Graph traversal algorithms

Summary

- Main problems
  - Bipartition
  - Biconnectivity
  - Topological ordering
  - Strong connectivity
- Main algorithms
  - Breadth-first search (BFS)
  - Depth-first search (DFS)

Connectivity

Definition: A graph is connected if there is a path from every node to every other node.

Definition: A digraph is strongly connected if there is a path from every node to every other node.

In a digraph the presence of a path from u to v does not imply the presence of a path from v to u

Distances

Definition: The length of a path is the number of edges it contains.

Definition: The distance between two nodes in a graph is the length of the shortest path between them.

Definition: The diameter of a graph is the greatest distance between any two of its nodes.

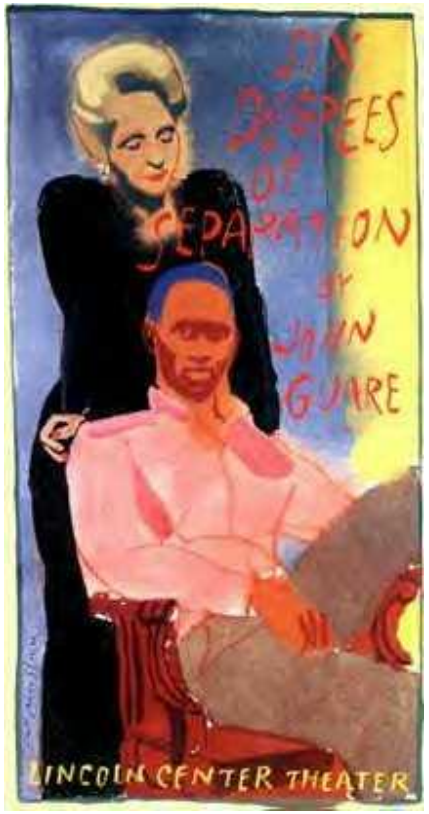
Small-world phenomenon

- The length of a typical shortest path is much smaller than the number of nodes in the graph
  - seven billion people, but “six-degrees of separation”
  - millions of routers, but tens of routers in an IP path
- Exponentially more paths at greater distances, although some do not discover new nodes!
- Randomness favors small-world phenomenon
- Hubs lead to ultra-small-worlds phenomenon

“Six degrees of separation,” play by John Guare

Ouisa’s monologue

I read somewhere that everybody on this planet is separated by only six other people. Six degrees of separation. Between us and everybody else on this planet. The president of the United States. A gondolier in Venice. fill in the names. I find that A) tremendously comforting that we're so close and B) like Chinese water torture that we're so close. Because you have to find the right six people to make the connection. It's not just big names. It's anyone. A native in a rain forest. A Tierra del Fuegan. An Eskimo. I am bound to everyone on this planet by a trail of six people. It's a profound thought. How Paul found us. How to find the man whose son he pretends to be. Or perhaps is his son, although I doubt it. How every person is a new door, opening up into other worlds. Six degrees of separation between me and everyone else on this planet. But to find the right six people.

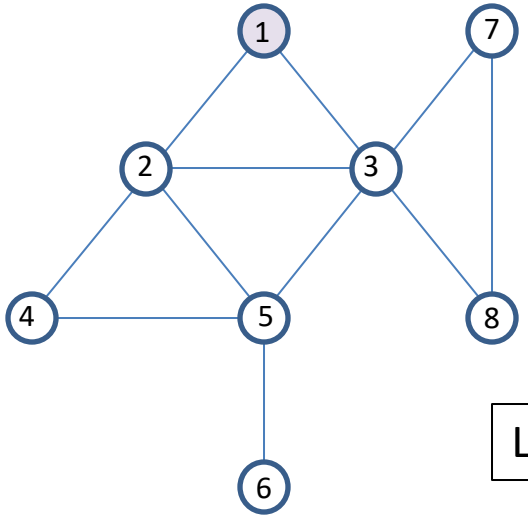


Breadth-First Search (BFS)

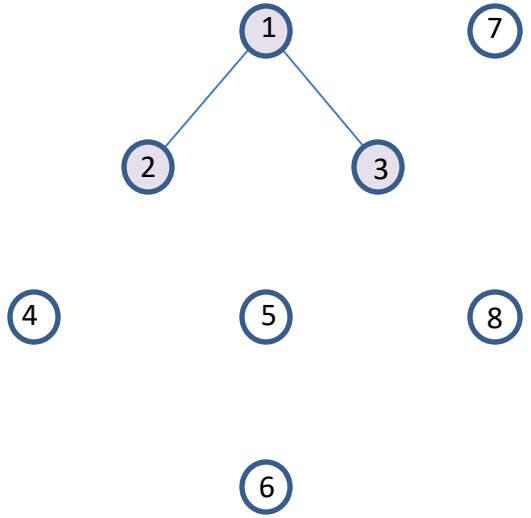
```
BFS(G, s)
  discovered[v] := FALSE for all v; parent[v] := NIL
  i : = 0; discovered[s] := TRUE; L[i] : = s
  while L[i] is not empty
    L[i + 1] := ∅
    for each u ∈ L[i]
      for each edge uv
        if discovered[v] = FALSE
          discovered[v] := TRUE
          parent[v] := u
          L[i + 1] := L[i + 1] ∪ v
        endif
      endfor
    endfor
    i := i+1
  endwhile
```

O(m+n) complexity

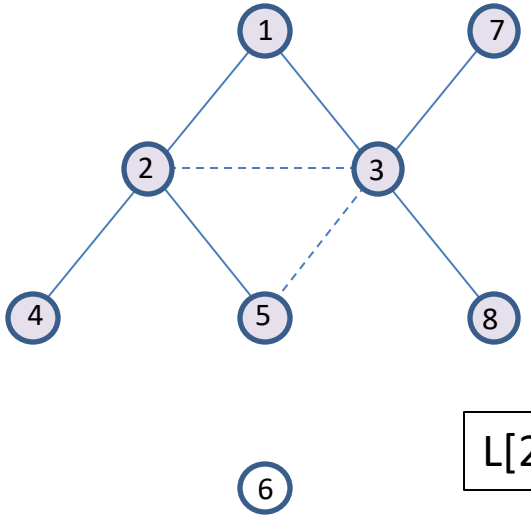
BFS example



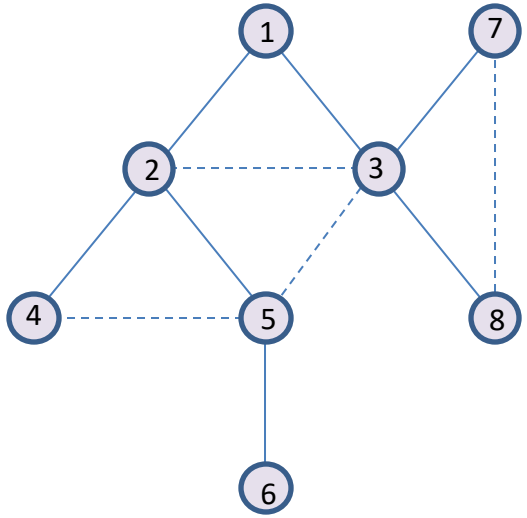
L[0] =1



L[1] = 2, 3



L[2] = 4, 5, 7, 8



L[3] = 6

BFS and shortest paths

Observations:

(1) Layer L[0] consists of s;

(2) Layer L[i + 1] consists of all nodes that do not belong to an earlier layer and are neighbors of a node at layer L[i].

Proposition: Nodes at layer L[i] are at distance i from s.

Proof (induction):

(1) Base case: s is at distance 0 from itself

(2) Induction step: The distance from s to a node v at layer L[i + 1] is greater than i because of the induction hypothesis. The distance equals or is smaller than i + 1 because there is a path of length i + 1 from s to v via a node in layer i

BFS tree

Proposition: The BFS tree consisting of the discovered nodes and the edges (parent[v], v) is a tree of shortest paths from s.

Proposition: Let xy be an edge between two nodes in the BFS tree. Then, the distances from s to x and to y differ by at most one.

Proof (contradiction): Suppose that the distance from s to x is i and that from s to y is j, with $j > i + 1$. Then x belongs to L[i] and since there is an edge xy, y must belong to L[i + 1].

