

Introduction to Algorithms

Lecture 11

Last Time

- Optimization Problems
- Greedy Algorithms
- Graph Representation & Algorithms
- Minimum Spanning Tree
 - Prim's Algorithm
 - Kruskal's Algorithm

Today's Topics

- Shortest paths
- Dijkstra's algorithm
- Breadth-First-Search
- Depth-First-Search
- Connected Components

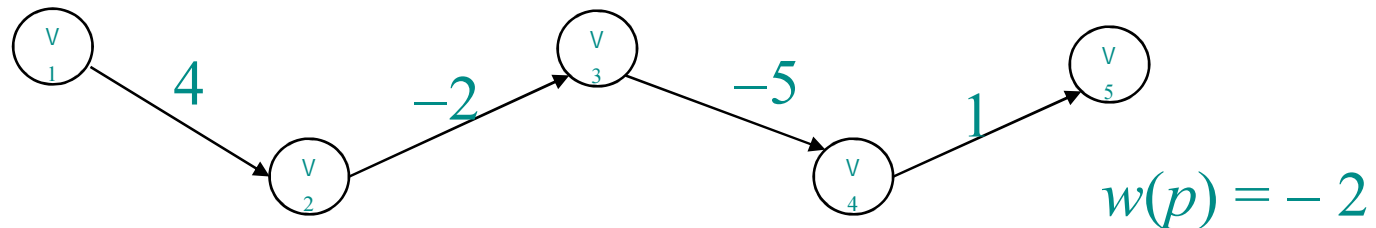
Shortest Paths in Weighted Graphs

Digraph $G = (V, E)$ with weight function $w: E \rightarrow R$

Weight of path $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ is

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

E.g.,

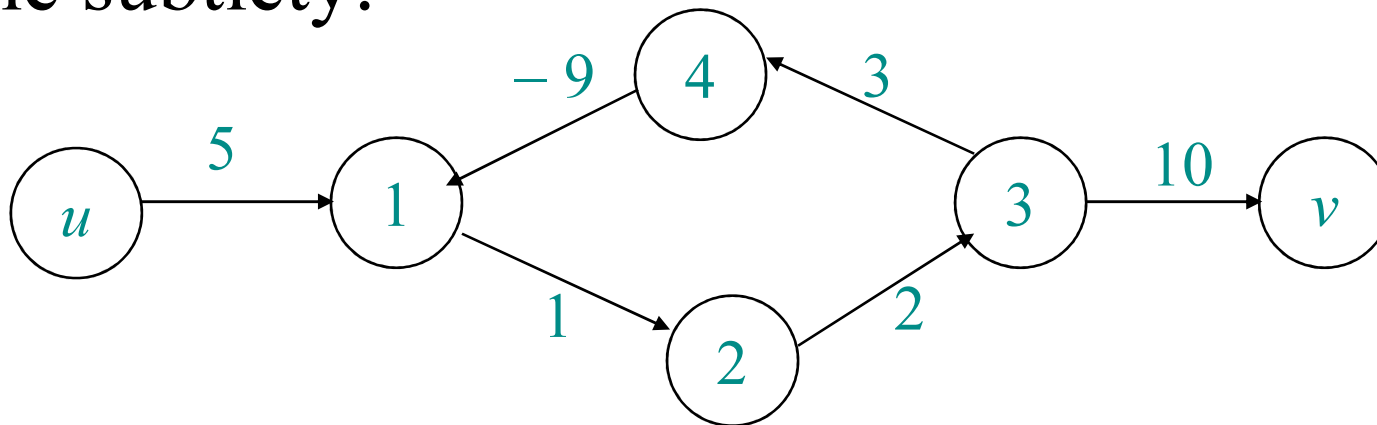


Shortest Paths in Weighted Graphs

Shortest path from u to v is a path of minimum weight from u to v . The **shortest path weight** is the weight of such a path:

$$\delta(u, v) = \begin{cases} \min \{w(p) : u \xrightarrow{p} v\} & \text{If there is a path from } u \text{ to } v, \\ \infty & \text{Otherwise.} \end{cases}$$

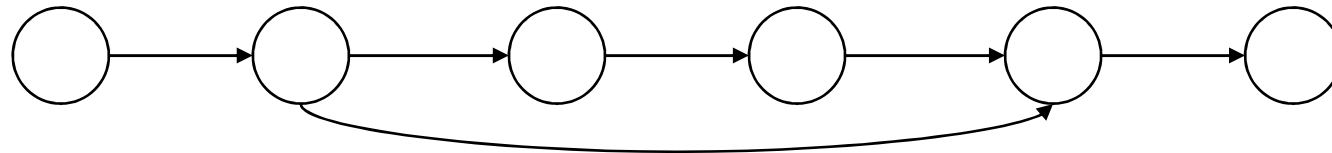
One subtlety:



Optimal Substructure

Theorem: Subpaths of shortest paths are shortest paths.

Proof: Cut and paste:



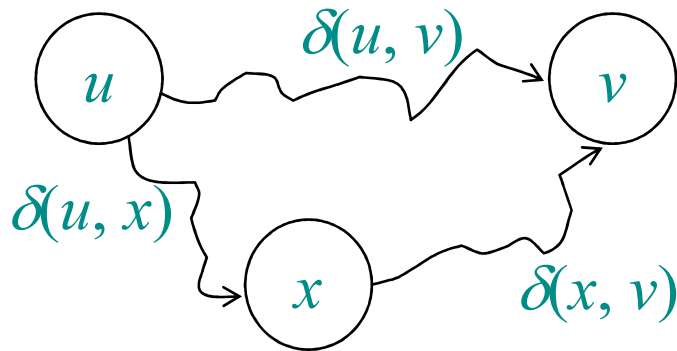
If some subpath were NOT a shortest path, could substitute the shorter subpath and create a shorter total path.

Triangle Inequality

Theorem: For all $u, v, x \in V$,

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

Proof:



If triangle inequality

violates then $u \rightsquigarrow x \rightsquigarrow v$

is a shorter path than $u \rightsquigarrow v$,

contradicting statement

that $\delta(u, v)$ corresponds

to a shortest path.

Single-source shortest paths problem

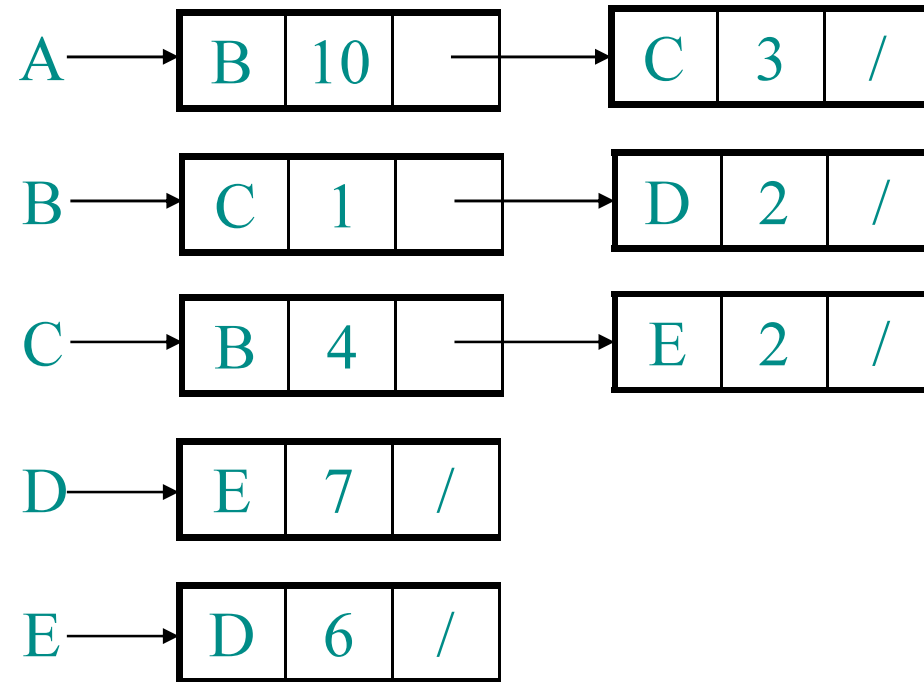
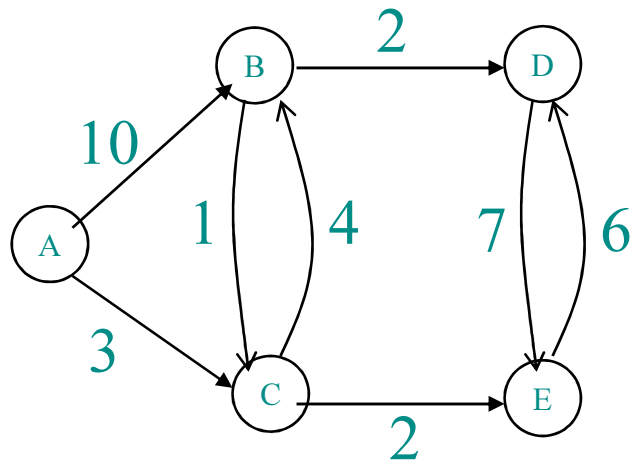
Goal: From a given source vertex $s \in V$, find the shortest-path weight $\delta(s, v)$ for all $v \in V$.

Here we assume $w(u, v) \geq 0$, so $\delta(s, v) \geq -\infty$.

Some variants:

- Single-destination shortest-path problem
- Single-pair shortest-path problem
- All-pairs shortest-path problem

Adjacency-List Representation



Mini-Priority Queue

A data structure for maintaining a set S of elements, each with an associated value (key), supporting:

- $\text{Insert}(S, x)$: insert the element x into S .
- $\text{Minimum}(S)$: returns element with smallest key.
- $\text{Extract-Min}(S)$: return and removes element with smallest key.
- $\text{Decrease-Key}(S, x, k)$: decreases the value of element x 's key to k .

Dijkstra's Algorithm

Idea: Greedy Algorithm

(only valid for non-negative weights)

1. Maintain set S of vertices whose shortest path distance from s are known.
2. At each step, add to S the vertex $u \in V - S$ whose distance estimate from s is minimum.
3. Update distance estimates of vertices adjacent to u .

Dijkstra(G, w, s)

$d[s] \leftarrow 0$

$d[v] \leftarrow \infty$ for each $v \in V - \{s\}$

$S \leftarrow \emptyset$

$Q \leftarrow V$ (priority queue of vertices keyed by d)

while $Q \neq \emptyset$

do $u \leftarrow \text{Extract-Min}(Q)$

$S \leftarrow S \cup \{u\}$

for each $v \in \text{Adj}[u]$

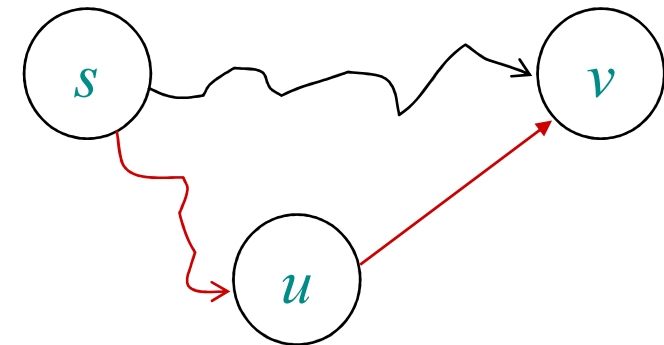
do if $d[v] > d[u] + w(u, v)$

then $d[v] \leftarrow d[u] + w(u, v)$

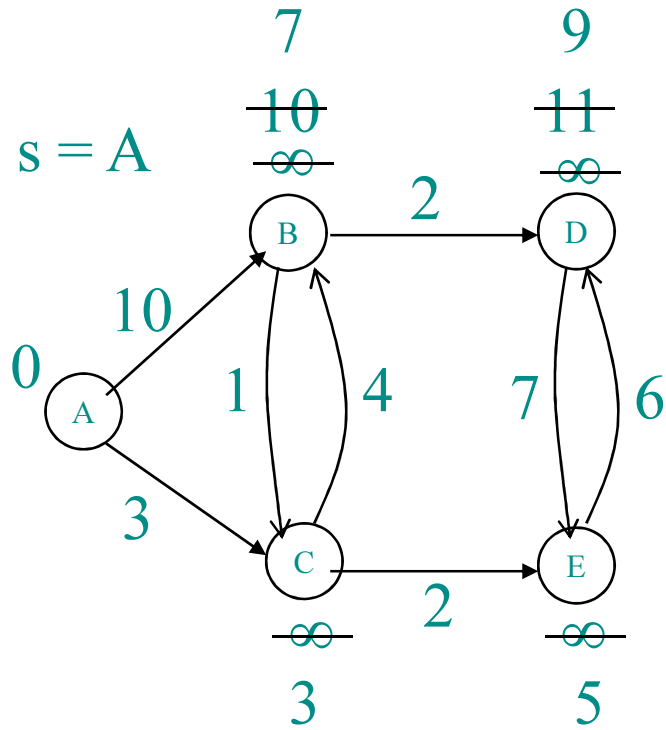
} relaxation step

Implicit Decrease-Key

Maintain set of distance estimates, d and update to shortest-path weights before adding to S



Example



| Q: | A | B | C | D | E |
|----|----------|----------|----------|----------|----------|
| d: | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | ∞ | 5 |
| | | 7 | | 11 | |
| | | | | 9 | |

$S = \{A, C, E, B, D\}$

Correctness (Part I)

Lemma: Invariant: $d[v] \geq \delta(s, v) \forall v \in V$ at all times.

Proof:

Init $d[s] = 0$ and $d[v] = +\infty$ for $v \neq s$; $\delta(s, s) = 0$
and $\delta(s, v) \leq \infty \forall v$, so OK.

Suppose invariant fails, that v is the first vertex
with $d[v] < \delta(s, v)$ and u is the vertex that cause
 $d[v]$ to change by $d[v] = d[u] + w(u, v)$.

