Basic Graph Algorithms

- Exploring Graphs
- Depth First Search
- DFS Forest - Start and Finish Time
- DAG, Topological Sorting
- Strongly Connected Components
- Breadth First Search
- Bipartite Graphs

IMDAD ULLAH KHAN

Depth First Search: Forest

Algorithm DFS(G)

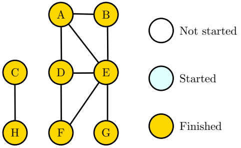
$visited \leftarrow$ ZEROS(n)
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

Algorithm DFS-EXPLORE(s)

function DFS-EXPLORE(s)
  $visited[s] \leftarrow 1$
  for $u \in N(s)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)

When call to DFS-EXPLORE(s) is executed, all vertices in $R(s)$ are visited

- When DFS-EXPLORE(u) is finished one 'DFS *tree*' is formed containing all vertices reachable from u
- The next DFS-EXPLORE(v) called from outer loop forms a new tree



DFS: Review

Algorithm DFS(G)

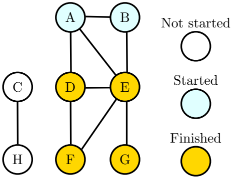
$visited \leftarrow$ ZEROS(n)
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

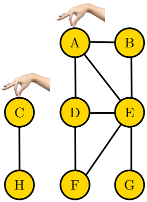
Algorithm DFS-EXPLORE(s)

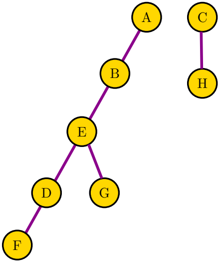
function DFS-EXPLORE(s)
  $visited[s] \leftarrow 1$
  for $u \in N(s)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)

- DFS explores the entire graph
- Explores one neighbor first (in depth), before going to next neighbor
- Works both for undirected and directed graphs
- Runtime of $O(n + m)$ means DFS doesn't add any cost (asymptotically) to any graph algorithm, we typically do it as a pre-processing step
- Answers many questions
  - is graph connected, how many components in the graph, find $R(s)$, $\cdots$
- A fundamental algorithm has many applications we will discuss some
- For applications we need some extra book-keeping with DFS

DFS Forest



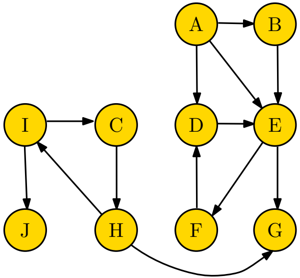


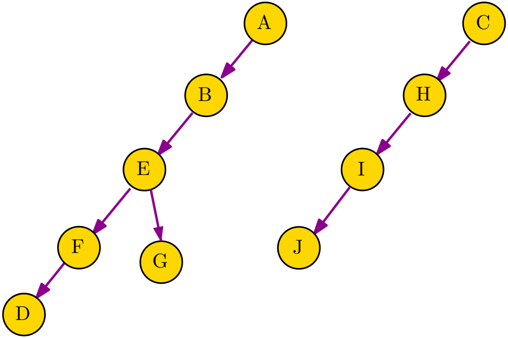


- Record predecessor relationships (save call hierarchy)
- Implicitly build a forest
- Predecessors subgraph (edges used for calling) forms DFS forest
- We first go as deep as we can
- For undirected graphs, each DFS tree is a connected component

