Graph Algorithms

Depth-First Search (DFS)

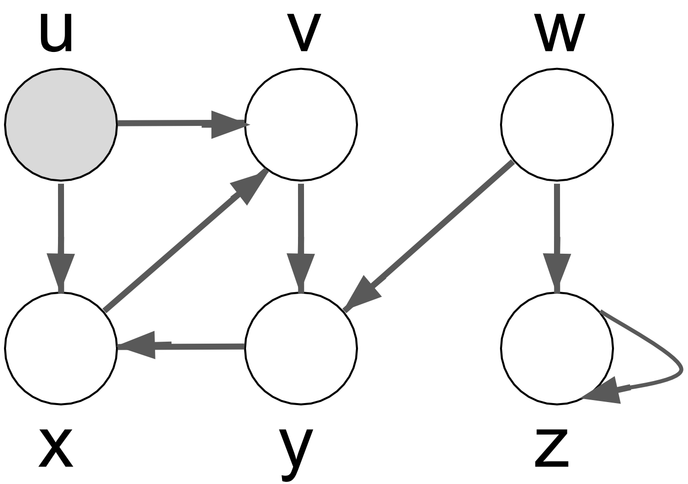
Depth-First Search (DFS)

- **Input**: A graph G = (V, E) and a vertex s
- **Output**:
  - Explores all the nodes in the graph reachable from **s**
  - generates a tree rooted at **s**
- Algorithm:
  - It starts from **s**, and follows the first path it finds and goes as deep as possible
  - During the execution of the algorithm, vertices are in one of the three following states:
    - UnDiscovered (white)
    - Discovered (gray)
    - Fully explored (black)
- Algorithm is implemented recursively
  - Can be implemented non-recursively using a stack

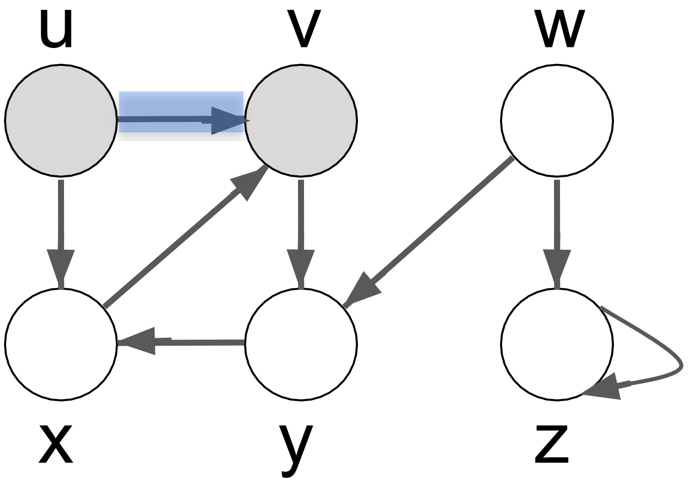
DFS: Basic Version

```
DFS(G)
    for each vertex u in G:
        color[u] = WHITE   #Mark w as undiscovered
    for each vertex u in G:
        if color[u] == WHITE   #if u is undiscovered
            DFS(G, u)
DFS(G, s)
    color[s] = GRAY    #discover s
    for each neighbor v of s:
        if (color[v] == WHITE)  #v is undiscovered
            DFS(G, v)
    color[v] = BLACK    #Mark s as fully-discovered(BLACK)
```

