Linear Time Algorithm for Update Games

DIMAP Seminar

University of Warwick, UK

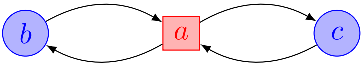
25 April 2017

Carlo Comin

On a joint ongoing work with *Romeo Rizzi*

University of Verona, Italy

from Graphs to Arenas



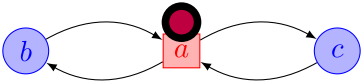
Arena

An *arena* $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is a finite directed graph $G^\Gamma = (V, A)$ whose vertices are divided into two classes, i.e., $V = V_\Box \cup V_\bigcirc$.

$G^\Gamma = (V, A)$:

- has no sink vertices;
- has no loops nor parallel arcs;
- is not required to be a bipartite graph on colour classes $V_\Box$ and $V_\bigcirc$.

Infinite Pebble Games

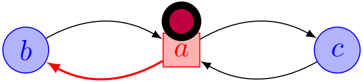


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

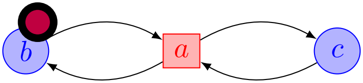


Game

A *game* on $\Gamma = (V, A, \langle V_\square, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

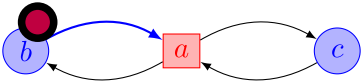


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

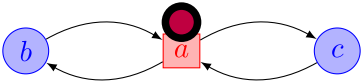


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

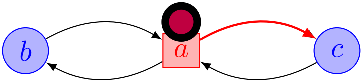


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

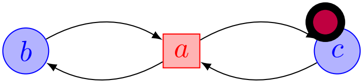


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

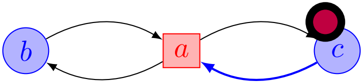


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

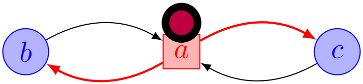


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ...

Infinite Pebble Games

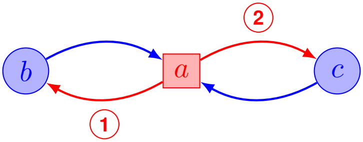


Game

A *game* on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one.

- Initially, the pebble is located on some $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player p chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on v';
- repeat rounds ad infinitum ... $(abac)^+$ ...

Infinite Pebble Games



Play, Strategy, Outcome Play

- A *play* is an infinite path $\pi = v_0 v_1 v_2 \ldots \in V^\omega$ such that $(v_i, v_{i+1}) \in A$;
- A *strategy* for Player p, where $p \in \{\Box, \bigcirc\}$, is a map,

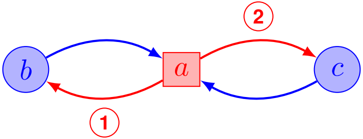
$$\sigma_p : V^* \times V_p \to V, \text{ such that } (v, \sigma_p(\pi', v)) \in A,$$

  for every finite path $\pi' v$ in G^Γ where $v \in V_i$;
- Given two strategies $\sigma_\Box \in \Sigma^\Gamma_\Box$ and $\sigma_\bigcirc \in \Sigma^\Gamma_\bigcirc$, and some $v_s \in V$, the *outcome* play,

$$\rho_\Gamma(v_s, \sigma_\Box, \sigma_\bigcirc),$$

  is the (unique) play that starts at v_s and is consistent with both $\sigma_\Box$, $\sigma_\bigcirc$.

Update Games



Winning Condition

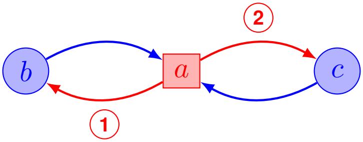
- Let $\mathrm{Inf}(\pi)$ be the set of all and only those vertices $v \in V$ that appear infinitely often in the play π,

$$\mathrm{Inf}(\pi) \triangleq \big\{ v \in V \mid \forall_{j \in \mathbb{N}} \, \exists_{k \in \mathbb{N}, k > j} \text{ such that } v_k = v \big\}.$$

- Player $\Box$ wins the Update Game Γ *iff* all vertices are visited infinitely often,

$$\exists_{\sigma_\Box \in \Sigma^\Gamma_\Box} \forall_{\sigma_\bigcirc \in \Sigma^\Gamma_\bigcirc} \forall_{v_s \in V} \, \mathrm{Inf}\big(\rho_\Gamma(v_s, \sigma_\Box, \sigma_\bigcirc)\big) = V.$$

Update Games



Main facts

- Firstly studied by:
  - ★ [Dinneen, Khoussainov, Update Networks and Their Routing Strategies. WG 2000];
- $O(mn)$ time algorithm [Dinneen, Khoussainov. WG 2000];
- Routing strategies are not positional.
- Basic subtask for solving Explicit Müller Games in polynomial time, as in:
  - ★ [Horn, Explicit Müller Games are PTIME. FSTTCS 2008];

