Graph Algorithms

Graph Traversals and Connectivity III

Michael Lampis

September 25, 2025

Graph Traversal

Problem

Given (di)graph G, determine connectivity properties:

- Is G (strongly) connected?
- What are the (strongly) connected components of G?
- Which vertices can be reached from a given source s?
- What is the shortest path distance from (given vertex) s to (given vertex) t?

Graph Traversal

Problem

Given (di)graph G, determine connectivity properties:

- Is G (strongly) connected?
- What are the (strongly) connected components of G?
- Which vertices can be reached from a given source s?
- What is the shortest path distance from (given vertex) s to (given vertex) t?

Algorithms:

- BFS (two lectures ago)
- DFS (last lecture)

Graph Traversal

Problem

Given (di)graph G, determine connectivity properties:

- Is G (strongly) connected?
- What are the (strongly) connected components of G?
- Which vertices can be reached from a given source s?
- What is the shortest path distance from (given vertex) s to (given vertex) t?

Algorithms:

- BFS (two lectures ago)
- DFS (last lecture)
- Today: Applications

Topological Sort

Definition of Topological Sort

Definition

A topological sort of a digraph $G = (V, A)$ is an ordering (numbering) of the vertices with the following property: if we have an arc from a vertex numbered i to a vertex numbered j, then $i < j$.

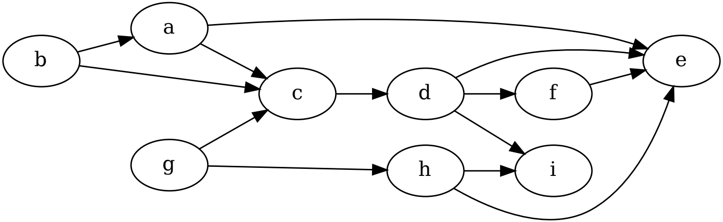
Definition of Topological Sort

Definition

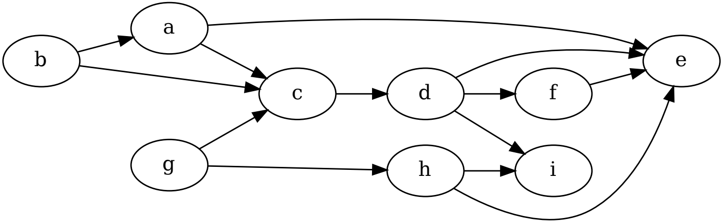
A topological sort of a digraph $G = (V, A)$ is an ordering (numbering) of the vertices with the following property: if we have an arc from a vertex numbered i to a vertex numbered j, then $i < j$.

- In other words, arcs go from lower to higher numbers.
- Vertices are numbered $1, \ldots, n$.

Topological Sort – Example



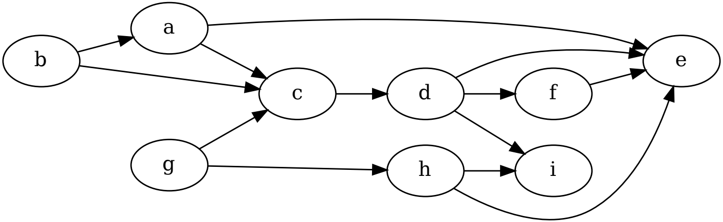
Topological Sort – Example



Ranking:

1	2	3	4	5	6	7	8	9
b	a	g	c	d	h	i	f	e

Topological Sort – Example



Ranking:

1	2	3	4	5	6	7	8	9
b	a	g	c	d	h	i	f	e
g	b	a	c	h	d	f	e	i

DAGs are topologically sortable

Definition

A digraph G is a Directed Acyclic Graph (DAG) if G contains no directed cycles.

NB: We also count cycles of length 2 (digons).

DAGs are topologically sortable

Definition

A digraph G is a Directed Acyclic Graph (DAG) if G contains no directed cycles.

NB: We also count cycles of length 2 (digons).

Lemma

A digraph G admits a topological ordering if and only if G is a DAG.

DAGs are topologically sortable

Definition

A digraph G is a Directed Acyclic Graph (DAG) if G contains no directed cycles.

NB: We also count cycles of length 2 (digons).

Lemma

A digraph G admits a topological ordering if and only if G is a DAG.

Proof.

- $\Rightarrow$: A cycle C cannot be topo-sorted, because all vertices have positive in- and out-degree.
- $\Leftarrow$: A DAG always contains a sink v (why?), number it n, order the rest inductively.

□

Every DAG has a source/sink

Lemma

If $G = (V, A)$ is a DAG, then G contains at least one source and at least one sink.

Every DAG has a source/sink

Lemma

If $G = (V, A)$ is a DAG, then G contains at least one source and at least one sink.

Proof.

- Let $P = x_1, x_2, \ldots, x_k$ be the **longest** directed simple path in G.

Every DAG has a source/sink

Lemma

If $G = (V, A)$ is a DAG, then G contains at least one source and at least one sink.

Proof.

- Let $P = x_1, x_2, \dots, x_k$ be the **longest** directed simple path in G.
- If there exists an arc $x_k y$ with $y \in \{x_1, \dots, x_{k-1}\}$, then G is not a DAG, contradiction.

Every DAG has a source/sink

Lemma

If $G = (V, A)$ is a DAG, then G contains at least one source and at least one sink.

Proof.

- Let $P = x_1, x_2, \ldots, x_k$ be the **longest** directed simple path in G.
- If there exists an arc $x_k y$ with $y \in \{x_1, \ldots, x_{k-1}\}$, then G is not a DAG, contradiction.
- If there exists an arc $x_k y$ with $y \notin \{x_1, \ldots, x_{k-1}\}$, then P is not longest, contradiction.

Every DAG has a source/sink

Lemma

If $G = (V, A)$ is a DAG, then G contains at least one source and at least one sink.

Proof.

- Let $P = x_1, x_2, \ldots, x_k$ be the **longest** directed simple path in G.
- If there exists an arc $x_k y$ with $y \in \{x_1, \ldots, x_{k-1}\}$, then G is not a DAG, contradiction.
- If there exists an arc $x_k y$ with $y \notin \{x_1, \ldots, x_{k-1}\}$, then P is not longest, contradiction.
- $\Rightarrow x_k$ is a sink (out-degree 0)
- Symmetric reasoning shows that x_1 is a source.

□

Straightforward Algorithm (for Matrices)

procedure Topo-Sort(G)
  for $i = 1$ to n **do**
    Find a source in $G \to v$
    Number of $v \leftarrow i$
    $G \leftarrow G - v$
  end for
  Output Numbers of $v \in V$
end procedure

Straightforward Algorithm (for Matrices)

procedure TOPO-SORT(G)
    for $i = 1$ to n **do**
        Find a source in $G \to v$ ▷ How?
        Number of $v \leftarrow i$ ▷ Num[v] $\leftarrow i$
        $G \leftarrow G - v$ ▷ How?
    end for
    Output Numbers of $v \in V$
end procedure

Straightforward Algorithm (for Matrices)

```
Active[v] ← 1 for all v ∈ V
procedure TOPO-SORT(G)
    for i = 1 to n do
        v ← Find-source(G,Active)
        Num[v] ← i
        Active[v] ← 0
    end for
    Output Numbers of v ∈ V
end procedure
procedure FIND-SOURCE(G,Active)
    for v ∈ V do
        if Active[v] == 1 and d⁻(v) == 0 then
            Return v
        end if
    end for
end procedure
```

Straightforward Algorithm (for Matrices)

```
Active[v] ← 1 for all v ∈ V
procedure TOPO-SORT(G)
    for i = 1 to n do
        v ← Find-source(G,Active)
        Num[v] ← i
        Active[v] ← 0
    end for
    Output Numbers of v ∈ V
end procedure
procedure FIND-SOURCE(G,Active)
    for v ∈ V do
        if Active[v] == 1 and d⁻(v) == 0 then        ▷ How?
            Return v
        end if
    end for
end procedure
```

Straightforward Algorithm (for Matrices)

```
procedure CHECK-IF-SOURCE(G,Active,v)
    for u ∈ V do
        if A[u, v] == 1 and Active[u] then
            Return No
        end if
    end for
    Return Yes
end procedure
```

Straightforward Algorithm (for Matrices)

procedure CHECK-IF-SOURCE(G,Active,v) $\triangleright$ $O(n)$ time
    for $u \in V$ **do**
        if $A[u, v] == 1$ and Active[u] **then**
            Return No
        end if
    end for
    Return Yes
end procedure

Straightforward Algorithm (for Matrices)

Active[v] $\leftarrow 1$ for all $v \in V$
procedure TOPO-SORT(G)
    for $i = 1$ to n **do**
        $v \leftarrow$ Find-source(G,Active)
        Num[v] $\leftarrow i$
        Active[v] $\leftarrow 0$
    end for
    Output Numbers of $v \in V$
end procedure
procedure FIND-SOURCE(G,Active) $\triangleright$ $O(n^2)$ time
    for $v \in V$ **do** $\triangleright$ $O(n)$ iterations
        if Active[v] $==1$ and $d^-(v) == 0$ **then** $\triangleright$ $O(n)$ time
            Return v
        end if
    end for
end procedure

Straightforward Algorithm (for Matrices)

```
Active[v] ← 1 for all v ∈ V
procedure Topo-Sort(G)                                  ▷ O(n^3) time
    for i = 1 to n do                                   ▷ O(n) iterations
        v ← Find-source(G,Active)
        Num[v] ← i
        Active[v] ← 0
    end for
    Output Numbers of v ∈ V
end procedure
procedure Find-source(G,Active)                         ▷ O(n^2) time
    for v ∈ V do                                        ▷ O(n) iterations
        if Active[v] == 1 and d^-(v) == 0 then          ▷ O(n) time
            Return v
        end if
    end for
end procedure
```

Optimality for Find-Source

Lemma

Find-Source cannot be solved in $o(n^2)$ time (for adjacency matrices).

