Algorithms and Data Structures (CSci 115)

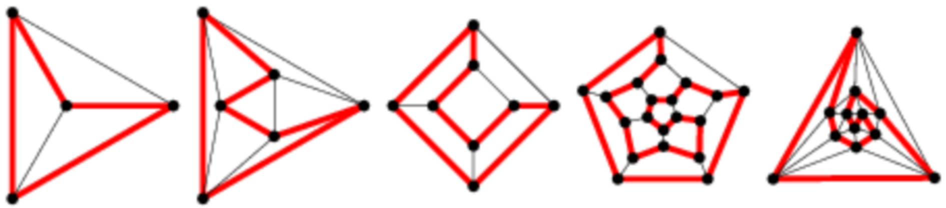
California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning outcomes

- Hamiltonian graphs
- Graph algorithms
  - Eulerian algorithms
    - **Fleury**
    - **Hierholzer**
  - Graph traversals
    - **Breadth First Search (BFS)**
    - **Depth First Search (DFS)**
  - Parenthesis theorem

Hamiltonian graphs

- Definition
  - Hamiltonian graph/Hamilton graph: a graph possessing a Hamiltonian cycle.
    - A graph that is not Hamiltonian is said to be non-hamiltonian.
  - Hamiltonian cycle
    - Hamiltonian path that is a cycle
  - Hamiltonian path
    - A path in an undirected or directed graph that visits each **vertex** exactly once!
- Property
  - Every platonic solid, considered as a graph, is Hamiltonian



Eulerian trails and circuits

Reminder:
Trail: edges are distinct
Eulerian trail: visit ALL the edges once
Path: trail with distinct vertices

- Fleury's algorithm (1883)
  1. Check to make sure that the graph is:
     1. **connected**
     2. **all vertices are of even degree**
  2. Start at any vertex
  3. Travel through an edge:
     - If it is **not a bridge for the untraveled part** or there is no other alternative
       - A bridge is an edge that if removed, it produces a disconnected graph !!
  4. Label the edges in the order in which you travel them.
  5. When you cannot travel any more, stop.

Eulerian trails and circuits

- **Fleury's algorithm**

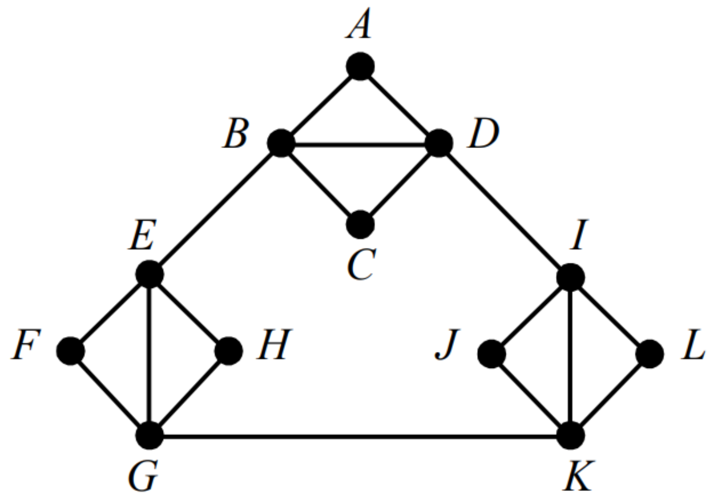
Input: A connected (p, q) graph $G = (V, E)$.
Output: An eulerian circuit C of G.
Method: Expand a trail C_i while avoiding bridges in $G - C_i$, until no other choice remains.

1. Choose any $v_0 \in V$ and let $C_0 = v_0$ and $i \leftarrow 0$.
2. Suppose that the trail $C_i = v_0, e_1, v_1, \ldots, e_i, v_i$ has already been chosen:
   a. At v_i, choose any edge e_{i+1} that is not on C_i and that is not a bridge of the graph $G_i = G - E(C_i)$, unless there is no other choice.
   b. Define $C_{i+1} = C_i, e_{i+1}, v_{i+1}$.
   c. Let $i \leftarrow i + 1$.
3. If $i = |E|$
   then halt since $C = C_i$ is the desired circuit;
   else go to 2.

- Theorem
  - If G is Eulerian, then any circuit constructed by Fleury's algorithm is Eulerian.

Eulerian trails and circuits

- Example
  - Start with the graph
  - Make sure that when you remove an edge, G is not disconnected



Start

Eulerian trails and circuits

- **Hierholzer's algorithm**
  - It produces circuits in a graph G
    - The circuits are pairwise edge disjoint.
  - When these circuits are put together properly, they form an Eulerian circuit of G.
  - This patching together of circuits hinges of course, on the circuits having a **common** vertex
    - following from the **connectivity** of the graph
  - Once 1 circuit is formed,
    - if (all edges have not been used)
    - then
      - there must be 1 edge that is incident to a vertex of the circuit, and
      - we use this edge to begin the next circuit.
      - These circuits then share a common vertex.