BFS with a FIFO queue

```
BFS(G, s)
  discovered[v] := FALSE for all v; parent[v] := NIL; L := ∅
  discovered[s] := TRUE; enqueue(L,s)
  while L is not empty
    u := dequeue(L)
    for each edge uv
      if discovered[v] = FALSE
        discovered[v] := TRUE
        parent[v] := u
        enqueue(L,v)
      endif
    endfor
  endwhile
```

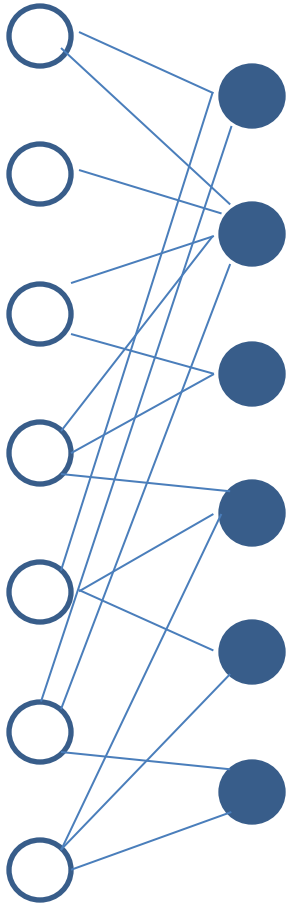
Variable discovered can be substituted by variable distance that records the distance from s to every node

Bipartite graphs

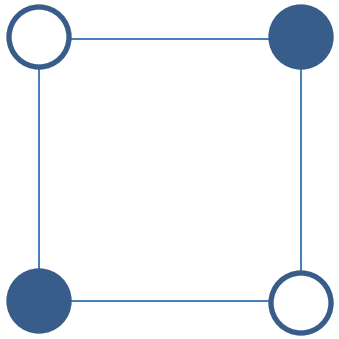
Definition: A graph is bipartite if its set of nodes can be partitioned into two subsets such that every edge joins nodes in different subsets.

Proposition: A graph is bipartite if and only if all its cycles are of even length.

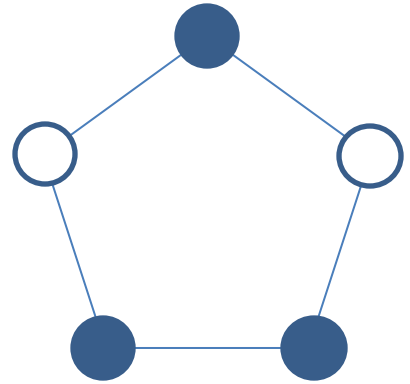
Examples



Bipartite



Bipartite



Non-bipartite

Bipartitness – sufficiency of even length cycles

Proposition: A graph is bipartite **if** and only if all its cycles are of even length.

Proof (sketch): Let T be a (any) spanning tree and s a root node. Let xTy denote the unique tree path from x to y.

Fix some node s. Partition the nodes such that x is in the first partition if sTx has even length and it is in the second partition if sTx has odd length. Hence, if the length of xTy is odd, then x and y belong to different partitions.

Let xy be an edge. If xy belongs to T, then x and y lie in different partitions. Otherwise, if xy does not belong to T, then $xTy + xy$ is a cycle. Since $xTy + xy$ is of even length, xTy is of odd length, and x and y lie in different partitions.

Necessity of even length cycles is easy

Test for bipartite graphs: algorithm

Algorithm Bipartiteness: Run a BFS with root s. The graph is bipartite if and only if no edge joins two nodes at the same distance from s.

Recall that every edge joins two nodes either at the same distance from s or whose distances from s differ by one.

The algorithm can be truncated as soon as an edge between two nodes at the same distance from s are found.

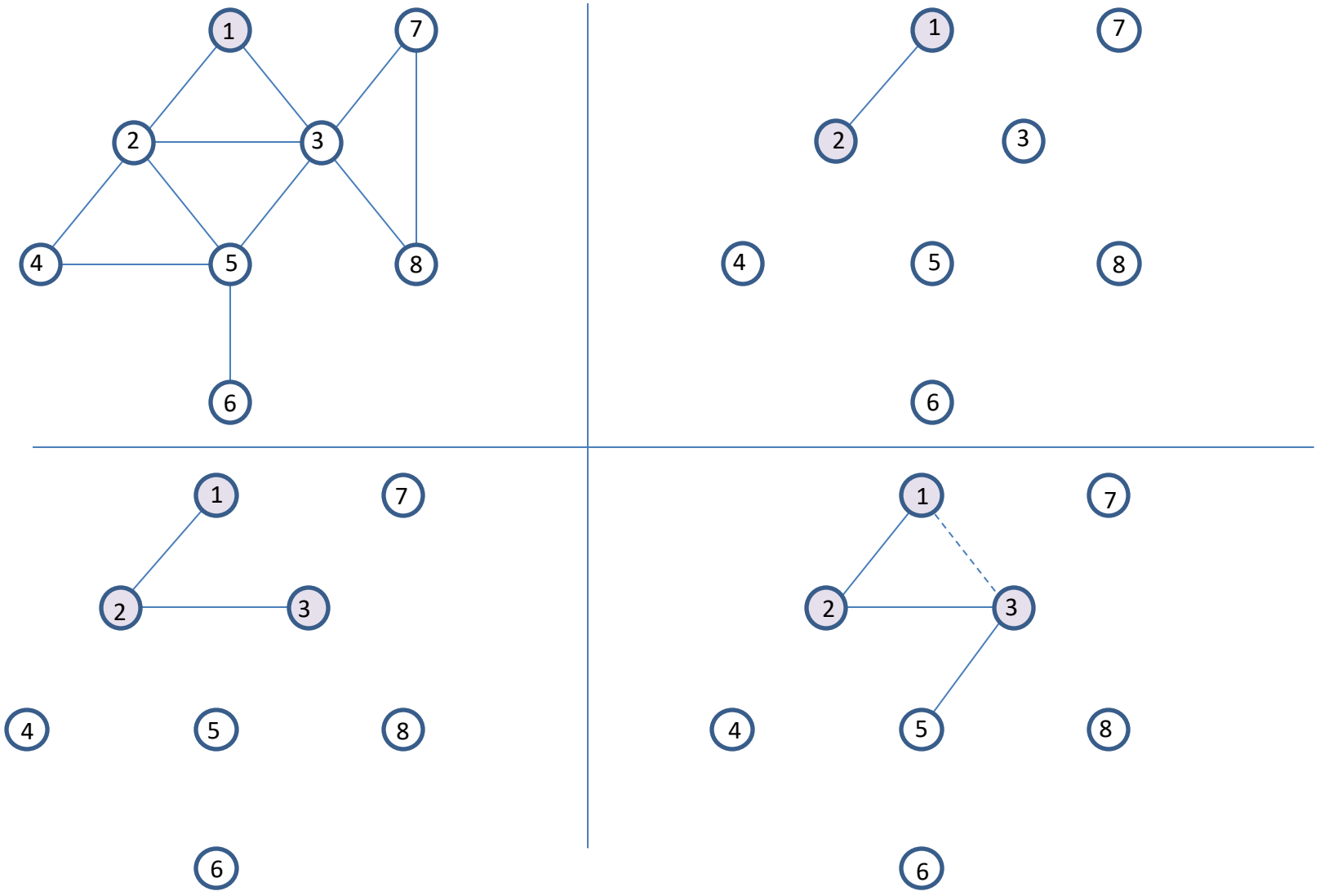
Depth-First Search (DFS)

```
DFS(G,u)
   discovered[u] := TRUE
   for each edge uv leaving u
      if discovered[v] = FALSE then
         parent[v] := u
         DFS(G, v)
      endif
   endfor
   finished[u] := TRUE
```

