GRAPH ALGORITHMS

CSX3009 DATA STRUCTURE AND ALGORITHMS

KWANKAMOL NONGPONG, PH.D.

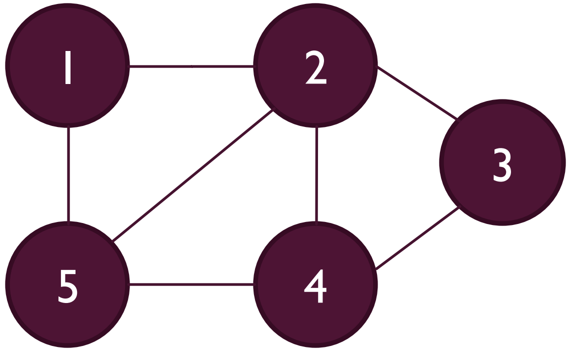
REPRESENTATION OF GRAPHS

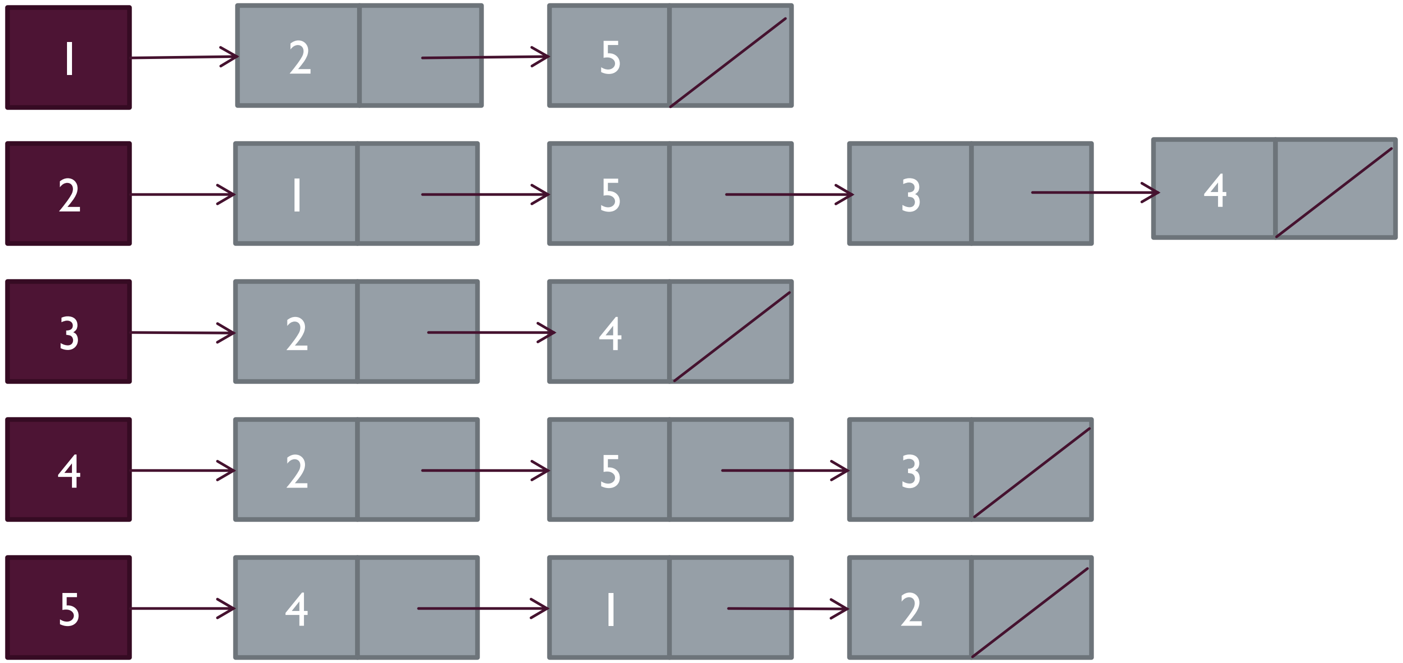
- Recall that a graph G = (V, E)
- Collection of adjacency lists
  - a compact way to represent sparse graphs.
  - $|E| << |V|^2$
- Adjacency matrix
  - Preferred when the graph is dense.
  - $|E|$ is close to $|V|^2$

ADJACENCY-LIST REPRESENTATION

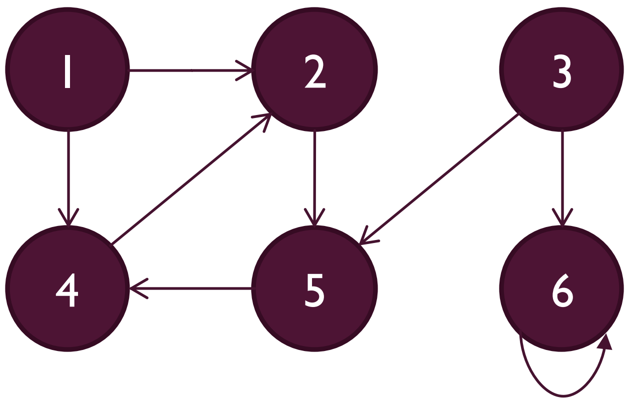
- An array *adj* of |V| lists, one for each vertex in V.
- For each u ∈ V, *adj*[u] contains pointers to all vertices v such that an edge (u, v) ∈ E.
  - *adj*[u] consists of all vertices adjacent to u in G.

