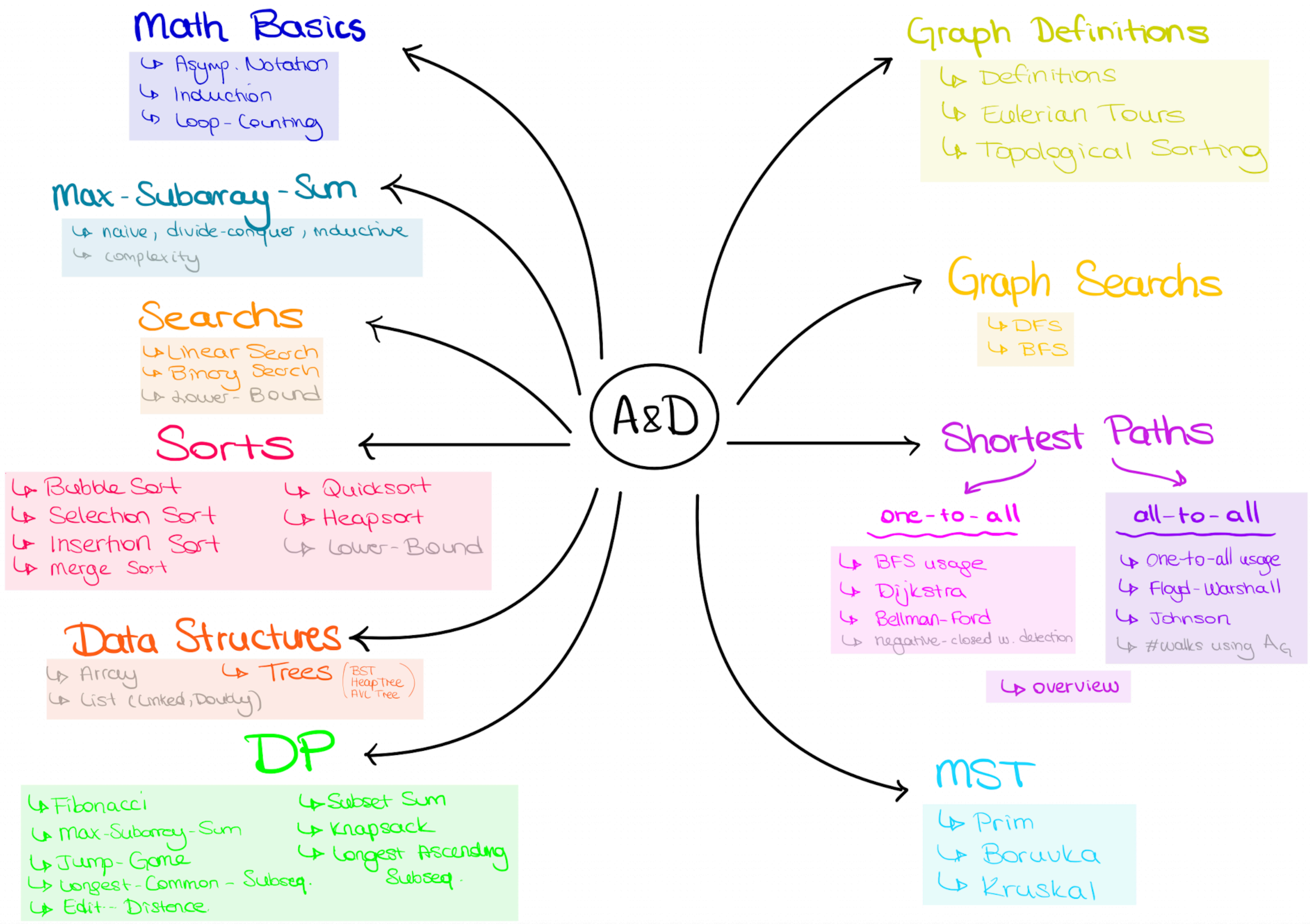
A&D

Exercise Session 9

Nil Ozer

A&D Overview


A&D
Math Basics
Asymp. Notation
Induction
Loop-Counting
Max-Subarray-Sum
naive, divide-conquer, inductive
complexity
Searchs
Linear Search
Binary Search
Lower-Bound
Sorts
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quicksort
Heapsort
Lower-Bound
Data Structures
Array
List (Linked, Doubly)
Trees
BST
Heap Tree
AVL Tree
DP
Fibonacci
Max-Subarray-Sum
Jump-Game
Longest-Common-Subseq.
Edit-Distance
Subset Sum
Knapsack
Longest Ascending Subseq.
Graph Definitions
Definitions
Eulerian Tours
Topological Sorting
Graph Searchs
DFS
BFS
Shortest Paths
one-to-all
BFS usage
Dijkstra
Bellman-Ford
negative-closed w. detection
all-to-all
One-to-all usage
Floyd-Warshall
Johnson
#walks using A_G
overview
MST
Prim
Boruvka
Kruskal

Outline

- Quiz
- Exercise Sheets
- Questions from you

- Graph Definitions - Exam Question

- Graph Searchs - DFS
- Topological Sorting

- DP Mini Exam - Proof at the end

Quiz

Exercise Sheets

Exercise Sheet 6

Bonus Feedback

- 6.1 :
  - Watch out for the comments !
  - Tree proof structure improved :) 👏🏻

- 6.3 :
  - Indexes ☠️
  - DP structure !!

- 6.4 : 👏🏻

Peergrading

- Exercise Sheet 8 peergrading
  - 8.1 this week
  - Emails will be sent

Questions from you

- **Asymptotic Bound Consistency in Algorithm Runtime Questions**
  - We specify explicitly what you need to prove
    - Often we say that you should create an algorithm with runtime at most O(x) and then you need to justify why your algorithm is in O(x)
  - If we don't specify, head-ta says that you should give a Theta bound or at least an O bound that is as tight as possible.

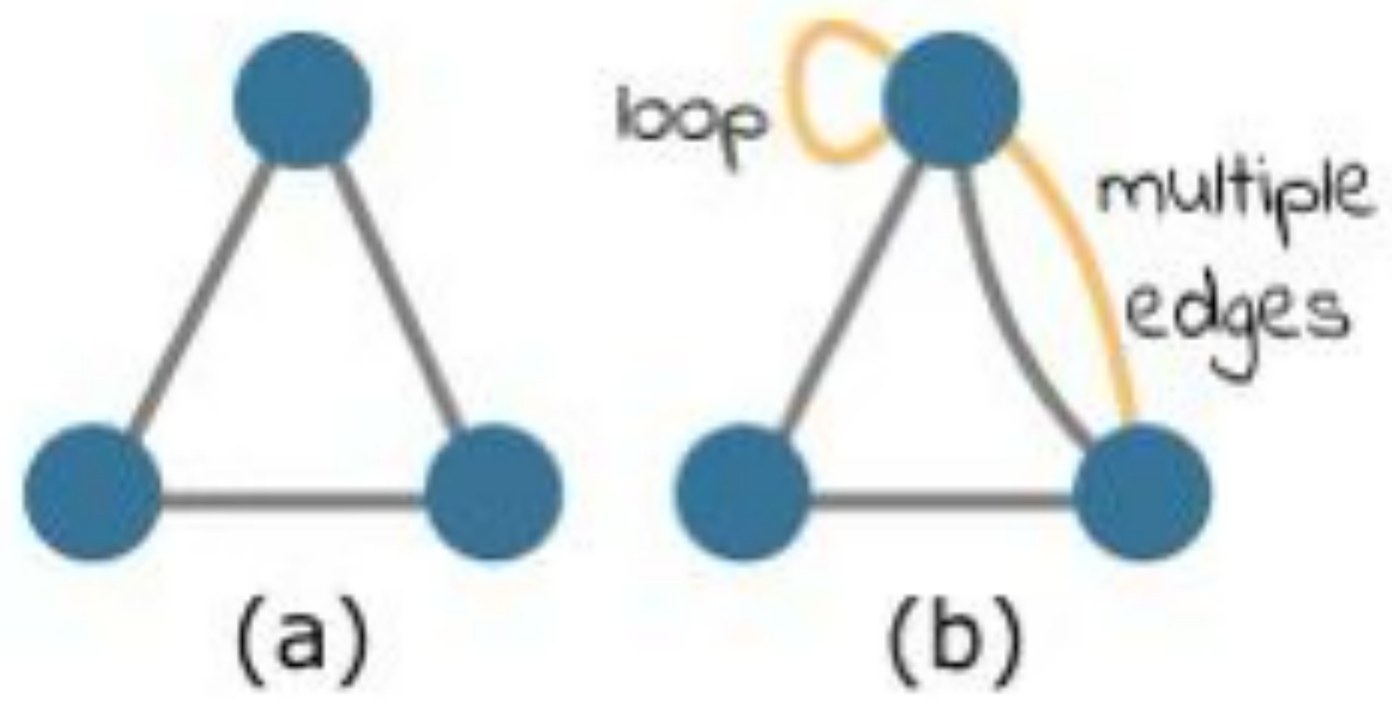
- **Is a tree a directed or undirected graph ?**
  - “tree”
  - “directed tree”

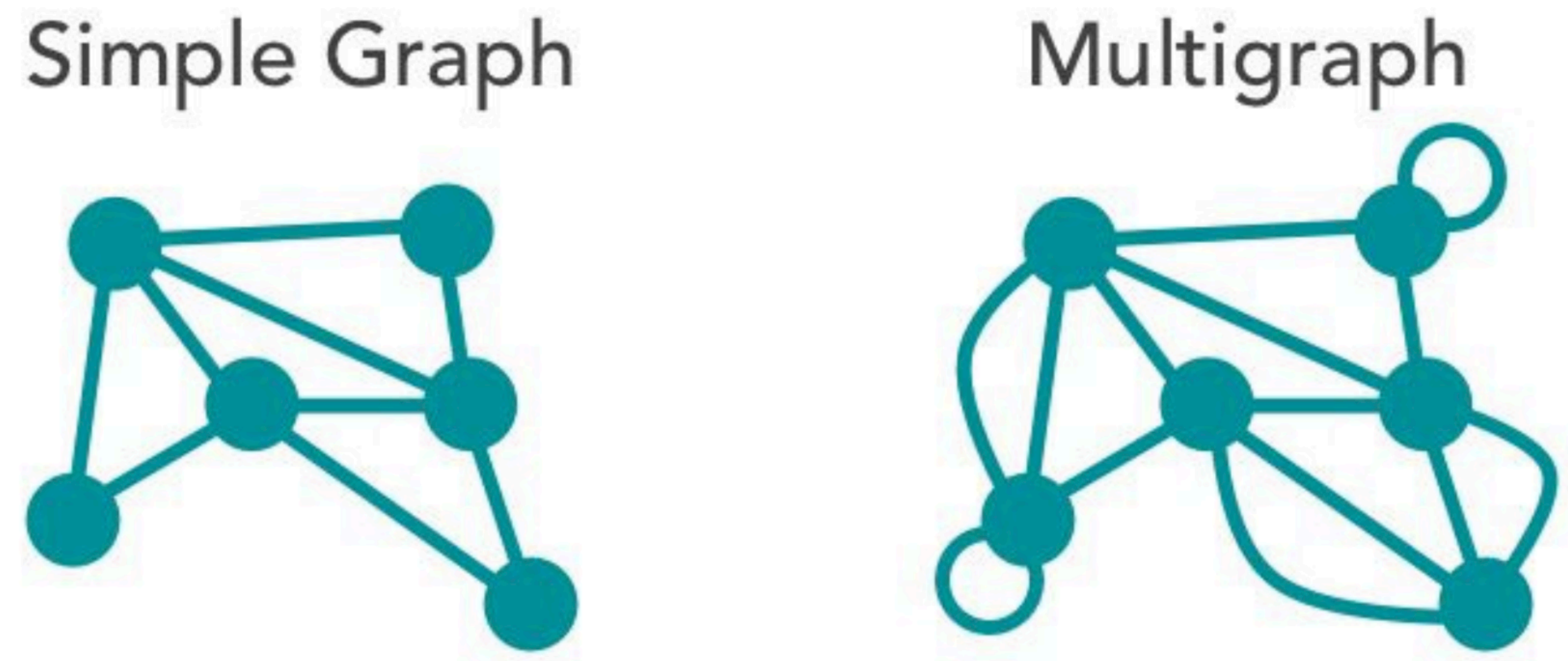
Questions from you

Valid Counterexamples for A&D !!

- For A&D we don't consider multigraphs
  - unless explicitly specified otherwise.

- Simple Graph : no self loops or multiple edges between same vertices
- Multigraphs : self loops or multiple edges are allowed





Graph Definitions

Graph
Definitions

Definition 1. Let $G = (V, E)$ be a graph.

- For $v \in V$, the **degree** $\deg(v)$ of v (german "Knotengrad") is the number of edges that are incident to v.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ (with $v_i \in V$ for all i) is a **walk** (german "Weg") if $\{v_i, v_{i+1}\}$ is an edge for each $0 \leq i \leq k-1$. We say that v_0 and v_k are the **endpoints** (german "Startknoten" and "Endknoten") of the walk. The **length** of the walk $(v_0, v_1, \ldots, v_k)$ is k.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **closed walk** (german "Zyklus") if it is a walk, $k \geq 2$ and $v_0 = v_k$.
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **path** (german "Pfad") if it is a walk and all vertices are distinct (i.e., $v_i \neq v_j$ for $0 \leq i < j \leq k$).
- A sequence of vertices $(v_0, v_1, \ldots, v_k)$ is a **cycle** (german "Kreis") if it is a closed walk, $k \geq 3$ and all vertices (except v_0 and v_k) are distinct.
- A **Eulerian walk** (german "Eulerweg") is a walk that contains every edge exactly once.
- A **closed Eulerian walk** (german "Eulerzyklus") is a closed walk that contains every edge exactly once.
- A **Hamiltonian path** (german "Hamiltonpfad") is a path that contains every vertex.
- A **Hamiltonian cycle** (german "Hamiltonkreis") is a cycle that contains every vertex.
- For $u, v \in V$, we say u **reaches** v (or v is **reachable** from u; german "u erreicht v") if there exists a walk with endpoints u and v.
- A **connected component** of G is an equivalence class of the (equivalence) relation defined as follows: Two vertices $u, v \in V$ are equivalent if u reaches v.
- A graph G is **connected** (german "zusammenhängend") if for every two vertices $u, v \in V$ u reaches v or equivalently if there is only one connected component.
- A graph G is a **tree** (german "Baum") if it is connected and has no cycles.

Graph

Exam Question

/ 5 P

c) *Graph quiz:* For each of the following claims, state whether it is true or false. You get 1P for a correct answer, -1P for a wrong answer, 0P for a missing answer. You get at least 0 points in total.

As a reminder, here are a few definitions for a (directed) graph $G = (V, E)$:

For $k \geq 2$, a (directed) *walk* is a sequence of vertices $v_1, \ldots, v_k$ such that for every two consecutive vertices v_i, v_{i+1}, we have $\{v_i, v_{i+1}\} \in E$ (resp. $(v_i, v_{i+1}) \in E$ for a directed walk).

A (directed) *closed walk* is a (directed) walk with $v_1 = v_k$.

A (directed) *cycle* is a (directed) closed walk where $k \geq 3$ and all vertices (except v_1 and v_k) are distinct.

A (directed) *closed Eulerian walk* is a (directed) closed walk which traverses every edge in E exactly once.

For a vertex v in a directed graph $G = (V, E)$, the *in-degree* of v is the number of edges in E that end in v (i.e., of the form (w, v)), and the *out-degree* of v is the number of edges in E that start in v (i.e., of the form (v, w)).

Claim	**true**	**false**
A connected graph must contain a cycle.	☐	☐
A graph $G = (V, E)$ with $\|E\| \leq \|V\| - 1$ is a tree.	☐	☐
Let $G = (V, E)$ be a graph with $\|E\| \geq 4$, which contains a closed Eulerian walk. If we remove one edge from E, the resulting graph does not contain a closed Eulerian walk, no matter which edge we remove (the vertex set does not change).	☐	☐
Let $G = (V, E)$ be a *directed* graph. If the in-degree and out-degree of every vertex $v \in V$ is even, then G contains a *directed* closed Eulerian walk.	☐	☐
Let $G = (V, E)$ be an undirected graph. Then there is a way to direct the edges of G such that the resulting directed graph does not contain a directed cycle.	☐	☐

Graph Definitions

Exam Tipps

- T/F or a proof !
- Know all of the definitions
  - Don’t mix up similar ones, realise connections!
    - walk , closed walk , eulerian
    - path , cycle, hamiltonian
- Gain an intuition
  - Don’t rush ! Don’t gamble !!!
  - Do this by actually coming up with short proofs !
- Practice, practice, practice!