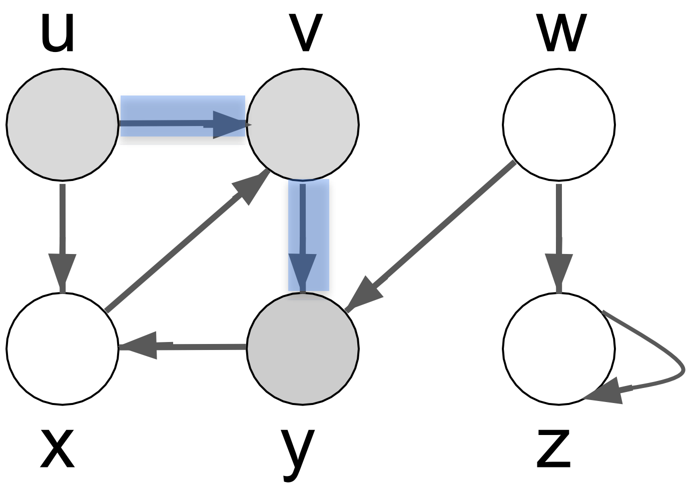
DFS Example



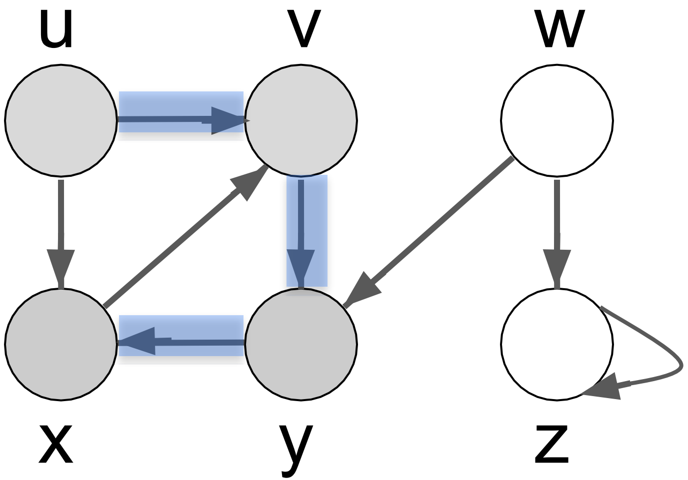
DFS Example



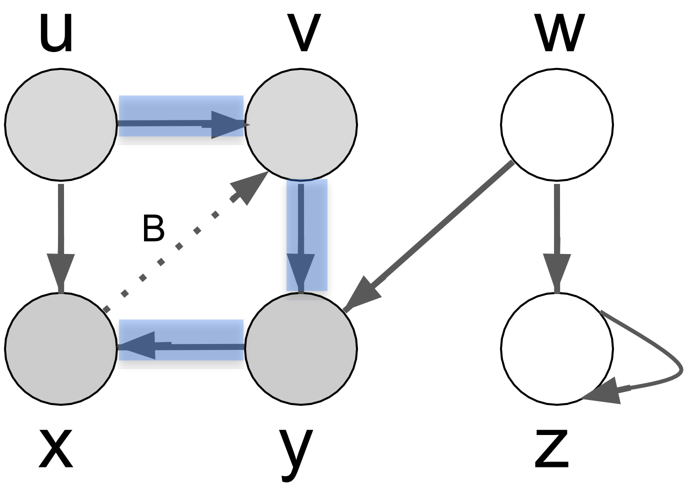
DFS Example



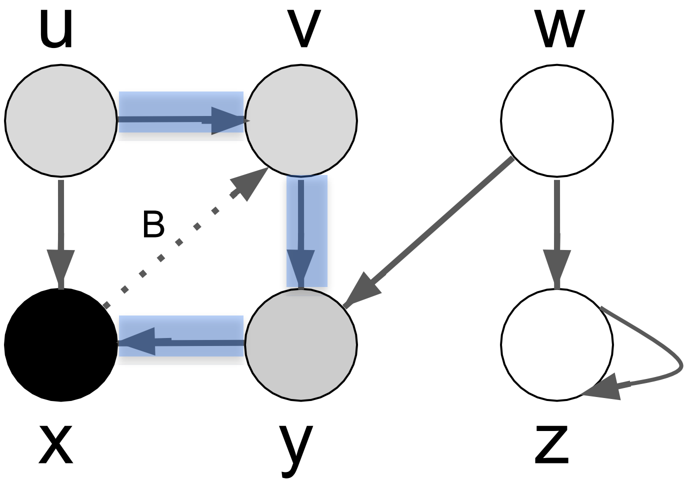
DFS Example



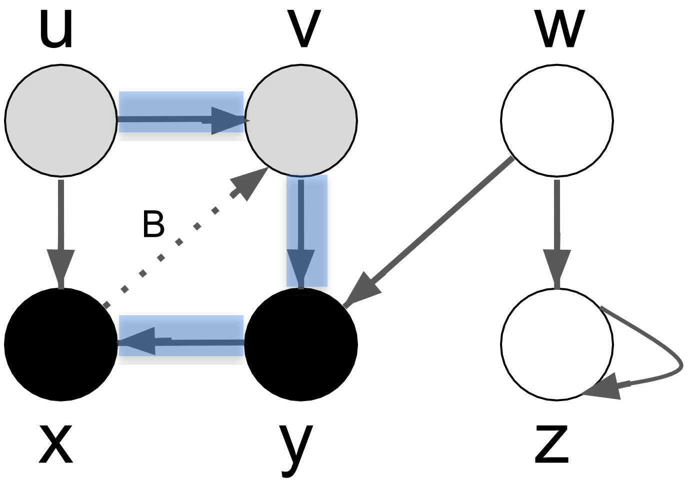
DFS Example



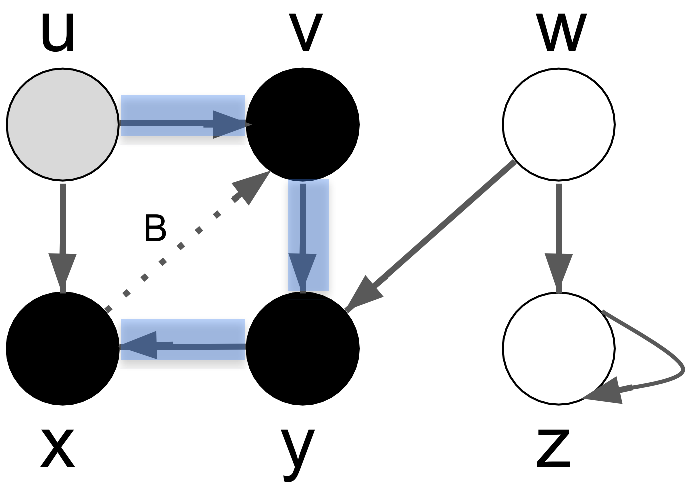
DFS Example



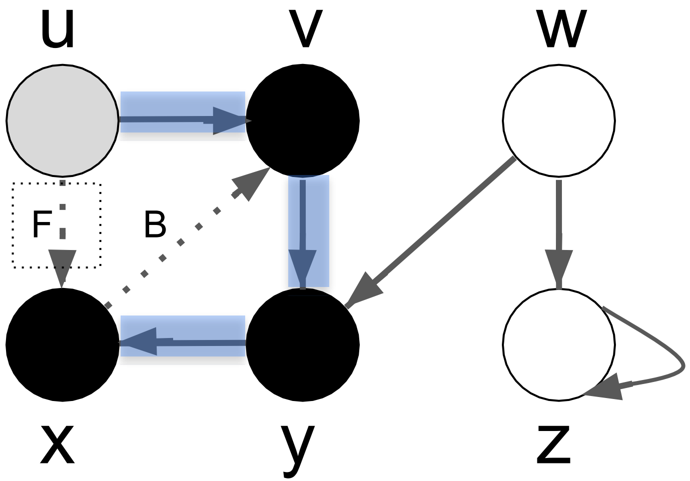
DFS Example



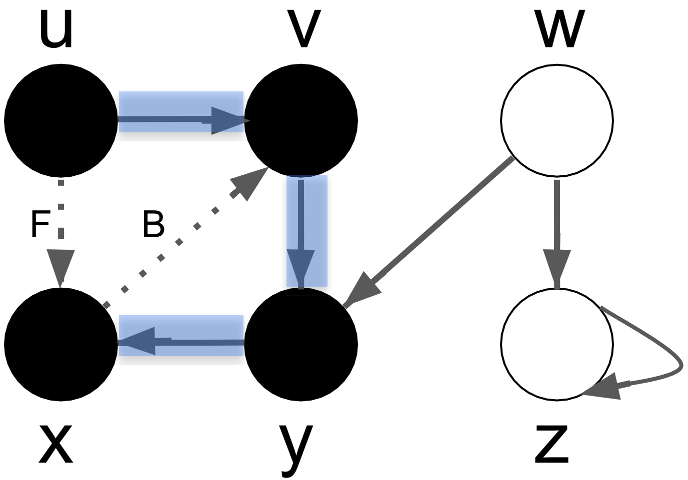
DFS Example



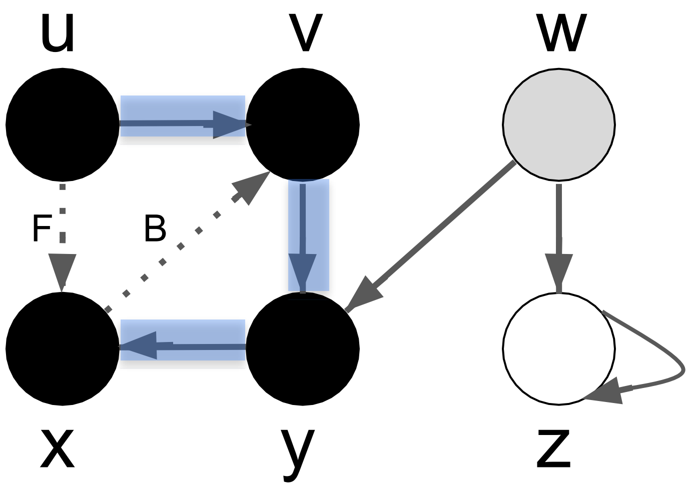
DFS Example



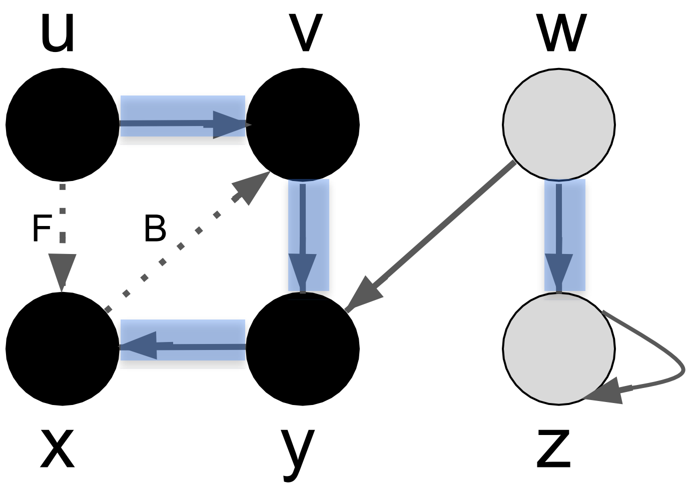
DFS Example



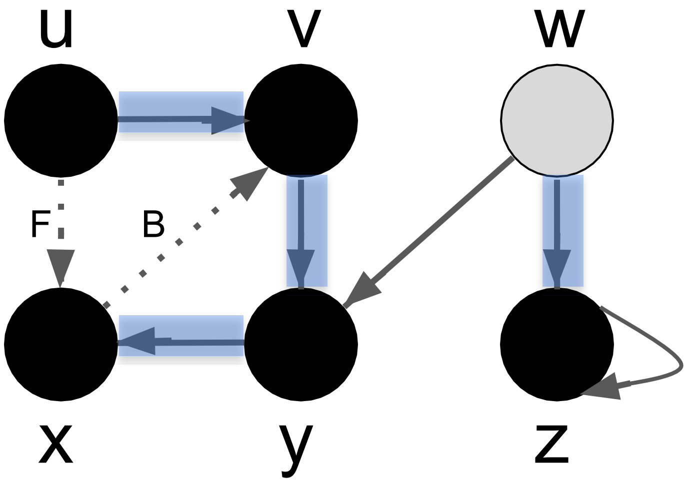
DFS Example



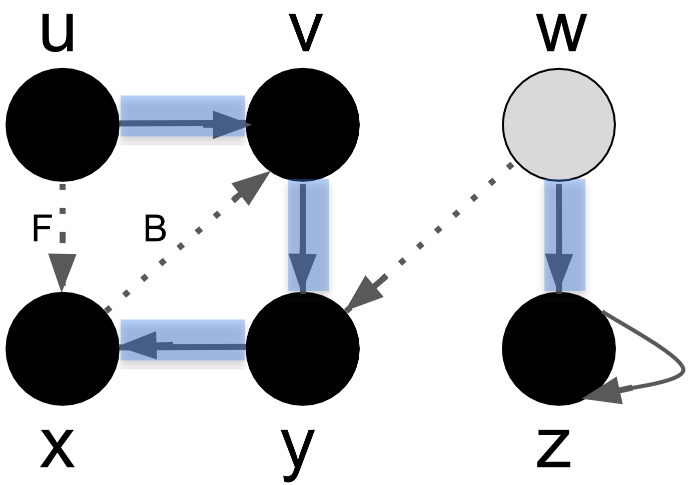
DFS Example



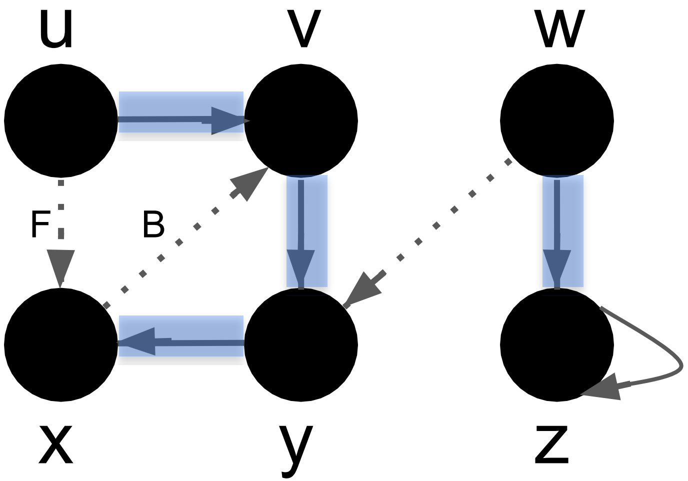
DFS Example



DFS Example



DFS Example



DFS: Basic Version

Θ(V)

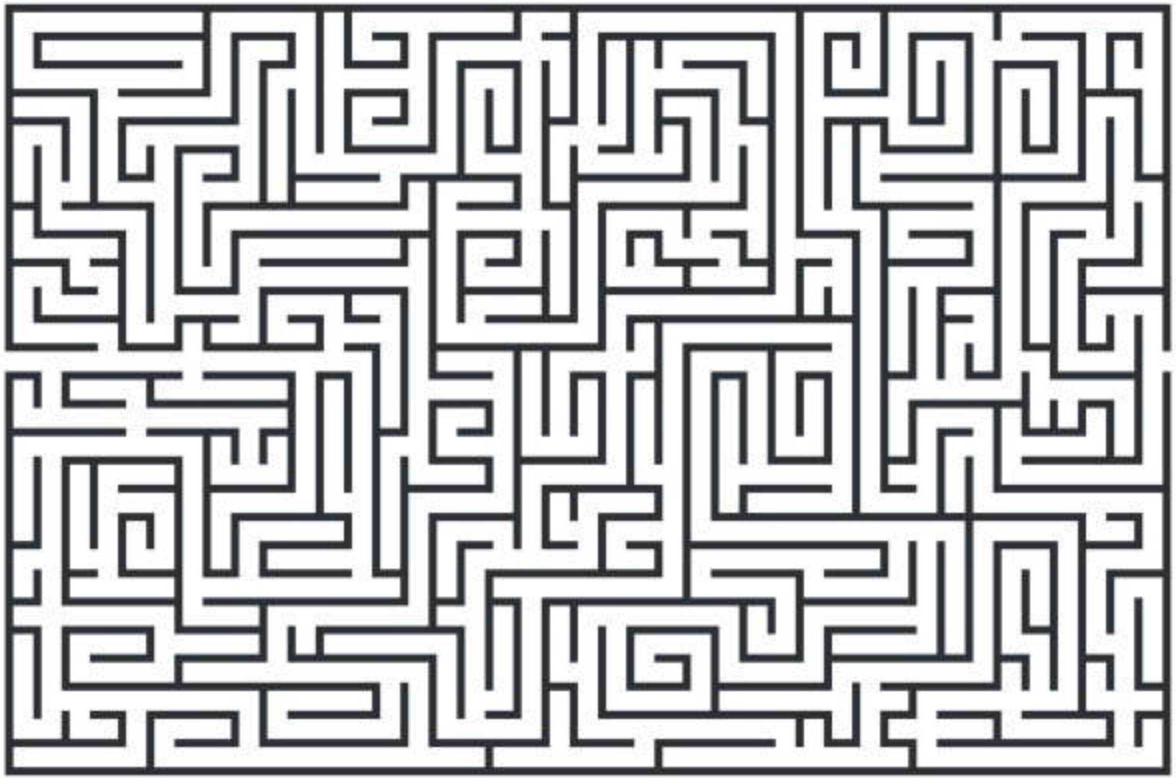
Θ(V)

Θ(E) in total:

- The loop iterates deg(u) times for each node s
- In total the runtime of for loop is Θ (∑deg(s)) = Θ(E)

```
DFS(G)
    for each vertex u in G:
          color[u] = WHITE  #Mark w as undiscovered
    for each vertex u in G:
         if color[u] == WHITE  #if u is undiscovered
              DFS(G, u)
DFS(G, s)
     color[s] = GRAY   #discover s
     for each neighbor v of s:
            if (color[v] == WHITE) #v is undiscovered
                 DFS(v)
     color[v] = BLACK   #Mark s as fully-discovered(BLACK)
```

DFS Application



DFS: Proof of correctness of DFS(G, s)

Lemma 1. There is a path from **s** to **v** if and only if **v** is discovered by DFS

- $\Leftarrow$ if **v** is discovered, then there is a path from **s** to **v**
  - **proof:** by induction on the number of vertices discovered
    - **Base case**: **s** is discovered
    - **Induction hypothesis**: Statement is true for the first i-1 vertices discovered
    - **Induction step**: Now, we need to prove it is true after i vertices are discovered. When the i-th vertex is discovered, it must have had a parent node that is already discovered. By induction hypothesis there is a path from s to the parent node (since it is one of those i-1 vertices). There is an edge connecting vertex i-1 and i. Therefore, there is a path from s to i
- $\Rightarrow$ if there is a path from **s** to **v**, then **v** is discovered
  - In other words, if **v** is not discovered then there is no path from **s** to **v**

