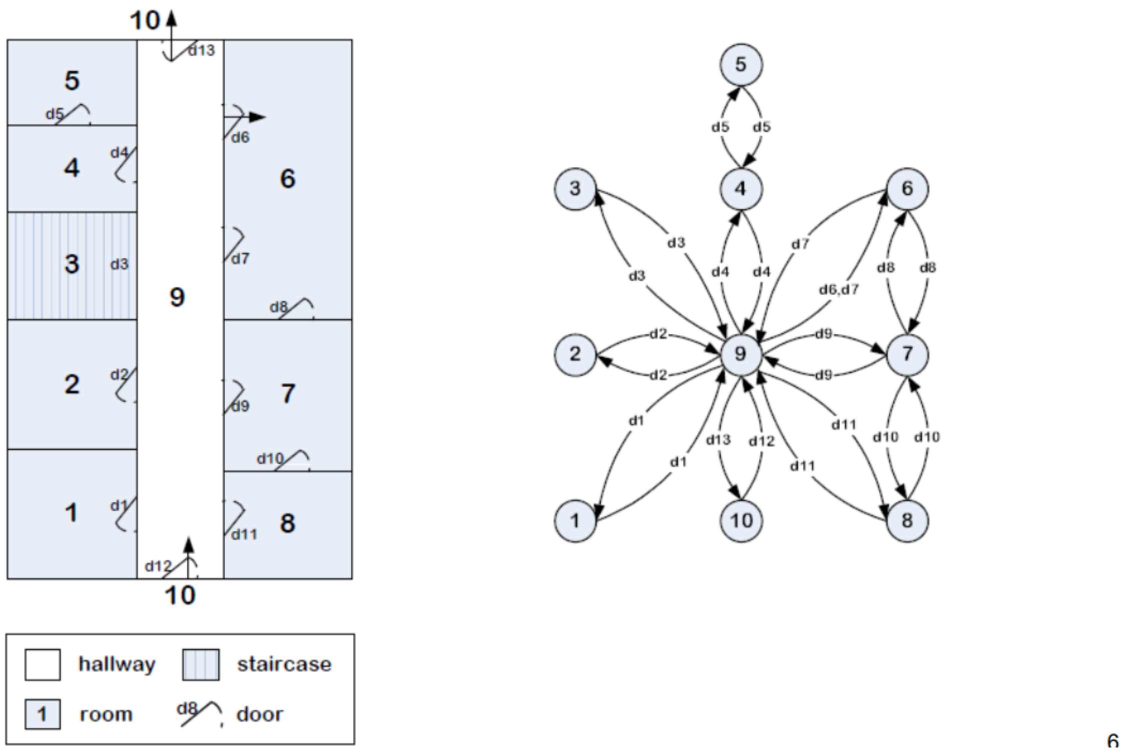
Lek10 - Elementary Graph Algorithms

Graphs, DAG, BFS, DFS, Topological sort.

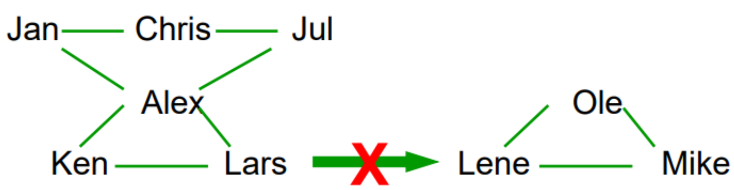
Graphs

- Applications with graphs
  - Road network: road intersections-> vertices, roads -> edges.
    - What is the shortest path from AAU to Aalborg airport?
  - Web: web pages->vertices, hyperlinks -> edges.
    - Which page is the most important page?
- Indoor space: rooms -> vertices, doors -> edges



Social network

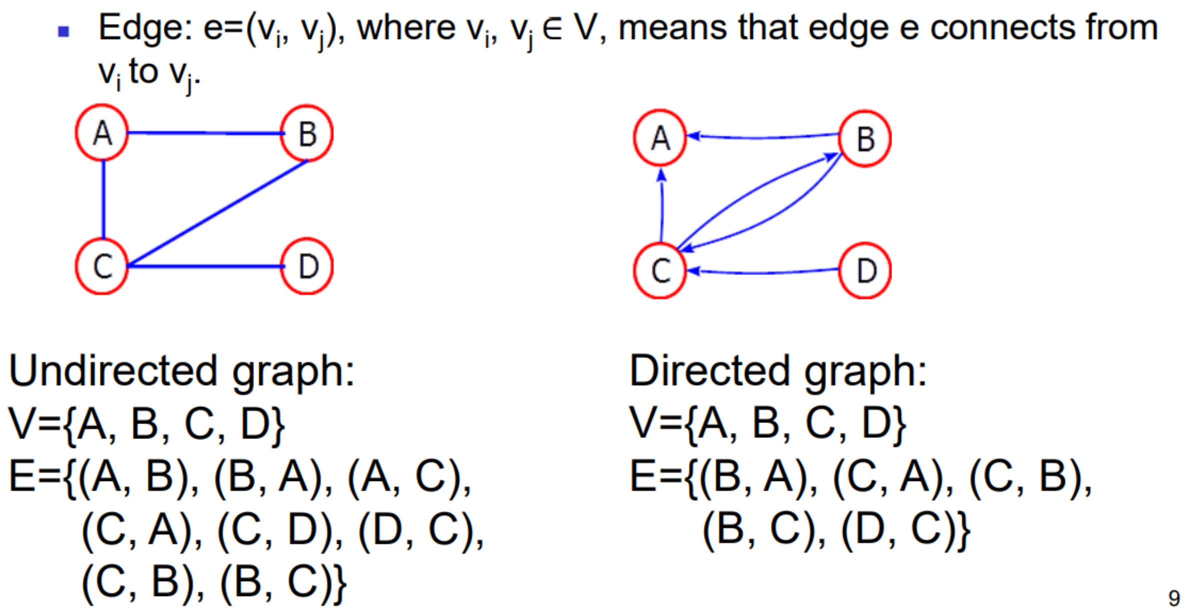
- Friends in Facebook
  - Jan-Chris; Chris-Jul; Ken-Alex; Ole-Mike; Lene-Ole; Lene-Mike; Alex-Jan; Alex-Jul; Alex-Lars; Ken-Lars;
- Are Lars and Lene friends in Facebook?
- We can answer it by representing friendship in a graph