Eulerian trails and circuits

- Hierholzer's algorithm (1873)
  1. Pick any **starting vertex v**
  2. Follow a trail of edges from that vertex until **returning to v**.

  ➢ It is not possible to get stuck at any vertex other than v
    - because the **even** degree of all vertices ensures that:
      - when the trail enters another vertex w, there must be an **unused** edge leaving w.

  ➢ The tour formed in this way is a **closed** tour,
    - but may not cover all the vertices and edges of the initial graph.

  ➢ As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour,
    - start another trail from u, following **unused** edges until returning to u,
    - and join the tour formed in this way to the previous tour.

Eulerian trails and circuits

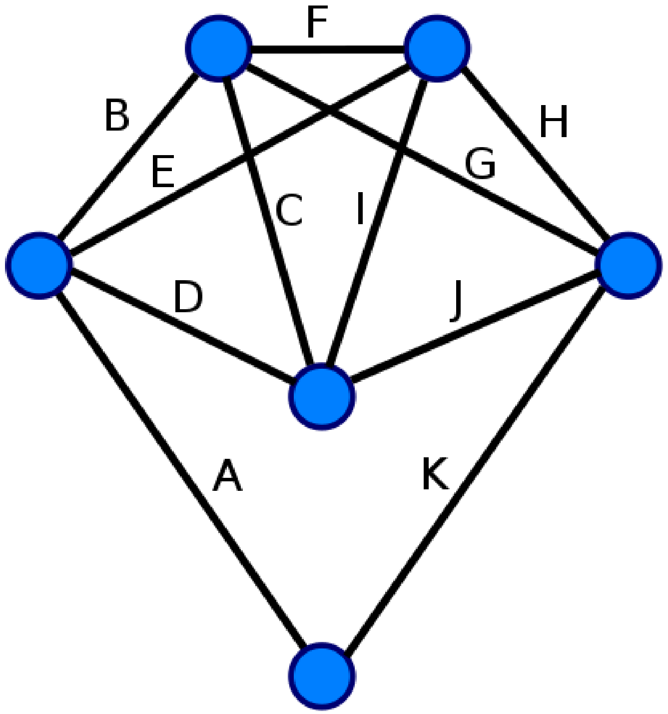
- Hierholzer's algorithm

Input: A connected graph $G = (V, E)$, each of whose vertices has even degree.
Output: An eulerian circuit C of G.
Method: Patching together of circuits.

1. Choose $v \in V$. Produce a circuit C_0 beginning with v by traversing at each step, any edge not yet included in the circuit. Set $i = 0$.
2. If $E(C_i) = E(G)$;
   then halt since $C = C_i$ is an eulerian circuit;
   else choose a vertex v_i on C_i that is incident to an edge not on C_i. Now build a circuit C_i^* beginning with v_i in the graph $G - E(C_i)$. (Hence, C_i^* also contains v_i.)
3. Build a circuit C_{i+1} containing the edges of C_i and C_i^* by starting at v_{i-1}, traversing C_i until reaching v_i, then traversing C_i^* completely (hence, finishing at v_i) and then completing the traversal of C_i. Now set $i \leftarrow i + 1$ and go to 2.

Eulerian trails and circuits

- Example
  - Run Hierholzer's algorithm on this example:
    - Pick a starting vertex *v*
    - Follow a trail of edges from that vertex until returning to *v*
      - We get a closed tour
    - As long as ∃ a vertex *u* ϵ the current tour
      - but that has adjacent edges not part of the tour
    - Start another trail from *u (create a new tour)*
    - Join the tour formed to the previous tour…



Applications

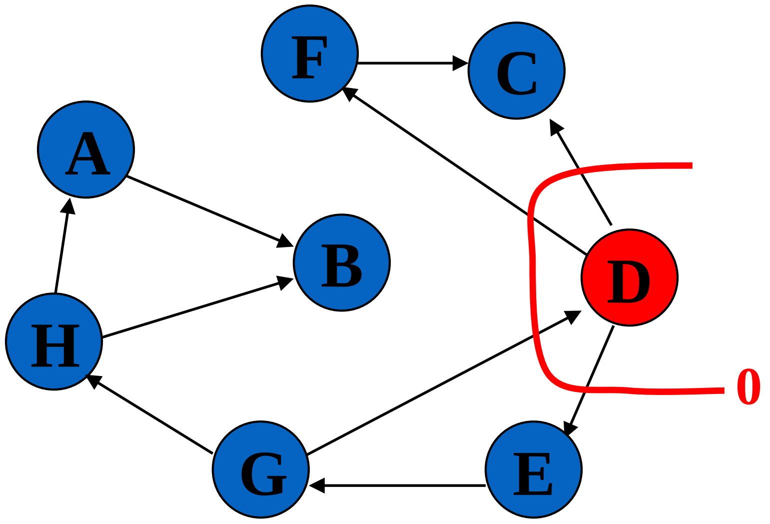
- Eulerian trails
  - Used in
    - Bioinformatics
      - to reconstruct the DNA sequence from its fragments
    - CMOS (Complementary metal–oxide–semiconductor) circuit design
      - to find an optimal logic gate ordering

