Graph Search: Depth First Search (DFS)

Lily Li

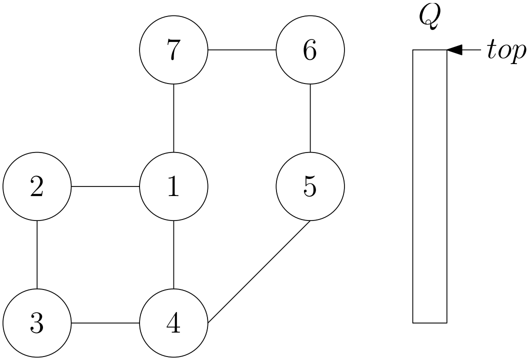
CSC 265: Enriched Data Structures and Analysis

27 November 2018

Outline

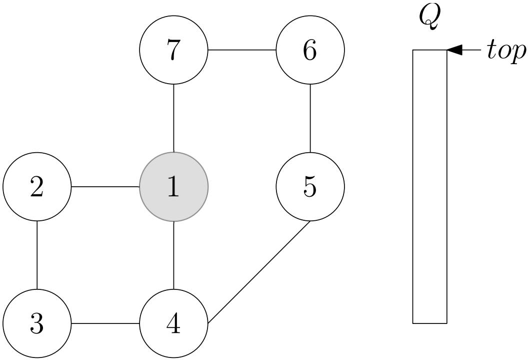
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



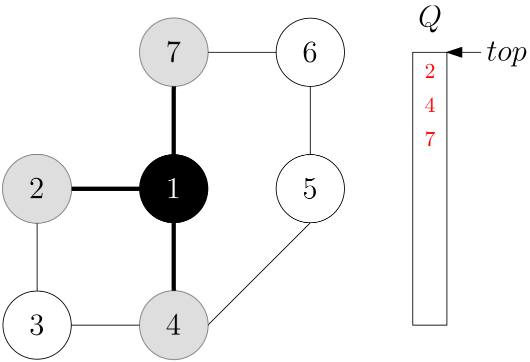
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



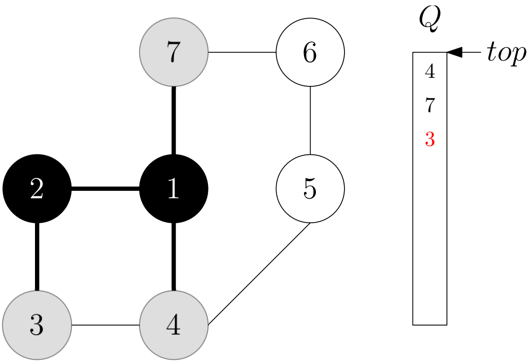
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



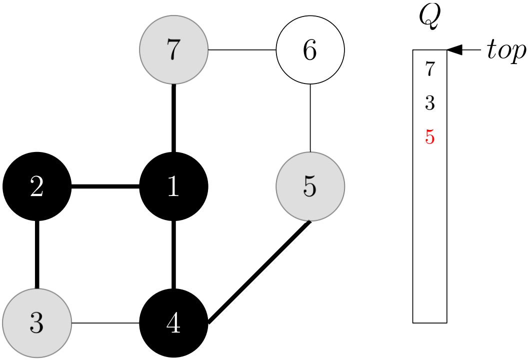
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



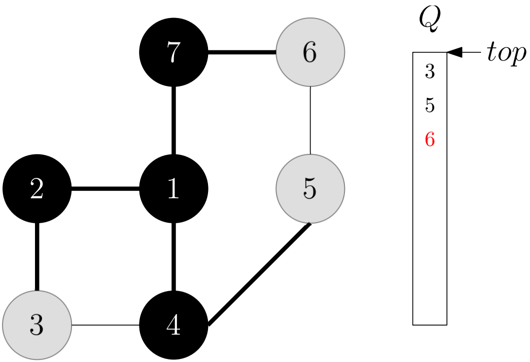
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



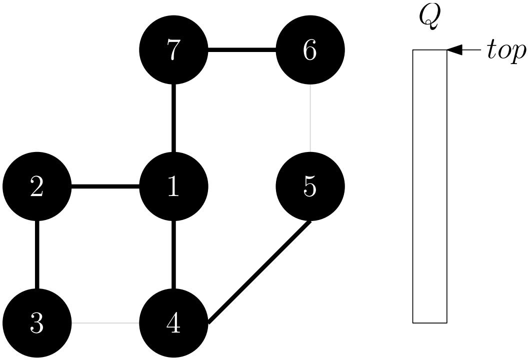
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



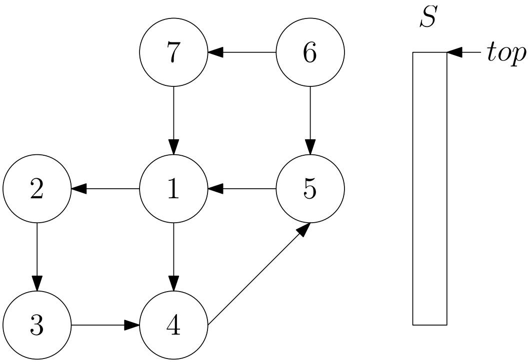
BFS

$G = (V, E)$ is a graph with with $|V| = n$ and $|E| = m$.



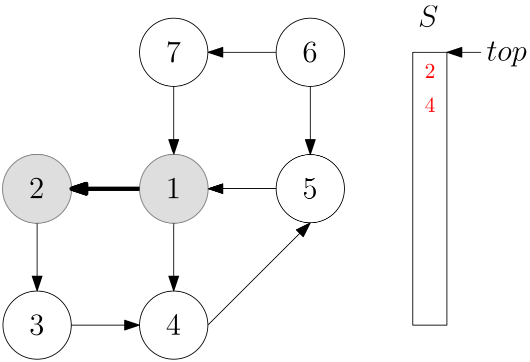
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



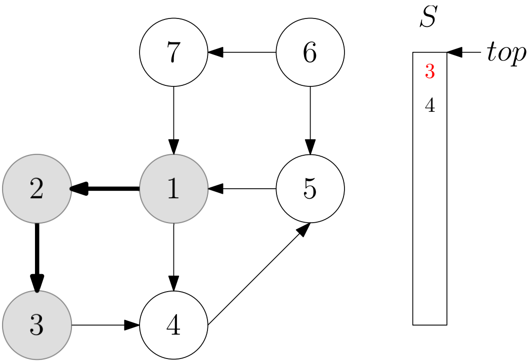
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