Optimality for Find-Source

Lemma

Find-Source cannot be solved in $o(n^2)$ time (for adjacency matrices).

Proof.

- Intuition: If an algorithm takes $< \binom{n}{2} - 1$ steps, there exists pair i, j for which neither $A[i, j]$ nor $A[j, i]$ was consulted, therefore impossible to know which of i, j are sources.
- Adversary argument: as long as possible, reply to an algorithm's queries by saying that a vertex has no incoming arcs, until the last step.

□

Optimality for Find-Source

Lemma

Find-Source cannot be solved in $o(n^2)$ time (for adjacency matrices).

Proof.

- Intuition: If an algorithm takes $< \binom{n}{2} - 1$ steps, there exists pair i, j for which neither $A[i, j]$ nor $A[j, i]$ was consulted, therefore impossible to know which of i, j are sources.
- Adversary argument: as long as possible, reply to an algorithm's queries by saying that a vertex has no incoming arcs, until the last step.

□

Caution!

- This does not imply that our topological sorting algorithm is optimal!

Topological sort in linear time

```
Initialize                          ▷ DFS Initialization as before
i ← n                               ▷ Next vertex to be added to list
for v ∈ V do
    if v is White then
        DFS-Visit(G,v)
    end if
end for
procedure DFS-Visit(G,u)
    ...                             ▷ DFS as before
    u.f = t, Color u Black          ▷ When u turns black, append.
    Num[u] ← i, i − −
end procedure
```

Correctness Analysis

Lemma

G is a DAG $\Leftrightarrow$ DFS produces no backward arcs.

Correctness Analysis

Lemma

G is a DAG $\Leftrightarrow$ DFS produces no backward arcs.

Theorem

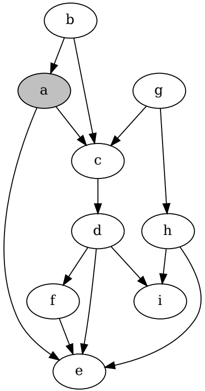
Previous algorithm is correct.

Proof.

- Consider arc uv at time when u became Gray:
  - If v was White $\Rightarrow$ v will be descendant of u $\Rightarrow$ v will become Black first $\Rightarrow$ v will be assigned higher number.
  - If v was Black $\Rightarrow$ v will be assigned higher number.
  - If v was Gray $\Rightarrow$ uv is a backward arc, contradiction!

□

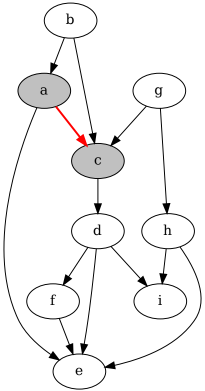
Example



Times:

	d	f
a	1	
b		
c		
d		
e		
f		
g		
h		
i		

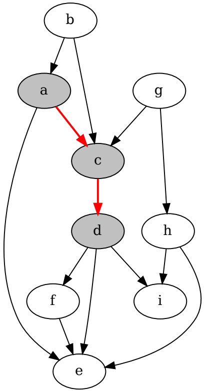
Example



Times:

	d	f
a	1	
b		
c	2	
d		
e		
f		
g		
h		
i		

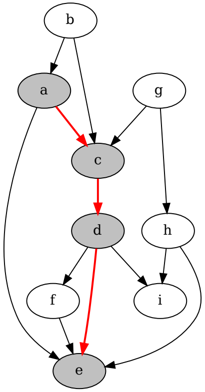
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e		
f		
g		
h		
i		

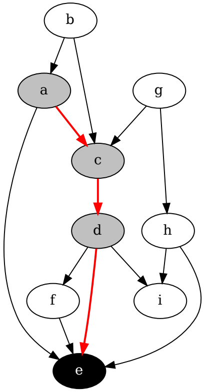
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	
f		
g		
h		
i		

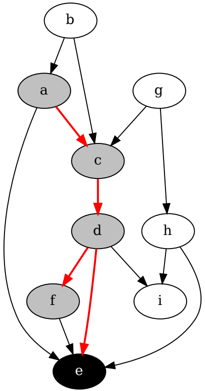
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f		
g		
h		
i		

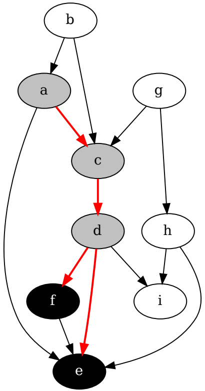
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f	6	
g		
h		
i		

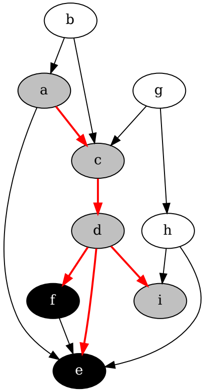
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f	6	7
g		
h		
i		

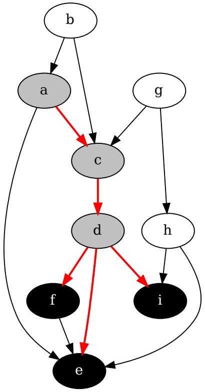
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f	6	7
g		
h		
i	8	

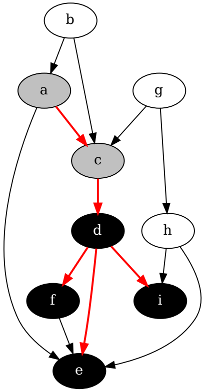
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f	6	7
g		
h		
i	8	9

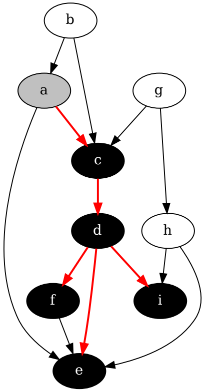
Example



Times:

	d	f
a	1	
b		
c	2	
d	3	10
e	4	5
f	6	7
g		
h		
i	8	9

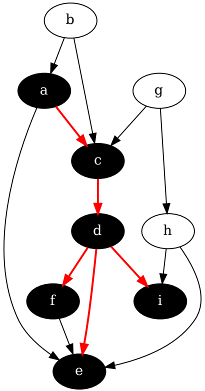
Example



Times:

	d	f
a	1	
b		
c	2	11
d	3	10
e	4	5
f	6	7
g		
h		
i	8	9

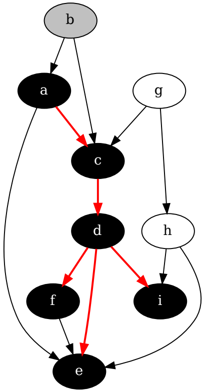
Example



Times:

	d	f
a	1	12
b		
c	2	11
d	3	10
e	4	5
f	6	7
g		
h		
i	8	9

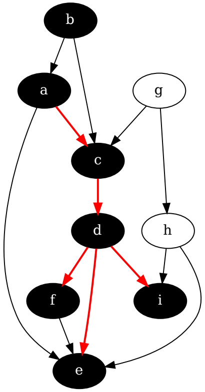
Example



Times:

	d	f
a	1	12
b	13	
c	2	11
d	3	10
e	4	5
f	6	7
g		
h		
i	8	9

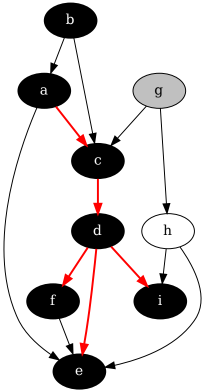
Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g		
h		
i	8	9

