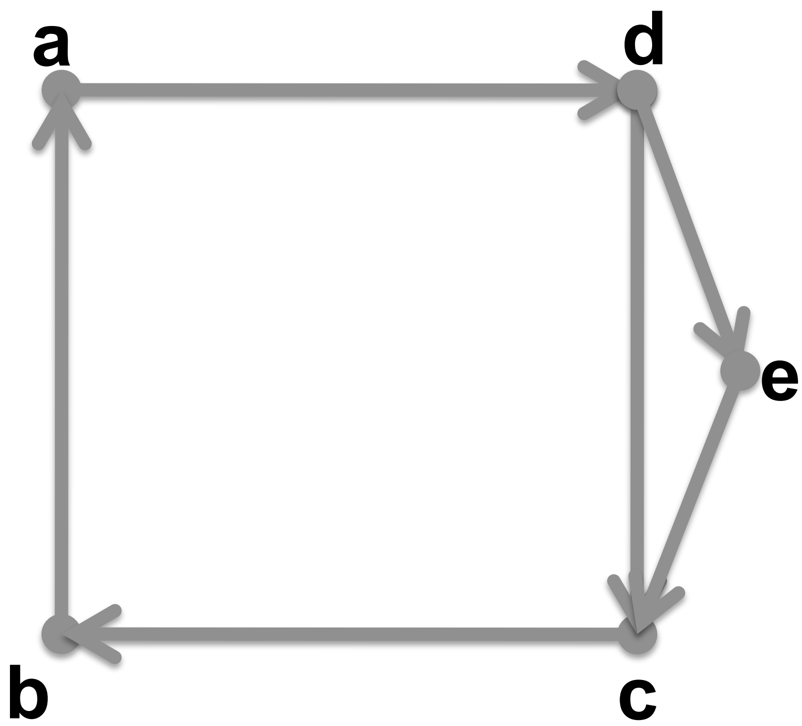
C. 2



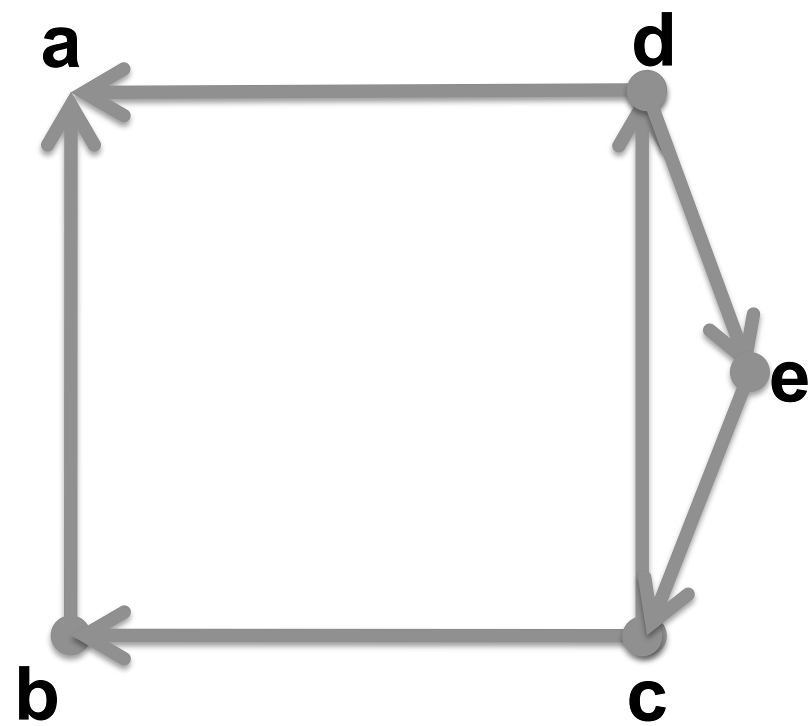
Connectedness in Directed Graphs

- A directed graph is ***strongly connected*** if there is a path from *a* to *b* and from *b* to *a* for all vertices *a* and *b* in the graph.
- A **directed** graph is ***weakly connected*** if there is a path between every two vertices in the underlying **undirected** graph.

A/B strongly/weakly connected?



Graph A



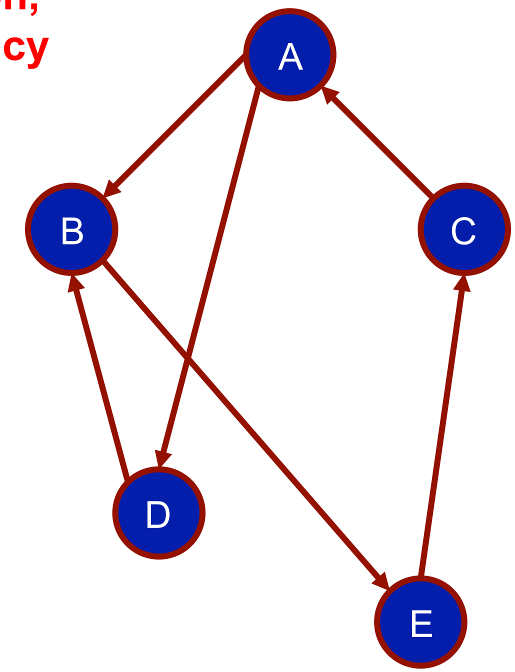
Graph B

Graph Data Structures -
Adjacency Matrix

- Vertices
  - labels mapped to row and column indices
  - one vertex mapped to one index
  - Values:
    - boolean to indicate presence/absence of edge in (un)directed graph
    - int to indicate value of weighted edge (0: no edge)
- Edges
  - square matrix of edges
    - size = number of vertices
    - edge: two (vertex) indices
- useful for dense graphs

Adjacency Matrix Example

For undirected graph, what would adjacency matrix look like?



Label	Index
A	0
B	1
C	2
D	3
E	4

mapping of vertex labels to array indices

In a weighted graph, cells would contain weights

	0	1	2	3	4
0	0	1	0	1	0
1	0	0	0	0	1
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0

Adjacency Matrix:
array of edges indexed by vertex number

Question

Is this an undirected graph?

A. Yes

B. No

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	0	1
3	0	1	0	1	0
4	0	1	1	0	0

Adjacency Matrix:

Question

Is this a simple graph?

A. Yes

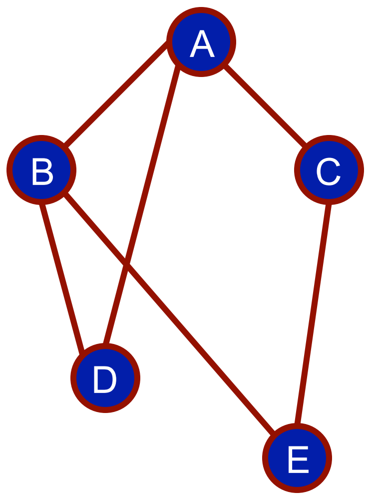
B. No

	0	1	2	3	4
0	0	1	1	0	0
1	1	0	0	1	1
2	1	0	0	0	1
3	0	1	0	1	0
4	0	1	1	0	0

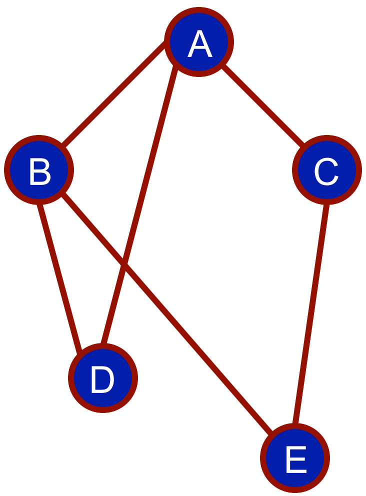
Adjacency Matrix:

Graph Data Structures -
Adjacency List

- Vertices
  - mapped to list of adjacencies
  - adjacency: edge
- Edges: lists of adjacencies
  - linked-list of out-going edges per vertex
- useful for sparse graphs



