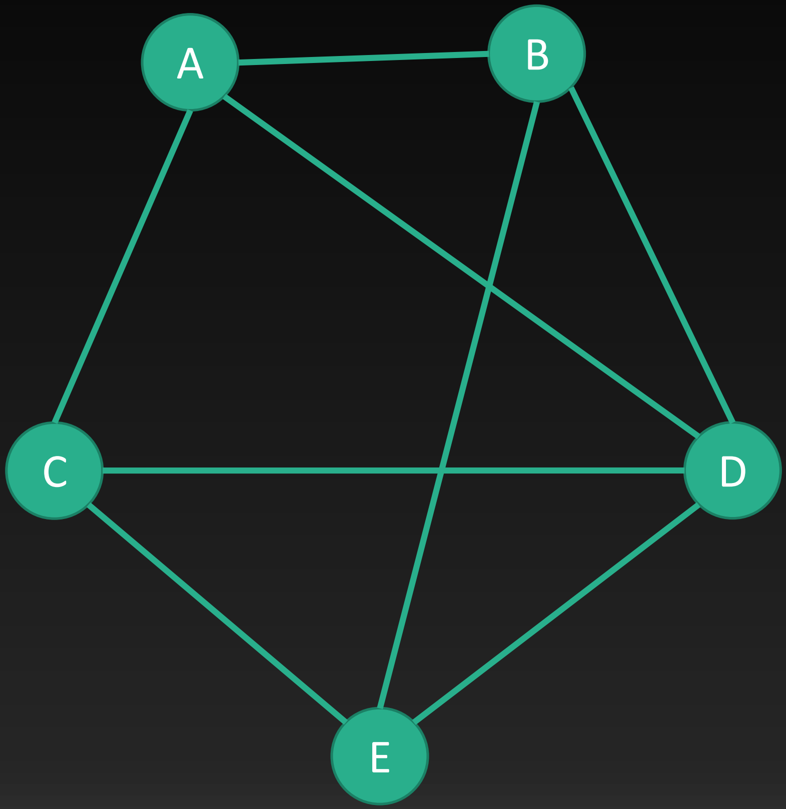
CS 4800: Algorithms & Data

Lecture 14

February 27, 2018

Graphs

- G = (V, E)
- Weight w(e) for edge e
- Undirected/directed



$$V = \{A, B, C, D, E\}$$

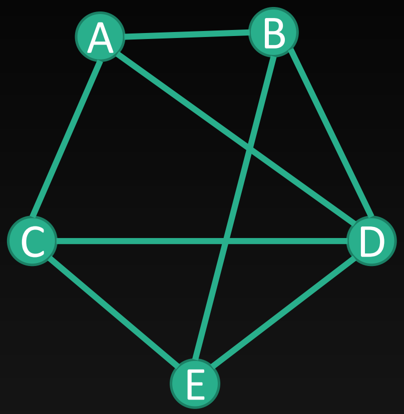
$$\mathrm{E} = \{(A,B),(A,C),(A,D),(B,D),(B,E),(C,D),(C,E),(D,E)\}$$

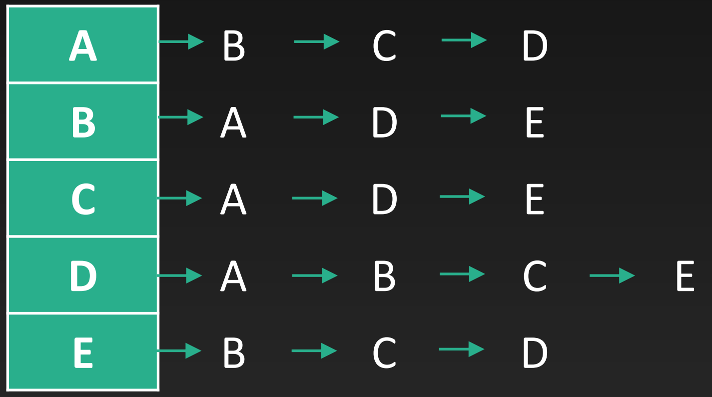
What do graphs model?

