Efficient Algorithms

Ekkapot Charoenwanit

Software Systems Engineering
TGGS
KMUTNB

Lecture 9: Graph Algorithms (Part I)

Breadth-First Search (BFS)

<u>Problem Definition</u>:

Given an ***unweighted*** graph $G = (V, E)$, find ***a shortest path*** from a given vertex s to every other vertex ***reachable*** from s.

<u>Key Idea</u>:

BFS is a graph traversal algorithm that explores each vertex $v \in V$ in the order of their distance from s (referred to as the ***root***), where distance $\delta(s, v)$ is defined as the *** length*** of a shortest path from s to v.

*******length = the number of edges***

Breadth-First Search (BFS)

BFS is named so because it expands the ***frontier*** between ***discovered*** and ***undiscovered*** vertices ***uniformly across the breadth*** of the frontier.

In other words, the algorithm of BFS discovers all vertices at distance k before they discover any at distance $k + 1$.

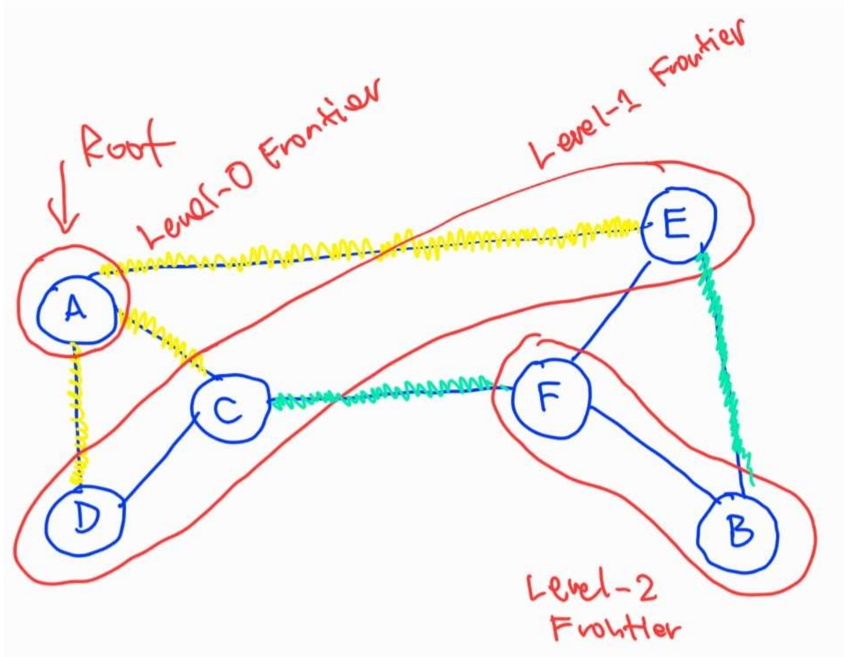
Breadth-First Search (BFS): Example

BFS traverses the graph, starting from A.

A itself forms the level-0 frontier.

C, D and E form the level-1 frontier.

B and F form the level-2 frontier.