Graph Searches

DFS

Graph Searchs

DFS - with pre and post order

Algorithm 4 DFS(G)

$T \leftarrow 1$
alle Knoten unmarkiert
for $u_0 \in V$, unmarkiert **do**
    Visit(u_0)

Graph Searchs

DFS - with pre and post order

Runtime : O (|V| + |E|)

Algorithm 4 DFS(G)

```
T ← 1
alle Knoten unmarkiert
for u₀ ∈ V, unmarkiert do
    Visit(u₀)
```

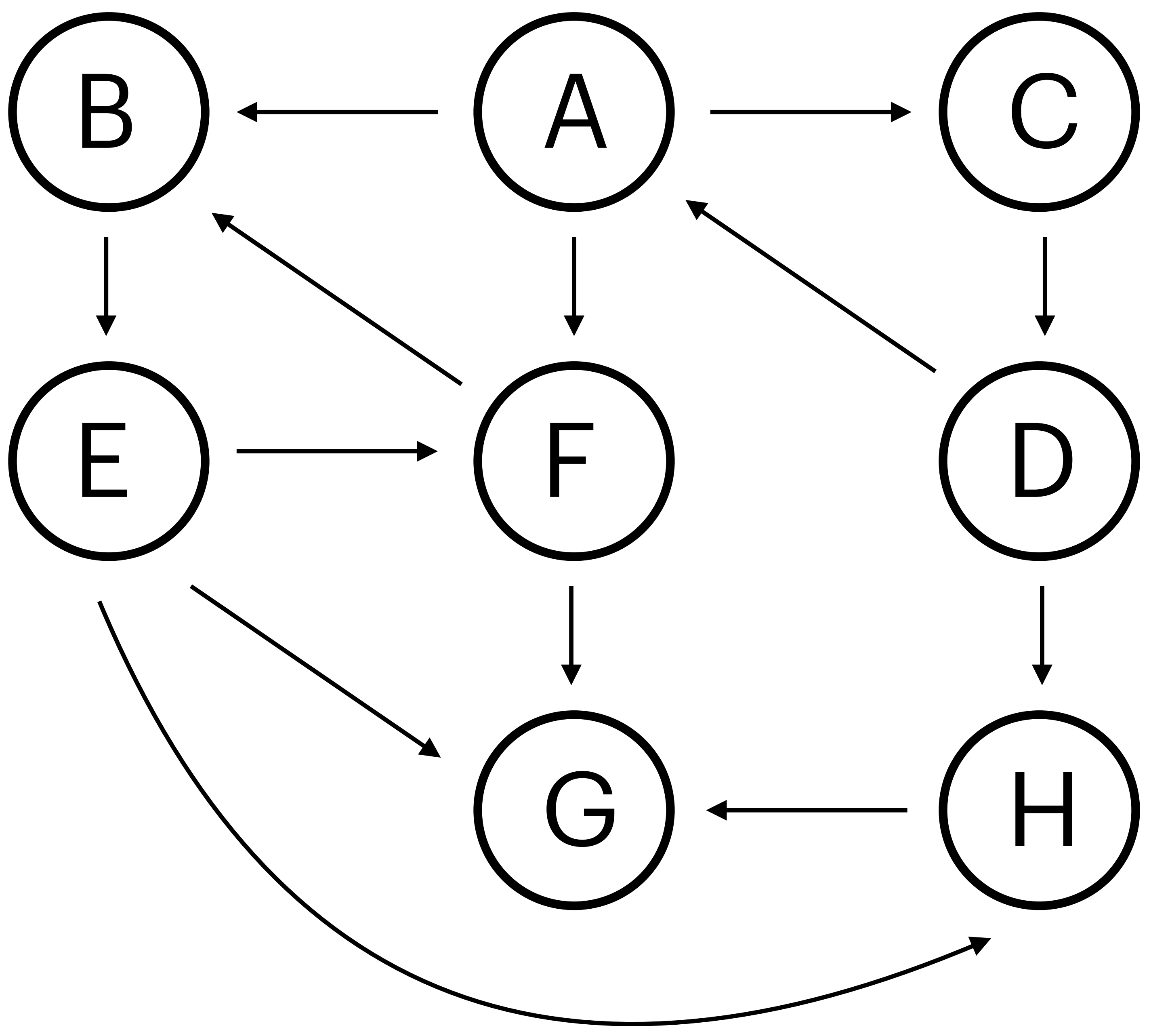
Algorithm 3 Visit(u)

```
pre[u] ← T; T ← T + 1
markiere u
for Nachvolger v von u, unmarkiert do
    Visit(v)
post[u] ← T; T ← T + 1
```

Let’s take a break

Graph Searchs

DFS - Example



pre-order

post-order

tree edge

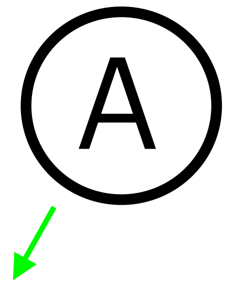
back edge

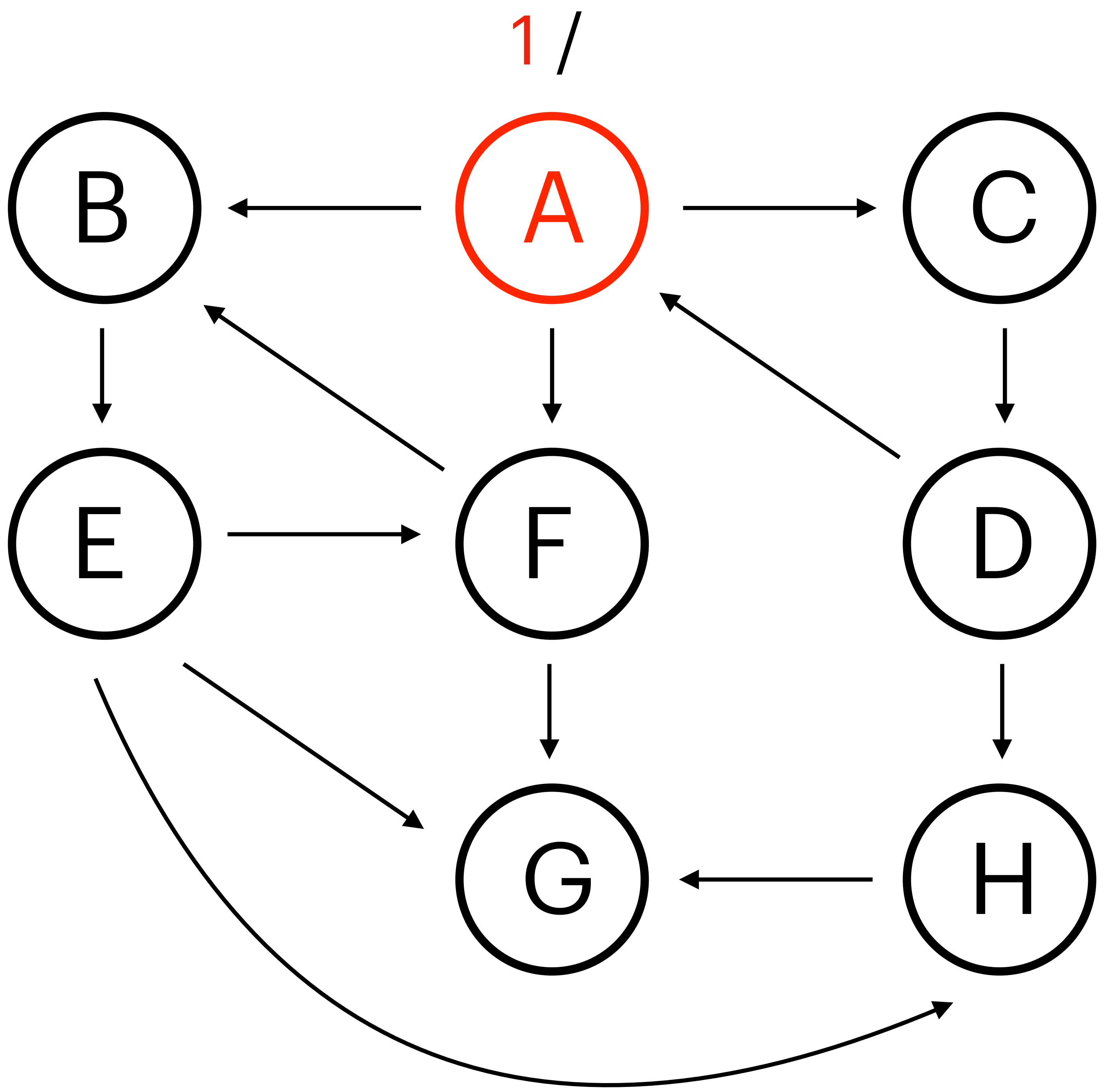
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order

post-order

tree edge

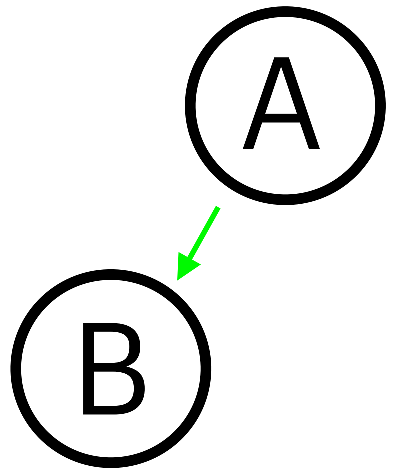
back edge

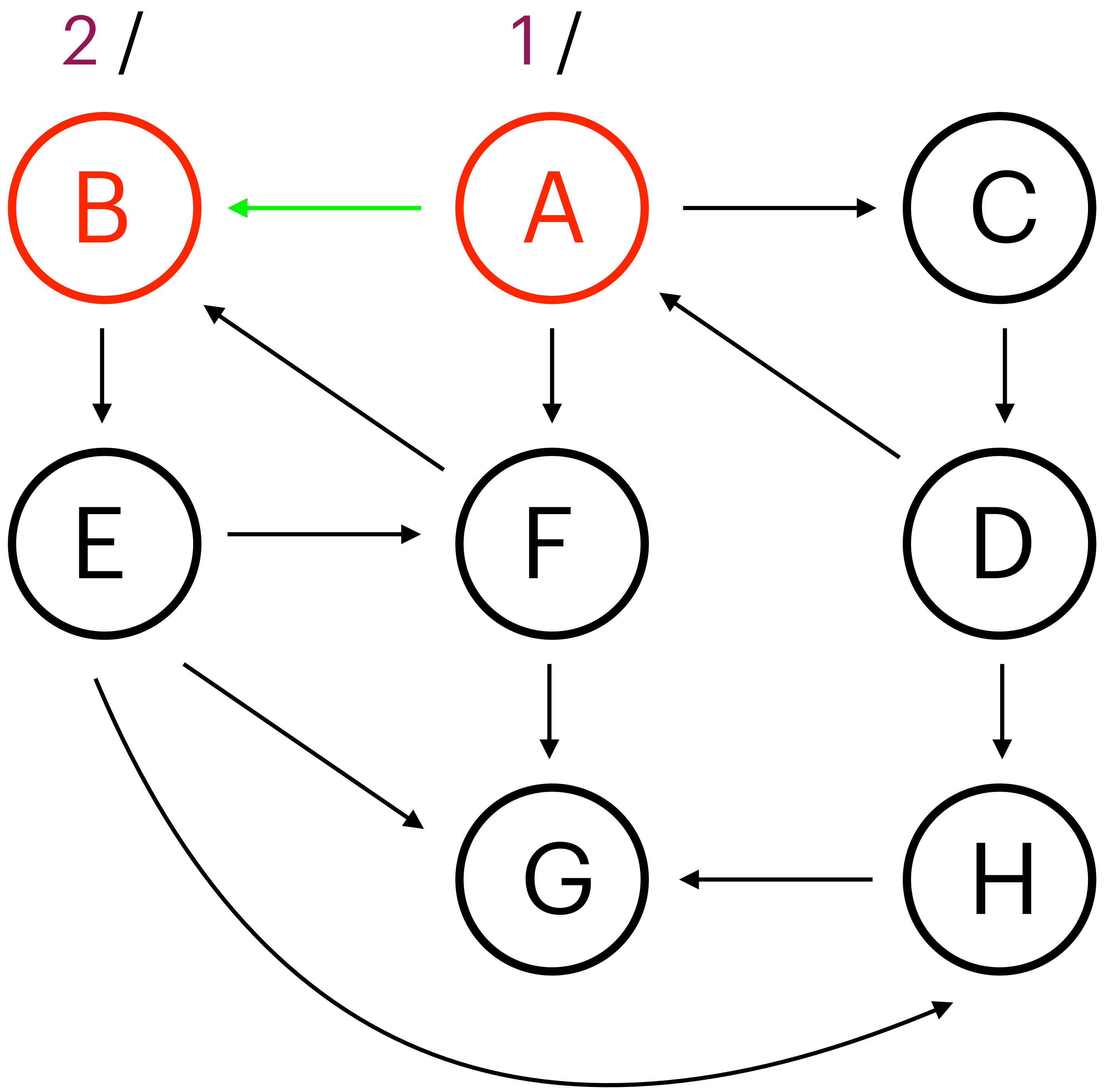
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