Update Games

$\square$-Attractor

- $\texttt{Reach}^{\Gamma}_{\square}(v,0) \triangleq \{v\}, \forall_{v\in V}$;
- $$\texttt{Reach}^{\Gamma}_{\square}(v,i) \triangleq \{u \in V_{\square} \mid \exists_{(u,u')\in A}\; u' \in \bigcup_{j=0}^{i-1} \texttt{Reach}^{\Gamma}_{\square}(v,j)\} \cup$$ $$\cup \{u \in V_{\bigcirc} \mid \forall_{(u,u')\in A}\; u' \in \bigcup_{j=0}^{i-1} \texttt{Reach}^{\Gamma}_{\square}(v,j)\}, \forall_{v\in V} \forall_{i>0}.$$
- $\texttt{Attr}^{\Gamma}_{\square}(v) \triangleq \bigcup_{i\geq 0} \texttt{Reach}^{\Gamma}_{\square}(v,i)$.

$O(mn)$ Time Algorithm [Dinneen, Khoussainov. WG 2000]

- Compute $\texttt{Attr}^{\Gamma}_{\square}(v)$ for each $v \in V$;
- If $\exists_{v\in V} \texttt{Attr}^{\Gamma}_{\square}(v) \neq V$, return NO;
- Otherwise, return YES.

Linear Time: what can be done ?

Note

If G^Γ is not *strongly-connected* as a graph, then return `NO`.

Alternating Strongly Connected Components

Say $u, v \in V$ are *a-strongly-connected* if $u \in \mathtt{Attr}^\Gamma_\Box(v) \wedge v \in \mathtt{Attr}^\Gamma_\Box(u)$.

$$\mathtt{Attr}^\Gamma_\Box(a) = \{a, b, c\}, \mathtt{Attr}^\Gamma_\Box(b) = \{b\}, \mathtt{Attr}^\Gamma_\Box(c) = \{a, c\}.$$

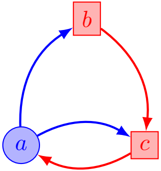
Linear Time: what can be done ?

Note

If G^Γ is not *strongly-connected* as a graph, then return `NO`.

Alternating Strongly Connected Components

Say $u, v \in V$ are *a-strongly-connected* if $u \in \texttt{Attr}^{\Gamma}_{\Box}(v) \wedge v \in \texttt{Attr}^{\Gamma}_{\Box}(u)$.



$$\texttt{Attr}^{\Gamma}_{\Box}(a) = \{a, b, c\}, \texttt{Attr}^{\Gamma}_{\Box}(b) = \{b\}, \texttt{Attr}^{\Gamma}_{\Box}(c) = \{a, c\}.$$

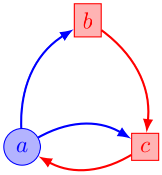
Linear Time: what can be done ?

Note

If G^{Γ} is not *strongly-connected* as a graph, then return `NO`.

Alternating Strongly Connected Components

Say $u, v \in V$ are *a-strongly-connected* if $u \in \texttt{Attr}^{\Gamma}_{\Box}(v) \wedge v \in \texttt{Attr}^{\Gamma}_{\Box}(u)$.



$$\mathcal{D}_{\text{a-scc}} = \Big\{\{a, c\}, \{b\}\Big\}.$$

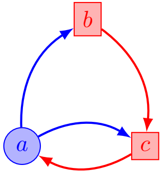
Linear Time: what can be done ?

Note

If G^{Γ} is not *strongly-connected* as a graph, then return $\texttt{NO}$.

Alternating Strongly Connected Components

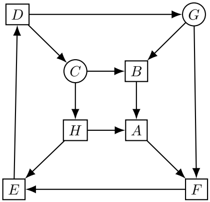
Say $u, v \in V$ are *a-strongly-connected* if $u \in \texttt{Attr}^{\Gamma}_{\Box}(v) \wedge v \in \texttt{Attr}^{\Gamma}_{\Box}(u)$.



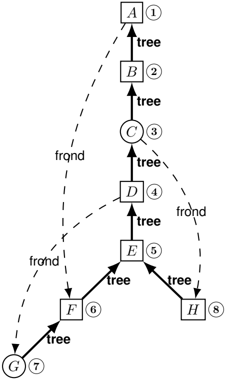
Deciding Update Games (Sketch)

If Γ is a-strongly-connected, i.e., $\mathcal{D}_{\text{a-scc}} = \{V\}$, return $\texttt{YES}$; otherwise, $\texttt{NO}$.

Depth-First-Search (rev-DFS)



(a) Arena Γ.