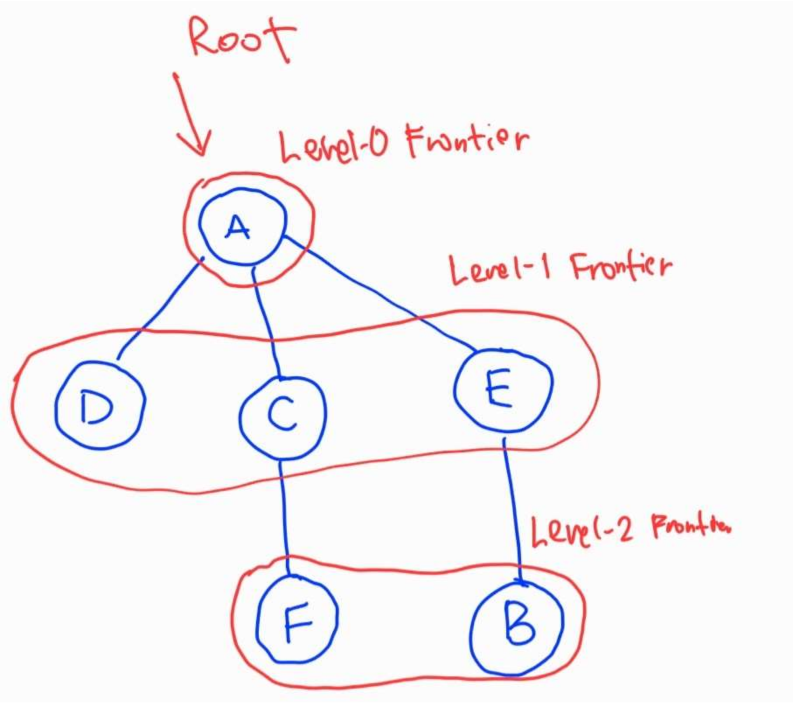


Breadth-First Search (BFS): Example

Not only does BFS find a shortest path from A to every other vertex, but it also produces a ***BFS tree***.

Note: BFS trees are not necessarily unique for all problem instances.

For this particular example, a different BFS tree would have been produced if E had been explored before C.



Breadth-First Search (BFS): Color

BFS ***colors*** each vertex to keep track of its progress.

- ***White*** = undiscovered
- ***Gray*** = discovered and still in the frontier
- ***Black*** = discovered but not in the frontier any more

All vertices start out white (***Line 2***), except for the root, which starts out gray (***Line 5***).

```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2       u.color = WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Breadth-First Search (BFS): Distance

We initialize the ***d-value*** to ∞ (***Line 3***), except for the ***d-value*** of the root s, which is set to *0* (***Line 6***).

Note:

- For any vertex v reachable from s, its d-value will ***converge*** to $\delta(s, v)$.
- For any vertex v not reachable, its d-value will stay ∞.

```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2       u.color = WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Breadth-First Search (BFS): Parent

We initialize the ***parent-value*** of all vertices to NIL (***Line 4***).

Note:

- For any vertex v reachable from s, its parent-value will be set to some predecessor vertex u. (***Line 16***)
- For any vertex v not reachable, its parent-value will stay NIL.
- For the root s, its parent-value will stay NIL.

The resulting BFS tree can be constructed based on these parent values found.

```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2       u.color = WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Breadth-First Search (BFS)

BFS relies on a ***FIFO queue*** Q to generate frontiers in a ***level-by-level*** manner.

BFS operates as follows:

- it adds the root s to Q, which is initially empty
- It removes vertex u from the head of Q and scans its adjacency list $Adj[u]$
- whenever the search discovers a white vertex v during the scanning of the adjacency list $Adj[u]$,
  - it colors v gray and updates its distance
  - it also adds the edge (u, v) to the BFS tree by setting $v.\pi$ to u
  - it then adds v to Q
- after $Adj[u]$ is completely scanned, it colors u black
- the search continues as long as Q is not empty

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Breadth-First Search (BFS): Analysis

We use ***aggregate analysis*** as follows.

After initialization, no vertices are colored white a second time.
This ensures that each vertex is enqueued at most once (***Line 17***),
and hence dequeued at most once as a consequence.

∴ The while loop executes at most $|V|$ iterations.
The operations of enqueuing and dequeuing cost $O(1)$ time.
The total time devoted to queue operations is $O(V)$.

```
BFS(G, s)
1   for each vertex u ∈ G.V − {s}
2       u.color = WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Breadth-First Search (BFS): Analysis

BFS scans the adjacency list of each dequeued vertex at most once

Therefore, the total time spent on examining adjacent vertices is $O(E)$.

The total running time is $O(V + E)$.

Note: For undirected graphs, the number of edges is $2E$.

For directed graphs, the number of edges is E.

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

Breadth-First Search (BFS): Correctness

Lemma I: Let $G = (V, E)$ be a directed or undirected graph, and let s be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1$$

Proof:

Case I: If u is reachable from s, so is v since there is at least the path from s to u followed by the edge (u, v).

In this case, a shortest from s to v cannot be longer than a shortest path from s to u followed by the edge (u, v).