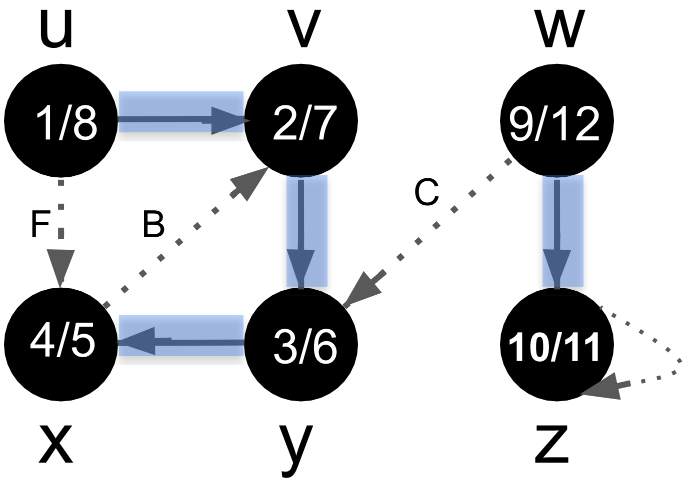
DFS vs. BFS

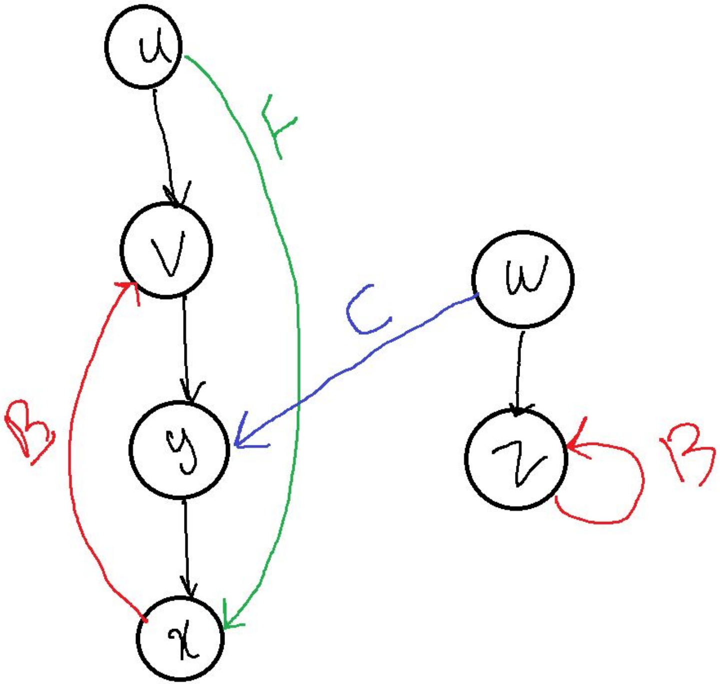
- Similarities
  - Both BFS and DFS could be used to
    - Check graph connectivity
    - Find the connected components containing s
      - Both start at an arbitrary node and explores the whole connected component
    - Check s-t connectivity
  - Both BFS(G, s) and DFS(G, s) generate a tree rooted at s
- Differences
  - BFS(G, s) finds the shortest path from s to all vertices reachable from s, but DFS(G, s) does not
  - BFS is implemented using a queue, but not DFS

DFS Tree / Forest

- Similar to BFS, we can construct a DFS tree
    - It could be used to find the path from s to all the reachable vertice
- When a vertex v is first discovered when exploring vertex u, we say vertex u is the parent of vertex v
- The edges (v, parent(v)) form a tree, and we can use them to find a path to s.
- A graph could have many different DFS trees depending on the order of exploring the neighbors of vertices

DFS Example





The same graph on the left: re-drawn to specify different classes of edges: B: Backward, F:Forward, C:Cross, and tree edges

DFS: classification of edges in the graph

- DFS classifies the edges of the input Graph G=(V, E).
- In a directed graph, an edge (u, v) is in one of the following classes:
  - Tree edge
    - If it is in the depth-first forest generated by the algorithm
  - Back edge
    - If **v** is ancestor of **u** in a depth-first tree
  - Forward edge
    - If **v** is a descendant of **u** in a depth-first tree
  - Cross edges
    - If it is not in any of the above category. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees

Discovery and finishing time

- We modify the algorithm to record the time when a vertex is first visited and the time when its exploring is finished
- These information will be very useful in design and analysis of algorithms

DFS: Enhanced Version

```
time =0
DFS(G)
    for each vertex u in G:
        Color[u] = WHITE
        Pred[u] = NULL
    time =0
    for each vertex u in G:
        if color[u] == WHITE
            DFS(G, u)
DFS(G, u)
    time = time+1
    discover[u] = time
    color[u] = GRAY
    for each neighbor v of u
        if (color[v] == WHITE)
            pred[v] = u
            DFS(G, v)
    color[u] = BLACK
    time = time + 1
    finish[u] = time
```

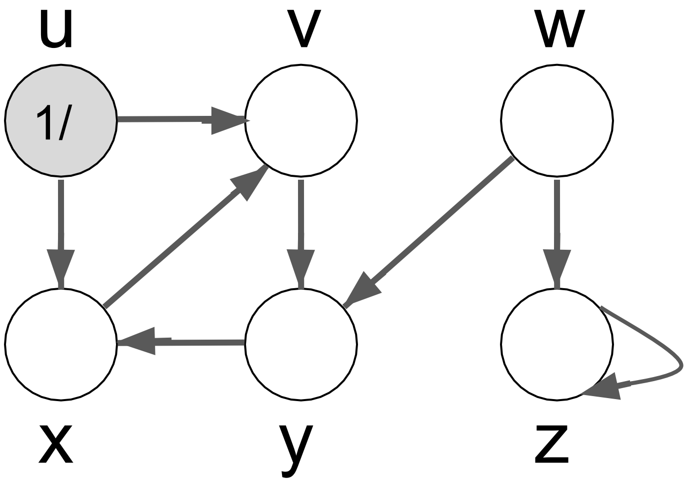
Enhanced DFS

- keeps track of whether a node is undiscovered, discovered, fully explored.
- keeps track of the time it started and finished with a vertex

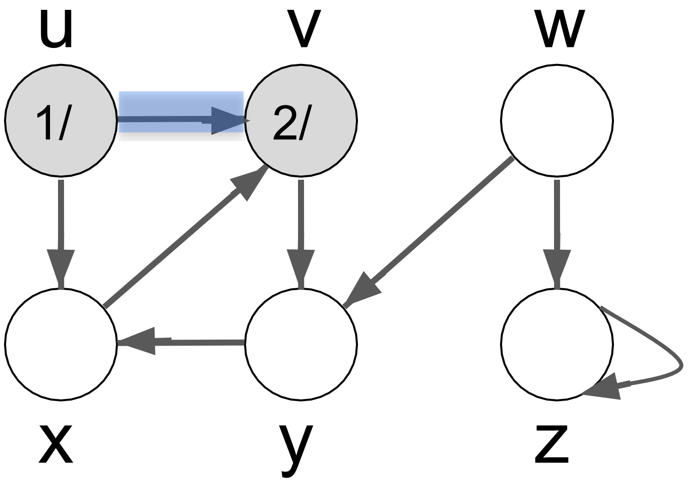
Output:

- a depth-first forest
- Timestamp each vertex:
  - d[u], when vertex u was discovered
  - f[u]: when u was fully explored (all its neighbors are discovered)