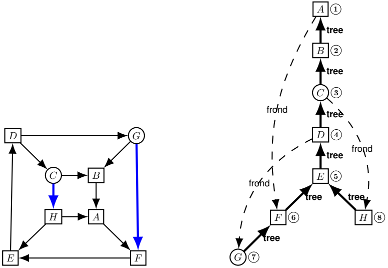
(b) The rev-palm-tree generated by rev-DFS, with indices of vertices.

1.(B, A)	7.(G, F)
2.(C, B)	8.(D, G)
3.(D, C)	9.(H, E)
4.(E, D)	10.(C, H)
5.(F, E)	11.(G, B)
6.(A, F)	12.(H, A)

(c) The order of arcs' exploration.

Figure: An arena Γ (a), and a rev-palm-tree (b), generated by rev-DFS on G^{Γ} (c).

Reverse Palm-Tree (rev-palm-tree)



If we run a DFS on G^Γ, then ...

- Arcs can be partitioned into: *tree*, *fronds*, *cross* (and *forward*).
- If G^Γ is strongly-connected, the rev-palm-tree is a spanning tree.
- Each vertex is reachable in G^Γ from any of its descendants,
  - however, e.g. $\texttt{Attr}^{\Gamma}_{\Box}(B) = \{B\}$; thus, rev-palm-trees are not (yet) *certificates* of a-reachability.

Safe-Reachability

Safe-Reachability

Given an arena Γ on vertex set V, let $U \subseteq V$ and $u, v \in U$. We say that v is *U-safe-reachable* from u (*i.e.* $u \overset{U}{\rightsquigarrow} v$) when $\exists \sigma_\Box \in \Sigma_\Box^\Gamma$ such that $\forall \sigma_\bigcirc \in \Sigma_\bigcirc^\Gamma$:

[a-reachability] $\quad v \in \mathtt{Occ}\big(\rho_\Gamma(u, \sigma_\Box, \sigma_\bigcirc)\big)$; and,

[safety] $\quad \mathtt{Occ}\big(\rho_\Gamma(u, \sigma_\Box, \sigma_\bigcirc, v)\big) \subseteq U$.

where for any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the *occurrence set* of p is:

$$\mathtt{Occ}(p) \triangleq \{v \in V \mid v \text{ appears in } p\};$$

and $\rho_\Gamma(u, \sigma_\Box, \sigma_\bigcirc, v)$ is the shortest prefix of $\rho_\Gamma(u, \sigma_\Box, \sigma_\bigcirc)$ ending with v.

Alternating Depth-First-Search (a-DFS)

Perform a DFS on the arena Γ in such a way as to always preserve $V_\mathcal{T}$-safe-reachability within the palm-tree $\mathcal{T}$ that is under construction. (Invariant Property)

- *i.e.*, $v \in V_\bigcirc$ joins the palm-tree $\mathcal{T}$ as soon as *all* of its outgoing neighbours $v' \in N_v^{\text{out}}$ have already did it; and its parent π_v is the LCA of N_v^{out} in $\mathcal{T}$.

Safe-Reachability

Safe-Reachability

Given an arena Γ on vertex set V, let $U \subseteq V$ and $u, v \in U$. We say that v is U*-safe-reachable* from u (*i.e.* $u \overset{U}{\rightsquigarrow} v$) when $\exists \sigma_\square \in \Sigma^\Gamma_\square$ such that $\forall \sigma_\bigcirc \in \Sigma^\Gamma_\bigcirc$:

[a-reachability] $v \in \texttt{Occc}\big(\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc)\big)$; and,

[safety] $\texttt{Occ}\big(\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc, v)\big) \subseteq U$.

where for any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the *occurrence set* of p is:

$$\texttt{Occ}(p) \triangleq \{v \in V \mid v \text{ appears in } p\};$$

and $\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc, v)$ is the shortest prefix of $\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc)$ ending with v.

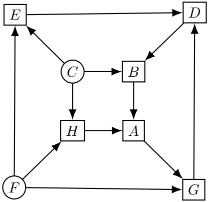
Alternating Depth-First-Search (a-DFS)

Perform a DFS on the arena Γ in such a way as to always preserve $V_\mathcal{T}$-safe-reachability within the palm-tree $\mathcal{T}$ that is under construction. (Invariant Property)

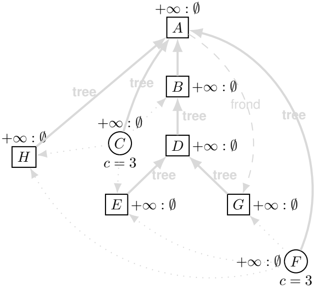
- *i.e.*, $v \in V_\bigcirc$ joins the palm-tree $\mathcal{T}$ as soon as *all* of its outgoing neighbours $v' \in N^{\text{out}}_v$ have already did it; and its parent π_v is the LCA of N^{out}_v in $\mathcal{T}$.