Then, $\delta(s, v) \leq \delta(s, u) + 1$ holds.

Breadth-First Search (BFS): Correctness

Case II: If u is not reachable from s, then $\delta(s, u) = \infty$.

If v is reachable from s, $\delta(s, v)$ is a finite number.
Then, $\delta(s, v) \leq \delta(s, u) + 1$ holds.

If v is not reachable from s, $\delta(s, v) = \infty$.
Then, $\delta(s, v) \leq \delta(s, u) + 1$ still holds. ■

Note: $\delta(s, v) \leq \delta(s, u) + 1$ is a special case of the ***triangle inequality*** $\delta(s, v) \leq \delta(s, u) + w(u, v)$ used in the process of ***edge relaxation*** in shortest-path algorithms for ***weighted*** graphs we will be studying.

Breadth-First Search (BFS): Correctness

Lemma II: Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G with a given source vertex s. Then, upon termination, for all $v \in V$, $v.d \geq \delta(s, v)$.

Proof: We will prove by induction that this property holds prior to the start of each iteration.

Base Case: Prior to the 1^{th} iteration,

$v.d = \infty$ for all $v \neq s \rightarrow v.d \geq \delta(s, v)$

$s.d = 0$ and $\delta(s, s)=0 \rightarrow s.d \geq \delta(s, s)$

Induction Hypothesis: Assume true for any k^{th} iteration.

Breadth-First Search (BFS): Correctness

Inductive Step: We will show true for the $k + 1^{th}$ iteration.

Suppose a vertex $u \in V$ is dequeued at $k + 1^{th}$ iteration.

For all $v \in Adj[u]$ that are white,

$\delta(s, v) \leq \delta(s, u) + 1$	[***Lemma I***]	---------(1)
$u.d \geq \delta(s, u)$	[***I.H.***]	---------(2)
$u.d + 1 \geq \delta(s, u) + 1$	[Adding ***1*** on both sides of (2)]	---------(3)
$\delta(s, v) \leq u.d + 1$	[Transitivity of (1) and (2)]	---------(4)
$v.d = u.d + 1$	[***Line 15***]	---------(5)
$\delta(s, v) \leq v.d$	[Substituting(4) into (3)]	

For all the other v (including u),

$\delta(s, v) \leq v.d$	[***I.H.***]	■

Breadth-First Search (BFS): Correctness

Lemma III:

Suppose during the execution of BFS, the queue Q contains the vertices $< v_1, v_2, \dots, v_r >$, where v_1 is the head of Q and v_r is the tail.

Then, the following loop invariant properties hold prior to the start of each queue operation:

(***P1***) $v_r.d \le v_1.d + 1$

(***P2***) $v_i.d \le v_{i+1}.d$ for all $i = 1,2, \dots r-1$.

Proof: We will prove by induction on the number of queue operations.

Base Case: Initially, Q contains only the root s.

In other words, $v_1 = v_r = s$.

P1 holds since $s.d \le s.d + 1$.

P2 vacuously holds.

Breadth-First Search (BFS): Correctness

Induction Hypothesis: Assume true for prior to any queue operation.

Inductive Step: We will show that the invariant still holds prior to the next queue operation.

Since there are two kinds of queue operations, namely, ***enqueue*** and ***dequeue***, we shall consider the following ***two cases*** separately:

Case I: Suppose $Q = < v_1, v_2, \ldots, v_r >$ and v_1 is the head. Hence, v_1 is dequeued and v_2 becomes the head and $Q = < v_2, \ldots, v_r >$.

We will have to show

P1 $\quad v_r.d \leq v_2.d + 1$

P2 $\quad v_2.d \leq v_3.d \leq \ldots \leq v_r.d$

Breadth-First Search (BFS): Correctness

If Q becomes empty, ***P1*** and ***P2*** vacuously hold.