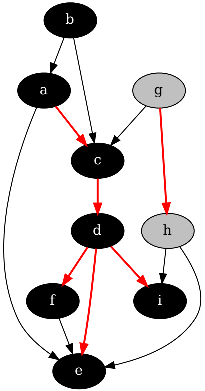
Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g	15	
h		
i	8	9

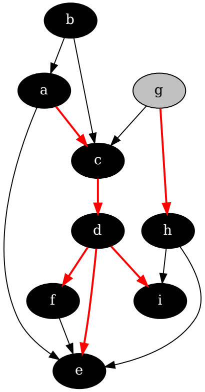
Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g	15	
h	16	
i	8	9

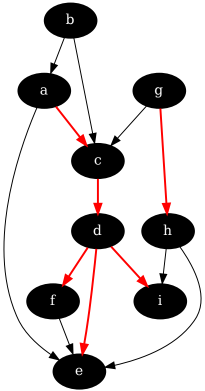
Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g	15	
h	16	17
i	8	9

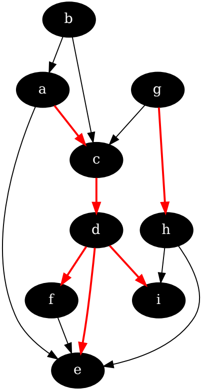
Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g	15	18
h	16	17
i	8	9

Example



Times:

	d	f
a	1	12
b	13	14
c	2	11
d	3	10
e	4	5
f	6	7
g	15	18
h	16	17
i	8	9

Ordering:

g, h, b, a, c, d, i, f, e

Strongly Connected Components

Strongly Connected Components

Definition

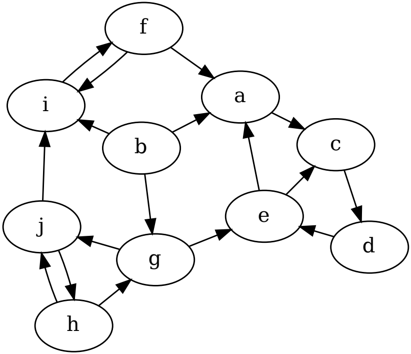
In a digraph G, a strongly connected component C is a maximal set of vertices such that for all $u, v \in C$. G contains a $u \to v$ and a $v \to u$ path.

Problem

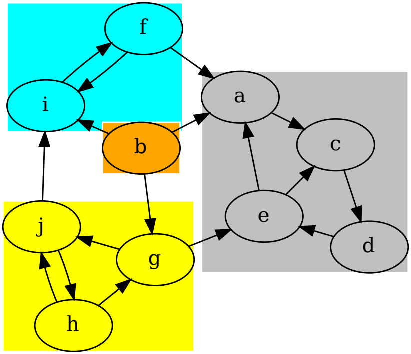
Given a digraph $G = (V, A)$, output a partition of V into strongly connected components.

NB: For example, compute an integer $cc(v)$ for each $v \in V$ such that $cc(v) = cc(u)$ if and only if v, u are in the same SCC.

SCC Example



SCC Example



Straightforward Algorithm

```
procedure SCC(G)
    cc(v) ← −1 for all v ∈ V
    cur ← 1
    for v ∈ V do
        if cc(v) == −1 then
            cc(v) ← cur
            for u ∈ V do
                if Reach(G, u, v) ∧ Reach(G, v, u) then
                    cc(u) = cc(v)
                end if
            end for
            cur + +
        end if
    end for
    Return cc
end procedure
```

Straightforward Algorithm

```
procedure SCC(G)                                  ▷ O(n^3 + n^2 m) time
    cc(v) ← −1 for all v ∈ V
    cur ← 1
    for v ∈ V do                                  ▷ O(n) iterations
        if cc(v) == −1 then
            cc(v) ← cur
            for u ∈ V do                          ▷ O(n) iterations
                if Reach(G, u, v) ∧ Reach(G, v, u) then   ▷ O(n + m)
                    cc(u) = cc(v)
                end if
            end for
            cur++
        end if
    end for
    Return cc
end procedure
```

SCC by 2-DFS algorithm

Algorithm idea:

1. Run DFS, compute a finish time for each vertex.
2. Compute G^T.
   - Reminder: G^T is G where arcs are reversed.
3. Run DFS on G^T.
   - **Important:** Consider vertices **not** in alphabetical order, but in decreasing order of finish time from first DFS.
   - (Topological sort order, if G was a DAG).
4. Each DFS tree from the previous step is a SCC of G.

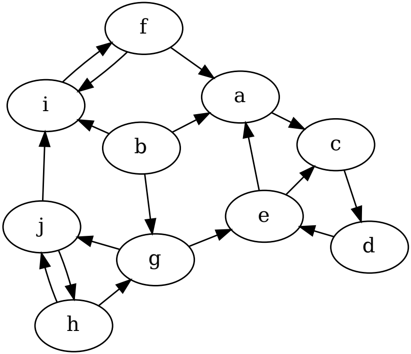
SCC by 2-DFS algorithm

Algorithm idea:

1. Run DFS, compute a finish time for each vertex.
2. Compute G^T.
   - Reminder: G^T is G where arcs are reversed.
3. Run DFS on G^T.
   - **Important:** Consider vertices **not** in alphabetical order, but in decreasing order of finish time from first DFS.
   - (Topological sort order, if G was a DAG).
4. Each DFS tree from the previous step is a SCC of G.

Sanity check: is this algorithm correct on DAGs?

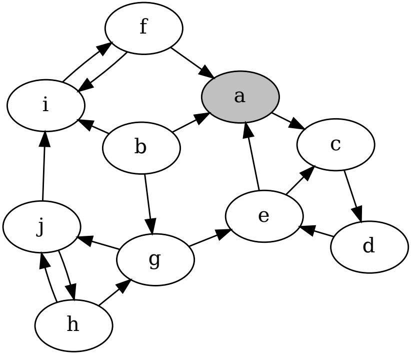
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a		
b		
c		
d		
e		
f		
g		
h		
i		
j		

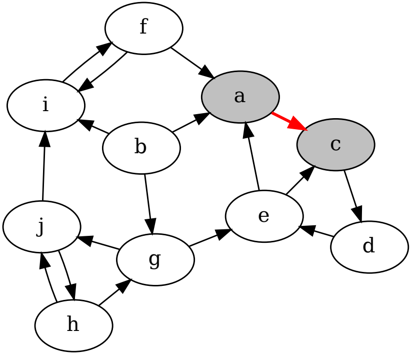
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	
b		
c		
d		
e		
f		
g		
h		
i		
j		

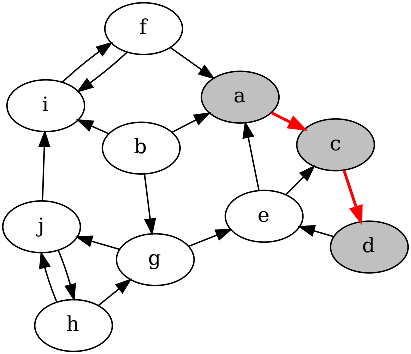
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	
b		
c	2	
d		
e		
f		
g		
h		
i		
j		

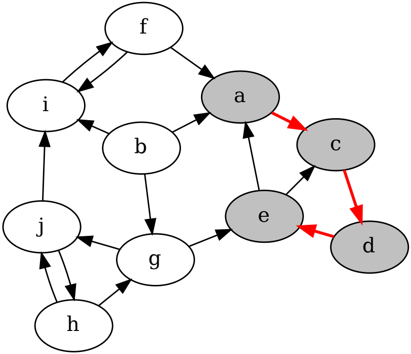
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	
b		
c	2	
d	3	
e		
f		
g		
h		
i		
j		

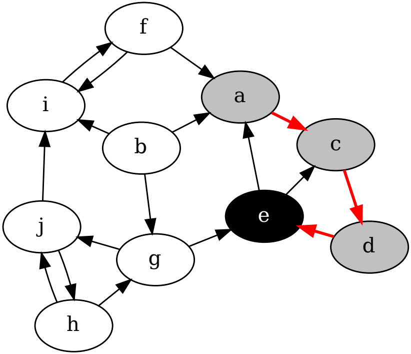
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	
b		
c	2	
d	3	
e	4	
f		
g		
h		
i		
j		

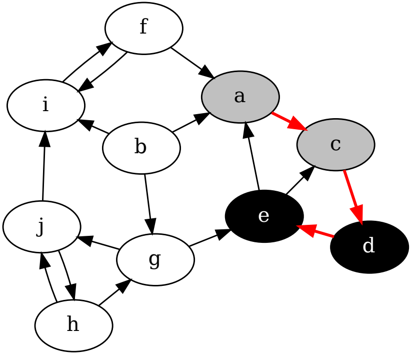
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	
b		
c	2	
d	3	
e	4	5
f		
g		
h		
i		
j		

