# Depth First Search

Data Structures and Algorithms (094224)

Yuval Emek

Winter 2022/23

# The depth first search algorithm

*depth first search (DFS)* ⟨סריקה לעומק⟩

- Complements BFS as a basic graph exploration algorithm
- Input: A (di)graph $G = (V, E)$
- Explores $G$, discovering all vertices
  - After $u$ is discovered, each edge $(u, v) \in E$ is explored, recursively continuing the discovery process at $v$
- Returns:
  - A *DFS forest*
    - A collection of rooted trees that spans the whole of $V$
    - $u$ is the parent of $v$ if $v$ was discovered by exploring the edge $(u, v) \in E$
  - Unique *timestamps* for each vertex
- The DFS forest and the timestamps encode a lot of information about $G$'s structure

# Data structures

- $G = (V, E)$ represented using adjacency lists
- Each vertex $v \in V$ maintains additional attributes:
  - $v.color$ = the discovery status of $v$
    - white = $v$ is still undiscovered
    - gray = $v$ has been discovered but it still has unexplored incident edges
    - black = $v$ has been discovered and all its incident edges have been explored
  - $v.\pi$ = the parent of $v$ in the DFS tree
  - $v.d$ = the *discovery time* ⟨זמן גילוי⟩ of $v$
  - $v.f$ = the *retraction time* ⟨זמן נסיגה⟩ of $v$
    - After all incident edges have been explored
- An integer variable *time*

# Pseudocode

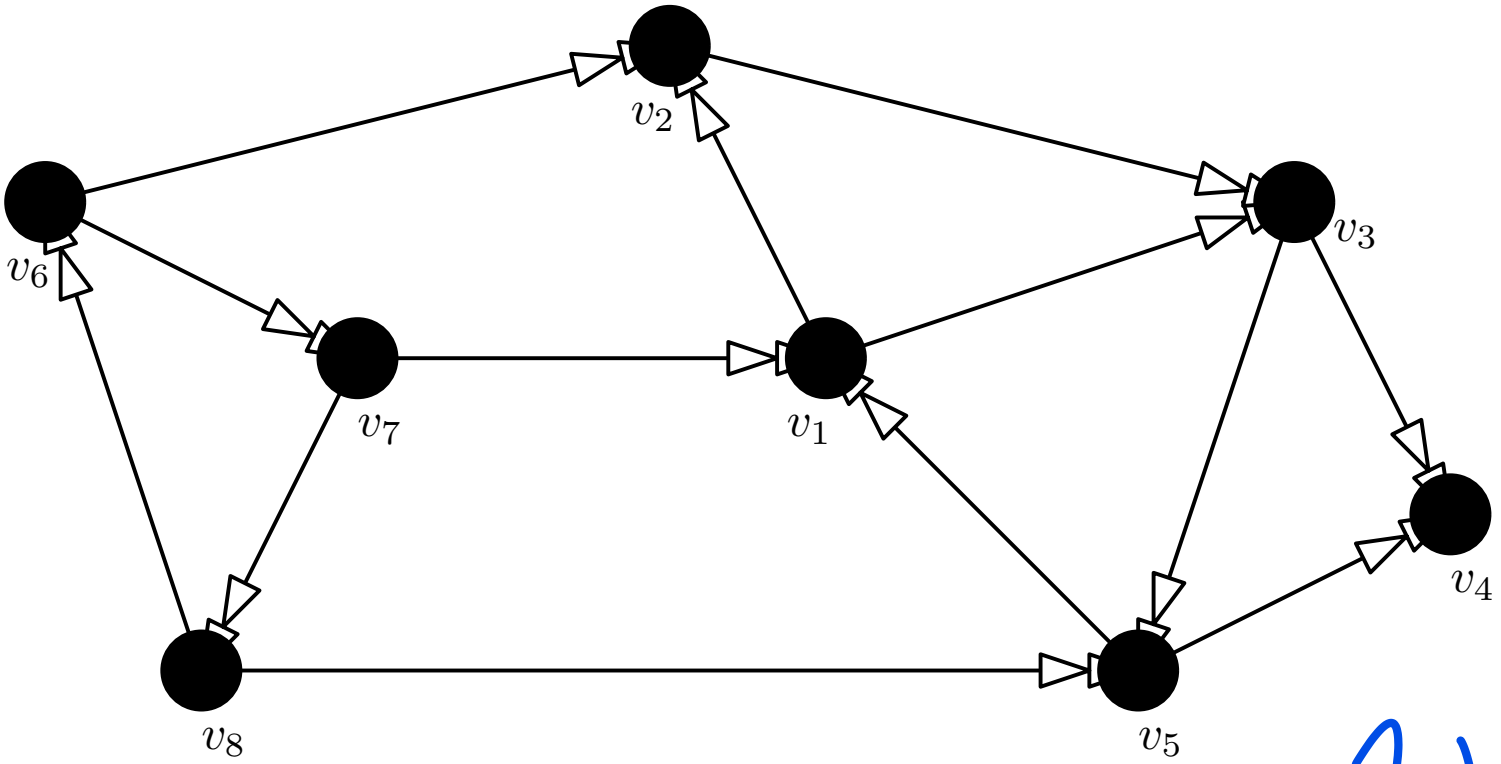
DFS($G$)

```
1: for all u ∈ G.V do
2:     u.color = white
3:     u.π = NIL
4: time = 0
5: for all u ∈ G.V do
6:     if u.color == white then
7:         DFS_Visit(G, u)
```