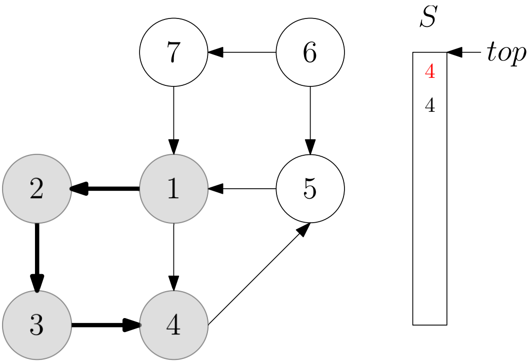
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



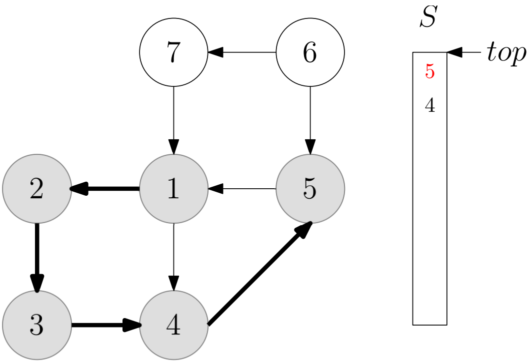
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



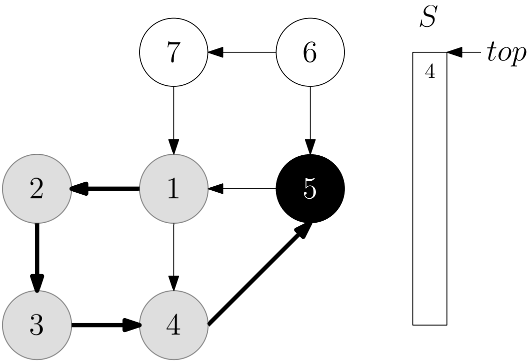
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



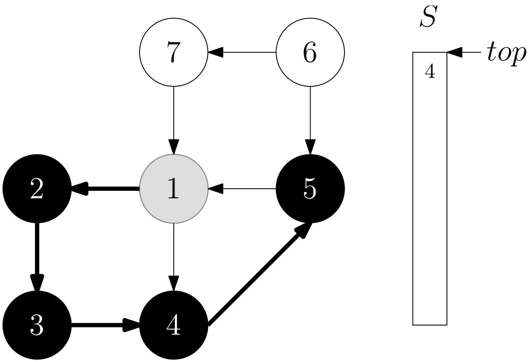
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



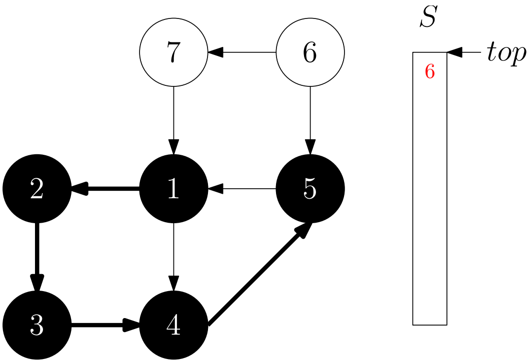
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



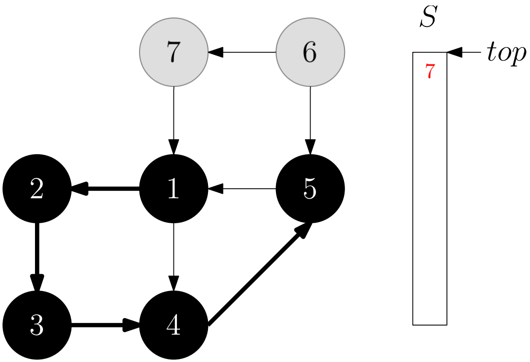
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



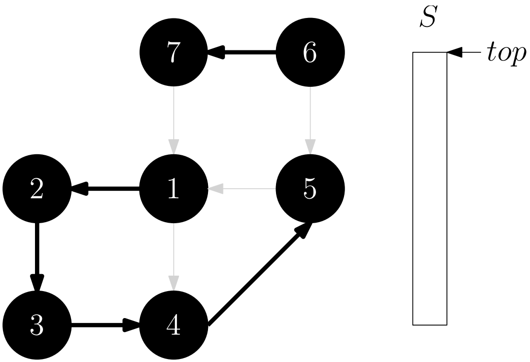
DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



DFS

$G = (V, E)$ is a **directed** graph with with $|V| = n$ and $|E| = m$.



DFS: Basics

(Special) Parameters of Vertex v:

- $d[v]$: the discovery time of v. *Different from* $v.d$ *of BSF.*
- $f[v]$: the finish time of v.
- $v.colour$: indicates state of v; v is white before $d[v]$, gray between $d[v]$ and $f[v]$, and black after $f[v]$.

DFS: Pseudo-code

DFS(G)

$t \leftarrow 0$
for $v \in V$
    $v.colour =$ white
for $v \in V$
    if $v.colour =$ white
        DFS-VISIT(v)

DFS: Pseudo-code

```
DFS(G)
t ← 0
for v ∈ V
    v.colour = white
for v ∈ V
    if v.colour = white
        DFS-Visit(v)
```

```
DFS-Visit(v)
v.colour ← gray
t ← t + 1
d[v] ← t
for u ∈ N(v)
    if u.colour = white
        u.parent = v
        DFS-Visit(u)
v.colour ← black
f[v] ← t
t ← t + 1
```

DFS: Pseudo-code

```
DFS(G)
t ← 0
for v ∈ V
    v.colour = white
for v ∈ V
    if v.colour = white
        DFS-Visit(v)
```

```
DFS-Visit(v)
v.colour ← gray
t ← t + 1
d[v] ← t
for u ∈ N(v)
    if u.colour = white
        u.parent = v
        DFS-Visit(u)
v.colour ← black
f[v] ← t
t ← t + 1
```

Q: What is the running time?