DFS Forest: Directed Graphs

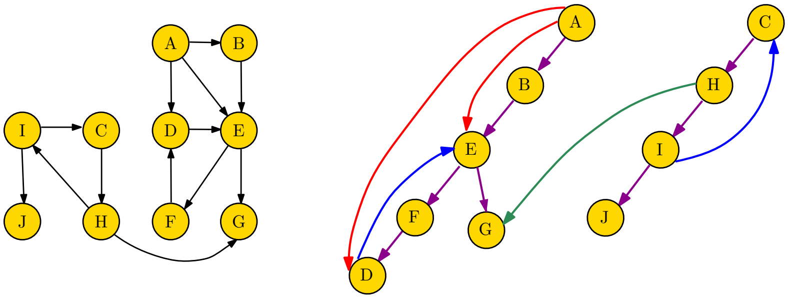
DFS Forest of a digraph





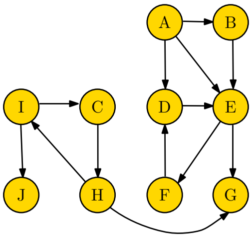
DFS Forest: Types of Edges

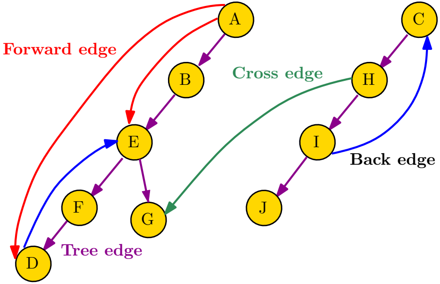
Overlay all edges of a digraph G onto its DFS forest



DFS Forest: Types of Edges

Overlay all edges of a digraph G onto its DFS forest





- Tree edge - Edge used in the DFS (parent to child)
- Back edge - Edge from a node to a non-parent ancestor
- Forward edge - Edge from a node to a non-child descendant
- Cross edge - Edge from a node in one tree to a node in another

DFS Forest: Start and Finish Exploration

- Extra book keeping: timestamps for each vertex
- **start time**: $s[v]$ and **end time**: $f[v]$

Algorithm DFS(G)

```
visited ← ZEROS(n)
time ← 1
for all v ∈ V do
    if visited[v] = 0 then
        DFS-EXPLORE(v)
```

Algorithm DFS-EXPLORE

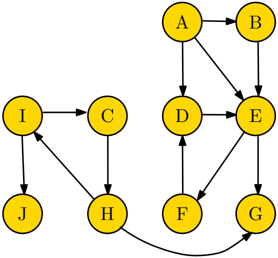
```
function DFS-EXPLORE(v)
    visited[v] ← 1
    s[v] ← time
    time ← time + 1
    for u ∈ N(v) do
        if visited[u] = 0 then
            DFS-EXPLORE(u)
    f[v] ← time
    time ← time + 1
```

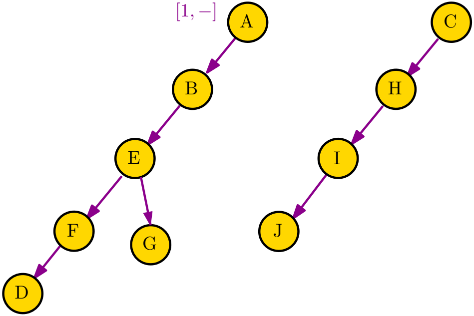
DFS Forest: Start and Finish Time

Algorithm DFS(G)

$\quad$ *visited* $\leftarrow$ ZEROS(n)
$\quad$ *time* $\leftarrow 1$
$\quad$ **for** all $v \in V$ **do**
$\quad\quad$ **if** *visited*$[v] = 0$ **then**
$\quad\quad\quad$ DFS-EXPLORE(v)

function DFS-EXPLORE(v)
$\quad$ *visited*$[v] \leftarrow 1$
$\quad$ $s[v] \leftarrow$ *time*
$\quad$ *time* $\leftarrow$ *time* $+ 1$
$\quad$ **for** $u \in N(v)$ **do**
$\quad\quad$ **if** *visited*$[u] = 0$ **then**
$\quad\quad\quad$ DFS-EXPLORE(u)
$\quad$ $f[v] \leftarrow$ *time*
$\quad$ *time* $\leftarrow$ *time* $+ 1$