DFS_Visit($G, u$)

```
1: time = time + 1
2: u.d = time
3: u.color = gray
4: for all v ∈ G.Adj[u] do
5:     if v.color == white then
6:         v.π = u
7:         DFS_Visit(G, v)
8: u.color = black
9: time = time + 1
10: u.f = time
```

# Illustrating a DFS execution




| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ |
|---|---|---|---|---|---|---|---|---|
| $d$ | 1 | 2 | 3 | 4 | 6 | 11 | 12 | 13 |
| $f$ | 10 | 9 | 8 | 5 | 7 | 16 | 15 | 14 |
| $\pi$ | — | $v_1$ | $v_2$ | $v_3$ | $v_3$ | — | $v_6$ | $v_7$ |

# Run-time analysis

- Initialization (lines 1–4 of `DFS`) takes $O(n)$ time
- After initialization, DFS never colors a vertex white
- $u$ is colored gray in the beginning of `DFS_Visit`$(G, u)$ (line 3)
- $\Longrightarrow$ `DFS_Visit`$(G, u)$ is called exactly once for each $u \in V$
  - $O(n)$ time for all operations outside the for loop of lines 4–7 in `DFS_Visit`
- $\Longrightarrow$ Every adjacency list is traversed exactly once
- Total size of all adjacency lists is $O(m)$
- Each entry of each adjacency list accounts for $O(1)$ time
- $\Longrightarrow$ In total, the run-time of DFS is $O(n + m)$

1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification
2. Identifying strongly connected components
   - Correctness

# The DFS forest

- Define the undirected graph $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = V$
  - $E_\pi = \{(v.\pi, v) \mid v \in V, v.\pi \neq NIL\}$
- Mirrors the structure of (recursive) calls to `DFS_Visit`
  - $\Longrightarrow$ cycle free (a forest)
- Root the trees in $G_\pi$ at the vertices $v$ with $v.\pi = NIL$
  - Treat $G_\pi$ as a collection of rooted trees
  - Parent-child relations reflect the "direction" of the recursive calls to `DFS_Visit`
- $G_\pi$ is called the *DFS forest*

**Observation**

*$v$ is a descendant of $u \neq v$ in $G_\pi$ iff $v$ is discovered when $u$ is gray.*

- Follows from the structure of recursive calls ■

1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification
2. Identifying strongly connected components
   - Correctness

