

Algorithm Mining

- Algorithms designed for one problem are often usable for a number of other computational tasks, some of which seem unrelated to the original goal
- Today, we are going to look at how to use the depth-first search algorithm to solve a variety of graph problems

Algorithm Mining techniques

- Deeper Analysis: What else does the algorithm already give us?
- Augmentation: What additional information could we glean just by keeping track of the progress of the algorithm?
- Modification: How can we use the same idea to solve new problems in a similar way?
- Reduction: How can we use the algorithm as a black box to solve new problems?

Graph Reachability and DFS

- Graph reachability: Given a directed graph G , and a starting vertex v , return an array that specifies for each vertex u whether u is reachable from v
- Depth-First Search (DFS): An efficient algorithm for Graph reachability
- Breadth-First Search (BFS): Another efficient algorithm for Graph reachability.

DFS as recursion

- procedure $\text{explore}(G, v)$
- Input: graph $G = (V, E)$; node v in V output:
- Output: array $\text{visited}[u]$
- 1. $\text{visited}[v] = \text{true}$
- 2. for each edge $(v, u) \in E$ do:
- if not $\text{visited}[u]$: $\text{explore}(G, u)$

Key Points of DFS

- No matter how the recursions are nested, for each vertex u , we only run $\text{explore}(G, u)$ ONCE, because after that, it is marked visited. (We need this for termination and efficiency)
- On the other hand, we discover a path to a new destination, we always explore all new vertices reachable
(We need this for correctness, to guarantee that we find ALL the reachable vertices)

DFS as iterative algorithm

procedure explore(G : directed graph, v : vertex)

Initialize array $\text{visited}[u]$ to False

Initialize stack of vertices F , PUSH v ; $\text{Visited}[v]=\text{True}$;

While F is not empty:

$v=\text{Pop}$;

For each neighbor u of v (in reverse order):

If not $\text{visited}[u]$:

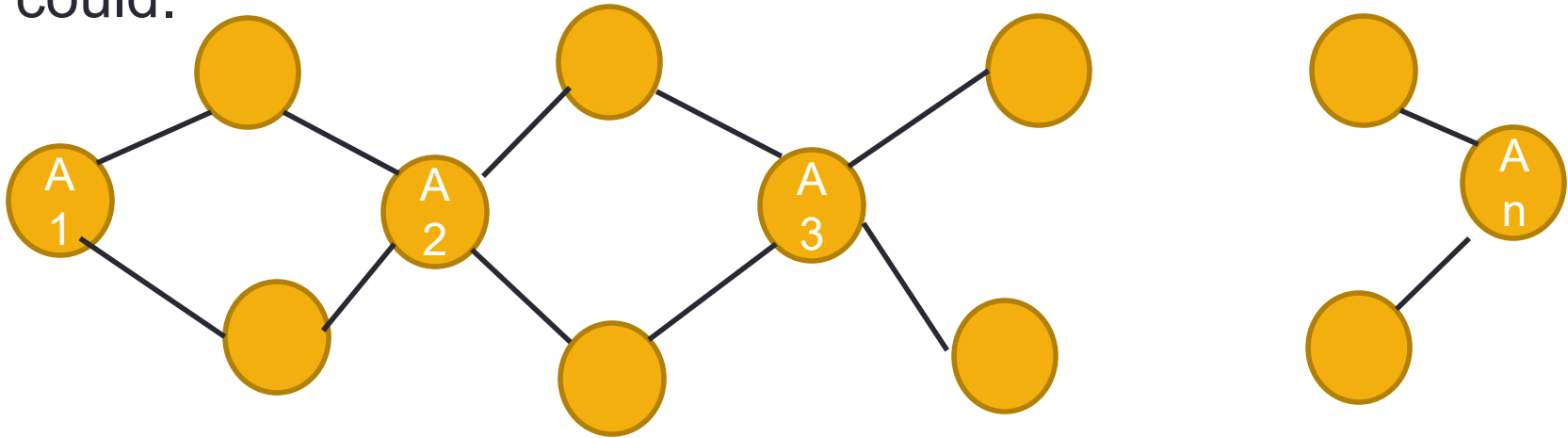
 Push u ; $\text{visited}[u] = \text{True}$;

Return visited

```
procedure explore( $G = (V, E), s$ )  
   $\text{visited}(s)=\text{true}$   
  for each edge  $(s, u)$ :  
    if not  $\text{visited}(u)$ :  
       $\text{explore}(G, u)$ 
```