DFS: Pseudo-code

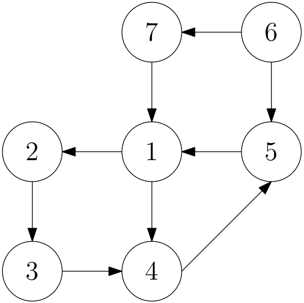
```
DFS(G)
t ← 0
for v ∈ V
        v.colour = white
for v ∈ V
        if v.colour = white
                DFS-Visit(v)
```

```
DFS-Visit(v)
v.colour ← gray
t ← t + 1
d[v] ← t
for u ∈ N(v)
        if u.colour = white
                u.parent = v
                DFS-Visit(u)
v.colour ← black
f[v] ← t
t ← t + 1
```

Q: What is the running time?

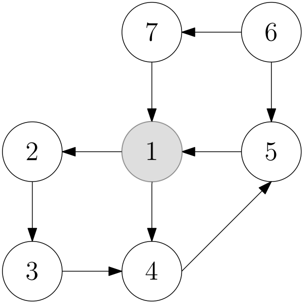
A: $O(m+n)$.

DFS: Example



		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v														

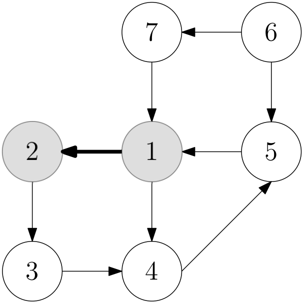
DFS: Example

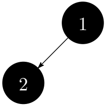


$d[v]$ $f[v]$

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1													

DFS: Example

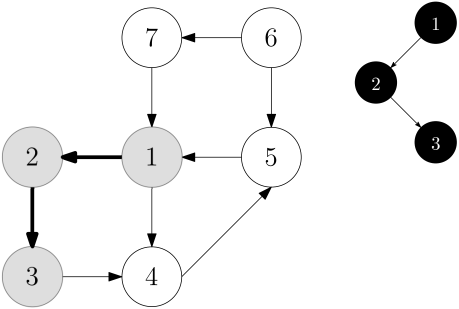




$d[v]$ $f[v]$

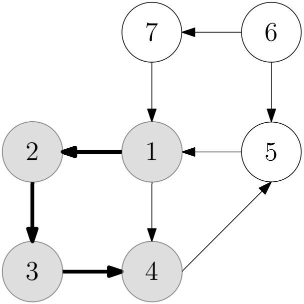
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1	2												

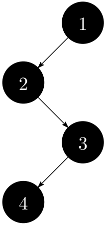
DFS: Example



$d[v]$	$f[v]$	t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		v	1	2	3											

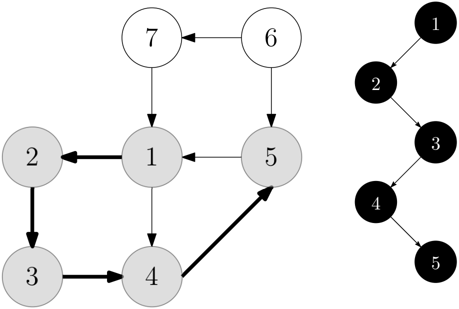
DFS: Example





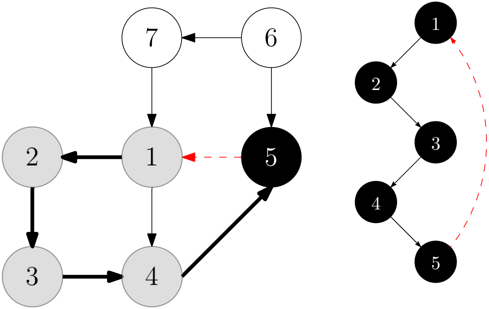
		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4										

DFS: Example



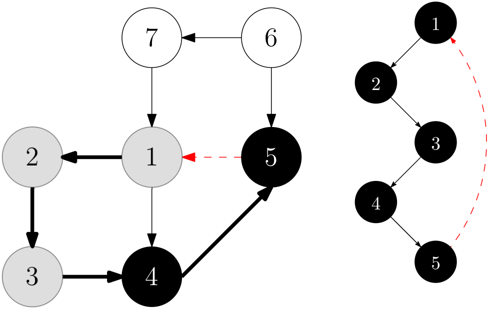
$d[v]$	$f[v]$	t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		v	1	2	3	4	5									

DFS: Example



			1	2	3	4	5	6	7	8	9	10	11	12	13	14
		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5								

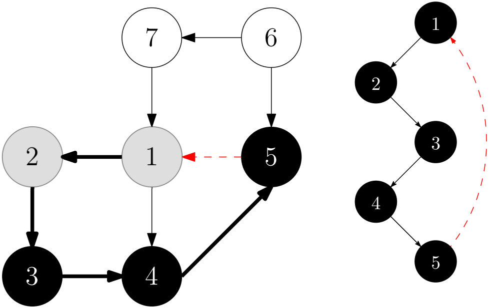
DFS: Example



$d[v]$ $f[v]$

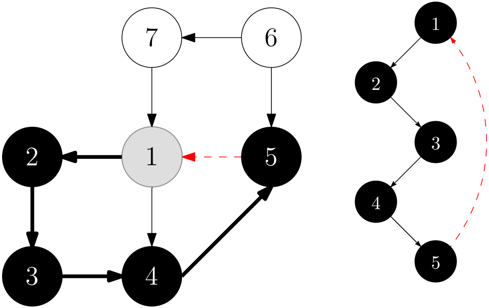
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1	2	3	4	5	5	4							

DFS: Example



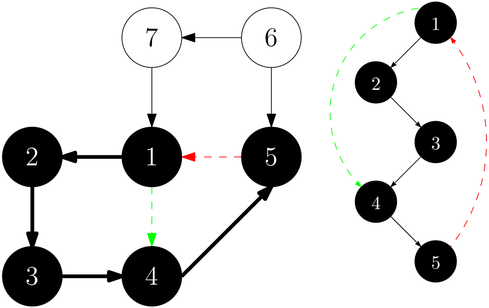
$d[v]$	$f[v]$	t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		v	1	2	3	4	5	5	4	3						