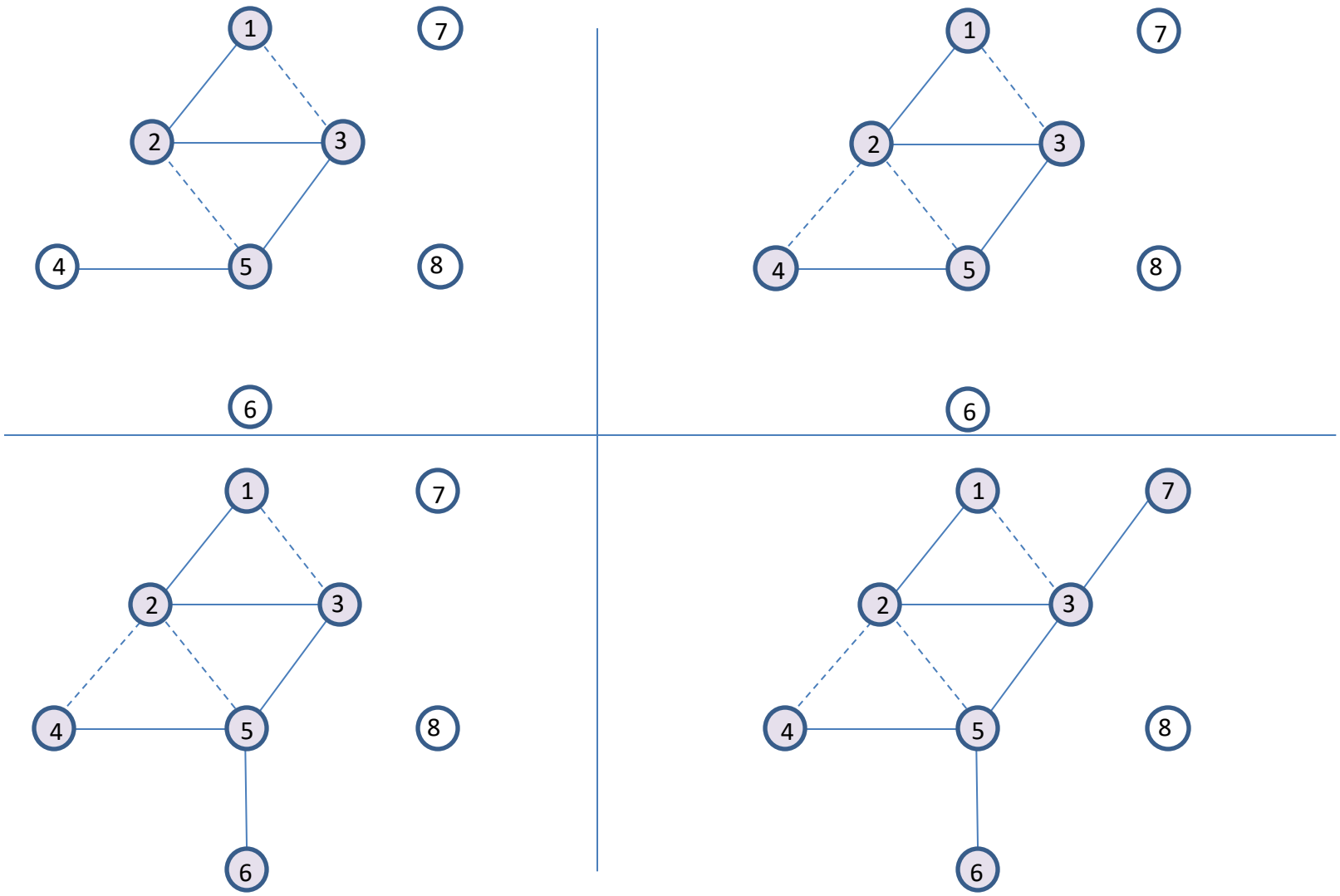
```
DFSInitial(G,s)
   for all v
      discovered[v] := FALSE
      finished[v] := FALSE
      DFS(G,s)
   endfor
```

O(m+n) complexity

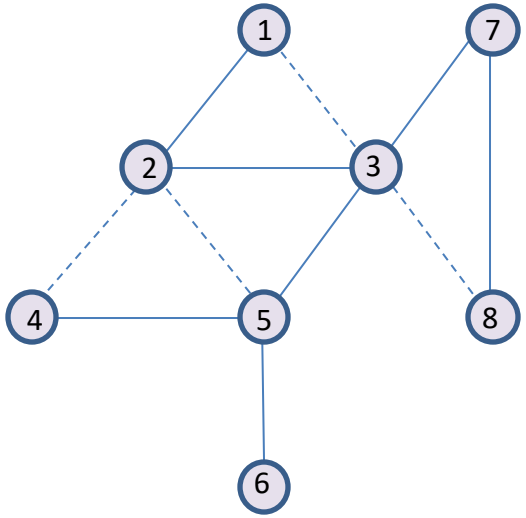
DFS example I



DFS example II



DFS example III



DFS tree

Proposition: In a DFS tree, every back edge (non-tree edge) joins a node to one of its descendants.

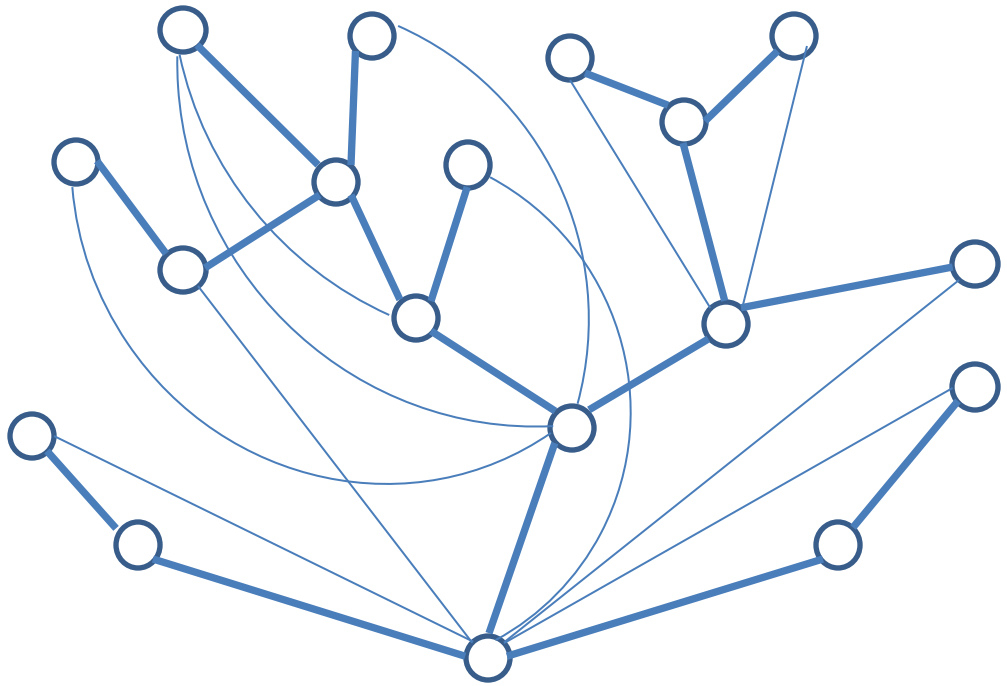
Proof: Let uv be a back edge.

For a recursive call to DFS(G,u), all nodes that are marked discovered between the times when u is discovered and when u is finished are the descendants of u in the DFS tree.

Suppose that u is discovered first by DFS: v is marked as discovered after u, while u is not finished; thus, v is a descendant of u.

When uv is examined in DFS(G, u), it is not added to the tree because v is marked as discovered.

Example of DFS tree



Bridges and articulation points

Definition: A bridge is an edge that separates a connected graph.

Definition: An articulation point is a node that separates a connected graph

1. *If all vertices of a graph have even degree, then it does not have a bridge;*
2. *The two vertices of a bridge are articulation points unless they have degree one;*
3. *There are graphs such that there is an edge that is not a bridge, but both its endpoints are articulation points.*

Articulation points

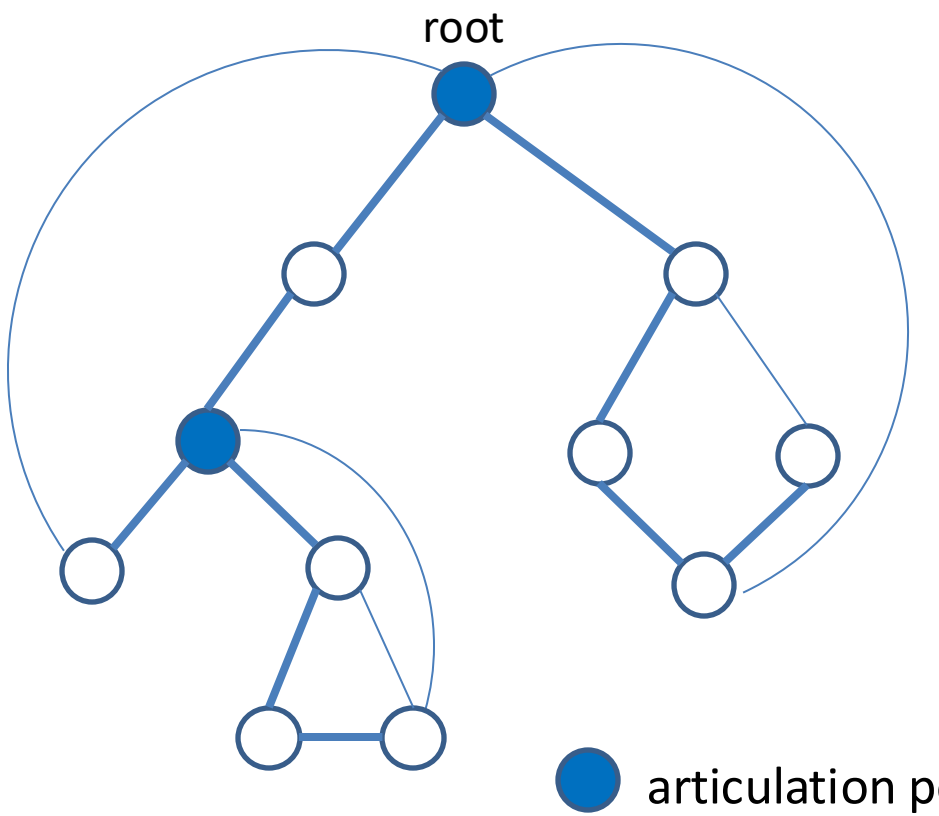
Definition: A node v in a connected graph is an articulation point if the graph G – v, obtained by removing v, is not connected.