- Transportation network
  - Vertices: cities/locations
  - Edges: roads
- Communication network
  - Vertices: computers/switches
  - Edges: cable links
- Social network
  - Vertices: people
  - Edges: social connection
- Digital image
  - Vertices: pixels
  - Edges: same objects
- Large software
  - Vertices: modules
  - Edges: dependencies

Representation

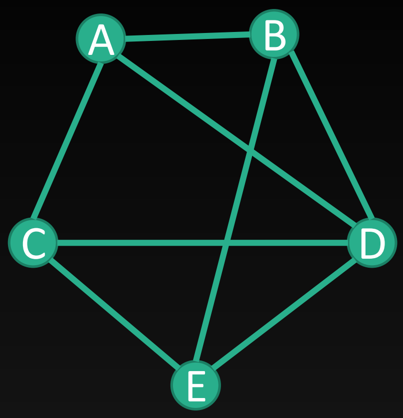
- Adjacency list
- Space: O(V+E)
- List neighbors: O(degree)
- Check edge existence: O(degree)





Representation

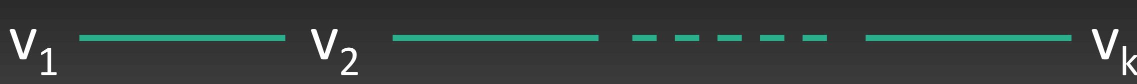
- Adjacency matrix
- Space: $O(V^2)$
- List neighbors: $O(V)$
- Check edge existence: $O(1)$



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	1
D	1	1	1	0	1
E	0	1	1	1	0

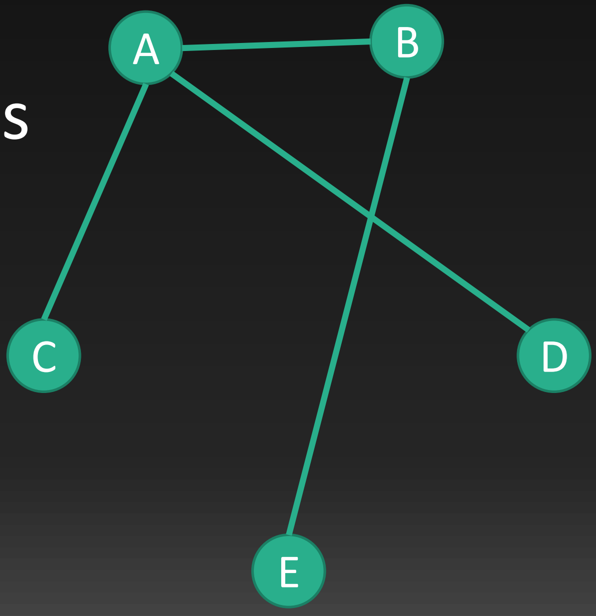
Path

- **Path**: sequence of nodes v_1, v_2, …, v_k such that $(v_i, v_{i+1}) \in E$ for all i=1, …,k-1
- **Simple path**: each vertex appears at most once
- **Cycle**: path with v_1=v_k and k > 1, each edge appears at most once
- **Simple cycle**: vertices v_1, v_2, …, v_{k-1} are distinct