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

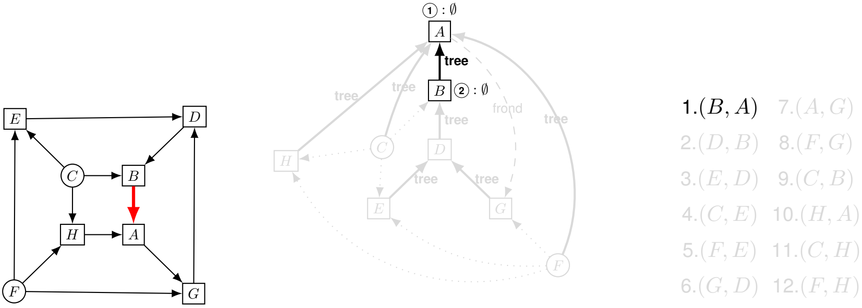
1. (B, A)
2. (D, B)
3. (E, D)
4. (C, E)
5. (F, E)
6. (G, D)
7. (A, G)
8. (F, G)
9. (C, B)
10. (H, A)
11. (C, H)
12. (F, H)

(c) The order of arcs' exploration.

Figure: An arena (a), and an a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



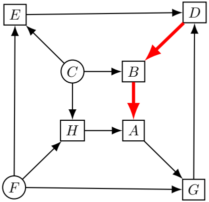
(a) An arena Γ.

(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

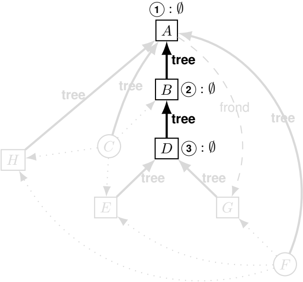
(c) The order of arcs' exploration.

Figure: An arena (a), and an a-palm-tree (b), generated by a-DFS (c).

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

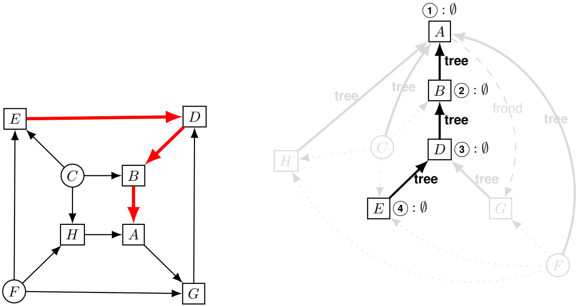
1.(B, A) 7.(A, G)
2.(D, B) 8.(F, G)
3.(E, D) 9.(C, B)
4.(C, E) 10.(H, A)
5.(F, E) 11.(C, H)
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\mathtt{idx}[v] \in \mathbb{N}$, $\mathtt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.

(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

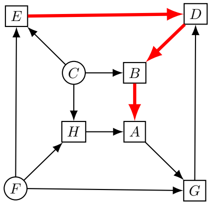
1.(B, A)
2.(D, B)
3.(E, D)
4.(C, E)
5.(F, E)
6.(G, D)
7.(A, G)
8.(F, G)
9.(C, B)
10.(H, A)
11.(C, H)
12.(F, H)

(c) The order of arcs' exploration.

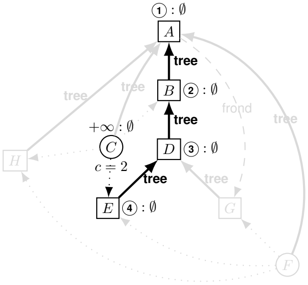
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

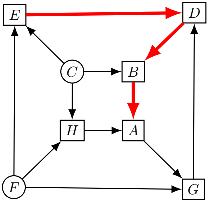
1.(B, A) 7.(A, G)
2.(D, B) 8.(F, G)
3.(E, D) 9.(C, B)
4.(C, E) 10.(H, A)
5.(F, E) 11.(C, H)
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

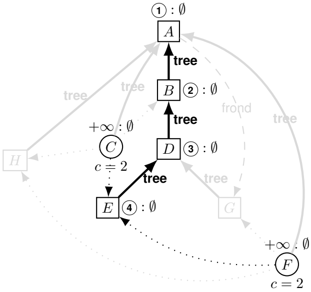
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

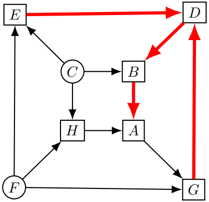
1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

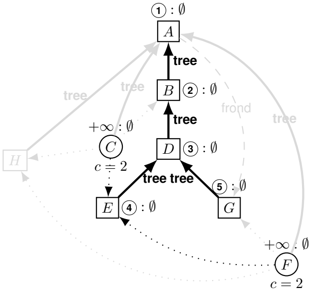
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