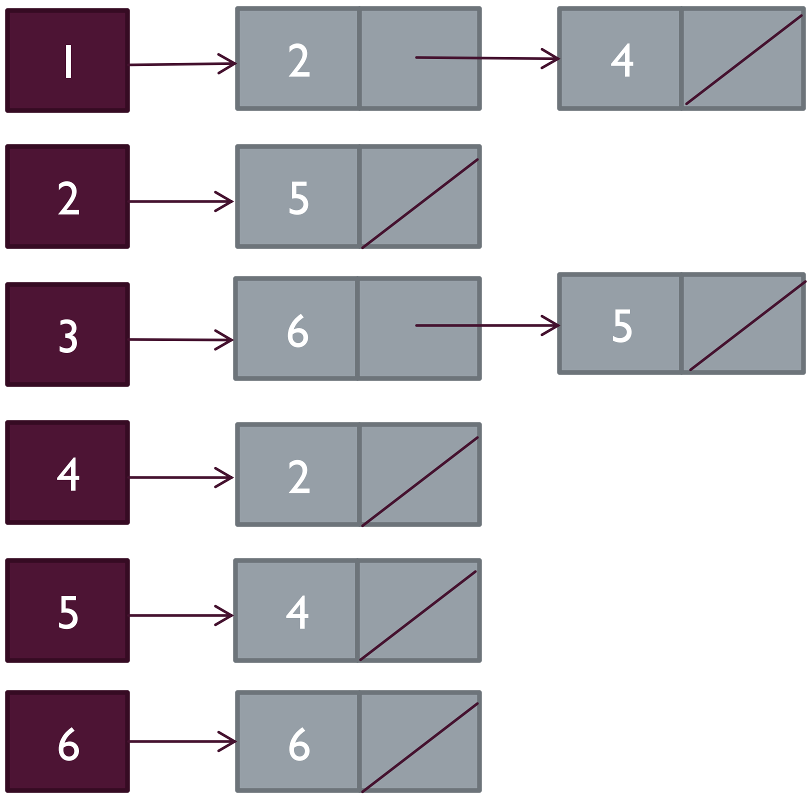
EXAMPLE: UNDIRECTED GRAPH





EXAMPLE: DIRECTED GRAPH



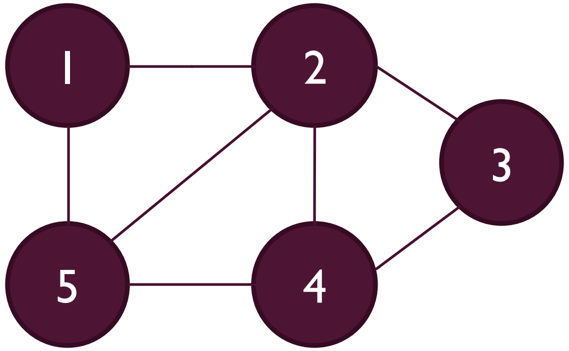


ADJACENCY MATRIX

- Assume that the vertices are numbered 1, 2, 3, …, |V|
- |V| x |V| matrix
- A = ($a_{i\,j}$) such that

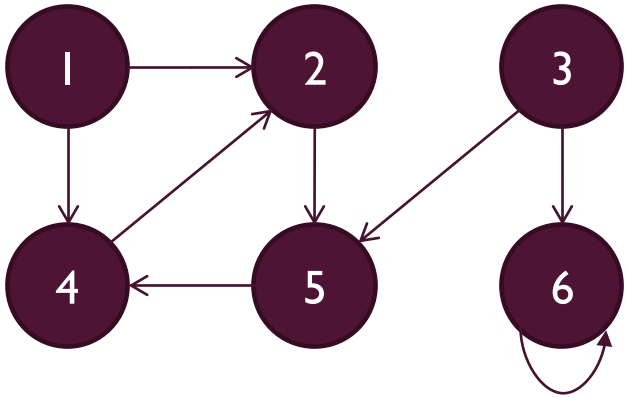
$$a_{i\,j} = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases}$$