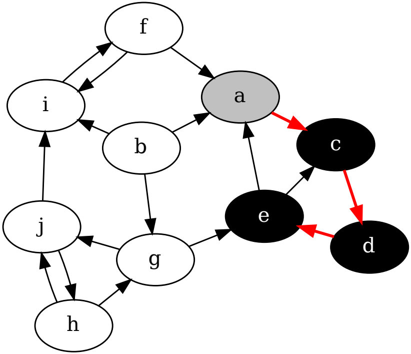
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	
b		
c	2	
d	3	6
e	4	5
f		
g		
h		
i		
j		

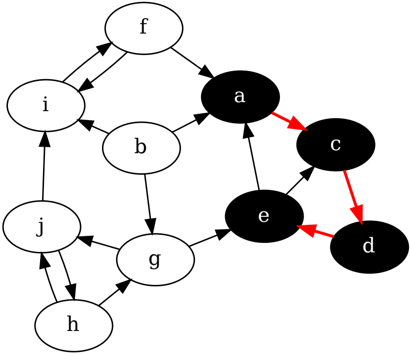
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	
b		
c	2	7
d	3	6
e	4	5
f		
g		
h		
i		
j		

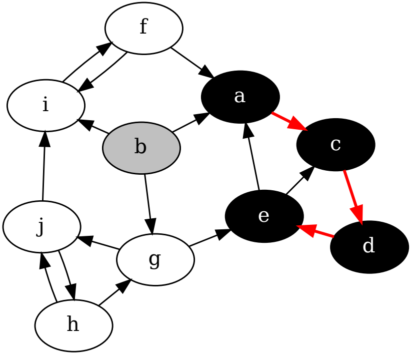
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b		
c	2	7
d	3	6
e	4	5
f		
g		
h		
i		
j		

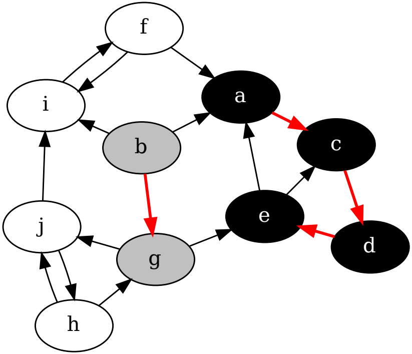
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g		
h		
i		
j		

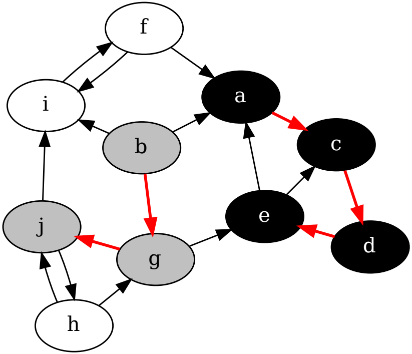
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g	10	
h		
i		
j		

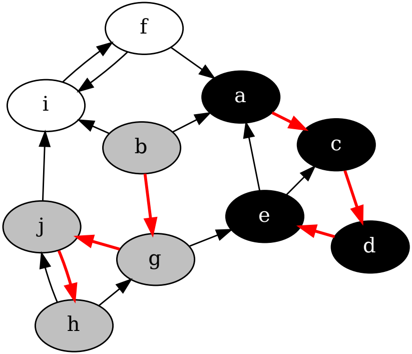
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g	10	
h		
i		
j	11	

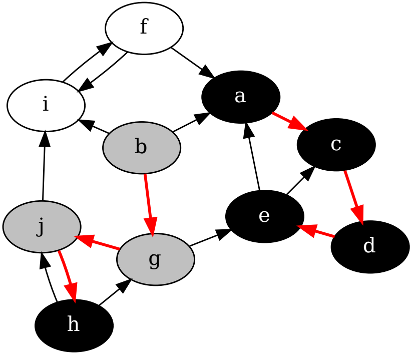
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g	10	
h	12	
i		
j	11	

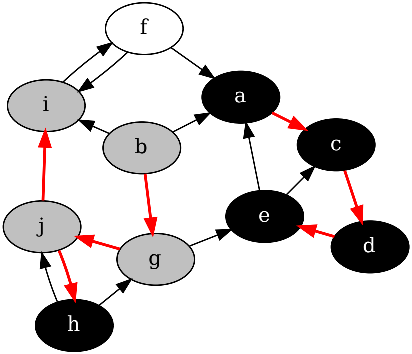
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g	10	
h	12	13
i		
j	11	

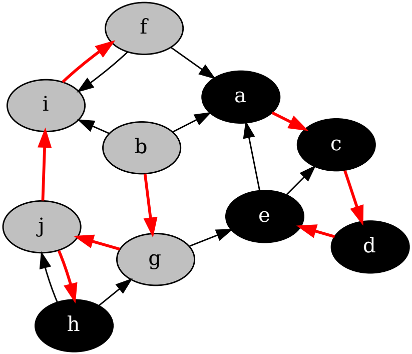
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f		
g	10	
h	12	13
i	14	
j	11	

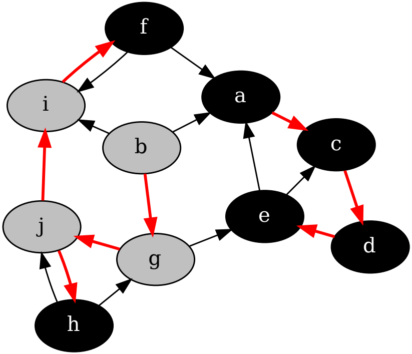
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f	15	
g	10	
h	12	13
i	14	
j	11	

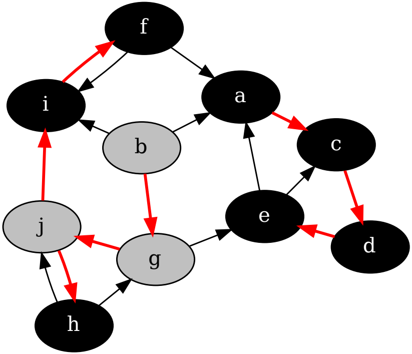
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f	15	16
g	10	
h	12	13
i	14	
j	11	

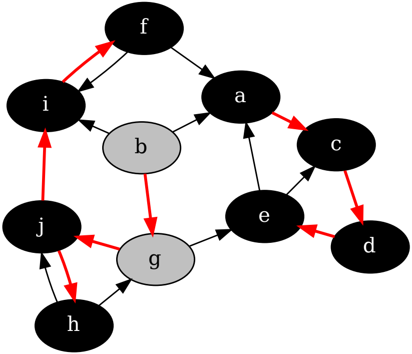
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f	15	16
g	10	
h	12	13
i	14	17
j	11	

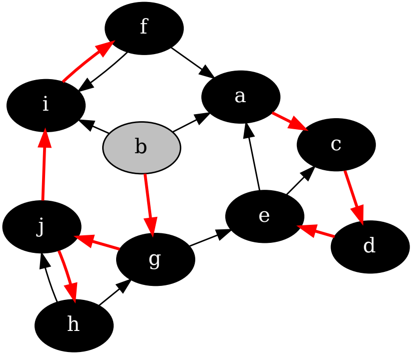
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f	15	16
g	10	
h	12	13
i	14	17
j	11	18

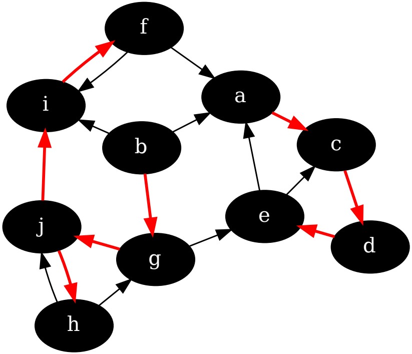
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	*f*
a	1	8
b	9	
c	2	7
d	3	6
e	4	5
f	15	16
g	10	19
h	12	13
i	14	17
j	11	18

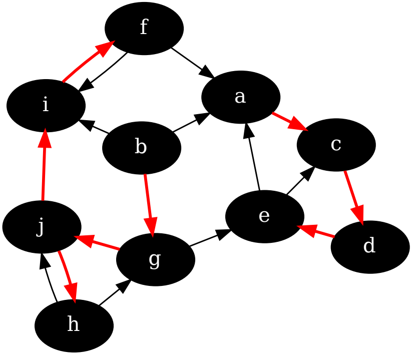
SCC by 2-DFS algorithm – Example – Phase 1



Times:

	d	f
a	1	8
b	9	20
c	2	7
d	3	6
e	4	5
f	15	16
g	10	19
h	12	13
i	14	17
j	11	18

SCC by 2-DFS algorithm – Example – Phase 1



Times:

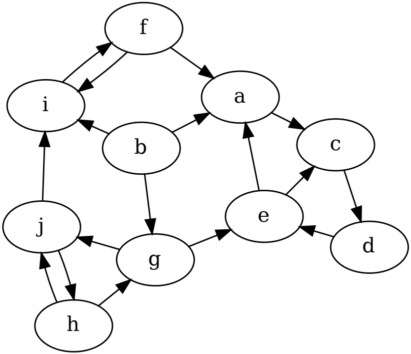
	d	f
a	1	8
b	9	20
c	2	7
d	3	6
e	4	5
f	15	16
g	10	19
h	12	13
i	14	17
j	11	18

Ordering in decreasing finish time:

b, g, j, i, f, h, a, c, d, e

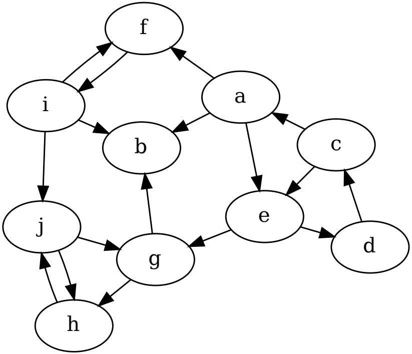
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



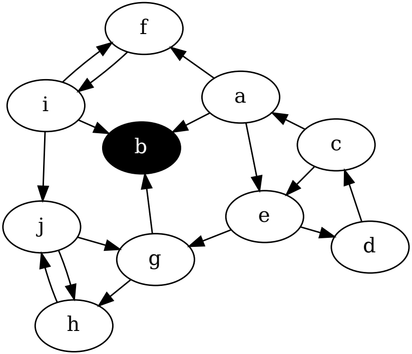
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



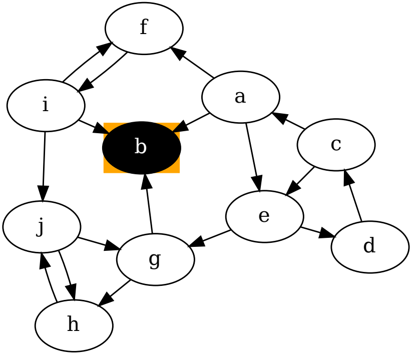
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



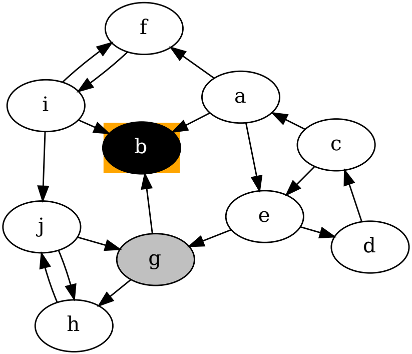
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



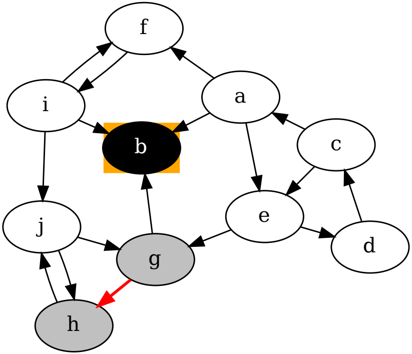
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



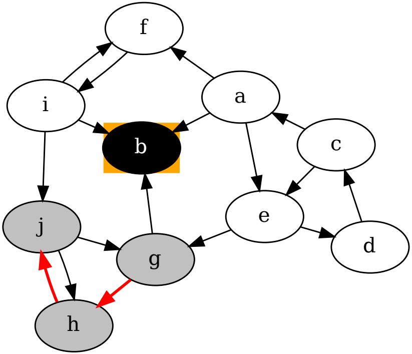
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



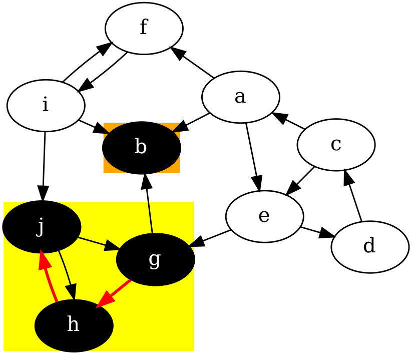
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



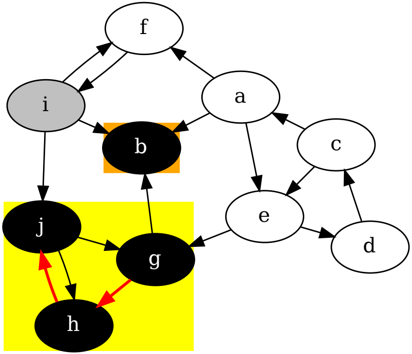
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



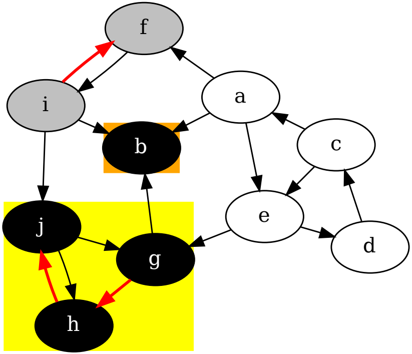
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



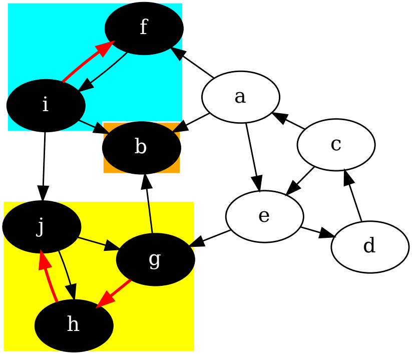
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



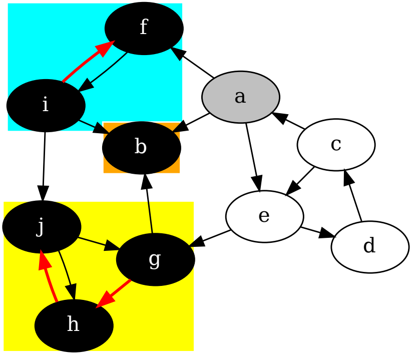
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



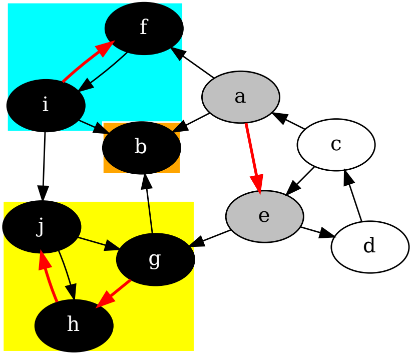
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



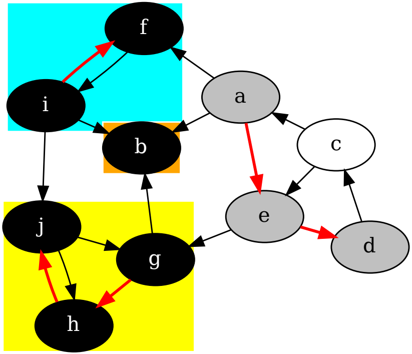
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



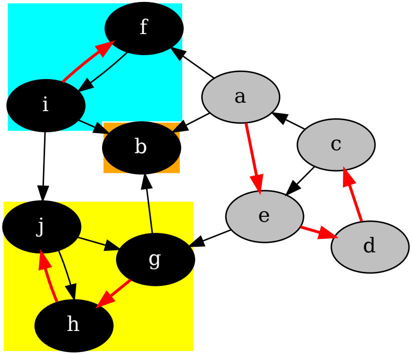
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



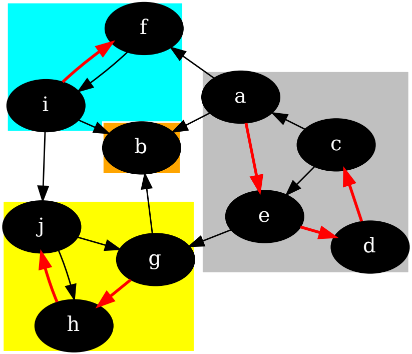
SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



SCC by 2-DFS algorithm – Example – Phase 2

Ordering: b, g, j, i, f, h, a, c, d, e



Intuition – Component digraph

Definition

If $G = (V, A)$ is a digraph, G^{SCC} is the digraph that has:

- A vertex for each SCC of G.
- An arc $C_1 C_2$ if there exist $x_1 \in C_1, x_2 \in C_2$ with $x_1 x_2 \in A$.

Lemma

G^{SCC} is always a DAG.