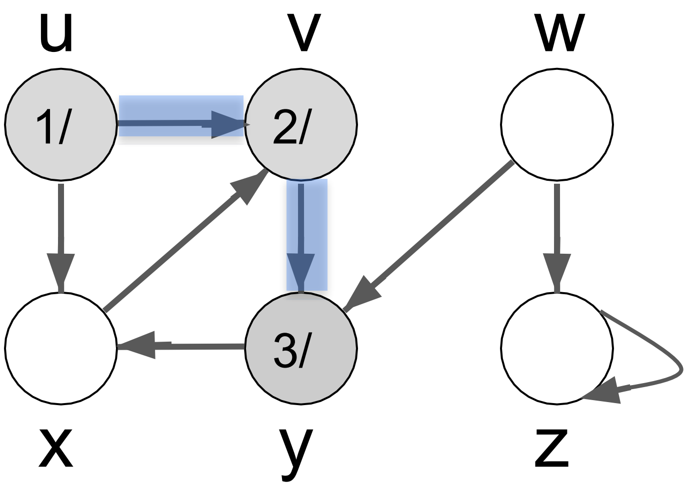
DFS Example



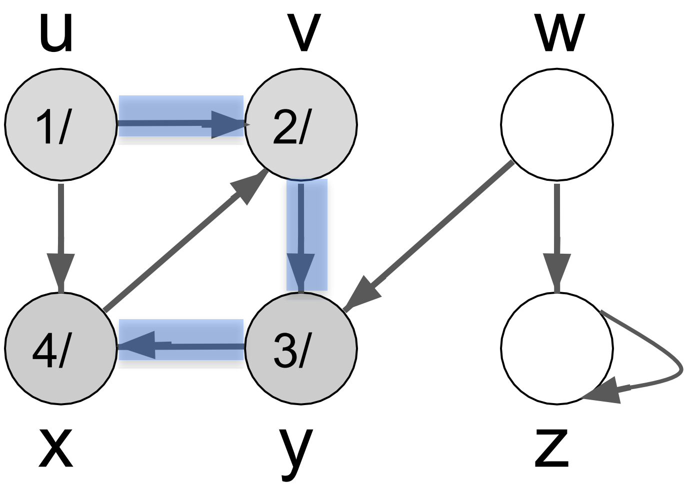
DFS Example



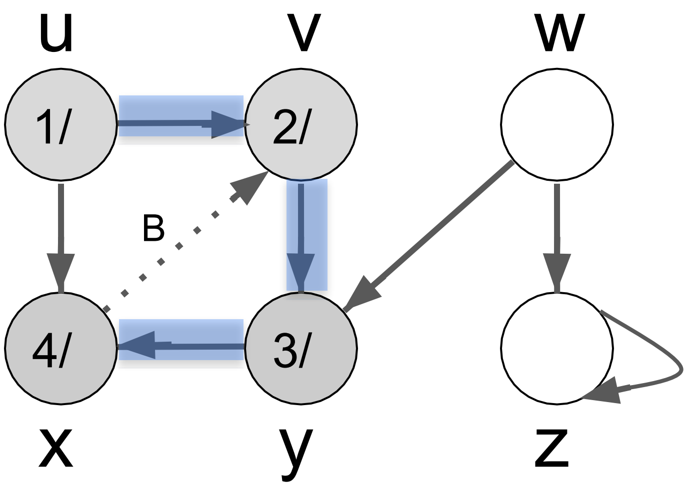
DFS Example



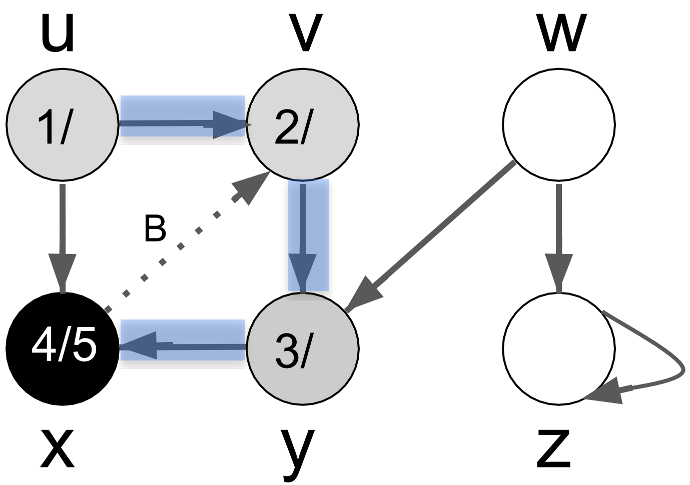
DFS Example



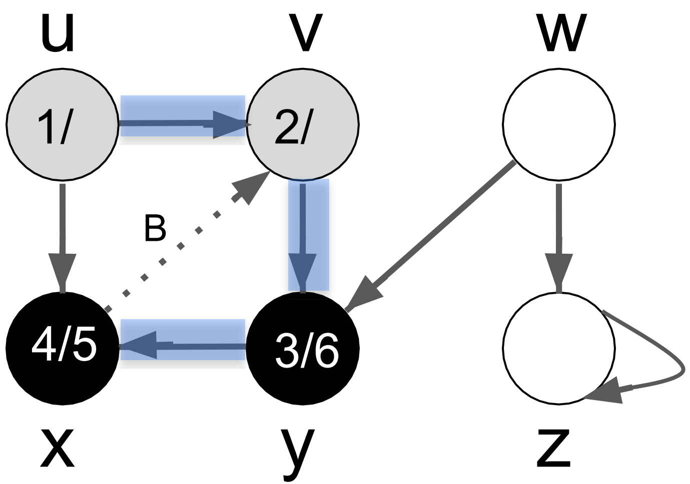
DFS Example



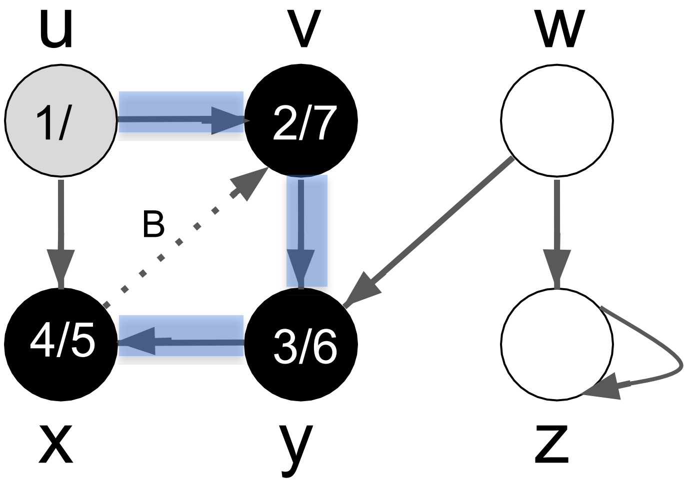
DFS Example



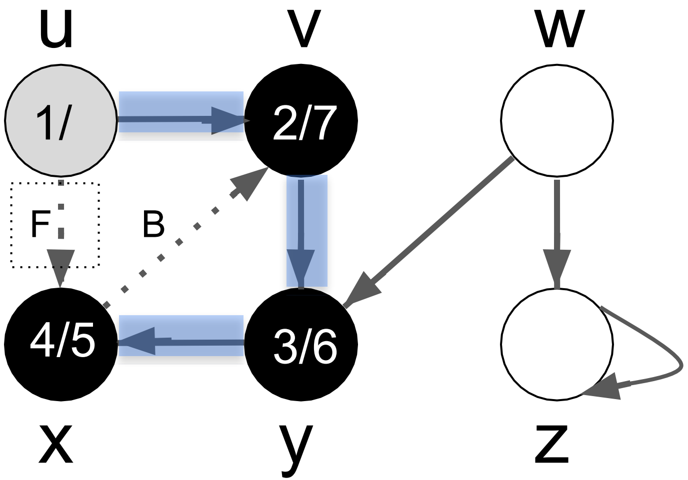
DFS Example



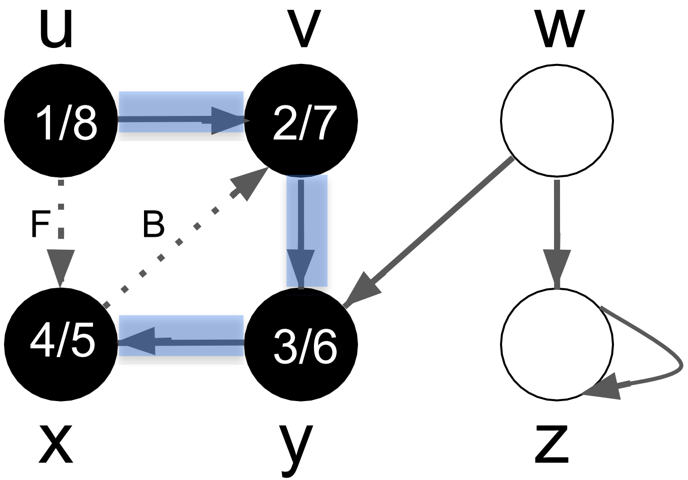
DFS Example



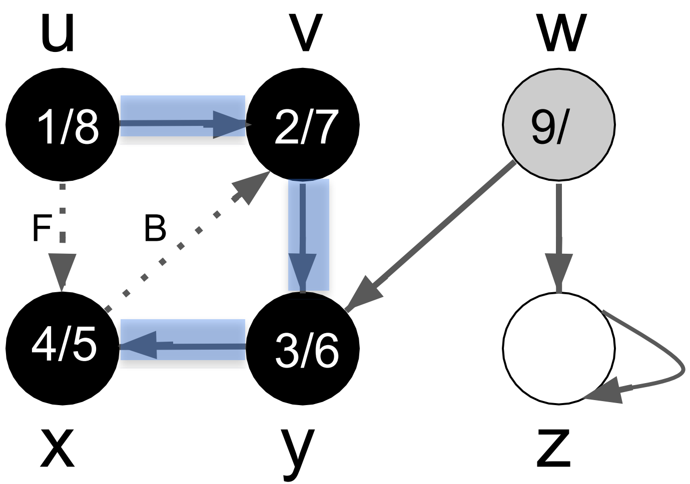
DFS Example



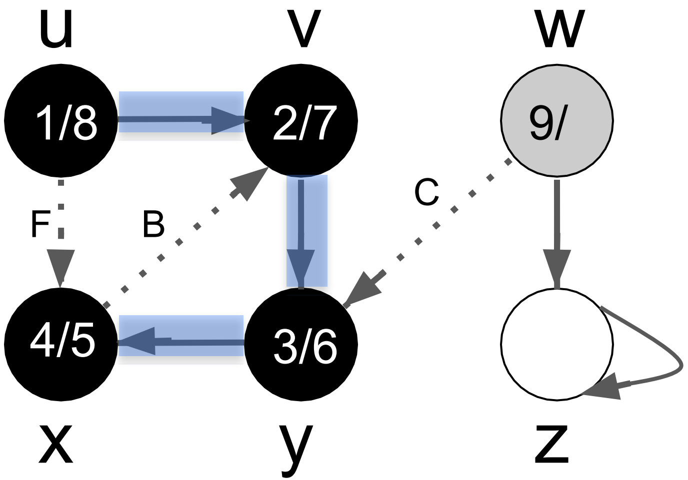
DFS Example



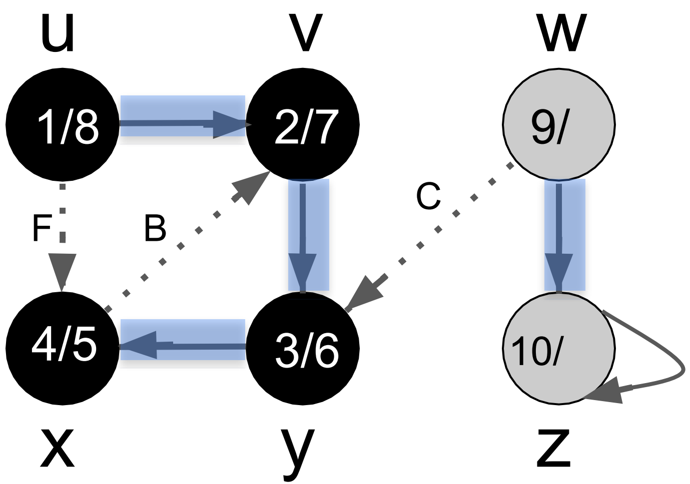
DFS Example



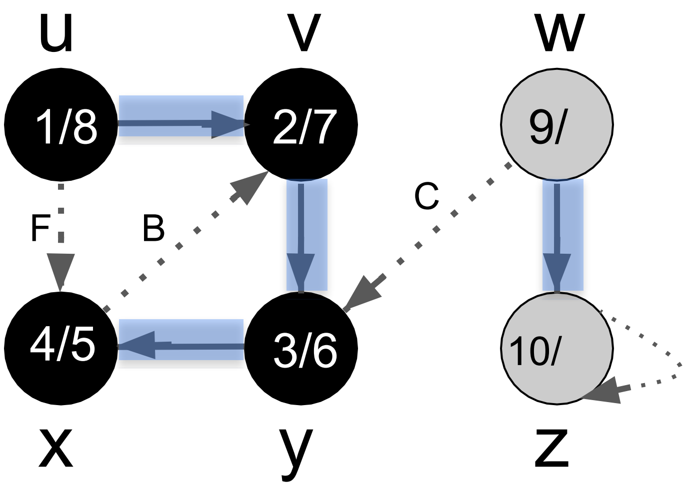
DFS Example



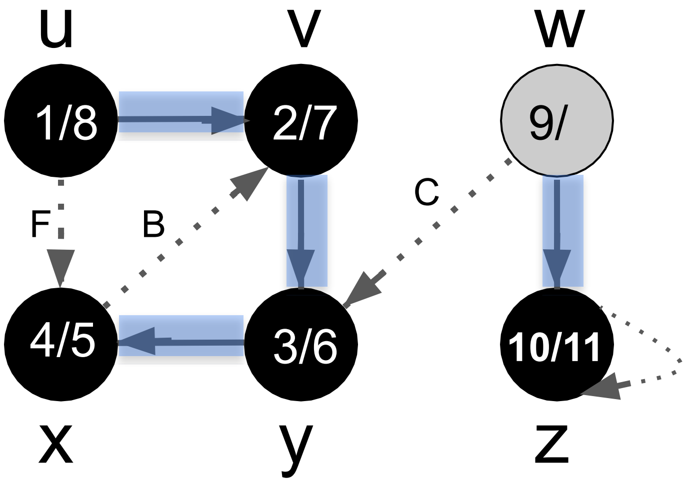
DFS Example



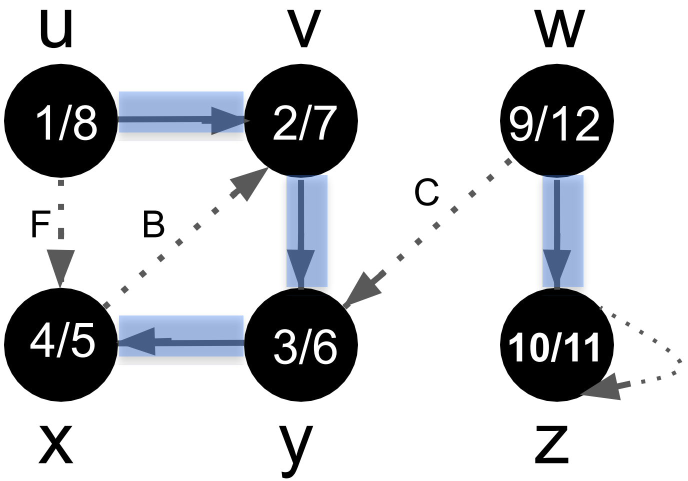
DFS Example