EXAMPLE: UNDIRECTED GRAPH



	1	**2**	**3**	**4**	**5**
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

EXAMPLE: DIRECTED GRAPH



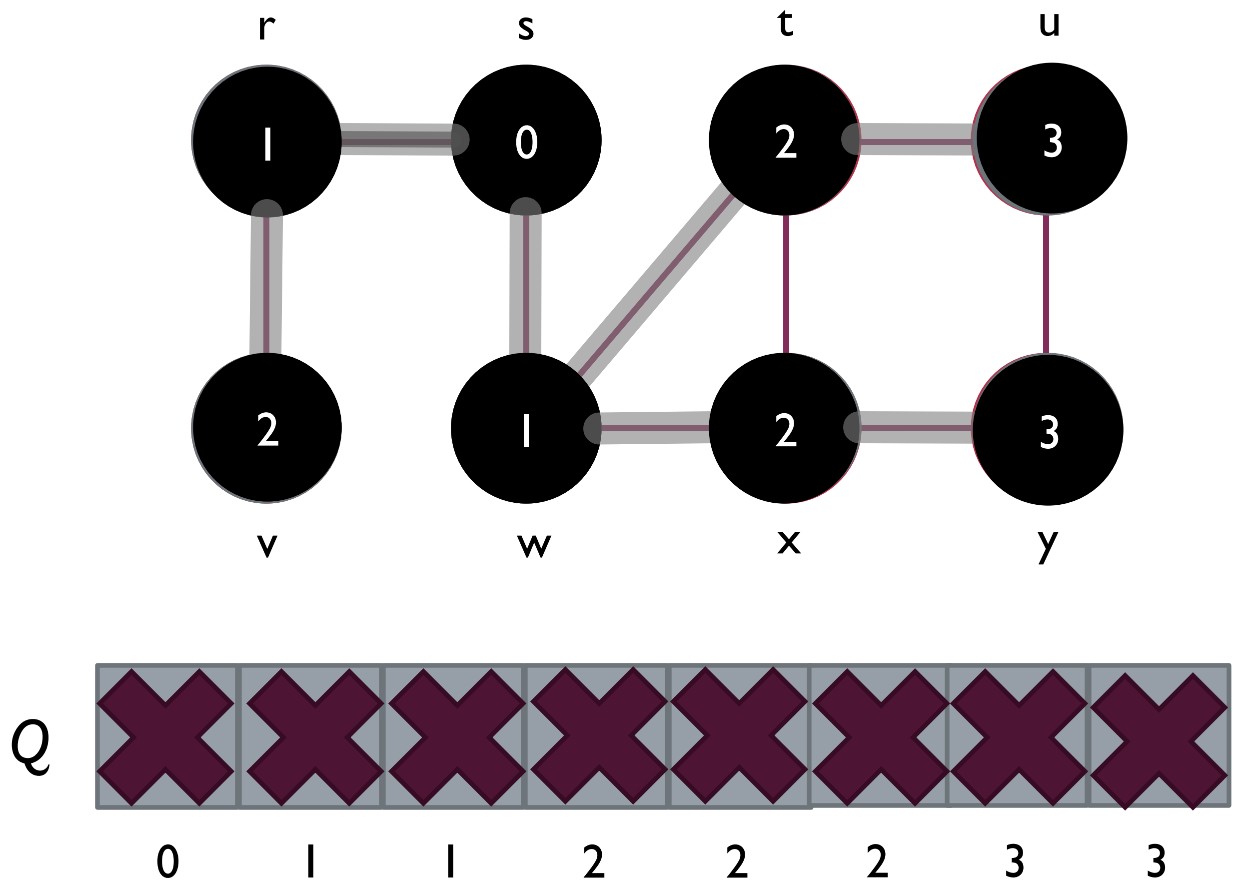
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

BREADTH-FIRST SEARCH

- Given a graph G = (V, E) and a distinguished source vertex *s*, BFS systematically explores the edges of G to discover every vertex that is reachable from s.
- It computes the distance (fewest number of edges) from *s* to all reachable vertices.
- It also produces a breadth-first tree with the root *s* that contains all reachable vertices.

- Which data structure should be used in BFS?

EXAMPLE: BREADTH-FIRST SEARCH



ALGORITHM: BREADTH-FIRST SEARCH

BFS(*G*, *s*)