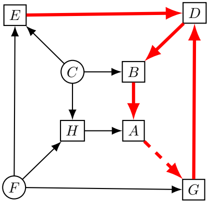
1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

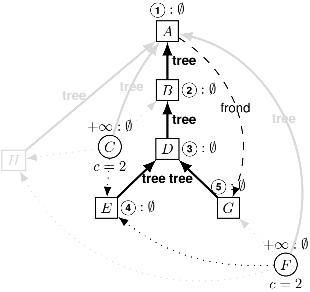
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_\bigcirc$; and, for every $u \in V_\bigcirc$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

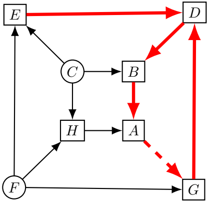
1.(B, A)
2.(D, B)
3.(E, D)
4.(C, E)
5.(F, E)
6.(G, D)
7.(A, G)
8.(F, G)
9.(C, B)
10.(H, A)
11.(C, H)
12.(F, H)

(c) The order of arcs' exploration.

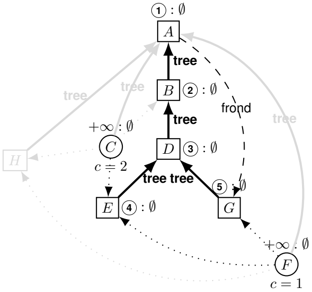
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

1.(B, A) 7.(A, G)

2.(D, B) 8.(F, G)

3.(E, D) 9.(C, B)

4.(C, E) 10.(H, A)

5.(F, E) 11.(C, H)

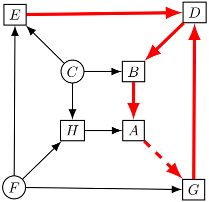
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

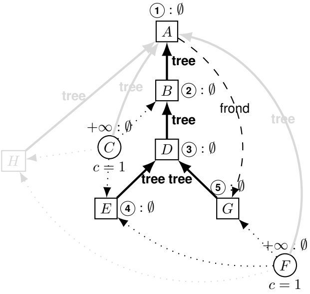
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

1.(B, A) 7.(A, G)

2.(D, B) 8.(F, G)

3.(E, D) 9.(C, B)

4.(C, E) 10.(H, A)

5.(F, E) 11.(C, H)

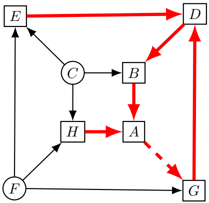
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

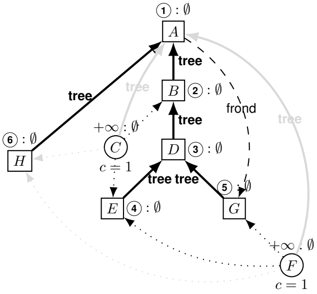
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

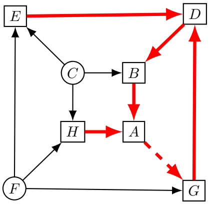
1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

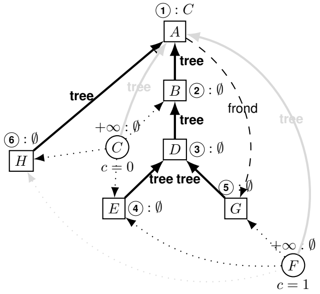
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

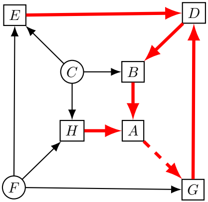
1.(B, A) 7.(A, G)
2.(D, B) 8.(F, G)
3.(E, D) 9.(C, B)
4.(C, E) 10.(H, A)
5.(F, E) 11.(C, H)
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

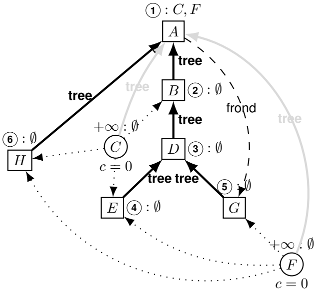
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

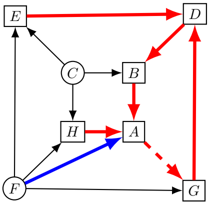
1.(B, A) 7.(A, G)
2.(D, B) 8.(F, G)
3.(E, D) 9.(C, B)
4.(C, E) 10.(H, A)
5.(F, E) 11.(C, H)
6.(G, D) 12.(F, H)

(c) The order of arcs' exploration.