DFS: Example



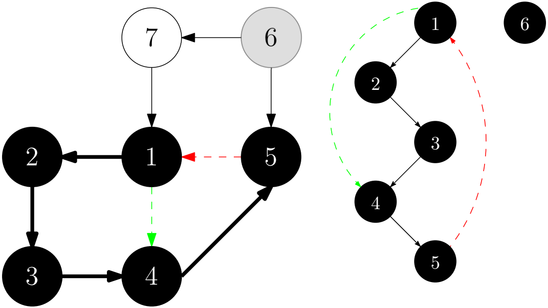
		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5	4	3	2					

DFS: Example



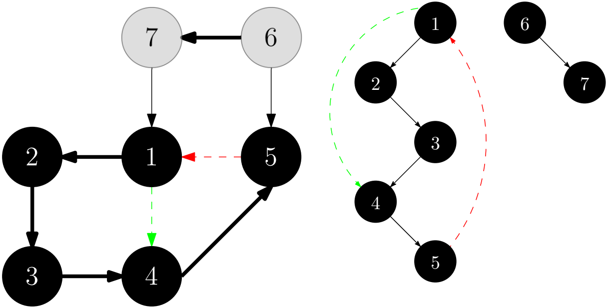
		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5	4	3	2	1				

DFS: Example



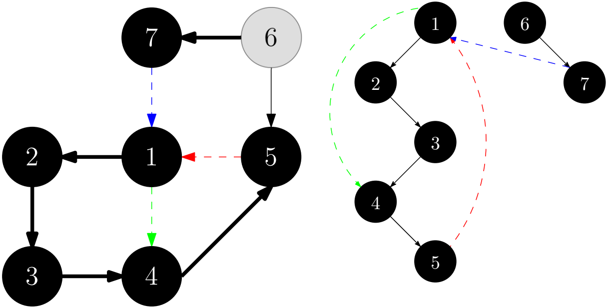
t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$ $f[v]$ v	1	2	3	4	5	5	4	3	2	1	6			

DFS: Example



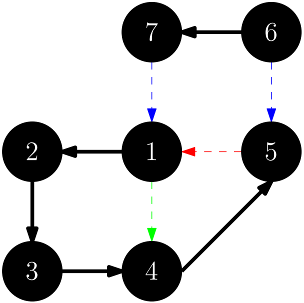
		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5	4	3	2	1	6	7		

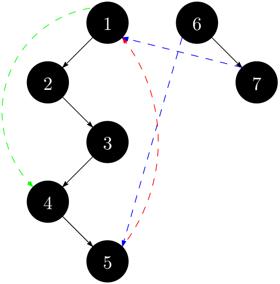
DFS: Example



		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5	4	3	2	1	6	7	7	

DFS: Example



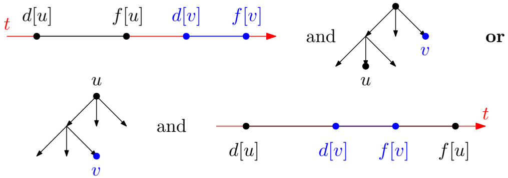


		t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$d[v]$	$f[v]$	v	1	2	3	4	5	5	4	3	2	1	6	7	7	6
			(	(	(	(	(	)	)	)	)	)	(	(	)	)

Parenthesis Structure

Theorem (Parenthetical Property)

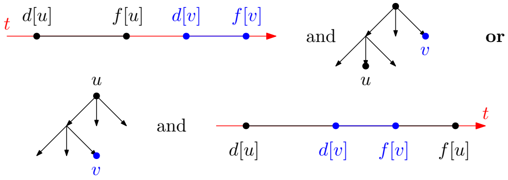
Let $G = (V, E)$ and $u, v \in V$. Then the following could occur:



Parenthesis Structure

Theorem (Parenthetical Property)

Let $G = (V, E)$ and $u, v \in V$. Then the following could occur:



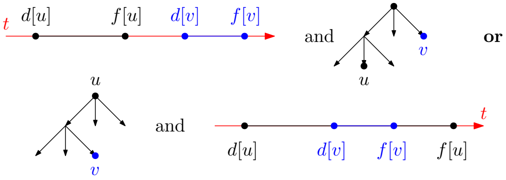
Proof.

W.l.o.g. assume that $d[u] < d[v]$.

Parenthesis Structure

Theorem (Parenthetical Property)

Let $G = (V, E)$ and $u, v \in V$. Then the following could occur:



Proof.

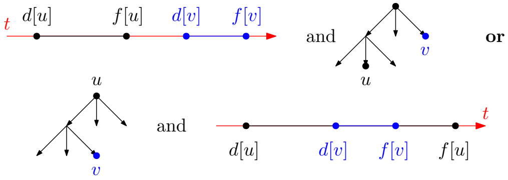
W.l.o.g. assume that $d[u] < d[v]$.

1. $f[u] < d[v]$: $d[u] < f[u] < d[v] < f[v]$. Neither vertex was gray when the other was discovered. No ancestor relationship.

Parenthesis Structure

Theorem (Parenthetical Property)

Let $G = (V, E)$ and $u, v \in V$. Then the following could occur:



Proof.

W.l.o.g. assume that $d[u] < d[v]$.

1. $f[u] < d[v]$: $d[u] < f[u] < d[v] < f[v]$. Neither vertex was gray when the other was discovered. No ancestor relationship.

2. $f[u] > d[v]$: u was gray when v was discovered. Thus v is a descendant of u and finish exploring v before u ($f[v] < f[u]$).

$\square$

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

Proof.

($\Longrightarrow$) Suppose that v is a descendant of u.

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

Proof.

$(\implies)$ Suppose that v is a descendant of u. Then **the path** $u \rightsquigarrow v$ on the depth-first tree **is a white path** at time $d[u]$.

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

Proof.

($\Longrightarrow$) Suppose that v is a descendant of u. Then **the path** $u \rightsquigarrow v$ on the depth-first tree **is a white path** at time $d[u]$.
($\Longleftarrow$) Suppose there exists a white path from u to v at time $d[u]$.

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

Proof.

($\Longrightarrow$) Suppose that v is a descendant of u. Then **the path** $u \rightsquigarrow v$ on the depth-first tree **is a white path** at time $d[u]$.
($\Longleftarrow$) Suppose there exists a white path from u to v at time $d[u]$.
Claim: all vertices w on this path satisfy $f[w] < f[u]$.

White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

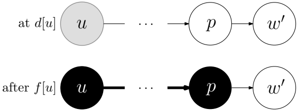
Proof.