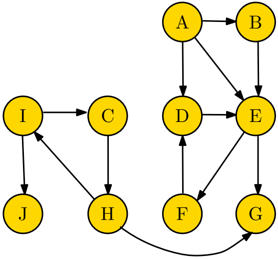
DFS Forest: Start and Finish Time

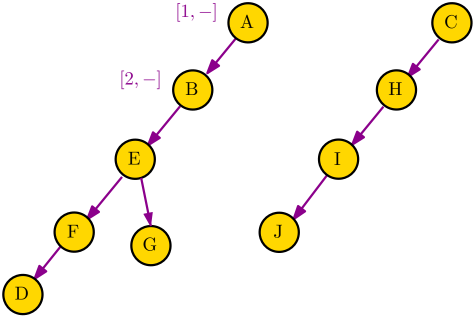
Algorithm DFS(G)

> *visited* $\leftarrow$ ZEROS(n)
> *time* $\leftarrow 1$
> **for** all $v \in V$ **do**
>   **if** *visited*$[v] = 0$ **then**
>   DFS-EXPLORE(v)

function DFS-EXPLORE(v)

> *visited*$[v] \leftarrow 1$
> $s[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$
> **for** $u \in N(v)$ **do**
>   **if** *visited*$[u] = 0$ **then**
>   DFS-EXPLORE(u)
> $f[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$





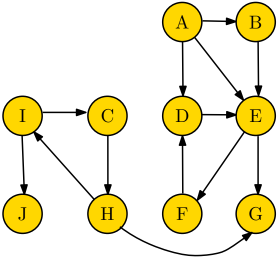
DFS Forest: Start and Finish Time

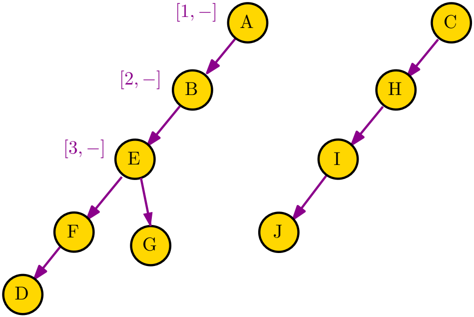
Algorithm $\text{DFS}(G)$

> $visited \leftarrow \text{ZEROS}(n)$
> $time \leftarrow 1$
> **for** all $v \in V$ **do**
>   **if** $visited[v] = 0$ **then**
> $\text{DFS-EXPLORE}(v)$

function $\text{DFS-EXPLORE}(v)$

> $visited[v] \leftarrow 1$
> $s[v] \leftarrow time$
> $time \leftarrow time + 1$
> **for** $u \in N(v)$ **do**
>   **if** $visited[u] = 0$ **then**
> $\text{DFS-EXPLORE}(u)$
> $f[v] \leftarrow time$
> $time \leftarrow time + 1$





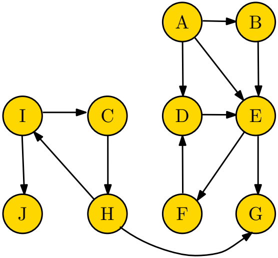
DFS Forest: Start and Finish Time

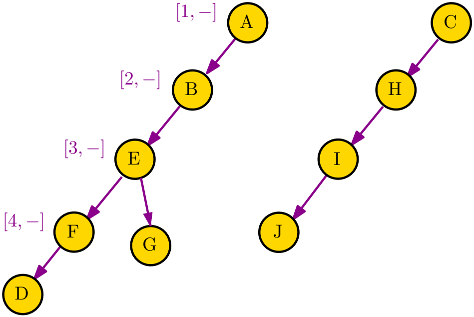
Algorithm DFS(G)

> *visited* $\leftarrow$ ZEROS(n)
> *time* $\leftarrow 1$
> **for** all $v \in V$ **do**
>   **if** *visited*$[v] = 0$ **then**
>   DFS-EXPLORE(v)

function DFS-EXPLORE(v)

> *visited*$[v] \leftarrow 1$
> $s[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$
> **for** $u \in N(v)$ **do**
>   **if** *visited*$[u] = 0$ **then**
>   DFS-EXPLORE(u)
> $f[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$