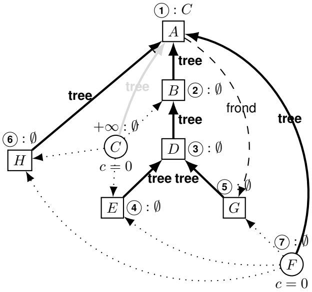
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

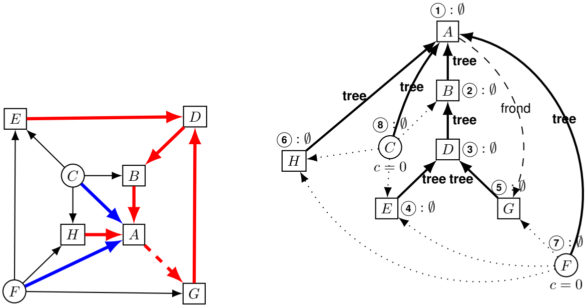
1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.

(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

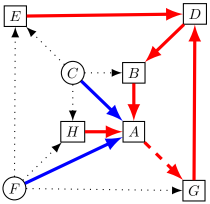
1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

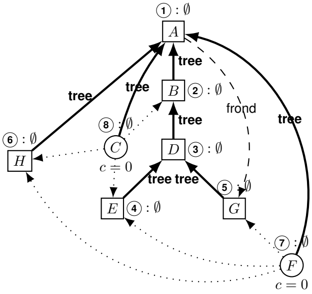
Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Alternating Depth-First-Search: an example

For every $v \in V$, $\texttt{idx}[v] \in \mathbb{N}$, $\texttt{ready\_St}[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.



(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.

1.(B, A)	7.(A, G)
2.(D, B)	8.(F, G)
3.(E, D)	9.(C, B)
4.(C, E)	10.(H, A)
5.(F, E)	11.(C, H)
6.(G, D)	12.(F, H)

(c) The order of arcs' exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$;
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:

1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$;
2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N_{\text{A}}^{\text{out}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\mathtt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\mathtt{low\_ready}[v] = \min\left\{\mathtt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \mathtt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\mathtt{Union}(u, v)$.
- When the condition $\mathtt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\mathtt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\mathtt{idx}[x] = \mathtt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\mathtt{Find}(\mathtt{low\_}v)$;
  3. if γ is still $\mathtt{active}$, execute $\mathtt{ready\_St}[\gamma].\mathtt{push}(u)$.

Proposition

If γ is still $\mathtt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v \in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min \left\{ \texttt{idx}[u] \mid u \in N_{\mathsf{A}}^{\mathsf{out}}(v) \right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\mathsf{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\mathsf{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\mathsf{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on V, i.e., init $\mathcal{D}.\texttt{MakeSet}(v)\ \forall_{v\in V}$;
- For all $v \in V_{\bigcirc}$, the following invariant is maintained during the a-DFS:

$$\texttt{low\_ready}[v] = \min\left\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\text{A}}(v)\right\}.$$

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \texttt{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\texttt{Union}(u, v)$.
- When the condition $\texttt{c}[u] = 0$ is met, for some $u \in V_{\bigcirc}$:
  1. let $\texttt{low\_}v$ be the unique $x \in N^{\text{out}}(u)$ with $\texttt{idx}[x] = \texttt{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low\_}v)$;
  3. if γ is still $\texttt{active}$, execute $\texttt{ready\_St}[\gamma].\texttt{push}(u)$.

Proposition

If γ is still $\texttt{active}$, the LCA of $N^{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really γ.

Strong Post-Order Path-Compression Systems

Theorem (Loebl, Nesetril, 1997)

Let S be a *strong post-order path compression system*, then $|S| \leq 5n$.

Loebl, Nesetril. Linearity and unprovability of set union problem strategies. Journal of Algorithms, 23, 2 (1997), 207 – 220.

Proposition

The sequence of path compressions performed by the a-DFS is a Loebl-Nesetril strong post-order path compression system.

- Then the a-DFS halts in linear time.

Strong Post-Order Path-Compression Systems

Theorem (Loebl, Nesetril, 1997)

Let S be a *strong post-order path compression system*, then $|S| \leq 5n$.

Loebl, Nesetril. Linearity and unprovability of set union problem strategies. Journal of Algorithms, 23, 2 (1997), 207 – 220.

Proposition

The sequence of path compressions performed by the a-DFS is a Loebl-Nesetril strong post-order path compression system.

- Then the a-DFS halts in linear time.

Safe-Strongly-Connected Components

Safe-Strongly-Connectedness

Say $U \subseteq V$ is *safe-strongly-connected (s-SC)* when for every $(u, v) \in U \times U$ $\sigma_\Box : u \stackrel{U}{\rightsquigarrow} v$ for some $\sigma_\Box \in \Sigma_\Box^\Gamma$.