DFS Example

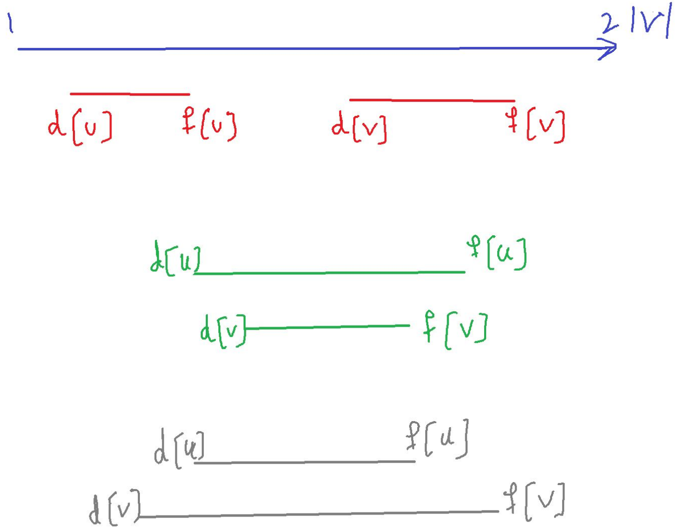


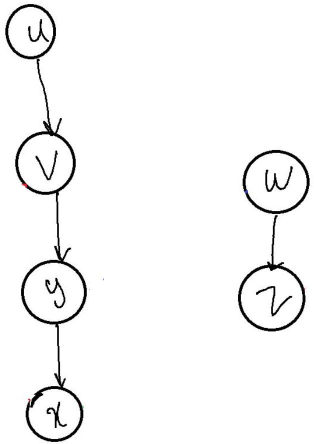
DFS Example

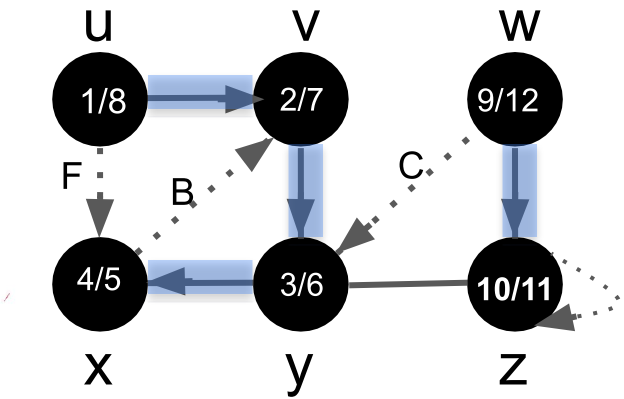


DFS properties: Parenthesis theorem

- In any DFS of a graph G, for any two vertices **u** and **v**, the intervals [d[u], f[u]] anad [d[v], f[v]] are either nested or disjoint
  - If two intervals overlap, then one is nested within the other: no partial overlap
  - Vertex corresponding to smaller interval is a descendant of the vertex corresponding to the larger one

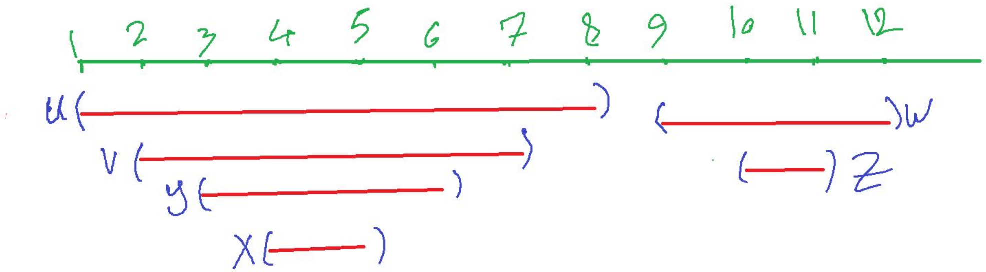


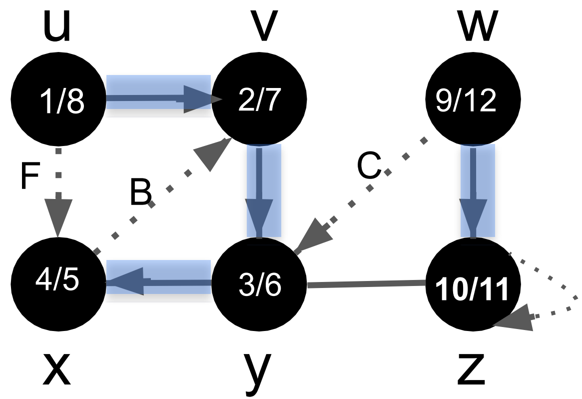




DFS properties

- Why is it called *parentheses* theorem
  - discovery time of a vertex: (
  - finish time of a vertex:)
  - The history of discoveries and finishing times makes a well-formed expression