($\Longrightarrow$) Suppose that v is a descendant of u. Then **the path** $u \rightsquigarrow v$ on the depth-first tree **is a white path** at time $d[u]$.

($\Longleftarrow$) Suppose there exists a white path from u to v at time $d[u]$.

Claim: all vertices w on this path satisfy $f[w] < f[u]$.

Proof: Suppose not and let w' be the first vertex on the path such that $f[w'] > f[u]$. Then the parent p of w' on the path finished before exploring the edge pw'. This is not possible.



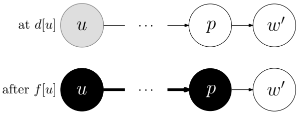
White Path

Theorem (White Path Property)

In a depth-first forest, v is a descendant of u if and only if at $d[u]$ there exists a path $u \rightsquigarrow v$ of only white nodes.

Proof.

($\Longrightarrow$) Suppose that v is a descendant of u. Then **the path** $u \rightsquigarrow v$ on the depth-first tree **is a white path** at time $d[u]$.
($\Longleftarrow$) Suppose there exists a white path from u to v at time $d[u]$.
Claim: all vertices w on this path satisfy $f[w] < f[u]$.
Proof: Suppose not and let w' be the first vertex on the path such that $f[w'] > f[u]$. Then the parent p of w' on the path finished before exploring the edge pw'. This is not possible.



By Paren. Prop. every vertex on the path, including v, is a descendant of u.

$\square$

DFS: Exercise

G is an undirected graph. Running BFS on G at vertex u produces a tree T. Running DFS on the u produces the same tree T.

DFS: Exercise

G is an undirected graph. Running BFS on G at vertex u produces a tree T. Running DFS on the u produces the same tree T.
Show that $G = T$ i.e. all edges in G belong to T.

Topological Sort: Setup

Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.

Topological Sort: Setup

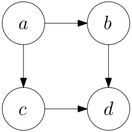
Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.

Topological Sort: Setup

Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.

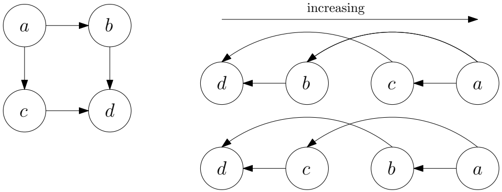
Topological Sort: Setup

Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.



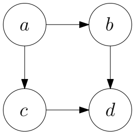
Topological Sort: Setup

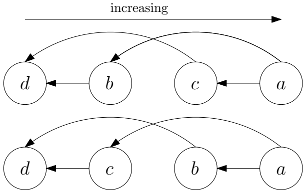
Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.

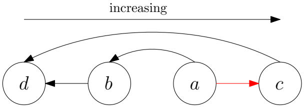


Topological Sort: Setup

Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.

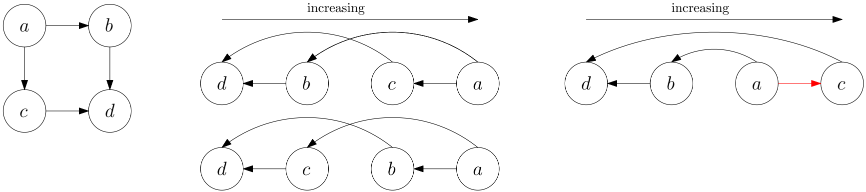






Topological Sort: Setup

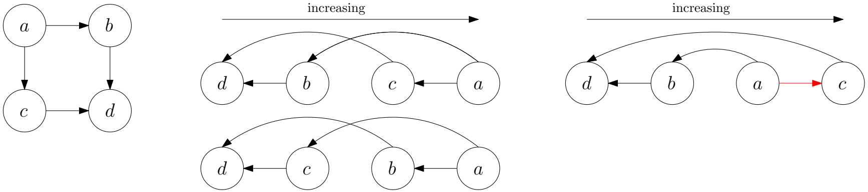
Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.



Q: Do all directed graph have a topological order? How would you perform topological sort on a graph with a topological order.

Topological Sort: Setup

Let G be a directed graph, $V = \{v_1, ..., v_n\}$ be the vertices of G and $[n] = \{1, 2, ..., n\}$.
A **topological order** is bijection $t : V \to [n]$ such that all edges (i, j) $(v_i \to v_j)$ satisfy $t(v_i) > t(v_j)$.
A **topological sort** on G finds a topological order.



Q: Do all directed graph have a topological order? How would you perform topological sort on a graph with a topological order.
A: G has a topological order if and only if it is acyclic (DAG).

Topological Sort: Algorithm 1

Using Sinks

A **sink** is a vertex with no out going edges.

Topological Sort: Algorithm 1

Using Sinks

A **sink** is a vertex with no out going edges.

Claim: If G is acyclic then there must exist a sink.

Topological Sort: Algorithm 1

Using Sinks

A **sink** is a vertex with no out going edges.

Claim: If G is acyclic then there must exist a sink.

Algorithm:

1. While graph G is not empty find a sink v.
2. Remove v and its adjacent edges.
3. Go back to step 1.

Topological Sort: Algorithm 2

Using DFS

Algorithm: run DFS and sort vertices by increasing $f[v]$.

Topological Sort: Algorithm 2

Using DFS

Algorithm: run DFS and sort vertices by increasing $f[v]$.

Lemma

If uv is an edge then $f[u] > f[v]$.

Topological Sort: Algorithm 2

Using DFS

Algorithm: run DFS and sort vertices by increasing $f[v]$.

Lemma

If uv is an edge then $f[u] > f[v]$.

Proof.

Two cases to consider:

1. DFS visits u before v: $u \to v$ a white path at time $d[u]$. By the White Path Theorem v is a descendant of u so $f[v] < f[u]$.

Topological Sort: Algorithm 2

Using DFS

Algorithm: run DFS and sort vertices by increasing $f[v]$.

Lemma

If uv is an edge then $f[u] > f[v]$.

Proof.

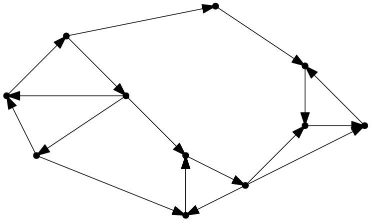
Two cases to consider:

1. DFS visits u before v: $u \to v$ a white path at time $d[u]$. By the White Path Theorem v is a descendant of u so $f[v] < f[u]$.
2. DFS visits v before u: Since the graph is acyclic, there does not exists a path $v \rightsquigarrow u$. Thus DFS on v must finish before even discovering u.