- The answer is NO, because they are not connected in their friendship graph.

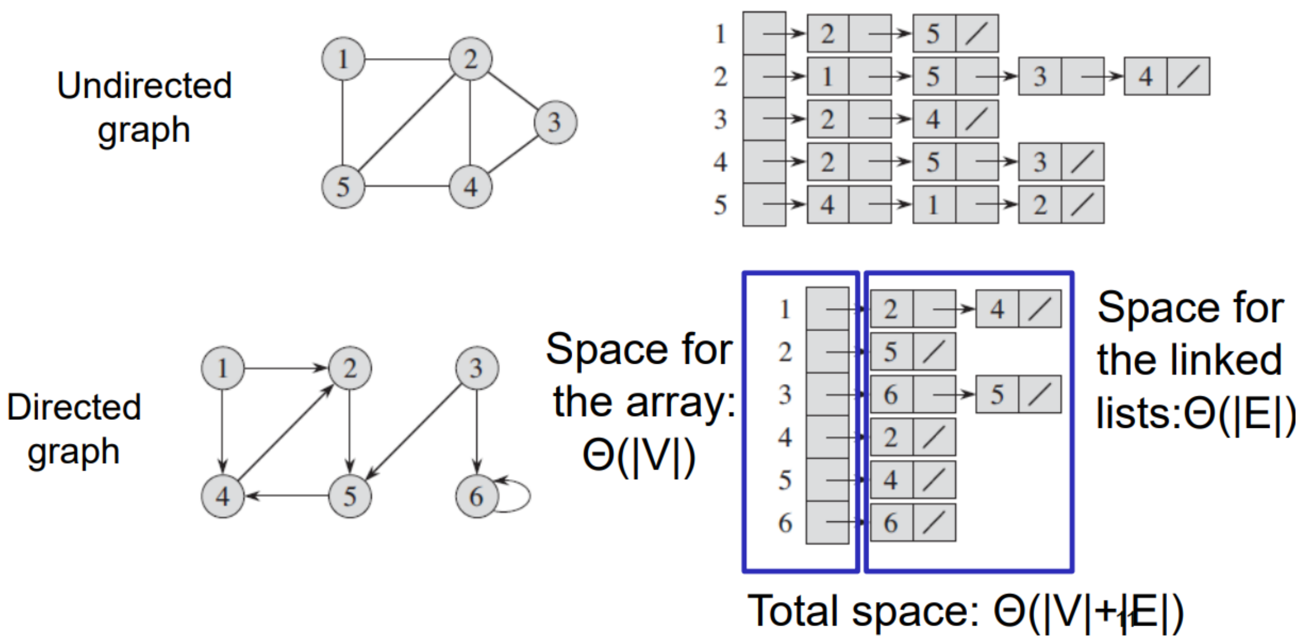
Definitions

- A graph G = (V,E) is composed of:
  - V: the set of **vertices**.
  - E: the set of **edges**.
- An **edge e** connects two vertices.



Adjacency list representation

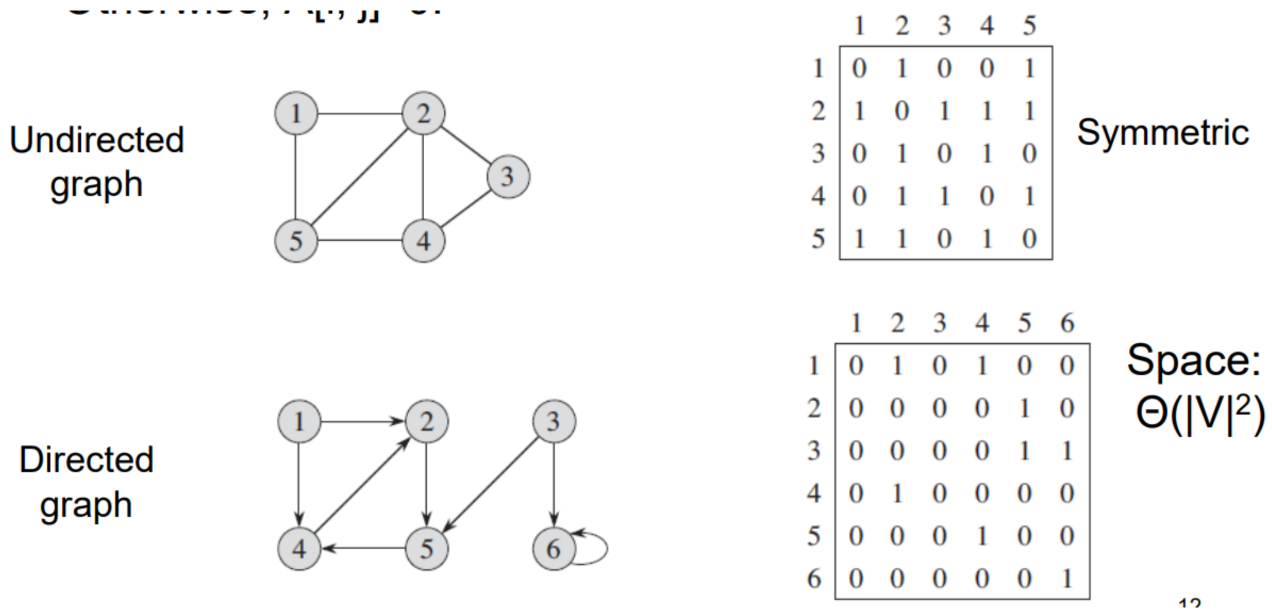
- The adjacency list representation of a graph G=(V, E) consists of an array of |V| lists, one for each vertex in V.
  - The adjacency list of a vertex v contains all vertices u such that (v, u) ∈ E.



