Decompositions of Graphs

— DFS/BFS, Cycle, DAG, Toposort, SCC, Bicomp

Hengfeng Wei

hfwei@nju.edu.cn

June 12, 2018

John Hopcroft

Robert Tarjan

"For fundamental achievements in the design and analysis of algorithms and data structures."

— *Turing Award*, 1986

SIAM J. Comput.
Vol. 1, No. 2, June 1972

DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by $k_1V + k_2E + k_3$ for some constants k_1, k_2, and k_3, where V is the number of vertices and E is the number of edges of the graph being examined.

Key words. Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

- "Depth-First Search And Linear Graph Algorithms" by Robert Tarjan.

SIAM J. COMPUT.
Vol. 1, No. 2, June 1972

DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an undirect graph are presented. The space and time requirements of both algorithms are bounded by $k_1V + k_2E + k_3$ for some constants k_1, k_2, and k_3, where V is the number of vertices and E is the number of edges of the graph being examined.

Key words. Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

"DFS is a powerful technique with many applications."

- "Depth-First Search And Linear Graph Algorithms" by Robert Tarjan.

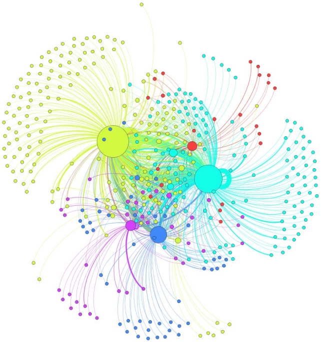
Power of DFS:

Graph Traversal $\Longrightarrow$ Graph Decomposition

Power of DFS:

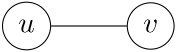
Graph Traversal $\Longrightarrow$ Graph Decomposition

Structure! Structure! Structure!



Graph *structure* induced by DFS:

states of 

types of 

Graph *structure* induced by DFS:

states of v

types of u —— v

life time of v:

v :d$[v]$, f$[v]$

d$[v]$: Bicomp

f$[v]$: Toposort, SCC

Definition (Classifying edges)

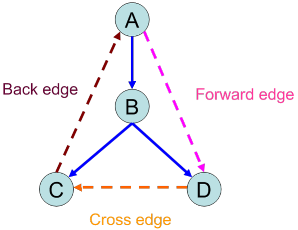
Given a DFS traversal $\implies$ DFS tree:

Tree edge: $\rightarrow$ child

Back edge: $\rightarrow$ ancestor

Forward edge: $\rightarrow$ *nonchild* descendant

Cross edge: $\rightarrow$ $(\neg\text{ancestor}) \wedge (\neg\text{descendant})$



Definition (Classifying edges)

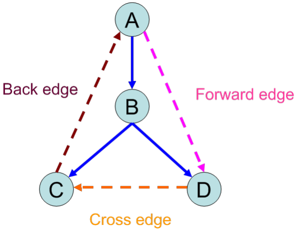
Given a DFS traversal $\implies$ DFS tree:

Tree edge: $\to$ child

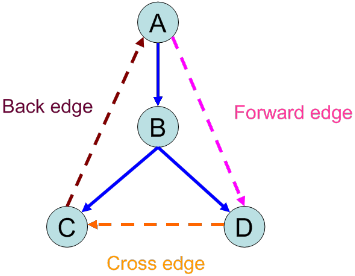
Back edge: $\to$ ancestor

Forward edge: $\to$ *nonchild* descendant

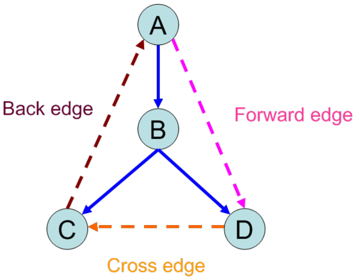
Cross edge: $\to$ $(\neg\text{ancestor}) \land (\neg\text{descendant})$



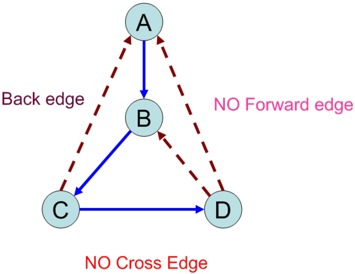
- Also applicable to BFS
- w.r.t. DFS/BFS trees