All reachable vertices, not all paths

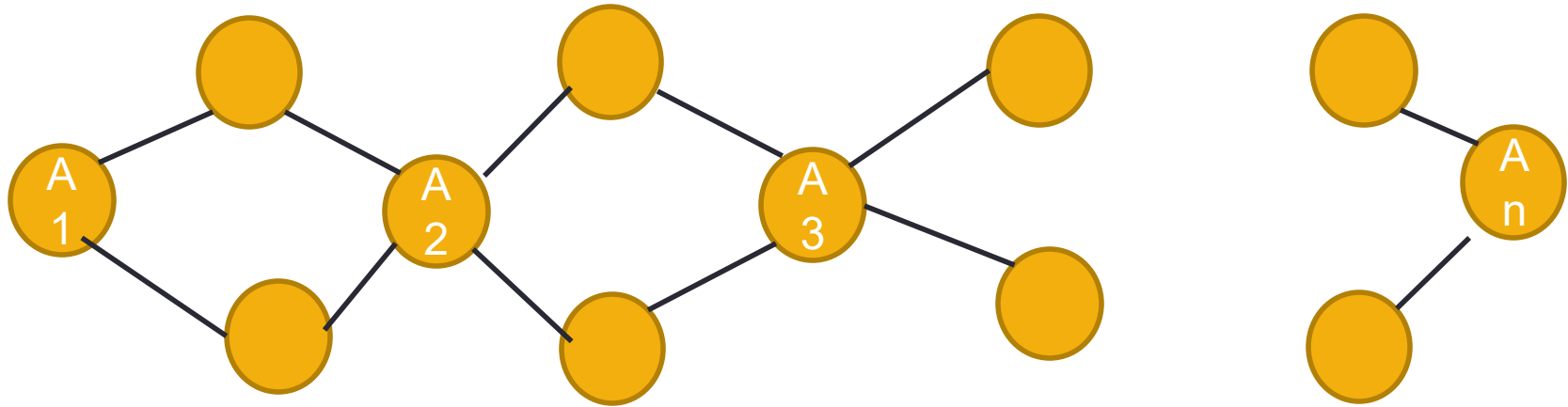
- While DFS finds all the reachable vertices, it doesn't consider all paths between them. No feasible algorithm could.



How many paths from A1 to An?

All reachable vertices, not all paths

- While DFS finds all the reachable vertices, it doesn't consider all paths between them. No feasible algorithm could.



2^{n-1} paths from A_1 to A_n

Finding paths: the DFS tree

- After the DFS, we know which vertices are reachable, but not how to get there

How long could a path in a graph be?

How about a simple path?

How many paths do we have to find?

Finding paths: the DFS tree

- After the DFS, we know which vertices are reachable, but not how to get there
- We have up to $|V|-1$ paths to find, and each path can be up to length $|V|$.

Synergy

- After the DFS, we know which vertices are reachable, but not how to get there
- We have up to $|V|-1$ paths to find, and each path can be up to length $|V|$.
- Sometimes, doing something similar many times costs less than doing it from scratch each time.

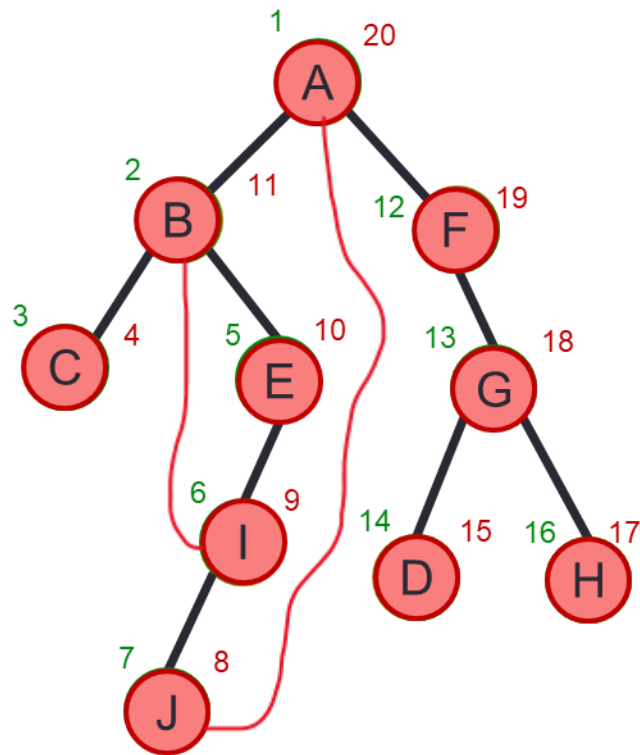
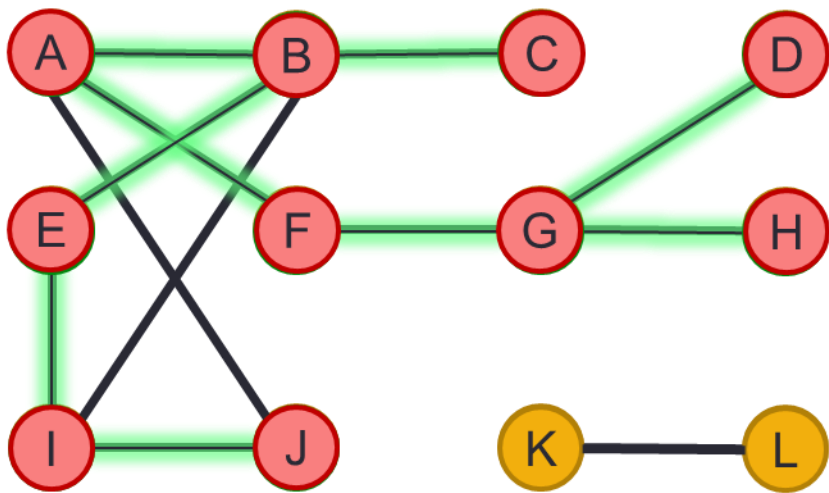
For DFS, the paths overlap, and form a $|V|-1$ edge tree