Adjacency matrix

- Matrix A with entries for all pairs of vertices.
- In other words, matrix A is with size |V|*|V|.

- If there is an edge (vi, vj), A[i, j]=1.
- Otherwise, A[i, j]=0.



How to choose a representation

- Two representations:
  - A collection of adjacency lists
  - An adjacency matrix
- When we have a sparse graph, adjacency list representation provides a compact way.
  - Sparse graph: |E| is much less than |V|*|V|.
  - Space of adjacency list: Θ(|V|+|E|).
- When we have a dense graph, we may prefer to use adjacency matrix representation.
  - Dense graph: |E| is close to |V|*|V|.
  - Space of adjacency matrix: Θ(|V|2).

Dense = verticies are connected to most other notes. Sparse = verticies are only connected to a small part of the total nodes (ex: social network).

Searching a graph

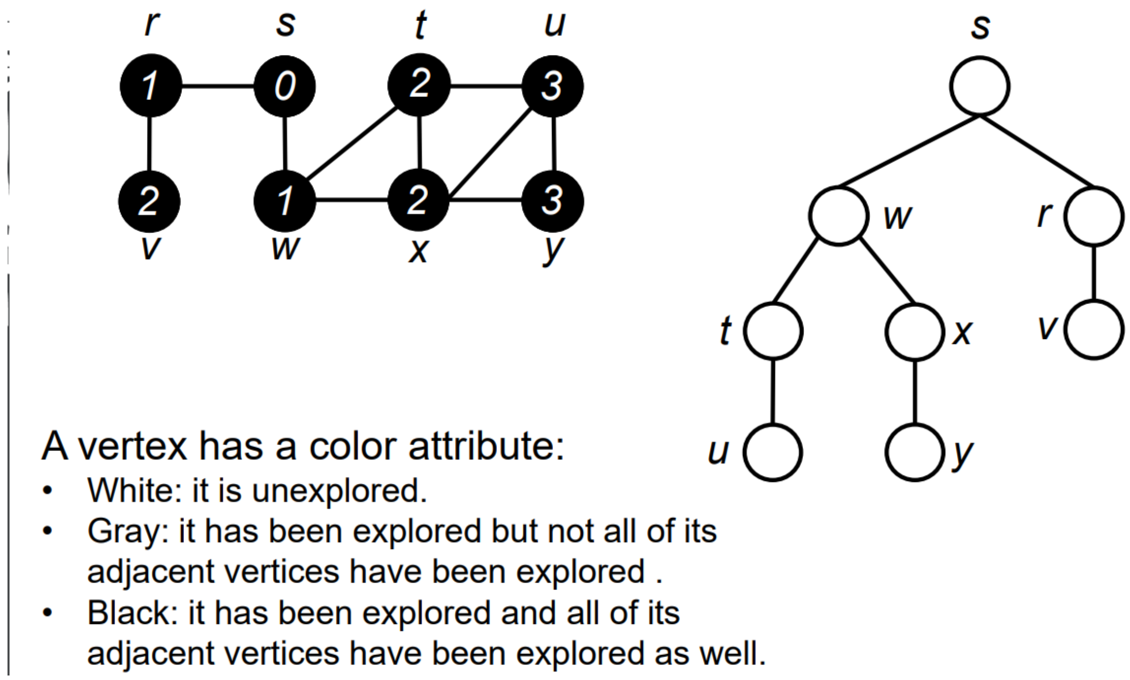
- Searching a graph means systematically following its edges so as to visit its vertices.
  - Can discover the structure of a graph.
  - Many algorithms begin by searching their input graph to obtain the structure information.
  - Searching a graph lies at the heart of the field of graph algorithms.
- Two search algorithms
  - Breadth-first search
  - Depth-first search

Breadth-first search (BFS)

- Input
  - A graph G=(V, E) and a source vertex **s**
- Aim

    - Systematically discovers **every** vertex that is reachable from **s**.
- Output
    - The distance from **s** to each reachable vertex.
        - Distance = the smallest number of edges (unweighted graph)
    - A breadth-first tree with root s that contains all reachable vertices.
- What does BFS mean?
    - It discovers all vertices at distance k from **s** before discovering any vertices at distance k+1.