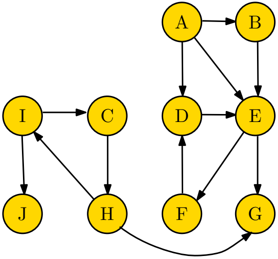


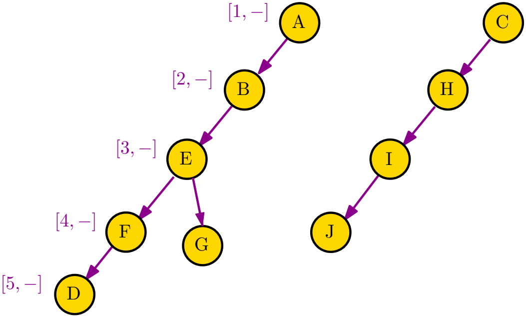
DFS Forest: Start and Finish Time

Algorithm DFS(G)

$visited \leftarrow$ ZEROS(n)
$time \leftarrow 1$
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

function DFS-EXPLORE(v)
  $visited[v] \leftarrow 1$
  $s[v] \leftarrow time$
  $time \leftarrow time + 1$
  for $u \in N(v)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)
  $f[v] \leftarrow time$
  $time \leftarrow time + 1$





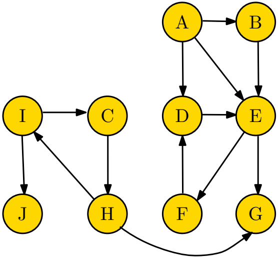
DFS Forest: Start and Finish Time

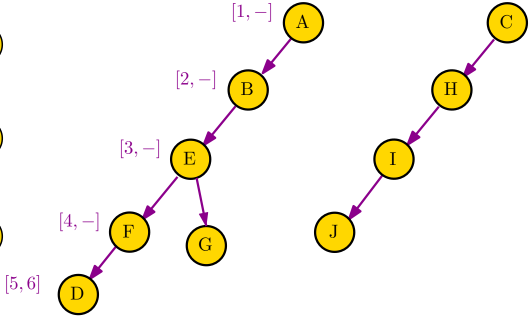
Algorithm DFS(G)

> *visited* ← ZEROS(n)
> *time* ← 1
> **for** all $v \in V$ **do**
>   **if** $visited[v] = 0$ **then**
>   DFS-EXPLORE(v)

function DFS-EXPLORE(v)

> $visited[v] \leftarrow 1$
> $s[v] \leftarrow time$
> $time \leftarrow time + 1$
> **for** $u \in N(v)$ **do**
>   **if** $visited[u] = 0$ **then**
>   DFS-EXPLORE(u)
> $f[v] \leftarrow time$
> $time \leftarrow time + 1$





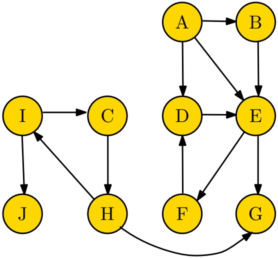
DFS Forest: Start and Finish Time

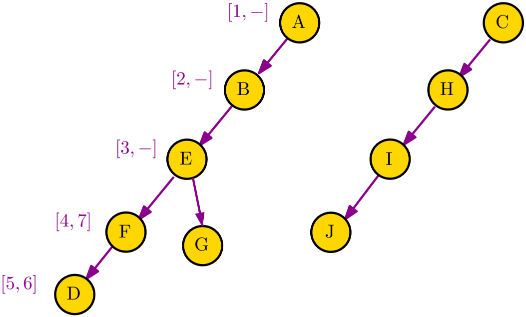
Algorithm DFS(G)

> *visited* $\leftarrow$ ZEROS(n)
> *time* $\leftarrow 1$
> **for** all $v \in V$ **do**
>   **if** *visited*$[v] = 0$ **then**
>   DFS-EXPLORE(v)

function DFS-EXPLORE(v)