Adjacency List: Undirected Graph



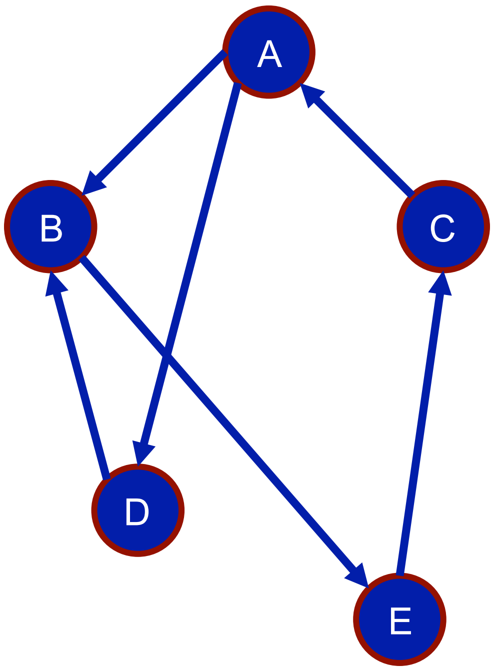
ArrayList

ArrayLists

Index	Label	
0	A	B C D
1	B	A D E
2	C	A E
3	D	A B
4	E	B C

mapping of vertex
labels to list of edges

Adjacency List: Directed Graph



ArrayList

ArrayLists

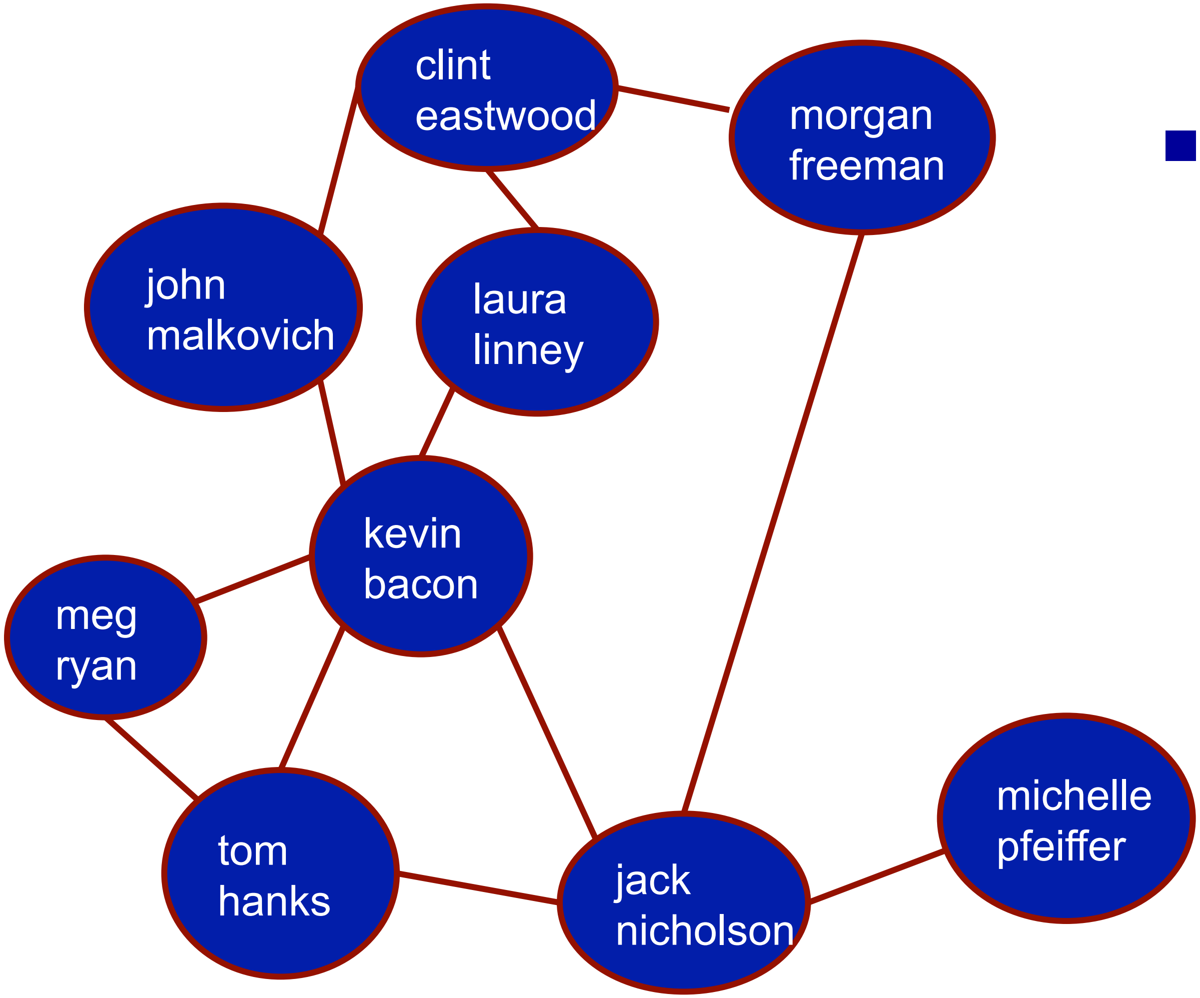
Index	Label	
0	A	B D
1	B	E
2	C	A
3	D	B
4	E	C

This representation is used in Graph recitation and assignment

Which Implementation Is Best?

- Which implementation best supports common Graph Operations:
  - Is there an edge between vertex i and vertex j?
  - Find all vertices adjacent to vertex j
- Which best uses space?

Six Degrees of Kevin Bacon



- Actor x has a Kevin Bacon Number of n if **the shortest path** between x and Kevin Bacon has length n

Graph made with the help of the oracle of Bacon: http://oracleofbacon.org/

Shortest Path Algorithms
(Dijkstra's Algorithm)

- Graph G(V,E) with positive weights ("distances")
- Compute shortest distances from source vertex s to ***every other vertex*** in the graph