Graph algorithms

- Searching in a graph
  - Systematically follow the **edges** of a graph to visit the **vertices** of the graph.
  - Used to discover the structure of a graph.
- Standard graph-searching algorithms.
  - **Breadth-first Search** (BFS).
  - **Depth-first Search** (DFS).

Breadth-First Search

- Goal
  - ➢Can be used to attempt to visit all nodes of a graph in a systematic manner
- Input:
  - ➢Graph
    - Directed or undirected graphs
    - Weighted or unweighted graphs
- BFS starts with given node
  - ➢Example: start with D

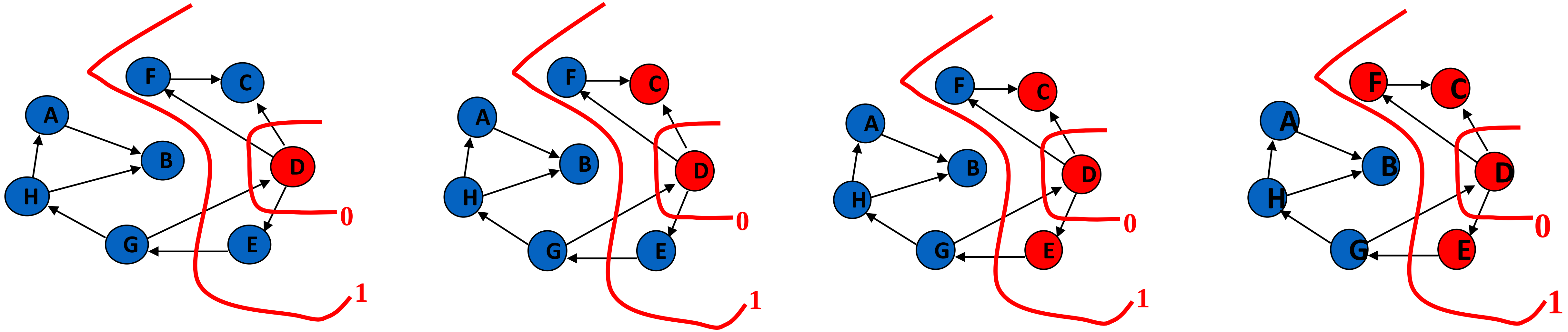


Breadth-First Search

- Start with a node, then visits nodes **adjacent** in some specified order
  - Example: Order given by the nodes in the adjacency matrix
  - Like ripples in a lake

