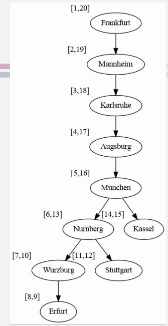
Entry and Exit Times



Recording entry and exit times –

- early process (before incident edges)

```
time = time+1
entry[v] = time
```

- late process (after incident edges)

```
time = time+1
exit[v] = time
```

Properties –

- the [entry,exit] interval for *v* is properly nested within interval for ancestor *u*
  - entry times for ancestors of *v* are smaller than for *v*, while exit times are larger
- the number of descendants of *v* is (exit[*v*]-entry[*v*])/2
  - the [entry,exit] interval for all of the descendants is properly nested within the interval for *v* – so there is both an entry and an exit for each
  - time is incremented once for each entry and once for each exit

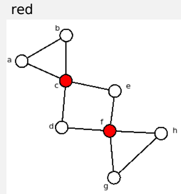
DFS

```
dfs(G,s)
  for each vertex u in V-{s} do
    state[u] = “undiscovered”
    prev[u] = null
  state[s] = “discovered”
  prev[s] = null
  dfshelper(G,s)

dfshelper(G,u)
  process vertex u (early)
  for each edge (u,v) in G.incidentEdges(u) do
    if state[v] = “undiscovered” then
      process edge (u,v)
      state[v] = “discovered”
      prev[v] = u
      dfshelper(G,v)
  state[u] = “processed”
  process vertex u (late)
```

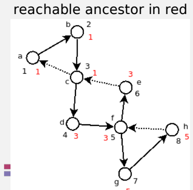

Applications of DFS – Undirected Graphs

- **articulation (cut) vertices**
  - a *cut vertex* is a vertex whose removal disconnects the graph (single point of failure)
  - a *biconnected graph* has no cut vertices (at least two vertices must be removed to disconnect)
  - observation – if a back edge connects a descendant of *v* with an ancestor of *v*, *v* is not a cut vertex
    - because the back edge forms a cycle
  - idea – for each vertex, determine its *earliest reachable ancestor* in the DFS search tree
    - number vertices in the order first encountered by DFS (entry time)
    - earliest reachable ancestor = lowest-numbered of *v*, the vertices adjacent to *v* via back edges, and the earliest reachable ancestors of children of *v*
    - *v* is a cut vertex if
      - the earliest reachable ancestor of at least one of *v*'s children is the child itself or *v*
      - if *v* is the root, it must also have two or more children

cut vertices marked in red

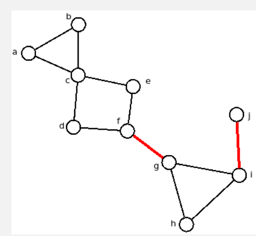


DFS tree – DFS entry order in black, earliest reachable ancestor in red

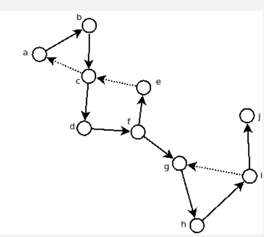


Applications of DFS – Undirected Graphs

- **bridges (cut edges)** – edges whose removal disconnects the graph
  - edge (*u*,*v*) is a cut edge if it is a tree edge and there's no back edge from *v* or a descendant of *v* to *u* or an ancestor of *u*



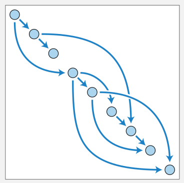
cut edges marked in red



DFS tree

Applications of DFS – Directed Graphs

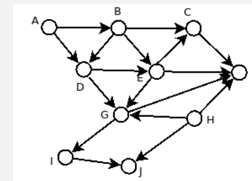
- **topological sort** – order the vertices of G so that all edges are oriented from an earlier vertex to a later one
  - possible if and only if G is a DAG (directed acyclic graph)
  - algorithm – the ordering is the reverse of the order in which vertex processing is completed (exit time) when dfs is started from a vertex *s* where indeg(*s*) = 0 (i.e. *s* has no incoming edges)

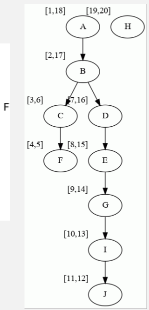


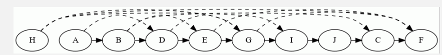
https://commons.wikimedia.org/wiki/User:David_Eppstein/Gallery

Topological Sort

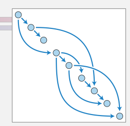
```
time = 0
for each vertex v in V do
  if G.inDegree(v) = 0
    dfs(G,v)

dfs(G,s)
  for each vertex u in V-{s} do
    state[u] = "undiscovered"
    prev[u] = null
  state[s] = "discovered"
  prev[s] = null
  dfshelper(G,s)

dfshelper(G,u)
  time = time+1
  entry[u] = time
  for each edge (u,v) in G.incidentEdges(u) do
    if state[v] = "undiscovered" then
      state[v] = "discovered"
      prev[v] = u
      dfshelper(G,v)
  state[u] = "processed"
  time = time+1
  exit[u] = time
```







Applications of DFS – Directed Graphs



- **topological sort** – order the vertices of G so that all edges are oriented from an earlier vertex to a later one
  - possible if and only if G is a DAG (directed acyclic graph)
  - algorithm – the ordering is the reverse of the order in which vertex processing is completed (exit time) when dfs is started from a vertex *s* where indeg(*s*) = 0 (i.e. *s* has no incoming edges)

---

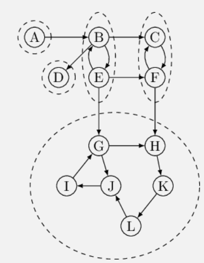
intuition

- exit timestamp for *u* is after all of the outgoing incident edges (*u*,*v*) have been processed, which means *u*'s exit timestamp is after the exit timestamps of its adacent vertices *v* and *u* occurs before *v* in the topological ordering
- edges are oriented (*u*,*v*) – *u* appears before *v* in the ordering so the edges are correctly oriented

https://commons.wikimedia.org/wiki/User:David_Eppstein/Gallery

Applications of DFS – Directed Graphs

- **is G strongly connected?** – *strongly connected* means a directed path exists between every pair of vertices
  - algorithm
    - dfs(s), then reverse all of the edges of G and repeat dfs(s) – G is strongly connected if the same set of vertices are discovered/processed each time
- **strongly connected components**
  - an algorithm
    - repeatedly compute the intersection of vertices reachable by dfs(s) and by dfs(s) with the graph's edges reversed, removing each set as a strongly connected component
  - another algorithm
    - repeatedly find a cycle and contract those vertices into a single vertex
    - when there are no more cycles, each remaining vertex represents a different strongly connected component



http://rosalind.info/glossary/algo-strongly-connected-component/

Takeaways

- DFS algorithm
- DFS-based algorithms / applications
  - graph traversal
  - reachability
  - finding cycles (undirected graphs)
  - cut vertices (undirected graphs)
  - cut edges (undirected graphs)
  - topological sort (directed graphs)
  - strongly connected / strongly connected components (directed graphs)