# Nested parenthesis

**Theorem (The parenthesis property)**

*For any two vertices $u, v \in V$, $u \neq v$, exactly one of the following three conditions hold:*

1. $[v.d, v.f] \subset [u.d, u.f]$ *and* $v$ *is a descendant of* $u$ *in* $G_\pi$;
2. $[u.d, u.f] \subset [v.d, v.f]$ *and* $u$ *is a descendant of* $v$ *in* $G_\pi$; *or*
3. $[u.d, u.f] \cap [v.d, v.f] = \emptyset$ *and* $u$ *and* $v$ *do not exhibit an ancestor-descendant relation in* $G_\pi$.

Why is it called the *parenthesis property* ⟨תכונת הסוגריים⟩?

# Proof of the parenthesis property

Without loss of generality

comparable in certain respects, typically in a way which makes clearer the nature of the things compared.

- Assume w.l.o.g. that $u.d < v.d$ (the case $v.d < u.d$ is analogous)
- Case $v.d < u.f$: $v$ was discovered while $u$ was gray
- $\Longrightarrow$ $v$ is a descendant of $u$ in $G_\pi$
- $v$ is colored black (line 8) before the discovery process returns to $u$
- $\Longrightarrow$ $v.f < u.f$
- Fits condition 1
- Case $v.d > u.f$: $v$ was discovered while $u$ was black
- $\Longrightarrow$ $v$ is not a descendant of $u$ in $G_\pi$
  - $v$ is clearly not an ancestor of $u$ in $G_\pi$ as it was discovered after $u$
- Fits condition 3 ■
- How come we didn't encounter condition 2? because of the assumption

1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification
2. Identifying strongly connected components
   - Correctness

# Enabling discovery

## Theorem (The white path property)

*$v$ is a descendant of $u$ in $G_\pi$ iff at time $u.d$, there is a $(u, v)$-path in $G$ containing only white vertices.*[a]

[a] Here, time $u.d$ refers to the time $u.d$ is set in line 2 of `DFS_Visit`$(G, u)$.

- Direction $\Longrightarrow$
- If $u = v$, then the assertion holds as $u$ is still white in line 2
- Assume that $v$ is a proper descendant of $u$ and consider $P = \langle u, \ldots, v.\pi.\pi, v.\pi, v \rangle$
- $P$ is a $(u, v)$-path in $G$
  - If $x.\pi = y$, then $(y, x) \in E$ (see line 6 of `DFS_Visit`$(G, y)$)
- By the parenthesis property, $u.d < w.d$ for every vertex $w \in P - \{u\}$
  - $\Longrightarrow$ $w$ is white at time $u.d$
- $\Longrightarrow$ path $P$ is white at time $u.d$

# The proof continues

- Direction $\Longleftarrow$
- Let $P$ be a path emerging from $u$ such that all its vertices are white at time $u.d$
- Assume by contradiction that some vertex in $P$ does not end up being a descendant of $u$
- Let $w$ be the first such vertex along $P$
  - $w \neq u$
- Let $w'$ be the vertex preceding $w$ in $P$

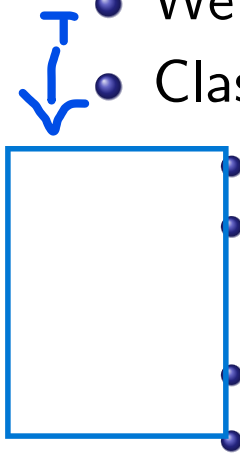
coming before something in order, position, or time

  - $w'$ is a descendant of $u$ by the choice of $w$
- By the parenthesis property, $w'.f \leq u.f$
  - Weak inequality because $w'$ may be $u$
- $w$ is discovered after $u$ is discovered and before $w'$ is retracted
- $\Longrightarrow u.d < w.d < w'.f \Longrightarrow u.d < w.d < u.f$
  - $w$ is discovered while $u$ is gray
- $\Longrightarrow w$ is a descendant of $u$ $_{(\rightarrow\leftarrow)}$ ■