Intuition – Component digraph

Definition

If $G = (V, A)$ is a digraph, G^{SCC} is the digraph that has:

- A vertex for each SCC of G.
- An arc C_1C_2 if there exist $x_1 \in C_1, x_2 \in C_2$ with $x_1x_2 \in A$.

Lemma

G^{SCC} is always a DAG.

Proof.

If there exist arcs $C_1 \to C_2$ and $C_2 \to C_1$, $C_1 \cup C_2$ is strongly connected, contradicting maximality. □

Intuition

Key ideas:

- If x finished last in first DFS of G

Intuition

Key ideas:

- If x finished last in first DFS of G
- $\Rightarrow$ the SCC of x is a source in G^{SCC}

Intuition

Key ideas:

- If x finished last in first DFS of G
- $\Rightarrow$ the SCC of x is a source in G^{SCC}
- $\Rightarrow$ the SCC of x is a sink in $(G^T)^{SCC}$

Intuition

Key ideas:

- If x finished last in first DFS of G
- $\Rightarrow$ the SCC of x is a source in G^{SCC}
- $\Rightarrow$ the SCC of x is a sink in $(G^T)^{SCC}$
- $\Rightarrow$ a DFS in G^T from x will visit exactly the SCC of x

Sources SCC finishes last

Lemma

Let $G = (V, A)$ *be a digraph,* C_1, C_2 *two SCCs, with* $x_1 \in C_1, x_2 \in C_2$ *and* $x_1x_2 \in A$*. Then, there exists a vertex of* C_1 *which finishes* **after** *all vertices of* C_2*.*

Sources SCC finishes last

Lemma

Let $G = (V, A)$ be a digraph, C_1, C_2 two SCCs, with $x_1 \in C_1, x_2 \in C_2$ and $x_1x_2 \in A$. Then, there exists a vertex of C_1 which finishes **after** *all vertices of C_2.*

Proof.

- First vertex of $C_1 \cup C_2$ to be discovered is $y \in C_1$:
  By White-Path theorem, all of $C_1 \cup C_2$ are y's descendants, so finish before y.

Sources SCC finishes last

Lemma

Let $G = (V, A)$ be a digraph, C_1, C_2 two SCCs, with $x_1 \in C_1, x_2 \in C_2$ and $x_1 x_2 \in A$. Then, there exists a vertex of C_1 which finishes **after** *all vertices of C_2.*

Proof.

- First vertex of $C_1 \cup C_2$ to be discovered is $y \in C_1$:
  By White-Path theorem, all of $C_1 \cup C_2$ are y's descendants, so finish before y.
- First vertex of $C_1 \cup C_2$ to be discovered is $y \in C_2$:
  By White-Path theorem, all vertices of C_2 are discovered after y and finish before y. There is no path from C_2 to C_1 (o/w G^{SCC} not a DAG), so at f_y all of C_1 still White $\Rightarrow$ finishes later than y.

□

Proof of Correctness

Proof.

Induction on number of SCCs.

- Suppose first k SCCs are correct.
- DFS for $(k+1)$-th SCC starts at x which has largest **finish** time of all White vertices.
  - All vertices of the SCC of x (C) are currently White (I.H.)
  - White-Path theorem: SCC computed contains **at least** all of C
  - Need to prove: we do not put **other stuff** in computed SCC of x

Proof of Correctness

Proof.

Induction on number of SCCs.

- Suppose first k SCCs are correct.
- DFS for $(k+1)$-th SCC starts at x which has largest **finish** time of all White vertices.
  - All vertices of the SCC of x (C) are currently White (I.H.)
  - White-Path theorem: SCC computed contains **at least** all of C
  - Need to prove: we do not put **other stuff** in computed SCC of x
- If $y \notin C$ is reachable from x in G^T:
  - Either y is in the first k components $\Rightarrow$ y Black
  - Or $y \in C'$ with some $C \to C'$ path in G^T

Proof of Correctness

Proof.

Induction on number of SCCs.

- Suppose first k SCCs are correct.
- DFS for $(k+1)$-th SCC starts at x which has largest **finish** time of all White vertices.
  - All vertices of the SCC of x (C) are currently White (I.H.)
  - White-Path theorem: SCC computed contains **at least** all of C
  - Need to prove: we do not put **other stuff** in computed SCC of x
- If $y \notin C$ is reachable from x in G^T:
  - Either y is in the first k components $\Rightarrow$ y Black
  - Or $y \in C'$ with some $C \to C'$ path in G^T
  - Then, $\exists C' \to C$ path in G
  - $\Rightarrow$ some vertex of C' finishes after x, contradiction!!

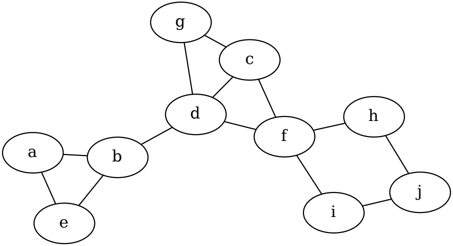
□

Articulation Points

Articulation Point

Definition

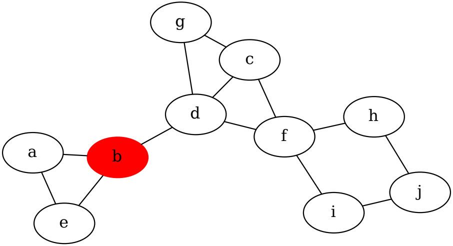
In a **undirected** graph $G = (V, E)$, $v \in V$ is a **cut vertex** or an **articulation point** if $G - v$ has more connected components than G.



Articulation Point

Definition

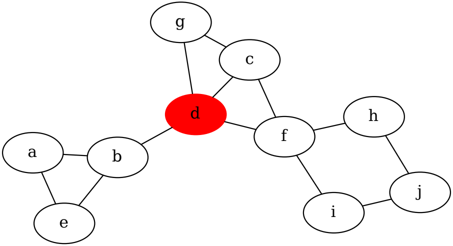
In a **undirected** graph $G = (V, E)$, $v \in V$ is a **cut vertex** or an **articulation point** if $G - v$ has more connected components than G.



Articulation Point

Definition

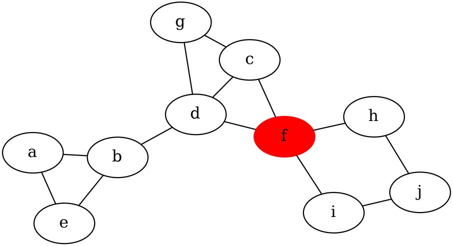
In a **undirected** graph $G = (V, E)$, $v \in V$ is a **cut vertex** or an **articulation point** if $G - v$ has more connected components than G.



Articulation Point

Definition

In a **undirected** graph $G = (V, E)$, $v \in V$ is a **cut vertex** or an **articulation point** if $G - v$ has more connected components than G.



Articulation Points

Problem

Given undirected graph G, output all articulation points (if any).

Articulation Points

Problem

Given undirected graph G, output all articulation points (if any).

Obvious algorithm:

$C \leftarrow$ number of components of G (DFS)
for $v \in V$ **do**
    if comps($G - v$)$>C$ **then**
        Add v to output
    end if
end for

Articulation Points

Problem

Given undirected graph G, output all articulation points (if any).

Obvious algorithm:

$C \leftarrow$ number of components of G (DFS)
for $v \in V$ **do**
    if comps$(G - v) > C$ **then**
        Add v to output
    end if
end for

Complexity $O(mn + n^2)$ for lists, $O(n^3)$ for matrices. Better?

Articulation Points – DFS

We will execute DFS on G and try to detect articulation points by studying the DFS tree.

Articulation Points – DFS

We will execute DFS on G and try to detect articulation points by studying the DFS tree.

Lemma

The root r of a DFS tree is an articulation point if and only if r has at least two children.

Lemma

An internal vertex v of a DFS tree is an articulation point, if and only if it has a child c such that no descendant of c is adjacent to a proper ancestor of v.

Articulation Points – Lemma 1

Lemma

The root r of a DFS tree is an articulation point if and only if r has at least two children.

Articulation Points – Lemma 1

Lemma

The root r of a DFS tree is an articulation point if and only if r has at least two children.

Proof.

- $\Leftarrow$: let c_1, c_2 be the two children with min discovery times. Then, c_1, c_2 in distinct components of $G - r$.
  - At time 2 only r, c_1 Gray, so if $c_1 \to c_2$ path exists, by White-Path theorem, c_2 would be descendant of c_1.

Articulation Points – Lemma 1

Lemma

The root r of a DFS tree is an articulation point if and only if r has at least two children.

Proof.