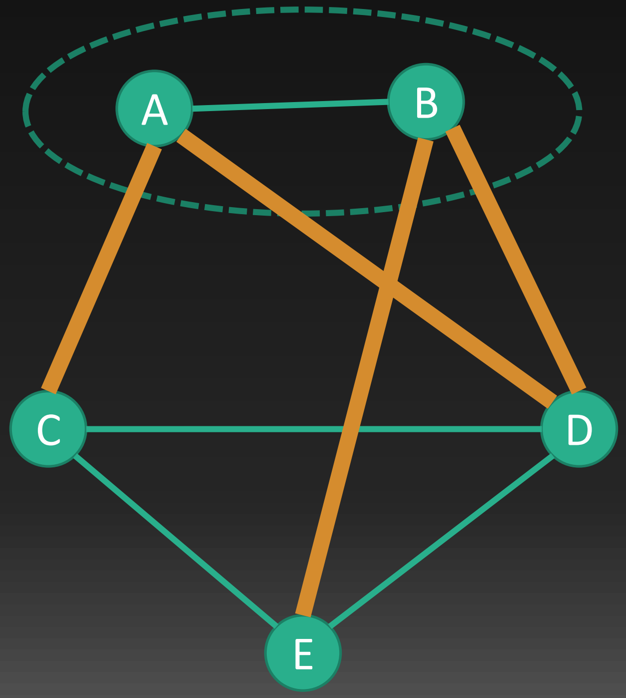
Tree

- u & v are **connected** if there is a path from u to v
- **Connected graph**: for any vertices u & v, there is a path from u to v
- **Tree**: connected graph with no cycles
- Tree on n nodes has n-1 edges



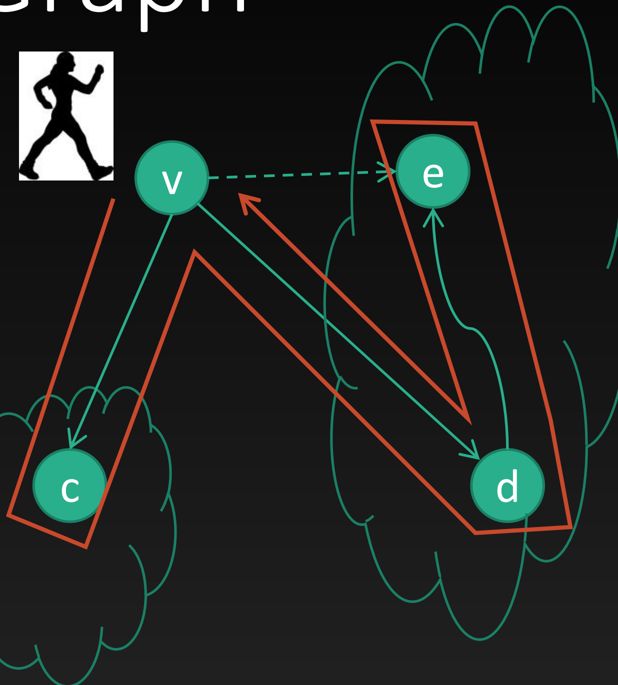
Cut

- **Cut** induced by subset $S \subset V$ is the set of edges with exactly one end point in S



(Depth-First) Search in Graph

- Search(vertex v)
  - $explored[v] \leftarrow 1$
  - For $(v, w) \in E$
    - If $explored[w] = 0$ then
      - $parent[w] \leftarrow v$
      - search(w)
  - post-visit(v)



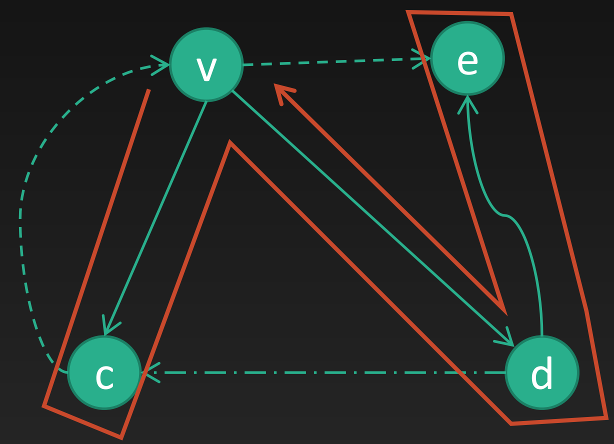
- Search(v) explores all vertices reachable from v