Intuition of BFS



explore one level at a time.

BFS Algorithm

- Before showing the algorithm, we need to define the following attributes to a vertex.
- Color attribute:
    - White: it is unexplored.
    - Gray: it has been explored but not all of its adjacent vertices have been explored.
    - Black: it has been explored and all of its adjacent vertices have been explored as well.
- A vertex has a distance attribute:
    - The distance to the source s.
- A vertex has a parent attribute:
    - It records the vertex that is its parent in the breadth-first tree.

```
BFS(G,s)
for each vertex a ∈ G.V()
    a.setcolor(white)
    a.setd(∞)
    a.setparent(NIL)
s.setcolor(gray)
s.setd(0)
Q.init()
Q.enqueue(s)
while not Q.isEmpty()
    a ← Q.dequeue()
    for each b ∈ a.adjacent() do
        if b.color() = white then
            b.setcolor(gray)
            b.setd(a.d() + 1)
            b.setparent(a)
            Q.enqueue(b)
    a.setcolor(black)
```

Θ(|V|) — Initialize all vertices

Constant time

O(|E|)

Each vertex *a:*

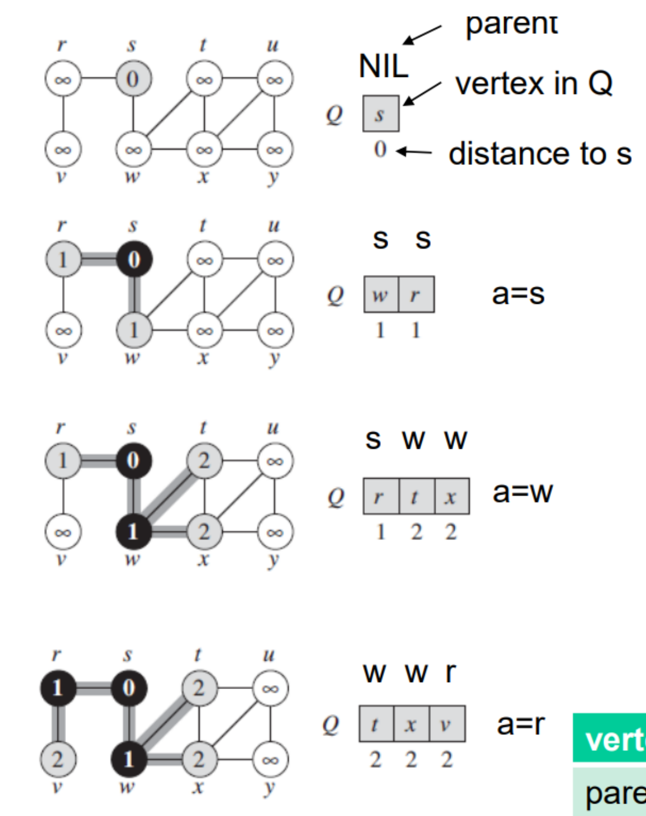
- De-(en-)queued ***at most*** once: constant time O(1) (total O(|V|))
- Its adjacency list is scanned, and the for loop executes:
  - |a.adjacent()| times,
  - Θ(|a.adjacent()|).

In total, O(|V|+|E|).

Edges among all vertices in V: $\sum_{a \in V} |a.adjacent()| = |E|$

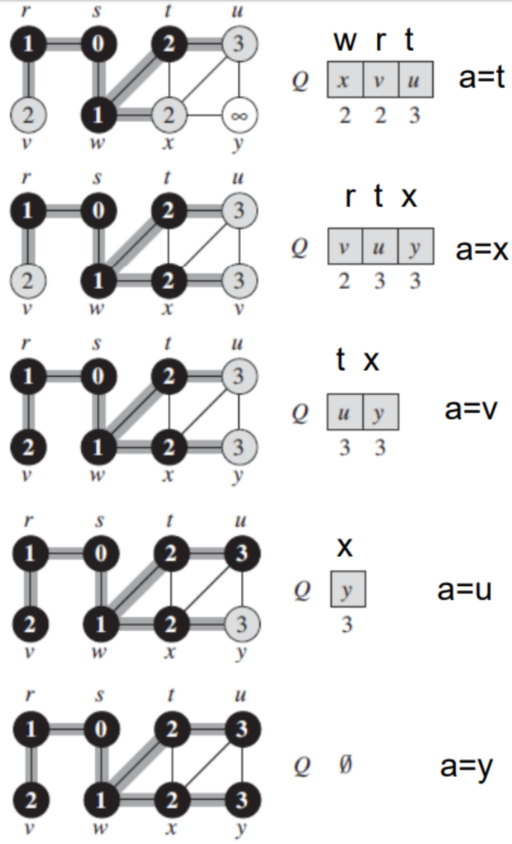
26

A running example




```
BFS(G,s)
for each vertex a ∈ G.V()
    a.setcolor(white)
    a.setd(∞)
    a.setparent(NIL)
s.setcolor(gray)
s.setd(0)
Q.init()
Q.enqueue(s)
while not Q.isEmpty()
    a ← Q.dequeue()
    for each b ∈ a.adjacent() do
        if b.color() = white then
            b.setcolor(gray)
            b.setd(a.d() + 1)
            b.setparent(a)
            Q.enqueue(b)
    a.setcolor(black)
```