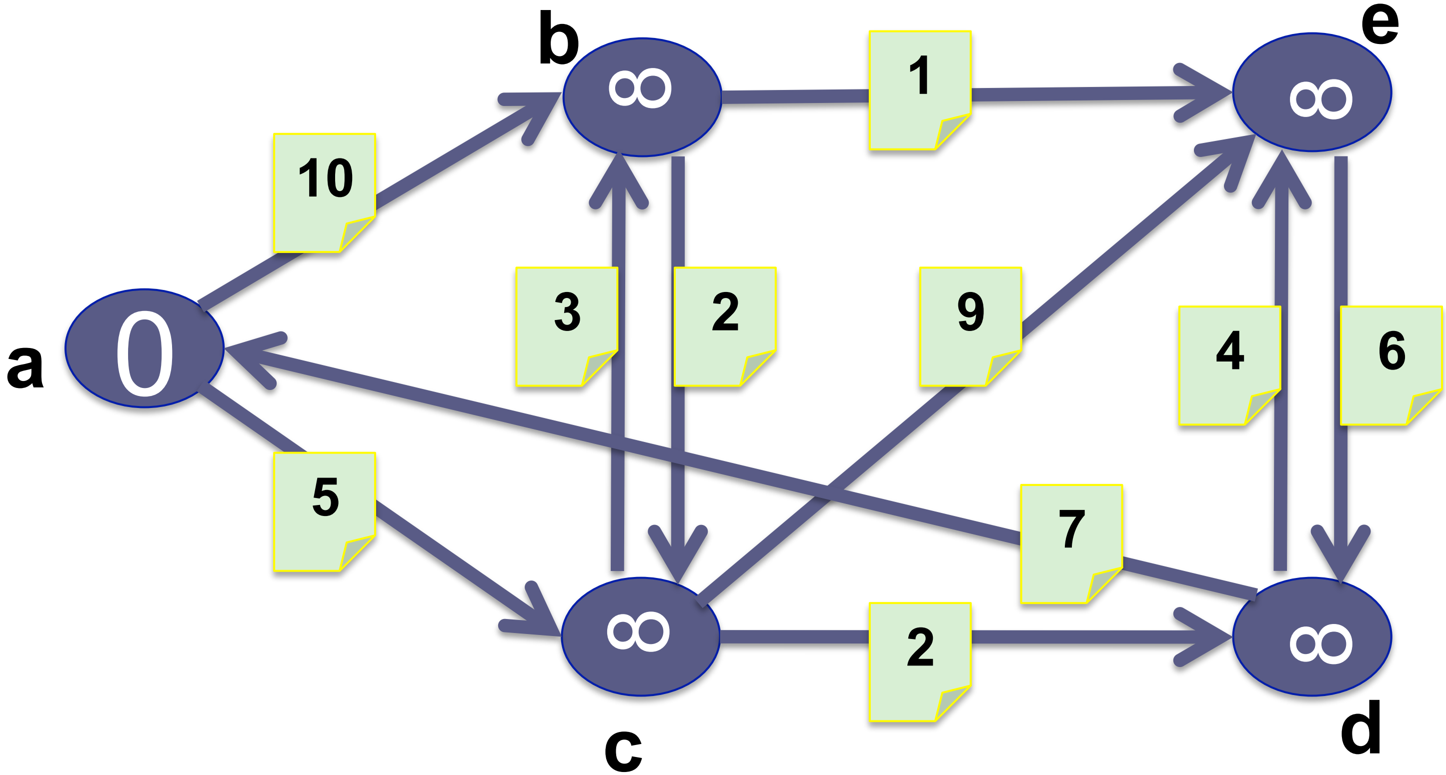
Shortest Path Algorithms
(Dijkstra's Algorithm)

- Algorithm
  - Stepwise create a minimal path sub tree
    initial: source
  - Maintain array d (minimum distance estimates)
    - Init: d[s]=0, d[v]=∞,   v∈V-s
    - ∞ means: yet unreachable
  - array of nodes not yet visited with shortest distance to already selected nodes
  - select minimum path distance node v, update neighbors