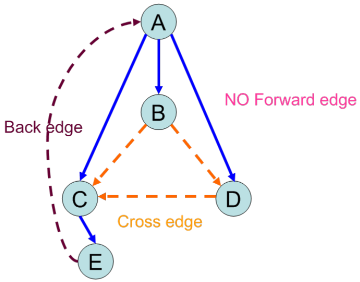
DFS on directed graph



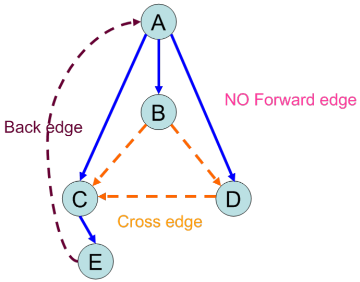
DFS on directed graph



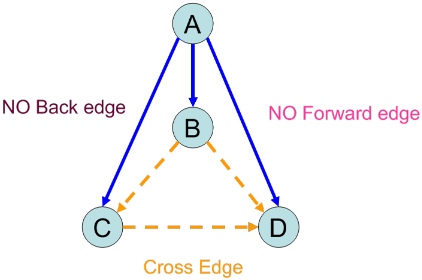
DFS on undirected graph



BFS on directed graph



BFS on directed graph



BFS on undirected graph (Problem 5.1)

DFS tree and BFS tree coincide (Problem 5.7)

Undirected connected graph $G = (V, E), v \in V$

DFS tree T from v $\equiv$ BFS tree T' from v

DFS tree and BFS tree coincide (Problem 5.7)

Undirected connected graph $G = (V, E), v \in V$

DFS tree T from $v \equiv$ BFS tree T' from v

$$G \equiv T$$

DFS tree and BFS tree coincide (Problem 5.7)

Undirected connected graph $G = (V, E), v \in V$

DFS tree T from $v \equiv$ BFS tree T' from v

$$G \equiv T$$

Proof.

G_{DFS}: tree + back *vs.* G_{BFS}: tree + cross

□

DFS tree and BFS tree coincide (Problem 5.7)

Undirected connected graph $G = (V, E), v \in V$

DFS tree T from $v \equiv$ BFS tree T' from v

$$G \equiv T$$

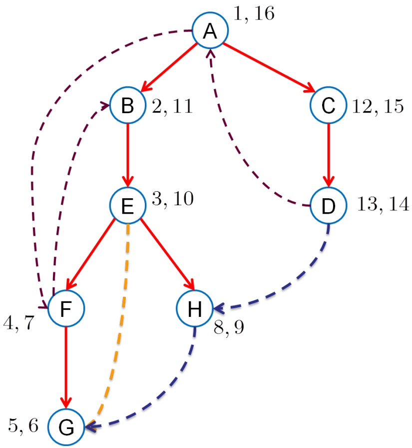
Proof.

G_{DFS}: tree + back *vs.* G_{BFS}: tree + cross

□

Q : What if G is a digraph?

Life time of vertices in DFS



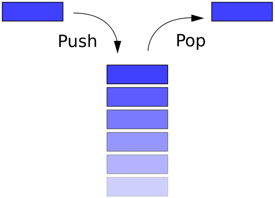
Theorem (Disjoint or Contained (Problem $4.2 : (1)\&(2)$))

$$\forall u, v : [_u\ ]_u \cap [_v\ ]_v = \emptyset \bigvee \Big([_u\ ]_u \subset [_v\ ]_v \vee [_v\ ]_v \subset [_u\ ]_u \Big)$$

Theorem (Disjoint or Contained (Problem $4.2 : (1)\&(2)$))

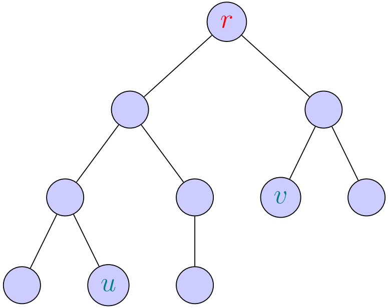
$$\forall u, v : [_u\ ]_u \cap [_v\ ]_v = \emptyset \bigvee \Big([_u\ ]_u \subset [_v\ ]_v \lor [_v\ ]_v \subset [_u\ ]_u\Big)$$

Proof.