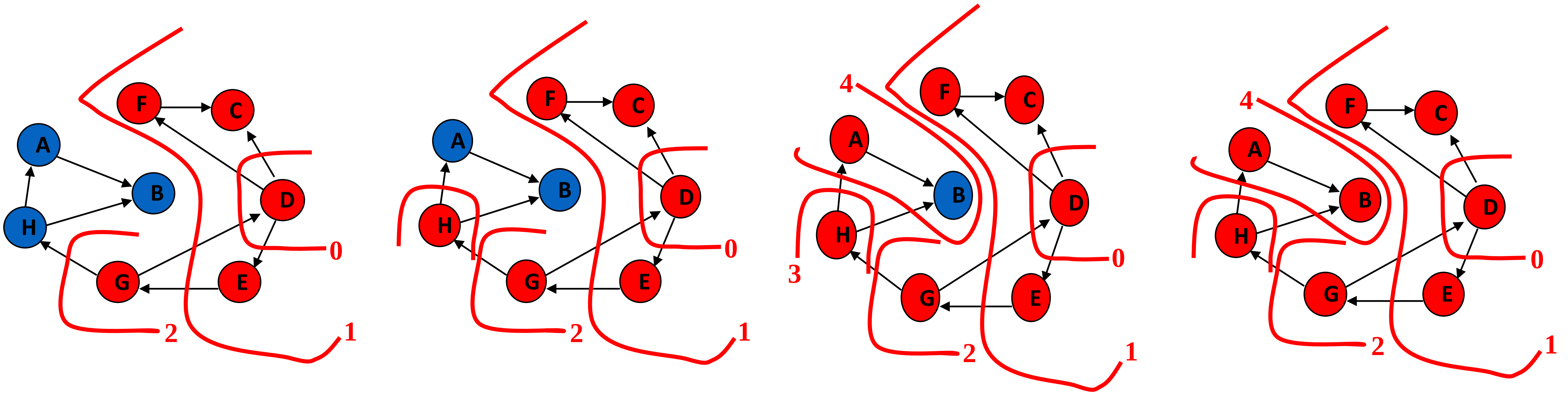
Breadth-First Search

- Steps
  - Red: visited nodes



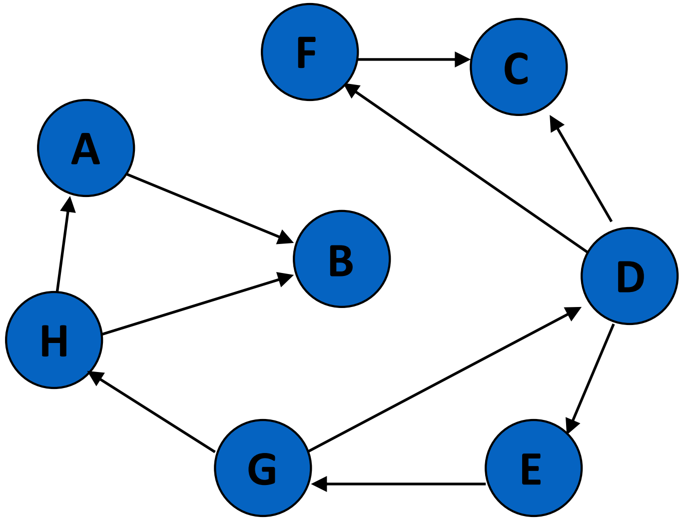
Breadth-First Search

- Steps
  - Red: visited nodes



Breadth-First Search

- Implementation
  - Maintain an enqueued array
  - Visit node when dequeued.
  - **Step 1**: Enqueue D



A	
B	
C	
D	
E	
F	
G	
H	

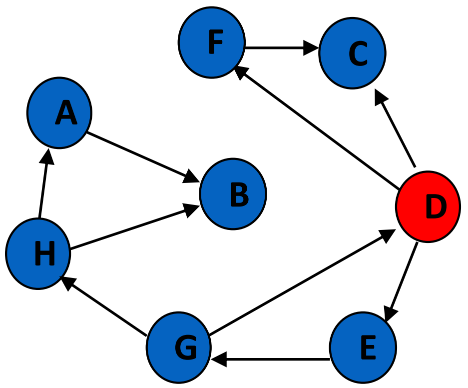
Enqueued Array

→

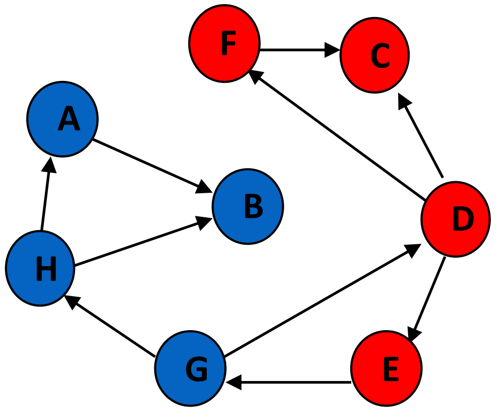
A	
B	
C	
D	√
E	
F	
G	
H	

Breadth-First Search

- Implementation



A	
B	
C	√
D	√
E	√
F	√
G	
H	



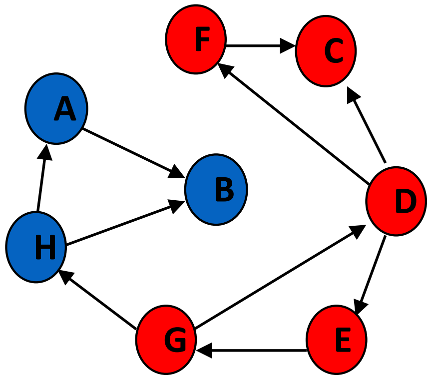
A	
B	
C	√
D	√
E	√
F	√
G	
H	

- Dequeue D.
- Visit D.
- Enqueue unenqueued nodes adjacent to D.

All the nodes adjacent to D are in the queue

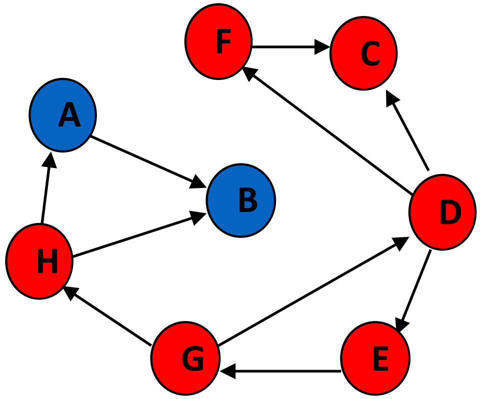
Breadth-First Search

- Implementation



A	
B	
C	√
D	√
E	√
F	√
G	√
H	√

- Dequeue G.
- Visit G
- Enqueue unenqueued nodes adjacent to G.