vertex	r	s	t	u	v	w	x	y
parent	s	Nil	w		r	s	w	

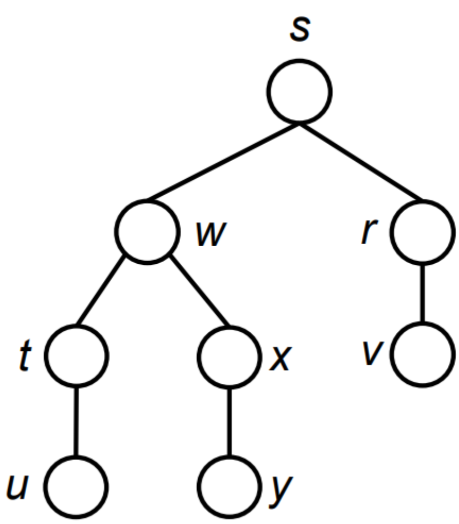


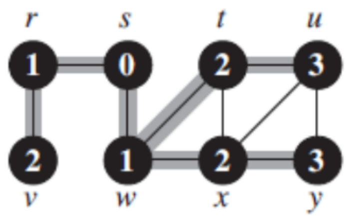

```
BFS(G,s)
for each vertex a ∈ G.V()
    a.setcolor(white)
    a.setd(∞)
    a.setparent(NIL)
s.setcolor(gray)
s.setd(0)
Q.init()
Q.enqueue(s)
while not Q.isEmpty()
    a ← Q.dequeue()
    for each b ∈ a.adjacent() do
        if b.color() = white then
            b.setcolor(gray)
            b.setd(a.d() + 1)
            b.setparent(a)
            Q.enqueue(b)
    a.setcolor(black)
```

vertex	r	s	t	u	v	w	x	y
parent	s	Nil	w	t	r	s	w	x

22

- A breadth-first tree
    - Consists of vertices reachable from s.
    - Contains a unique simple path from s to a vertex v, that is also the shortest path from s to v.



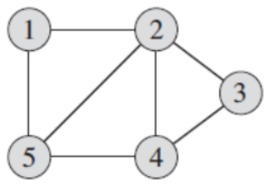


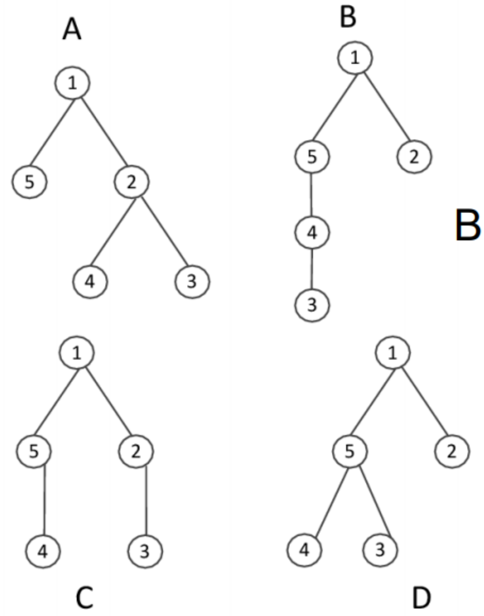
vertex	r	s	t	u	v	w	x	y
parent	s	NIL	w	t	r	s	w	x

Mini quiz

- Is the breadth-first tree unique? Does the breadth-first tree depend on the order in which the neighbor vertices of a given vertex are visited?

- Consider the graph. Which of the following trees cannot be BFS trees?





B and D cannot be BFS trees

BFS Summary

- BFS discovers all vertices that are reachable from a given source vertex s.
- BFS computes the shortest distance to all reachable vertices from s.
- BFS computes a breath-first tree that contains all reachable vertices from s.
- For any vertex v reachable from s, the path in the breadthfirst tree from s to v is a shortest path.

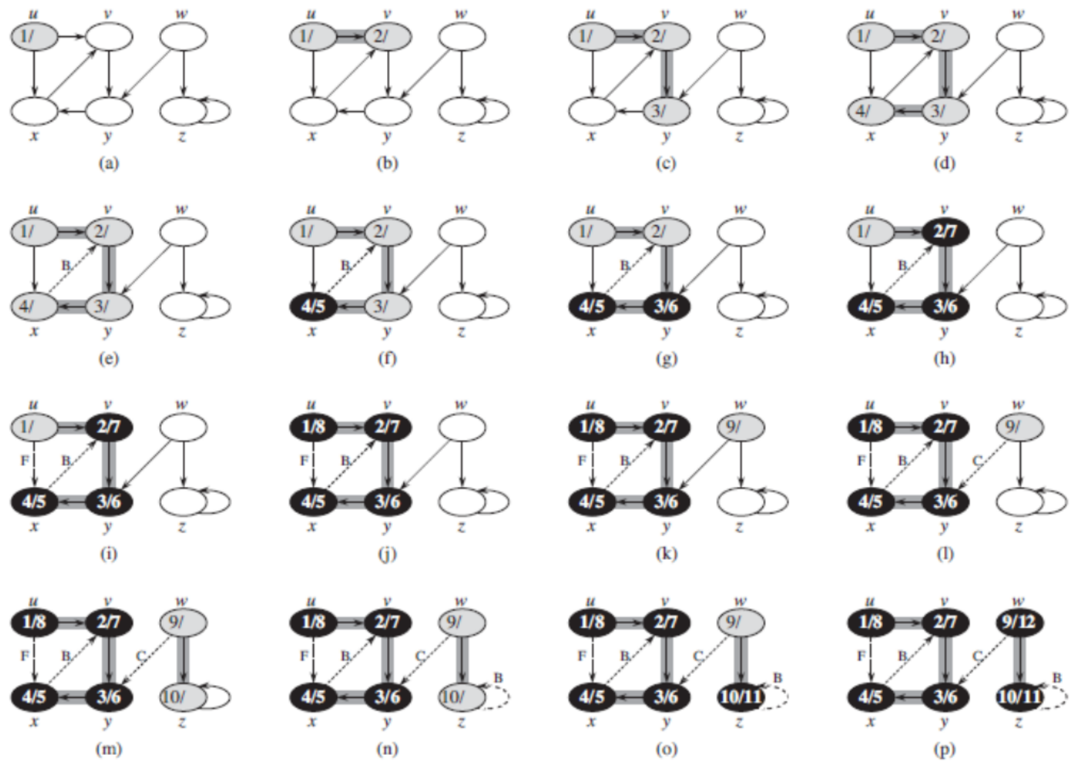
Depth-first search (DFS)