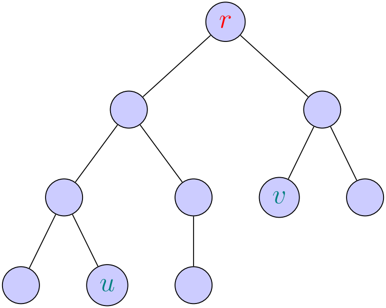
□

Preprocessing for ancestor/descendant relation (Problem 5.6)



Q : Is u an ancestor of v? $O(1)$

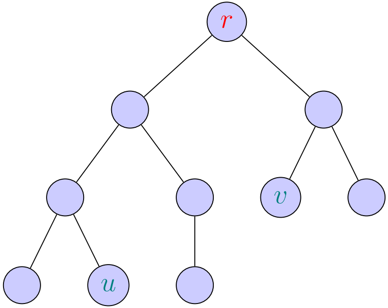
Preprocessing for ancestor/descendant relation (Problem 5.6)



Q : Is u an ancestor of v? $O(1)$

$$v : \mathsf{d}[v], \mathsf{f}[v]$$

Preprocessing for ancestor/descendant relation (Problem 5.6)



Q : Is u an ancestor of v? $O(1)$

$$v : \mathsf{d}[v], \mathsf{f}[v]$$

Q : # of descendants of any v?

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v :$$

- tree/forward edge: $[_u \ [_v \ ]_v \ ]_u$
- back edge: $[_v \ [_u \ ]_u \ ]_v$
- cross edge: $[_v \ ]_v \ [_u \ ]_u$

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v :$$

- tree/forward edge: $[_u \ [_v \ ]_v \ ]_u$
- back edge: $[_v \ [_u \ ]_u \ ]_v$
- cross edge: $[_v \ ]_v \ [_u \ ]_u$

$$\mathsf{f}[v] < \mathsf{d}[u] \iff \qquad \text{edge}$$

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v:$$

- tree/forward edge: $[_u\ [_v\ ]_v\ ]_u$
- back edge: $[_v\ [_u\ ]_u\ ]_v$
- cross edge: $[_v\ ]_v\ [_u\ ]_u$

$$\mathsf{f}[v] < \mathsf{d}[u] \iff \text{cross edge}$$

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v :$$

- tree/forward edge: $[_u \; [_v \;]_v \;]_u$
- back edge: $[_v \; [_u \;]_u \;]_v$
- cross edge: $[_v \;]_v \; [_u \;]_u$

$$\mathsf{f}[v] < \mathsf{d}[u] \iff \text{cross edge}$$

$$\mathsf{f}[u] < \mathsf{f}[v] \iff$$

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v:$$

- tree/forward edge: $[_u\ [_v\ ]_v\ ]_u$
- back edge: $[_v\ [_u\ ]_u\ ]_v$
- cross edge: $[_v\ ]_v\ [_u\ ]_u$

$$\mathsf{f}[v] < \mathsf{d}[u] \iff \text{cross edge}$$

$$\mathsf{f}[u] < \mathsf{f}[v] \iff \text{back edge}$$

Edge types and life time of vertices in DFS (Problem 4.5)

$$\forall u \to v :$$

- tree/forward edge: $[_u \; [_v \;]_v \;]_u$
- back edge: $[_v \; [_u \;]_u \;]_v$
- cross edge: $[_v \;]_v \; [_u \;]_u$

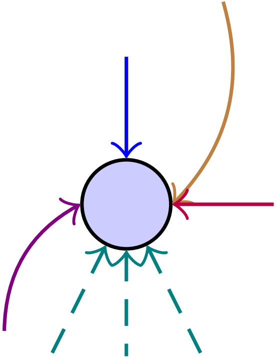
$$\text{f}[v] < \text{d}[u] \iff \text{cross edge}$$

$$\text{f}[u] < \text{f}[v] \iff \text{back edge}$$

$$\nexists \text{ cycle} \implies \boxed{u \to v \iff \text{f}[v] < \text{f}[u]}$$

DFS from the perspective of a single node:

DFS from the perspective of a single node:



Height and diameter of tree (Problem 5.4)

Binary tree $T = (V, E)$ with $|V| = n$ and the root r:

(I) Height $H(T)$ in $O(n)$

(II) Diameter $D(T)$ in $O(n)$

Height and diameter of tree (Problem 5.4)

Binary tree $T = (V, E)$ with $|V| = n$ and the root r:

(I) Height $H(T)$ in $O(n)$

(II) Diameter $D(T)$ in $O(n)$

$$\begin{cases} \\ H(T) = \max\Big(H(L_T), H(R_T)\Big) + 1, \end{cases}$$

Height and diameter of tree (Problem 5.4)

Binary tree $T = (V, E)$ with $|V| = n$ and the root r:

(I) Height $H(T)$ in $O(n)$

(II) Diameter $D(T)$ in $O(n)$

$$\begin{cases} H(T) = 0, & T \text{ is a leave} \\ H(T) = \max\Big(H(L_T), H(R_T)\Big) + 1, & \text{o.w.} \end{cases}$$

Height and diameter of tree (Problem 5.4)

Binary tree $T = (V, E)$ with $|V| = n$ and the root r:

(I) Height $H(T)$ in $O(n)$

(II) Diameter $D(T)$ in $O(n)$

$$\begin{cases} H(T) = 0, & T \text{ is a leave} \\ H(T) = \max\Big(H(L_T), H(R_T)\Big) + 1, & \text{o.w.} \end{cases}$$

$$\begin{cases} D(T) = 0, & T \text{ is a leave} \\ D(T) = \max\Big(D(L_T), D(R_T), \qquad\qquad\qquad\qquad \Big), & \text{o.w.} \\ \end{cases}$$

Height and diameter of tree (Problem 5.4)

Binary tree $T = (V, E)$ with $|V| = n$ and the root r:

(I) Height $H(T)$ in $O(n)$

(II) Diameter $D(T)$ in $O(n)$

$$\begin{cases} H(T) = 0, & T \text{ is a leave} \\ H(T) = \max\Big(H(L_T), H(R_T)\Big) + 1, & \text{o.w.} \end{cases}$$

$$\begin{cases} D(T) = 0, & T \text{ is a leave} \\ D(T) = \max\Big(D(L_T), D(R_T), \underbrace{H(L_T) + H(R_T) + 2}_{\text{through the root}}\Big), & \text{o.w.} \end{cases}$$

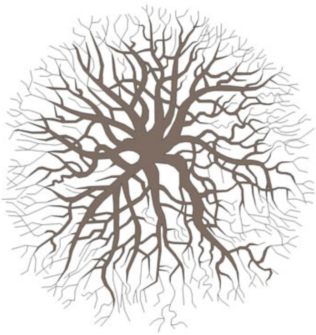
Binary tree $T = (V, E)$ with $|V| = n$ and the root r

Binary tree $T = (V, E)$ with $|V| = n$ and the root r

Q : Diameter of a *tree* *without* a designated root

Binary tree $T = (V, E)$ with $|V| = n$ and the root r

Q : Diameter of a *tree* *without* a designated root



Q : Diameter of a tree *without* a designated root

Q : Diameter of a tree *without* a designated root

A beautiful algorithm:

- Pick any u
- Run BFS from u, obtain the farthest v
- Run BFS from v, obtain the farthest w

$$(v, w)$$

Q : Diameter of a tree *without* a designated root

A beautiful algorithm:

- Pick any u
- Run BFS from u, obtain the farthest v
- Run BFS from v, obtain the farthest w

$$(v, w)$$

Your Job: Prove it!

Q : Diameter of a tree *without* a designated root

A beautiful algorithm:

- Pick any u
- Run BFS from u, obtain the farthest v
- Run BFS from v, obtain the farthest w

$$(v, w)$$

Your Job: Prove it!

Back to our original problem with $u \leftarrow r$.

Cycle detection (Problem $5.8 - 1$)

	Digraph	Undirected graph
DFS		
BFS		

Cycle detection (Problem $5.8 - 1$)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	
BFS		

Cycle detection (Problem $5.8 - 1$)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		

Cycle detection (Problem $5.8 - 1$)

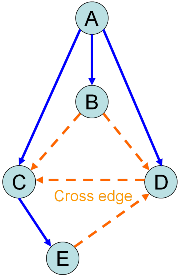
	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		cross edge $\iff$ cycle

Cycle detection (Problem $5.8-1$)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS	back edge $\implies$ cycle cycle $\not\Longrightarrow$ back edge	cross edge $\iff$ cycle

Cycle detection (Problem $5.8-1$)

	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS	back edge $\implies$ cycle cycle $\not\Longrightarrow$ back edge	cross edge $\iff$ cycle



Evasiveness of acyclicity of undirected graphs (Problem $5.8 - 2$)

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Evasiveness of acyclicity of undirected graphs (Problem $5.8-2$)

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is acyclicity evasive?

Evasiveness of acyclicity of undirected graphs (Problem $5.8-2$)

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is acyclicity evasive?

By Adversary Argument.