$\square$

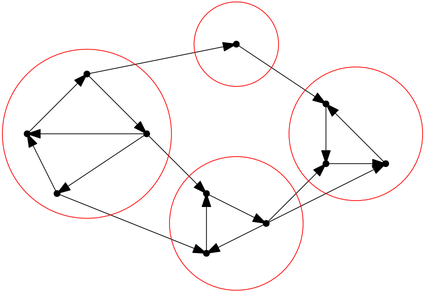
SCC: Setup

Consider this graph.



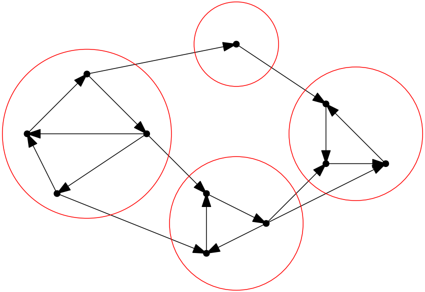
SCC: Setup

Consider this graph. Its strongly connected components (SCC) are



SCC: Setup

Consider this graph. Its strongly connected components (SCC) are



In the language of equivalence relations:

$$R: \text{ "is in the same SCC as"}$$

then uRv if there exists paths $u \rightsquigarrow v$ and $v \rightsquigarrow u$.

SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.

SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.

.

SCC: Algorithm

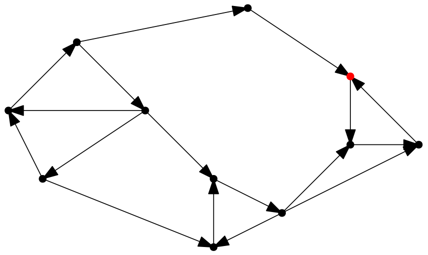
1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).

SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).
4. All vertices in the same DSF tree are in the same SCC.

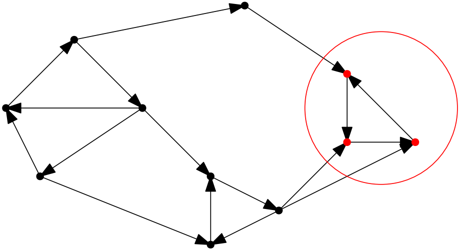
SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).
4. All vertices in the same DSF tree are in the same SCC.



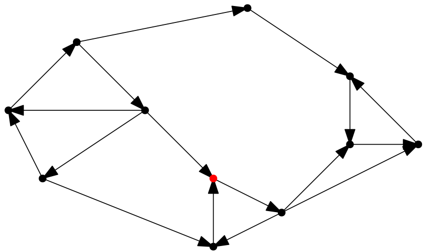
SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).
4. All vertices in the same DSF tree are in the same SCC.



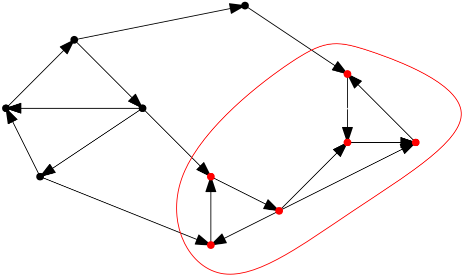
SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).
4. All vertices in the same DSF tree are in the same SCC.



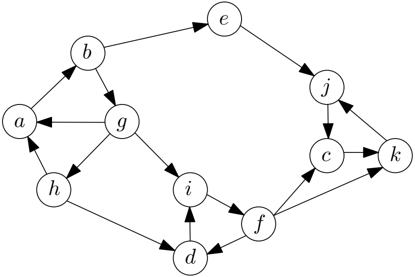
SCC: Algorithm

1. Reverse all edges in G. Call this graph G_R.
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.
3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).
4. All vertices in the same DSF tree are in the same SCC.



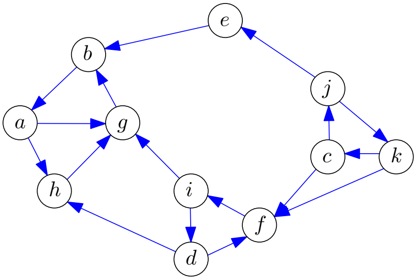
SCC: Example

0. Begin with a graph G.



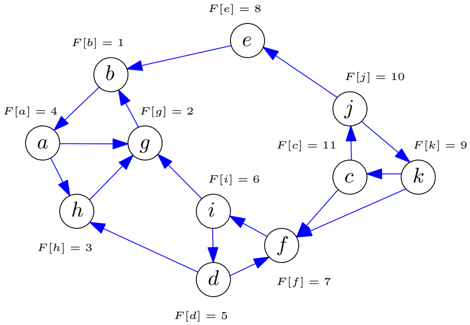
SCC: Example

1. Reverse all edges in G and call this graph G_R.



SCC: Example

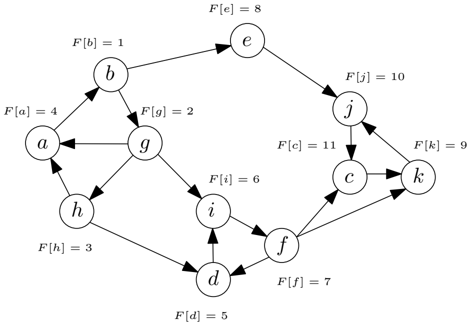
2. Run $\mathrm{DFS}(G_R)$. Record the finish times of the vertices.



Heuristic: break ties lexicographically (first a, then b, etc.)