Otherwise,

$v_1.d \leq v_2.d$	[***I.H.***]	---------(1)
$v_1.d + 1 \leq v_2.d + 1$	[Adding *1* on both sides of (1)]	---------(2)
$v_r.d \leq v_1.d + 1$	[***I.H.***]	---------(3)
$v_r.d \leq v_2.d + 1$	[Transitivity of (2) and (3)]	

P1 still holds right after a dequeue operation.

P2 still holds since the values of $v_2.d$, $v_3.d$,…, $v_r.d$ are not affected.

Then, ***P1 &P2*** hold after a dequeue operation.

Breadth-First Search (BFS): Correctness

Case II: Suppose $Q = < v_1, v_2, \dots, v_r >$, and $v = v_{r+1}$ is added and becomes the tail.

Then, $Q = < v_1, \dots, v_r, v_{r+1} >$. We will have to show

P1 $\quad v_{r+1}.d \leq v_1.d + 1$

P2 $\quad v_1.d \leq v_3.d \leq \dots \leq v_{r+1}.d$

Let's consider what happens in the code.

v is enqueued (***Line 17***) as part of the scanning of the adjacency list of some vertex $u = v_0$ that has just been dequeued by ***Line 11***.

$v_0.d \leq v_1.d$	[***I.H.***]	---------(1)
$v_0.d + 1 \leq v_1.d + 1$	[Adding *1* on both sides of (1)]	---------(2)
$v_0.d + 1 = v_{r+1}.d$	[***Line 15***]	---------(3)
$v_{r+1}.d \leq v_1.d + 1$		

Thus, ***P1*** holds.

Breadth-First Search (BFS): Correctness

Case II:

$$v_r.d \le v_0.d + 1 \qquad [\textit{I.H.}] \qquad \text{---------(1)}$$

$$v_r.d \le v_{r+1}.d \qquad [v_{r+1}.d = v_0.d + 1] \qquad \text{---------(2)}$$

P2 holds since the values of $v_1.d$, $v_2.d$,…, $v_r.d$ are not affected so the inequalities $v_1.d \le v_2.d \le \cdots \le v_r.d$ still hold, and $v_r.d \le v_{r+1}.d$ additionally holds.

Then, ***P1 &P2*** hold after an enqueue operation.

This proves the lemma. ∎

Immediate Implication: The lemma immediately implies that the ***d-values*** at the time that vertices are ***enqueued*** are ***monotonically increasing over time***.

Breadth-First Search (BFS): Correctness

Theorem: Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G with a given source vertex s.

(***Claim I***) Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from s, and, upon termination, $v.d = \delta(s, v)$ for all $v \in V$.

(***Claim II***) Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

Proof:

Claim I : We will first show that $v.d = \delta(s, v)$ for all $v \in V$.
Assume for the purpose of contradiction some vertex v receives a value not equal to its shortest-path distance.

Let v be the vertex with the minimum $\delta(s, v)$ that receives such an incorrect value of $v.d$; clearly, $v \neq s$ since $s.d$ is correctly initialized to zero (***Line 6***).

That is,

$$v.d \neq \delta(s, v).$$

Breadth-First Search (BFS): Correctness

$v.d \geq \delta(s, v)$ [***Lemma II***] ---------(1)

$v.d > \delta(s, v)$ [(1) & Assumption $v.d \neq \delta(s, v)$] ---------(2)

Let u be the vertex immediately preceding v on a shortest path from s to u.

$$\delta(s, v) = \delta(s, u) + 1$$

$$\delta(s, u) < \delta(s, v)$$

We chose v such that v is the vertex with the minimum distance that received an incorrect value of $v.d$; any vertex u preceding v on the corresponding shortest path must have received the correct value.

$$u.d \ = \delta(s, u)$$

Putting these properties together,

$$\begin{aligned} v.d &> \delta(s, v) \\ &= \delta(s, u) + 1 \\ &= u.d + 1 \end{aligned}$$

$v.d > u.d + 1$ ---------(*)

Breadth-First Search (BFS): Correctness

Consider the time when BFS chooses to dequeue u from Q in ***Line 11***.

At this time, v is either white, gray or black. We will show that, in all the three cases, we will arrive at a contradiction to (*).