> *visited*$[v] \leftarrow 1$
> $s[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$
> **for** $u \in N(v)$ **do**
>   **if** *visited*$[u] = 0$ **then**
>   DFS-EXPLORE(u)
> $f[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$



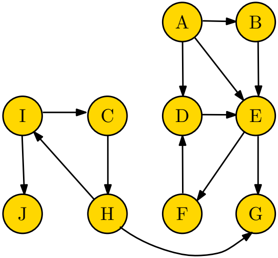


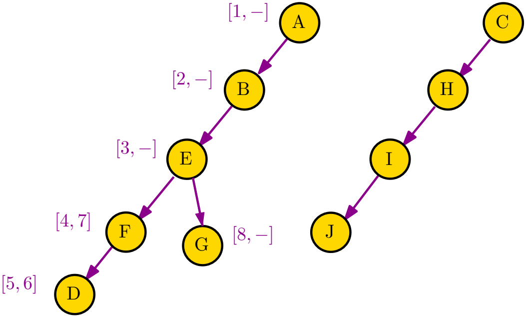
DFS Forest: Start and Finish Time

Algorithm DFS(G)

$visited \leftarrow$ ZEROS(n)
$time \leftarrow 1$
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

function DFS-EXPLORE(v)
  $visited[v] \leftarrow 1$
  $s[v] \leftarrow time$
  $time \leftarrow time + 1$
  for $u \in N(v)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)
  $f[v] \leftarrow time$
  $time \leftarrow time + 1$



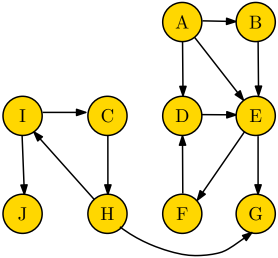


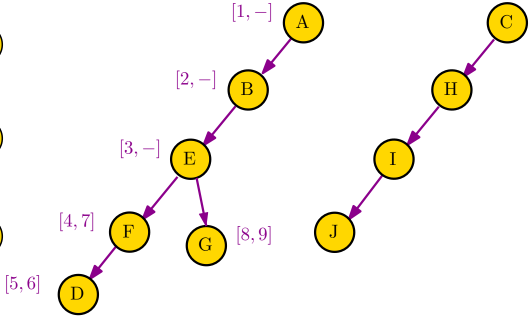
DFS Forest: Start and Finish Time

Algorithm DFS(G)

$visited \leftarrow$ ZEROS(n)
$time \leftarrow 1$
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

function DFS-EXPLORE(v)
  $visited[v] \leftarrow 1$
  $s[v] \leftarrow time$
  $time \leftarrow time + 1$
  for $u \in N(v)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)
  $f[v] \leftarrow time$
  $time \leftarrow time + 1$





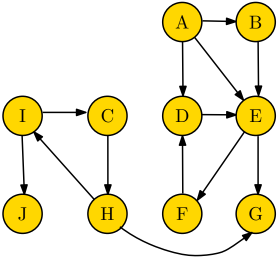
DFS Forest: Start and Finish Time

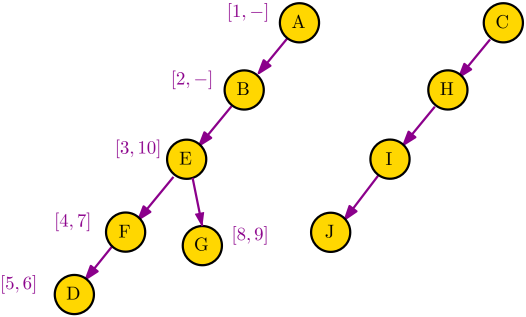
Algorithm DFS(G)

> *visited* $\leftarrow$ ZEROS(n)
> *time* $\leftarrow 1$
> **for** all $v \in V$ **do**
>   **if** *visited*[v] $= 0$ **then**
>   DFS-EXPLORE(v)

function DFS-EXPLORE(v)