Connected components in undirected graphs

- Search(v) explores all vertices reachable from v
- These are exactly vertices in v's connected component
- DFS(G = (V,E))
  - For each $v \in V$
    - $explored[v] \leftarrow 0$
  - For each $v \in V$
    - If $explored[v] = 0$ then
      - search(v) // explores a new connected component

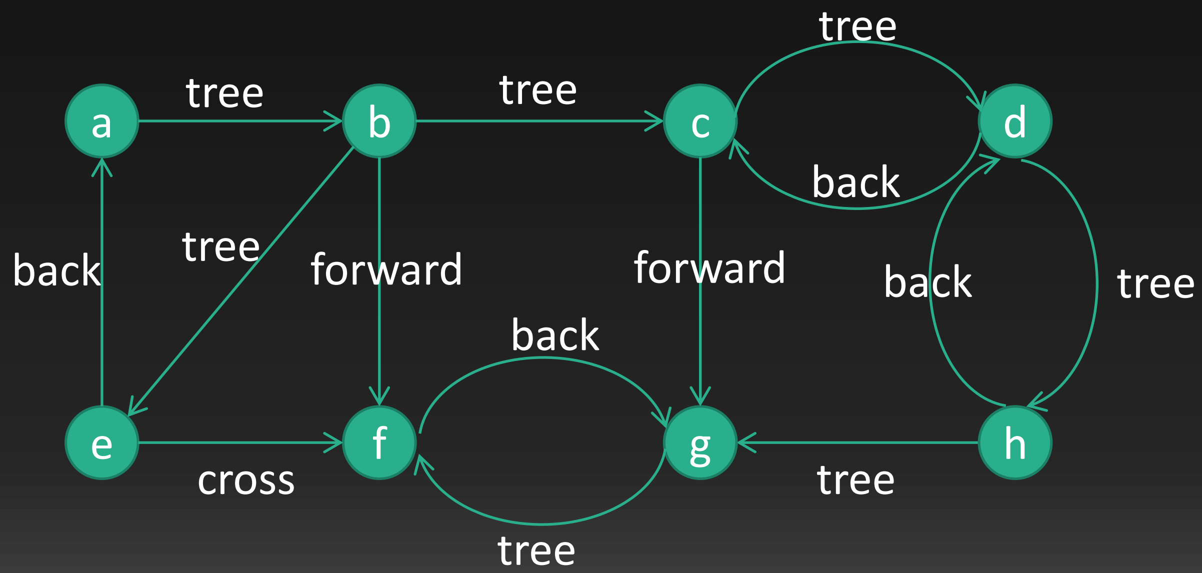
Search tree in directed graph

- The parent-child edges found by search() form a (directed) tree
- Tree edges: $(v, c), (v, d), (d, e)$
- (v, e): forward edge (edge from ancestor to descendant)
- (c, v): backward edge (edge from descendant to ancestor)
- (d, c): cross edge (no ancestral relation)