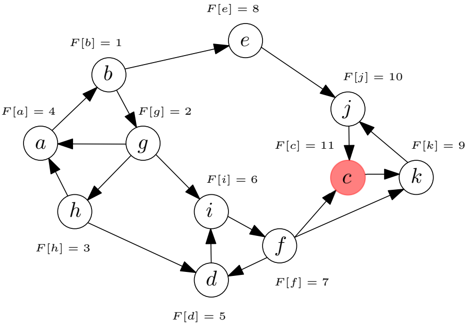
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



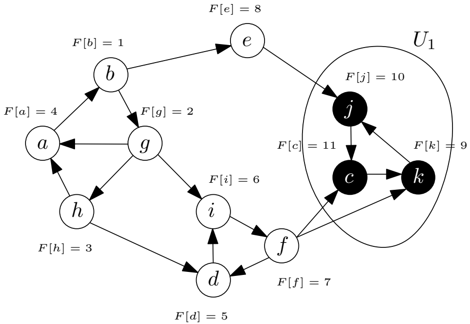
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



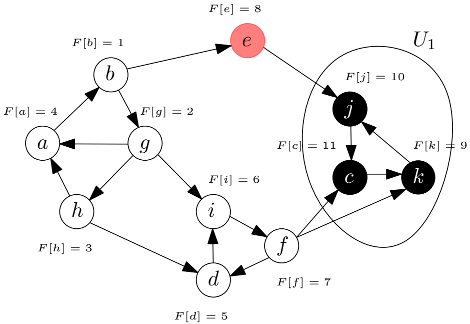
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



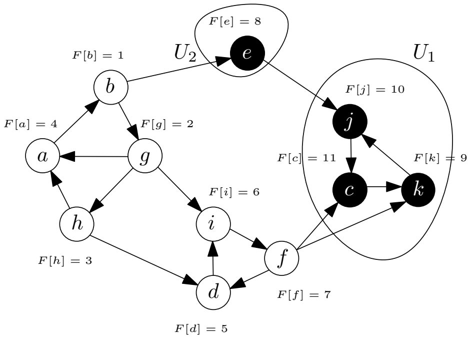
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



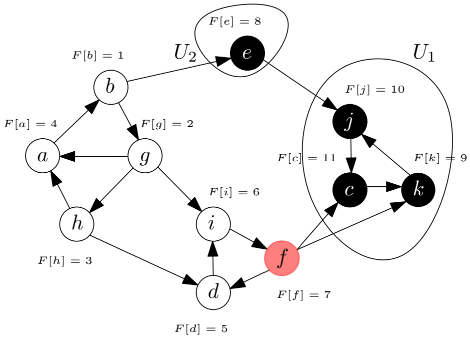
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



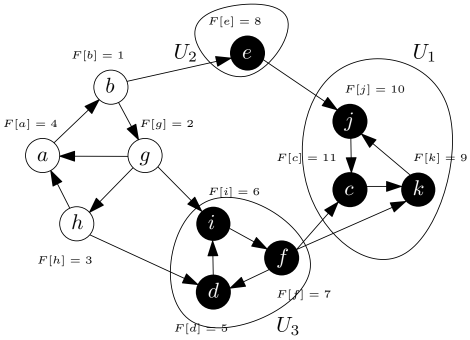
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



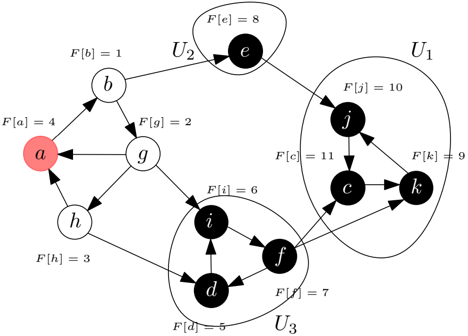
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



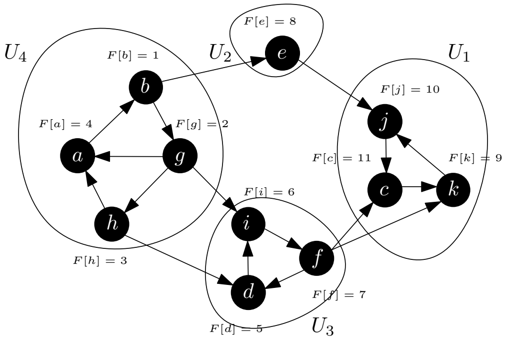
SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



SCC: Example

3. Run $\mathrm{DFS}(G)$ (in decreasing order of finish times).



SCC: Proof of Correctness

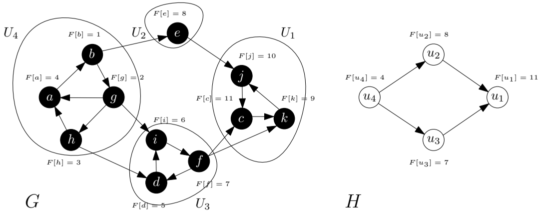
Proof.

1. **Let** $G = (V, E)$ and $G_R = (V, E_R)$, $(v_1, v_2) \in E \iff (v_2, v_1) \in E_R$.

SCC: Proof of Correctness

Proof.

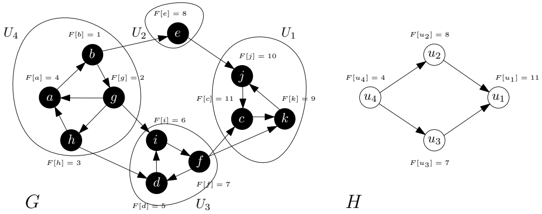
1. **Let** $G = (V, E)$ and $G_R = (V, E_R)$, $(v_1, v_2) \in E \iff (v_2, v_1) \in E_R$.
2. **Construct a new graph** $H = (U, D)$ induced by SCC. $u_i \in U$ for component U_i. $(u_1, u_2) \in D$ if and only if there exists edge $(v_1, v_2) \in E$ for $v_1 \in U_1$ and $v_2 \in U_2$.



SCC: Proof of Correctness

Proof.

1. **Let** $G = (V, E)$ and $G_R = (V, E_R)$, $(v_1, v_2) \in E \iff (v_2, v_1) \in E_R$.
2. **Construct a new graph** $H = (U, D)$ induced by SCC. $u_i \in U$ for component U_i. $(u_1, u_2) \in D$ if and only if there exists edge $(v_1, v_2) \in E$ for $v_1 \in U_1$ and $v_2 \in U_2$.

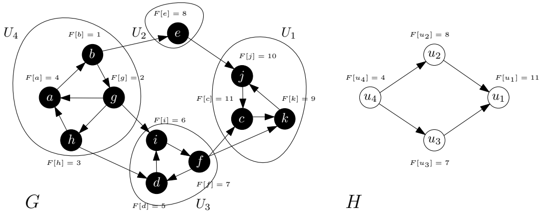


3. Note that H **is acyclic**. Let $f[u_i] = \max_{v \in U_i} f[v]$:

SCC: Proof of Correctness

Proof.