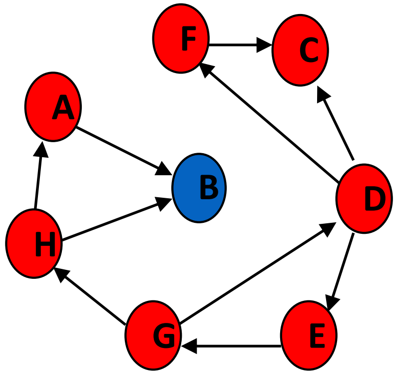


A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

- Dequeue H
- Visit H
- Enqueue unenqueued nodes adjacent to H.

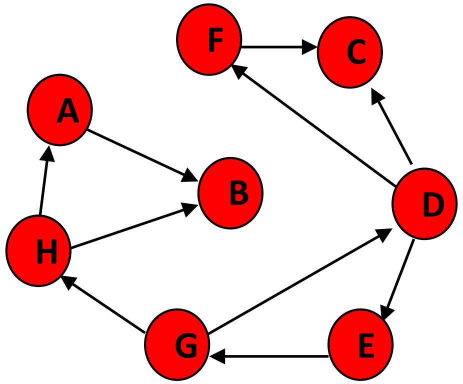
Breadth-First Search

- Implementation



A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

- Dequeue A
- Visit A
- Enqueue unenqueued nodes adjacent to A



A	√
B	√
C	√
D	√
E	√
F	√
G	√
H	√

- Dequeue B
- Visit B
- Enqueue unenqueued nodes adjacent to B

Breadth-First Search

- Pseudo-code
  - ➢See section 22.2 in Intro to Algo book.

Pseudo code: BFS(G,s)

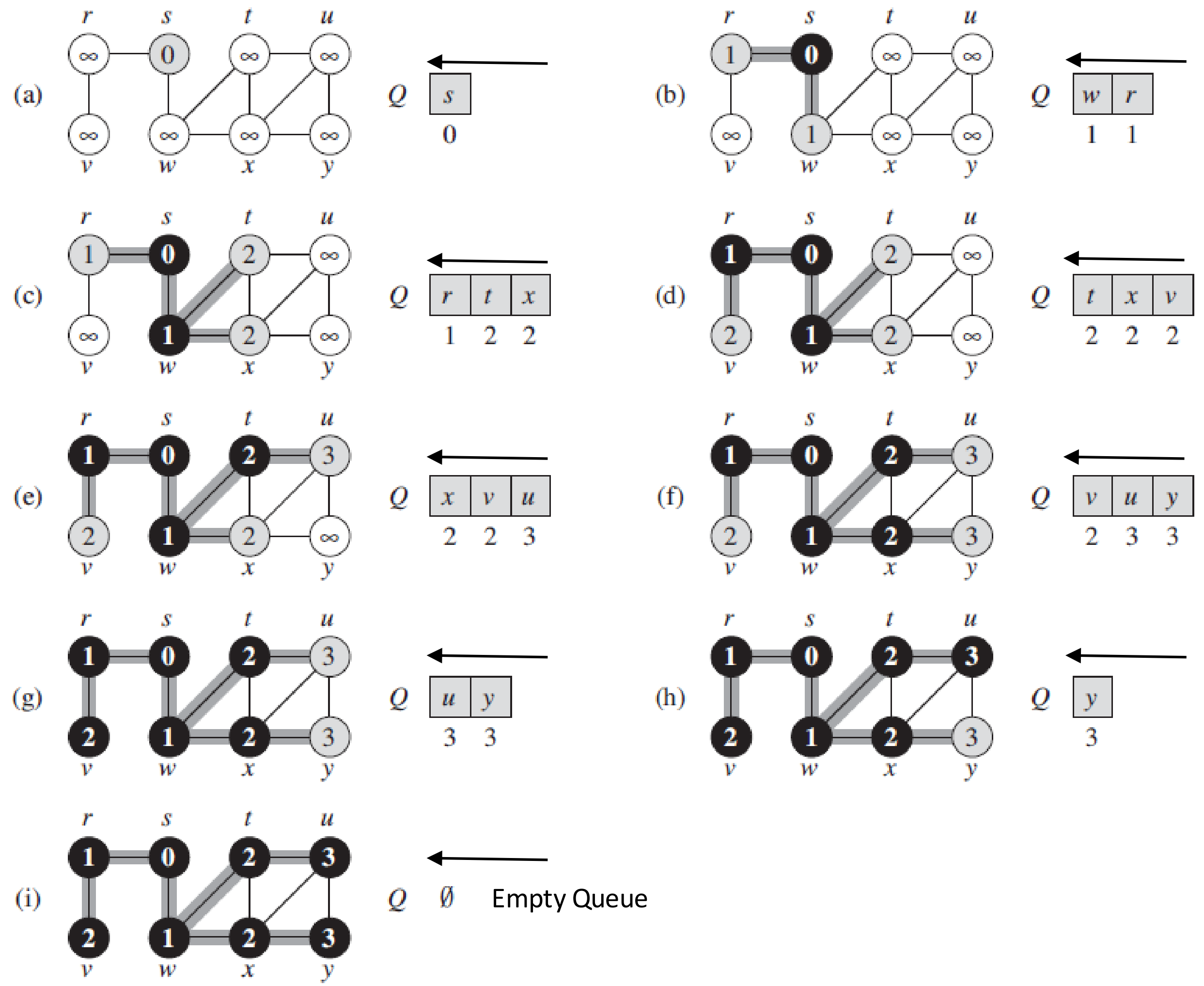
```
// white: undiscovered, gray: discovered, black: finished
//Q: a queue of discovered vertices
// color[v]: color of v
// d[v]: distance from s to v
// π[u]: predecessor of v
for each vertex u in V[G] – {s}
    do color[u] ← white
       d[u] ← ∞
       π[u] ← nil
color[s] ← gray
d[s] ← 0
π[s] ← nil
Q ← Ø
enqueue(Q,s)
while Q ≠ Ø
    do u ← dequeue(Q)
         for each v in Adj[u]
             do if color[v] = white
                  then color[v] ← gray
                       d[v] ← d[u] + 1
                       π[v] ← u
                       enqueue(Q,v)
         color[u] ← black
```