> *visited*[v] $\leftarrow 1$
> $s[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$
> **for** $u \in N(v)$ **do**
>   **if** *visited*[u] $= 0$ **then**
>   DFS-EXPLORE(u)
> $f[v] \leftarrow$ *time*
> *time* $\leftarrow$ *time* $+ 1$



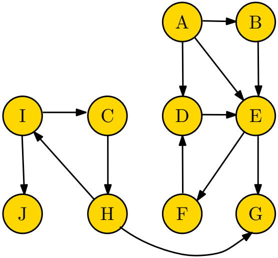


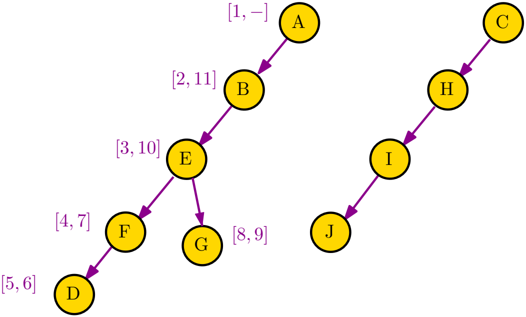
DFS Forest: Start and Finish Time

Algorithm DFS(G)

$visited \leftarrow$ ZEROS(n)
$time \leftarrow 1$
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

function DFS-EXPLORE(v)
  $visited[v] \leftarrow 1$
  $s[v] \leftarrow time$
  $time \leftarrow time + 1$
  for $u \in N(v)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)
  $f[v] \leftarrow time$
  $time \leftarrow time + 1$



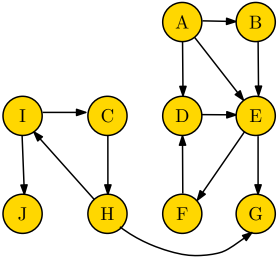


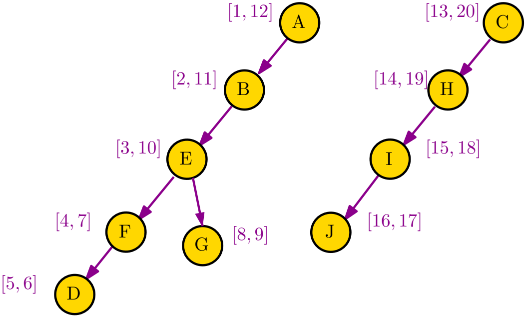
DFS Forest: Start and Finish Time

Algorithm DFS(G)

visited $\leftarrow$ ZEROS(n)
time $\leftarrow 1$
for all $v \in V$ **do**
  if $visited[v] = 0$ **then**
    DFS-EXPLORE(v)

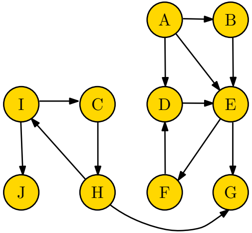
function DFS-EXPLORE(v)
  $visited[v] \leftarrow 1$
  $s[v] \leftarrow time$
  $time \leftarrow time + 1$
  for $u \in N(v)$ **do**
    if $visited[u] = 0$ **then**
      DFS-EXPLORE(u)
  $f[v] \leftarrow time$
  $time \leftarrow time + 1$

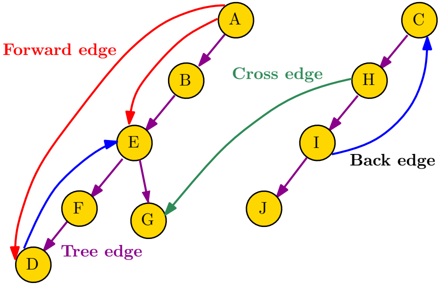




DFS Forest: Types of Edges

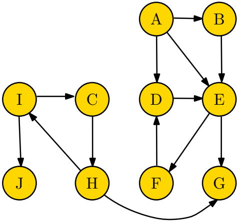
Overlay all edges of a digraph G onto its DFS forest