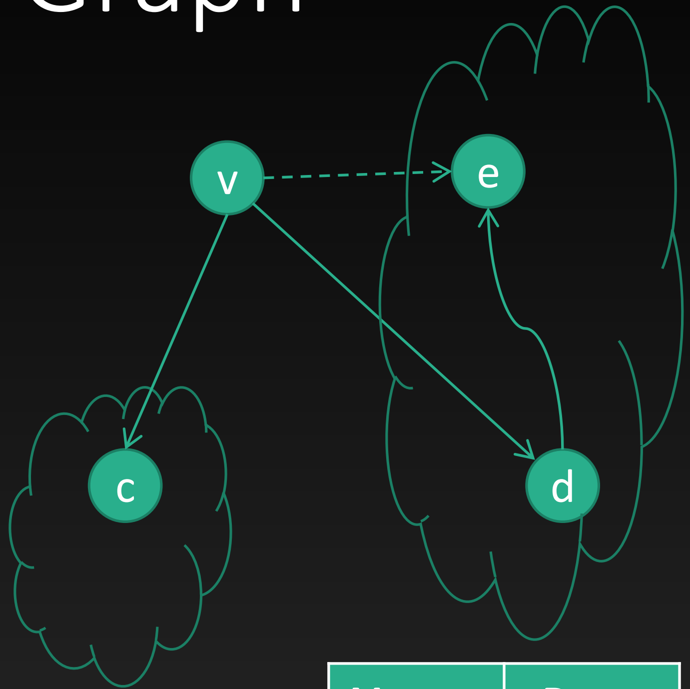
Exercise

- Label edges as tree/forward/backward/cross edges (assume we explore neighbors in alphabetical order from a)



(Depth-First) Search in Graph

- Search(vertex v)
  - $explored[v] \leftarrow 1$
  - For $(v, w) \in E$
    - If $explored[w] \ = \ 0$ then
      - $parent[w] \leftarrow v$
      - search(w)
  - post-visit(v)



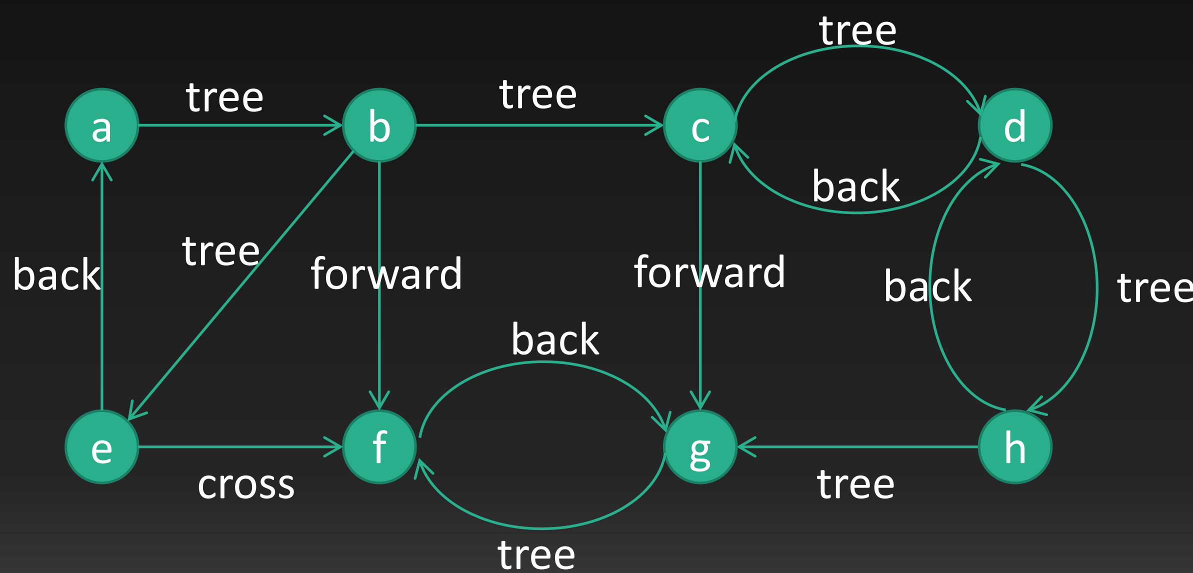
Vertex	Post-order
v	4
c	1
d	3
e	2

- Keep global counter p initialized to 0
- In post-visit(v), increase p and set postorder[v] = p

Exercise

```
Search(vertex v)
    explored[v] ← 1
    For (v,w) ∈ E
        If explored[w] = 0 then
            parent[w] ← v
            search(w)
    post-visit(v)
```