    - Parenthesis are properly nested





DFS Properties

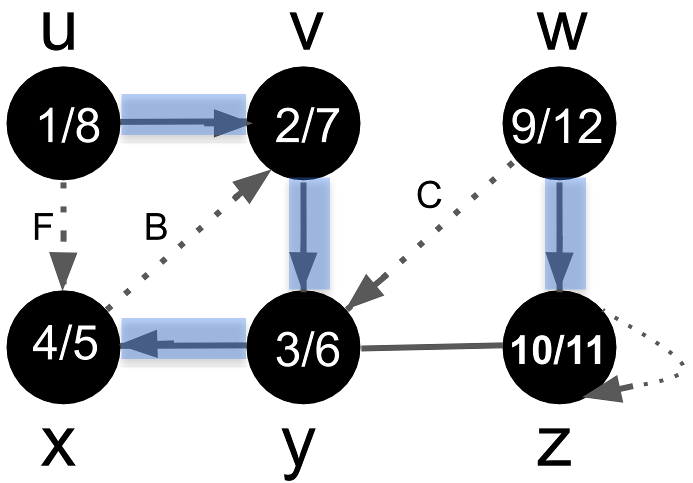
- Vertex v is a proper descendant of vertex u in the depth-first forest of graph G if and only if $d[u] < d[v] < f[v] < f[u]$
  - Follows from parenthesis theorem

DFS: Classification of edges: directed graph

- When we first explore an edge (u, v), the color of vertex v determine edge class
- WHITE: a tree edge
- GRAY: a back edge
- BLACK: a forward or cross edge
  - d[u] < d[v]
    - Forward edge
  - d[u] > d[v]
    - Cross edge

DFS: Classification of edges: undirected graph

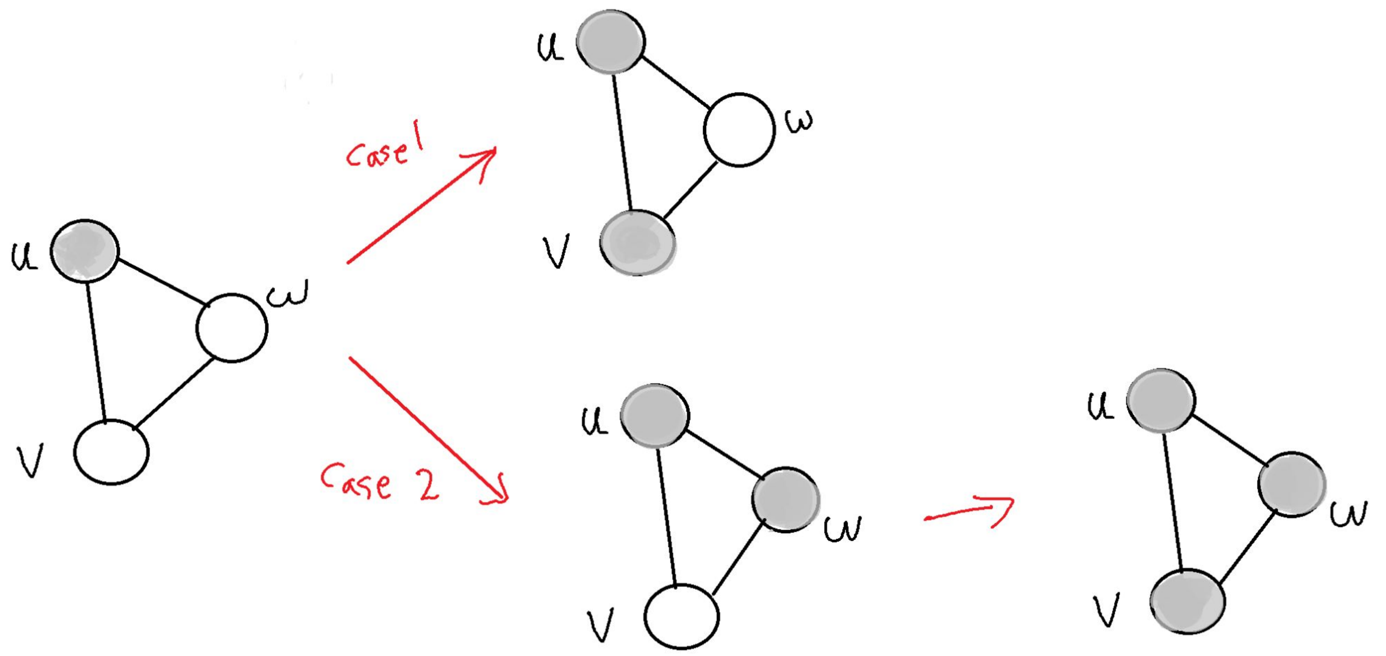
- In a DFS of an undirected graph G, every edge of the tree:
  - Tree edge
  - Back edge
- There are no cross or forward edges