Initialize color, v, pi

Initialize s and Q

Breadth-First Search

- Example
  - Source=s
    - black=finished
    - white=unvisited
    - gray=discovered



Breadth-First Search

- Complexity analysis
  - Initialization takes $O(V)$.
  - Traversal Loop
    - After initialization
      - Each vertex is Enqueued and Dequeued **at most once**
        - each operation takes $O(1)$
      - → total time for queuing is $O(V)$.
    - The adjacency list of each vertex is scanned at most once.
      - The sum of lengths of all adjacency lists is $\Theta(E)$.
  - Summing up over all vertices
    - → Total running time of BFS is $O(V+E)$
      - Linear in the size of the adjacency list representation of graph

Breadth-First Search

- More definitions
  - For a graph $G = (V, E)$ with source s
    - The **predecessor subgraph** of G is $G_\pi = (V_\pi, E_\pi)$ where
      - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$
      - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
  - The predecessor subgraph G_π is a **BF tree** if:
    - V_π consists of the vertices reachable from s
    - $\forall\, v \in V_\pi$, $\exists!$ simple path from s to v in G_π that is also a shortest path from s to v in G.
  - The edges in E_π are called **Tree edges**.
    $|E_\pi| = |V_\pi| - 1$.

$\exists!$: there exists a unique …

Depth-First Search

- Goal
  - To attempt to visit all nodes of a graph in a systematic manner
- Input
  - Graph
    - directed or undirected graphs
    - weighted or unweighted graphs
- Steps
  - Explore edges out of the **most recently discovered** vertex *v*.
  - When all edges of *v* have been explored
    - backtrack to explore other edges leaving the vertex from which *v* was discovered (its *predecessor*).
  - Search as **deep as possible** first
  - Continue until all vertices reachable from the original source are discovered
    - If any undiscovered vertices remain
    - then one of them is chosen as a new source and search is repeated from that source.

Depth-First Search

- Pseudo-code
  - See section 22.3 in Intro to Algo book.

d: discovery
f: finishing

Pseudo-code: DFS(G)