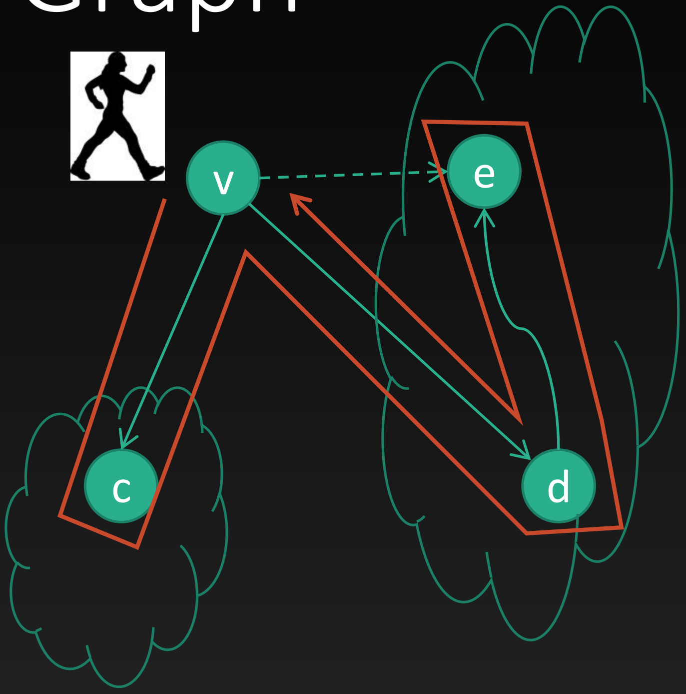
- Compute post-order array

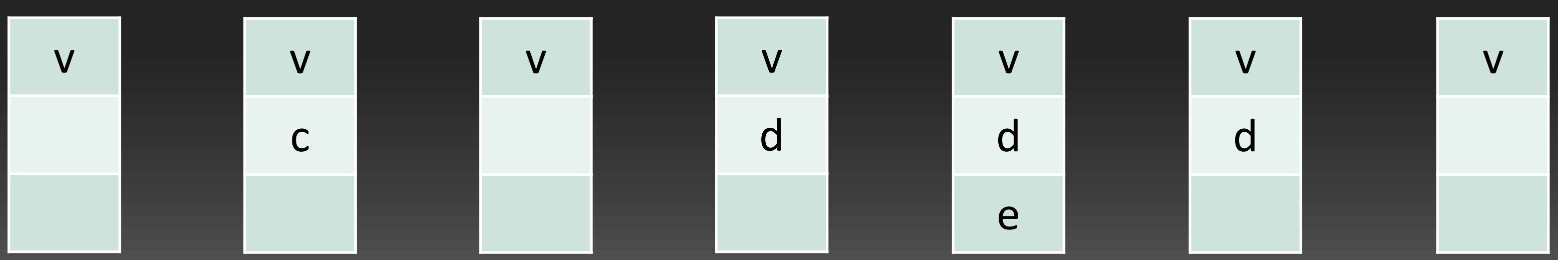


Vertex	Post-order
a	8
b	7
c	5
d	4
e	6
f	1
g	2
h	3

(Depth-First) Search in Graph

- Search(vertex v)
  - $explored[v] \leftarrow 1$
  - For $(v, w) \in E$
    - If $explored[w] = 0$ then
      - $parent[w] \leftarrow v$
      - search(w)
  - post-visit(v)