Correctness (Part I)

Then

$$\begin{aligned}d[v] &< \delta(s, v) && \text{supposition} \\ &\leq \delta(s, u) + \delta(u, v) && \text{triangle inequality} \\ &\leq \delta(s, u) + w(u, v) && \text{shortest path } \leq \text{specific path} \\ &\leq d[u] + w(u, v) && v \text{ is first violation, so } \delta(s, u) \leq d[u]\end{aligned}$$

$$d[v] < d[u] + w(u, v) \text{ contradiction!}$$

Correctness (Part II)

Theorem: When Dijkstra's algorithm terminates,

$$d[v] = \delta(s, v) \quad \forall v \in V.$$

Proof: $d[v]$ doesn't change once added to S , so suffices to show true when added.

Suppose u is the **first** vertex about to be added to S for which $d[u] \neq \delta(s, u)$.

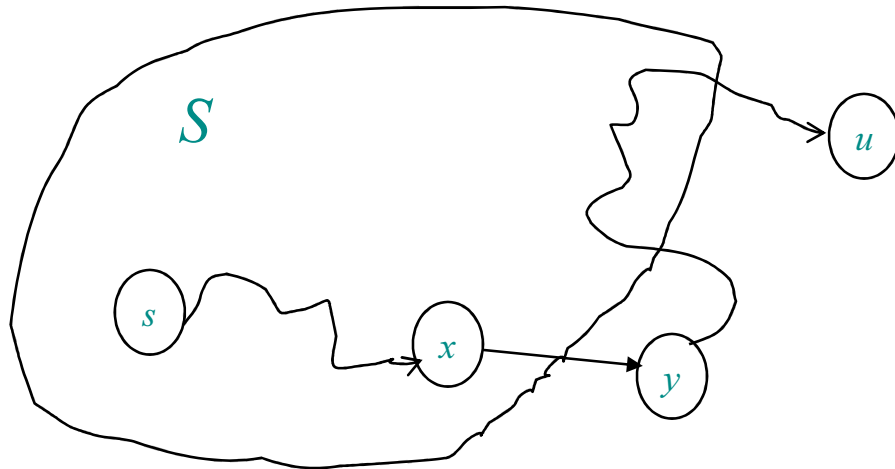
$\Rightarrow d[u] > \delta(s, u)$ by previous lemma.

Let p be a shortest path from s to u [$w(p) = \delta(s, u)$]

Consider **first** place p exits S [via edge (x, y)]

(y is first vertex along p in $V - S$; x is predecessor of y along p)

Correctness (Part II)



Because u is first violation, $d[x] = \delta(s, x)$.

When x was added to S , we relaxed (x, y) and set

$d[y] = \delta(s, x) + w(x, y) = \delta(s, y)$ because subpaths of shortest path are shortest paths

Thus $d[y] = \delta(s, y) \leq \delta(s, u) \leq d(u)$

sub-path previous lemma

Correctness (Part II)

But $d[u] \leq d[y]$ by Dijkstra's choice of u

Emphasizes need
for **greedy** step.

So $d[y] = \delta(s, y) = \delta(s, u) = d[u]$.

Contradiction!

Analysis

$d[v] \leftarrow \infty$ for each $v \in V - \{s\}$

while $Q \neq \emptyset$

do $u \leftarrow \text{Extract-Min}(Q)$

$S \leftarrow S \cup \{u\}$

for each $v \in \text{Adj}[u]$

do if $d[v] > d[u] + w(u, v)$

then $d[v] \leftarrow d[u] + w(u, v)$

$|V|$ times

$\text{degree}(u)$
times

Decrease-Key: $\Theta(E)$

worst-case aggregate analysis

Analysis

$$\text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}$$

(same as Prim's MST algorithm)

| Q | $T_{\text{Extract-Min}}$ | $T_{\text{Decrease-Key}}$ | Total |
|----------------|--------------------------|---------------------------|-----------------------|
| array | $\Theta(V)$ | $\Theta(1)$ | $\Theta(V^2)$ |
| binary heap | $\Theta(\lg V)$ | $\Theta(\lg V)$ | $\Theta(E \lg V)$ |
| Fibonacci heap | $\Theta(\lg V)$ | $\Theta(1)$ | $\Theta(E + V \lg V)$ |
| | amortized | amortized | worst case |

Unweighted Graphs

Suppose $w(u, v) = 1 \forall (u, v) \in E$. Then
Dijkstra's algorithm can be improved using
FIFO queue in place of priority queue.

- BFS
- DFS

Google Problem: Graph Searching

- How to search and explore in the Web Space?
- How to find all web-pages and all the hyperlinks?
 - Start from one vertex “www.yahoo.com”
 - Systematically follow hyperlinks from the discovered vertex/web page
 - Build a search tree along the exploration

Breadth-First-Search

BFS(G, w, s)

Time: $\Theta(V + E)$

$d[s] \leftarrow 0$

$d[v] \leftarrow \infty$ for each $v \in V - \{s\}$

$Q \leftarrow \{s\}$

while $Q \neq \emptyset$

do $u \leftarrow \text{Dequeue}(Q)$

for each $v \leftarrow \text{Adj}[u]$

do if $d[v] = \infty$

then $d[v] \leftarrow d[u] + 1$

$\text{Enqueue}(Q, v)$

Correctness of BFS

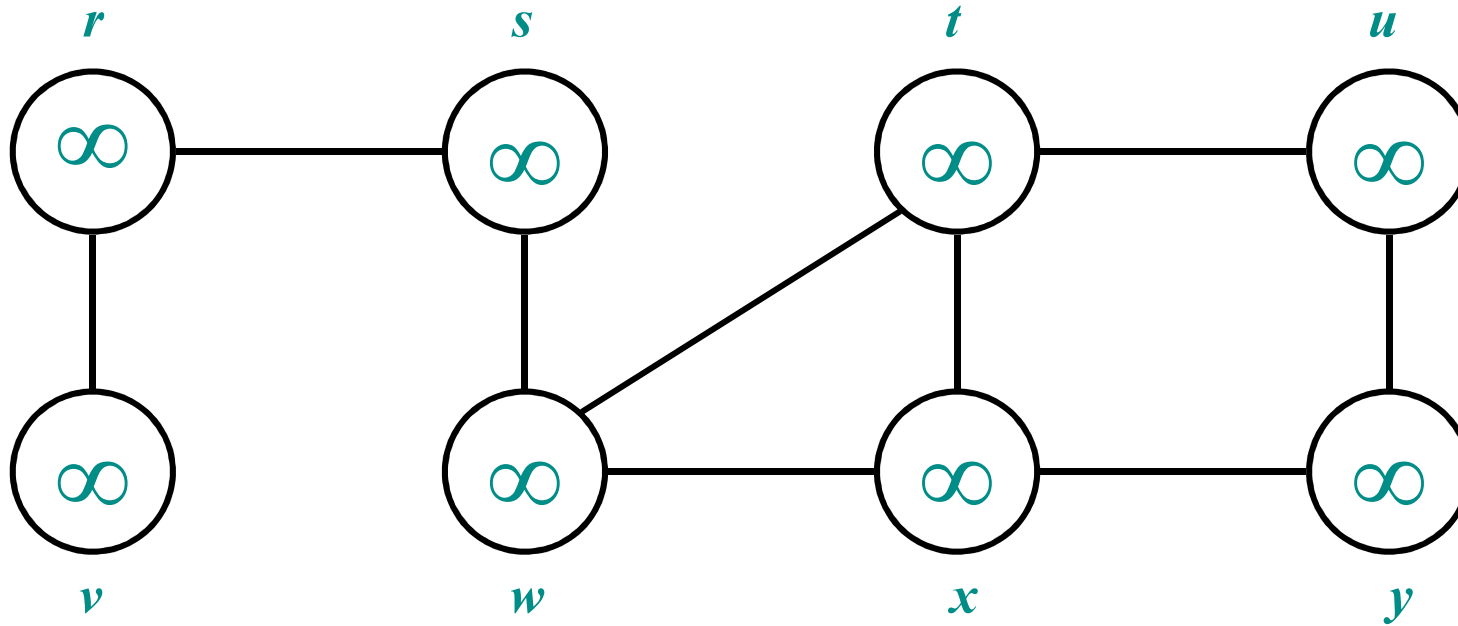
Key Idea: FIFO queue in BFS mimics priority queue in Dijkstra.

Invariant: v immediately after u in queue

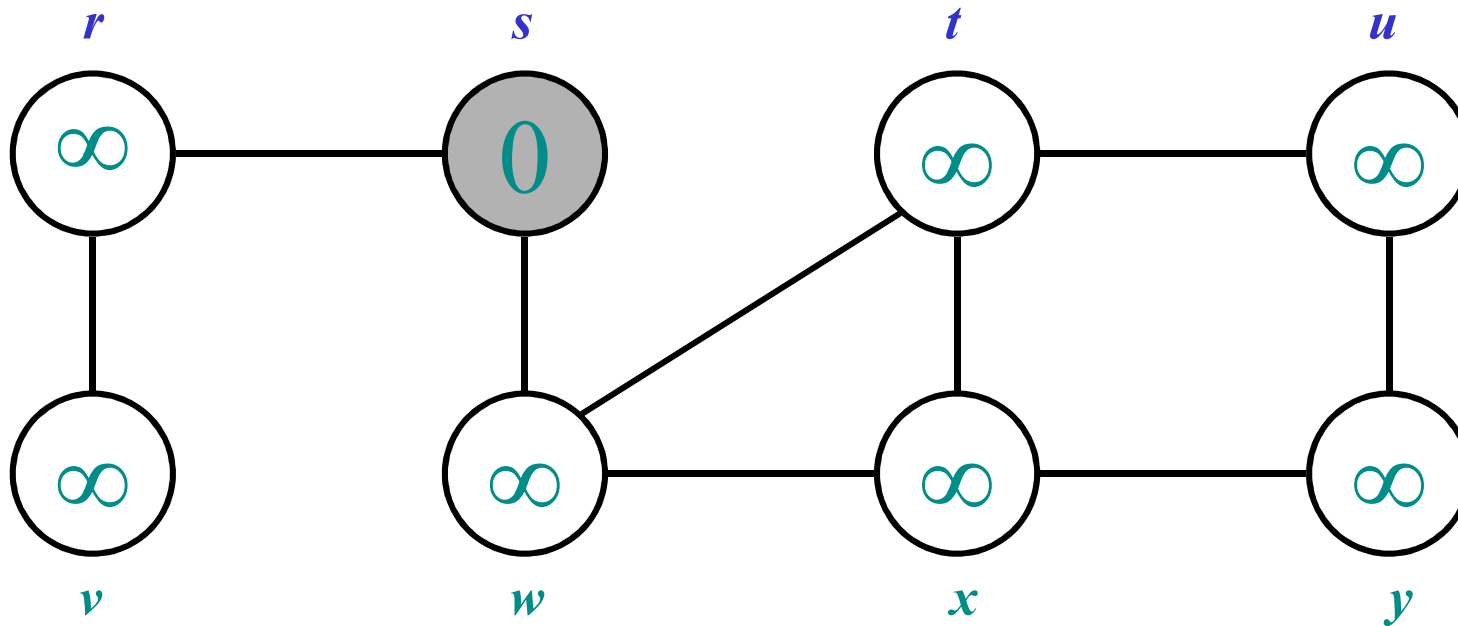
$\Rightarrow d[v]$ is either $d[u]$ or $d[u] + 1$.

(At all times, there are two distinct d values in the queue.)