- $\Leftarrow$: let c_1, c_2 be the two children with min discovery times. Then, c_1, c_2 in distinct components of $G - r$.
  - At time 2 only r, c_1 Gray, so if $c_1 \to c_2$ path exists, by White-Path theorem, c_2 would be descendant of c_1.
- $\Rightarrow$: if r has one child, there is a path between any two vertices of $G - r$ (using tree edges only).

□

Articulation Points – Lemma 2

Lemma

An internal vertex v of a DFS tree is an articulation point, if and only if it has a child c such that no descendant of c is adjacent to a proper ancestor of v. (assume G connected)

Articulation Points – Lemma 2

Lemma

An internal vertex v of a DFS tree is an articulation point, if and only if it has a child c such that no descendant of c is adjacent to a proper ancestor of v. (assume G connected)

Proof.

- $\Leftarrow$: c and its descendants form a component of $G - v$ which does not contain the proper ancestors of v. (also: no cross edges)

Articulation Points – Lemma 2

Lemma

An internal vertex v of a DFS tree is an articulation point, if and only if it has a child c such that no descendant of c is adjacent to a proper ancestor of v. (assume G connected)

Proof.

- $\Leftarrow$: c and its descendants form a component of $G - v$ which does not contain the proper ancestors of v. (also: no cross edges)
- $\Rightarrow$: Let C_1 be the component of $G - v$ where DFS started.
  - v has earlier discovery than all vertices of $G \setminus (C_1 \cup \{v\})$
  - $\Rightarrow$ all other vertices of $G \setminus (C_1 \cup \{v\})$ are descendants of v by White-Path theorem
  - All proper ancestors of v are in C_1
  - No edges between C_1 and other components of $G - v$

□

Towards an algorithm

- Execute DFS
- Decide if root is an articulation point (easy!)
- Decide if each internal vertex is an articulation point

Towards an algorithm

- Execute DFS
- Decide if root is an articulation point (easy!)
- Decide if each internal vertex is an articulation point
  - Problem: obvious algorithm is $O(m)$ per vertex...
  - Need to store some information so we don't repeat work...

Going MADD

Definition

Given G and DFS tree, we define $\mathrm{madd}(v)$ to be the **minimum ancestor-descendant discovery** time among the neighbors of v. Formally,

$$\mathrm{madd}(v) = \min_{u \in \mathrm{desc}(v), w \in N[u]} d_w$$

Going MADD

Definition

Given G and DFS tree, we define $\mathrm{madd}(v)$ to be the **minimum ancestor-descendant discovery** time among the neighbors of v. Formally,

$$\mathrm{madd}(v) = \min_{u \in \mathrm{desc}(v), w \in N[u]} d_w$$

- In other words: for each v, we check all the sub-trees rooted at v, and find who has the **highest** neighbor in the tree.

Going MADD

Definition

Given G and DFS tree, we define $\mathrm{madd}(v)$ to be the **minimum ancestor-descendant discovery** time among the neighbors of v. Formally,

$$\mathrm{madd}(v) = \min_{u \in \mathrm{desc}(v), w \in N[u]} d_w$$

- In other words: for each v, we check all the sub-trees rooted at v, and find who has the **highest** neighbor in the tree.
- Claim: For internal vertex v with child x, v is an articulation point if and only if

$$\mathrm{madd}(x) = d_v$$

.

- Claim: $\mathrm{madd}(v)$ can be computed for all vertices in linear time.

MADD in linear time

Lemma

We can compute all minimum ancestor-descendant discovery times in linear time.

MADD in linear time

Lemma

We can compute all minimum ancestor-descendant discovery times in linear time.

Proof.

$$\mathrm{madd}(v) = \min_{u \in \mathrm{desc}(v), w \in N[u]} d_w$$

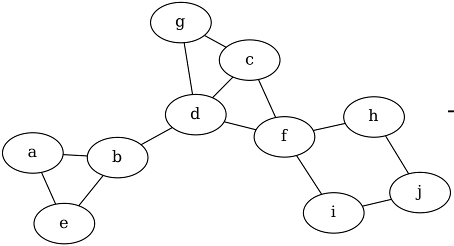
is equivalent to:

$$\mathrm{madd}(v) = \min\{(\min_{u \in \mathrm{child}(v)} \mathrm{madd}(u)), (\min_{u \in N(v)} d_u\})$$

which can be computed bottom-up by checking computed values for the children of each node, when a vertex becomes Black.

□

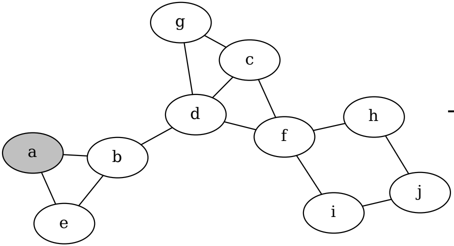
Articulation points – Example



Times:

	d	f	madd
a			*
b			
c			
d			
e			
f			
g			
h			
i			
j			

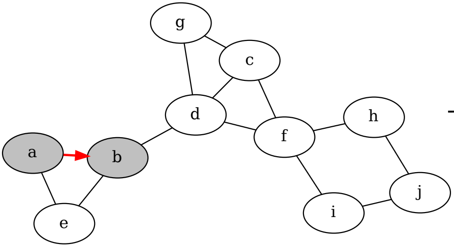
Articulation points – Example



Times:

	d	f	madd
a	1		*
b			
c			
d			
e			
f			
g			
h			
i			
j			

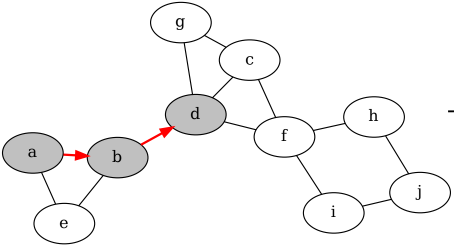
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c			
d			
e			
f			
g			
h			
i			
j			

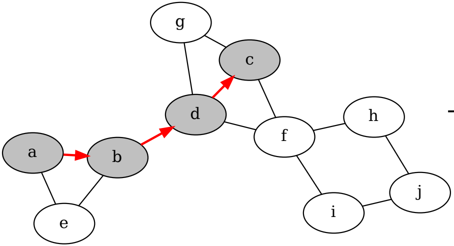
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c			
d	3		
e			
f			
g			
h			
i			
j			

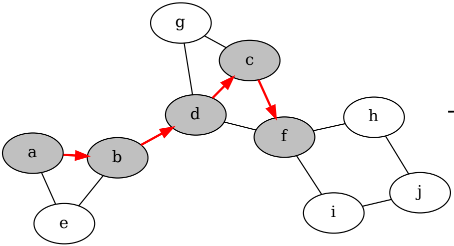
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f			
g			
h			
i			
j			

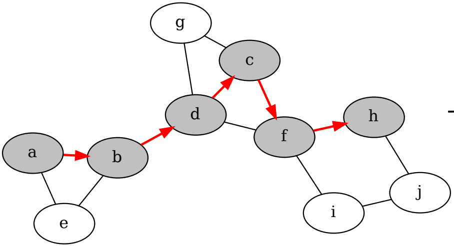
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h			
i			
j			

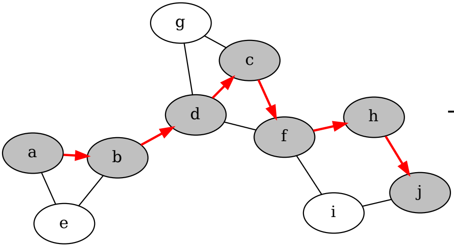
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6		
i			
j			

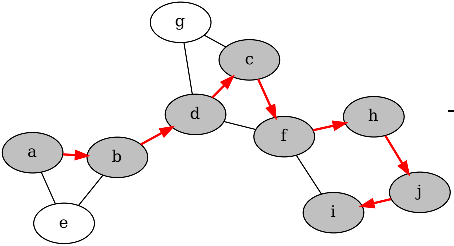
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6		
i			
j	7		

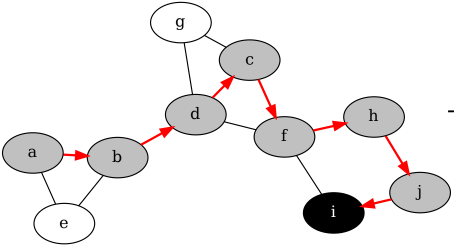
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6		
i	8		
j	7		

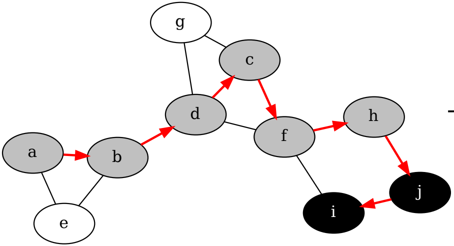
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6		
i	8	9	
j	7		