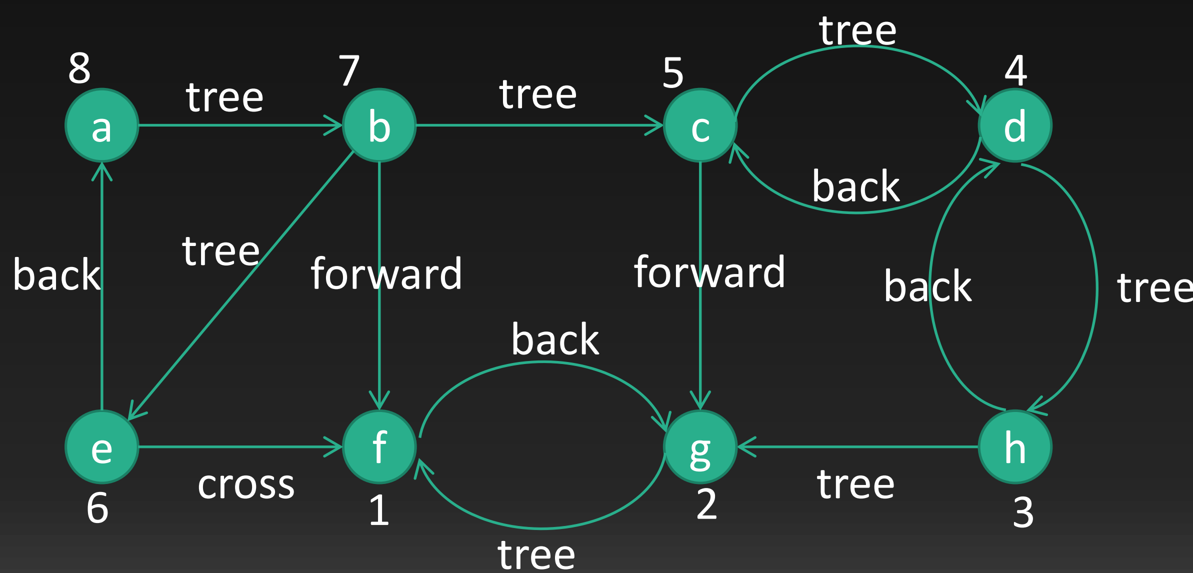




Observations

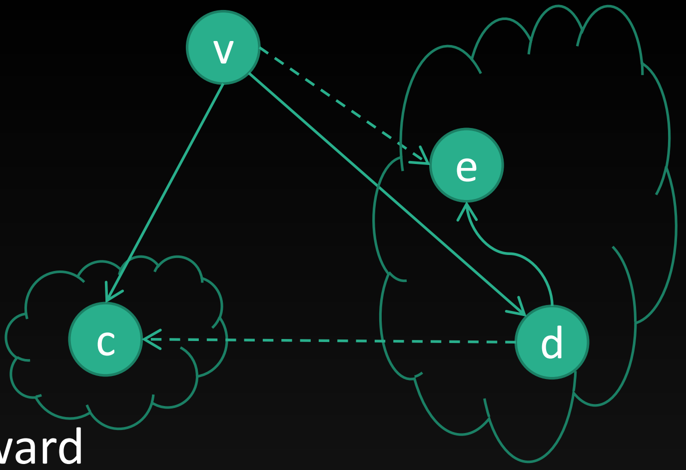
Search(vertex v)
    $explored[v] \leftarrow 1$
    For $(v, w) \in E$
        If $explored[w] = 0$ then
            $parent[w] \leftarrow v$
            search(w)
    post-visit(v)

- If $(u, v) \in E$ then $postorder[u] < postorder[v] \leftrightarrow (u, v)$ is backward



Vertex	Post-order
a	8
b	7
c	5
d	4
e	6
f	1
g	2
h	3

Observations



- If $(u, v) \in E$ then $postorder[u] < postorder[v] \leftrightarrow (u, v)$ is backward

Proof:

- search(v) finishes after searches for its children finish
  - If (u,v) is tree edge then postorder[u] > postorder[v]
  - If (u,v) is forward edge then postorder[u] > postorder[v]
  - If (u,v) is backward then postorder[u] < postorder[v]
- If $postorder[u] < postorder[v]$ then search(u) finishes before search(v).
- Thus, search(v) is not called by search(u)
- explored[v]=1 when running search(u) i.e. search(v) started before search(u)
- Search(v) starts before and ends after search(u)
  - Can only happen for backward edge
  - Cannot happen for cross edge