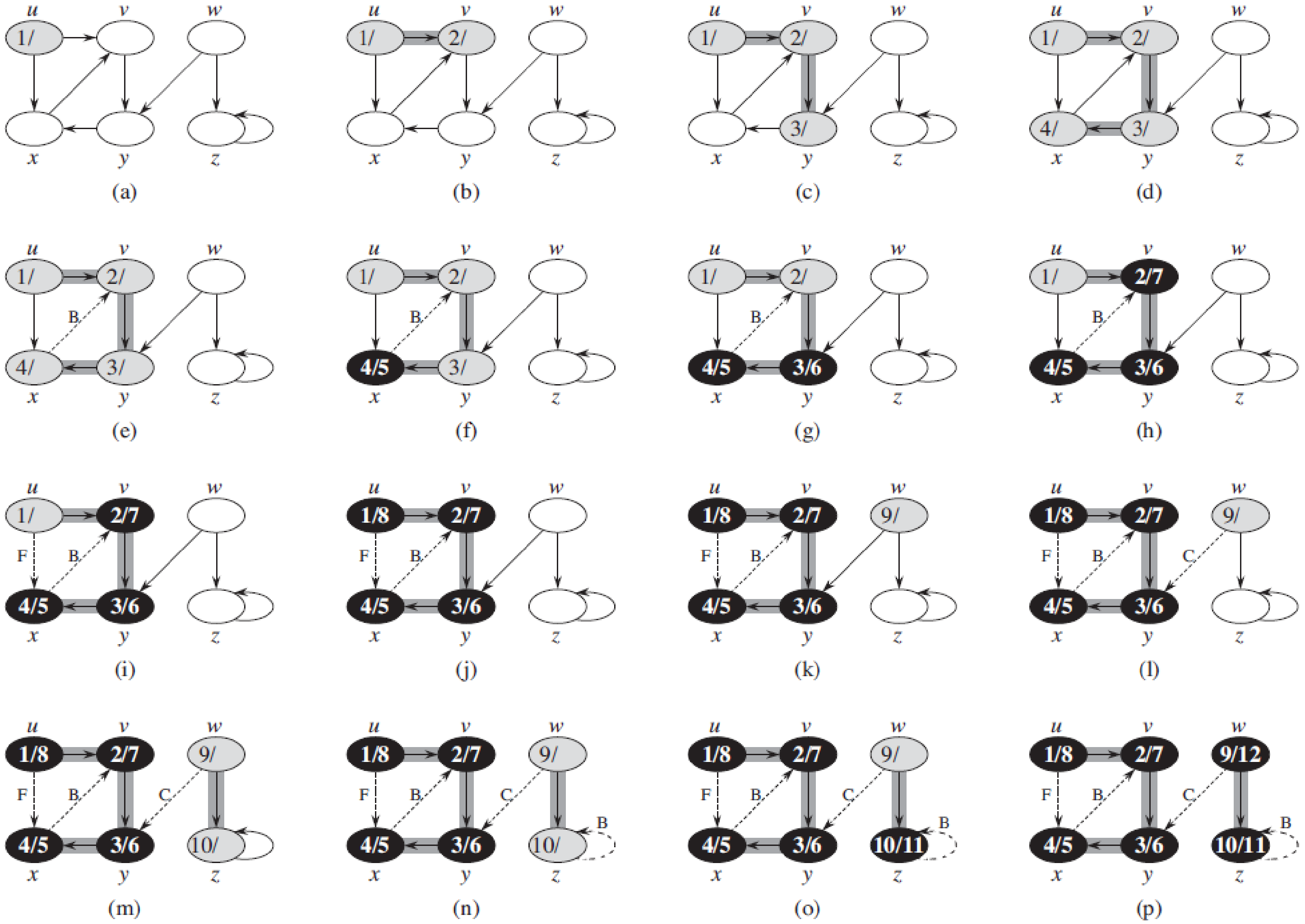
for each vertex $u \in V[G]$
    do $color[u] \leftarrow$ white
        $\pi[u] \leftarrow$ NIL
$time \leftarrow 0$
    for each vertex $u \in V[G]$
        do if $color[u]$ = white
            then DFS-Visit(u)

Pseudo-code: DFS-Visit(u)

$color[u] \leftarrow$ gray   // White vertex u has been discovered
$time \leftarrow time + 1$
$d[u] \leftarrow time$
for each $v \in Adj[u]$
    do if $color[v]$ = white
        then $\pi[v] \leftarrow u$
            DFS-Visit(v)
$color[u] \leftarrow$ black   // Blacken u; it is finished.
$time \leftarrow time + 1$
$f[u] \leftarrow time$

Depth-First Search

- Example
  - Edges=shaded if tree edges
  - Edges=dashed otherwise
  - Non-tree edges:
    - B (back), C (cross), or F (forward)
  - Timestamps:
    - (discovery time/finishing times)



Depth-First Search

- Complexity analysis
  - Loops on lines 1-2 & 5-7 take $\Theta(V)$ time
    - Excluding time to execute DFS-Visit.
  - DFS-Visit
    - Called once for each white vertex $v \in V$ when it is painted gray the first time.
    - Lines 3-6 of DFS-Visit
      - executed $|\text{Adj}[v]|$ times.
    - Total cost of executing DFS-Visit: $\sum_{v \in V} |\text{Adj}[v]| = \Theta(E)$
  - Total running time of DFS: $\Theta(V+E)$.