Shortest Path Algorithms

(Dijkstra's Algorithm): Initialize

a	b	c	d	e
0	∞	∞	∞	∞

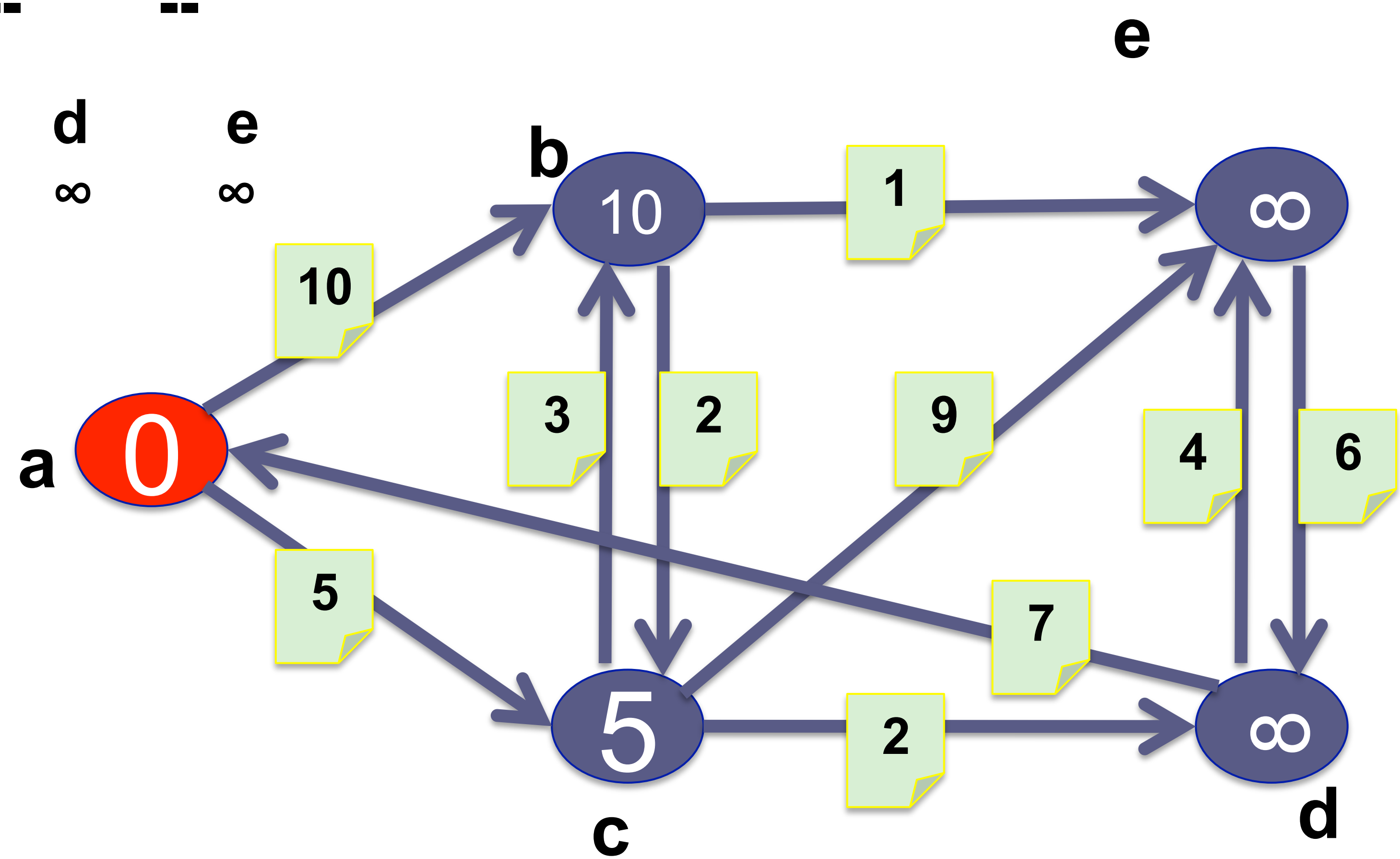


Shortest Path Algorithms

(Dijkstra's Algorithm): step 1

a	b	c	d	e
0	10/a	5/a	--	--

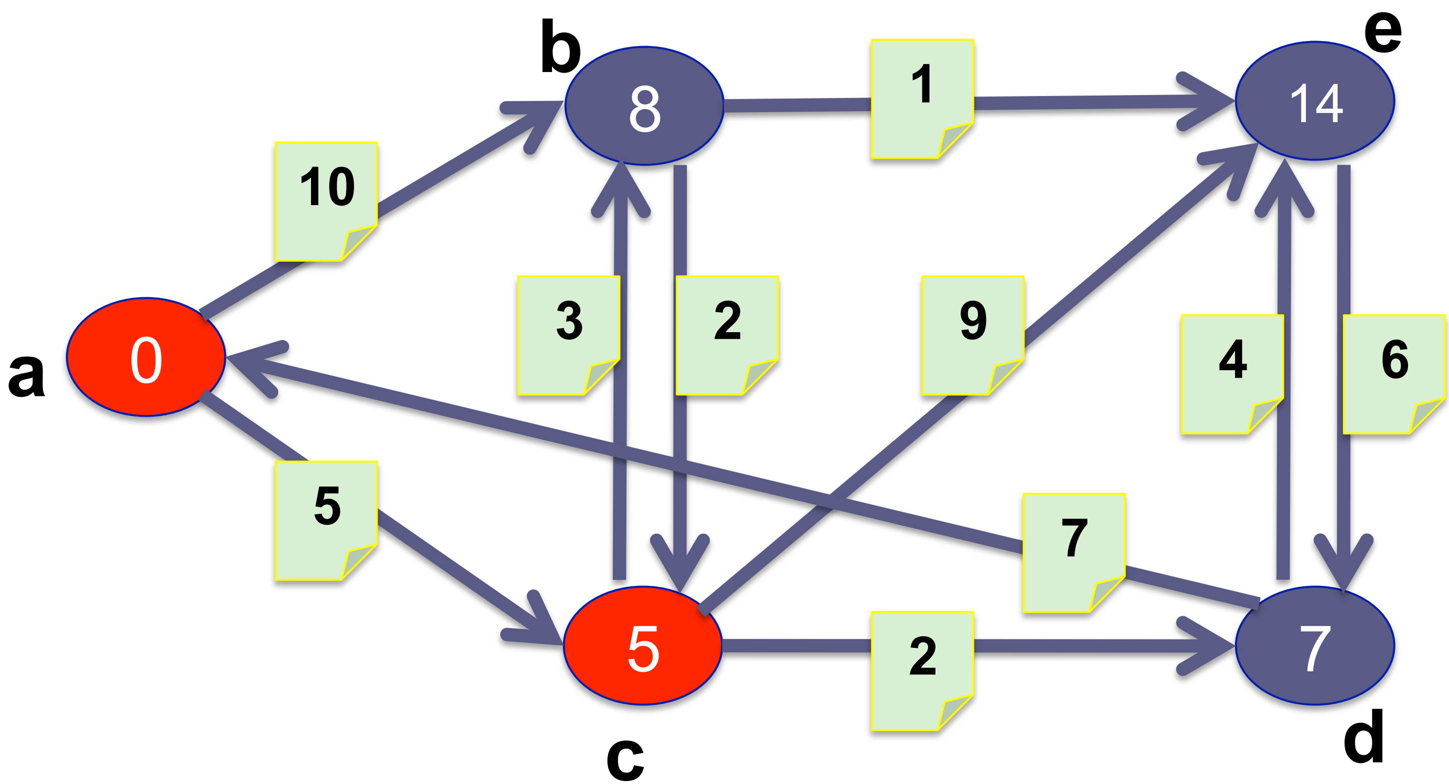
a	b	c	d	e
0	∞	∞	∞	∞



Shortest Path Algorithms

(Dijkstra's Algorithm): step 2

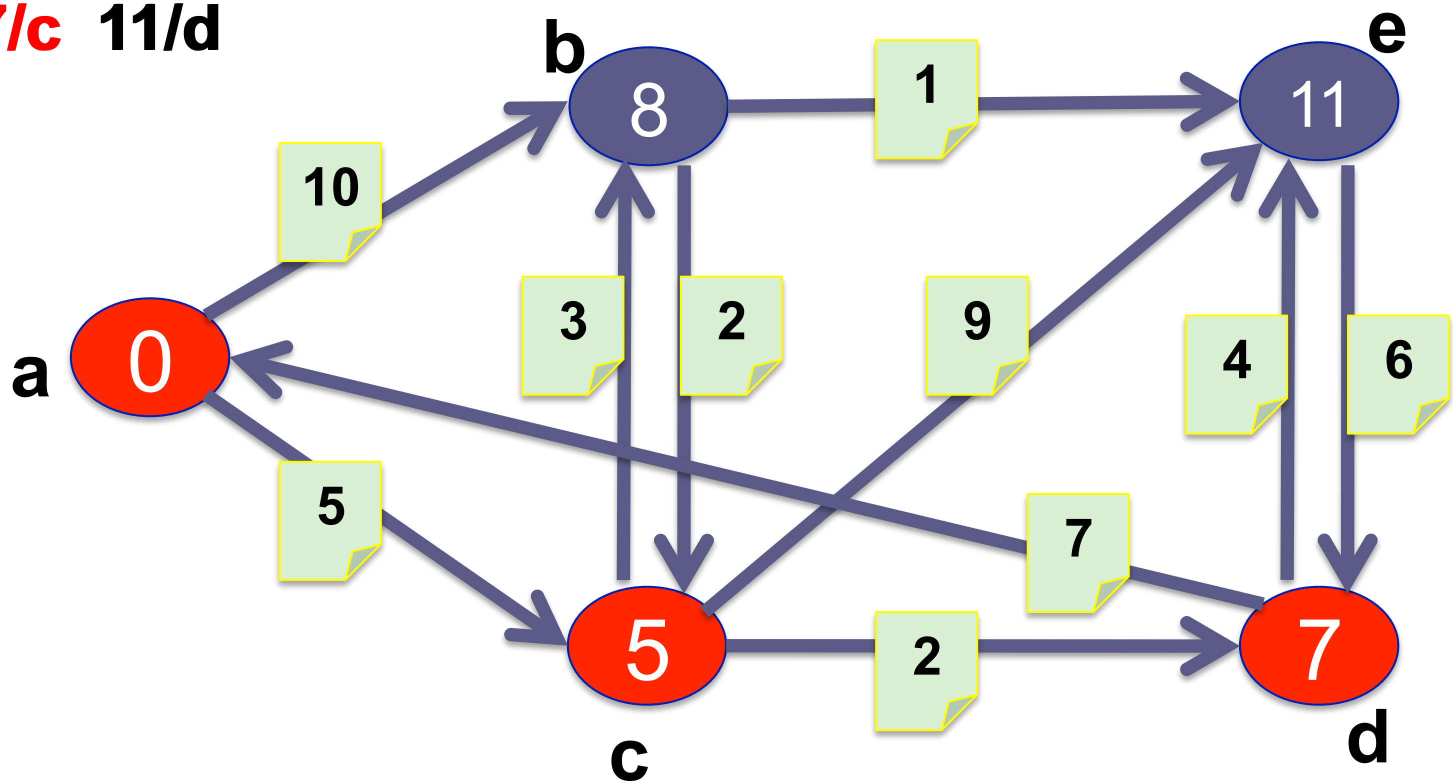
a	b	c	d	e
0	10/a	5/a	--	--
0	8/c	5/a	7/c	14/c