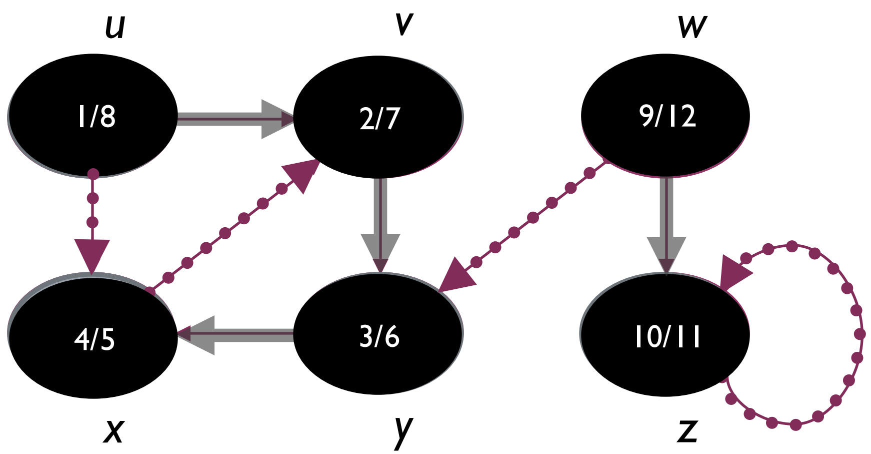
```
for each vertex u ∈ V(G) – {s}
    do color[u] ← WHITE
        d[u] ← ∞
        π[u] ← NIL
color[s] ← GRAY
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø
    do u ← head[Q]
        for each v ∈ Adj[u]
            do if color[v] = WHITE
                then color[v] ← GRAY
                    d[v] ← d[u] + 1
                    π[v] ← u
                    ENQUEUE(Q, v)
        DEQUEUE(Q)
        color[u] ← BLACK
```

DEPTH-FIRST SEARCH

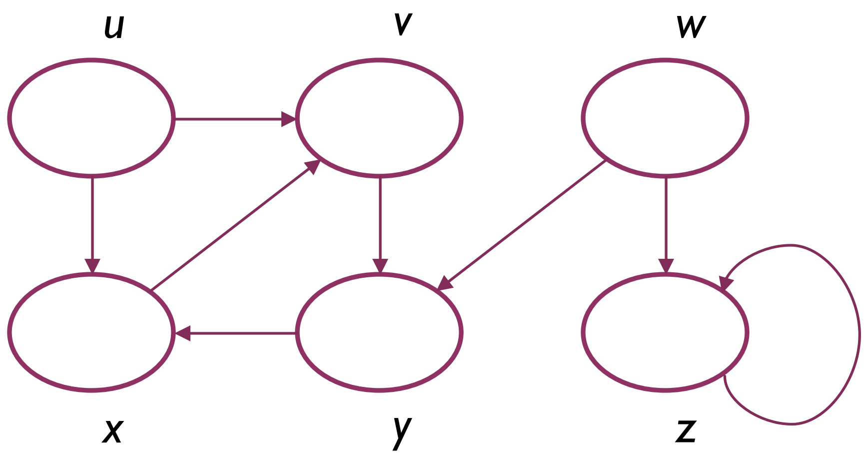
- Unlike BFS, DFS searches deeper in the graph whenever possible.
- In DFS, edges are explored out of the most recently discovered vertex v that still has unexplored (outgoing) edges.
- When all of v's edges have been explored, the search backtracks to explore edges leaving the vertex from which v was discovered.
- The process continues until all vertices that are reachable from the original source have been discovered.

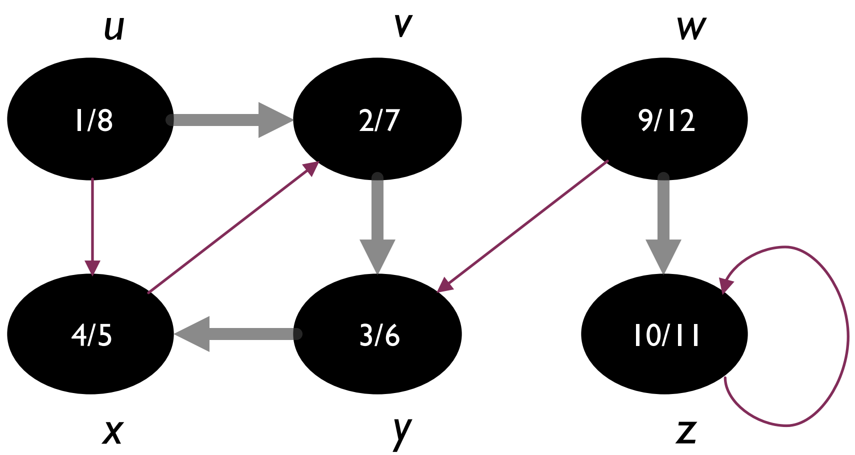
- Which data structure should be used in DFS?

EXAMPLE: DEPTH-FIRST SEARCH



EXAMPLE: DEPTH-FIRST SEARCH





ALGORITHM: DEPTH-FIRST SEARCH

DFS(G)

1. for each vertex $u \in V(G)$
2. do color[u] $\leftarrow$ WHITE
3. $\pi[u] \leftarrow$ NIL
4. time $\leftarrow$ 0
5. for each vertex $u \in V(G)$
6. do if color[u] = WHITE
7. then DFS-Visit(u)

DFS-Visit(u)

1. Color[u] $\leftarrow$ GRAY
2. d[u] $\leftarrow$ time $\leftarrow$ time + 1
3. For each $v \in Adj(u)$
4. do if color[v] = WHITE
5. then $\pi[v] \leftarrow u$
6. DFS-Visit(v)
7. color[u] $\leftarrow$ BLACK
8. f[u] $\leftarrow$ time $\leftarrow$ time + 1