Safe-Strongly-Connected Components

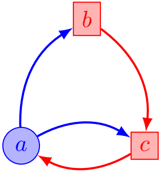
Consider the following binary relation $\sim_{\text{s-scc}}$ on V:

$$\sim_{\text{s-scc}} \triangleq \Big\{(u, v) \in V \times V \mid \exists_{U \subseteq V} \text{ s.t. } U \text{ is s-SC and } u, v \in U\Big\}.$$

- It holds that $\sim_{\text{s-scc}}$ is an equivalence relation on V.

Equivalence classes of $\sim_{\text{s-scc}}$ are *safe-strongly-connected components* of Γ.

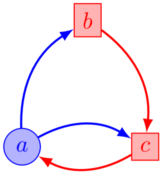
s-SCC $\sqsubseteq$ a-SCC



$$\mathcal{D}_{\text{a-scc}} = \Big\{\{a, c\}, \{b\}\Big\};$$

$$\mathcal{D}_{\text{s-scc}} = \Big\{\{a\}, \{b\}, \{c\}\Big\}.$$

s-SCC $\sqsubseteq$ a-SCC

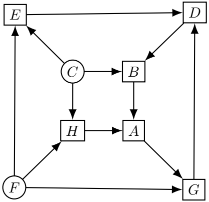


$$\mathcal{D}_{\text{a-scc}} = \Big\{\{a, c\}, \{b\}\Big\};$$

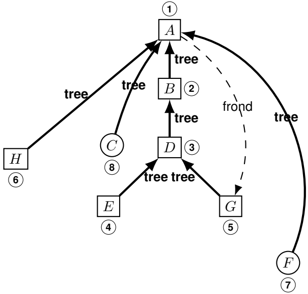
$$\mathcal{D}_{\text{s-scc}} = \Big\{\{a\}, \{b\}, \{c\}\Big\}.$$

Subtrees, Roots, and LowLinks

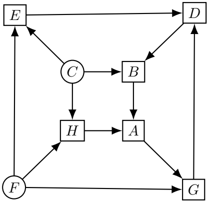
- Each s-SCC $\mathcal{C}$ induces a *subtree* $T_{\mathcal{C}}$ in one of the a-palm-trees constructed by the a-DFS.



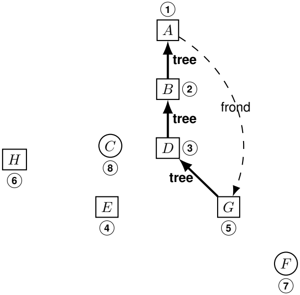
(a) An arena Γ.



(b) The a-palm-tree generated by a-DFS rooted at A, with indices of vertices and labelled arcs.



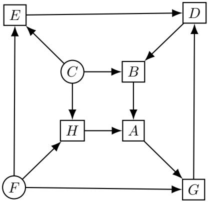
(c) An arena Γ.



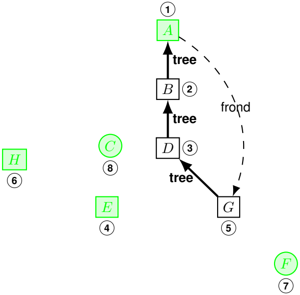
(d) The s-SCCs of Γ.

Subtrees, Roots, and LowLinks

- Each s-SCC $\mathcal{C}$ induces a *subtree* $T_{\mathcal{C}}$ in one of the a-palm-trees constructed by the a-DFS.
- The problem of computing the s-SCCs reduces to that of finding the *roots* of the s-SCCs in $\mathcal{T}$, as the classical problem of finding the SCCs of a directed graph reduced to that of finding the roots of the SCCs.



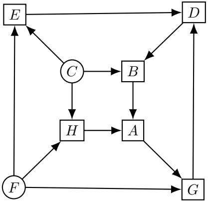
(e) An arena Γ.



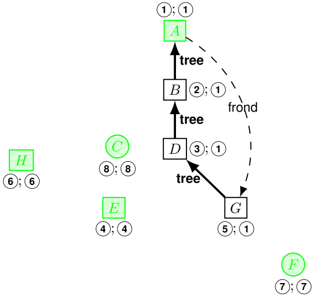
(f) The roots of the s-SCCs of Γ.

Subtrees, Roots, and LowLinks

- Each s-SCC $\mathcal{C}$ induces a *subtree* $T_{\mathcal{C}}$ in one of the a-palm-trees constructed by the a-DFS.
- The problem of computing the s-SCCs reduces to that of finding the *roots* of the s-SCCs in $\mathcal{T}$, as the classical problem of finding the SCCs of a directed graph reduced to that of finding the roots of the SCCs.
- We have identified a simple test to determine if a vertex is the root of a s-SCCs. It is based on an *a-lowlink* indexing, similar to the lowlink calculation performed by Tarjan's SCC algorithm.
  - i.e., $v \in V$ is the root of some s-SCCs of Γ iff a-lowlink$(v) = \mathtt{idx}[v]$.