Example

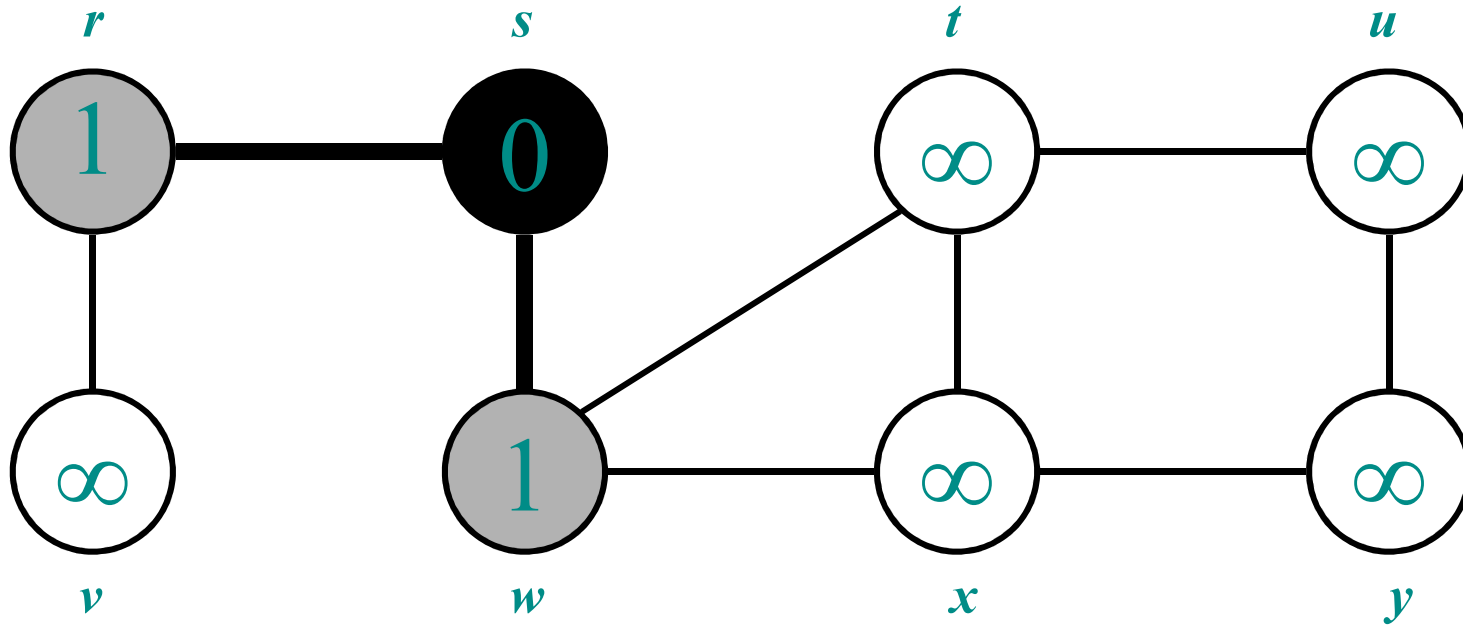


Example



$Q:$ s

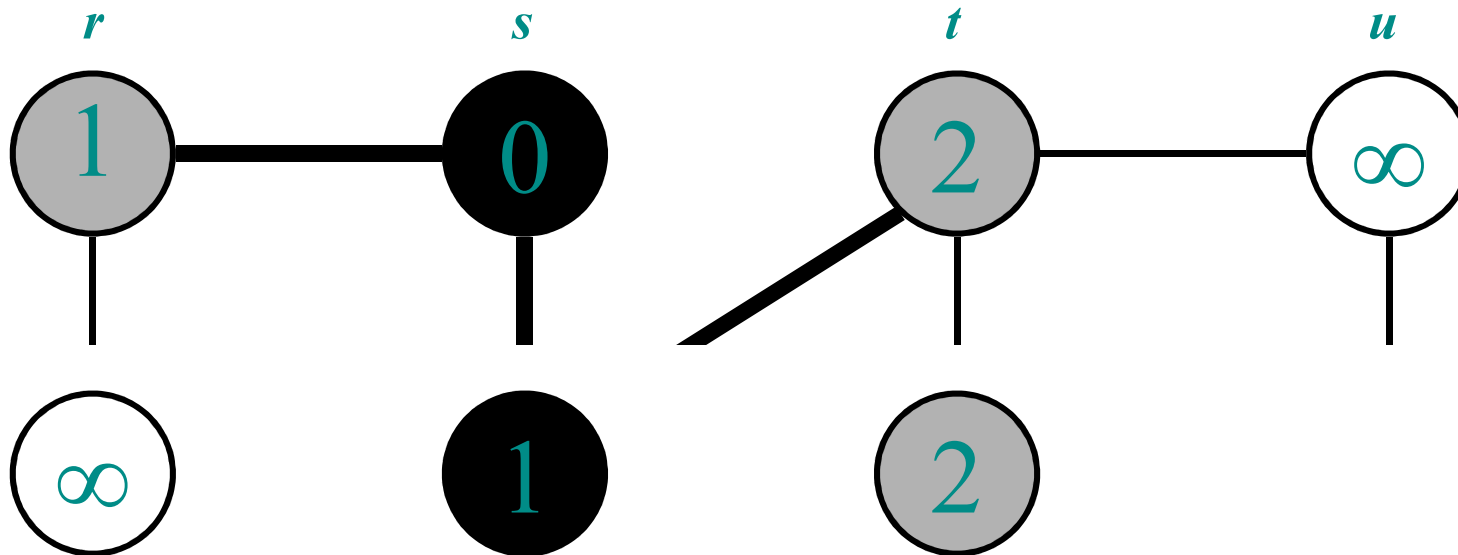
Example



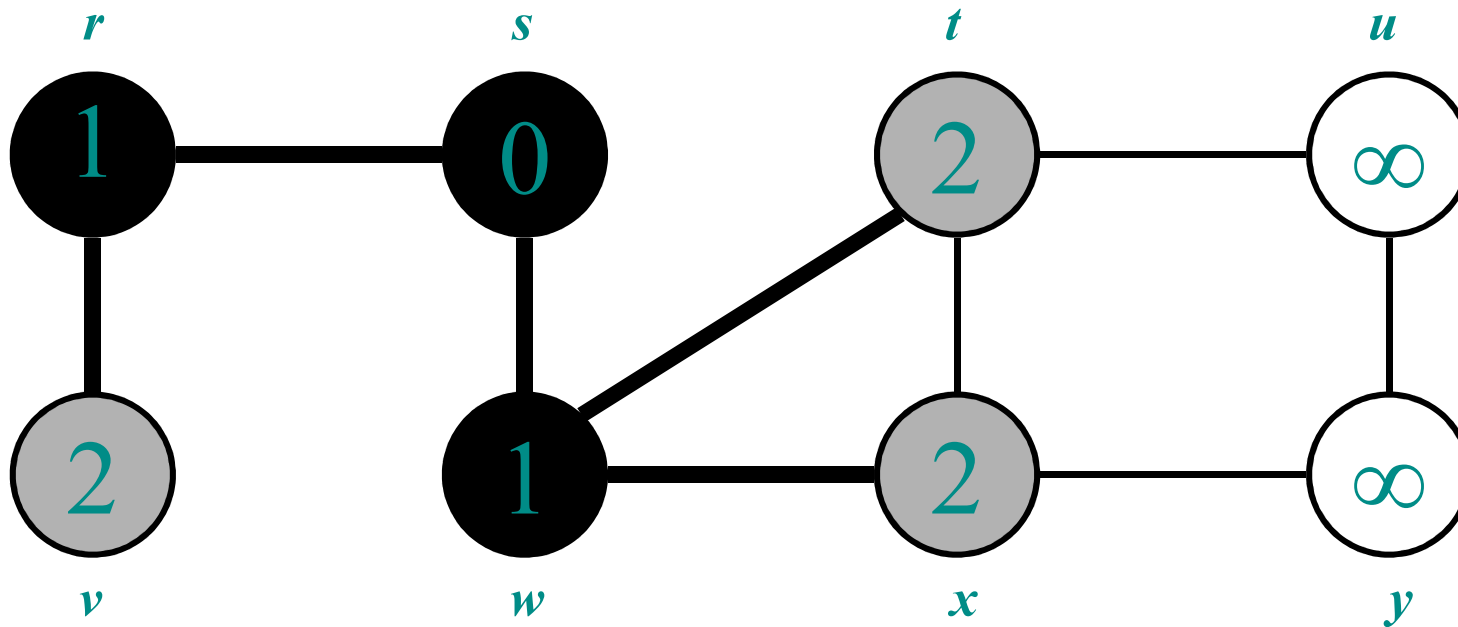
$Q:$

| | |
|-----|-----|
| w | r |
|-----|-----|

Example



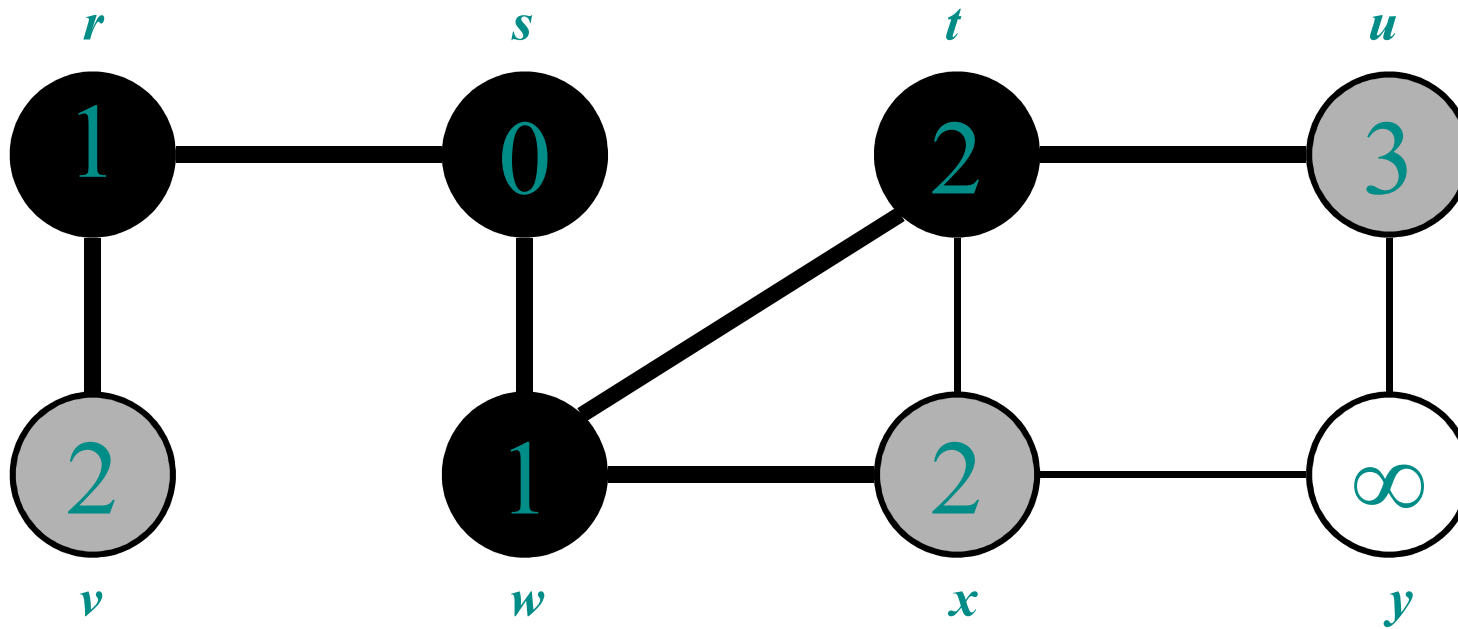
Example



Q :

| | | |
|-----|-----|-----|
| t | x | v |
|-----|-----|-----|

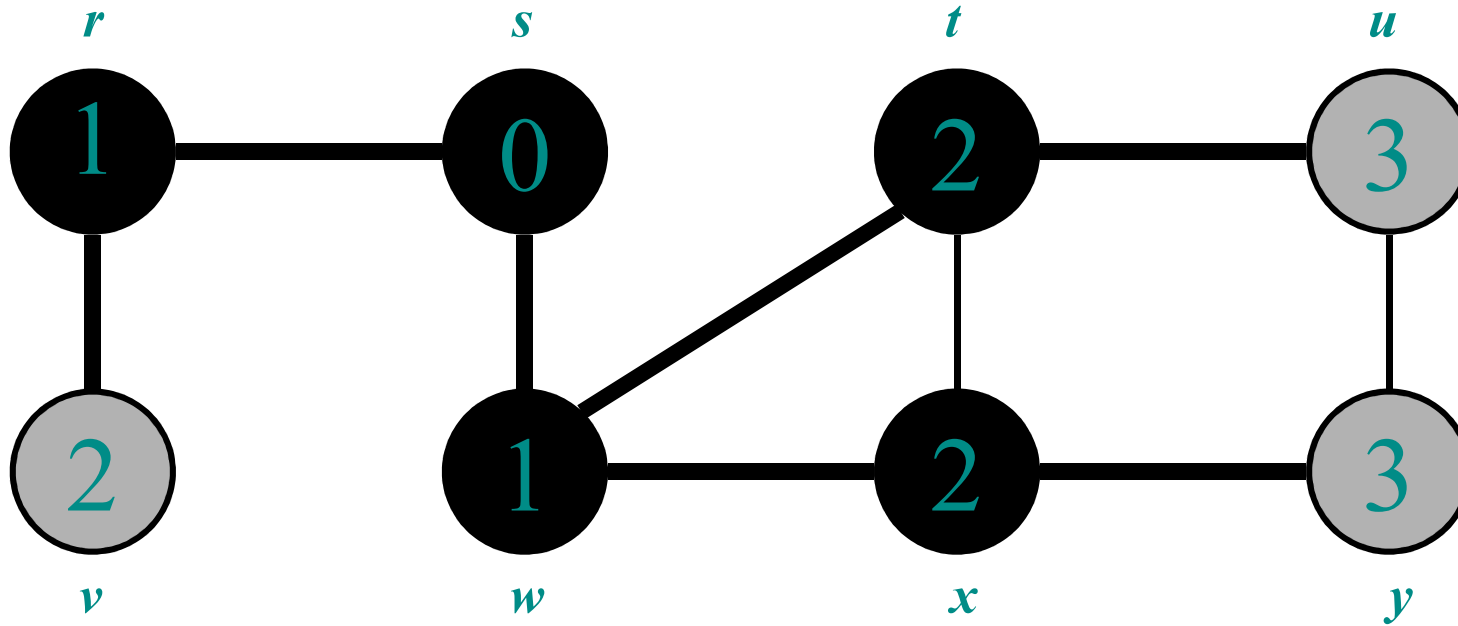
Example



Q :