APPLICATIONS OF BFS

APPLICATIONS OF BFS

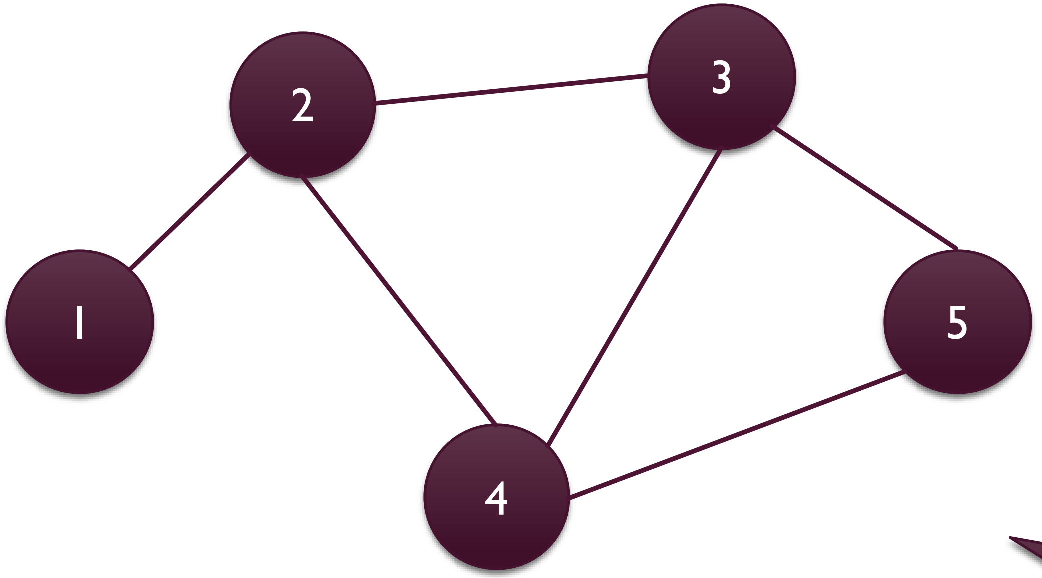
- Shortest paths
- Minimum spanning trees
- Crawlers in search engines
- Broadcasting in the network
- Social networks
  - Find people within the given distance *k*
- and etc.

SHORTEST PATH

SHORTEST PATH

- Shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.
- Given a weighted graph G = (V, E) and a source vertex v
- Goal
  - Find shortest paths from a source vertex v to all other vertices in the graph.

PATH

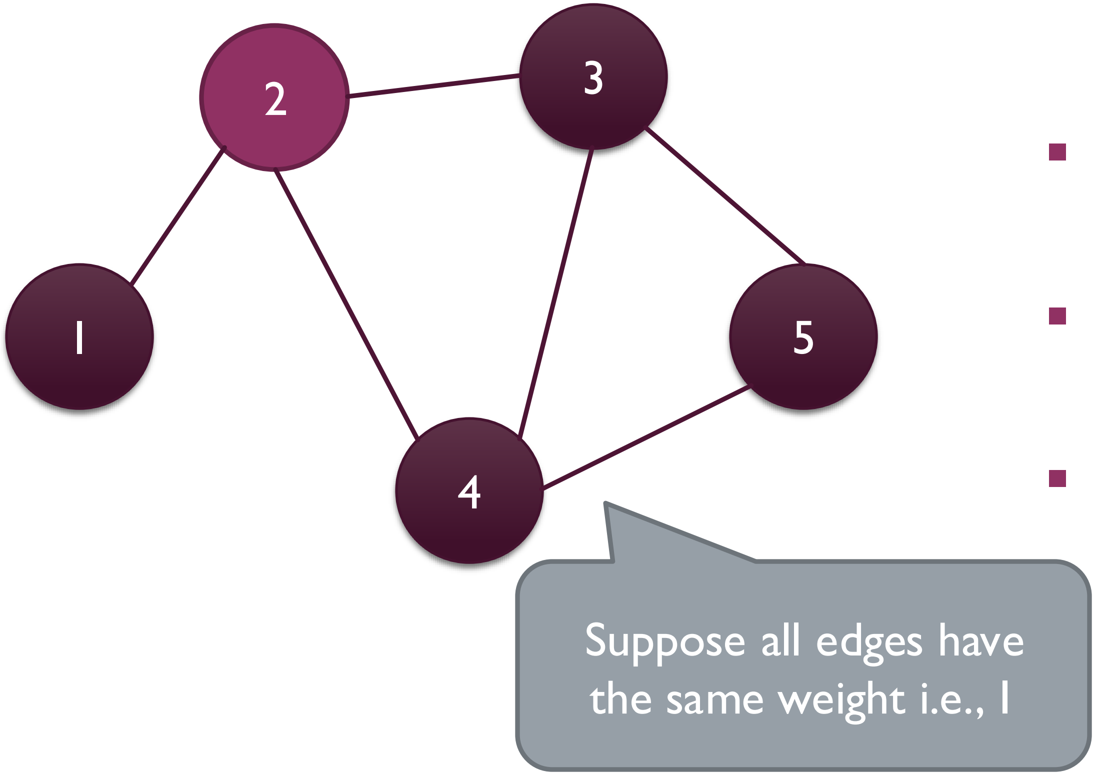


Is there a path from 1 to 4?