Shortest Path Algorithms

(Dijkstra's Algorithm): step 3

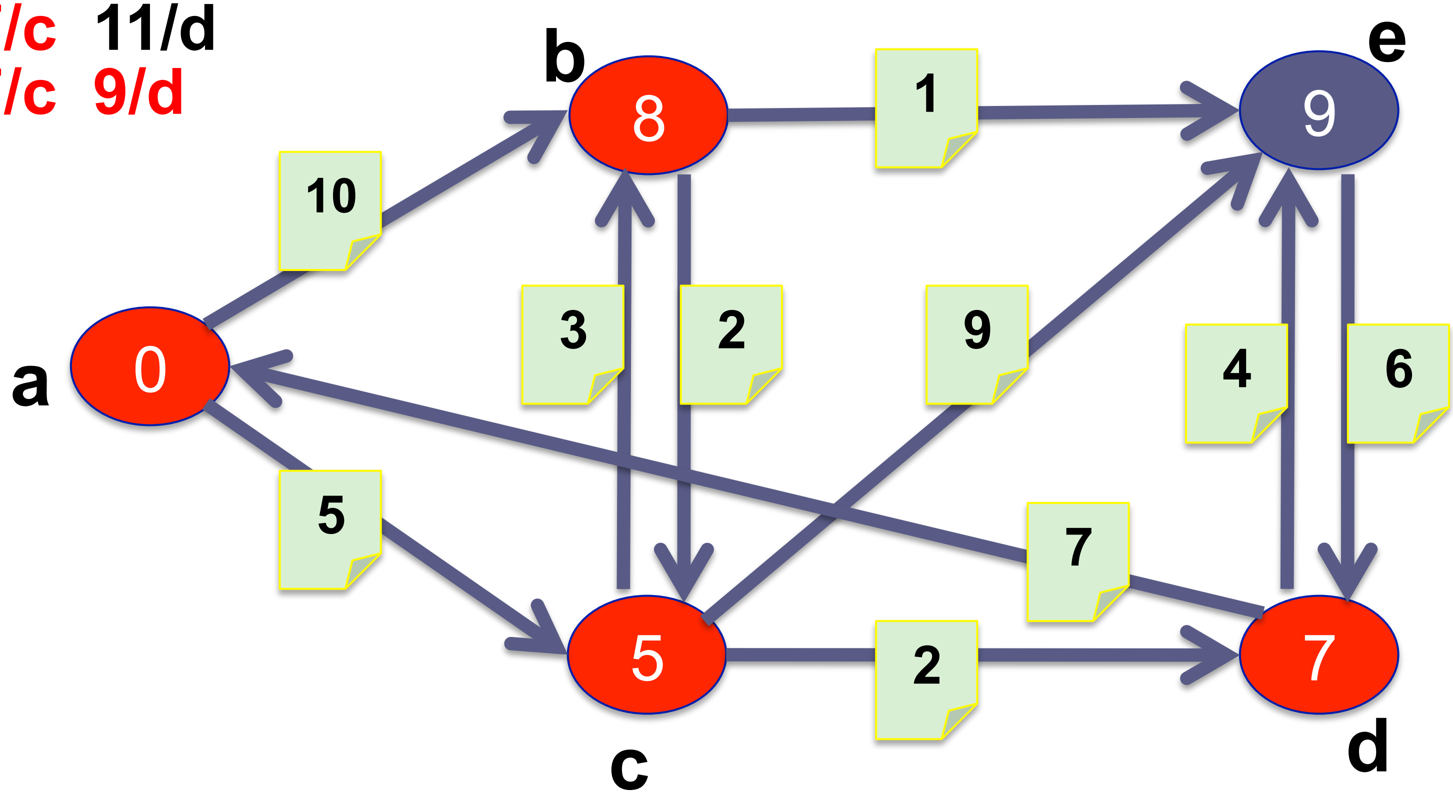
a	b	c	d	e
0	10/a	5/a	--	--
0	8/c	5/a	7/c	14/c
0	8/c	5/a	7/c	11/d



Shortest Path Algorithms

(Dijkstra's Algorithm): step 4

a	b	c	d	e
0	10/a	5/a	--	--
0	8/c	5/a	7/c	14/c
0	8/c	5/a	7/c	11/d
0	8/c	5/a	7/c	9/d