(g) An arena Γ.



(h) The idx and a-lowlinks indices.

The $\texttt{a-lowlink}[v]$ is the smallest index of any vertex u which is in the same s-SCC as v and such that u can reach v by traversing: at most one frond or cross arc, and then zero or more tree arcs.

Procedure *compute-s-SCC*($\mathcal{A}$)

input : An arena $\mathcal{A} = (V, A, (V_{\bigcirc}, V_{\square}))$.

output: The s-SCC of $\mathcal{A}$.

foreach $u \in V$ **do**
- $\texttt{idx}[u] \leftarrow +\infty$;
- $\texttt{a-lowlink}[u] \leftarrow +\infty$;
- $\texttt{on\_Stack}[u] \leftarrow \texttt{false}$;
- $\mathcal{D}.\texttt{make\_set}(u)$;
- $\texttt{ready\_St}[u] \leftarrow \emptyset$;
- **if** $u \in V_{\bigcirc}$ **then**
  - $\texttt{low\_ready}[u] \leftarrow +\infty$;
  - $\texttt{cnt}[u] \leftarrow |N^{\text{out}}_{\mathcal{A}}(u)|$;

$\texttt{next\_idx} \leftarrow 1$; $\texttt{St} \leftarrow \emptyset$;

foreach $u \in V_{\square}$ **do**
- **if** $idx[u] = +\infty$ **then**
  - $\texttt{s-SCC-visit}(u, \mathcal{A})$;

foreach $u \in V_{\bigcirc}$ **do**
- **if** $idx[u] = +\infty$ **then**
  - $\texttt{idx}[u] \leftarrow \texttt{next\_idx}$;
  - $\texttt{next\_idx} \leftarrow \texttt{next\_idx} + 1$;
  - $\texttt{ta\_lowlink}[u] \leftarrow \texttt{idx}[u]$;

```
Procedure s-SCC-visit(v, 𝒜)
    input : A vertex v ∈ V.
    idx[v] ← next_idx;
    a-lowlink[v] ← next_idx;
    next_idx ← next_idx + 1;
    St.push(v);
    on_Stack[v] ← true
    // Check the in-neighbourhood of v
    foreach u ∈ N^in_𝒜(v) do
        if idx[u] = +∞ then
            if u ∈ V_□ then
                s-SCC-visit(u, 𝒜);
                a-lowlink[v] ← min(a-lowlink[v], a-lowlink[u]);
                𝒟.Union(u, v);
            else
                low_ready[u] ← min(low_ready[u], idx[v]);
                cnt[u] ← cnt[u] − 1;
                if cnt[u] = 0 then
                    low_v ← the unique x such that idx[x] = low_ready[u];
                    γ ← 𝒟.find(low_v);
                    if on_Stack[γ] = true then
                        ready_St[γ].push(u);
        else if on_Stack[u] = true then
            a-lowlink[v] ← min(a-lowlink[v], idx[u]);
```

```
// Check the ready-stack of v, i.e., ready_St[v]
while ready_St[v] ≠ ∅ do
    u ← ready_St[v].pop(); // u ∈ V○
    if ∀(x ∈ N^out_A(u)) on_Stack[x] = true then
        SCCs-visit(u, A);
        a-lowlink[v] ← min(a-lowlink[v], a-lowlink[u]);
        D.union(u, v);

// Check whether a new s-SCC has to be constructed and printed to output
if a-lowlink[v] = idx[v] then
    C ← ∅;
    repeat
        u ← St.pop();
        on_Stack[u] ← false;
        add u to C;
    until u = v
    output(C);
```

Deciding Update Games

If $\mathcal{C} = V$, return YES; otherwise, NO.

```
// Check the ready-stack of v, i.e., ready_St[v]
while ready_St[v] ≠ ∅ do
    u ← ready_St[v].pop(); // u ∈ V○
    if ∀(x ∈ N_A^out(u)) on_Stack[x] = true then
        SCCs-visit(u, A);
        a-lowlink[v] ← min(a-lowlink[v], a-lowlink[u]);
        D.union(u, v);

// Check whether a new s-SCC has to be constructed and printed to output
if a-lowlink[v] = idx[v] then
    C ← ∅;
    repeat
        u ← St.pop();
        on_Stack[u] ← false;
        add u to C;
    until u = v
    output(C);
```

Deciding Update Games

If $\mathcal{C} = V$, return `YES`; otherwise, `NO`.

Thank you for the attention