1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification
2. Identifying strongly connected components
   - Correctness

# Four types of edges

there is an explanation in https://www.geeksforgeeks.org/tree-back-edge-and-cross-edges-in-dfs-of-graph/



- We use $G_\pi$ to partition the edges of $G$ into 4 types
- Classify edge $(u, v) \in E$ as
  - Tree edge if it is included in $G_\pi$ edges that appear in specific one forest
  - Back edge if $v$ is an ancestor of $u$ in $G_\pi$
    - May be a self-loop (if $G$ is directed)
  - Forward edge if not a tree edge and $v$ is a (proper) descendant of $u$
  - Crossing edge in all other cases
- Resolving ambiguity for undirected graphs: classify $(u, v) = (v, u)$ according to the direction explored first by the algorithm

# Classification in real-time

- When DFS explores edge $e = (u, v)$ (for the first time) in lines 4–7 of `DFS_Visit(G, u)`, it has enough information to classify it:
  - If $v.color = white$, then $e$ is a tree edge
    - $v.\pi \leftarrow u$ in line 6
  - If $v.color = gray$, then $e$ is a back edge
    - $u$ has been discovered while $v$ is gray
  - If $v.color = black$ and $u.d < v.d$, then $e$ is a forward edge
    - Implies $u.d < v.d < v.f < u.f$
  - If $v.color = black$ and $u.d > v.d$, then $e$ is a crossing edge
    - $v$ must have been retracted before $u$ was discovered

# Edge classification in undirected graphs

**Lemma**

*When DFS runs on an undirected graph, every edge is classified either as a tree edge or as a back edge.*

- Proved in the tutorial

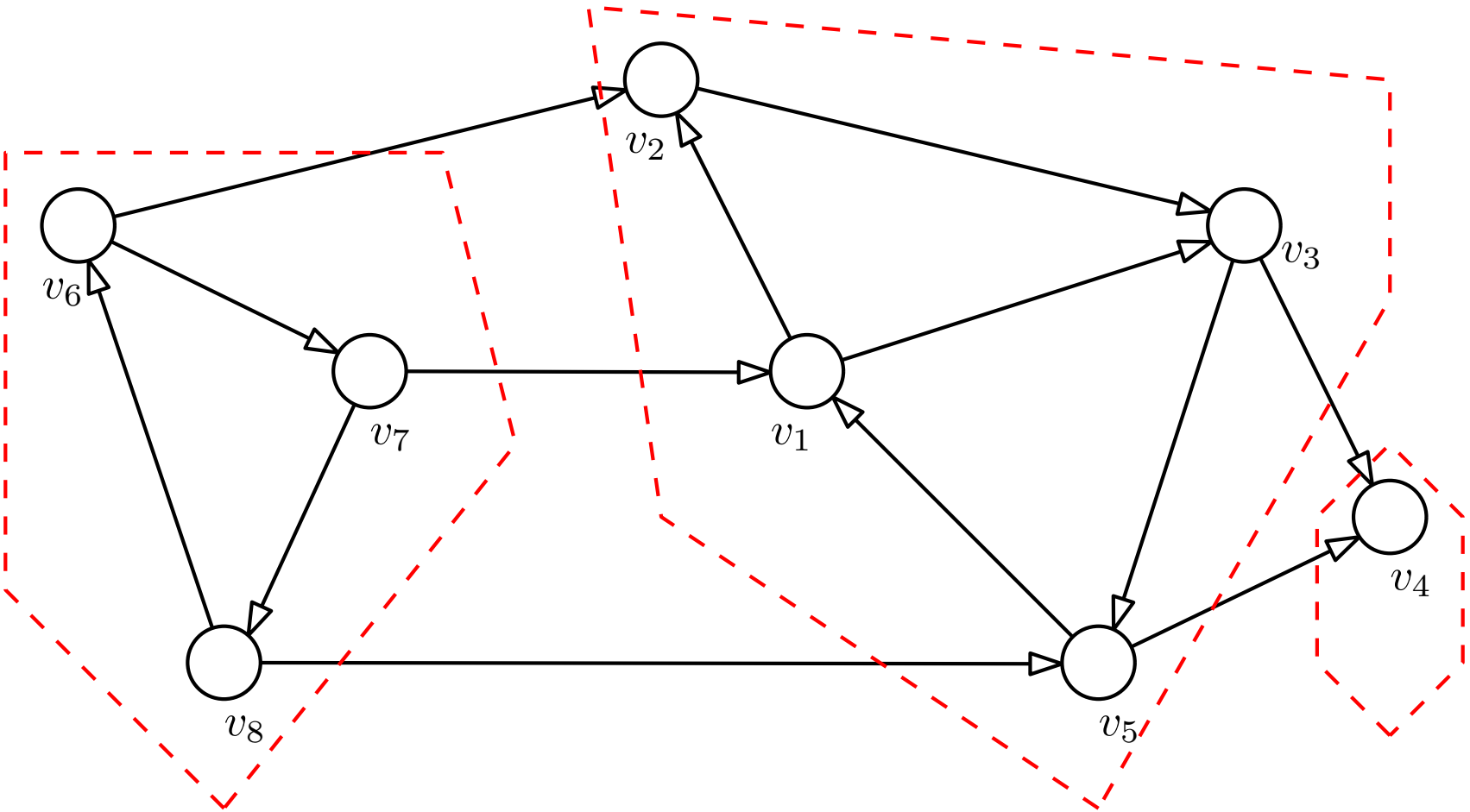
1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification