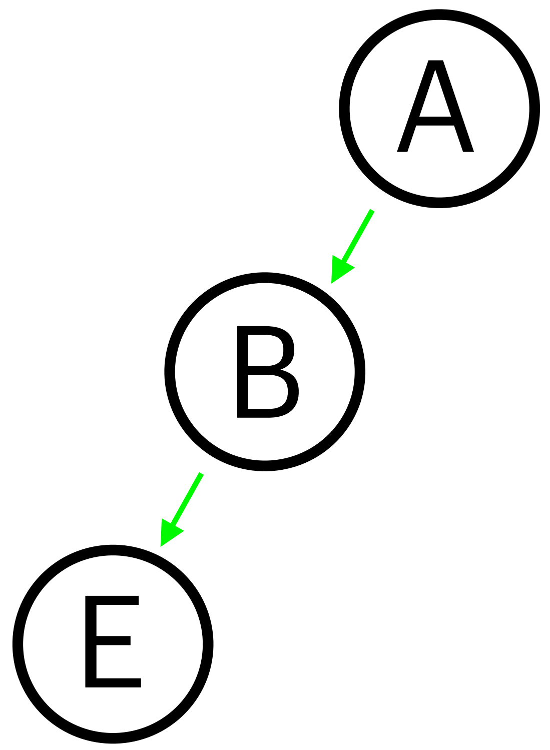
back edge

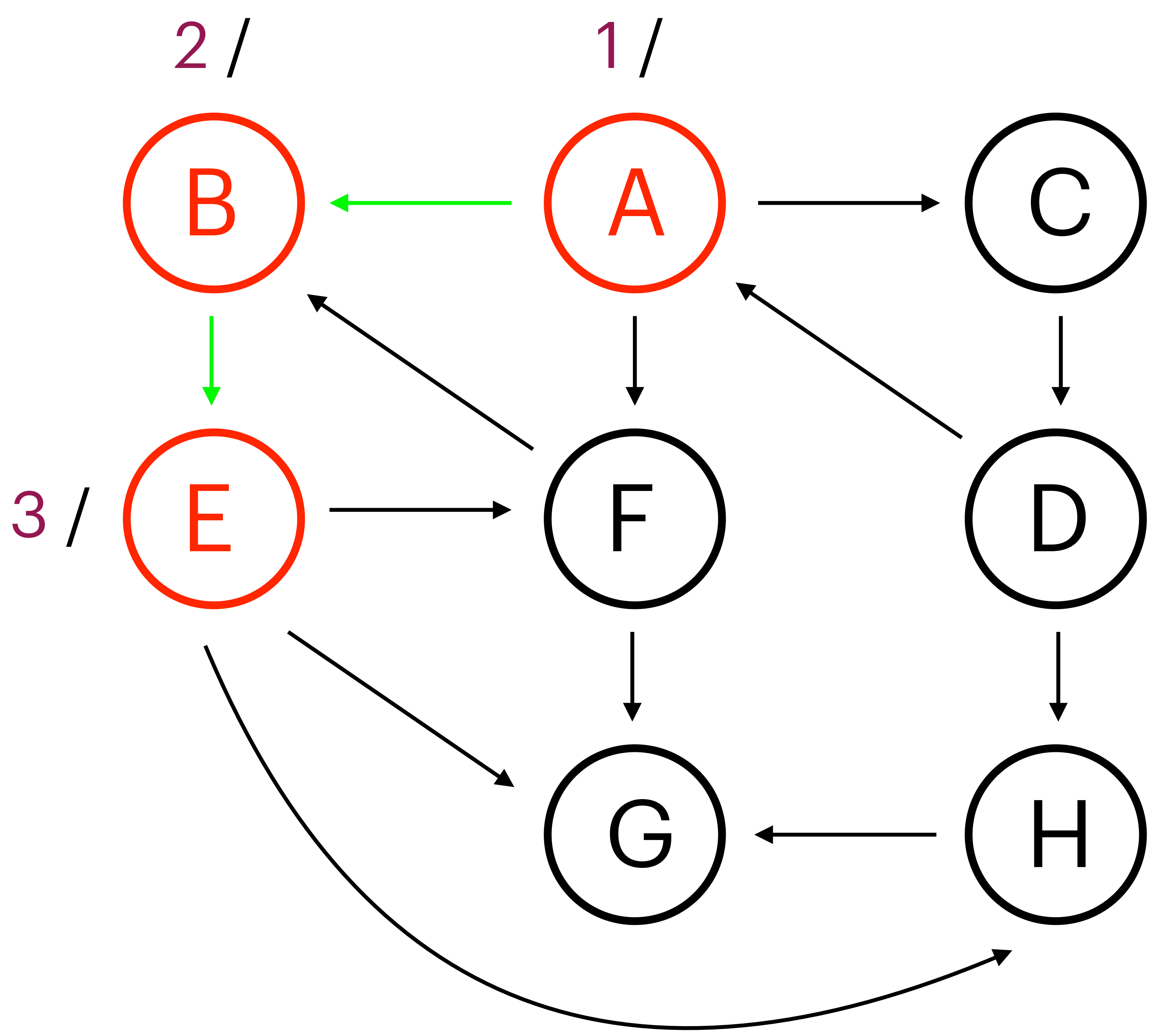
forward edge

cross edge

Graph Searches

DFS - Example





pre-order

post-order

tree edge

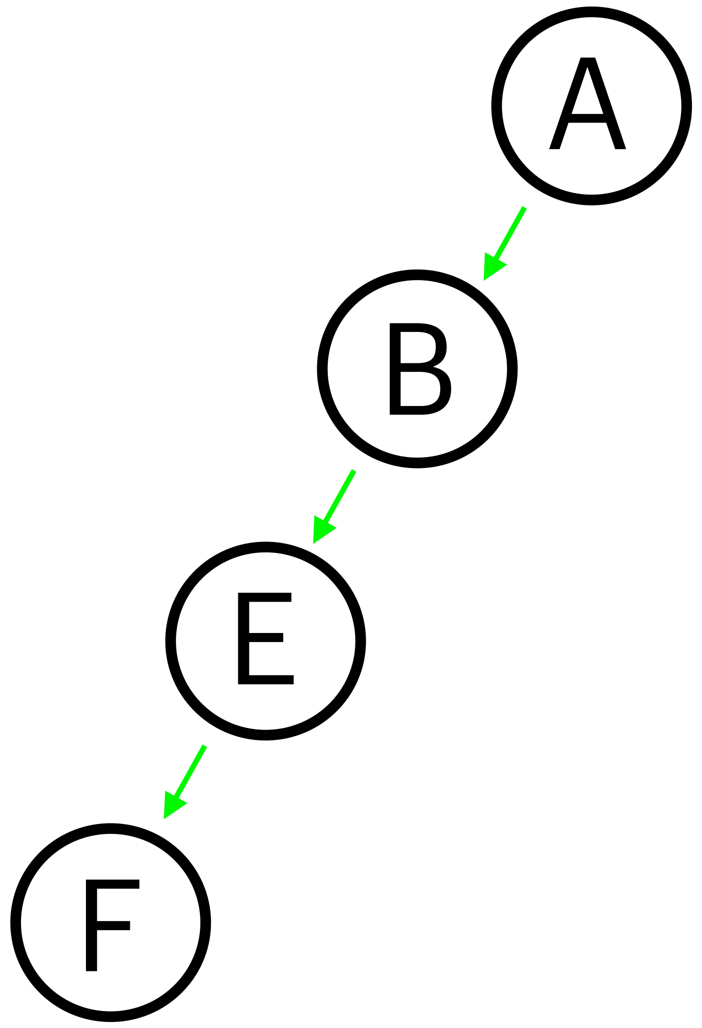
back edge

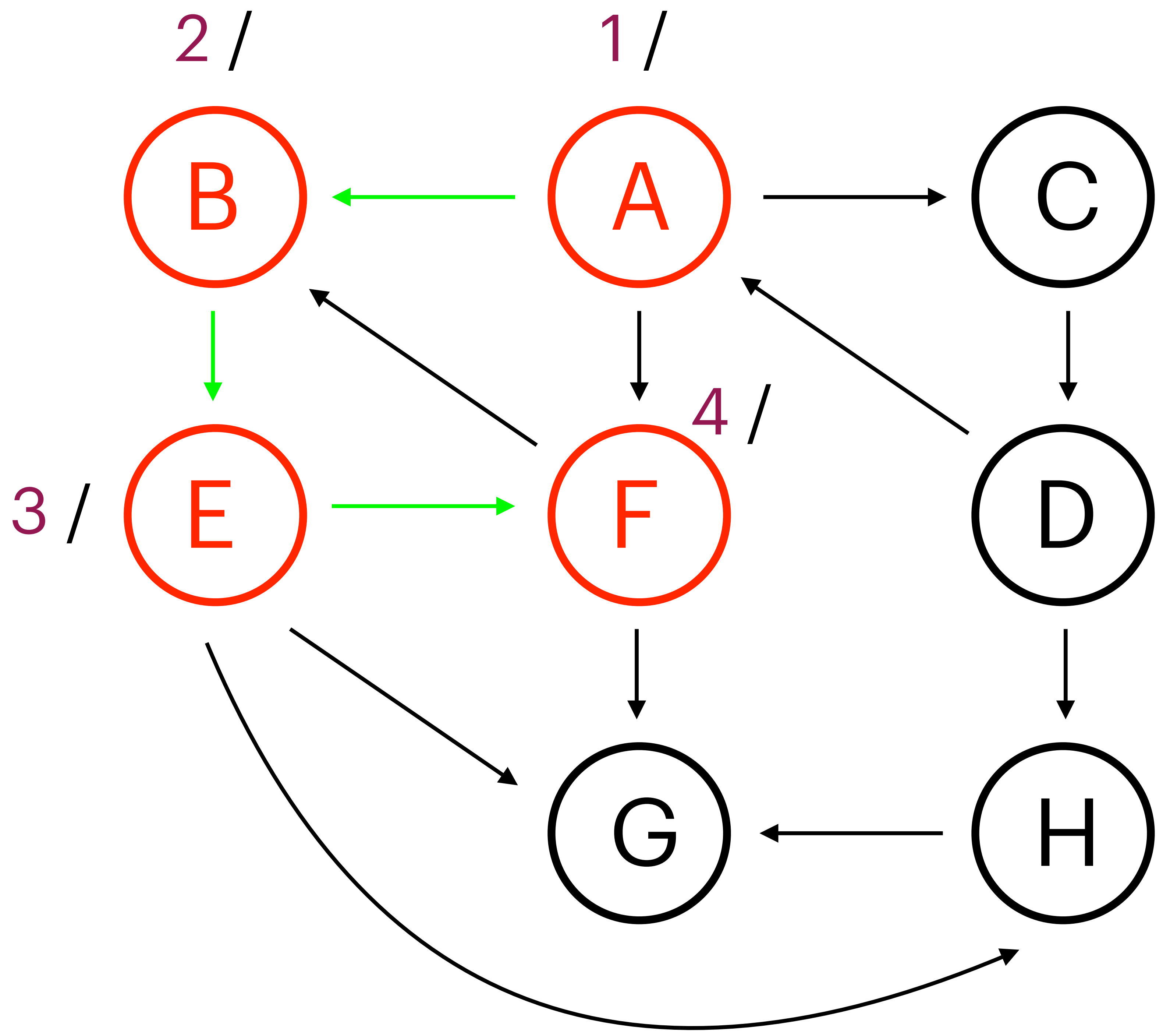
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order

post-order

tree edge

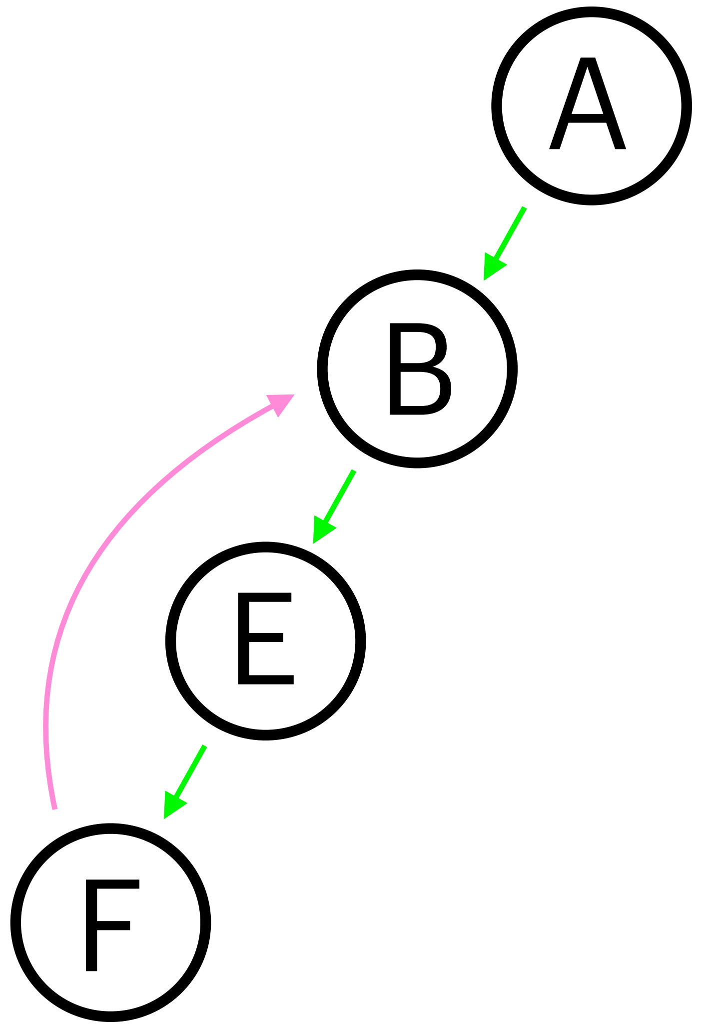
back edge

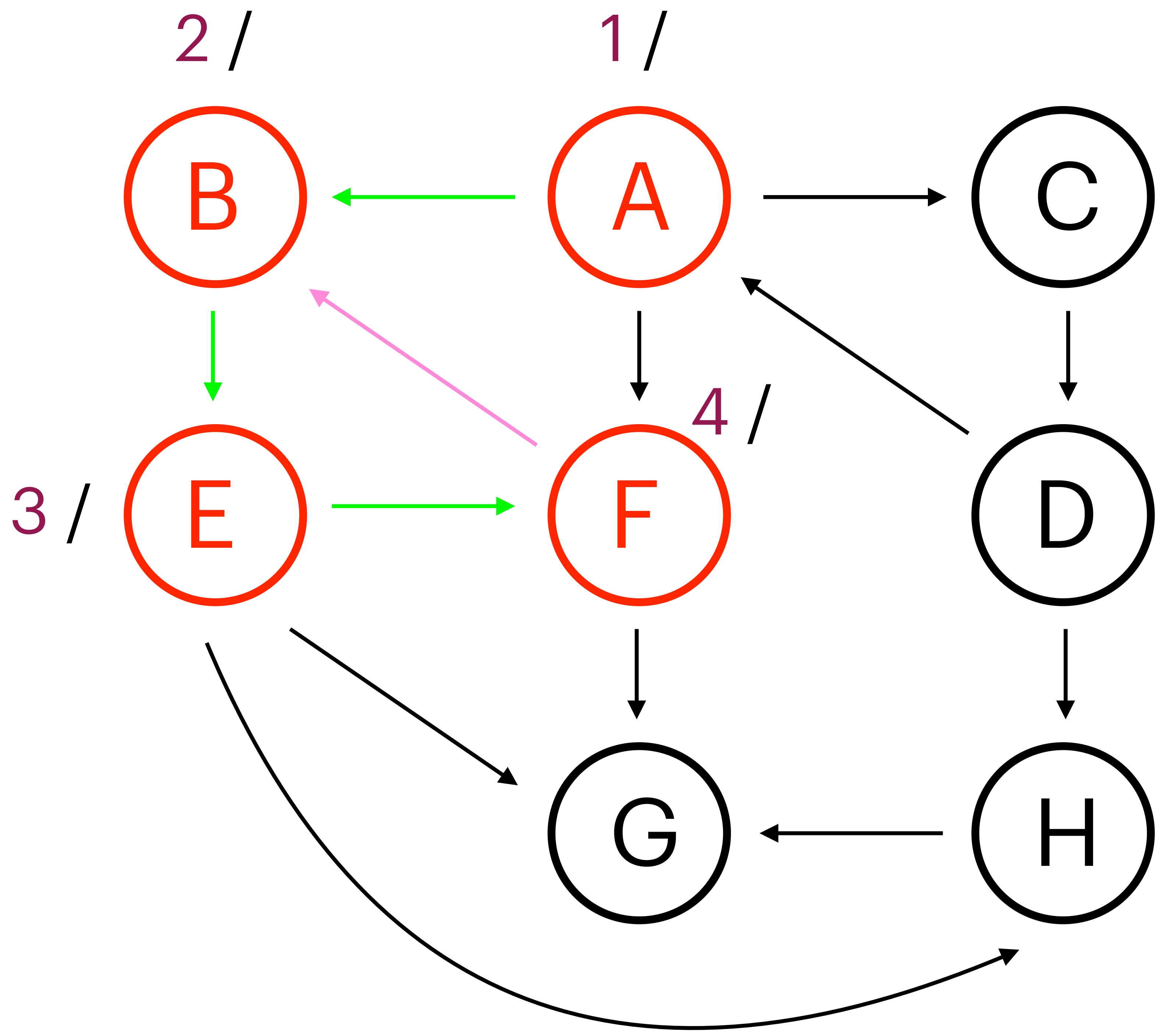
forward edge

cross edge

Graph Searches

DFS - Example





pre-order

post-order

tree edge

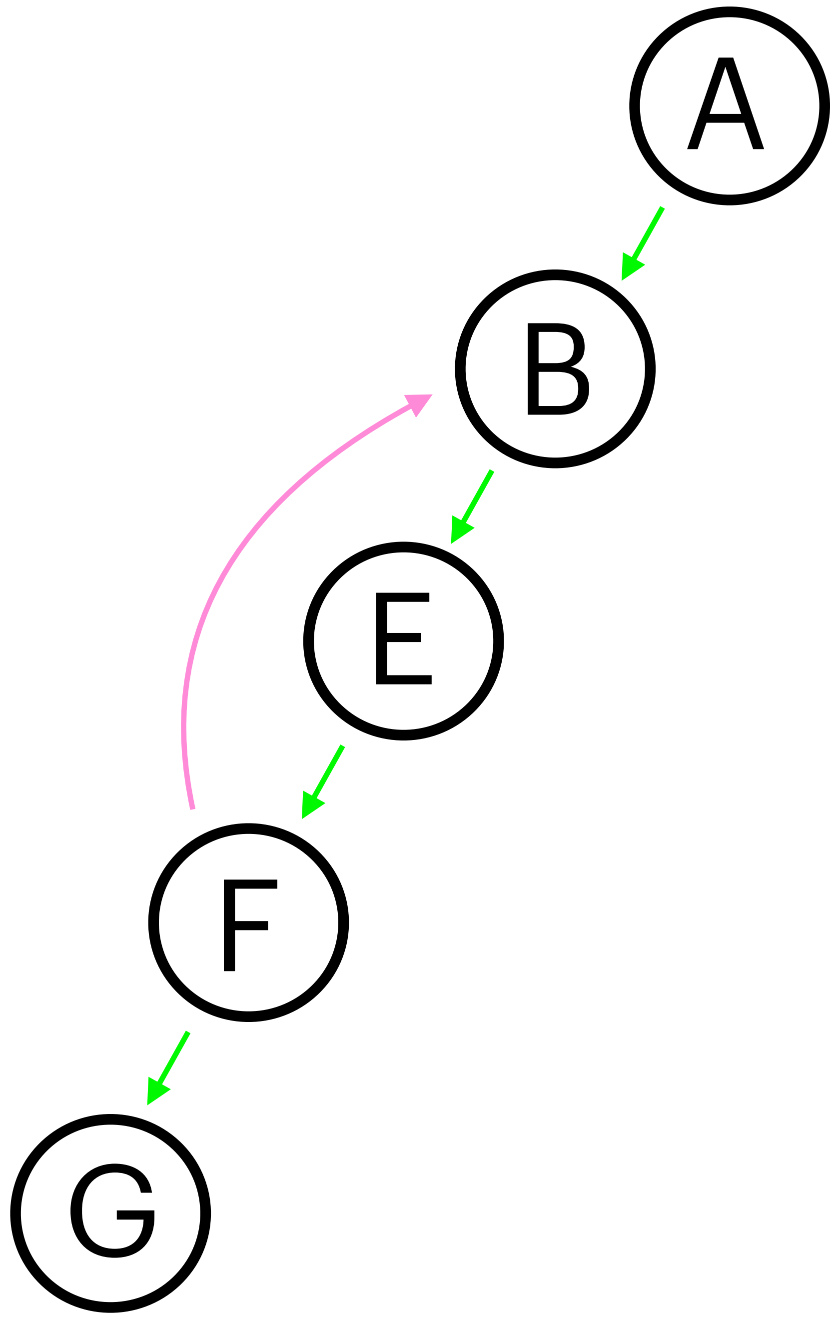
back edge

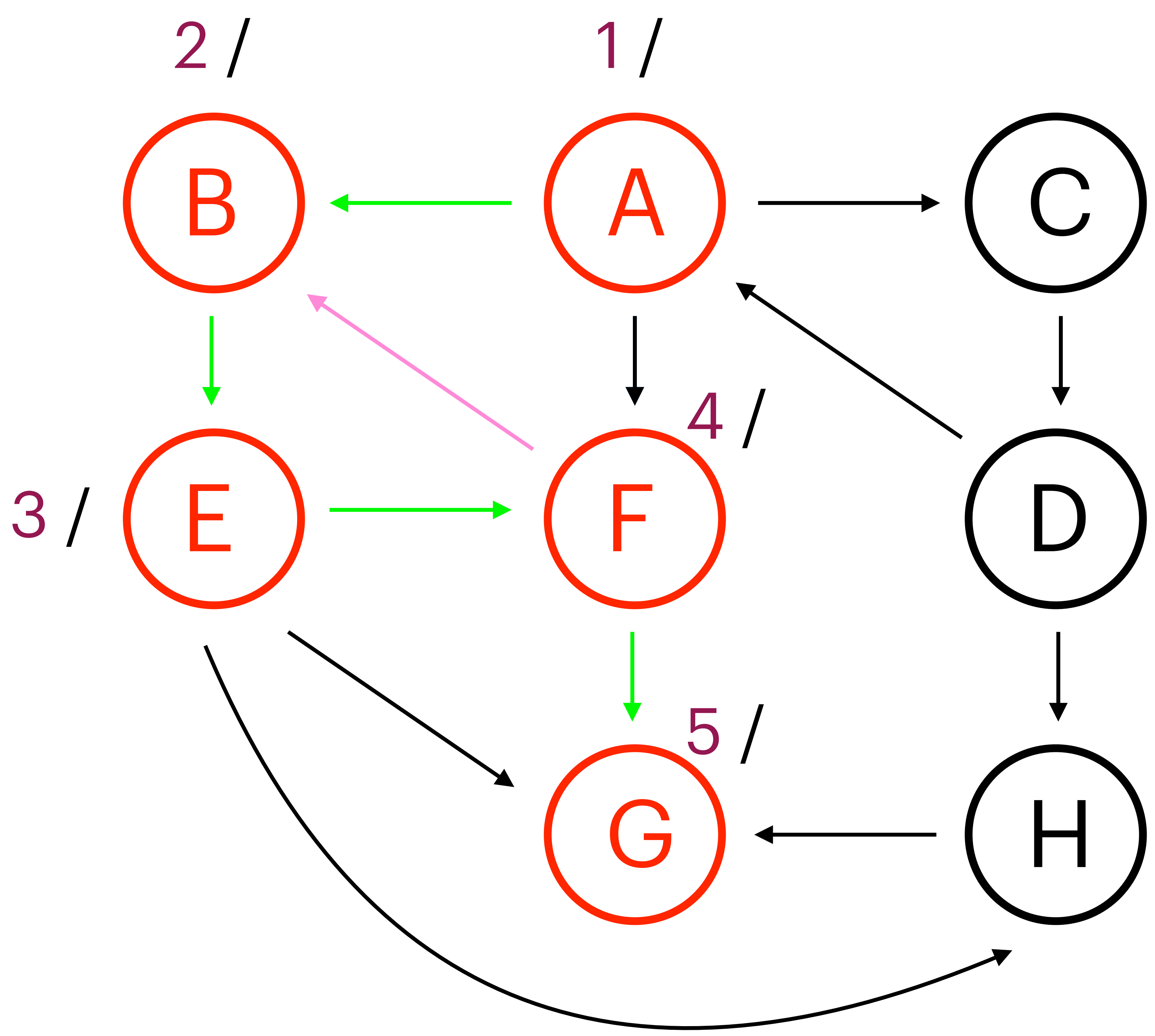
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order