| | | |
|-----|-----|-----|
| x | v | u |
|-----|-----|-----|

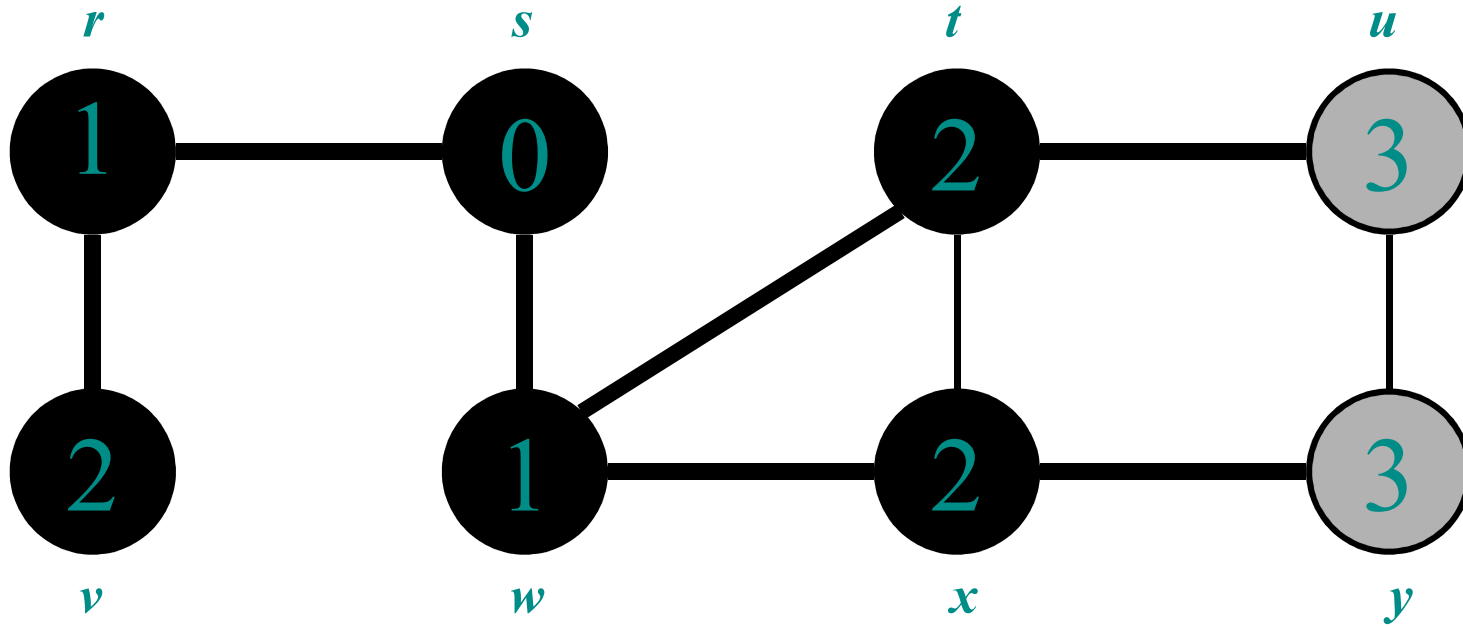
Example



$Q:$

| | | |
|-----|-----|-----|
| v | u | y |
|-----|-----|-----|

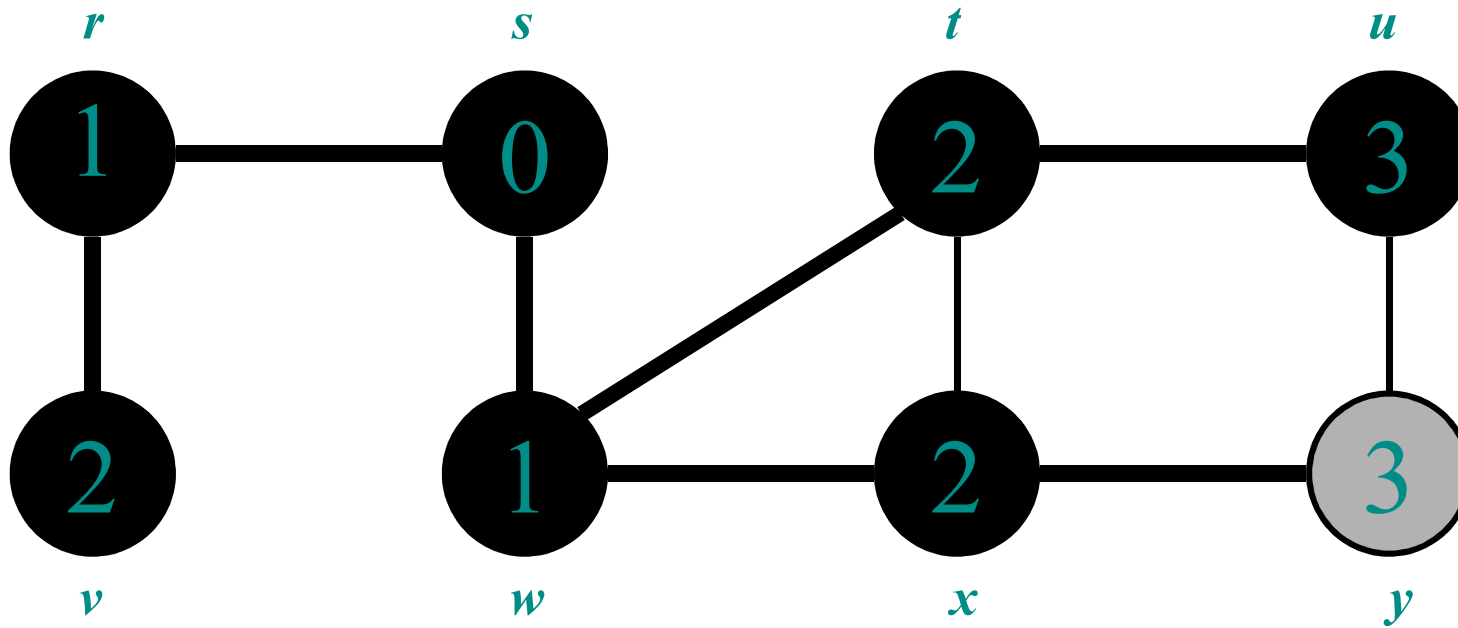
Example



Q :

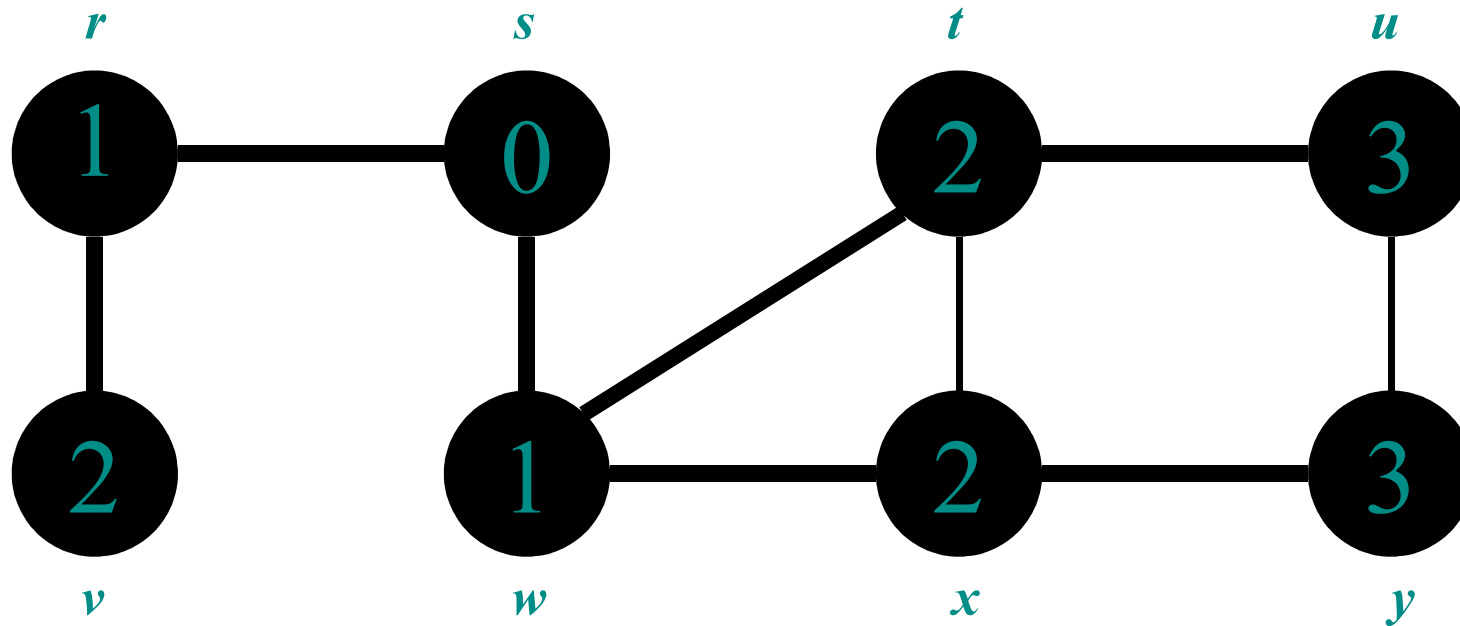
| | |
|-----|-----|
| u | y |
|-----|-----|

Example



Q : y

Example



$Q: \emptyset$

Properties of BFS

- **Proposition:** Let G be an undirected graph on which a **BFS** traversal starting at vertex s has been performed. Then
 - The traversal visits all vertices in the connected component of s .
 - The discovery-edges form a spanning tree T , which we call the **BFS tree**, of the connected component of s

Properties of BFS

- For each vertex v at level i , the path of the **BFS tree** T between s and v has i edges, and any other path of G between s and v has at least i edges.
- If (u, v) is an edge that is not in the BFS tree, then the level numbers of u and v differ by at most one.

Properties of BFS

- **Proposition:** Let G be a graph with n vertices and m edges. A BFS traversal of G takes time $O(n + m)$. Also, there exist $O(n + m)$ time algorithms based on BFS for the following problems:
 - Testing whether G is connected.
 - Computing a spanning tree of G
 - Computing the connected components of G
 - Computing, for every vertex v of G , the minimum number of edges of any path between s and v .

Simple DFS: Pseudocode

Simple-DFS(G)

1. **for** each vertex $u \in V$
 2. **do** $color[u] \leftarrow \text{WHITE}$
 3. **for** each vertex $u \in V$
 4. **do if** $color[u] = \text{WHITE}$
 5. **then** Simple-DFS-Visit(u)
-

Simple-DFS-Visit(u)

1. $color[u] \leftarrow \text{BLACK}$
2. **for** each $v \in Adj[u]$
3. **do if** $color[v] = \text{WHITE}$
4. **then** Simple-DFS-Visit(v)

Timed DFS: Pseudocode

- input graph G may be directed or undirected.
- $time$ is global variable used for time-stamping.

DFS(G)

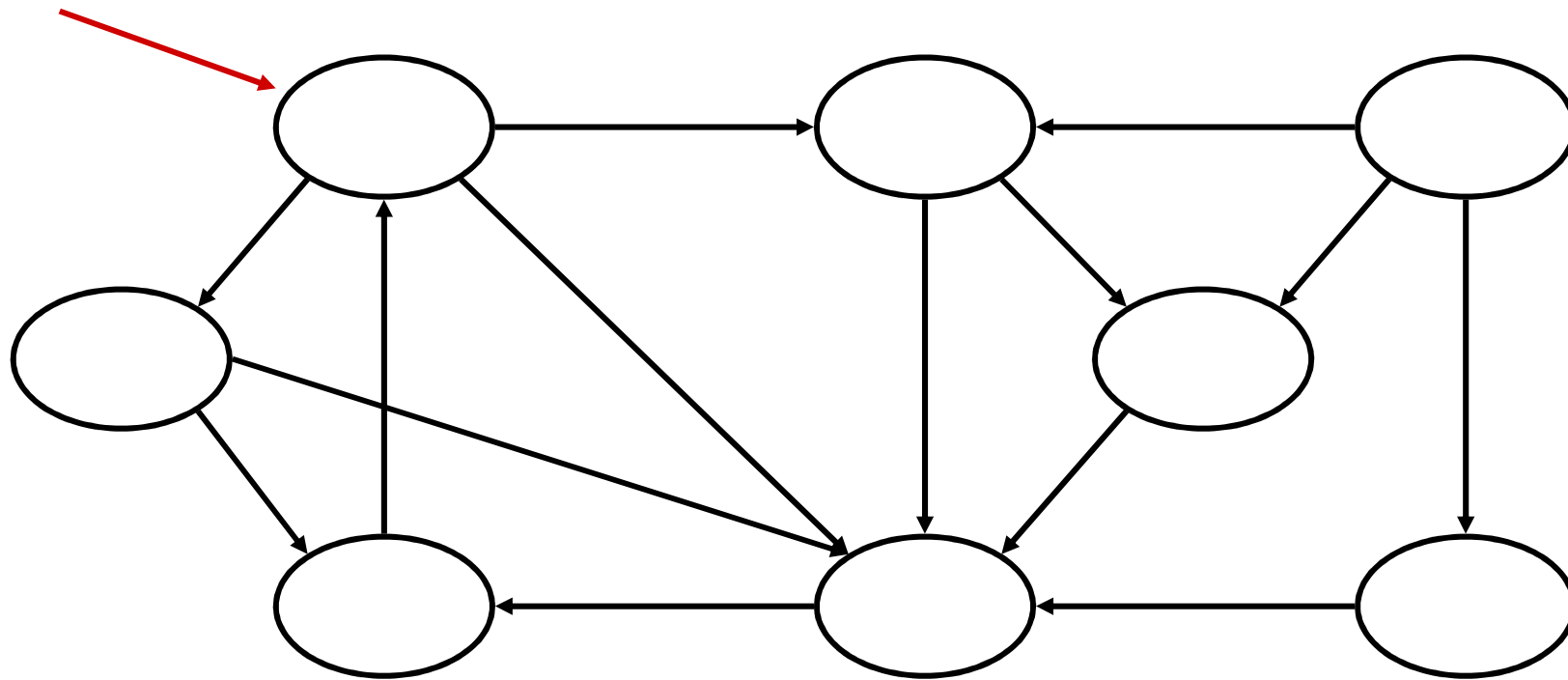
1. **for** each vertex $u \in V$
2. **do** $color[u] \leftarrow \text{WHITE}$
3. $time \leftarrow 0$
4. **for** each vertex $u \in V$
5. **do if** $color[u] = \text{WHITE}$
6. **then** DFS-Visit(u)

DFS-Visit(u)

1. $color[u] \leftarrow \text{GRAY}$ //White vertex u discovered.
2. $d[u] \leftarrow time$ //Mark with **discovered time**.
3. $time \leftarrow time + 1$ //Tick global time.
4. **for** each $v \in Adj[u]$ //Explore all edges (u, v) .
5. **do if** $color[v] = \text{WHITE}$
6. **then** DFS-Visit(v)
7. $color[u] \leftarrow \text{BLACK}$ //Blacken u ; it is finished.
8. $f[u] \leftarrow time$ //Mark with **finishing time**.
9. $time \leftarrow time + 1$ //Tick global time.