Definition: A graph is biconnected if it is connected and does not have articulation points.

Proposition: The following are equivalent characterizations of biconnected graphs:

1. Every two nodes belong to a common cycle;
2. Given two nodes and an edge there is a path between the two nodes that contains the edge;
3. Given three vertices, there is a path between two of them that does not include the third.

Articulation points via DFS tree



- Root is an articulation point if and only if it has more than one child.
- Other than the root, a node is an articulation point if and only if there is a sub-tree rooted at a child of the node without a back edge to a proper ancestor of the node.

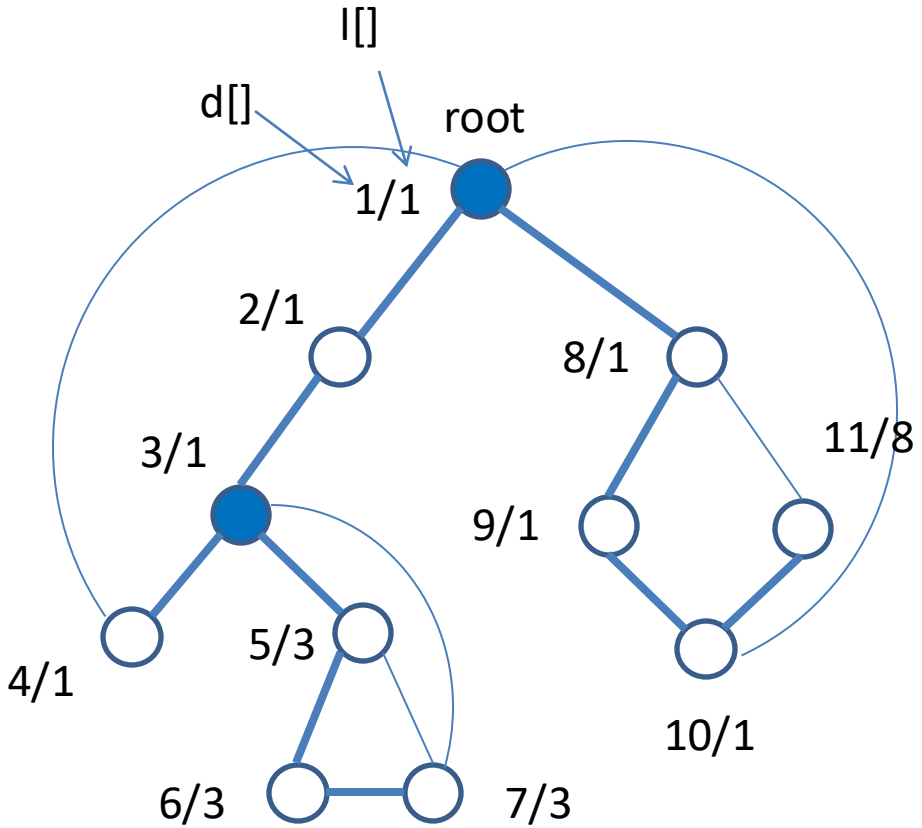
Leaves are never articulation points

Variables

It is not efficient to keep track of all back edges emanating from the sub-tree rooted at a node u; just keep track of the back edge ending at the node with the lowest discovery time.

Variables

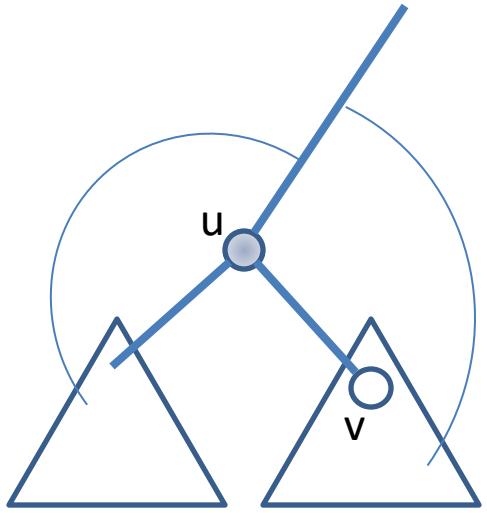
d[u] – **discovery time** of u
l[u] – **low point** of u: minimum of d[u] and all d[w] for which there is a back edge vw such that v is a descendent of u (including u).



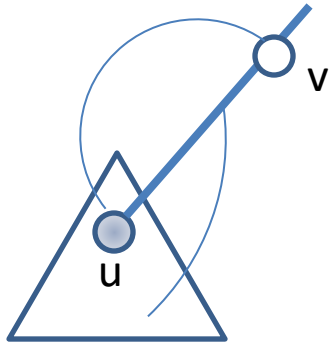
Update variables and test

Initialization: l[u] := d[u]

v is a child of u: l[u] := min{l[u], l[v]}



uv is a back edge: l[u] := min{l[u], d[v]}



Condition for u to be an articulation point after child v is analyzed: l[v] ≥ d[u]

Algorithm for articulation points

```
artPoints(G, u)
  discovered [u] := TRUE; time := time +1; d[u] := time; l[u] := d[u]
  for each edge uv
    if discovered[v] = FALSE
      if u is root and v is it second child
        u is an articulation point
      endif
      pred[v] := u
      artPoints(G, v)
      if l[v] ≥ d[u]
        u is an articulation point
      endif
      l[u] := min{l[u], l[v]}
    else if v ≠ pred[u]
      l[u] := min{l[u], d[v]}
    endif
  endfor
```

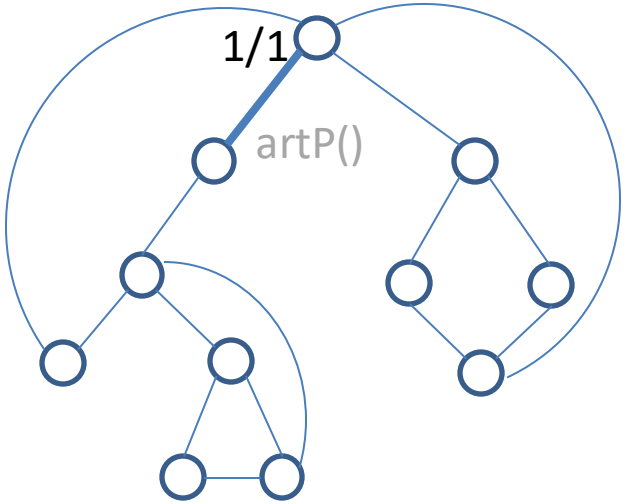
when artPoints(v) is completed, l[v] is the low point of v

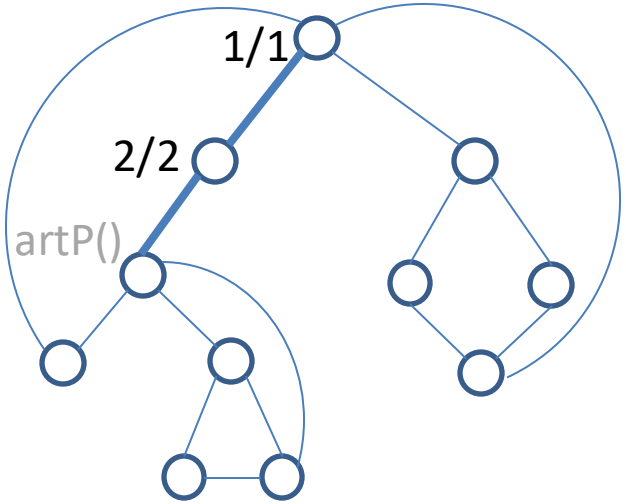
update l[u] if child v can reach an older ancestor