Case I: v is white.

$$v.d = u.d + 1 \qquad \text{[\textit{Line 15}]}$$

Hence, we arrive at a contradiction to (*).

Case II: v is black; v has been dequeued.

Then, $v.d \leq u.d$ [***Lemma III***]

Hence, we arrive at a contradiction to (*).

Breadth-First Search (BFS): Correctness

Case III: v is gray; Hence, we arrive at a contradiction to (*).

was marked gray upon dequeuing some vertex w, which was removed from Q earlier than u.

Then	$v.d = w.d + 1$	[***Line 15***]	---------(1)
	$w.d \le u.d$	[***Lemma III***]	---------(2)
	$w.d + 1 \le u.d + 1$	[Adding *1* on both sides of (2)]	---------(3)
	$v.d \le u.d + 1$	[Transitivity of (1) and (3)]	

Hence, we arrive at a contradiction to (*).

Therefore, such a vertex with $v.d \ne \delta(s, v)$ does not exist.

Claim I $v.d = \delta(s, v)$ for all $v \in V$ holds. ■

Breadth-First Search (BFS): Correctness

Claim II: we will show, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

Observe that If $v.\pi = u$,

$v.d = u.d + 1$ [***Lines 15 & 16***]

This means if there is a path from s to v, we can obtain a shortest path from s to v by taking a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$. ■

Depth-First Search (DFS)

Problem Definition:

Given an ***unweighted*** graph $G = (V, E)$, find ***a path*** from a given vertex s to every other vertex ***reachable*** from s.

Key Idea:

DFS is a graph traversal algorithm that explores all outgoing edges of the most recently visited vertex v. Once all of v's outgoing edges have been explored, the algorithm ***backtracks*** to explore edges leaving the vertex from which v was discovered.

Depth-First Search (DFS) : Parent

As in BFS, whenever DFS discovers a vertex v during a scan of its adjacency list of the most recently discovered vertex u, it records this event by setting $v.\pi = u$.

Unlike BFS whose predecessor subgraph forms a tree, the predecessor subgraph produced by DFS forms a ***forest*** (multiple trees), because the search may be repeated from ***multiple sources***.

We can define the ***predecessor subgraph*** of DFS as follows:

$$G_\pi = (V, E_\pi)$$
$$E_\pi = \{(v.\pi, v) : v \in V \ \wedge v.\pi \neq \text{NIL}\}$$

We call the edges $(v.\pi, v)$ in E_π ***tree edges***.

Depth-First Search (DFS): Color

DFS relies on a similar ***vertex-coloring scheme*** to that of BFS as follows:

- Initially, Each vertex is white.
- It becomes gray when it is discovered during the search.
- Eventually, it becomes black when it is finished, i.e., when its adjacency list is completely explored.

This coloring scheme can also ensures that each vertex ends up in ***exactly one*** DFS tree so that all the resulting DFS trees are ***disjoint***.

Depth-First Search (DFS) : Timestamp

Associated with each vertex v are ***two timestamps*** $v.d$ and $v.f$:

- The first timestamp $v.d$ records when v is first ***discovered*** and ***grayed***.
- The second timestamp $v.f$ records when DFS finishes exploring v's adjacency list and ***blackens*** v.

These timestamps are helpful in reasoning about the behavior and correctness of DFS.

Depth-First Search (DFS) : Color and Timestamp

Timestamps can be implemented using ***integers*** between 1 and $2|V|$.

For every vertex v,

$$v.d < v.f$$

v is white before $v.d$, gray between $v.d$ and $v.f$ and black after $v.f$.