post-order

tree edge

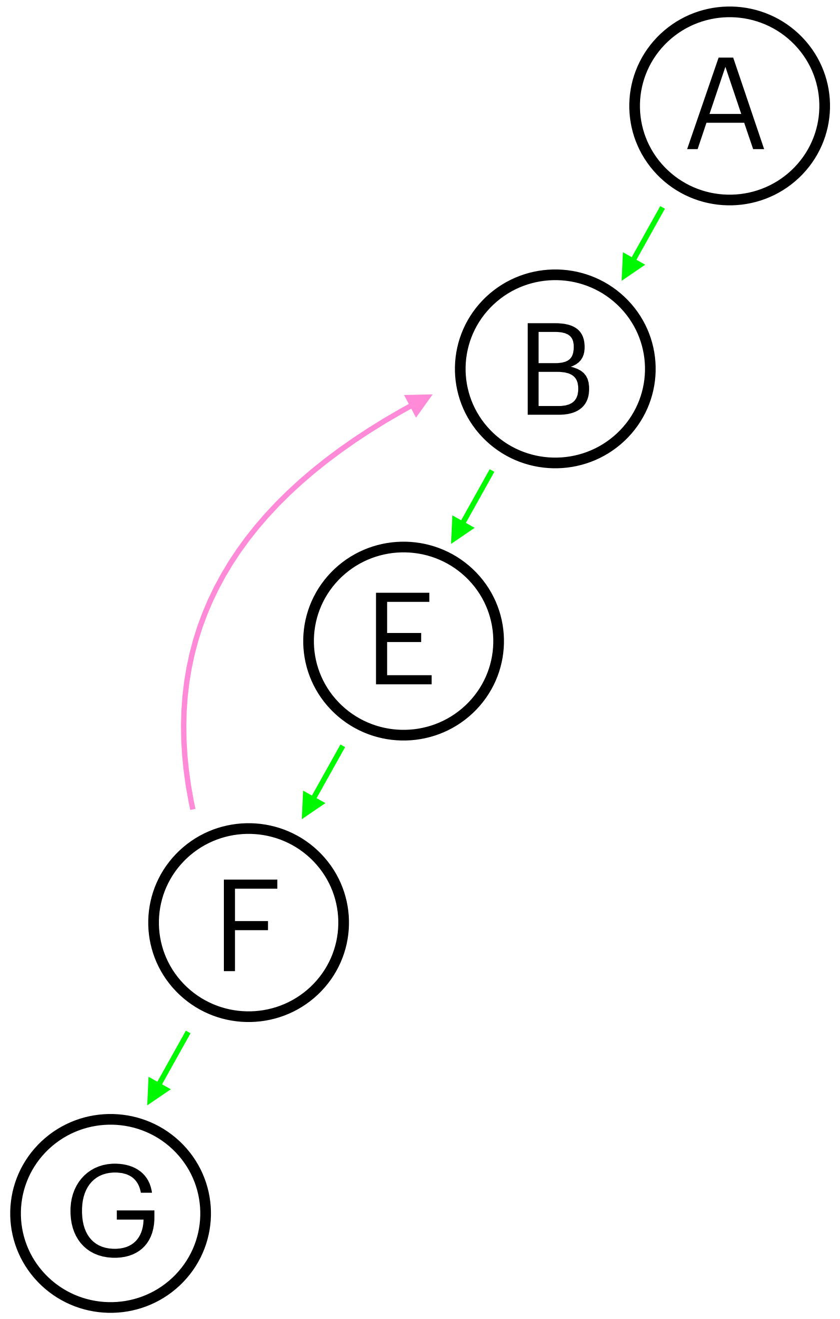
back edge

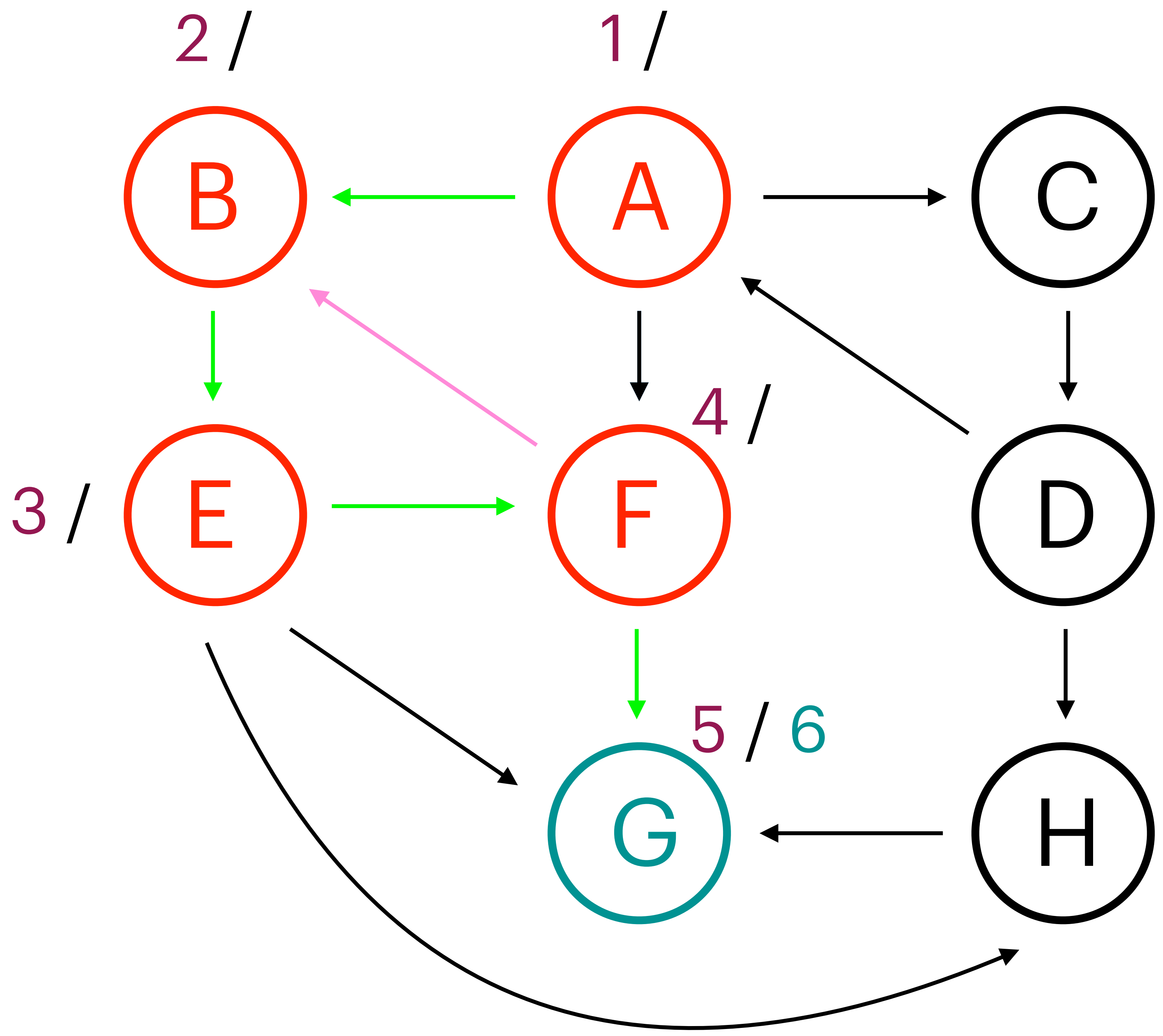
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order

post-order

tree edge

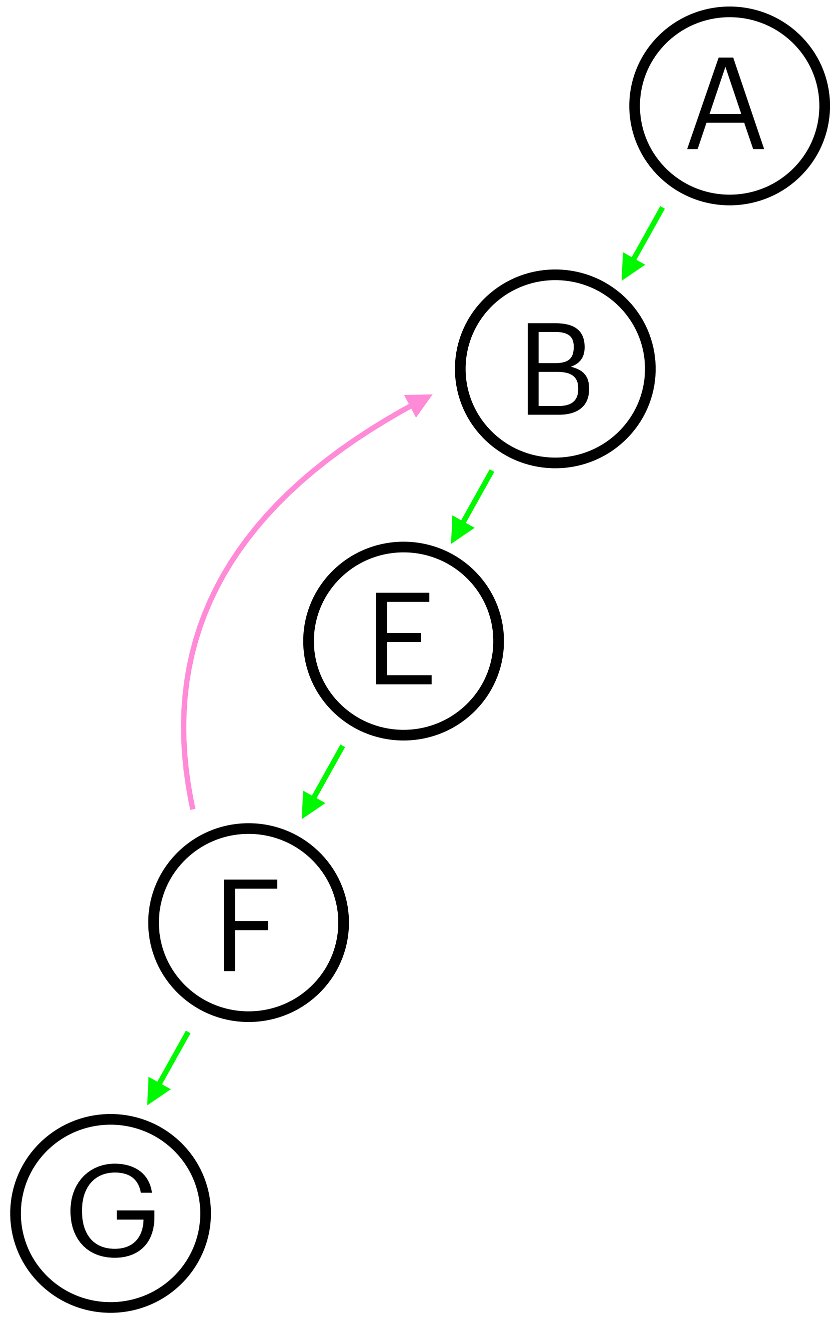
back edge

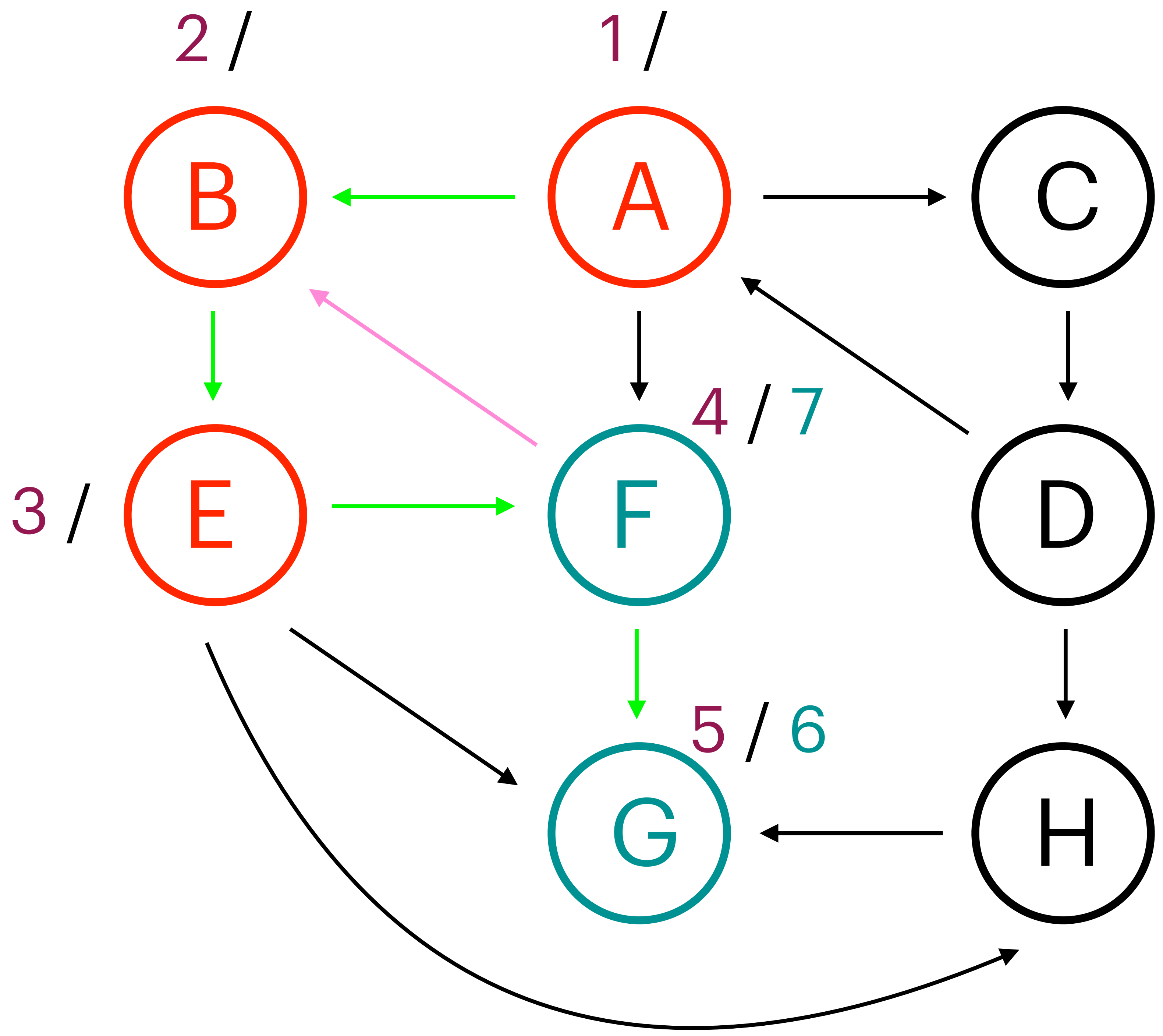
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order