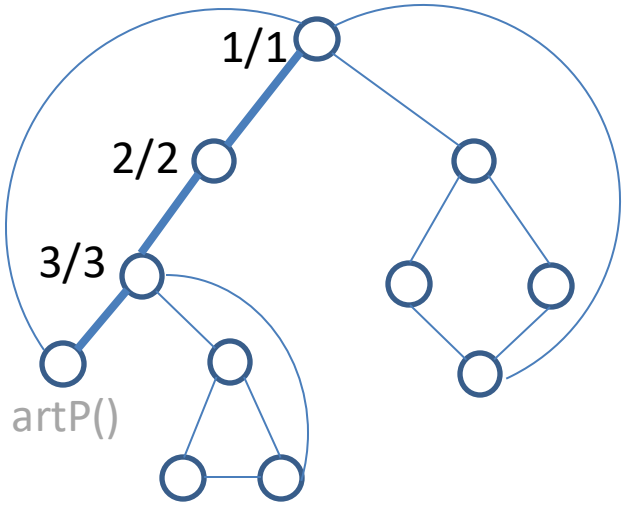
update l[u] if uv is a back edge

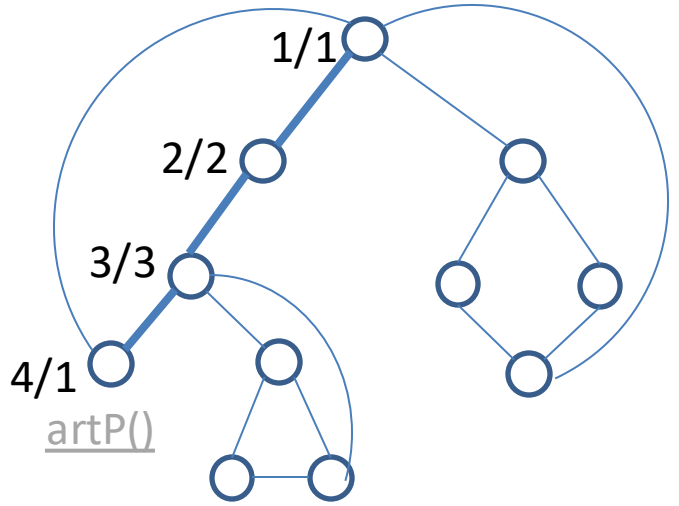
O(m) complexity

Articulation points: example I

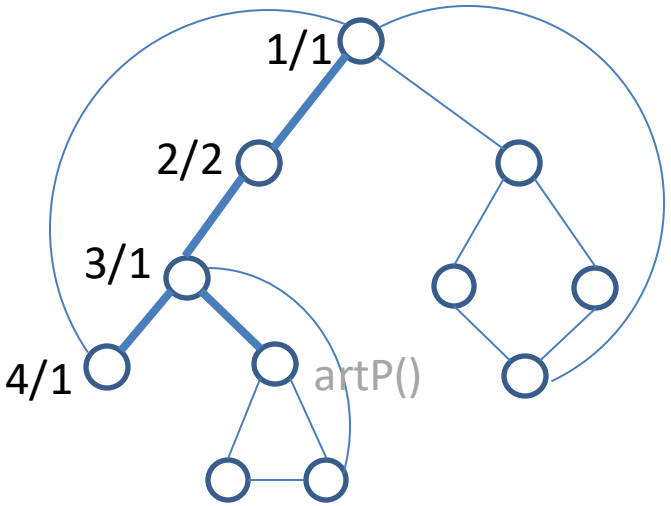


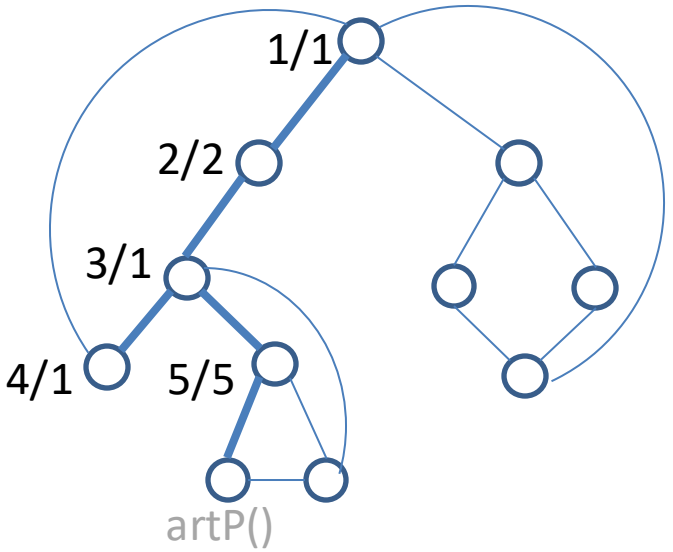


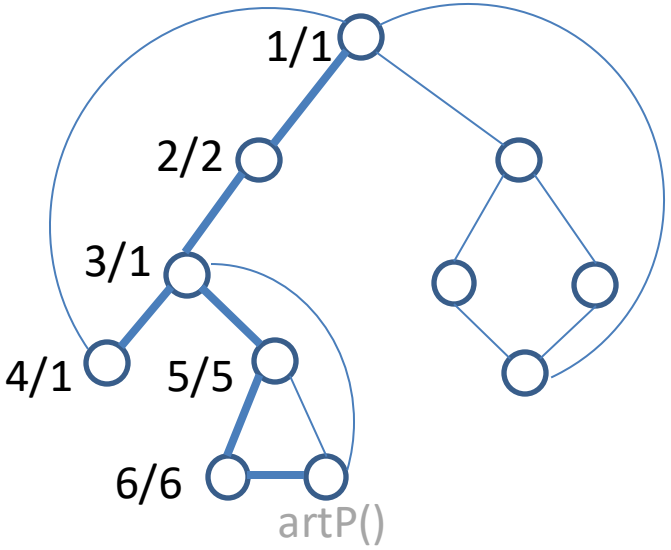


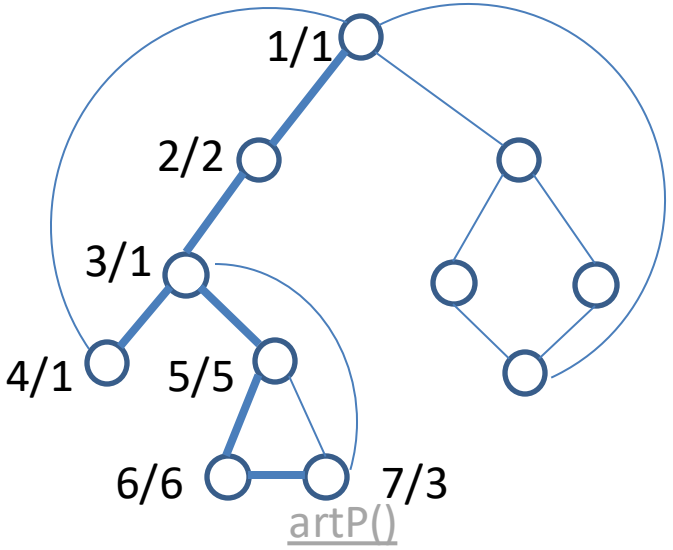


Articulation points: example II

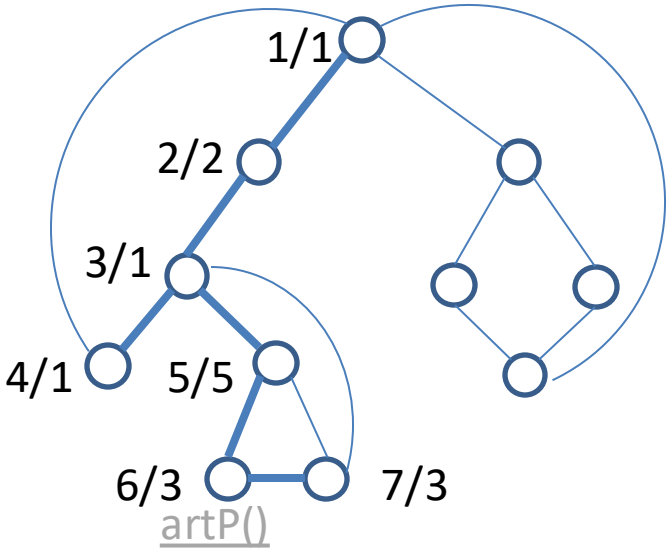


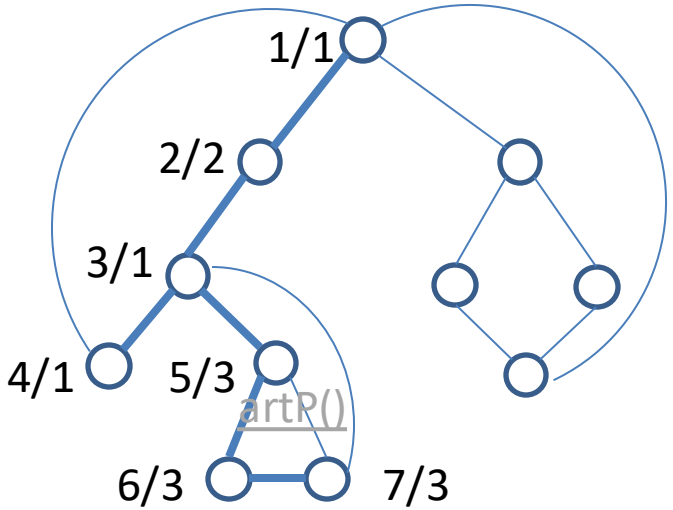


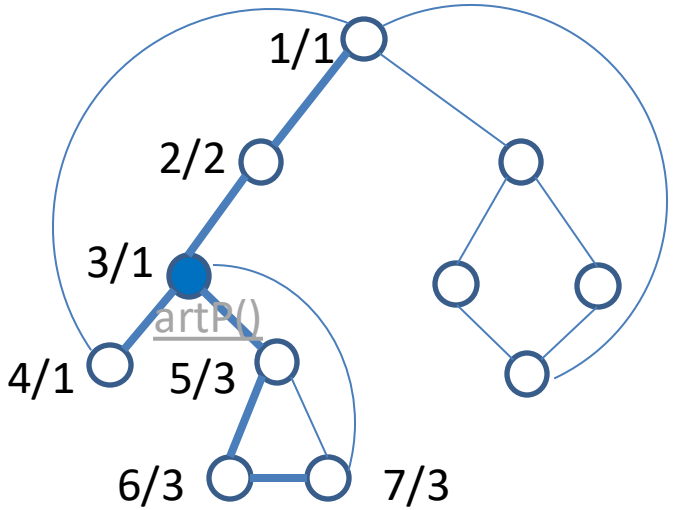


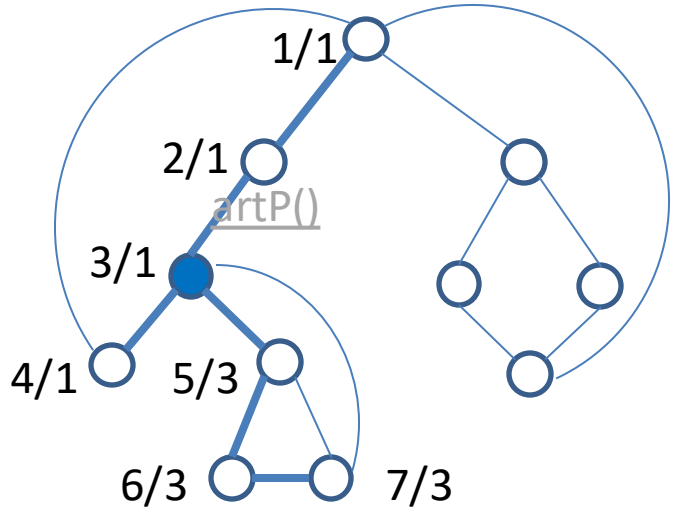


Articulation points: example III