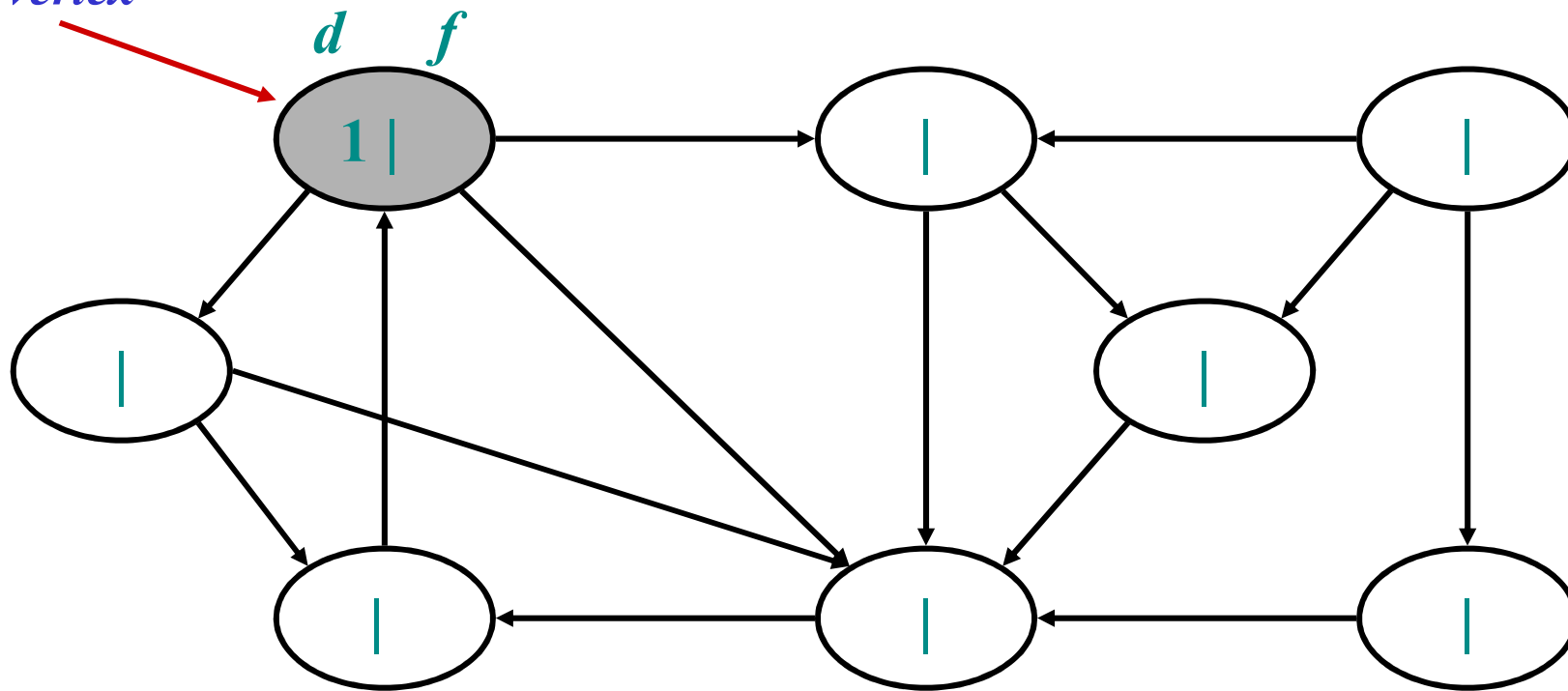
DFS Example

*source
vertex*

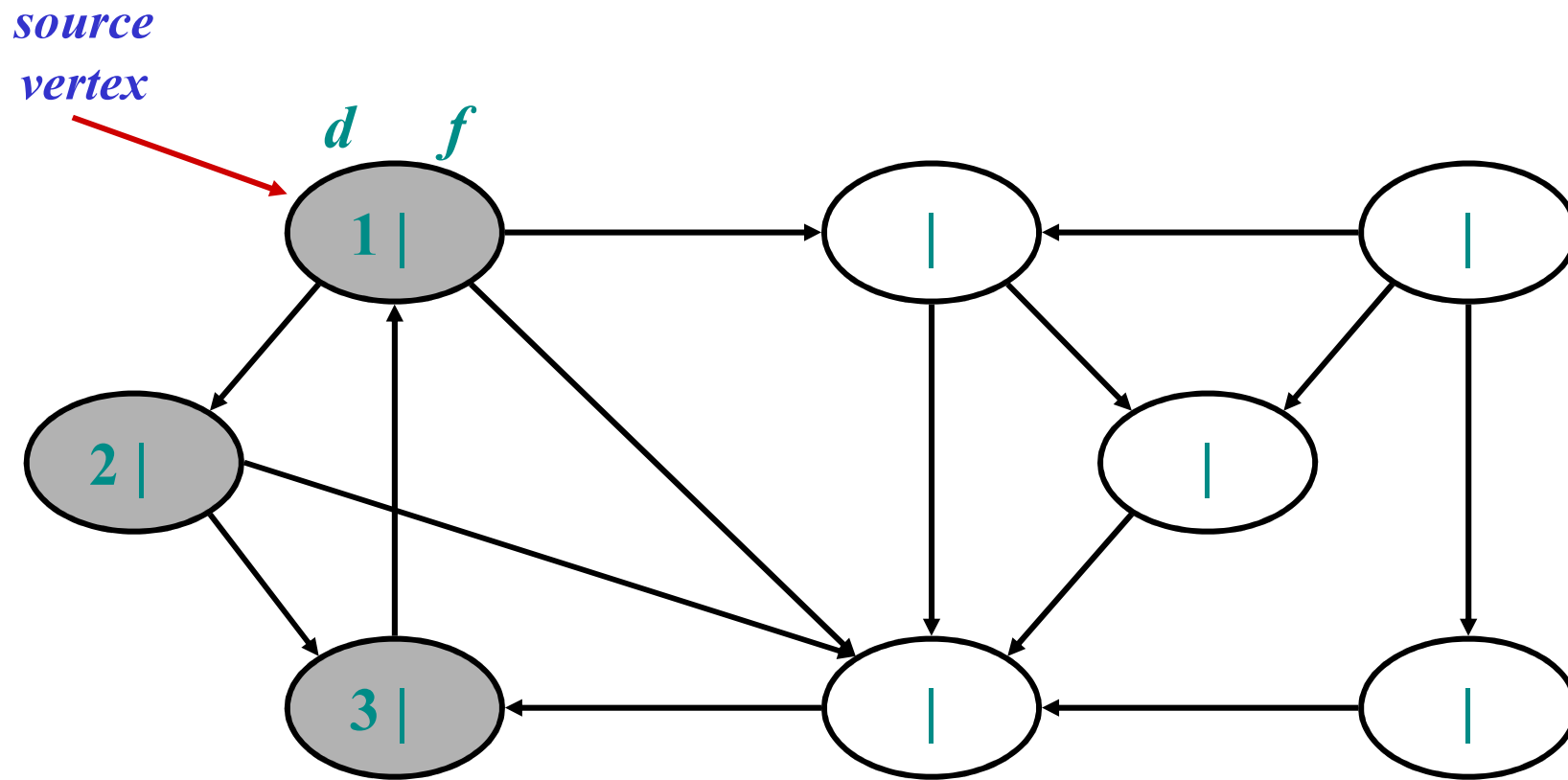


DFS Example

*source
vertex*

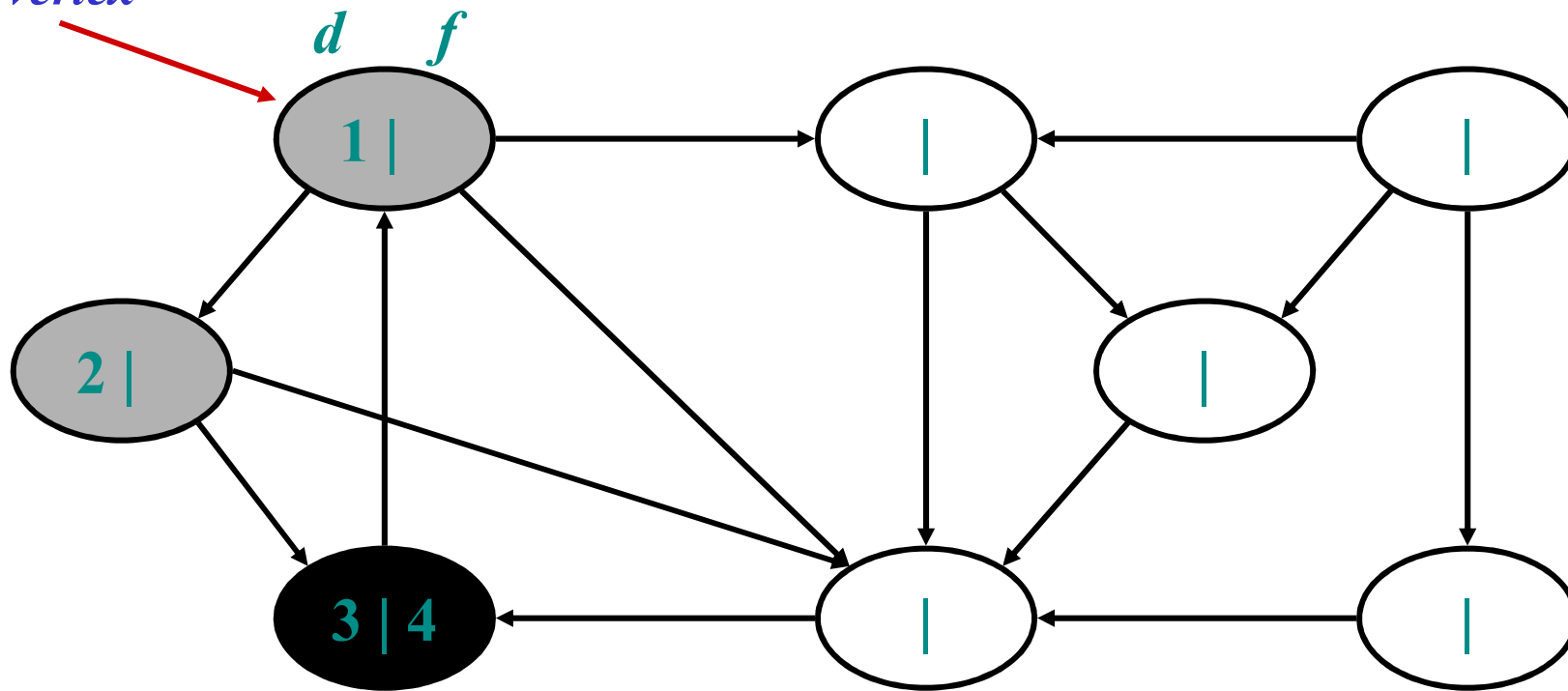


DFS Example

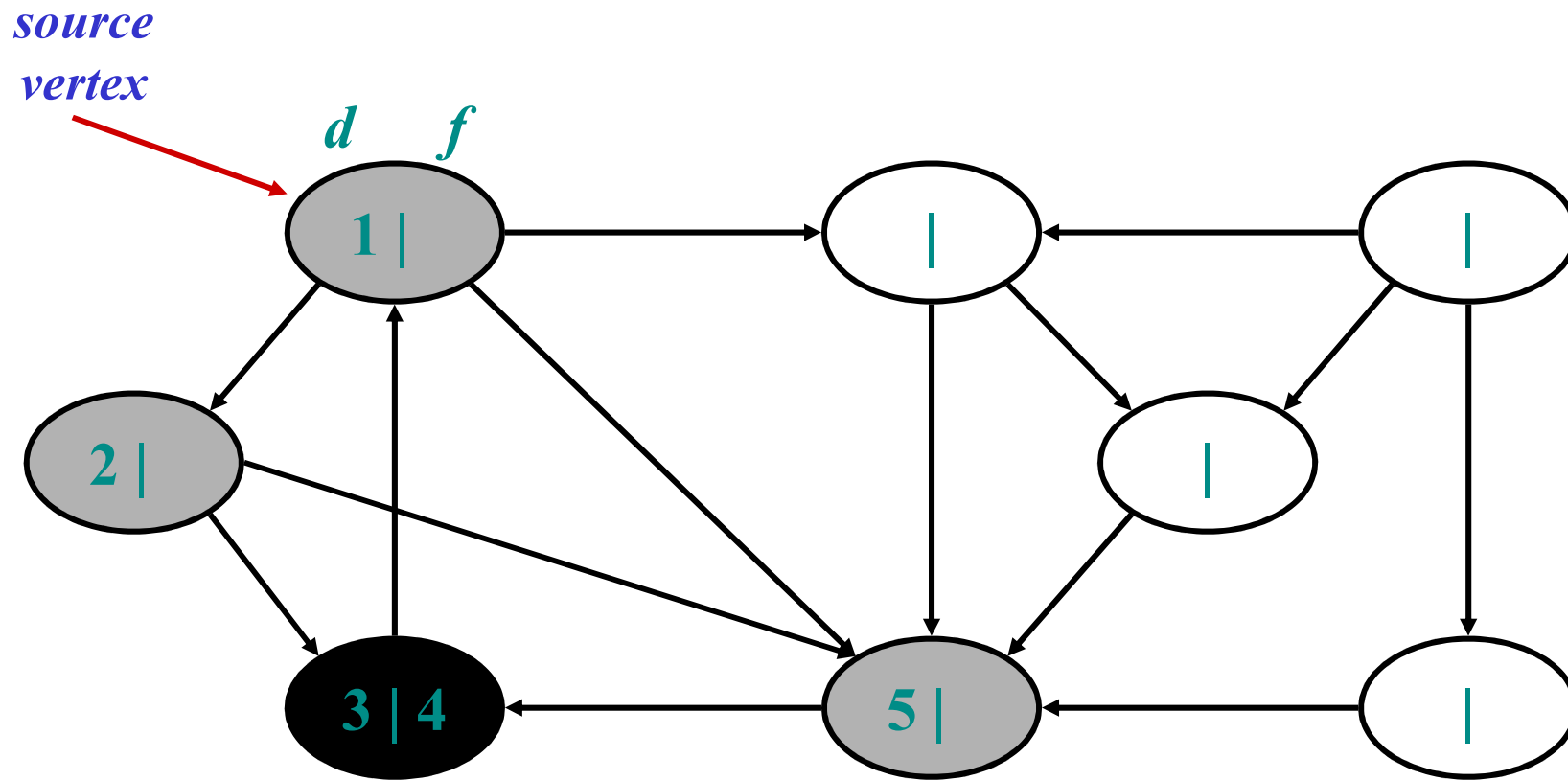


DFS Example

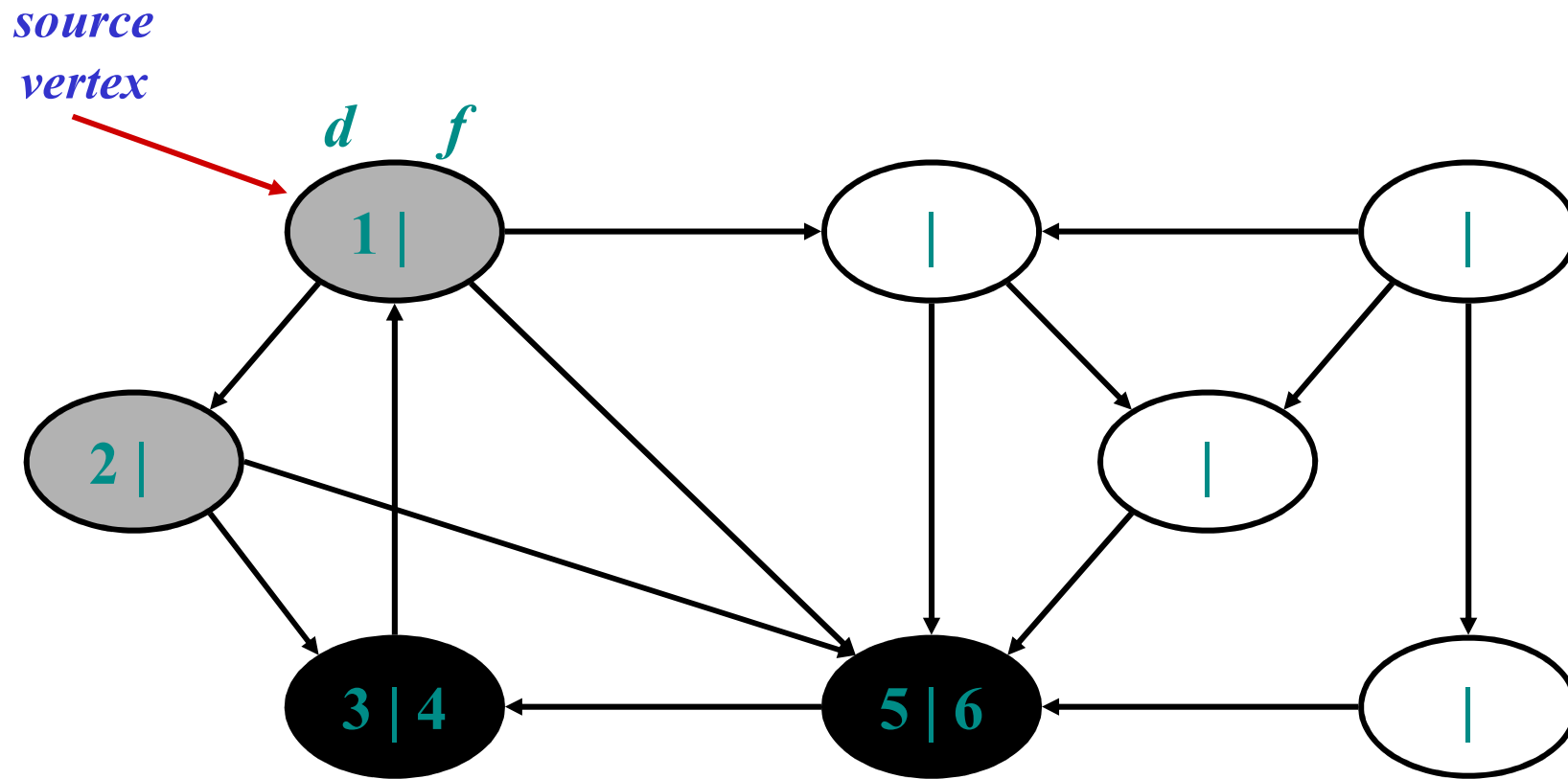
*source
vertex*



DFS Example

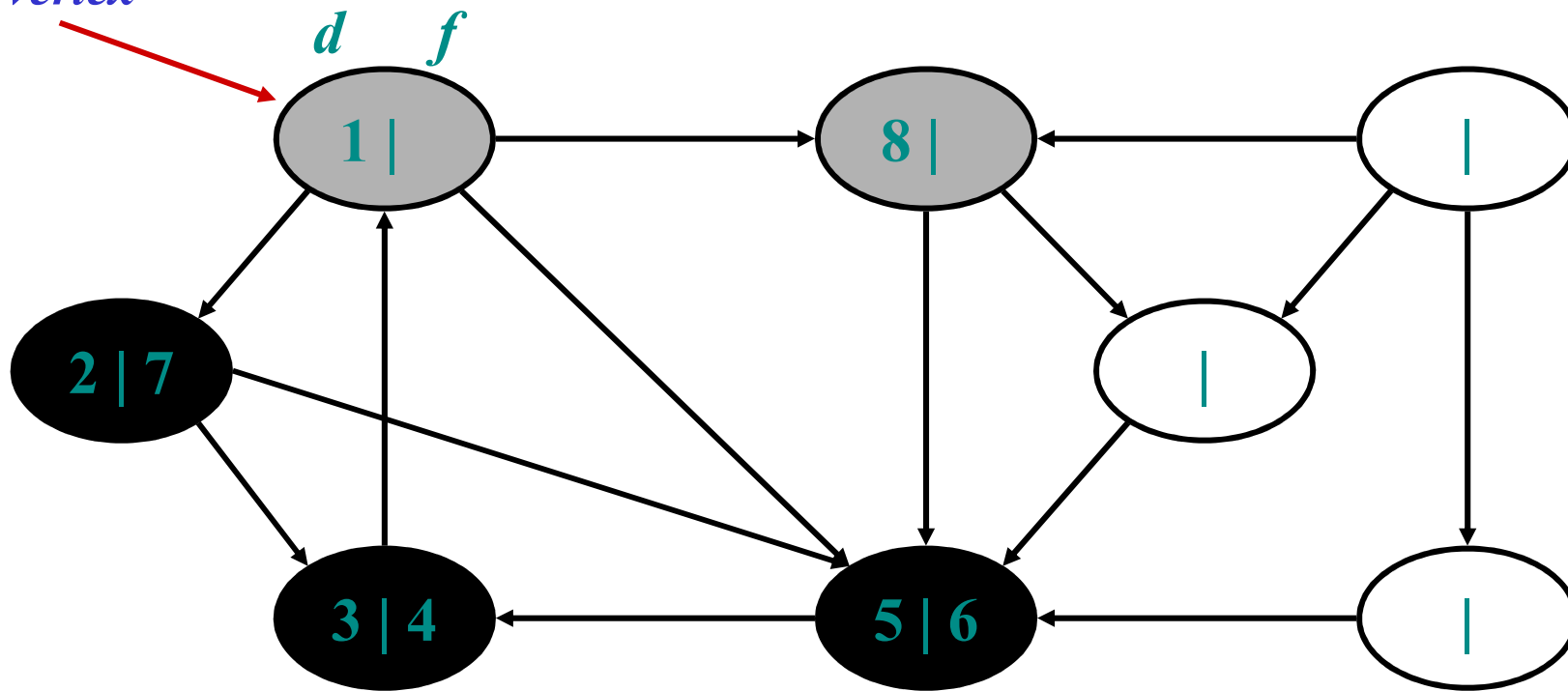


DFS Example



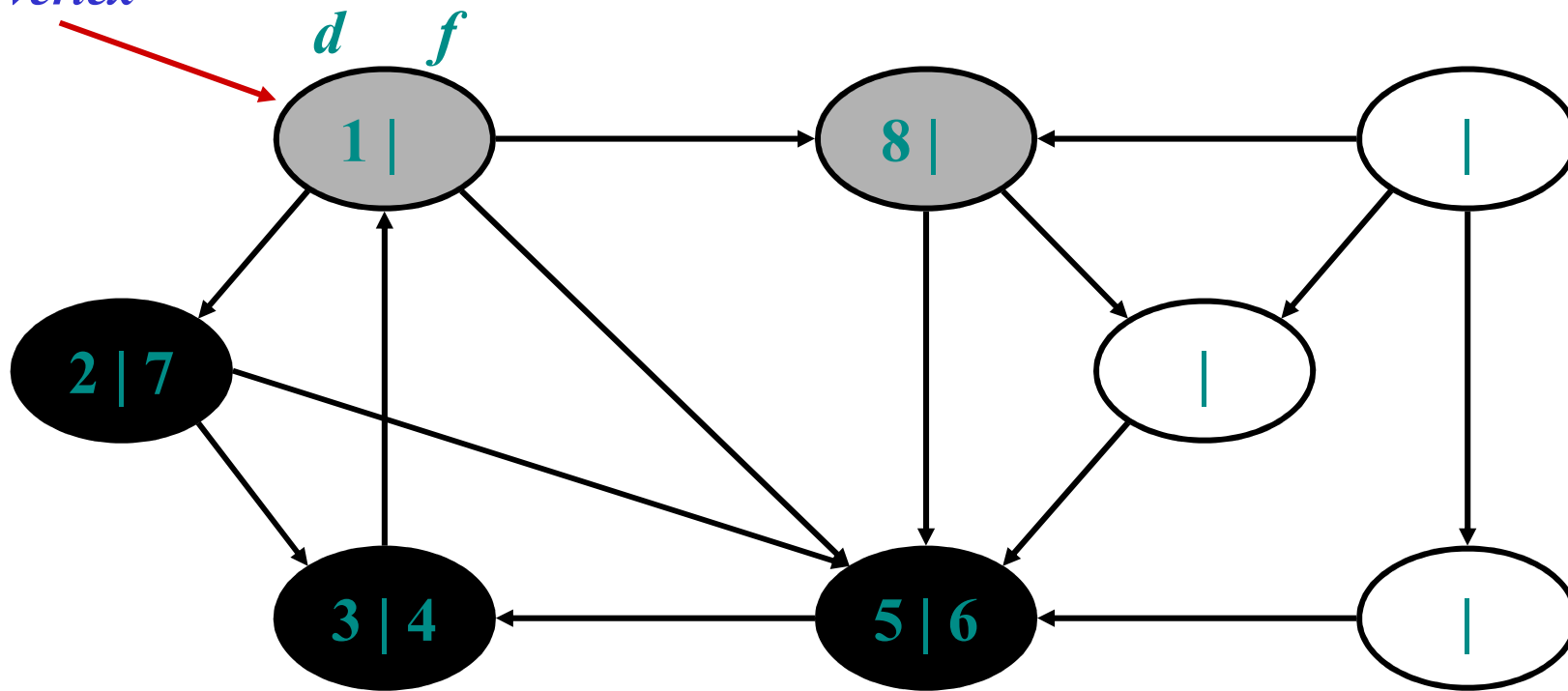
DFS Example

*source
vertex*



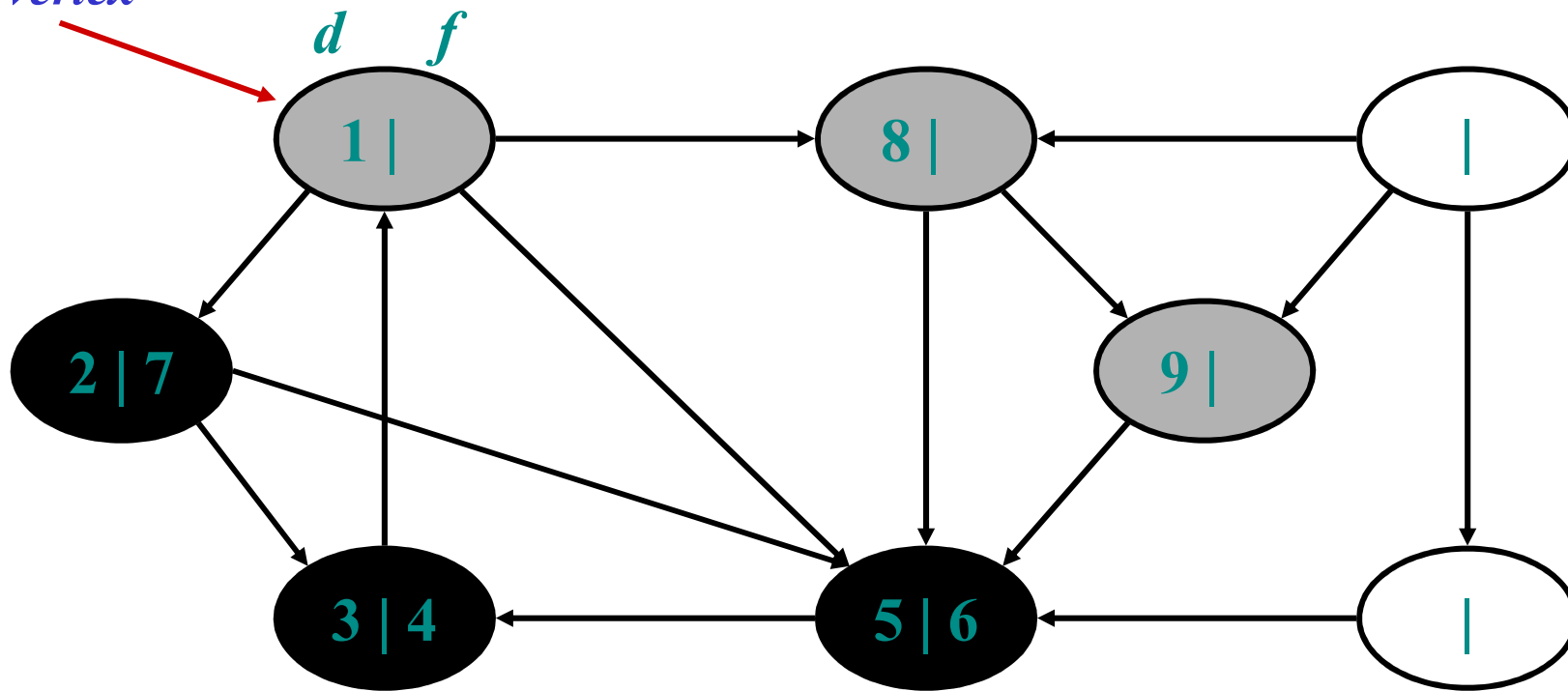
DFS Example

*source
vertex*

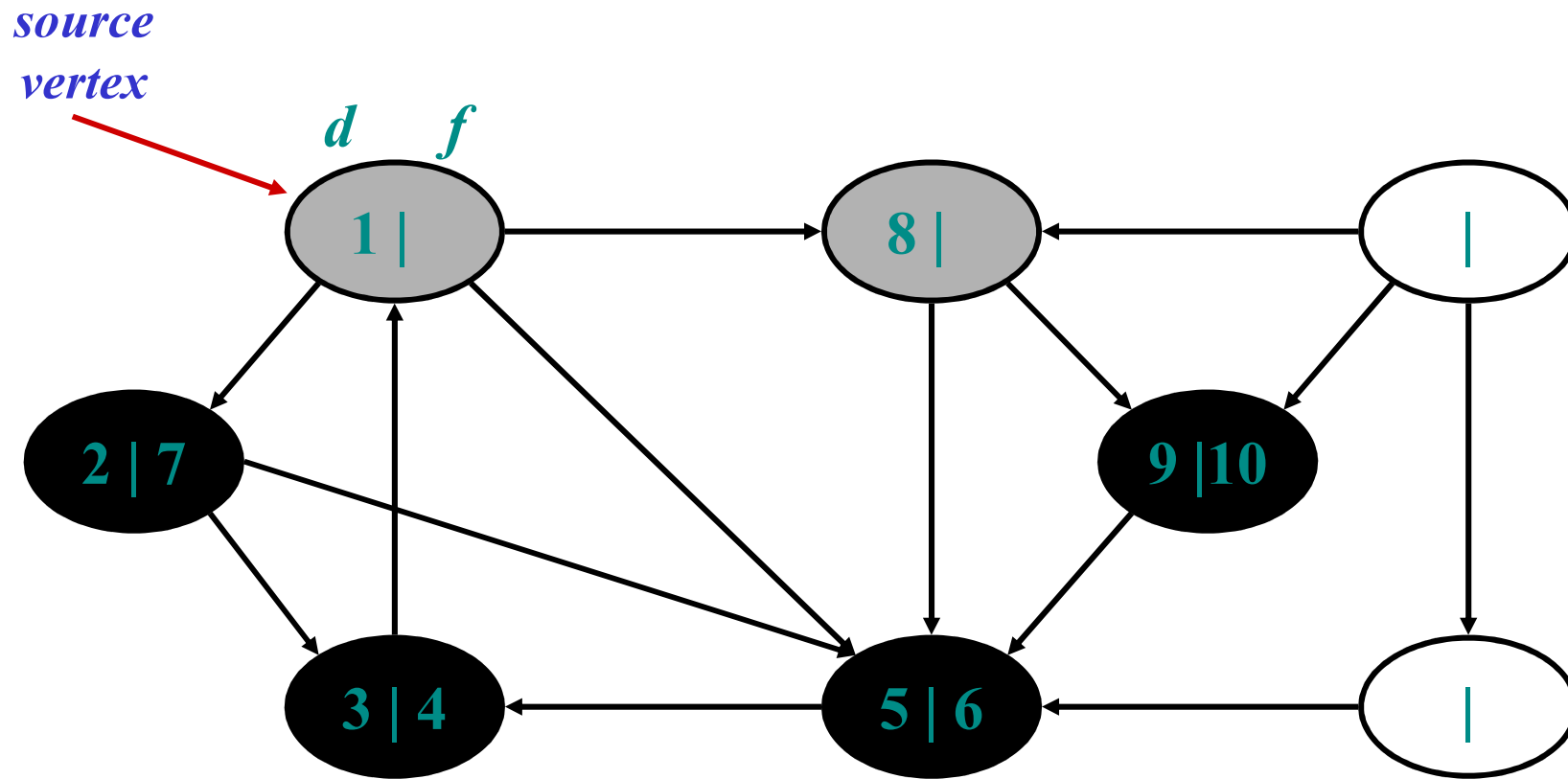