post-order

tree edge

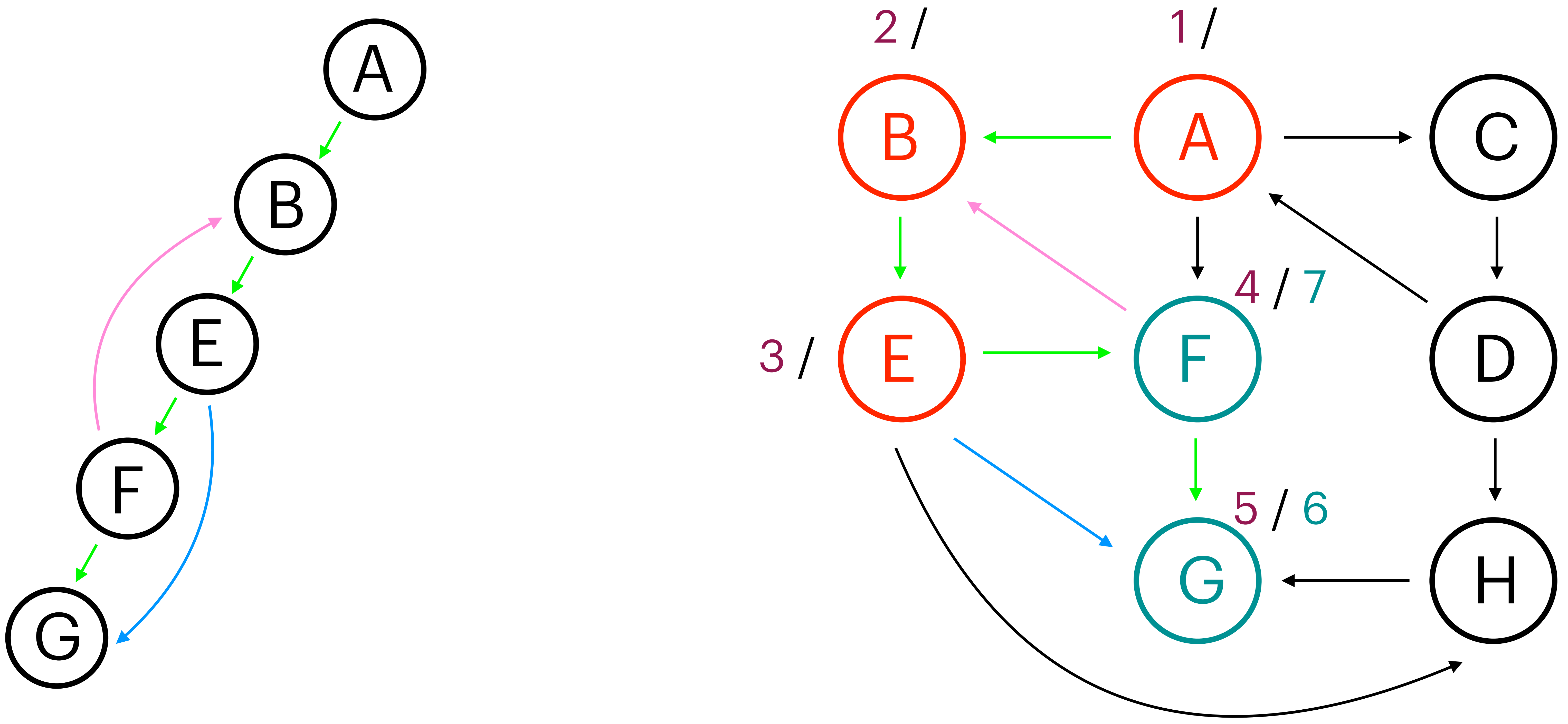
back edge

forward edge

cross edge

Graph Searchs

DFS - Example



pre-order

post-order

tree edge

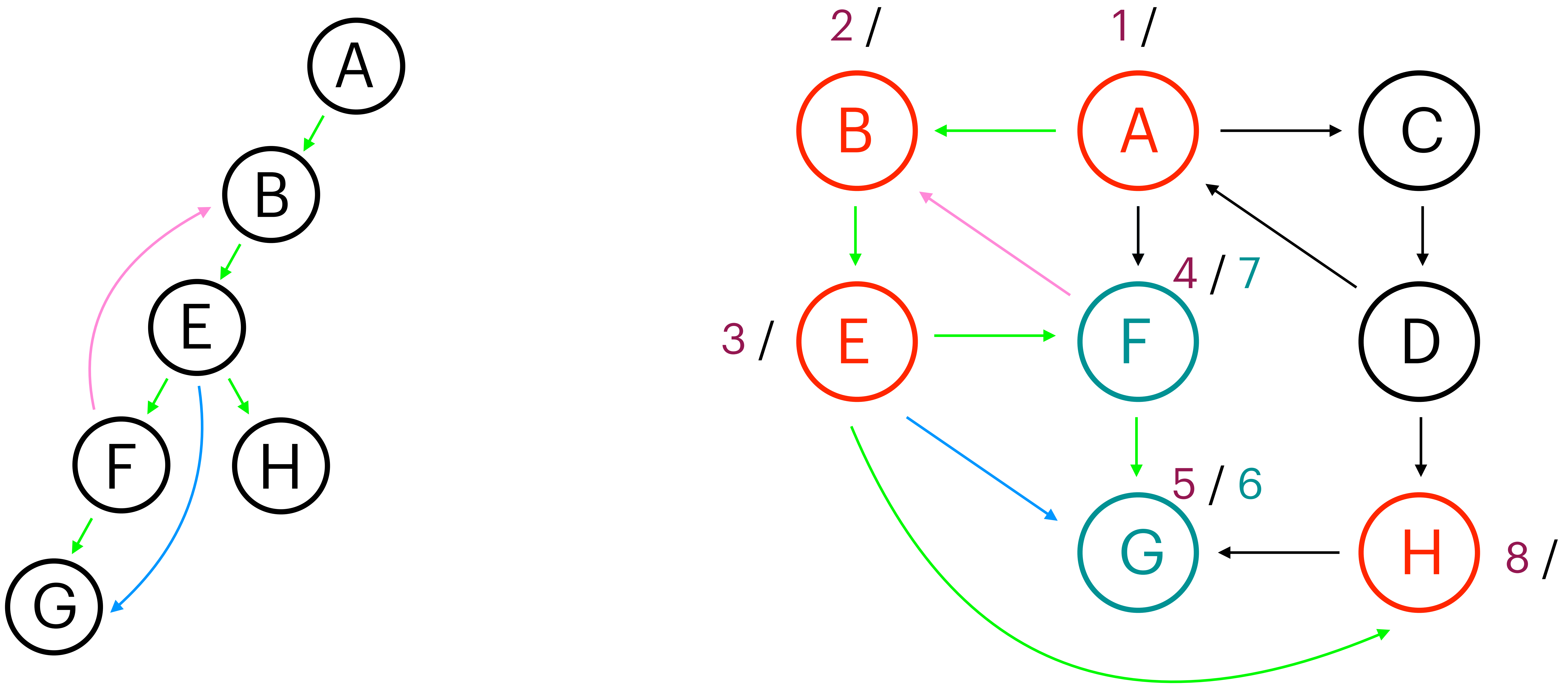
back edge

forward edge

cross edge

Graph Searchs

DFS - Example



pre-order

post-order

tree edge

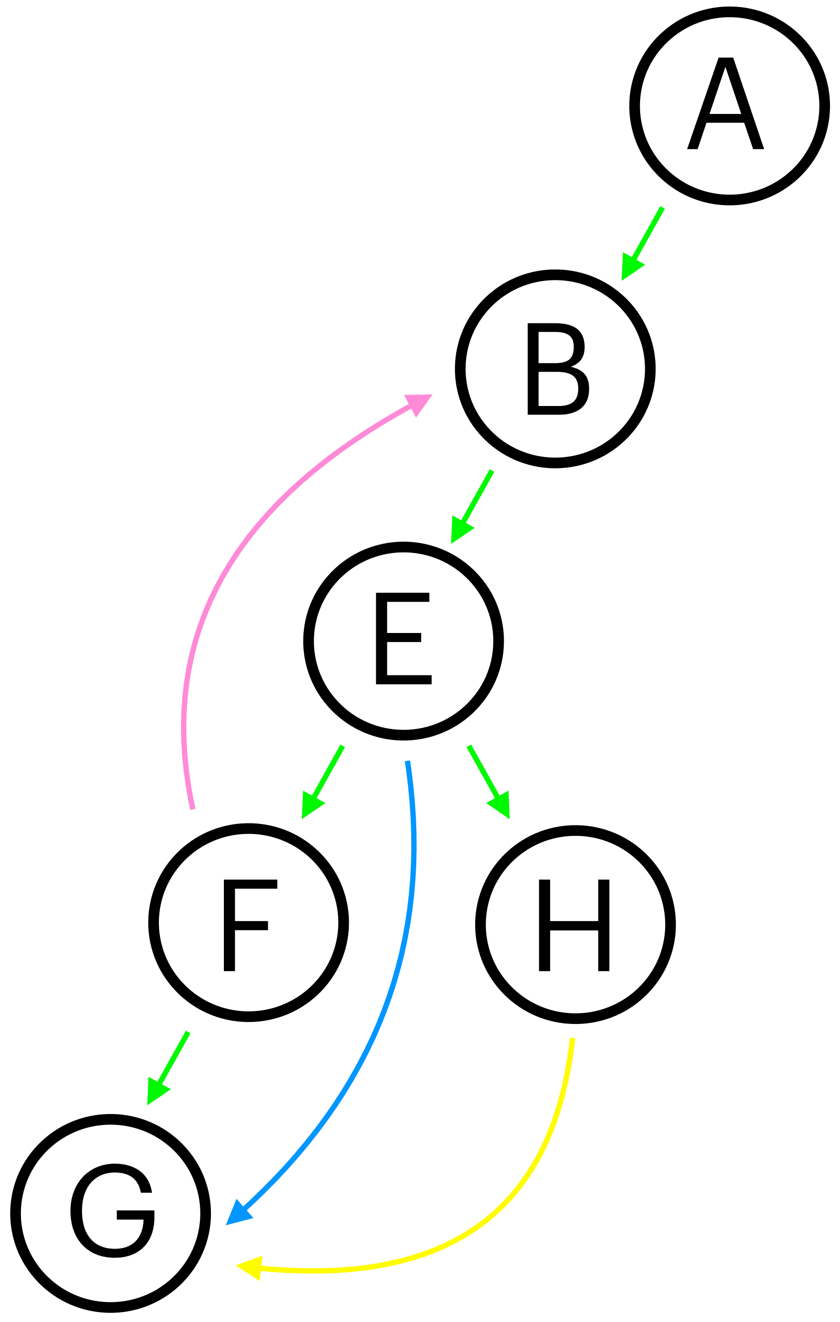
back edge

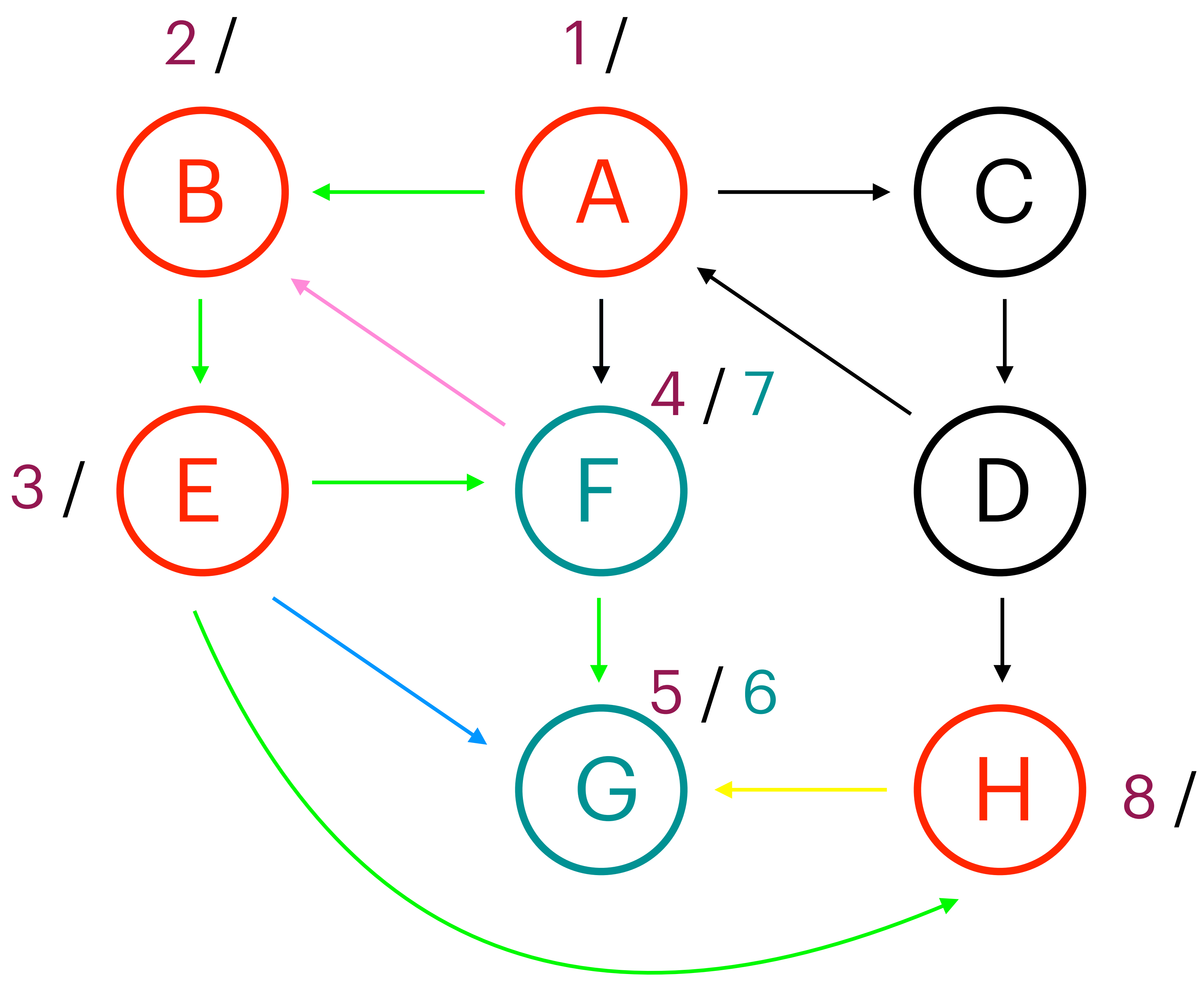
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

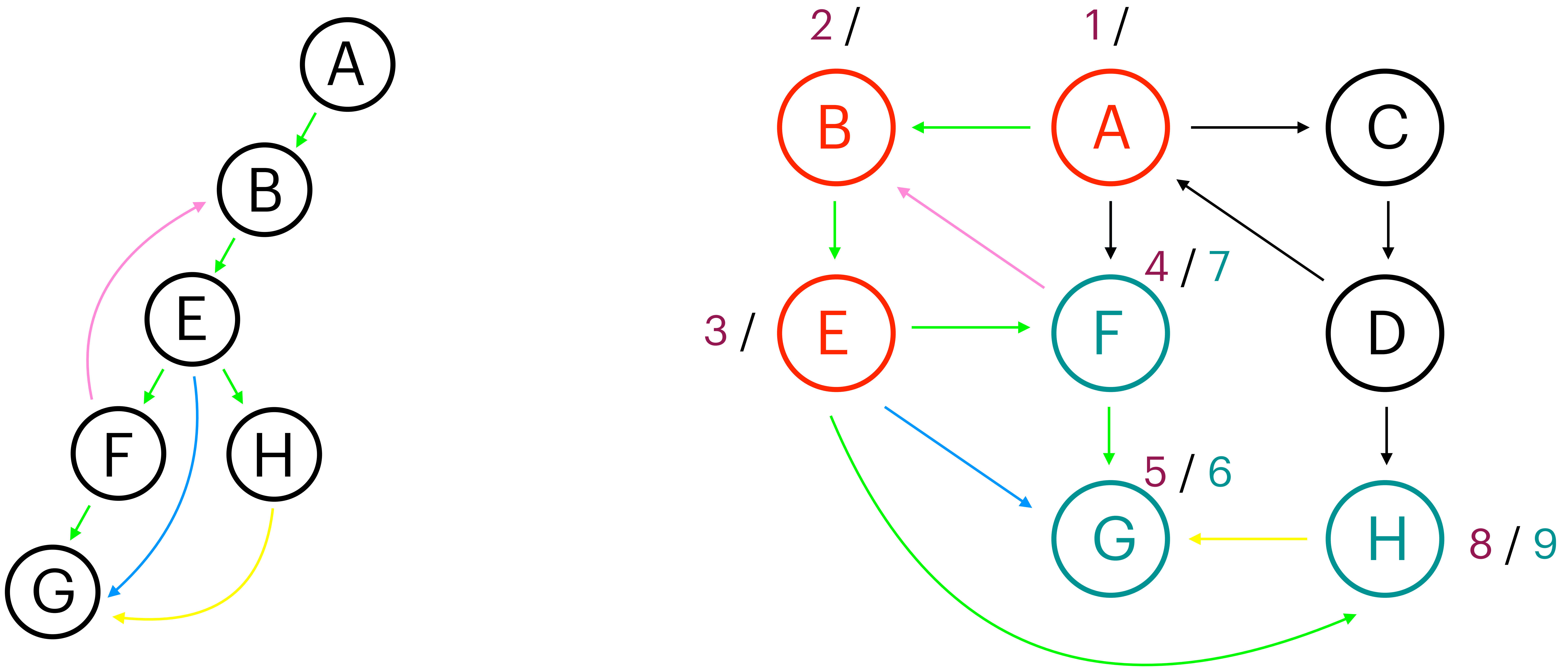
back edge

forward edge

cross edge

Graph Searches

DFS - Example



pre-order

post-order

tree edge

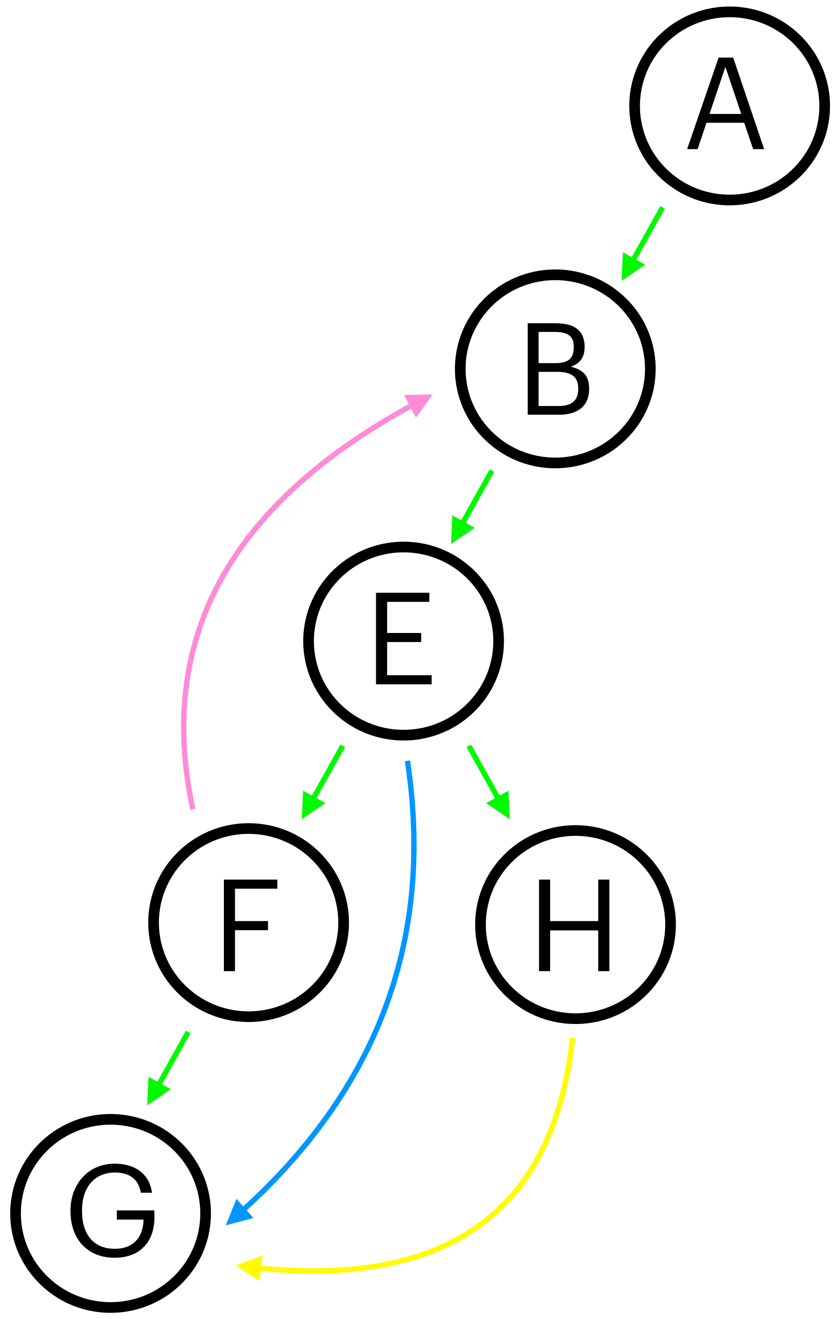
back edge

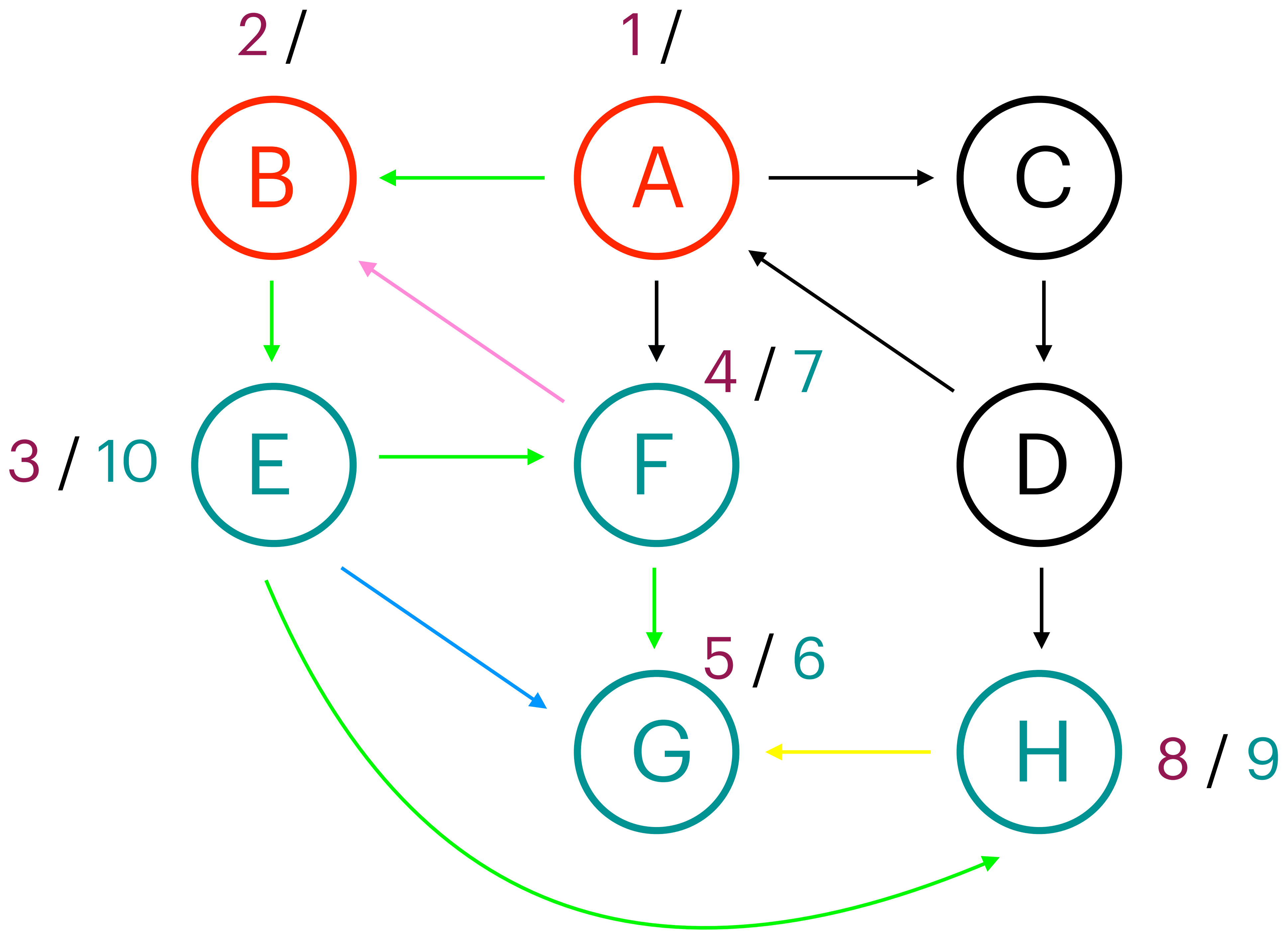
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