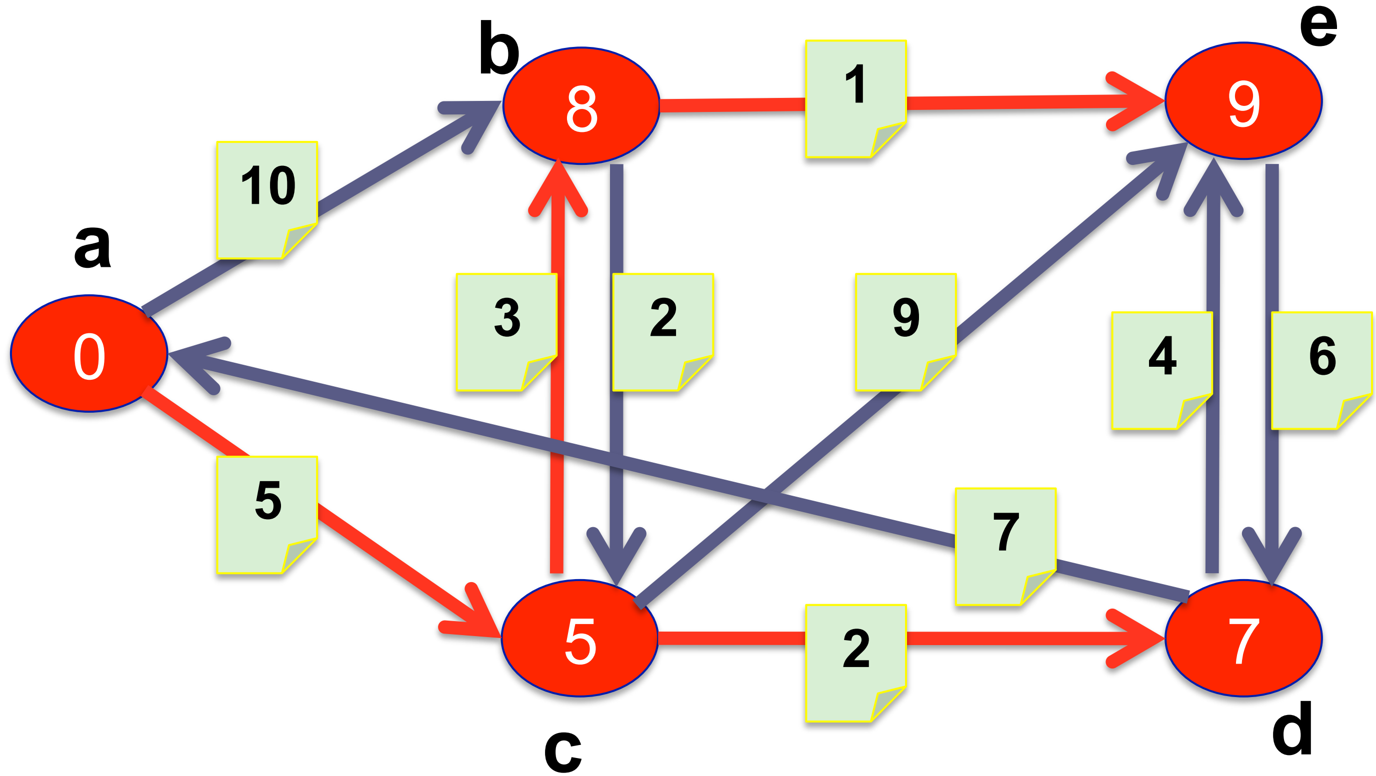
Shortest Path Algorithms

(Dijkstra's Algorithm): Done

a	b	c	d	e
0	8/c	5/a	7/c	9/b

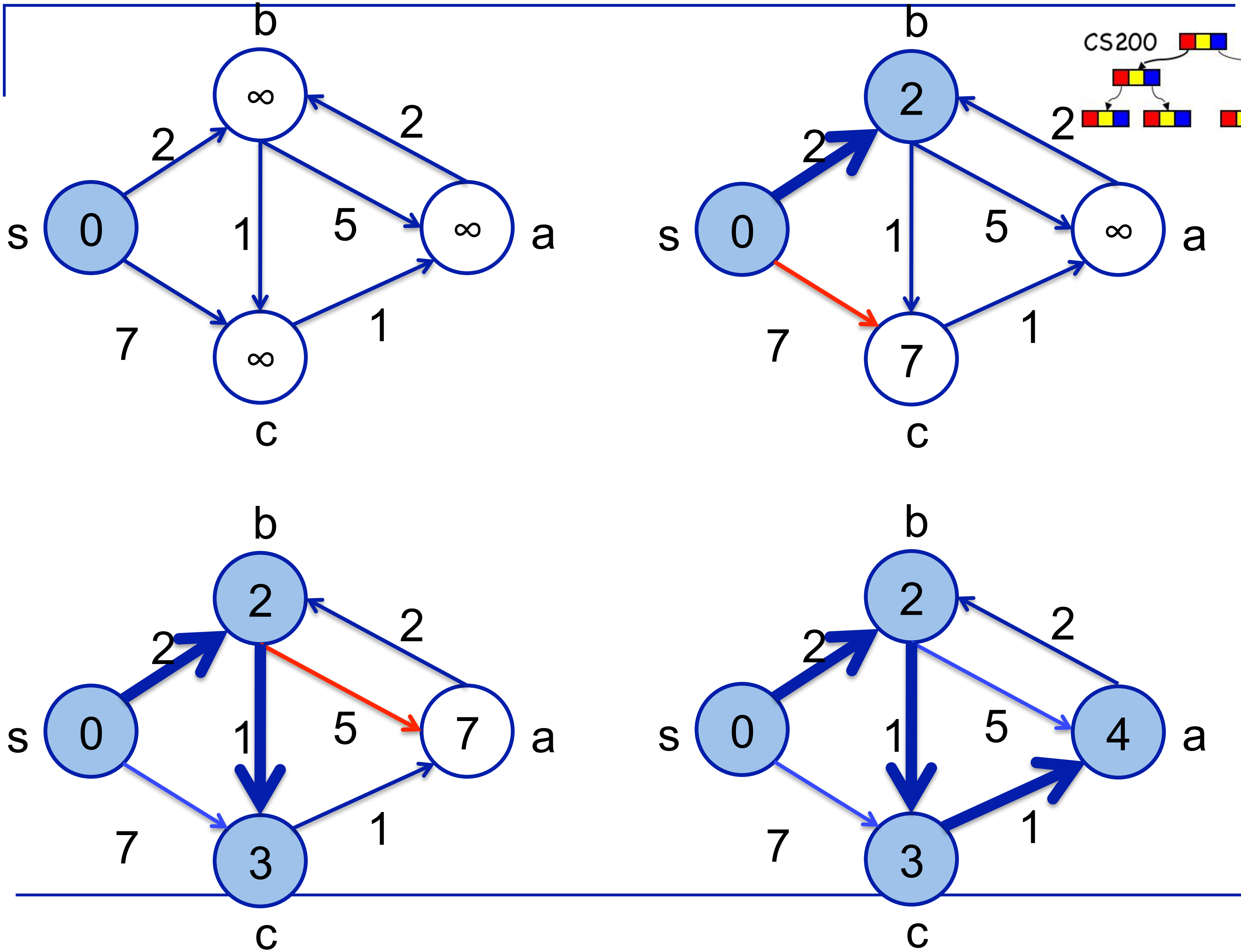
a
0   a is the source

b
8/c   min dist to b: 8
predecessor: c



Dijkstra's Algorithm

```
Dijkstra(G: graph with vertices v_0…v_{n-1} and weights w[u][v])
// computes shortest distance of vertex 0 to every other vertex
   create a set vertexSet that contains only vertex 0
   d[0] = 0
   for (v = 1 through n-1)
       d[v] =  infinity
   for (step = 2 through n)
       find the smallest d[v] such that v is not in vertexSet
       add v to vertexSet
       for (all vertices u not in vertexSet)
             if (d[u] > d[v] + w[v][u])
                   d[u] = d[v] + w[v][u]
```


b
∞
2
2
s
0
1
5
∞
a
7
∞
1
c
b
2
2
2
s
0
1
5
∞
a
7
7
1
c
b
2
2
2
s
0
1
5
7
a
7
3
1
c
b
2
2
2
s
0
1
5
4
a
7
3
1
c

Recap: Priority Queue

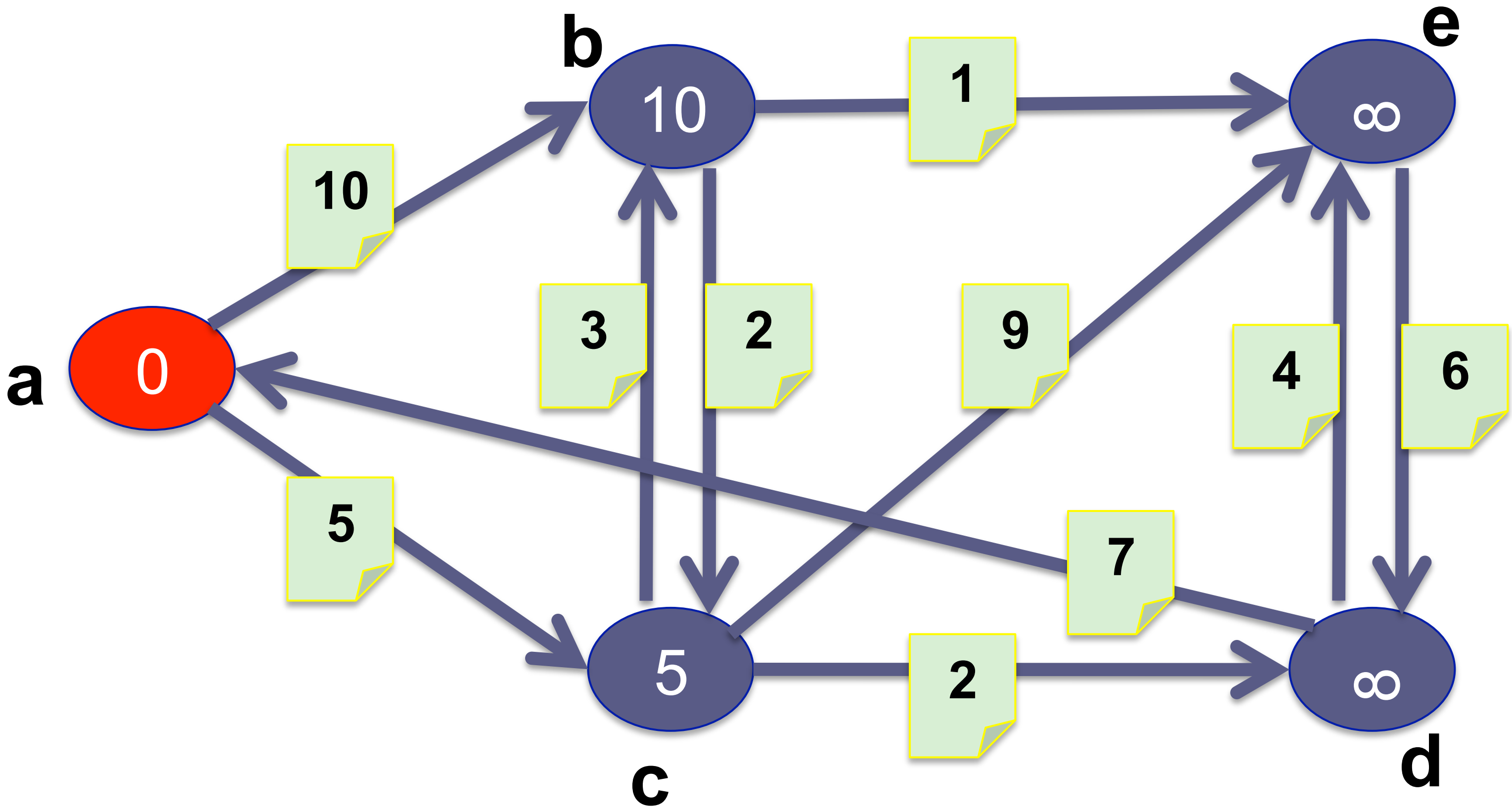
- A Priority Queue is a data structure that keeps a set of items (P,V), consisting of a Priority P and a Value V, in sorted order of priority.
- Possible operations:

  insert($X(P_x,V_x)$)

  delete($X(P_x,V_x)$)