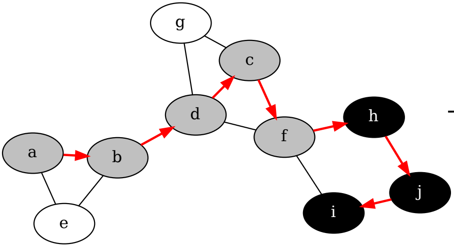
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6		
i	8	9	
j	7	10	

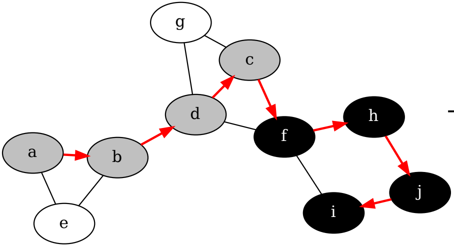
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5		
g			
h	6	11	
i	8	9	
j	7	10	

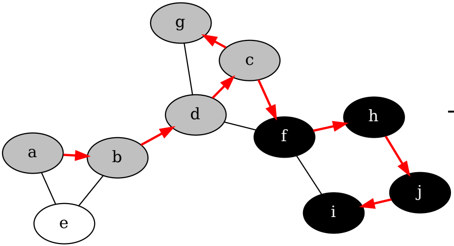
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5	12	
g			
h	6	11	
i	8	9	
j	7	10	

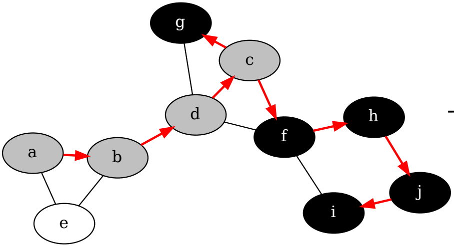
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5	12	
g	13		
h	6	11	
i	8	9	
j	7	10	

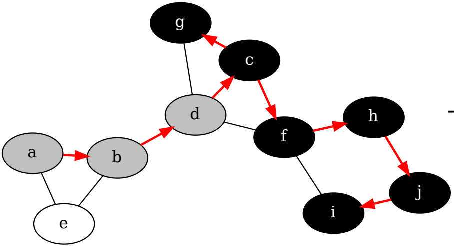
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4		
d	3		
e			
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

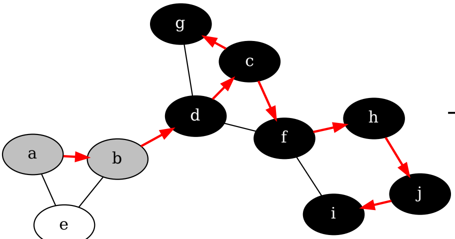
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4	15	
d	3		
e			
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

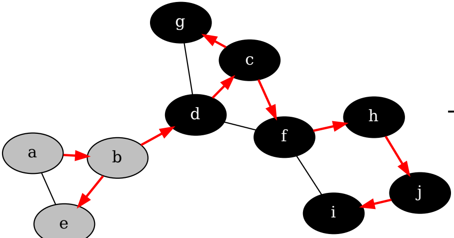
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4	15	
d	3	16	
e			
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

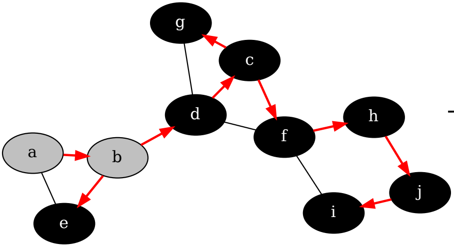
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4	15	
d	3	16	
e	17		
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

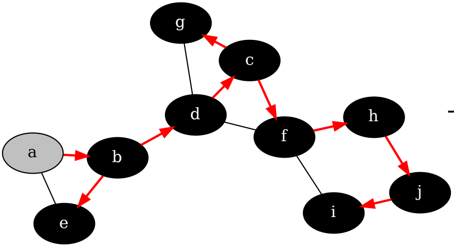
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2		
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

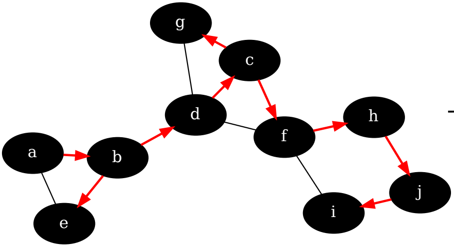
Articulation points – Example



Times:

	d	f	madd
a	1		*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

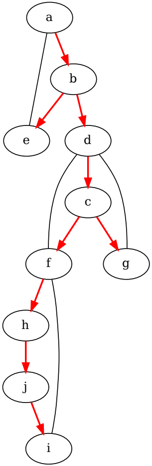
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

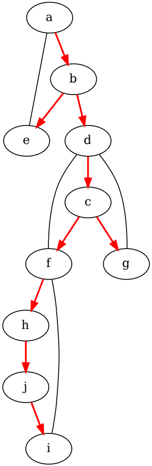
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	
j	7	10	

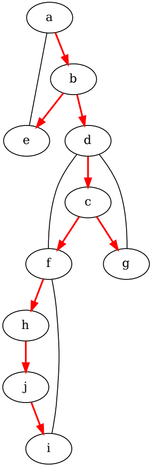
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	5
j	7	10	

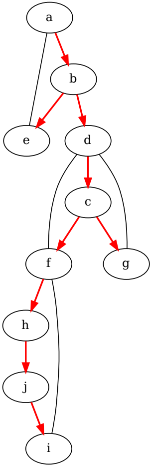
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	
i	8	9	5
j	7	10	5

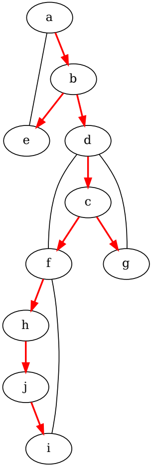
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	
g	13	14	
h	6	11	5
i	8	9	5
j	7	10	5

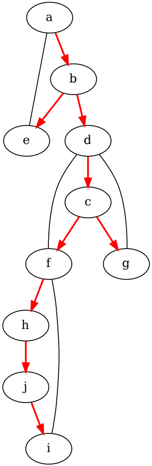
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	3
g	13	14	
h	6	11	5
i	8	9	5
j	7	10	5

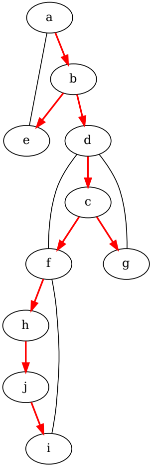
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	
d	3	16	
e	17	18	
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

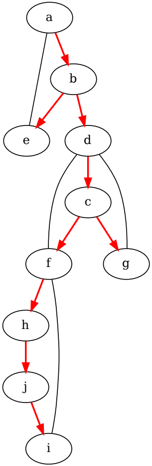
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	3
d	3	16	
e	17	18	
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

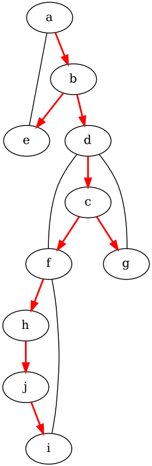
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	3
d	3	16	2
e	17	18	
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

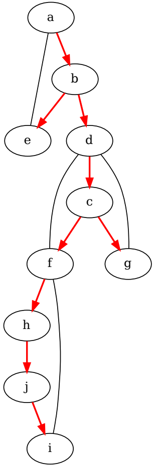
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	
c	4	15	3
d	3	16	2
e	17	18	1
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

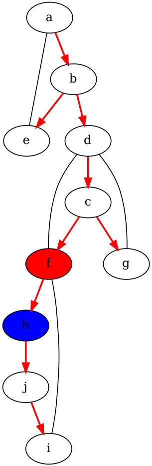
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	1
c	4	15	3
d	3	16	2
e	17	18	1
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

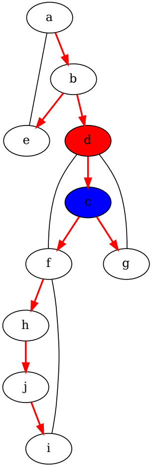
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	1
c	4	15	3
d	3	16	2
e	17	18	1
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

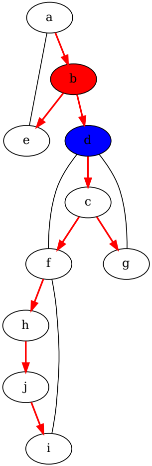
Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	1
c	4	15	3
d	3	16	2
e	17	18	1
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5

Articulation points – Example



Times:

	d	f	madd
a	1	20	*
b	2	19	1
c	4	15	3
d	3	16	2
e	17	18	1
f	5	12	3
g	13	14	3
h	6	11	5
i	8	9	5
j	7	10	5