- Input
  - A graph G=(V, E)
- Aim
  - Systematically visit **every** vertex in V.
- Output
  - A depth-first forest that is composed of several depth-first trees.
- What does DFS mean?
  - It search “deeper” in the graph whenever possible.

DFS algorithm - 1

- A vertex has a color attribute:
  - White: unexplored.
  - Gray: it has been explored, but not all of its adjacent vertices have been explored.
  - Black: it has been explored, and all of its adjacent vertices have been explored as well.
- A vertex has a timestamp:
  - v.d: discovery time, i.e., when v is first explored;
  - v.f: finishing time, i.e., when v finishes examining v’s adjacency list;

- A vertex has a parent attribute:
    - It records the vertex that is its parent in the depth-first tree.



Depth-first forest: u -> v -> y -> x og w -> z

```
DFS(G)
for each vertex u ∈ G.V()
    u.setcolor(white)
    u.setparent(NIL)
time ← 0
for each vertex u ∈ G.V()
    if u.color() = white then DFS-Visit(u)
```

Initialize all vertices: Θ(|V|)

DFS-Visit is called ***exactly once*** for each vertex, when it is white: Θ(|V|)

```
DFS-Visit(u)
u.setcolor(gray)
time ← time + 1
u.setd(time)
for each v ∈ u.adjacent()
    if v.color() = white then
        v.setparent(u)
        DFS-Visit(v)
u.setcolor(black)
time ← time + 1
u.setf(time)
```

For each vertex u, the loop executes
- |u.adjacent()| times.

$\sum_{u \in V} |u.adjacent()| = |E|$

Thus, Θ(|V| + |E|)

BFS vs. DFS

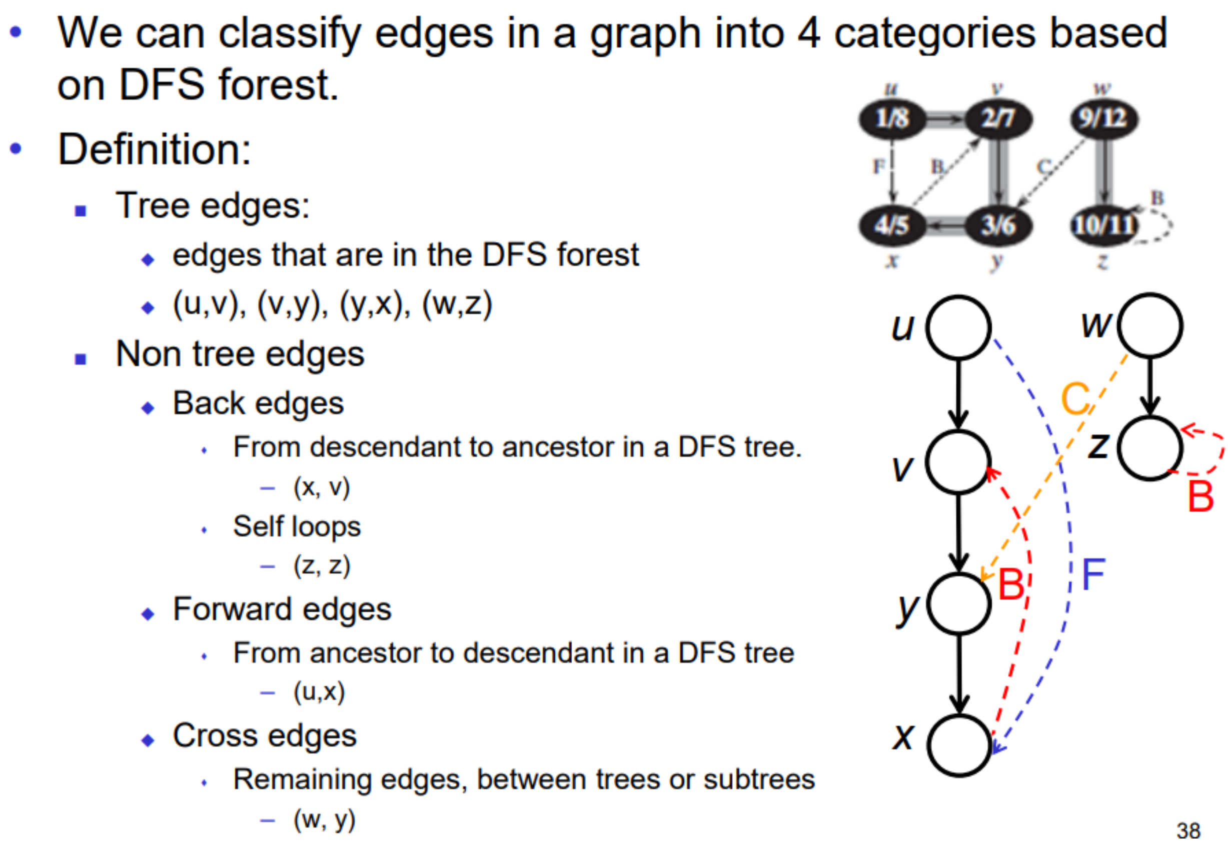
BFS:

- Search from one source.
- Only visit the vertices that are reachable from the source.
- BFS tree.
- Often serves to find shortest paths and shortest path distances.
- O(|V| + |E|)

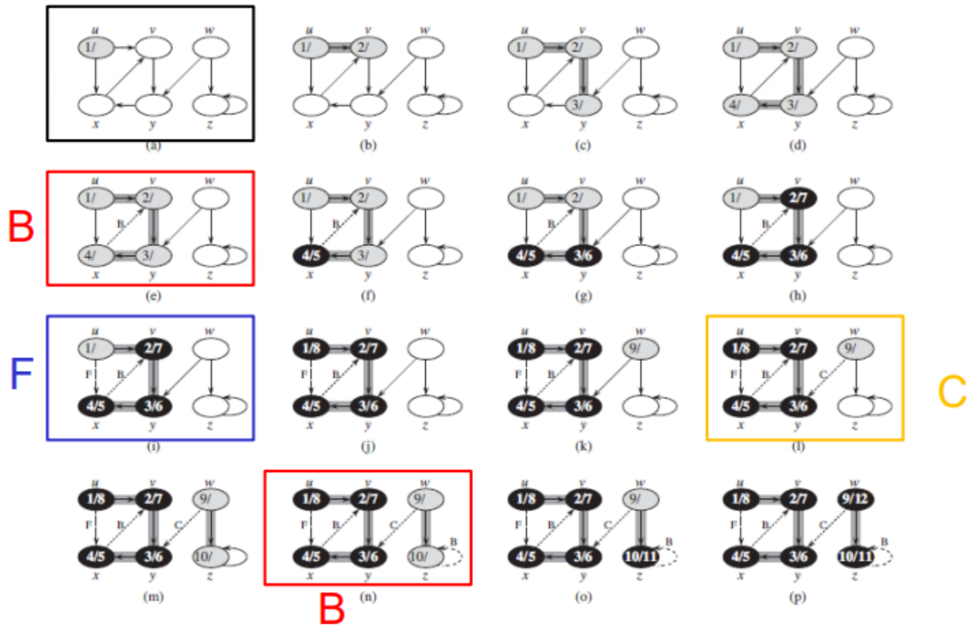
DFS:

- May search from multiple sources.
- Visit every vertex.
- DFS forest.
- Often as a subroutine in another algorithm, e.g.,
  - Classifying edges (we will see it shortly).
  - Topological sort (we will see it shortly).
  - Strongly connected components (next lecture).
- Θ(|V| + |E|)

Edge Classification based on DFS



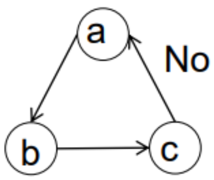
- When exploring an edge (x, y), y's color tells something:
  - If y is white – visit x, then y, edge (x, y) is a tree edge.
  - If y is gray – visit y, later x, then y again, edge (x, y) is a back edge.
  - If y is black, edge (x, y) is a forward or cross edge.

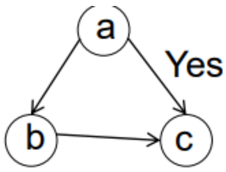


39

DAG: Directed Acyclic Graph

- A DAG is a directed graph with no cycles.

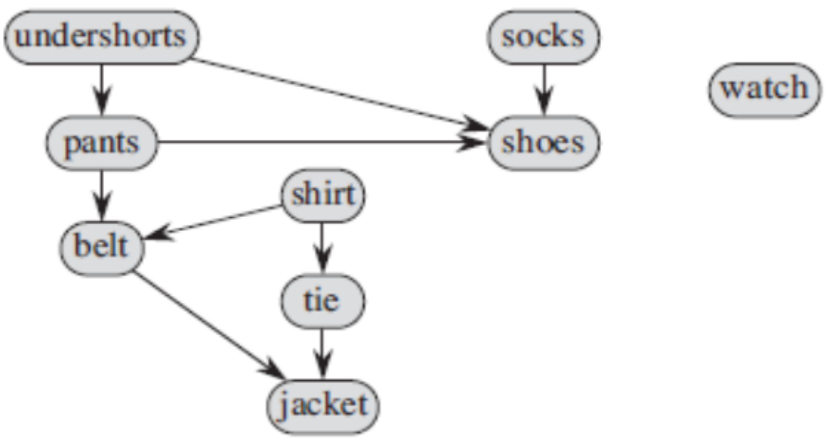




- Applications:
  - Indicate precedence relationship: an edge e=(a, b) from a to b means that event a must happen before event b.