We have studied a clever data structure for priority queues: heaps

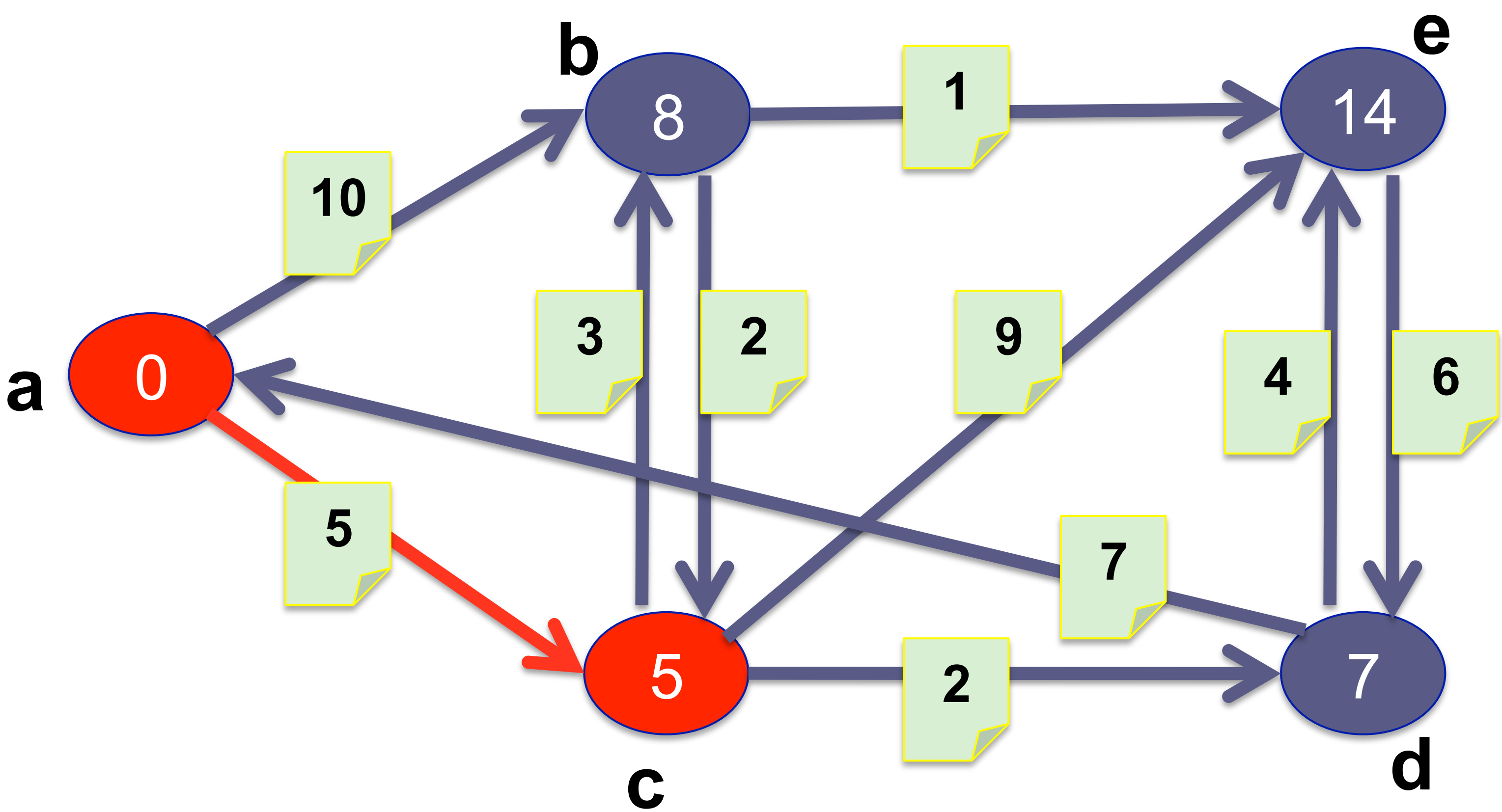
Shortest Path Algorithms Using a Priority Queue (Dijkstra's Algorithm): step 1



[5,c],[10,b]

Shortest Path Algorithms

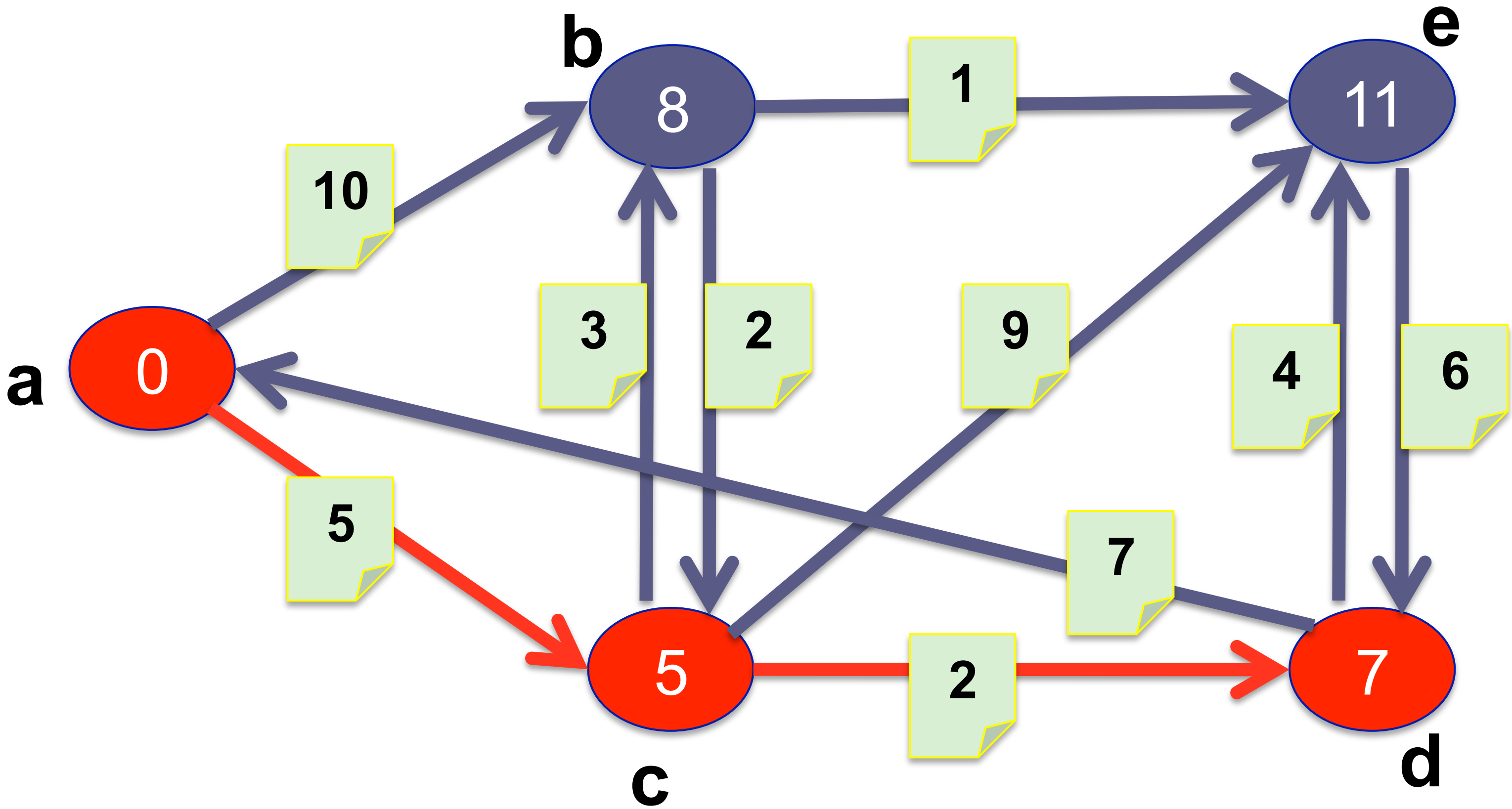
(Dijkstra's Algorithm): step 2



[7,d],[8,b],[14,e]