DFS Example

*source
vertex*

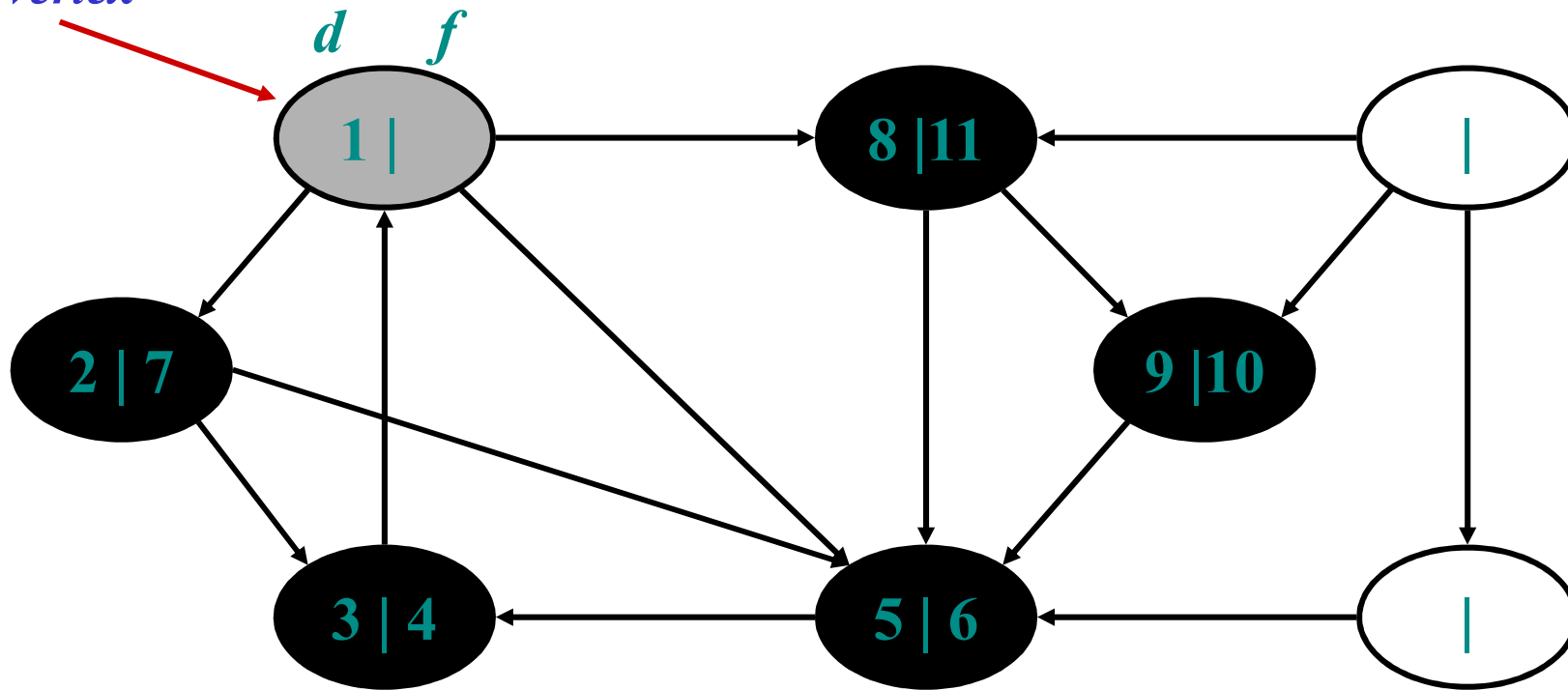


DFS Example



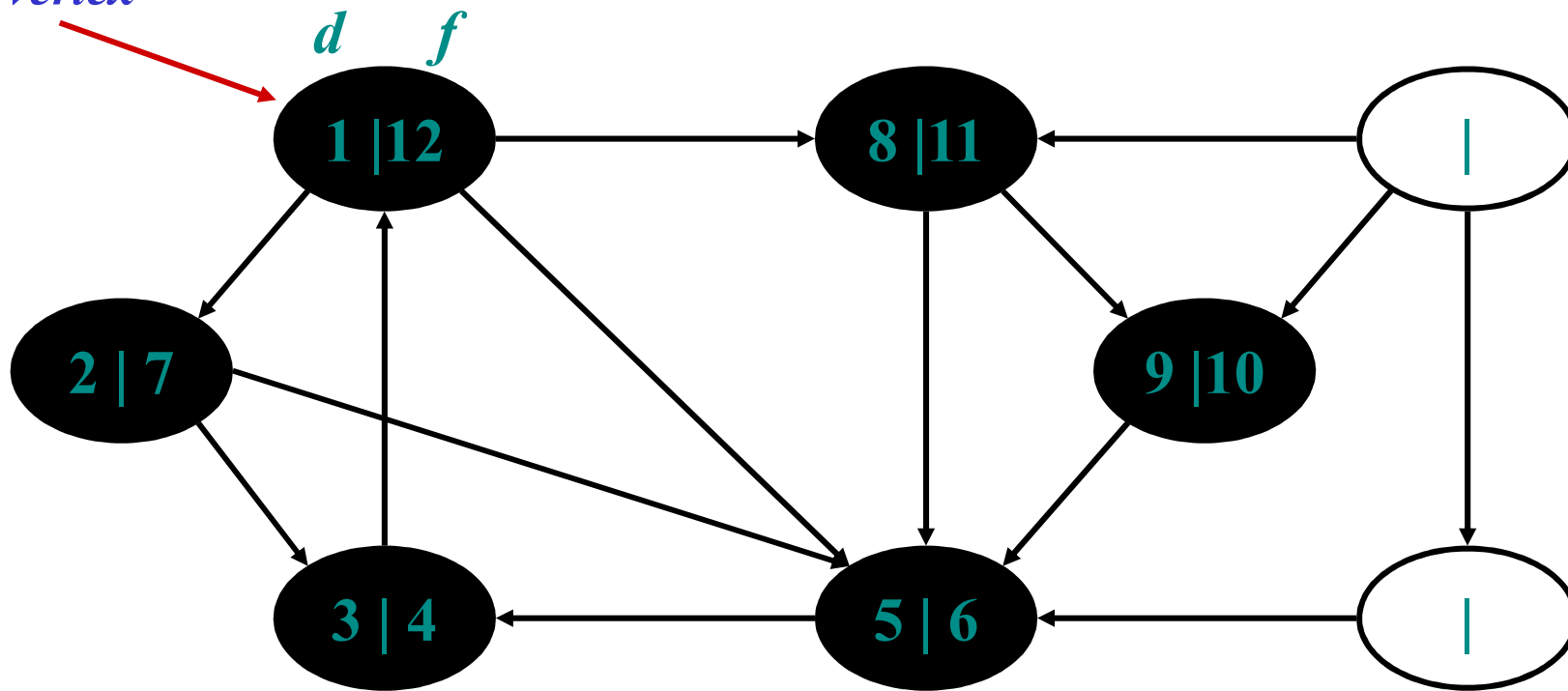
DFS Example

*source
vertex*



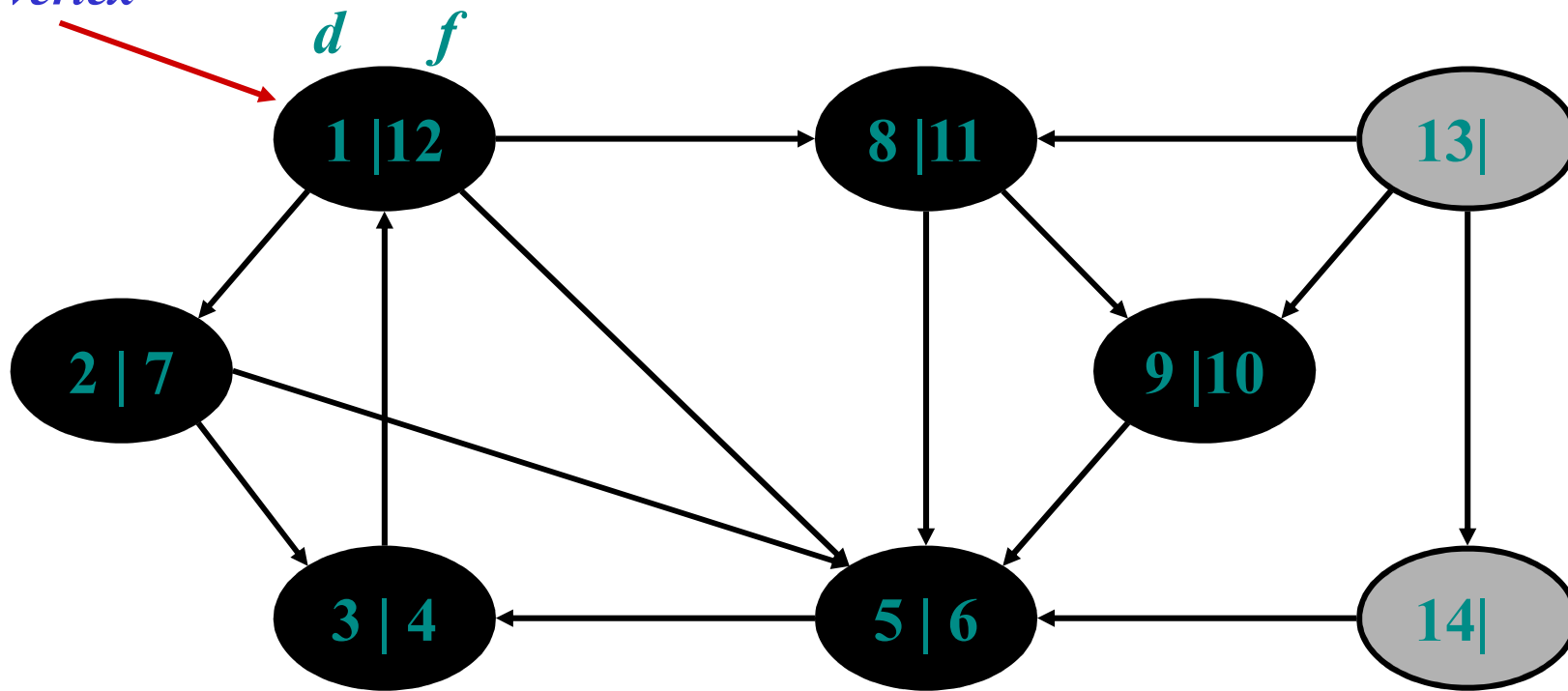
DFS Example

*source
vertex*



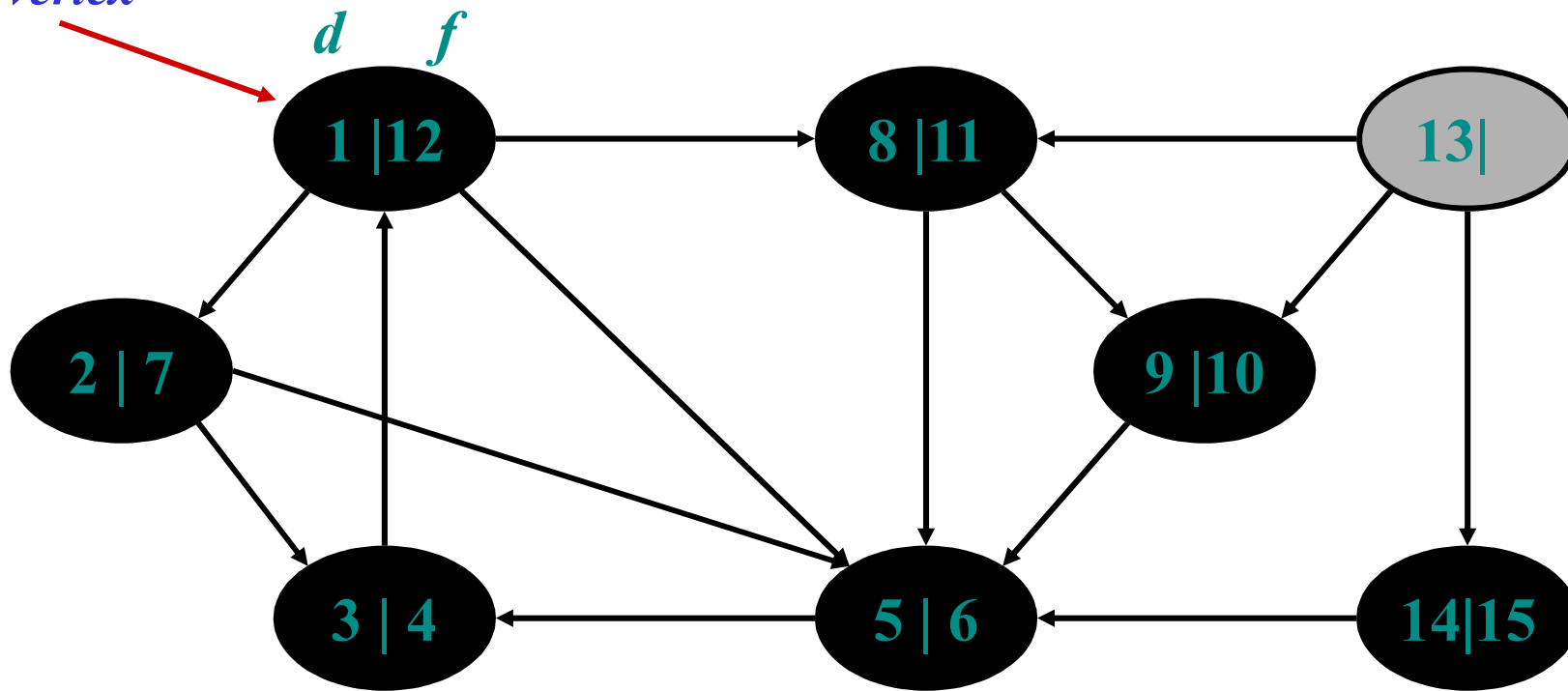
DFS Example

*source
vertex*



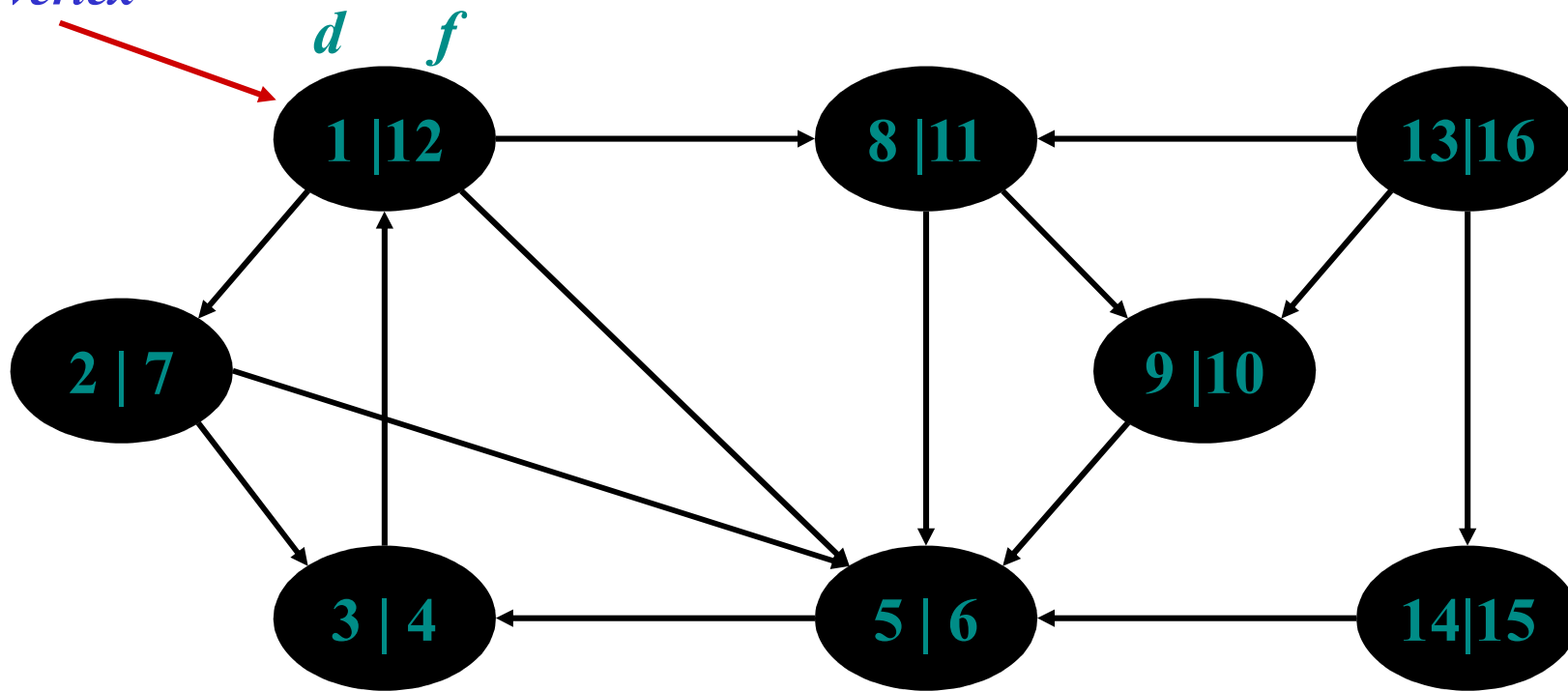
DFS Example

*source
vertex*



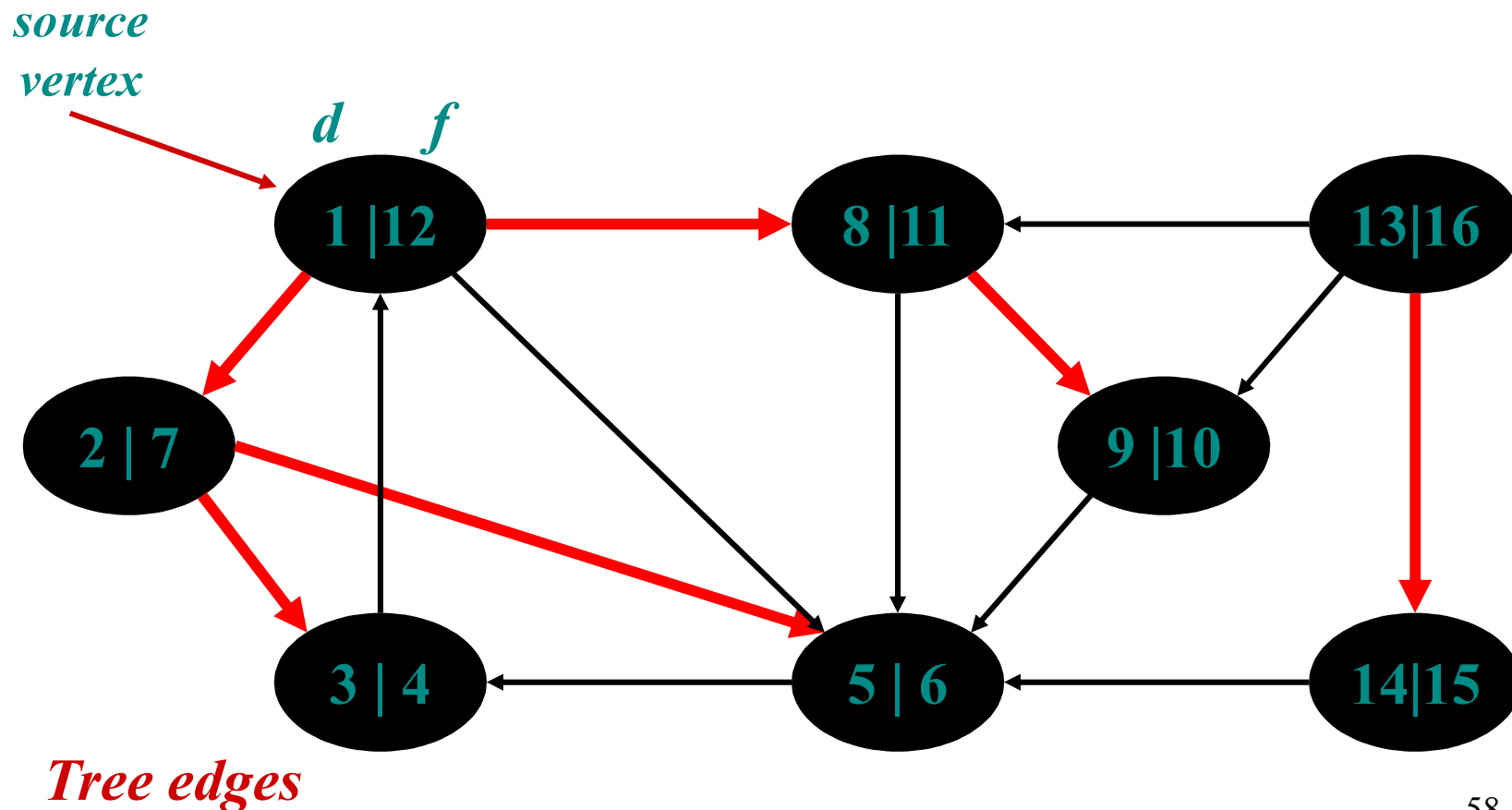
DFS Example

*source
vertex*



Properties of DFS

- Vertices traversed in search form depth-first forest of trees.



Properties of DFS