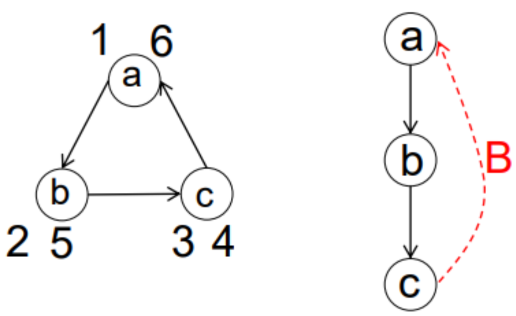
DAG: Example

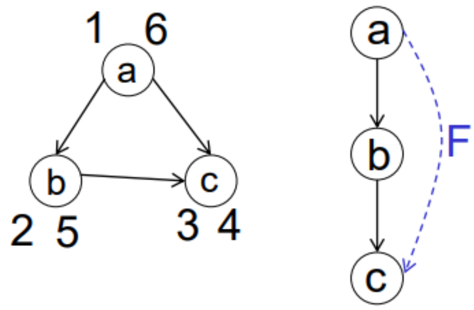
- Indication of precedence:
  - Some events must happen before some other events
- Example: professor gets dressed in the morning.
  - The professor must put certain garments before others (e.g., socks before shoes).



How to check DAG

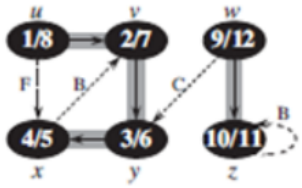
- A directed graph is acyclic if and only if the graph has no back edges.





No, (c,a) is a back edge

Yes, no back edges.



No, (x,v) and (z, z) are back edges.

Topological sort

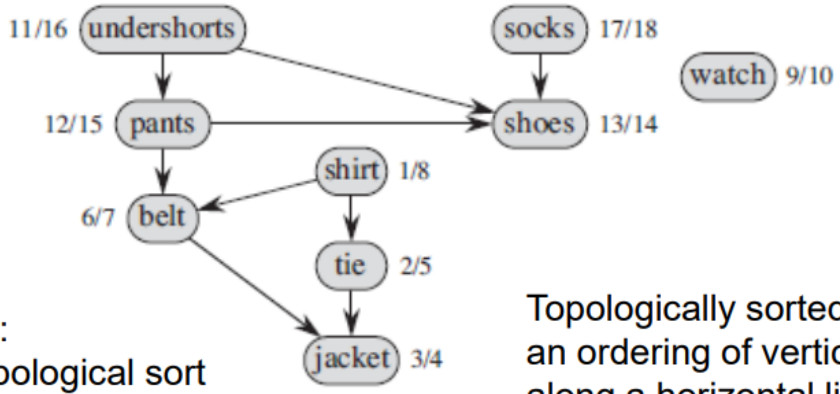
- Input:
  - DAG G = (V, E)
- Aim:
  - Introduce a linear ordering of all its vertices, such that for any edge (u,v) in the DAG, event u appears before event v in the ordering.
- Output:
  - Topologically sorted DAG, i.e., a linked list of vertices, showing an order.

- Algorithm: TOPOLOGICAL-SORT(G)

```
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

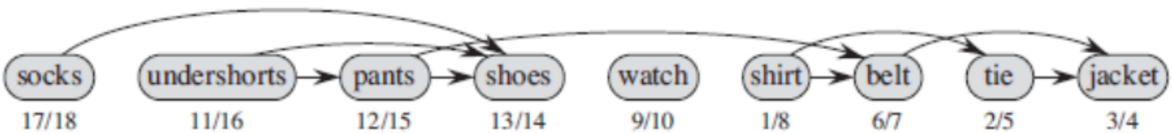
  - Intuition: reversely sort vertices according to the finishing times obtained from a DFS.
    - If *v.f* < *u.f*, u → v
    - event *u* happens before event *u*.

Topological sort example



Mini quiz:
Is this topological sort unique? Is there other valid topological sort for the same vertices?

Topologically sorted DAG is an ordering of vertices along a horizontal line such that all directed edges go from left to right.



Run Time of Topological Sort

Algorithm:

```
TOPOLOGICAL-SORT(G)
call DFS(G) to compute finishing times v.f for each vertex v
as each vertex is finished, insert it onto the front of a linked list
return the linked list of vertices
```

- Run-time:
  - DFS takes Θ(|V|+|E|)
  - It takes constant time Θ(1) to insert a vertex onto the front of a linked list.
    - In total, |V| vertices. Thus, Θ(|V|).
  - In total, Θ(|V|+|E|).

Topological sort correctness

- Topological sort of a DAG G
  - Produce a linear order of vertices in G, such that if an edge (u, v) exists in G, event u appears before event v in the ordering.

- Prove: Topoligical-Sort(G) produces a topological sort of G.
  - i.e., Topoligical-Sort(G) can produces an order that u appears before v.
  - Just need to prove, for any edge (u,v) in a DAG G, if we use a DFS to explore (u,v), we must obtain u.f > v.f.
  - Since Topoligical-Sort(G) uses an reversed order to arrange vertices by their finishing time, as long as we have u.f > v.f, we can have the order that u appears before v.

- We just need to show that , if we use a DFS to explore edge (u,v) in a DAG G, we must obtain u.f > v.f.

- When explore (u, v) by a DFS, we distinguish three cases:
  - Case 1: v is white;
    - v becomes a descendant of u, thus v will be finished before u, i.e., u.f v.f.
  - Case 2: v is gray;
    - (u, v) is a back edge. However, DAG should not have a back edge. So this won’t happen.
  - Case 3: v is black;
    - v has already finished. Thus, u.f > v.f.