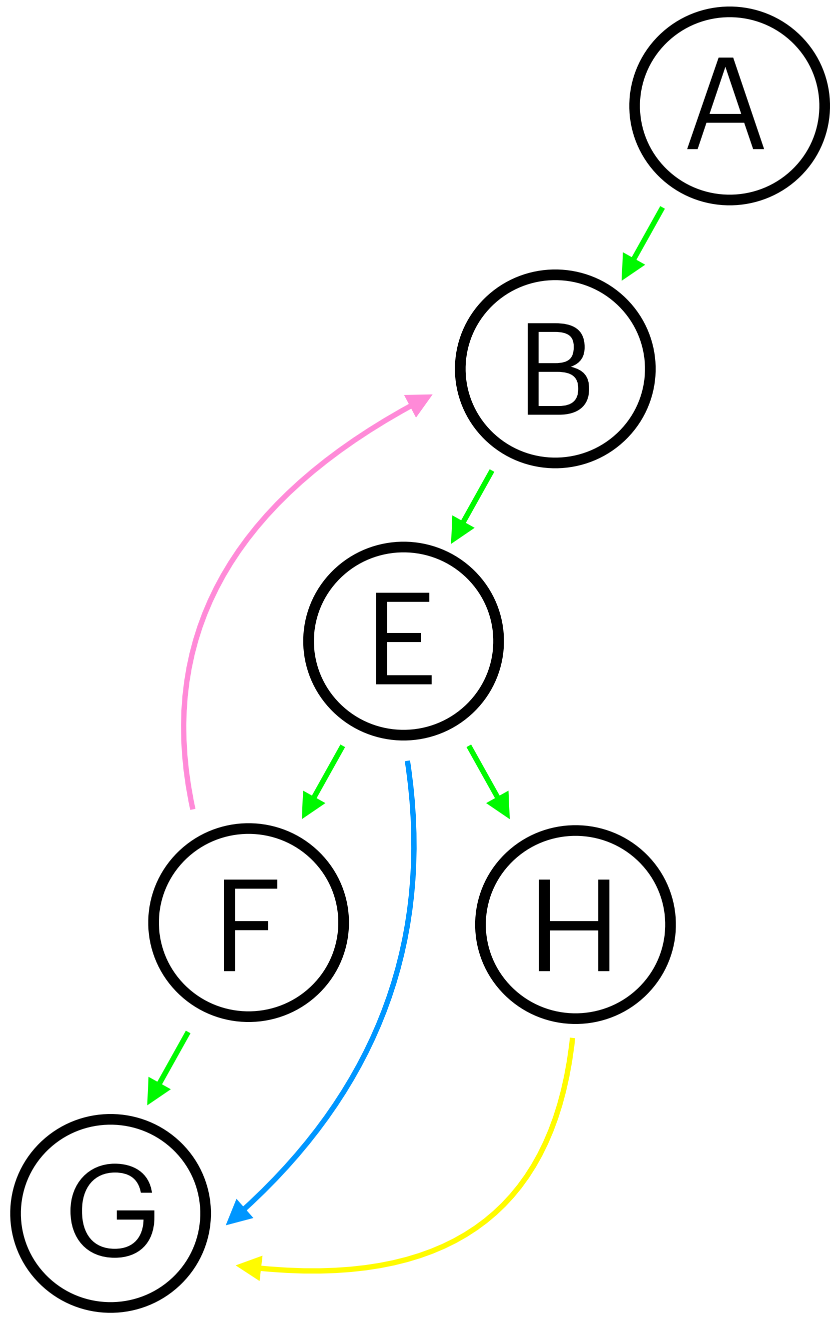
back edge

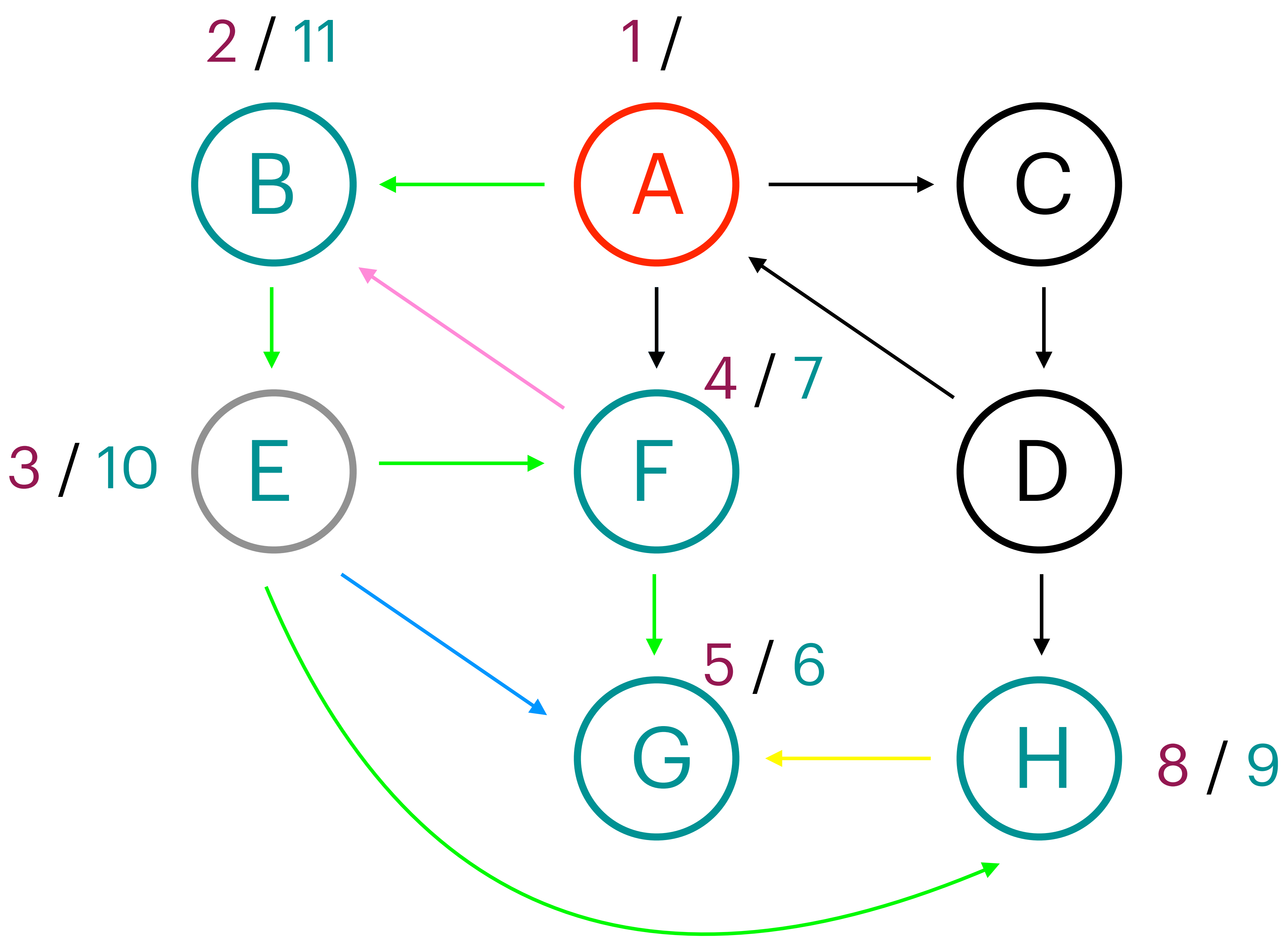
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