- Discovery and finishing times have parenthesis structure.
 - u is an ancestor of v if and only if $[d[u], f[u]] \supseteq [d[v], f[v]]$
 - u is a descendent of v if and only if $[d[u], f[u]] \subseteq [d[v], f[v]]$
 - u and v are not related if and only if $[d[u], f[u]]$ and $[d[v], f[v]]$ are disjoint

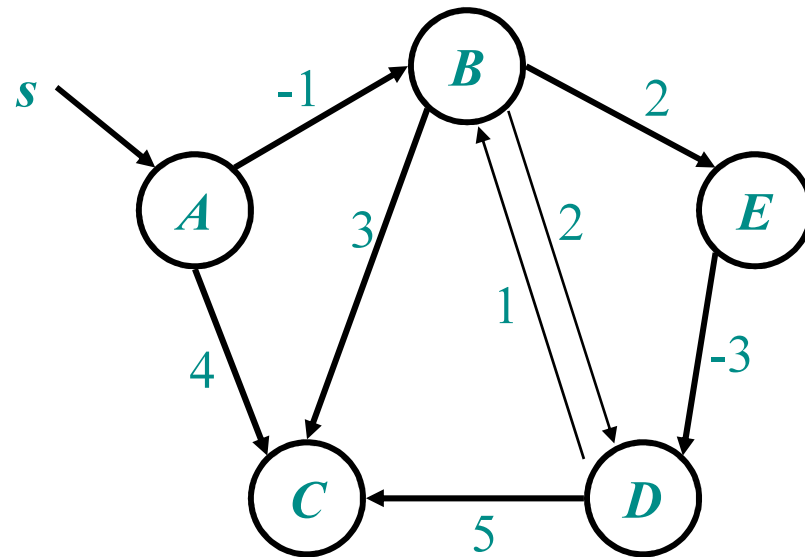
Negative-weight cycles

- If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.
- **Bellman-Ford algorithm:** Finds all shortest-path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Bellman-Ford Algorithm

- $d[s] \leftarrow 0$
 - **for** each $v \in V - \{s\}$
 - do** $d[v] \leftarrow \infty$
 - **for** each $i \leftarrow 1$ to $|V| - 1$
 - do for** each edge $(u, v) \in E$
 - do if** $d[v] > d[u] + w(u, v)$
 - then** $d[v] \leftarrow d[u] + w(u, v)$
 - **for** each edge $(u, v) \in E$
 - do if** $d[v] > d[u] + w(u, v)$
 - then** report that a negative-weight cycle exists
- Time = $O(VE)$

Example of Bellman-Ford



work on board

Correctness

- **Theorem** If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.
- **Corollary** If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in G reachable from s .