DFS augmented to create DFS tree

- procedure $\text{explore}(G, v)$
- Input: graph $G = (V, E)$; node v in V output:
- Output: array $\text{visited}[u]$; $\text{parent}[u]$
- 1. $\text{visited}[v] = \text{true}$
- 2. for each edge $(v, u) \in E$ do:
- if not $\text{visited}[u]$: $\text{parent}[u] = v$; $\text{explore}(G, u)$;

keeping track of paths

Example:

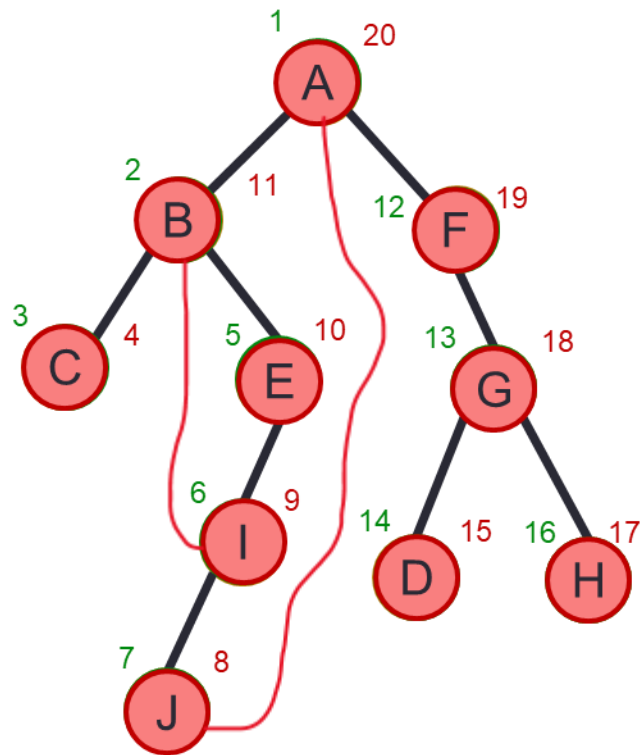
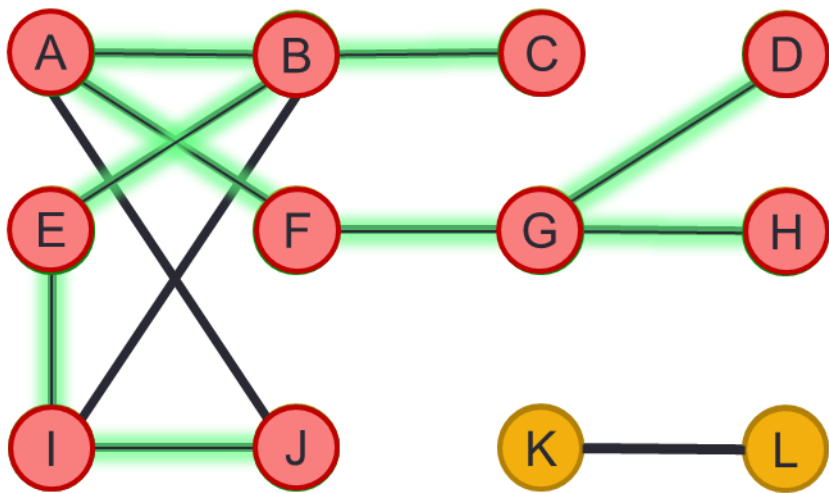


DFS augmtd. with pre, post numbers

- procedure `explore( $G, v$ )`
- Input: graph $G = (V, E)$; node $v \in V$ output: count starts at 1
- Output: array `visited[ $u$ ]`; `parent[ $u$ ]`; `pre[ $u$ ]`; `post[ $u$ ]`
- 1. `visited[ $v$ ] = true ;`
- 2. for each edge $(v, u) \in E$ do:
 - if not `visited[ $u$ ]`: `parent[ $u$ ]= $v$ ; pre[ $u$ ]=count; count++`; `explore( $G, u$ )`;
- 3. `post[ $v$ ] = count, count++`

keeping track of paths

Example:



Inferring relative position in tree

- u is below v in the DFS tree iff $\text{pre}(v) < \text{pre}(u)$ and $\text{post}(u) < \text{post}(v)$.
 - In this case, an edge from u to v creates a cycle
- u is to the right of v iff $\text{pre}(v) < \text{pre}(u)$ and $\text{post}(v) < \text{post}(u)$

Edge types (directed graph)

- Tree edge: solid edge included in the DFS output tree
- Back edge: leads to an ancestor
- Forward edge: leads to a descendent
- Cross edge: leads to neither anc. or des.: always from right to left

DFS on Directed Graphs