1. **Let** $G = (V, E)$ and $G_R = (V, E_R)$, $(v_1, v_2) \in E \iff (v_2, v_1) \in E_R$.
2. **Construct a new graph** $H = (U, D)$ induced by SCC. $u_i \in U$ for component U_i. $(u_1, u_2) \in D$ if and only if there exists edge $(v_1, v_2) \in E$ for $v_1 \in U_1$ and $v_2 \in U_2$.

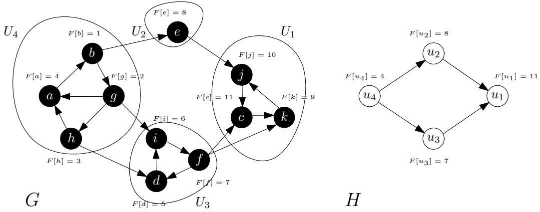


3. Note that H **is acyclic**. Let $f[u_i] = \max_{v \in U_i} f[v]$:
   Claim: if there exists an edge $(u_i, u_j) \in D$ then $f[u_i] < f[u_j]$.

SCC: Proof of Correctness

Proof.

1. **Let** $G = (V, E)$ and $G_R = (V, E_R)$, $(v_1, v_2) \in E \iff (v_2, v_1) \in E_R$.
2. **Construct a new graph** $H = (U, D)$ induced by SCC. $u_i \in U$ for component U_i. $(u_1, u_2) \in D$ if and only if there exists edge $(v_1, v_2) \in E$ for $v_1 \in U_1$ and $v_2 \in U_2$.



3. Note that H **is acyclic**. Let $f[u_i] = \max_{v \in U_i} f[v]$:

   Claim: if there exists an edge $(u_i, u_j) \in D$ then $f[u_i] < f[u_j]$.
4. **Perform DFS on** G in decreasing order of finish times. Show that this discovers the SCCs by induction.

SCC: Inductive Proof

Claim: if there exists an edge $(u_i, u_j) \in D$ then $f[u_j] > f[u_i]$.

Lemma

$\mathrm{DFS}(G)$ *in decreasing order of finish times discovers the SCCs.*

SCC: Inductive Proof

Claim: if there exists an edge $(u_i, u_j) \in D$ then $f[u_j] > f[u_i]$.

Lemma

$\mathrm{DFS}(G)$ *in decreasing order of finish times discovers the SCCs.*

Proof.

By induction on the number of calls of $\mathrm{DFS\text{-}V{\scriptsize ISIT}}(v)$ in $\mathrm{DFS}(G)$.

1. **Base Case:** By the claim, $v \in V$ with the largest $F[v]$ is in a sink SCC, U_1. $\mathrm{DFS\text{-}V{\scriptsize ISIT}}(v)$ will find all vertices in U_1.

SCC: Inductive Proof

Claim: if there exists an edge $(u_i, u_j) \in D$ then $f[u_j] > f[u_i]$.

Lemma

$\mathrm{DFS}(G)$ *in decreasing order of finish times discovers the SCCs.*

Proof.

By induction on the number of calls of $\mathrm{DFS}\text{-}\mathrm{VISIT}(v)$ in $\mathrm{DFS}(G)$.

1. **Base Case:** By the claim, $v \in V$ with the largest $F[v]$ is in a sink SCC, U_1. $\mathrm{DFS}\text{-}\mathrm{VISIT}(v)$ will find all vertices in U_1.
2. **Inductive Step:** $\mathcal{S}$ is the set of components already found. Run $\mathrm{DFS}\text{-}\mathrm{VISIT}(v)$ for v in component U_i not yet discovered. Consider edge (v_i, v_j) for $v_i \in U_i$ and $v_j \in U_j \in S$. By the claim, $f[u_i] < f[u_j]$ so there exists a vertex in U_j with larger finish time than v. Since we are running DFS by decreasing finish times, component U_j must already have been explored. Thus we only visit the vertices in U_i.

□

SCC: Proof of Claim

Claim

If there exists an edge $(u_i, u_j) \in D$ *then* $f[u_j] > f[u_i]$.

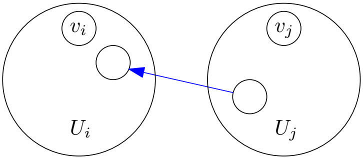
SCC: Proof of Claim

Claim

If there exists an edge $(u_i, u_j) \in D$ then $f[u_j] > f[u_i]$.

Proof.

Let $v_i \in U_i$, $v_j \in U_j$ be the first vertices explored by $\mathrm{DFS}(G_R)$ their respective component.



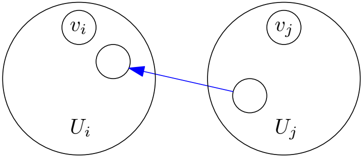
SCC: Proof of Claim

Claim

If there exists an edge $(u_i, u_j) \in D$ then $f[u_j] > f[u_i]$.

Proof.

Let $v_i \in U_i$, $v_j \in U_j$ be the first vertices explored by $\mathrm{DFS}(G_R)$ their respective component.



1. v_i **is explored before** v_j: No path $v_i \rightsquigarrow v_j \in G_R$ since v_j and v_i are in different components. Finish exploring v_i before beginning with v_j. Thus $f[v_j] > f[v_i]$.

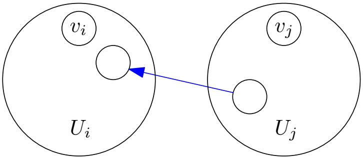
SCC: Proof of Claim

Claim

If there exists an edge $(u_i, u_j) \in D$ *then* $f[u_j] > f[u_i]$.

Proof.

Let $v_i \in U_i$, $v_j \in U_j$ be the first vertices explored by $\mathrm{DFS}(G_R)$ their respective component.



1. **v_i is explored before v_j**: No path $v_i \rightsquigarrow v_j \in G_R$ since v_j and v_i are in different components. Finish exploring v_i before beginning with v_j. Thus $f[v_j] > f[v_i]$.
2. **v_j is explored before v_i**: There exists a white path from v_j to v_i, so by White Path Prop. and Paren. Prop. $f[v_i] < f[v_j]$.

$\square$