Depth-First Search (DFS) : Pseudocode

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1               // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]         // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK               // blacken u; it is finished
time = time + 1
u.f = time
```

Depth-First Search (DFS) : Pseudocode

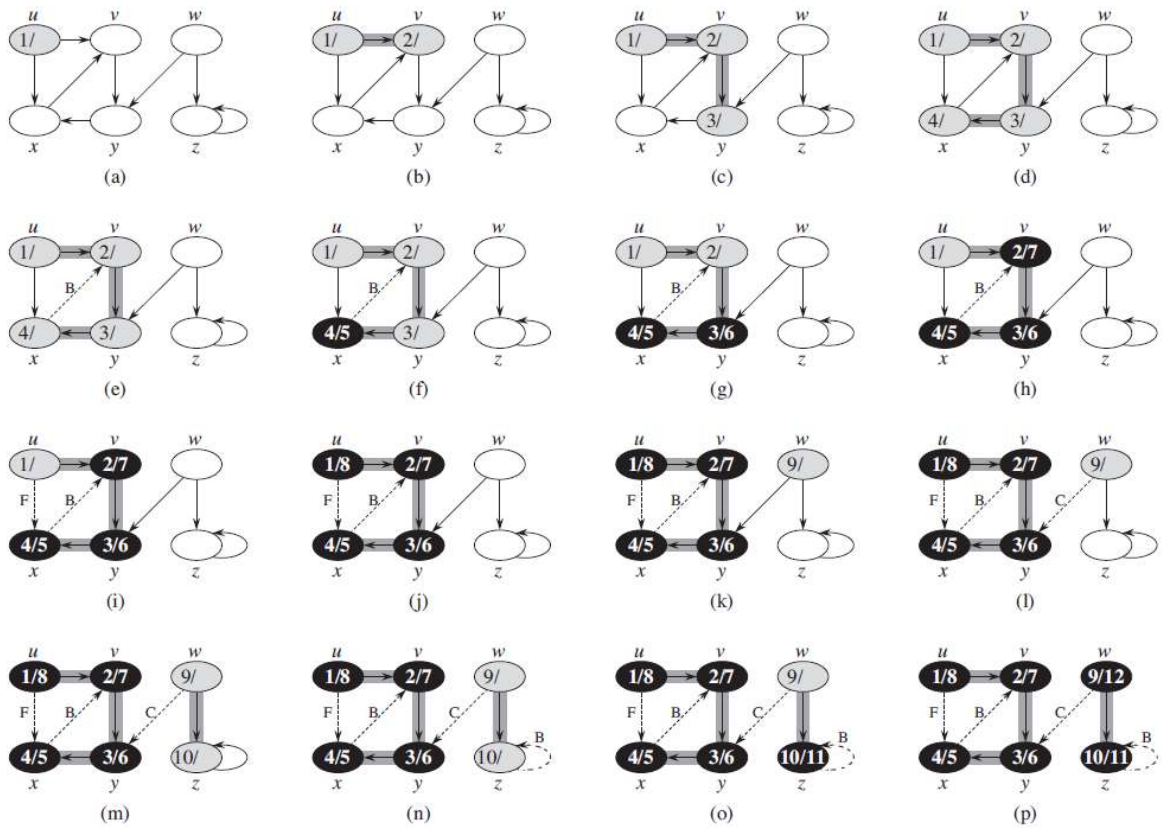
DFS works as follows:

- All vertices are initially painted white with their parent values set to NIL.
- The ***global time counter*** is reset.
- DFS checks each vertex u and explores v using $DFS - Visit(G, u)$ if u is white
- Every time $DFS - Visit(G, u)$ is called, u becomes the root of a new tree.
- When DFS returns, all vertices u are assigned with $u.d$ and $u.f$

$DFS - Visit$ works as follows:

- In each call $DFS - Visit(G, u)$, u is initially white
- Increment the global time counter and record it as the discovery time of u
- u is then painted gray
- u's adjacency list is recursively explored
    - For each $v \in Adj[u]$ and v is white, explore v
- After all edges leaving u are explored, blacken it increment the time counter and record it as the finishing time of u

Depth-First Search (DFS) : Example



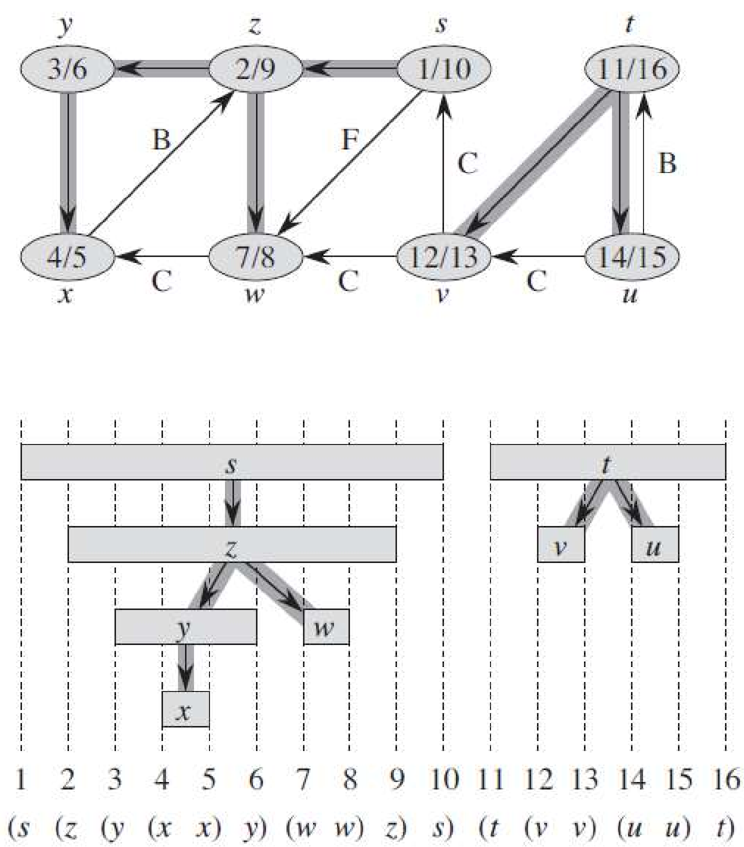
Depth-First Search (DFS) : Parenthesis Structure

In DFS, discovery and finishing times exhibit ***parenthesis structure***.

In other words, if we represent the discovery of a vertex u with a ***left parenthesis*** $(u$ and represent its finishing time with a ***right parenthesis*** $u)$.

Then, the history of discovering and finishing all vertices makes a ***well-form expression*** in that the parentheses are ***properly nested***.

Depth-First Search (DFS) : Parenthesis Structure



Depth-First Search (DFS) : Parenthesis Structure

Parenthesis Theorem: In any depth-first search of a directed or undirected graph $G = (V, E)$, for any two vertices u and v, exactly one of the following conditions holds.

- 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 resulting DFS forest.
- The interval $[u.d, u.f]$ is contained entirely in the interval $[v.d, v.f]$, u is a descendant of v in a DFS tree.
- The interval $[v.d, v.f]$ is contained entirely in the interval $[u.d, u.f]$, v is a descendant of u in a DFS tree.

Depth-First Search (DFS) : Parenthesis Structure

Lemma (Nesting of Descendants' Intervals): Vertex v is a proper descendant of vertex u in the DFS forest for a directed or undirected graph $G = (V, E)$ if and only if $u.d < v.d < v.f < u.f$.

Proof: The lemma follows Immediately from ***Parenthesis Theorem.*** ■

Depth-First Search (DFS) : Parenthesis Structure

White-path Theorem: In a DFS forest of a directed or an undirected graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a ***white path*** from u to v consisting entirely of white vertices.

Proof:

⇒: If $u = v$, then the path from u to v contains just u, which is still white when we set the value of $u.d$.

Now suppose v is a proper descendant of u.

$$u.d < v.d$$ [***Nesting of Descendants' Intervals***]

Then, v is white at time $u.d$.

Since v can be any descendant of u, all vertices on the unique path from u to v are white at time $u.d$.

