DFS

Data Structures and Algorithms (094224)

Tutorial 5

Winter 2022/23

The depth first search algorithm

depth first search (DFS) ⟨סריקה לעומק⟩

- 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*

DFS – Pseudocode

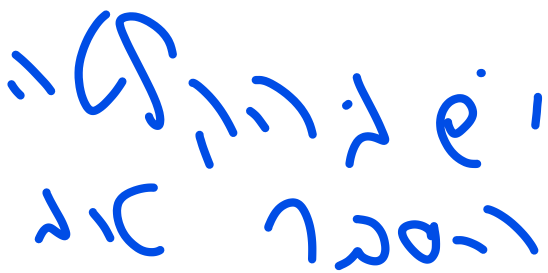
DFS(G)

1: **for all** $u \in G.V$ **do**
2: $u.color = white$
3: $u.\pi = NIL$
4: $time = 0$
5: **for all** $u \in G.V$ **do**
6: **if** $u.color == white$ **then**
7: DFS_Visit(G, u)

init

DFS_Visit(G, u)

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



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_π at the vertices v with $v.\pi = NIL$
  - Treat G_π as a collection of rooted trees
  - Parent-child relations reflect the "direction" of the recursive calls to `DFS_Visit`
- G_π is called the *DFS forest*

Edge Classification

- We use G_π to partition the edges of G into 4 types
  - Tree edge, back edge, forward edge, and crossing edge
- Resolving ambiguity for undirected graphs: classify $(u, v) = (v, u)$ according to the direction explored first by the algorithm
- 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
- Classify edge $(u, v) \in E$ as
  - Tree edge if it is included in G_π
    - real time: $v.color = white$
  - Back edge if v is an ancestor of u in G_π
    - may be a self-loop (if G is directed)
    - real time: $v.color = gray$
  - Forward edge if not a tree edge and v is a (proper) descendant of u
    - real time: $v.color = black$ and $u.d < v.d$
  - Crossing edge in all other cases
    - real time: $v.color = black$ and $u.d > v.d$
- Classification of edges depends on the graph representation

DFS — Theorems

Observation

v is a descendant of u in G_π iff v is discovered when u is gray.

Theorem (The white path property)

v is a descendant of u in G_π iff at time $u.d$, there is a (u, v)-path in G containing only white vertices.

Theorem (The parenthesis property)

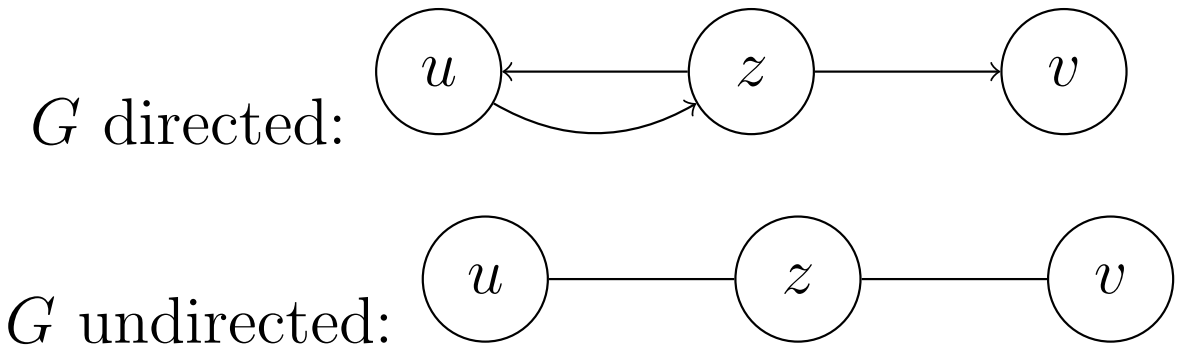
For any two vertices $u, v \in V$, $u \neq v$, exactly one of the following three conditions hold:

1. $[v.d, v.f] \subset [u.d, u.f]$ *and v is a descendant of u in G_π;*
2. $[u.d, u.f] \subset [v.d, v.f]$ *and u is a descendant of v in G_π; or*
3. $[u.d, u.f]$ *and* $[v.d, v.f]$ *are disjoint and u and v do not exhibit an ancestor-descendant relation in G_π.*

Question – 1

1. Give a counterexample to the claim that if a directed or undirected graph G contains a (u, v)-path, and if $u.d < v.d$ in $\texttt{DFS}(G)$, then v is a descendant of u in G_π
2. Give a counterexample to the claim that if a directed or undirected graph G contains a (u, v)-path, then any $\texttt{DFS}(G)$ must result in $v.d < u.f$