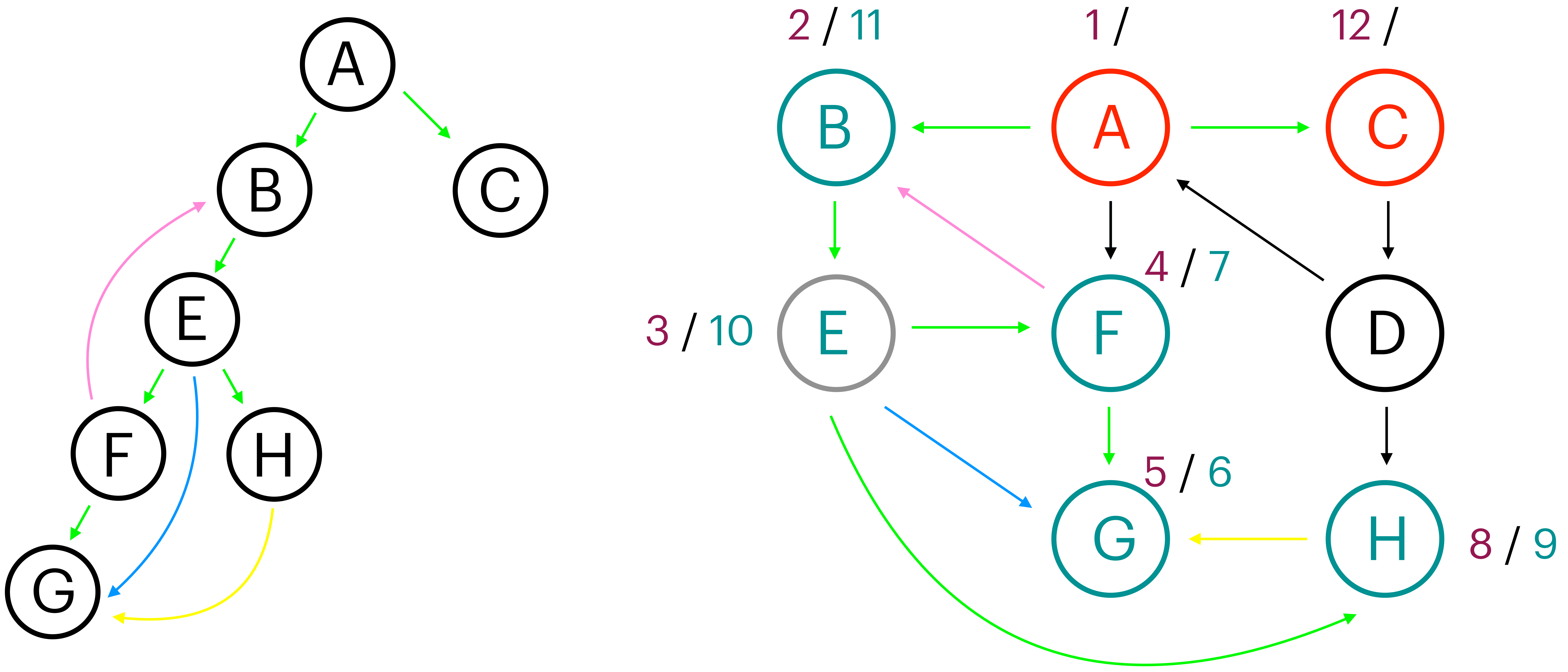
back edge

forward edge

cross edge

Graph Searches

DFS - Example



pre-order
post-order

tree edge

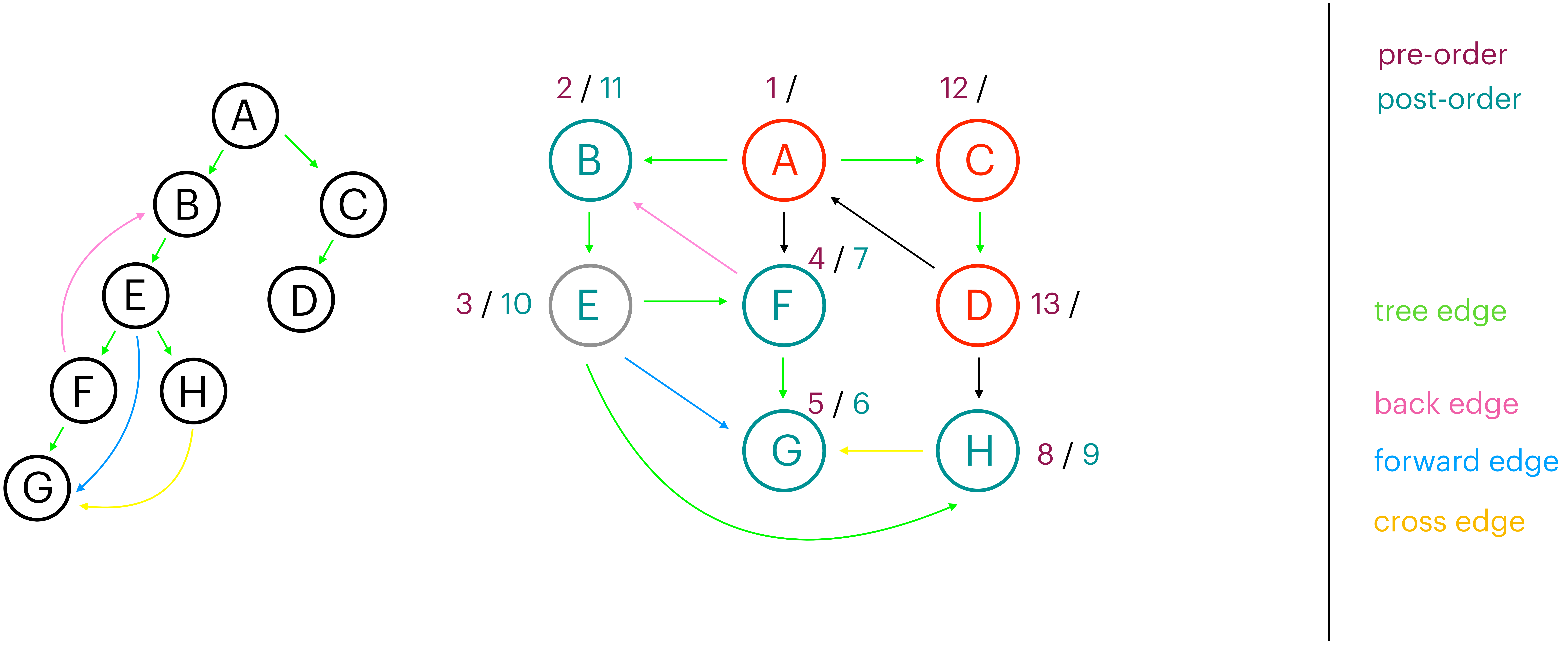
back edge

forward edge

cross edge

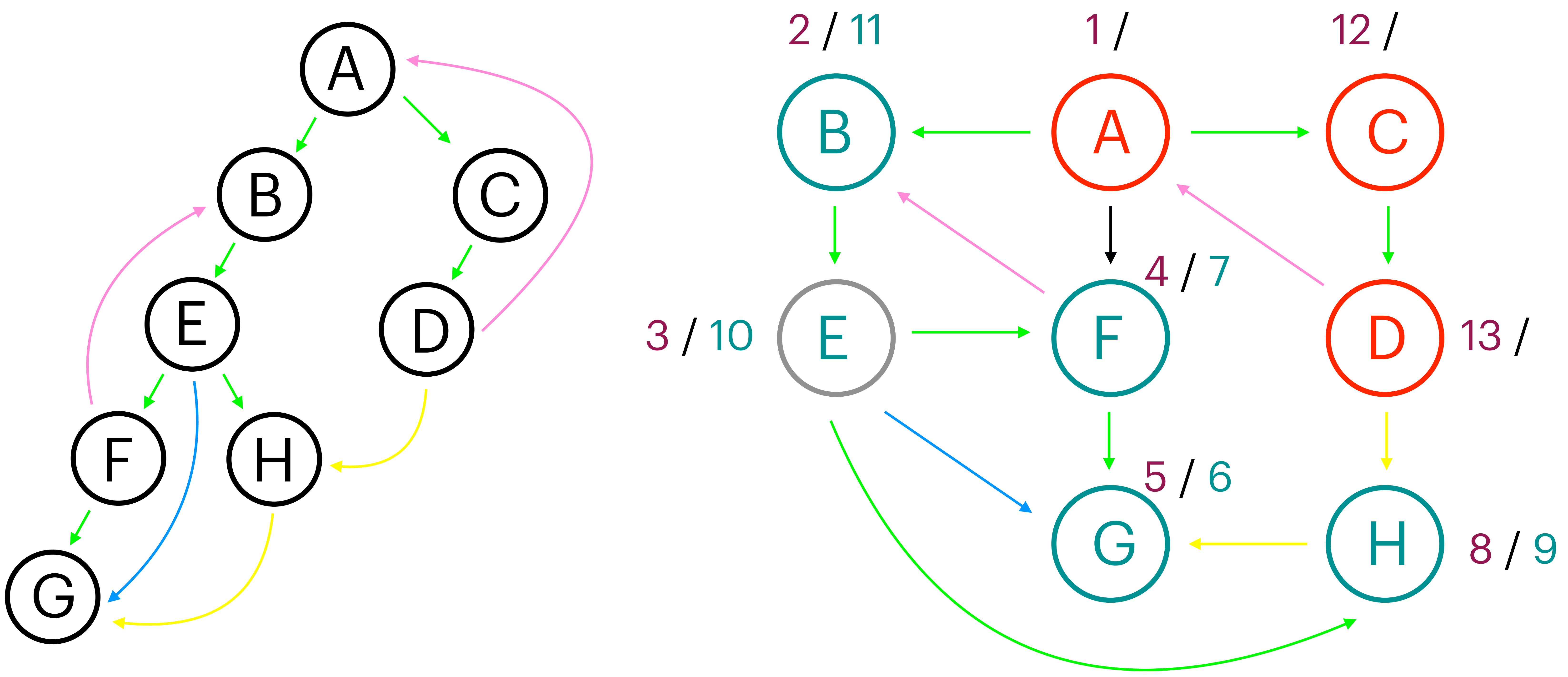
Graph Searchs

DFS - Example



Graph Searches

DFS - Example



pre-order
post-order

tree edge

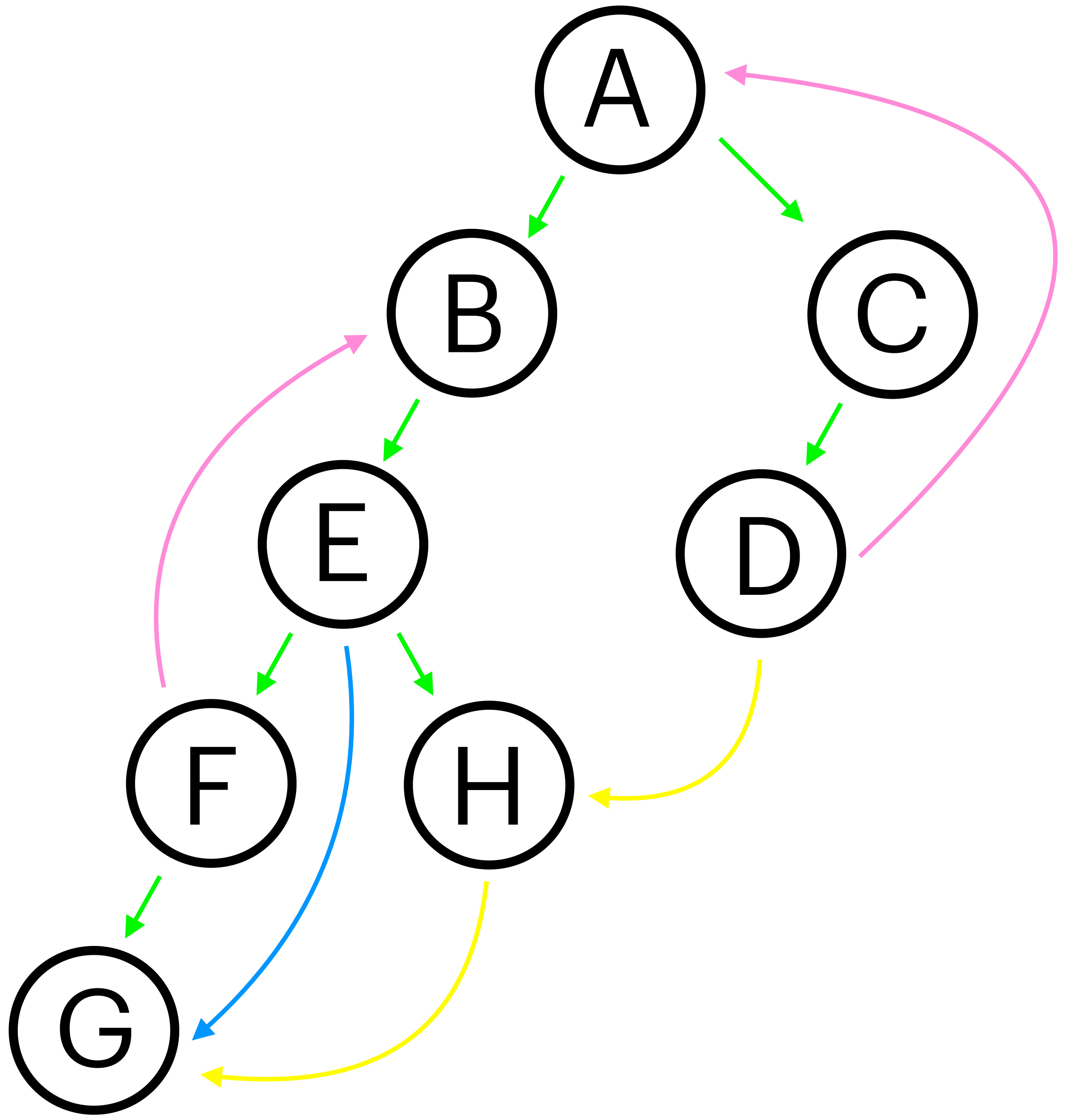
back edge

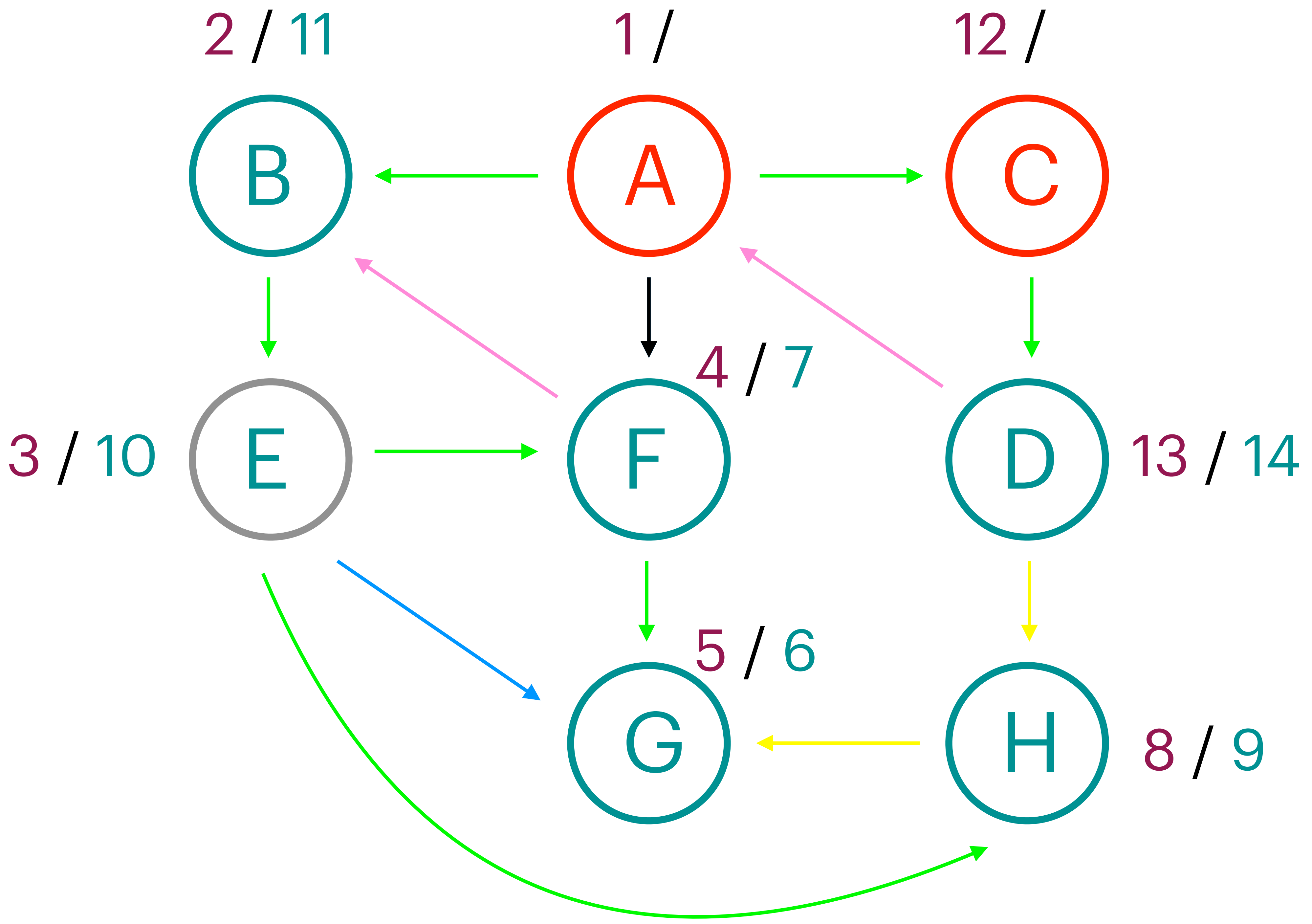
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