Depth-First Search (DFS) : Parenthesis Structure

$\Leftarrow$: Suppose there is a path from of white vertices from u to v at time $u.d$, but ***FPOC***, v does not become a descendant of u in the resulting DFS tree.

Assume wlog that the other vertices than v along the path become a descendant of u.

Let vertex w be a predecessor of v on that path so w is a descendant of u (u and w may be the same vertex).

Then, $w.f \leq u.f$ [***Nesting of Descendants' Intervals***]

Because v must be discovered after u is discovered, but before w is finished, we have

$$u.d < v.d < w.f \leq u.f$$

Parenthesis Theorem implies that the interval $[v.d, v.f]$ is entirely contained within $[u.d, u.f]$.

Hence, v is a descendant of u after all by ***Nesting of Descendants' Intervals***. ■

Depth-First Search (DFS) : Edge Classification

Another property of DFS is that the search can be used to ***classify*** the edges in the input graph $G = (V, E)$.

The type of each edge can provide information about the graph.

We can define ***four*** edge types in terms of the DFS forest G_π produced by DFS on G as follows:

- ***Tree edges*** are edges in G_π. Edge (u, v) is a tree edge if v was first discovered by exploring (u, v).
- ***Back edges*** are those edges (u, v) connecting a vertex u to a ancestor v in a DFS tree.
- ***Forward edges*** are those non-tree edges connecting a vertex u to a descendant v in a DFS tree.
- ***Cross edges*** are all other edges that go between vertices that are not ***ancestor-descendant-related***.

Depth-First Search (DFS) : Edge Classification

The DFS algorithm has enough information to classify some edges as it encounters them.

The key idea is that when we first explore an edge (u, v), the color of vertex v tells us something about the edge:

- ***White*** indicates a ***tree edge***
- ***Gray*** indicates a ***back edge***
- ***Black*** indicates a ***forward*** or a ***cross edge***

To distinguish between forward and cross edges,

(u, v) is a forward edge if $u.d < v.d$

(u, v) is a cross edge if $u.d > v.d$

******In undirected graphs, there are only two types of edges, namely, tree and back edges.***

Depth-First Search (DFS) : Edge Classification

Theorem: In a DFS tree of an undirected graph $G = (V, E)$, every edge of G is either a tree edge or a back edge.

Proof: Let (u, v) be an arbitrary edge of G and suppose wlog that $u.d < v.d$.

The search must discover and finish v before it finishes u while u is still gray, since v is on u's adjacency list.

If the first time that the search explores (u, v), it is in the direction from u to v, then v is undiscovered (white). Otherwise, the search would have explored the search already in the opposite direction from v to u.

Thus,(u, v) becomes a tree edge.

If the search explores (u, v) first in the direction from v to u, then (u, v) is a back edge, since u is still gray at the time the edge is first explored. ■

Topological Sorting

DFS can be used to perform a ***topological sort*** of a ***directed acyclic graph*** (***DAG***).

Definition: A topological sort of a DAG $G = (V, E)$ is a linear (partial) ordering of all its vertices such that G contains an edge (u, v), then u appears before v in that ordering.