Solution



- We will show that there exists an execution of DFS such that $u.d < u.f < v.d$ and v is not a descendant of u in G_π
  - Solving both questions at once
- Assume `DFS`(G) first call `DFS_Visit`(G, z)
- Assume the vertex u is before vertex v on vertex z adjacency list
- `DFS_Visit`(G, z):
  - Will first call `DFS_Visit`(G, u) and then `DFS_Visit`(G, v)
    - $u.d < u.f < v.d$
    - $\Longrightarrow$ v is not a descendant of u in G_π by the parenthesis property

Question 2 — edge classification

Let $G = (V, E)$ be a directed or undirected graph. Show that an edge $(u, v) \in E$ after $\texttt{DFS}(G)$ is

1. A tree edge or a forward edge iff $u.d < v.d < v.f < u.f$
2. A back edge iff $v.d \leq u.d < u.f \leq v.f$
3. A crossing edge iff $v.d < v.f < u.d < u.f$

Recall: For an undirected graph we classify the edge $(u, v) \in E$ (and not (v, u)) since $\texttt{DFS}(G)$ explored the edge $(u, v) = (v, u)$ first from u to v

Question 2 — Solution

- Direction $\Longrightarrow$
- By definition, a tree edge or a forward edge is an edge from a vertex to a descendant in G_π, i.e., v is a descendant of u
- Edge (u, v) is not a back edge, hence $u \neq v$
- By the Parenthesis property (1) $u.d < v.d < v.f < u.f$
- Direction $\Longleftarrow$
- $u.d \neq v.d \Longrightarrow v \neq u$
- By the Parenthesis property (1) v is a descendant of u
- The edge (u, v) is from a vertex to a descendant
- (u, v) is either a tree edge or a forward edge

Solution — cont.

Solution 2.2:

- Direction $\Longrightarrow$
- By definition, a back edge is an edge from a vertex to an ancestor
- If $u = v$, then $v.d = u.d < u.f = v.f$
- Otherwise ($u \neq v$), By the Parenthesis property (2) $v.d < u.d < u.f < v.f$
- Direction $\Longleftarrow$
- If $u.d = v.d$, then $u = v$ and edge (u, v) is a back edge by definition
- Otherwise ($v.d < u.d$), $u \neq v$ thus $u.f < v.f$. By the Parenthesis property (2) u is a descendant of v
- (u, v) is a back edge

Solution — cont.

Solution 2.3:

- Direction $\Longrightarrow$
- By definition neither u nor v is an ancestor of the other
- By the Parenthesis property (3) either:
  - $v.d < v.f < u.d < u.f$; or
  - $u.d < u.f < v.d < v.f$
- The case $u.d < u.f < v.d < v.f$ is not possible since $u.d < v.d$ is not possible if edge (u, v) is a cross edge
  - If $u.d < v.d$, then at time $u.d$ vertex v has not been discovered yet ($v.d$ has not been set yet) and $v.color = white$
  - By the white path property, v is a descendant of u which contradicts edge (u, v) being a cross edge
- Direction $\Longleftarrow$
- By the parenthesis property (3) u and v do not exhibit an ancestor-descendant relation
- (u, v) is a cross edge

Question – 3

Prove that in a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.

Solution

- Let $e = (u, v) = (v, u) \in E$ be an edge and assume w.l.o.g. that $u.d < v.d$
- At time $u.d$ it holds that $v.color = white$, since $u.d < v.d$
- Since $(u, v) \in E$, at time $u.d$ there exists a (u, v)-path containing only white vertices
- By the white path property v is a descendant of u, hence by the parenthesis property (1) it holds that $u.d < v.d < v.f < u.f$, i.e., $\texttt{DFS}(G)$ explores and finish v before it finishes exploring u (while $u.color = gray$)
- If the first time $\texttt{DFS}(G)$ explores edge e it is in the direction from u to v, then $v.color = white$, thus (u, v) is a tree edge
  - If $v.color \neq white$, then $\texttt{DFS\_Visit}(G, v)$ was invoked before the edge e was explored
  - During $\texttt{DFS\_Visit}(G, v)$, all of v's incident edges are explored
  - Edge e would be explored in the direction from v to u
- Otherwise, (v, u) is a back edge since it was explored for the first time in the direction from v to u and $u.color = gray$

Question – 4

Let $G = (V, E)$ be a simple directed graph (i.e. G does not contain self-loops). Let $F \subseteq E$ be the set of back-edges after $\texttt{DFS}(G)$.
Prove/Disprove:

- $G' = (V, F)$ is a DAG
- $G' = (V, E - F)$ is a DAG

Solution

Solution 4.1 (the claim is true):

- Assume by contradiction that there exists a cycle $C = \langle v_0, \cdots, v_{k-1}, v_k = v_0 \rangle$ in G'
- Since all of the edges of C are back edges by Question 2 it holds that
  - $v_i.d > v_{i+1}.d$ for all $0 \leq i \leq k-1$
- Thus $v_0.d > \cdots > v_{k-1}.d > v_k.d = v_0.d$ $(\rightarrow\leftarrow)$
  - The d attribute is set only once

Solution 4.2 (the claim is true):

- Assume by contradiction there exists a cycle $C = \langle v_0, \cdots, v_{k-1}, v_k = v_0 \rangle$ in G'
- Since all of the edges of C are not back edges by Question 2 it holds that
  - $v_i.f > v_{i+1}.f$ for all $0 \leq i \leq k-1$
- Thus $v_0.f > \cdots > v_{k-1}.f > v_k.f = v_0.f$ $(\rightarrow\leftarrow)$
  - The f attribute is set only once

Question – 5

Prove that **every** depth-first search on $G = (V, E)$ (directed or undirected) produces a back edge iff G has a cycle.

Solution

Direction $\Longrightarrow$

- Let $e = (u, v) \in E$ be a back edge after $\texttt{DFS(G)}$
  - u is a descendant of v in G_π
- Case 1 – G is directed:
  - If $u = v$, then $(u, u) \in E$, hence $\langle u, u \rangle$ is a cycle
  - Otherwise $(u \neq v)$, let $P = \langle v, \cdots, u \rangle$ be the simple (v, u)-path in G_π
  - Path P does not traverse edge e otherwise it was a tree edge
  - Path $P' = \langle P, v \rangle$ is a cycle in G
- Case 2 – G is undirected:
  - Let $P = \langle v, w, \cdots, u \rangle$ be the simple (v, u)-path in G_π
  - $w \neq v \neq u$
    - There are no self-loops
    - Edge e is not a tree edge
  - Path $P' = \langle P, v \rangle$ is a cycle in G

Solution — cont.

Direction $\Longleftarrow$

- G contains a cycle $C = \langle v_0, v_1 \cdots, v_k = v_0 \rangle$
- Case 1 – G is directed:
  - Let v_i be the first vertex to be discovered in C
  - Cycle $C' = \langle v_i, v_{i+1}, \cdots, v_{i-1}, v_i \rangle$ is the same cycle as C
  - Path $P = \langle v_i, v_{i+1}, \cdots, v_{i-1} \rangle$ contains only white vertices at time $v_i.d$
  - By the white path property v_{i-1} is a descendant of v_i
  - Edge (v_{i-1}, v_i) is a back edge
- Case 2 – G is undirected:
  - According to question 3 every edge in C is either a tree or back edge
    - Graph G is undirected
  - Assume by contradiction that every edge in C is a tree edge
  - The DFS forest G_π contains all the edge in C
  - $\Longrightarrow$ G_π contains a cycle $_{(\rightarrow\leftarrow)}$
  - At least one edge in C will be classified as a back edge

Question – 6

Design an $O(V)$ time complexity algorithm for determine if an undirected graph $G = (V, E)$ contains a cycle.

Solution — Algorithm:

1. Run `DFS`(G) and classify the edges in real time
2. Upon encounter of a back edge return "G contains a cycle"
3. Otherwise, return "G is acyclic"

Correctness:

- Follows directly from question 5

Solution — cont.

Time Complexity:

Lemma

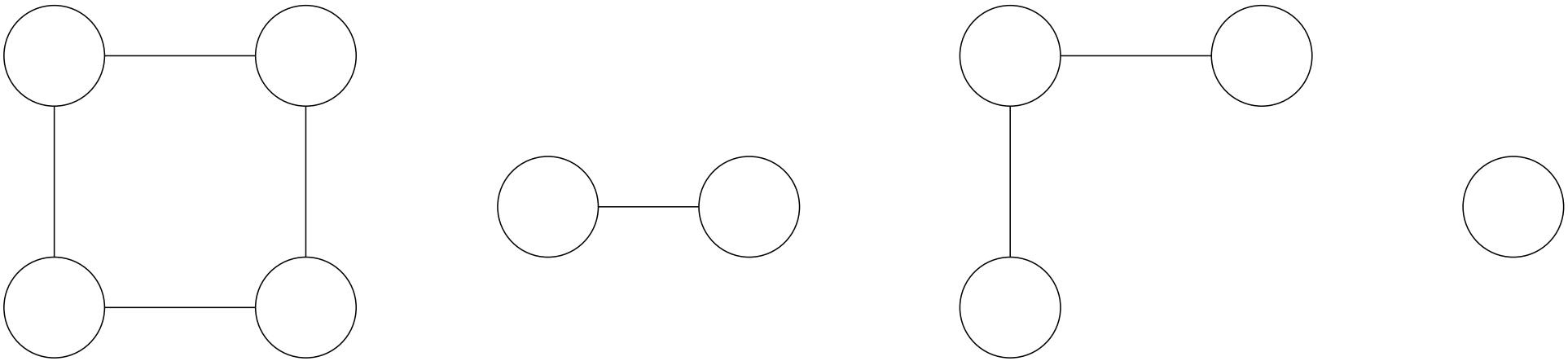
Let $G = (V, E)$ be an undirected graph. If G is acyclic then $|E| \leq |V| - 1$

Proof.

- Let $C_1 = (V_1, E_1), \cdots, C_k = (V_k, E_k)$ be the connected components of G
  - $V = V_1 \cup \cdots \cup V_k$, $E = E_1 \cup \cdots \cup E_k$
  - $V_1 \cap \cdots \cap V_k = \emptyset$
    - Otherwise, it is the same connected component
  - $E_1 \cap \cdots \cap E_k = \emptyset$
    - Since $V_1 \cap \cdots \cap V_k = \emptyset$
- G is acyclic $\Longrightarrow$ each connected component is acyclic
- $\Longrightarrow$ each connected component is a tree
- $\Longrightarrow$ for every connected component ,$1 \leq i \leq k$, $|E_i| = |V_i| - 1$
- $|E| = |E_1| + \cdots + |E_k| = |V_1| + \cdots + |V_k| - k = |V| - k$
- Since $k \geq 1$ it holds $|E| \leq |V| - 1$

Connected components — example

Graph G:



Solution — cont.

- According to question 3, every edge of G is either a tree edge or a back edge
  - G is undirected
- The run time of $\texttt{DFS}(G)$ is $O(V + E)$
- If $|E| \leq |V| - 1$ then the run time is $O(V)$
- Otherwise ($|E| \geq |V|$), let $E' \subseteq E$ be any subset of the edges of G such that $|E'| = |V|$
- According to the Lemma graph $G' = (V, E')$ contains a cycle
- According to question 5 every $\texttt{DFS}(G')$ will produce a back edge
- After traversing $O(V)$ edge $\texttt{DFS}(G)$ will produce a back edge
- $\Longrightarrow$ The run time is $O(V)$

Topological Sort – Definition

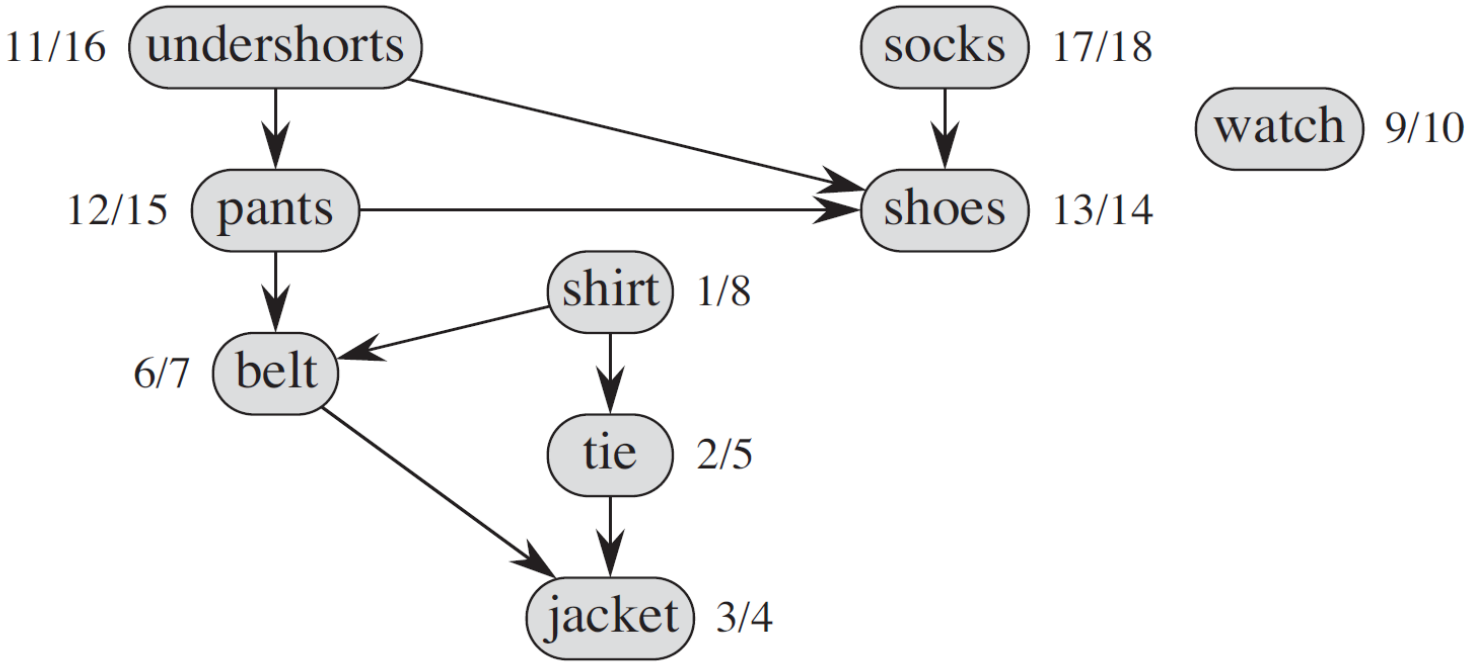
A *topological sort* of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.