2. Identifying strongly connected components
   - Correctness

# Outline

- Digraph $G = (V, E)$
  - Adjacency lists representation
- Recall: the strongly connected components (SCCs) of $G$ are the equivalence classes of the mutually reachable relation on $V$
- Alternatively: $C \subseteq V$ is a SCC if $G(C)$ is strongly connected and $G(C')$ is not strongly connected for any $C \subset C' \subseteq V$
  - $G(U)$ = the subgraph of $G$ induced by $U \subseteq V$
- We are looking for an efficient algorithm that partitions a given digraph into its SCCs
- Will design the desired algorithm based on DFS

# Example: partition a digraph into SCCs



# The transpose graph

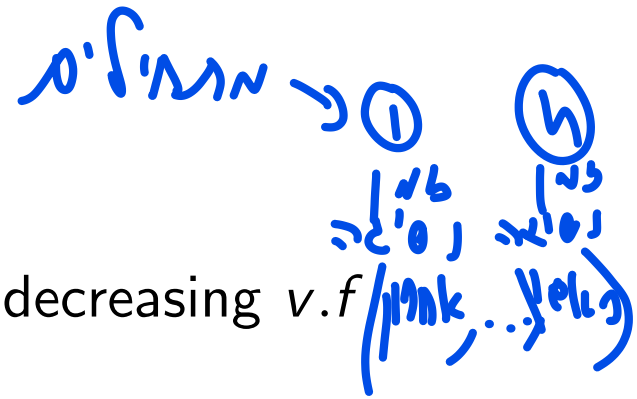
- Define the transpose digraph $\overleftarrow{G} = (\overleftarrow{V}, \overleftarrow{E})$ of $G$
  - $\overleftarrow{V} = V$
  - $\overleftarrow{E} = \{(x, y) \mid (y, x) \in E\}$
- The transpose digraph can be constructed in time $O(m + n)$
  - Adjacency lists representation
  - How? תרגיל.