```
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
```

Note: If a graph is not a DAG, such a linear ordering cannot be constructed among all vertices.

Directed Acyclic Graph

Lemma: If there is a back edge if and only if G contains a cycle.

Proof:

$\Rightarrow$: If there is a back edge (u, v), that means there is a path from v to u.

Thus, the back edge (u, v) completes a cycle.

$\Leftarrow$: Suppose G contains a cycle c.

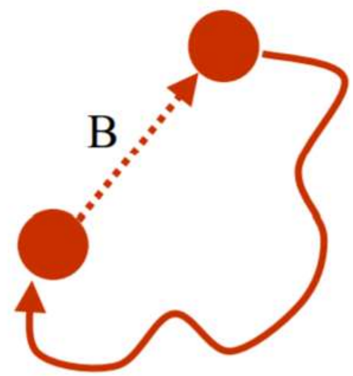
Let v be the first vertex to be discovered in the cycle c.

Let (u, v) be the preceding edge in the cycle c.

Thus, when v is discovered, all the other vertices in c are still white.

By the ***White-path Theorem***, u becomes a descendant of v.

Therefore, (u, v) is a back edge. ■



Directed Acyclic Graph

Lemma: If there is a back edge if and only if G contains a cycle. [$\mathrm{p} \leftrightarrow q$]

This is logically equivalent to:

Lemma: DFS yields no back edge if and only if G is a DAG. [$\neg\mathrm{p} \leftrightarrow \neg q$]