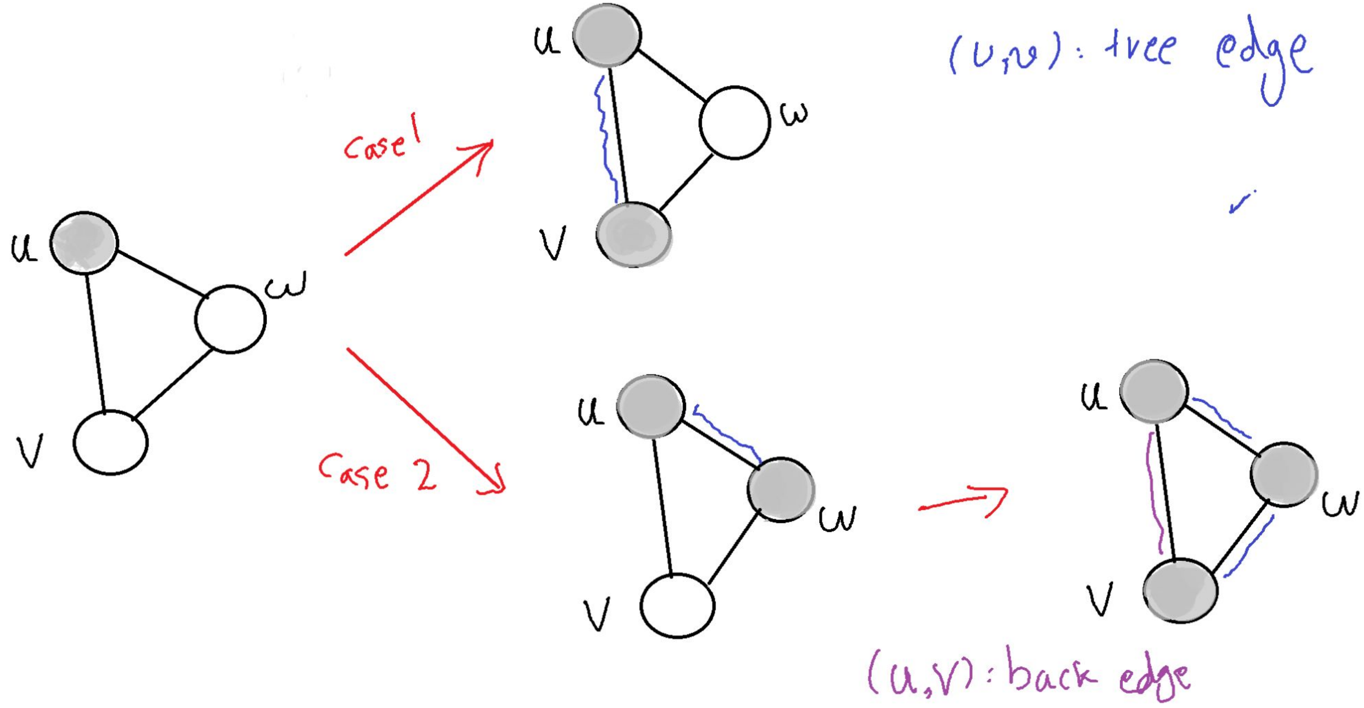
DFS: Classification of edges: undirected graph

- In a DFS of an undirected graph G, every edge of the tree:
  - Tree edge
  - Back edge
- There are no cross or forward edges
- Proof. Let (u, v) be an edge in G and assume WLOG d[u] < d[v]
  - d[u] < d[v]
    - u becomes gray first, vertex v is finished before u
    - Once u becomes grays there will be two possibilities
      - The neighbor v is discovered → (u,v) becomes a tree edge
      - Some other neighbor w will be discovered and that neighbor discovers v → (u, v) becomes a back edge

DFS: Classification of edges: undirected graph

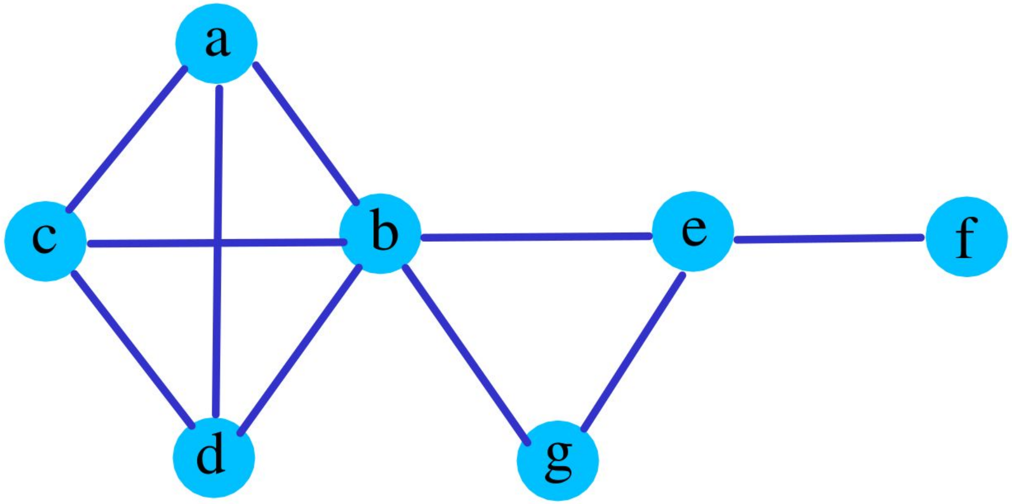


DFS: Classification of edges: undirected graph



Cut vertices and Cut Edges

- Definitions
  - A vertex v is a cut vertex if $G - v$ is not a connected graph.
  - An edge e is a cut edge if $G - e$ is not a connected graph.
- Example:
  - Cut vertices: b, e
  - Cut edges: (e, f)



Cut vertices and Cut Edges

- Problem. Designing an algorithm to identify cut vertices
  - **Input:** Graph G = (V, E)
  - **Output**: identify all cut vertices and cut edges of G

- Brute-force solution:
  - for all v in V
    - Compute $G - \{v\}$, check whether $G - \{v\}$ connected or not
  - Time: O(n (n+m))
  - n=|V|
  - m=|E|

Application of DFS

Detecting cycles in directed graphs

Lemma. A directed graph has a (directed) cycle iff DFS has a back edge.