Shortest Path Algorithms

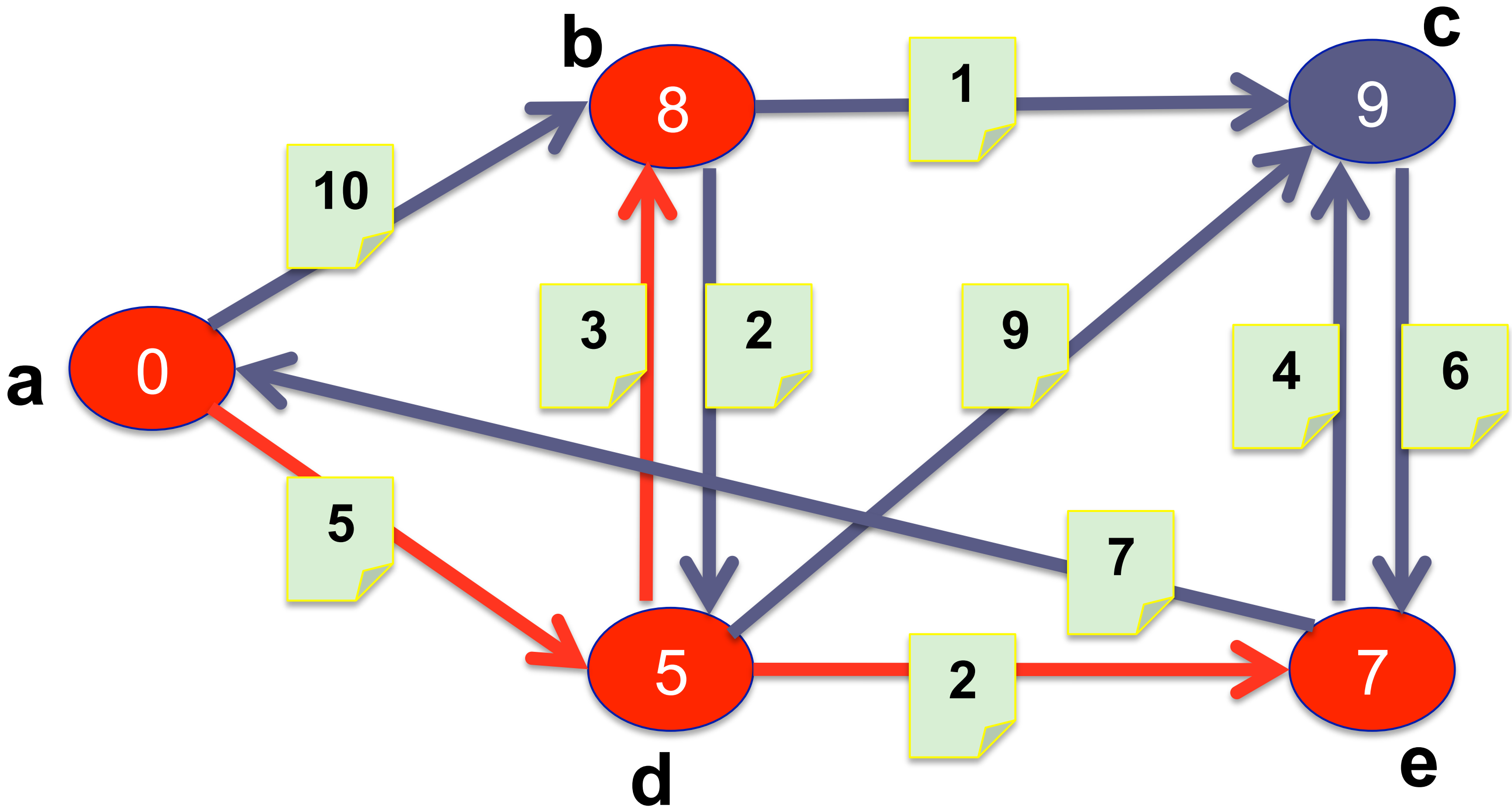
(Dijkstra's Algorithm): step 3



[8,b],[11,e]

Shortest Path Algorithms

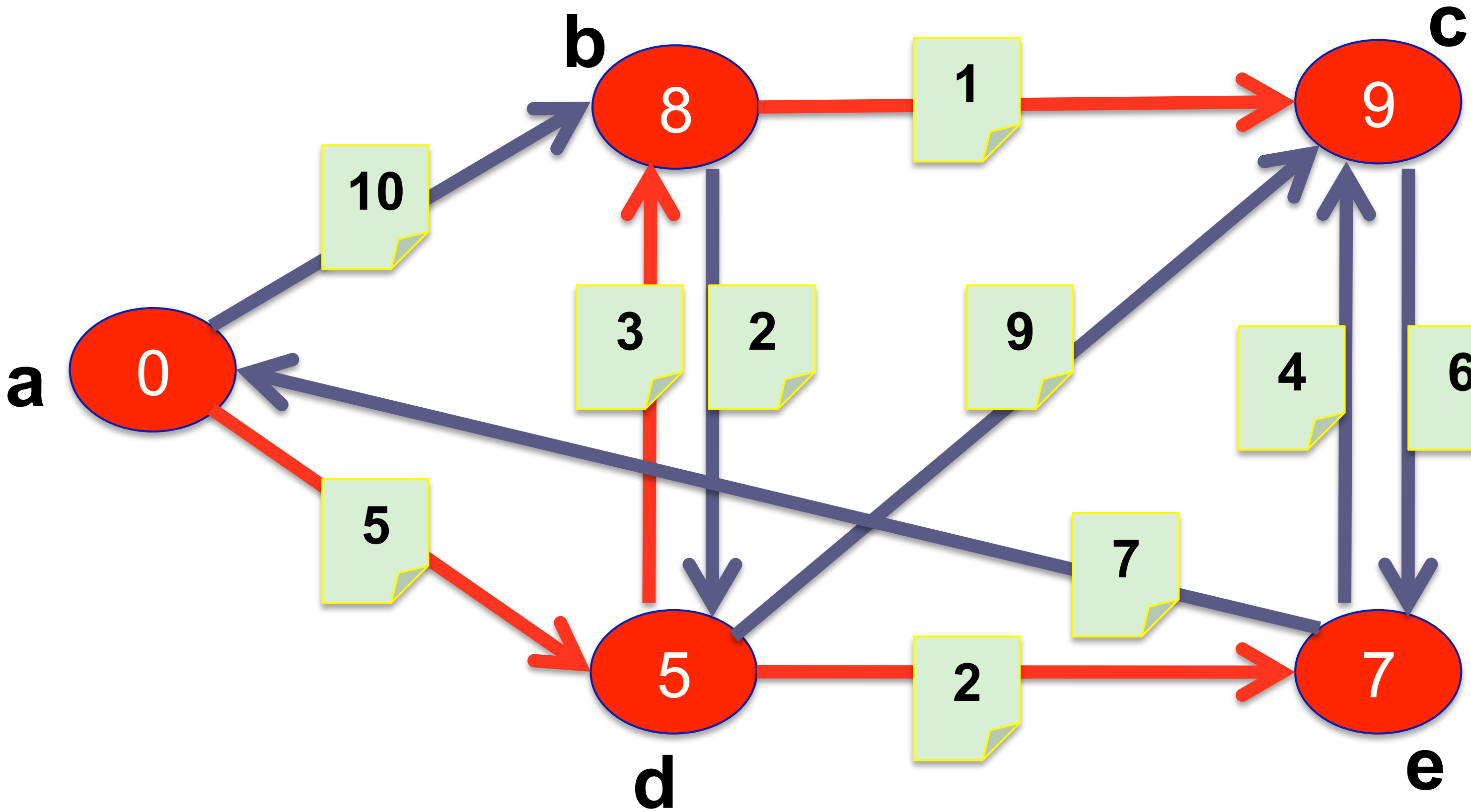
(Dijkstra's Algorithm): step 4



[9,c]

Shortest Path Algorithms

(Dijkstra's Algorithm): Done



Dijkstra's Algorithm

- We have computed the shortest distances.
- How to obtain the shortest paths?

  - At each vertex maintain predecessor on path (parent in the minimal path tree)
  - From each node you can trace back to the source (the root of the minimal path tree)

  - Why maintain predecessor, why not successor?