**Observation**

*$G$ and $\overleftarrow{G}$ have the same SCCs.*

- $u$ reachable form $v$ in $G$ iff $v$ reachable from $u$ in $\overleftarrow{G}$
- mutually-reachable in $G$ iff mutually-reachable in $\overleftarrow{G}$ ■

# The algorithm



SCC($G$)

1: run DFS($G$)
2: $\psi$ = the sequence of vertices $v \in V$ in order of decreasing $v.f$
3: construct the transpose digraph $\overleftarrow{G}$ of $G$
4: run DFS($\overleftarrow{G}$), processing the vertices in line 5 in $\psi$-order
5: output the vertices of each tree in the DFS forest as a SCC

- Some missing implementation details
- Run-time:
  - $O(m+n)$ for each DFS invocation
  - Construct $\psi$ during first DFS invocation with same asymptotic run-time
  - $O(m+n)$ for the transpose digraph construction
  - $\Longrightarrow$ $O(m+n)$ in total

1. DFS properties
   - The parenthesis property
   - The white path property
   - Edge classification

2. Identifying strongly connected components
   - Correctness

# A refining partition

## Lemma

*Consider some digraph $G = (V, E)$ and let $G_\pi$ be the result of* $\texttt{DFS}(G)$*. If $u$ and $v$ belong to the same SCC in $G$, then $u$ and $v$ belong to the same rooted tree in $G_\pi$.*

- Let $C$ be the SCC of $u$ and $v$
- Let $x$ be the vertex that minimizes $x.d$ among all vertices in $C$
- At time $x.d$, there are white paths connecting $x$ to $u$ and $v$
- By the white path property, both $u$ and $v$ are descendants of $x$ ■