(1, 2, 3, 5, 4)

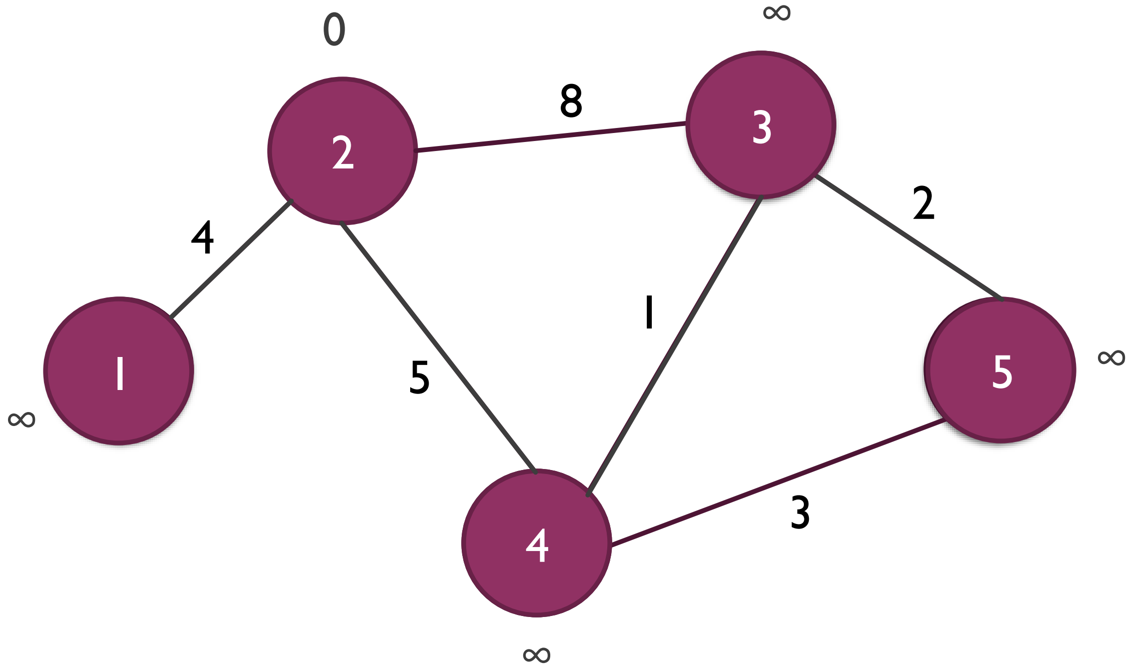
(1, 2, 4)