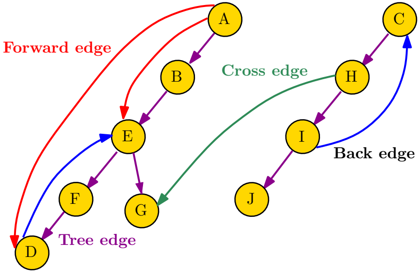


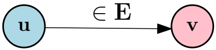


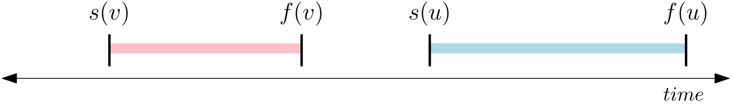
- Tree edge - Edge used in the DFS (parent to child)
- Back edge - Edge from a node to a non-parent ancestor
- Forward edge - Edge from a node to a non-child descendant
- Cross edge - Edge from a node in one tree to a node in another

DFS Forest: Identifying Edge Type

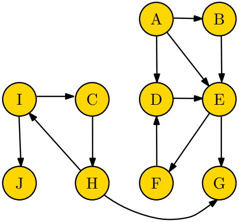


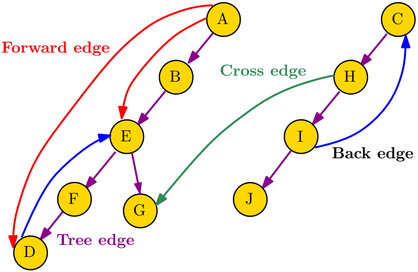


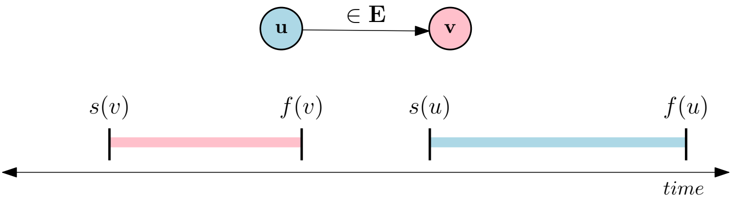




DFS Forest: Identifying Edge Type

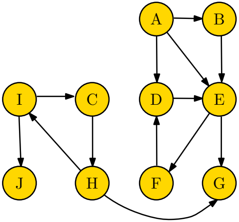


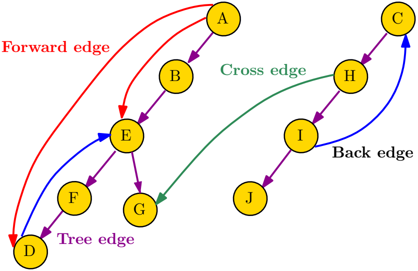


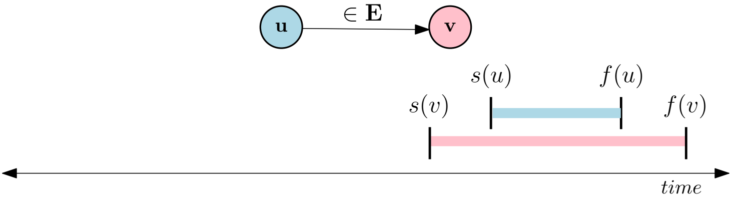


cross edge

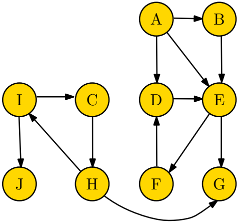