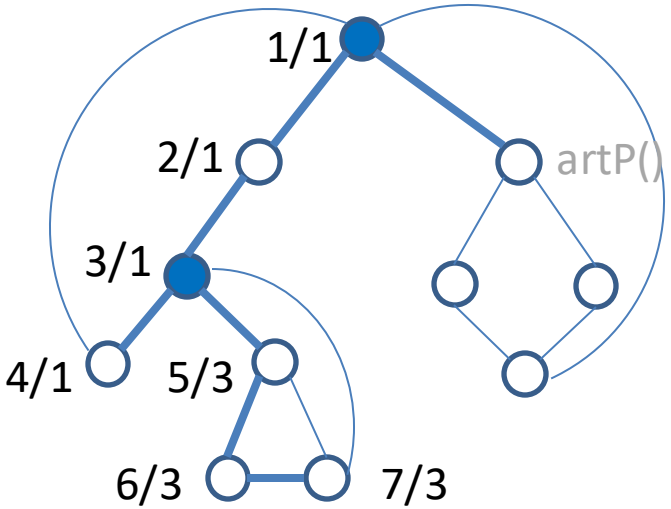


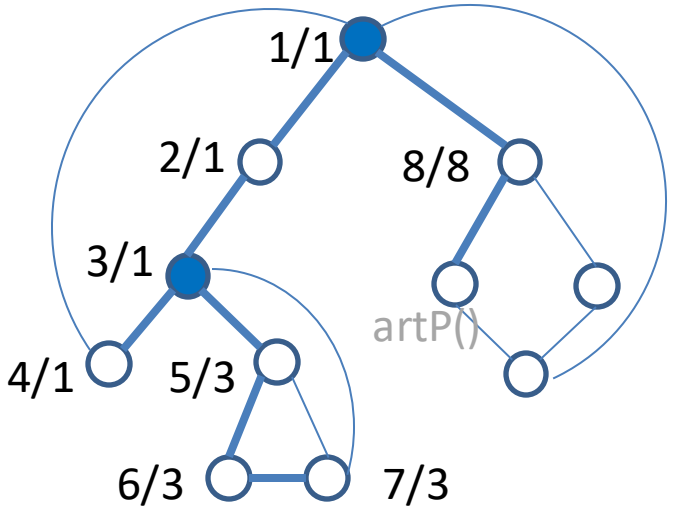


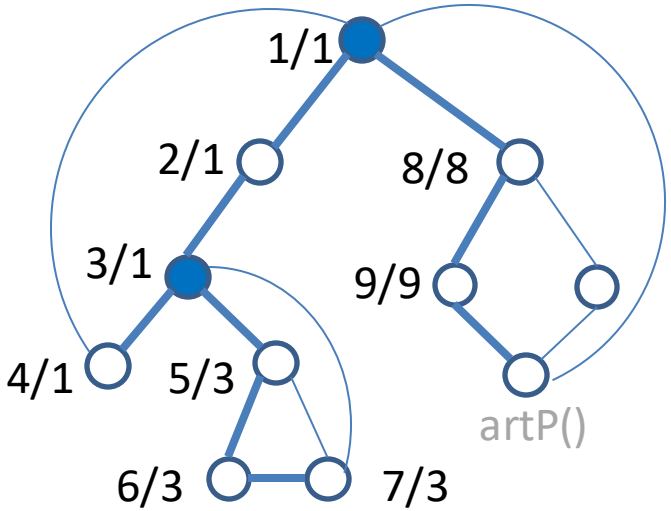


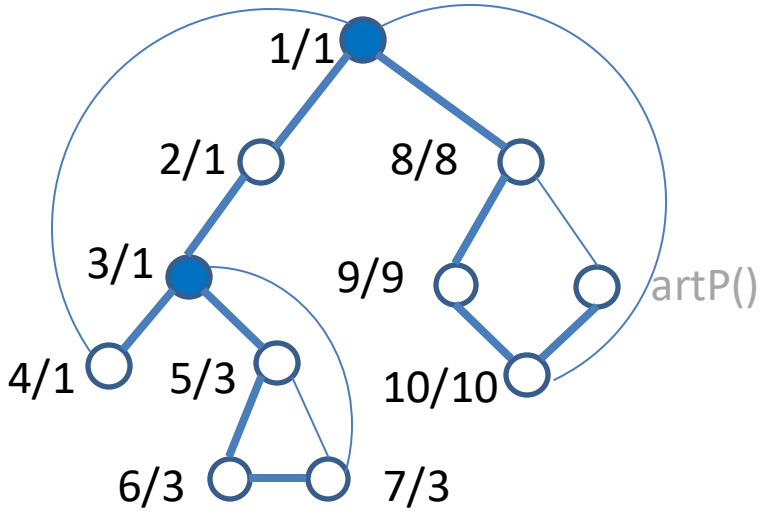


Articulation points: example IV

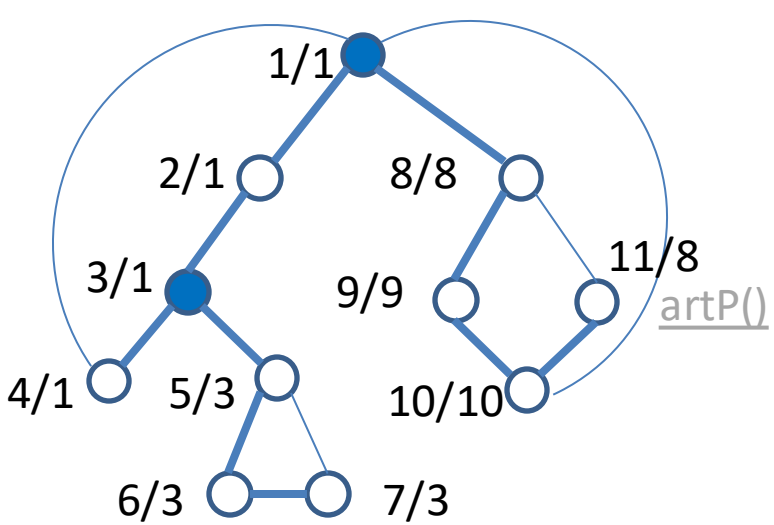


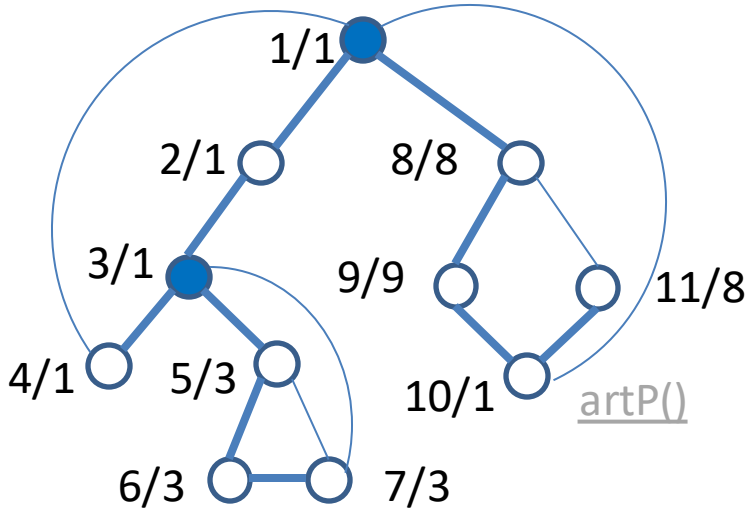


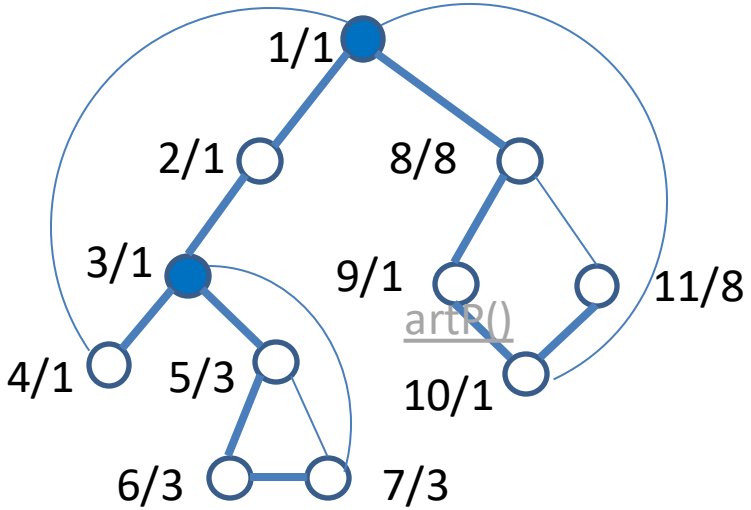


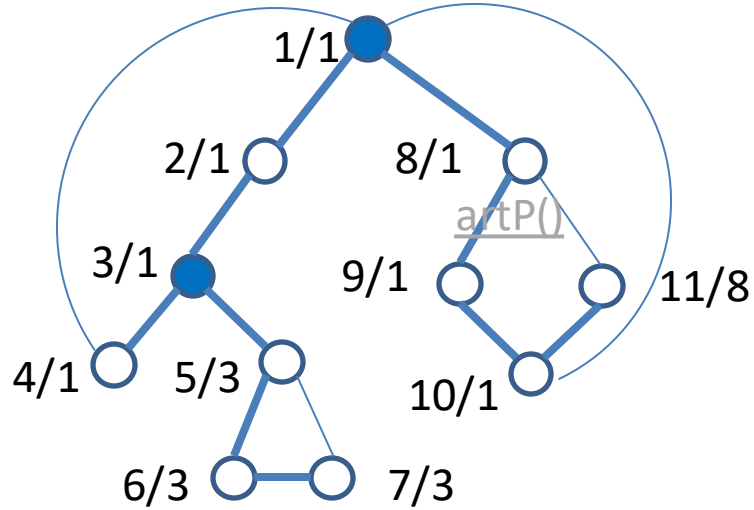


Articulation points: example V



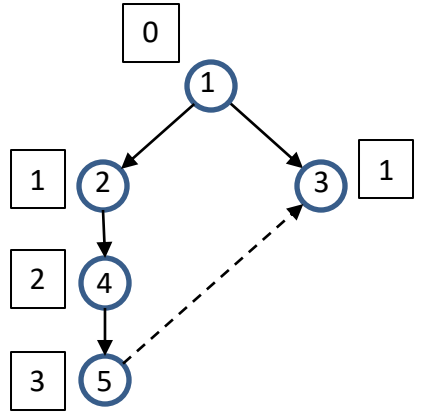


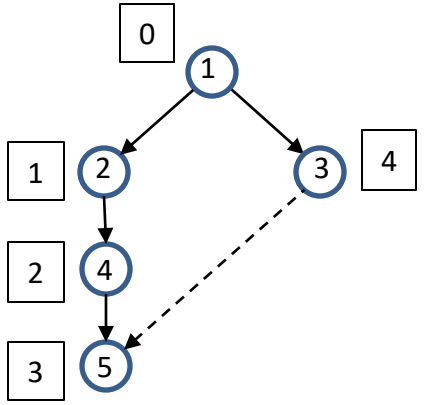




BFS and DFS in digraphs

- BFS