SHORTEST PATH

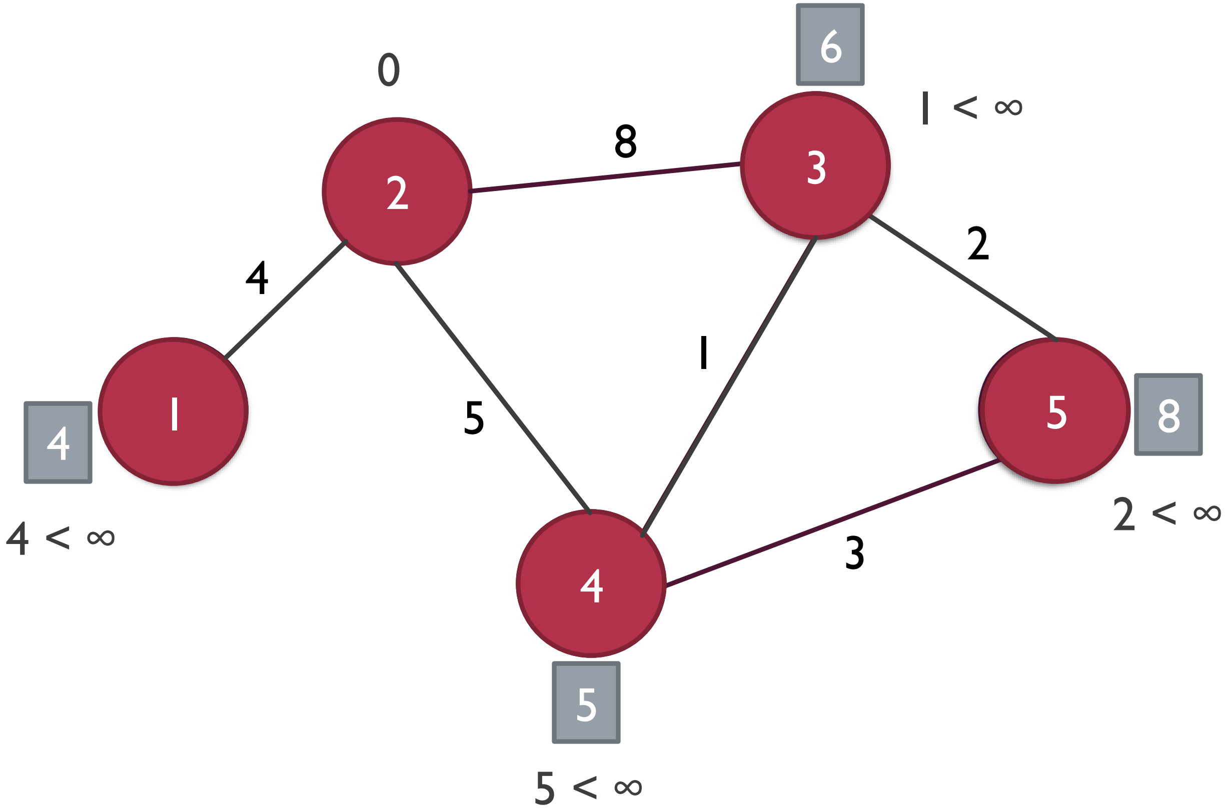


- Shortest path to 1
  - (2, 1)
- Shortest path to 3
  - (2, 3)
- Shortest path to 4
  - (2, 4)
- Shortest path to 5
  - (2, 3, 5) or (2, 4, 5)

EXAMPLE



ALGORITHM DEMONSTRATION



DIJKSTRA'S ALGORITHM: SHORTEST PATH

```
for each node i to n
    distance[i] = ∞
    previous[v] = undefined
for each node i to n
     visited[i] = false
distance[s] = 0
current = s
Q = set of all nodes in the graph G
```

Let s be the source node

```
while Q is not empty
    u = vertex in Q with smallest distance in distance[]
    Q = Q – {u}
    if distance[u] = ∞
        break;
    endif
    for each neighbor v of u
        alternative = distance[u] + distance_between(u, v)
        if alternative < distance[v]
            distance[v] = alternative
            previous[v] = u
       endif
    endfor
 endwhile
 return distance
```

ALGORITHM: ALL-PAIR SHORTEST PATH

(Floyd-Warshall)

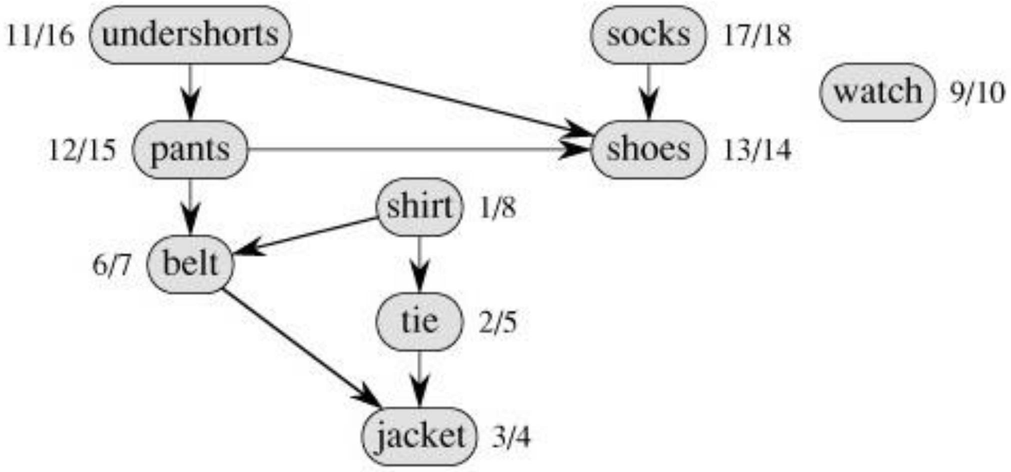
```
for k := 1 to n
    for i := 1 to n
        for j := 1 to n
            if path[i][k] + path[k][j] < path[i][j]
                path[i][j] := path[i][k]+path[k][j];
                next[i][j] := k;
            endif
function Path (i,j)
    if path[i][j] is infinity
        return "no path";
    int intermediate := next[i][j];
    if intermediate is null then 1
        return " "; /* there is an edge from i to j, with no vertices between */
    else
        return Path(i, intermediate) + intermediate + Path(intermediate, j);
```