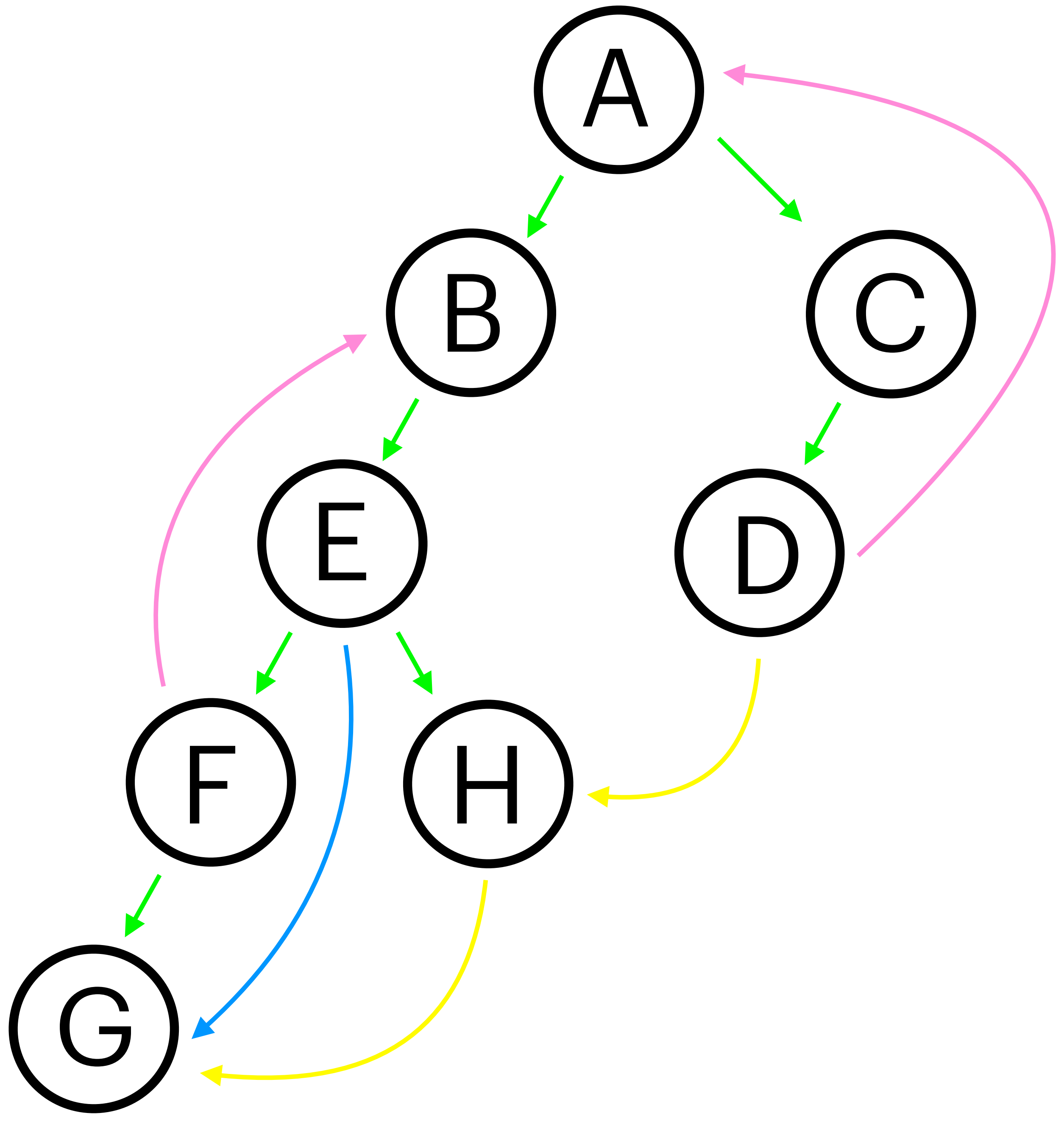
back edge

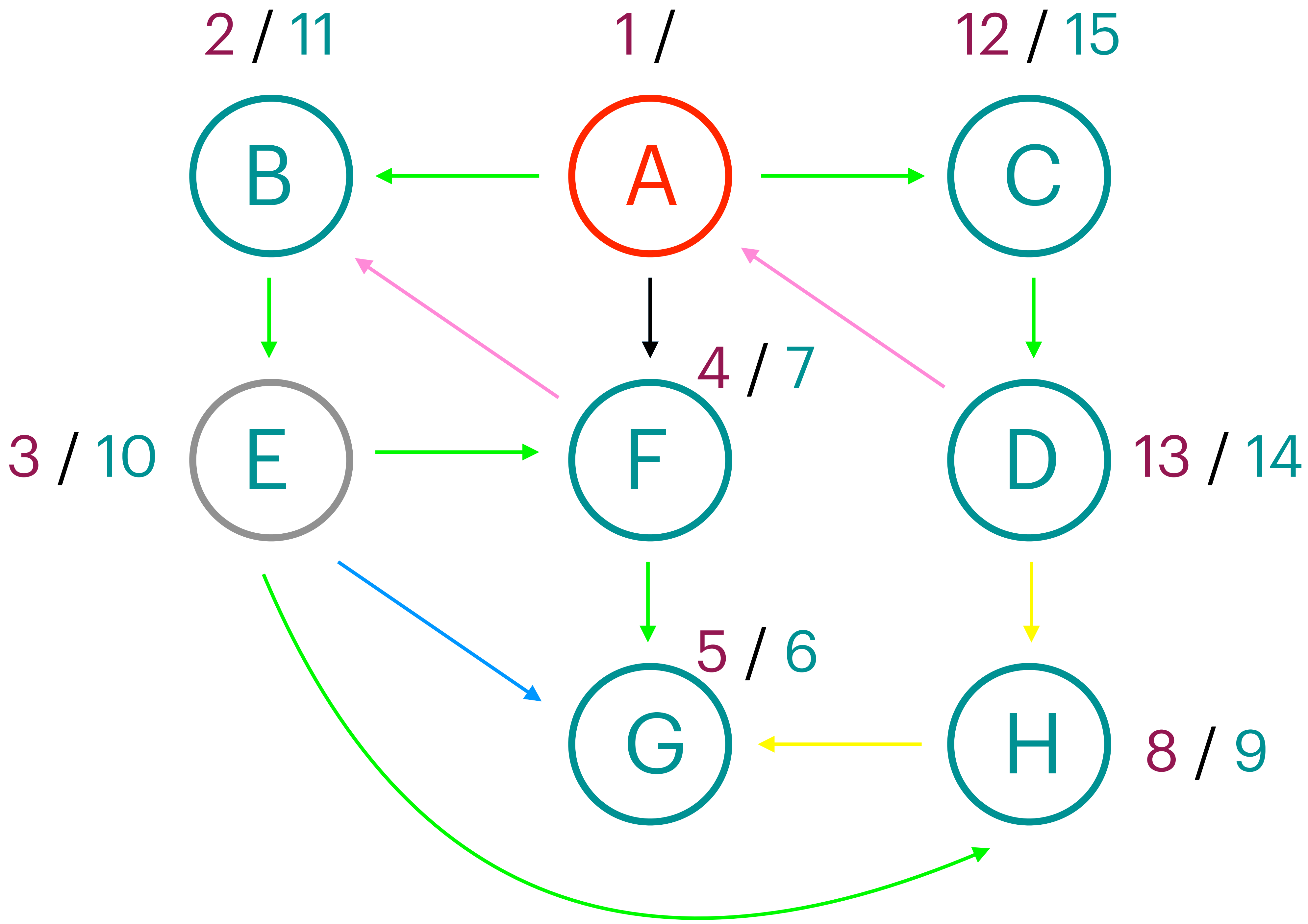
forward edge

cross edge

Graph Searches

DFS - Example





pre-order
post-order

tree edge

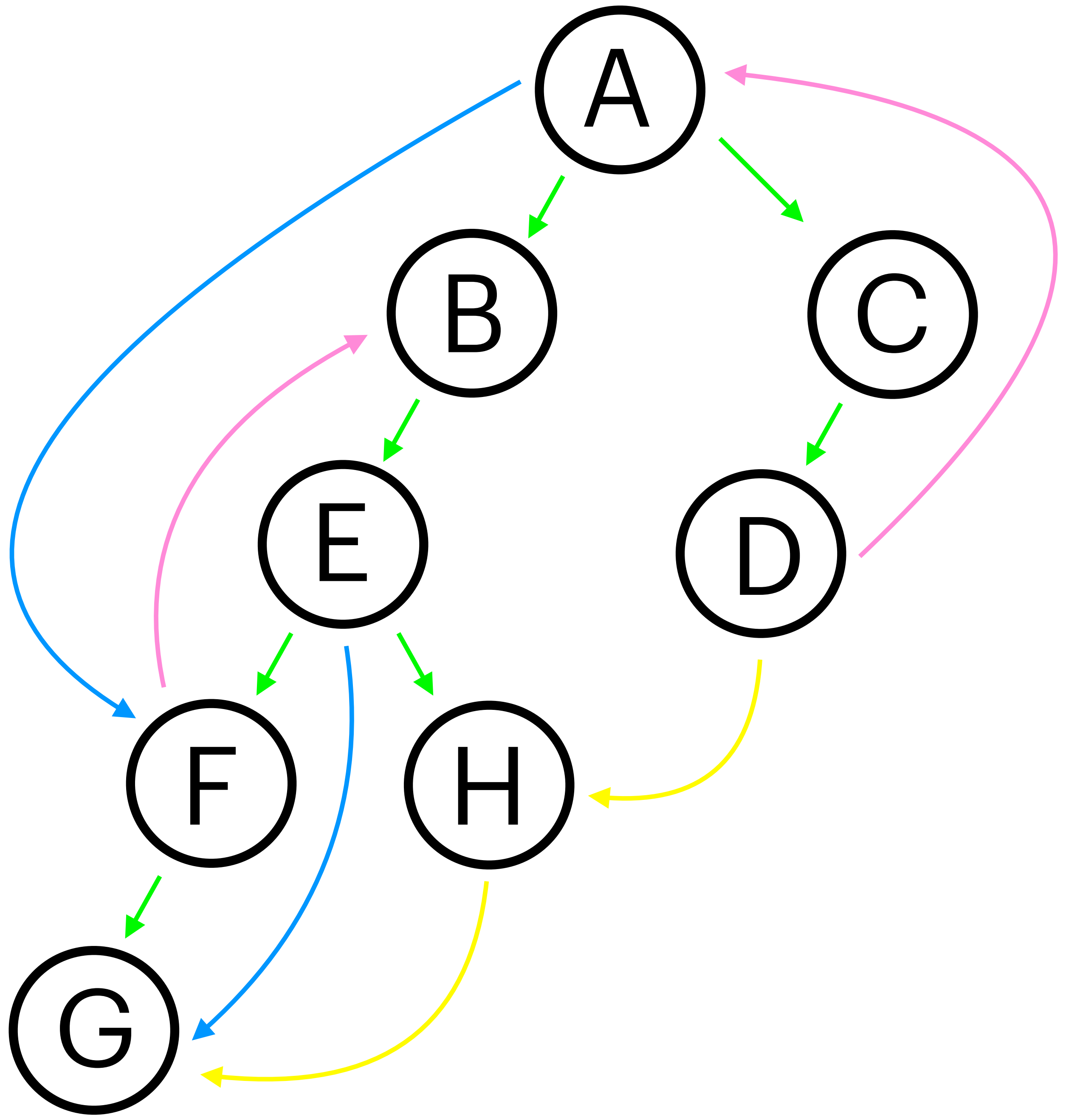
back edge

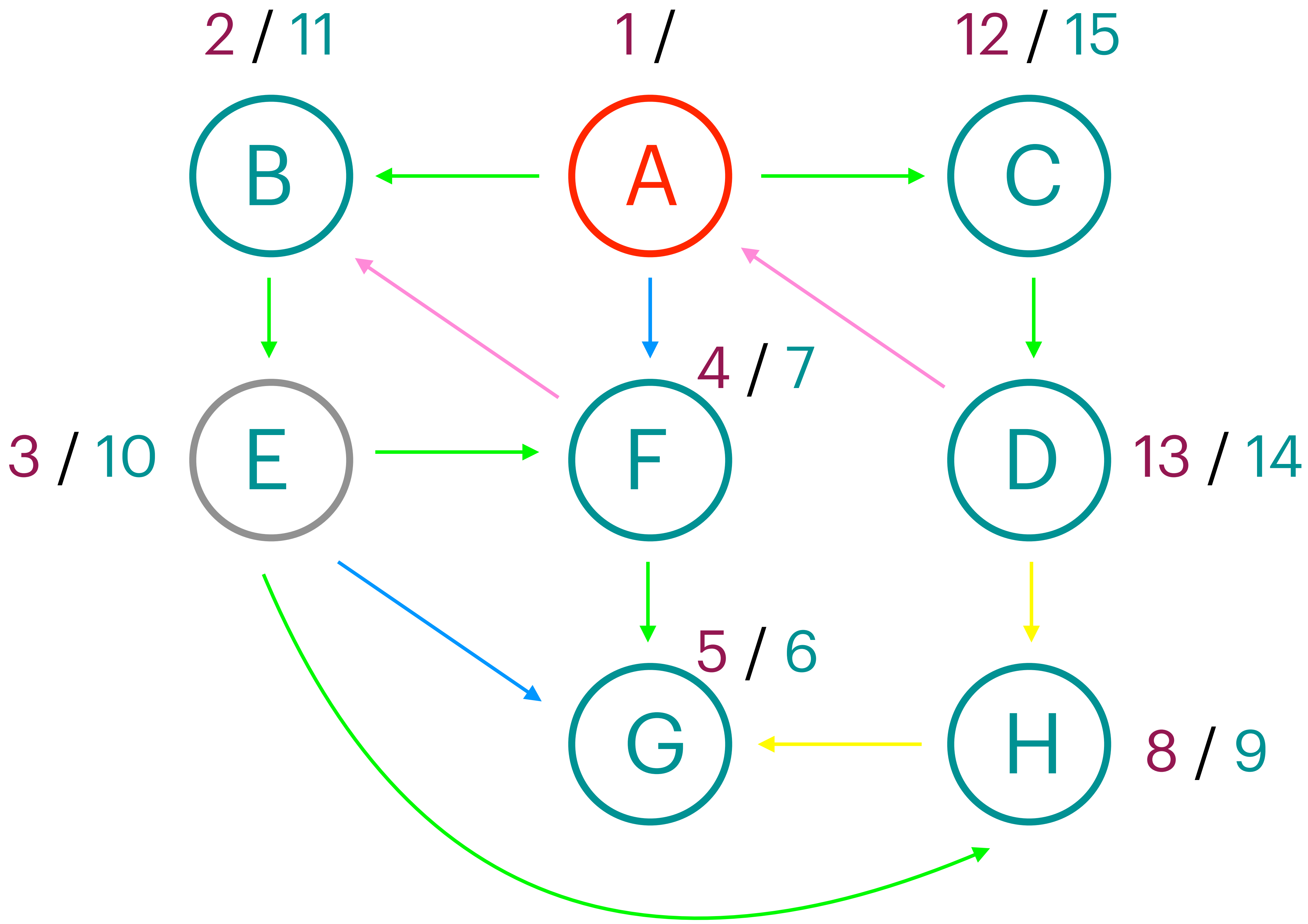
forward edge

cross edge

Graph Searchs

DFS - Example





pre-order
post-order

tree edge

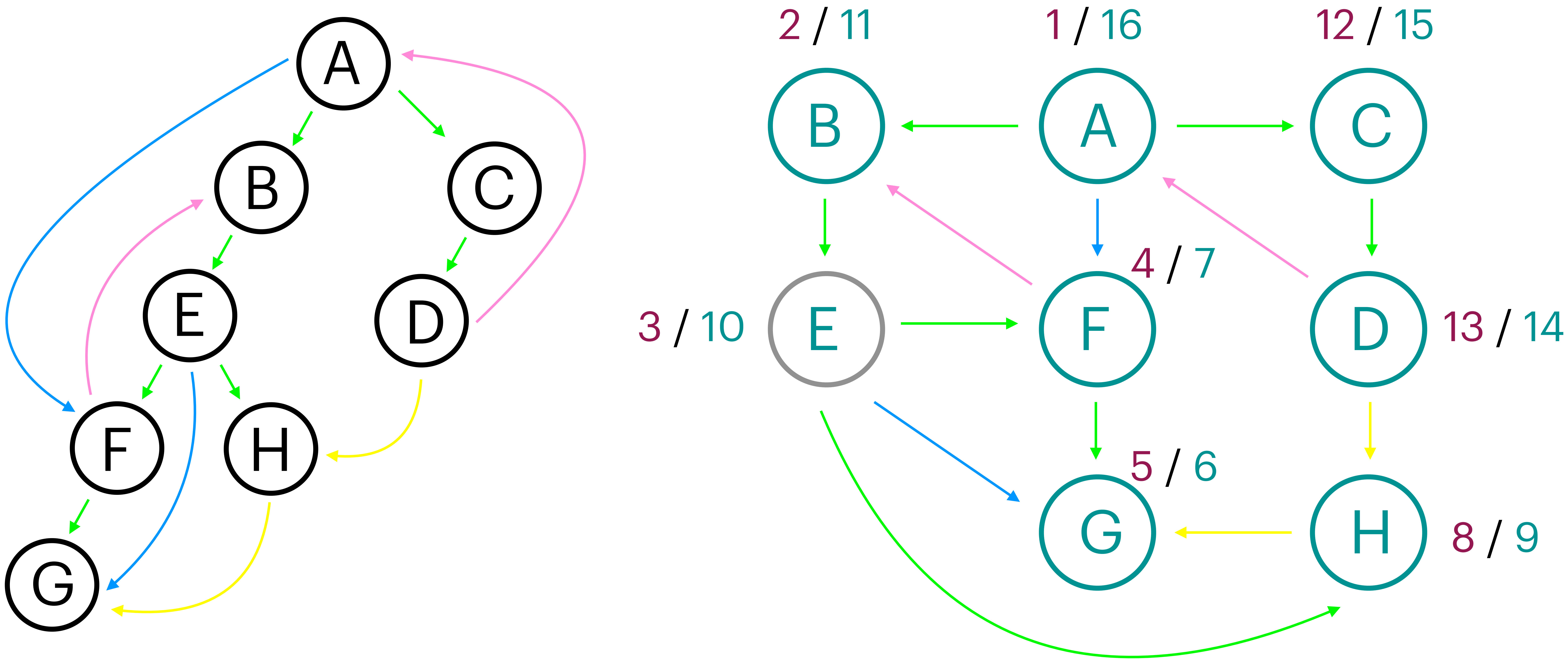
back edge

forward edge

cross edge

Graph Searchs

DFS - Example

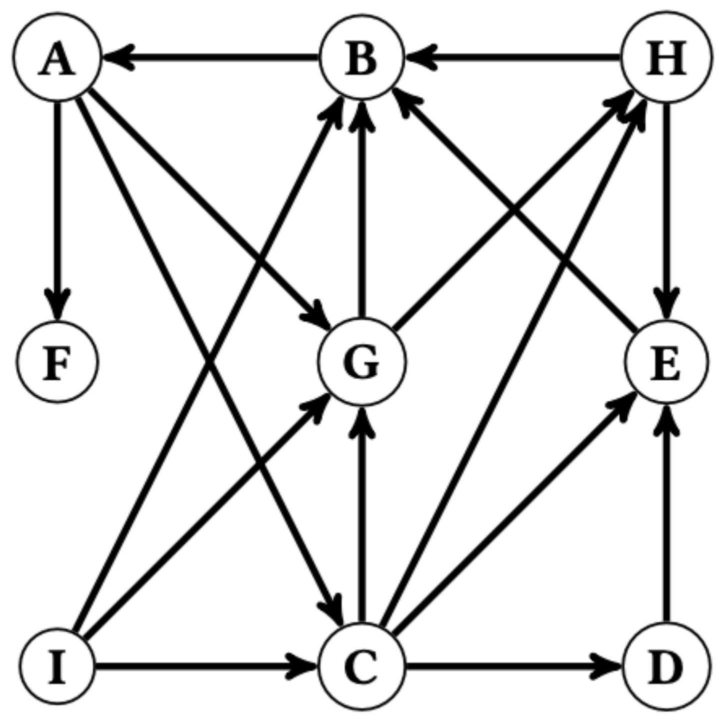


Graph Searches

DFS - Exercise Sheet Question

Exercise 10.2 *Depth-first search* **(1 point)**.

Execute a depth-first search (*Tiefensuche*) on the following graph. Use the algorithm presented in the lecture. Always do the calls to the function "visit" in alphabetical order, i.e. start the depth-first search from A and once "visit(A)" is finished, process the next unmarked vertex in alphabetical order. When processing the neighbors of a vertex, also process them in alphabetical order.



(a) Mark the edges that belong to the depth-first forest (*Tiefensuchwald*) with a "T" (for tree edge).

(b) For each vertex in the depth-first forest, give its *pre*- and *post*-number.

(c) Give the vertex ordering that results from sorting the vertices by pre-number. Give the vertex ordering that results from sorting the vertices by post-number.

(d) Mark every forward edge (*Vorwärtskante*) with an "F", every backward edge (*Rückwärtskante*) with a "B", and every cross edge (*Querkante*) with a "C".

(e) Does the above graph have a topological ordering? If yes, write down the topological ordering we get from the above execution of depth-first search; if no, argue how we can use the above execution of depth-first search to find a directed cycle.

(f) Draw a scale from 1 to 18, and mark for every vertex v the interval I_v from pre-number to post-number of v. What does it mean if $I_u \subset I_v$ for two different vertices u and v?

(g) Consider the graph above where the edge from B to A is removed and an edge from F to I is added. How does the execution of depth-first search change? Does the graph have a topological ordering? If yes, write down the topological ordering we get from the execution of depth-first search; if no, argue how we can use the execution of depth-first search to find a directed cycle. If you sort the vertices by *pre-number*, does this give a topological sorting?

Graph Searchs

DFS - Lemmas, Facts

$\exists$ a back edge $\iff$ $\exists$ a directed closed walk

For all edges (u,v) in E except back edges : post(u) > post(v)

Reversed post-order is the topological ordering !!!

Topological Sorting
Reversed post-order

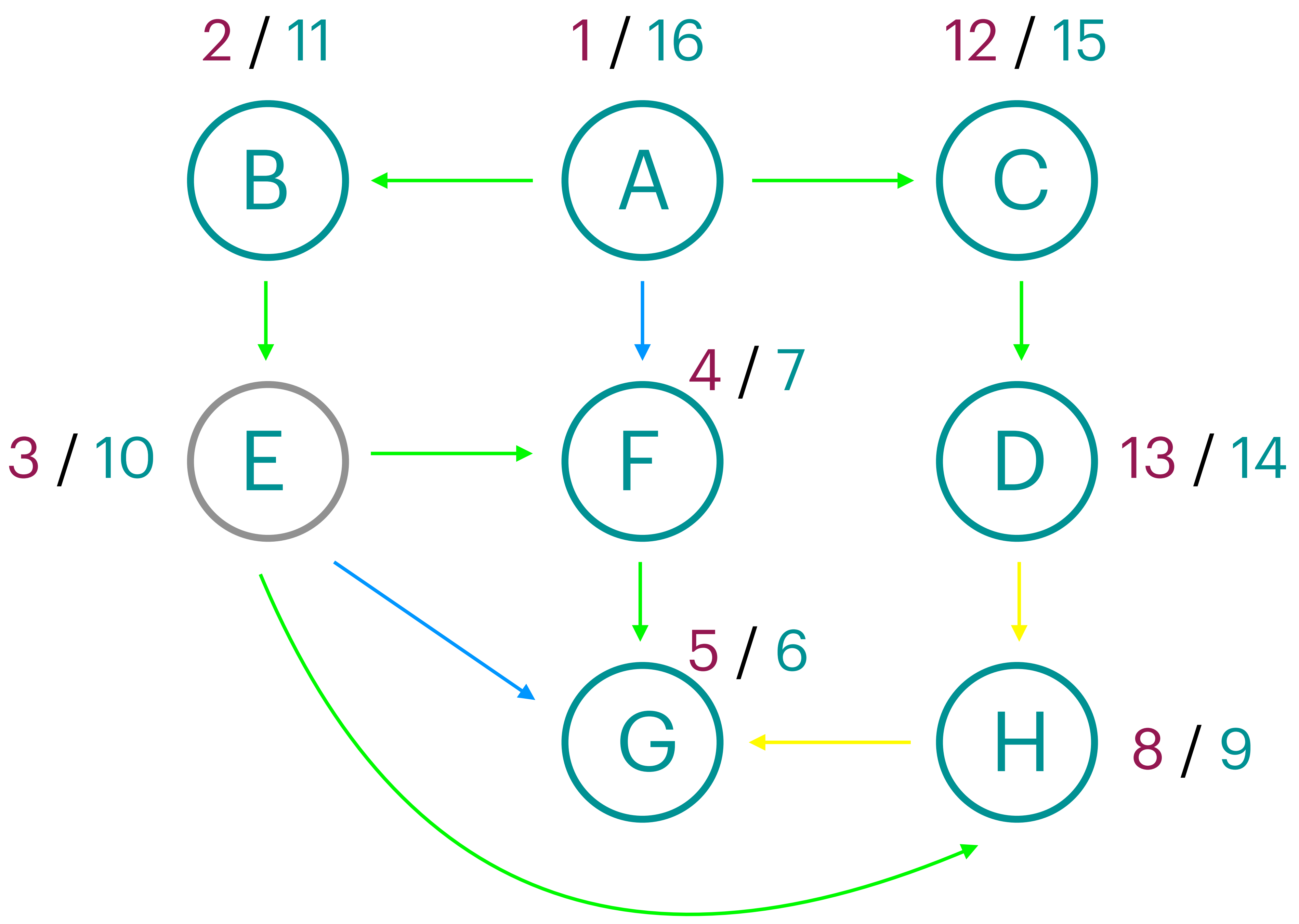
post-order :

G, F, H, E, B, D, C, A

reversed post-order :

A, C, D, B, E, H, F, G

topological sort



pre-order

post-order

tree edge

back edge

forward edge

cross edge

Topological Sorting

Lemmas, Facts

Term (German)	Term (English)	Definition
Quelle	Source	Vertex with only outgoing edges (in-degree = 0).
Senke	Sink	Vertex with only incoming edges (out-degree = 0).

$\exists$ a topological sorting $\Longleftrightarrow$ G is a DAG (Directed Acyclic Graph)

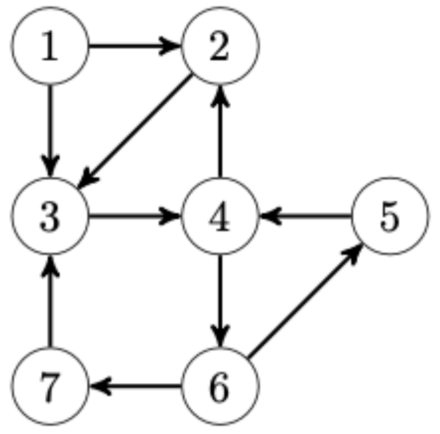
Topological Sorting doesn’t have to be unique, there can be multiple valid orders depending on the graph's structure.

DFS , Topological Sorting

HS23 , HS22

Exam Questions

/ 2 P d) *Depth-first search:* Consider the following directed graph:

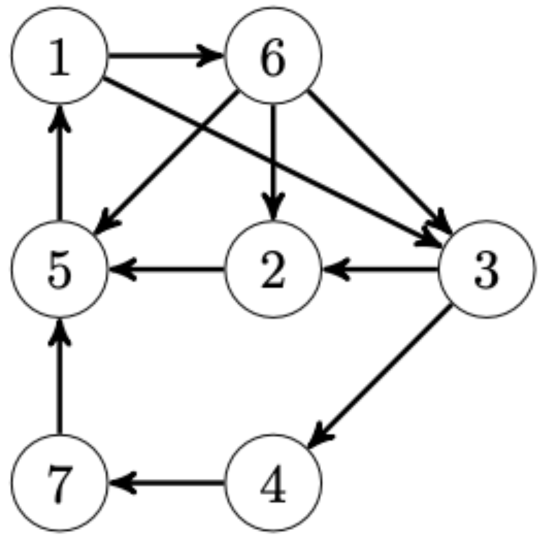


i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.

ii) Write out two edges e_1, e_2 such that the directed graph above has a topological ordering after removing e_1 and e_2 (the vertex set does not change).

Remark: There could be multiple valid solutions. In this case, you only need to write down one of them.

/ 2 P d) *Depth-first search:* Consider the following directed graph:



i) Draw the depth-first tree resulting from a depth-first search starting from vertex 1. Process the neighbors of a vertex in increasing order.

ii) Write out all the cross edges and all the back edges (specify which ones are cross edges, and which ones are back edges).

Topological Sorting

Exam Question

HS21

d) *Directed Acyclic Tournament*

A *tournament* is a directed graph $G = (V, E)$ such that:

- G has no self loops, i.e., $(v, v) \notin E$, for all $v \in V$. (Note that the graphs that we usually consider have no self loops.)
- For every two distinct vertices $u, v \in V$, either $(u, v) \in E$ or $(v, u) \in E$ but not both.

Let G be a directed acyclic graph that is also a tournament. Show that G has a unique topological sorting.

Next Week...

BFS

DFS + BFS Code Example !!

DP Mini Exam - Proof

Questions

Feedbacks , Recommendations

Nil Ozer