  - for link uv, the layer of v may be smaller than that of u by more than one
- DFS
  - for link uv not in the out-branching, neither u nor v needs be an ancestor of the other





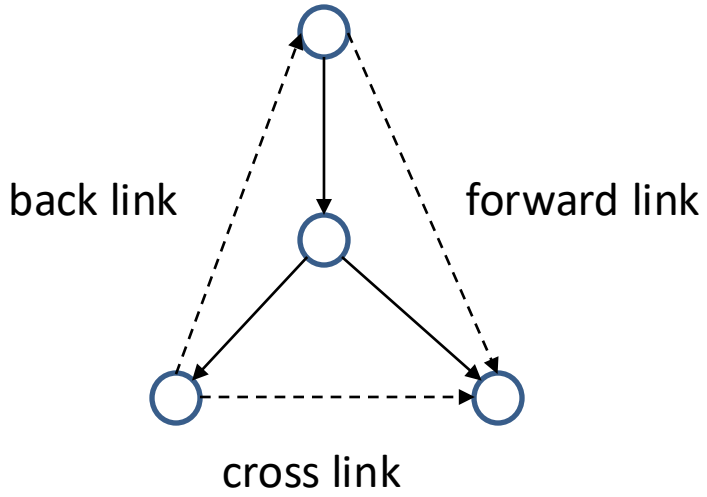
DFS with start and finishing times

```
DFS-Global(G)
  for each v
    discovered[v] := FALSE
  c := 0
  for each v
      if discovered[v] = FALSE
        DFS(G, v)
```

```
DFS(G, u)
  c := c + 1
  discovered[u] := TRUE; s[u] := c
  for each link uv leaving u
     if discovered[v] = FALSE then
        DFS(G, v)
  c := c + 1
  finished[u] := TRUE; f[u] := c
```