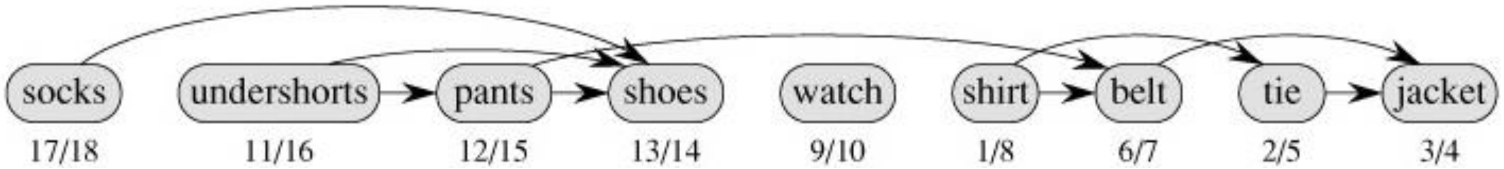
APPLICATIONS OF DFS

TOPOLOGICAL SORT

- Topological sort of directed acyclic graphs (DAGs) G = (V, E)
  - Linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.
  - Can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.

EXAMPLE: TOPOLOGICAL SORT





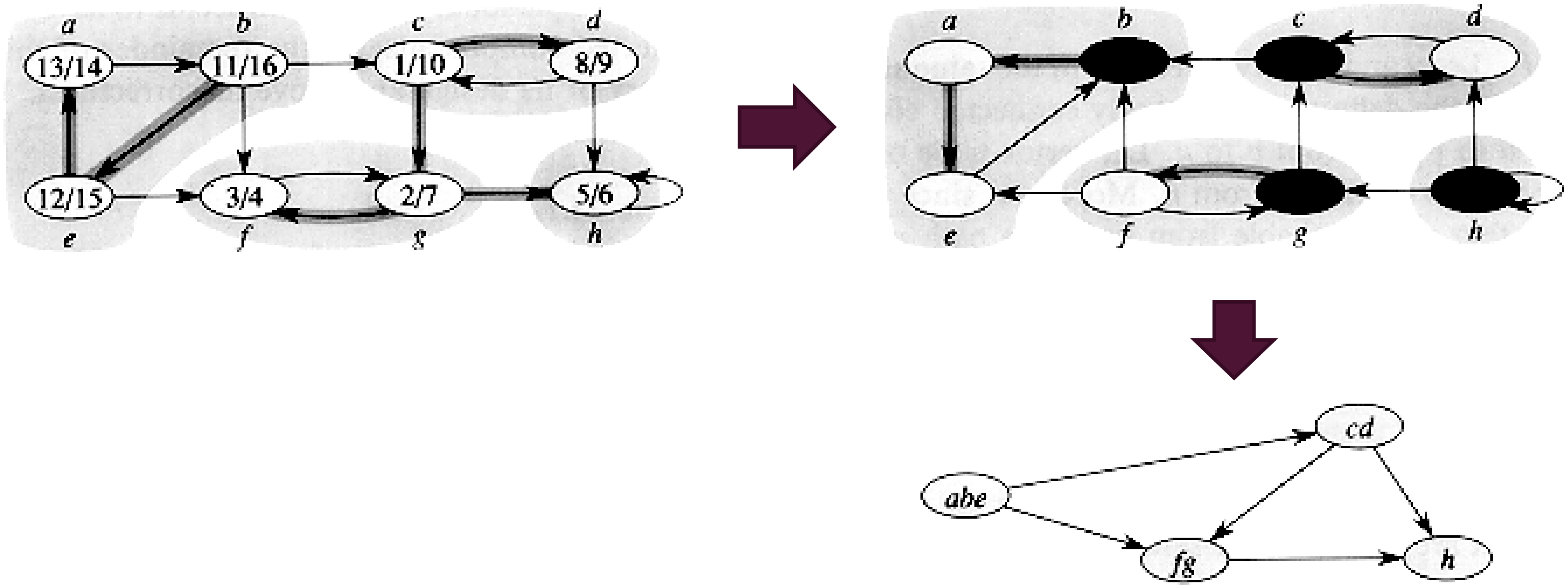
ALGORITHM: TOPOLOGICAL SORT

Topological-Sort(G)

1. Call DFS(G) to compute finishing times f[v] for each vertex v
2. As each vertex is finished, insert in onto the front of a linked list
3. Return the linked list of vertices

STRONGLY CONNECTED COMPONENTS

- A strongly connected component of a directed graph G = (V, E) is a maximal set of vertices U V such that for every pair of vertices u and v in U, the vertices u and v are reachable from each other.



BASIC CONCEPTS OF TREE SEARCH

Breadth-First Search(G, A)

```
s = A
while not Goal(s)
    for each successor x of s
        enqueue(x)
    s = dequeue()
```

Depth-First Search(G, A)

```
s = A
while not Goal(s)
    for each successor x of s
        push(x)
    s = pop()
```

EXERCISES

- Implement an algorithm that performs a breadth-first search on a given graph and a source node.
- Implement an algorithm that performs a depth-first search on a given graph.
- Give an efficient algorithm to determine if an undirected graph is bipartite.
  - Sample graphs