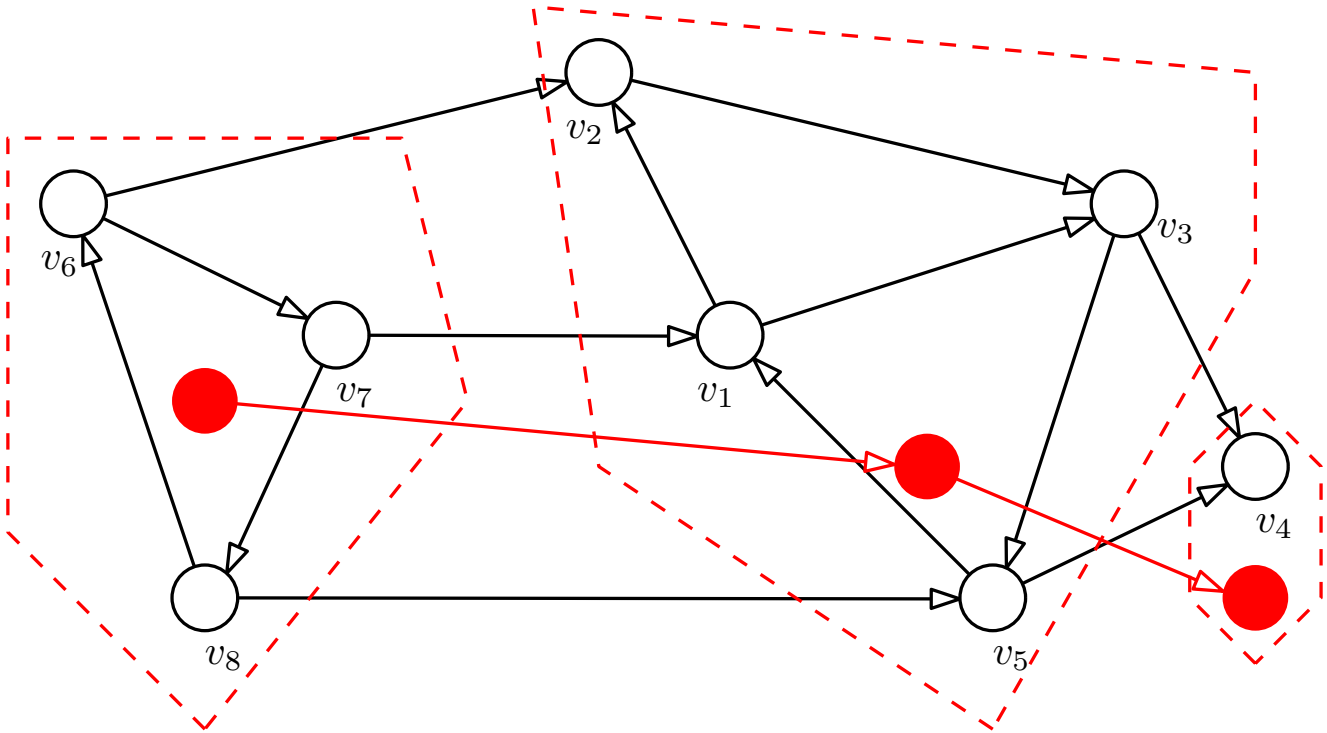
## Corollary

*If $u$ and $v$ belong to the same SCC in $G$, then the algorithm places $u$ and $v$ in the same cluster.*

- Remains to show: the other way round is also true

# The component digraph

- Define the component digraph $G^S = (V^S, E^S)$
  - $V^S = \{z(C) \mid C \text{ is a SCC of } G\}$
  - $E^S = \{(z(C), z(C')) \mid C \neq C' \wedge \exists v \in C, v' \in C' \text{ s.t. } (v, v') \in E\}$



# Properties of the component graph

## Lemma

$G^S$ *is a DAG.* directed acyclic graph

- Assume by contradiction that

$$\langle z(C_0), z(C_1), \ldots, z(C_{k-1}), z(C_k) = z(C_0) \rangle$$

  is a (directed) cycle in $G^S$
  - $k \geq 2$
- Then $v_0 \in C_0$ and $v_1 \in C_1$ are mutually reachable in $G$
- $\Longrightarrow$ $v_0$ and $v_1$ belong to the same SCC of $G$ $_{(\rightarrow\leftarrow)}$ ■

## Observation

$\overleftarrow{G}^S = \overleftarrow{G^S}$.

# The key lemma

- For $U \subseteq V$, define
  - $d(U) = \min\{v.d : v \in U\}$
  - $f(U) = \max\{v.f : v \in U\}$

**Lemma**

*Consider two distinct SCCs $C, C'$ of $G$. If $(z(C), z(C')) \in E^S$, then $f(C') < f(C)$.* הוכחת הלמה

- Case $d(C) < d(C')$:
- Let $x$ be the vertex that realizes $d(C)$
- At time $x.d$, there is a white path from $x$ to every vertex in $C$
- Since $(z(C), z(C')) \in E^S$, it follows that at time $x.d$, there is a white path from $x$ to every vertex in $C'$
- White path property: all vertices in $C'$ are descendants of $x$ in $G_\pi$
- Parenthesis property: $f(C') < x.f \leq f(C)$

# The proof continues

- Case $d(C') < d(C)$:
- Let $x'$ be the vertex that realizes $d(C')$
- At time $x'.d$, there is a white path from $x'$ to every vertex in $C'$
- White path property: all vertices in $C'$ are descendants of $x'$ in $G_\pi$
- Parenthesis property: $f(C') = x'.f$
- Since $(z(C), z(C')) \in E^S$ and since $G^S$ is acyclic, the vertices in $C$ are not reachable from $x'$
- $\Longrightarrow f(C') = x'.f < d(C)$
- The assertion follows as $d(C) < f(C)$ ■

# Wrapping up

**Corollary**

*If $C_{max}$ is the SCC of $G$ that maximizes $f(C_{max})$, then no edges enter $C_{max}$ in $G$*

- $\Longrightarrow$ No edges leave $C_{max}$ in $\overleftarrow{G}$
- Call to DFS($\overleftarrow{G}$) in line 4 starts with the vertex $u$ that maximizes $u.f$
  - $u.f = f(C_{max})$, thus $u \in C_{max}$
- When it retracts from $u$, all and only the vertices in $C_{max}$ have been discovered
  - The "first" rooted tree in $\overleftarrow{G}_\pi$ is indeed an SCC of $G$
- Remove $C_{max}$ from $G$ and repeat...
  - Proof by induction on #rooted trees constructed in line 4