Classification of DFS links

- **Out-branching link uv**: from a node to its child (s[u]<s[v]<f[v]<f[u])
- **Forward link uv**: from a node to a no-child descendent in the out-branching (s[u]<s[v]<f[v]<f[u])
- **Back link uv**: from a node to an ancestor in the out-branching (s[v]<s[u]<f[u]<f[v]); reveals a cycle
- **Cross link uv**: from a node to another that is neither a descendent nor an ancestor (s[v]<f[v]<s[u]<f[u])



s[] – discovery/starting time; f[] – finishing time

Topological ordering and DAGs

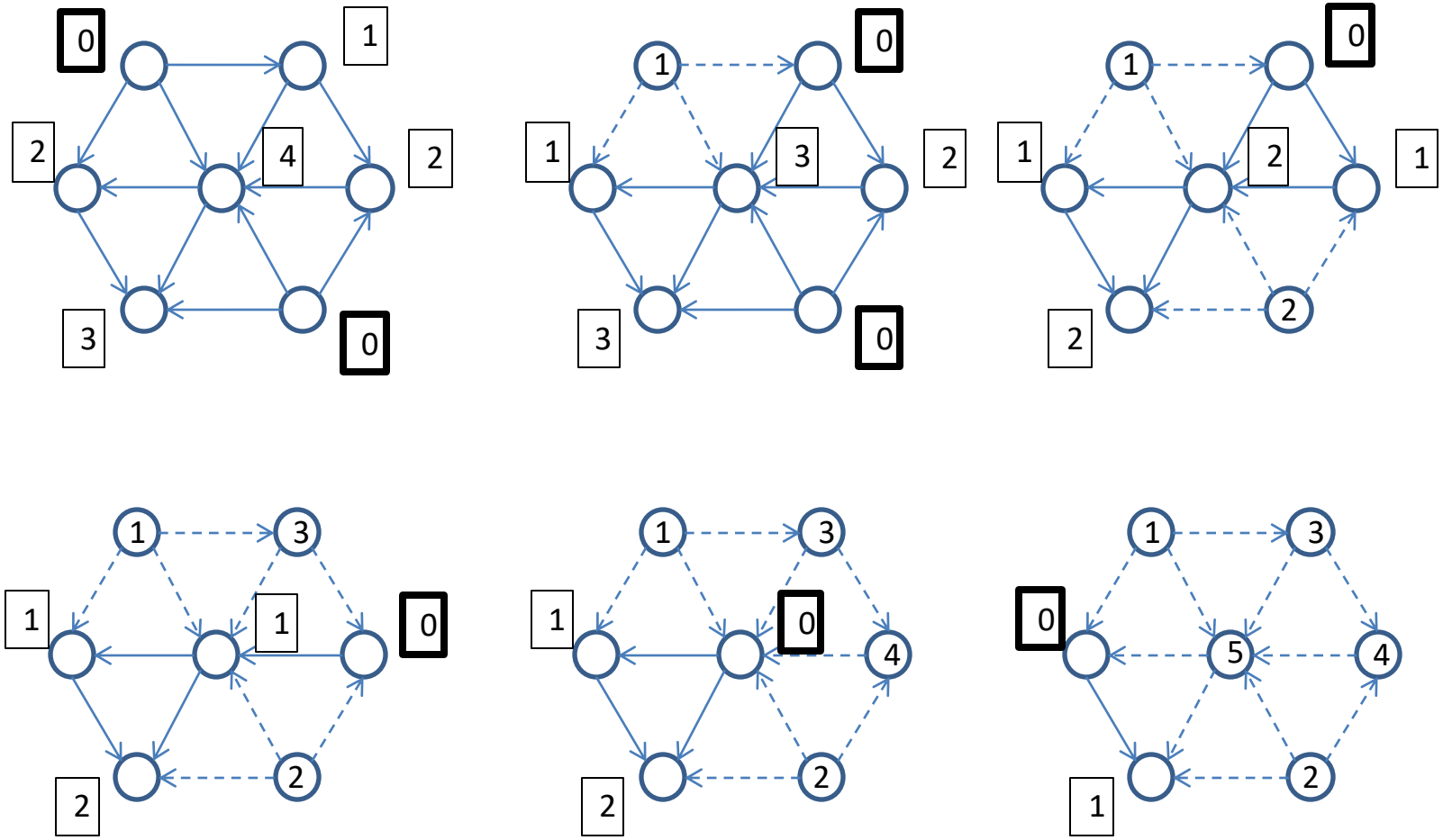
- Topological ordering: function g: V -> |V| such that for every link uv, g(u) < g(v)
- A digraph has a topological ordering if and only if it is a DAG
- A DAG can have more than one topological ordering
- A DAG has a node with zero in-degree

Algorithm for topological ordering

```
TopoOrder(G)
for all u
  g := 0; g[u] := 0; S := ∅
  in[u] := number of incoming links of u
  if in[u] = 0
    S := S ∪ {u}
while S is not empty
  take u from S
  g := g+1
  g[u] := g
  for each out-neighbor v of u
    in[v] := in[v]-1
    if in[v] = 0
      S := S ∪ {v}
if g[u] ≠ 0 for all  u
  nodes are topologically ordered
```

O(m+n) complexity

Topological ordering: example

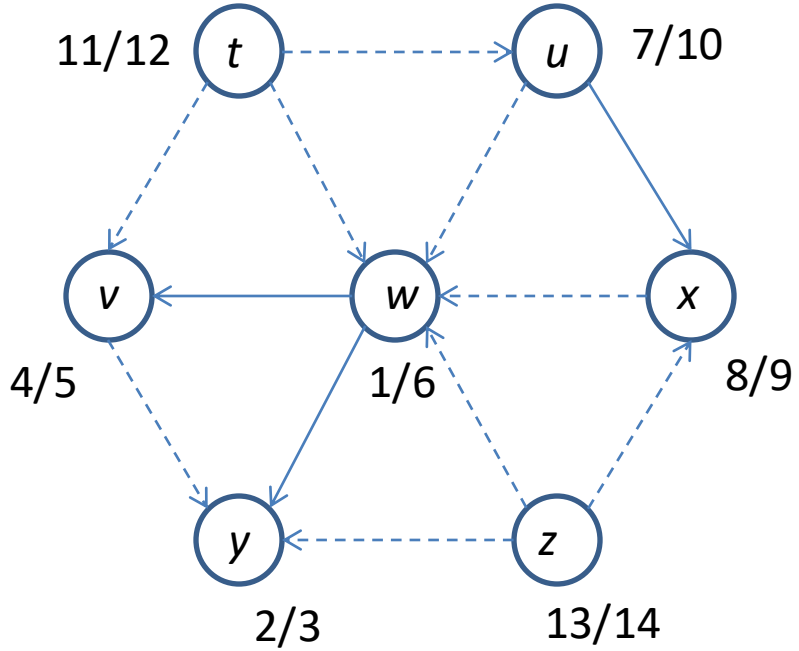


Topological ordering with DFS

call DFS-Global
each time a node is finished insert it in a stack
if there is no back link, then
    the stack contains the topological ordering
else
    there is no topological ordering

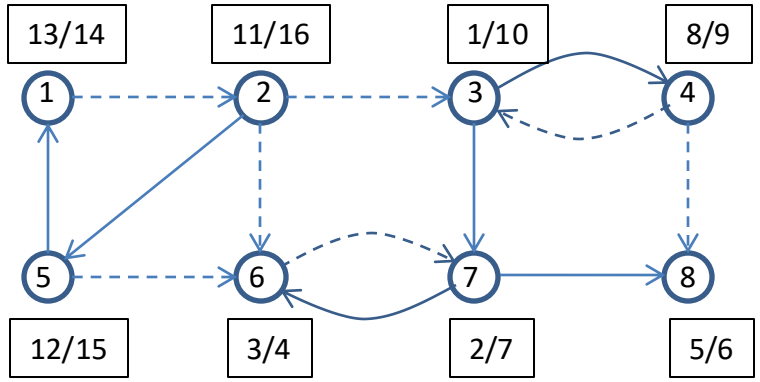
The finishing times provide a reverse topological order

Topological ordering: DFS example



Topological ordering: *z-t-u-x-w-v-y*

Strongly connected components



DFS-Global
reverse the links of the digraph
DFS-Global but taking nodes in decreasing order of finishing times

O(m+n) complexity