Depth-First Search

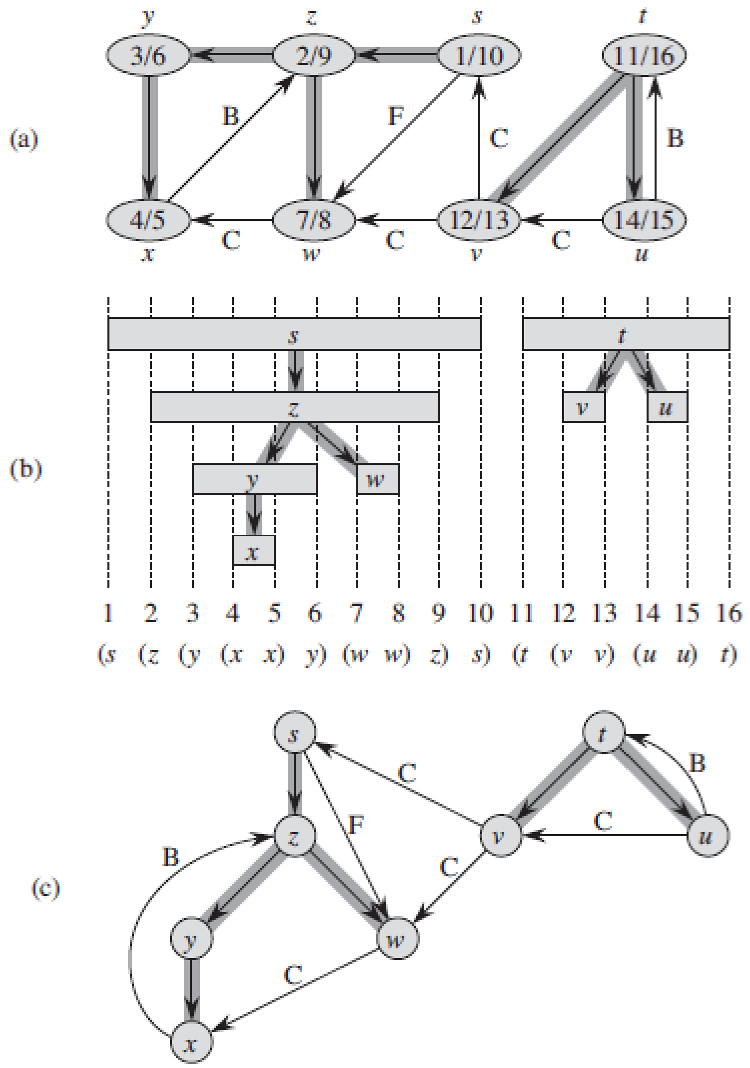
- DFS property
  - that discovery and finishing times have ***parenthesis structure***.
    - If we represent
      - the discovery of vertex u with a “(u”
      - the finishing by a right “u)”
    - Then the history of **discoveries** and **finishings** makes a well-formed expression
      - → Parentheses are properly nested.

Parenthesis theorem

- For all u, v, exactly one of the following holds:
  1. $d[u] < f[u] < d[v] < f[v]$ or $d[v] < f[v] < d[u] < f[u]$ and neither u nor v is a descendant of the other.
     - The intervals [u.d;u.f] and [v.d;v.f] are **entirely disjoint**, and
     - neither u nor v is a descendant of the other in the depth-first forest
  2. $d[u] < d[v] < f[v] < f[u]$ and v is a descendant of u
     - The interval [v.d; v.f] is **contained entirely** within the interval [u.d;u.f], and
     - v is a descendant of u in a depth-first tree
  3. $d[v] < d[u] < f[u] < f[v]$ and u is a descendant of v.
     - The interval [u.d;u.f] is **contained entirely** within the interval [v.f;v.f], and
     - u is a descendant of v in a depth-first tree.

Parenthesis theorem

- Example



Classification of edges

- **Property of DFS**:
  - Used to classify the edges of the input graph G=(V,E)
  - The type of each edge
    - provide important information about a graph.
    - Example
      - A directed graph is **acyclic** if and only if a DFS yields no "back" edges
- We can define 4 edge types in terms of the DF forest G_π produced by a DFS on G:
  1. ***Tree edges*** in the DF forest G_π. Edge (u,v) is a tree edge if v was **first discovered** by exploring edge (u,v)
  2. ***Back edges:*** edges (u,v) connecting a vertex u to an **ancestor** v in a DF tree.
     - self-loops that may occur in directed graphs: back edges.
  3. ***Forward edges:*** non-tree edges (u,v) connecting a vertex u to a descendant v in a DF tree.
  4. ***Cross edges:*** all other edges.
     - They can go between vertices in the same DF tree, as long as
       - 1 vertex is not an ancestor of the other or
       - they can go between vertices in different DF trees.
- **Theorem**
  - In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.
- **Proof**: decomposition with the different cases (see Intro to Algo, Theorem 22.10)

Conclusion

- Now, you should be able to implement and use:
  - Eulerian algorithms
    - Fleury
    - Hierholzer
  - Graph traversals
    - Breadth First Search (**BFS**)
    - Depth First Search (**DFS**)
  - Parenthesis theorem

Questions ?

- Reading
  - ➢Canvas: Csci 115 book Chapter 9
  - ➢Introduction to algorithms, Chapter 22.