DFS Forest: Identifying Edge Type

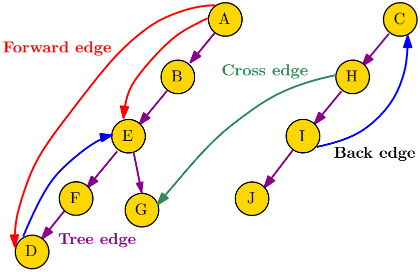


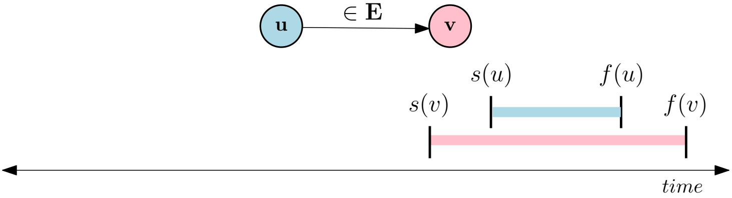




DFS Forest: Identifying Edge Type

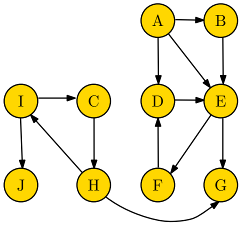


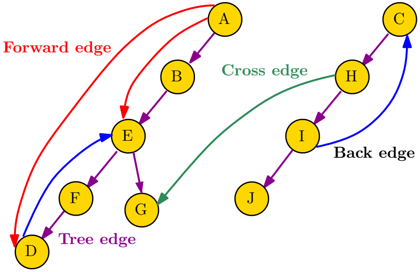


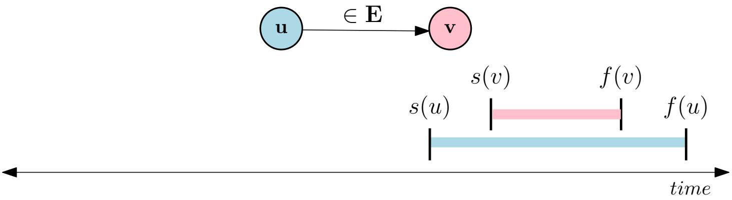


back edge

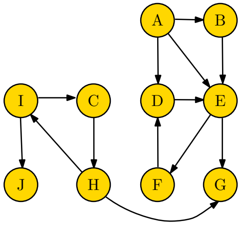
DFS Forest: Identifying Edge Type

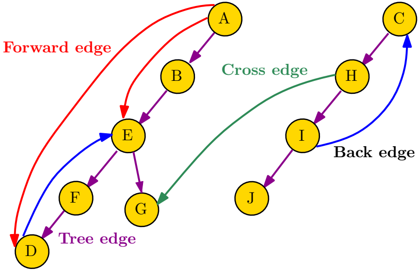


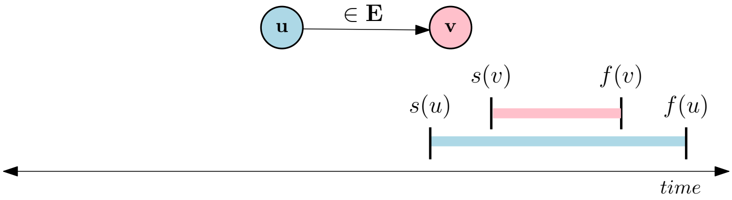




DFS Forest: Identifying Edge Type



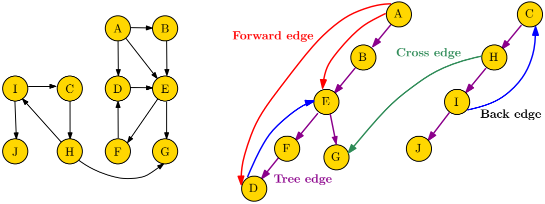


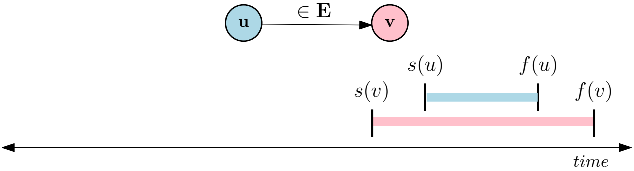


tree or forward edge (check predecessor)

DFS Forest: Cycles in Graphs

Lemma: A digraph G has a directed cycle if and only if the DFS forest of G has a back edge





back edge