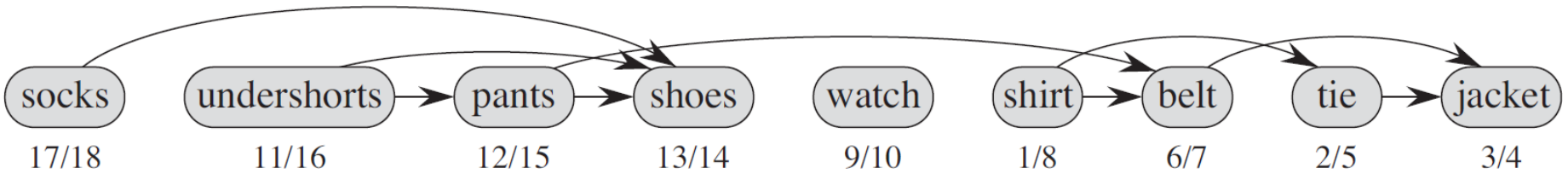
Comments:

- We can view a topological sort of a graph as an ordering of its vertices along a horizontal line so that all directed edges go from left to right
- Useful to indicate precedence among events
- Topological sort exists if and only if graph is a DAG
  - Topological sort is impossible if a cycle exists
  - We will show an algorithm that computes a topological sort of a DAG
- Topological sort is not necessarily unique

Topological Sort – Example

Professor Bumstead uses a DAG in order to get dressed in the morning. Nodes of the graph represent clothing items, e.g., socks, shoes, shirt. Edges represent precedence, e.g., edge from socks to shoes, wear socks before shoes.





Question – 7

Show how to use *DFS* to perform a topological sort of a DAG. Show complexity and prove correctness.

Solution

The main idea is to arrange all vertices of G from left to right in decreasing order of their retraction time for some *DFS* execution

Algorithm (Not a proper pseudocode):

`Topological_Sort(G)`

1. Create an empty linked list
2. Call `DFS(G)` to compute finishing times $v.f$ for each vertex v
   1. After $v.f$ is set, insert v to the front of the linked list
3. Return the linked list

Complexity Analysis:

- Create an empty list — $\Theta(1)$
- Insert a vertex to a linked list — $\Theta(1)$
- We add $\Theta(1)$-time operation to `DFS(G)`
- A total of $\Theta(V + E)$

Solution — cont.

Theorem

Topological_Sort(G) produces a topological sort of the DAG G

Proof.

- We will show that for every edge $(u, v) \in E$ it holds $v.f < u.f$
- Let $e = (u, v) \in E$ be an edge in G
- When DFS(G) explores edge e vertex v cannot be gray
  - If v is gray then v is an ancestor of u thus, e is a back edge
  - According to question 5 there are no back edges since G is acyclic
- At that time $v.color$ is either black or white
- If $v.color = white$ then v is a descendant of u. By the Parenthesis Property $v.f < u.f$
- Otherwise, $v.color = black$, i.e., v has already been finished, thus $v.f$ has already been set. Because u's edges are still being explored $u.f$ is not assigned. Once it does we get $v.f < u.f$