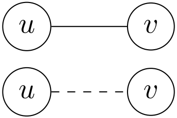
Evasiveness of acyclicity of undirected graphs (Problem $5.8-2$)

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Algorithm $\mathbb{A}$:

$\textsc{CheckEdge}(u, v)$

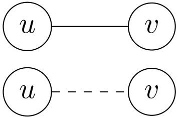
Evasiveness of acyclicity of undirected graphs (Problem $5.8 - 2$)

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Algorithm $\mathbb{A}$:

$\textsc{CheckEdge}(u, v)$

Hint: Kruskal

$$\mathbb{A} : \textsc{CheckEdge}(u, v) \leftarrow \mathcal{A} : \; (u) \text{———} (v)$$

$$\Longleftrightarrow$$

$$\mathcal{A} : \nexists \; \text{cycle} \; \in G + (u) \text{———} (v)$$

$$\mathbb{A} : \textsc{CheckEdge}(u, v) \leftarrow \mathcal{A} : \; (u) \text{———} (v)$$

$$\iff$$

$$\mathcal{A} : \nexists \text{ cycle } \in G + (u) \text{———} (v)$$

Q : Why adjacency matrix?

After-class Exercise: Evasiveness of connectivity of undirected graphs

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is connectivity evasive?

After-class Exercise: Evasiveness of connectivity of undirected graphs

$$\text{Evasiveness} \triangleq \text{check} \binom{n}{2} \text{ edges (adjacency matrix)}$$

Q : Is connectivity evasive?

Hint: Anti-Kruskal

Orientation of undirected graph (Problem 4.13)

- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \mathsf{in}[v] \geq 1$$

Orientation of undirected graph (Problem 4.13)

- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation $\iff \exists$ cycle C

Orientation of undirected graph (Problem 4.13)

- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation $\iff \exists$ cycle C

DFS from $v \in C$

Orientation of undirected graph (Problem 4.13)

- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation $\iff \exists$ cycle C

DFS from $v \in C$

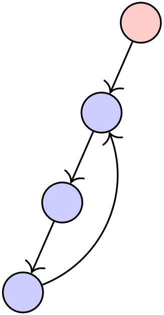
Orientation of undirected graph (Problem 4.13)

- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation $\Longleftrightarrow$ $\exists$ cycle C

DFS from $v \in C$



Orientation of undirected graph (Problem 4.13)

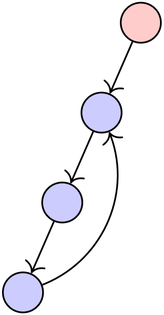
- undirected (connected) graph G
- edges oriented *s.t.*

$$\forall v, \text{in}[v] \geq 1$$

orientation $\Longleftrightarrow \exists$ cycle C

DFS from $v \in C$

Q : BFS?



Shortest cycle of undirected graph (Problem 4.12)

A WRONG DFS-based algorithm:

$$\forall v : \text{level}[v]$$

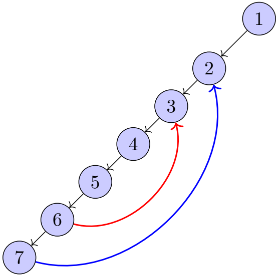
$$\text{Back edge } u \to v : \text{level}[u] - \text{level}[v] + 1$$

Shortest cycle of undirected graph (Problem 4.12)

A WRONG DFS-based algorithm:

$$\forall v : \mathsf{level}[v]$$

$$\text{Back edge } u \to v : \mathsf{level}[u] - \mathsf{level}[v] + 1$$



Shortest cycle of digraph (Problem 4.12)

A DFS-based algorithm:

$$\forall v : \text{level}[v]$$

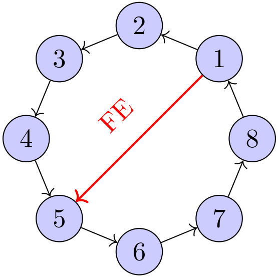
$$\text{Back edge } u \to v : \text{level}[u] - \text{level}[v] + 1$$

Shortest cycle of digraph (Problem 4.12)

A WRONG DFS-based algorithm:

$$\forall v : \textsf{level}[v]$$

$$\text{Back edge } u \to v : \textsf{level}[u] - \textsf{level}[v] + 1$$



On digraphs:

$\nexists$ back edge $\iff$ DAG

On digraphs:

$\nexists$ back edge $\iff$ DAG $\iff$ $\exists$ topo. ordering

On digraphs:

$$\nexists \text{ back edge} \iff \text{DAG} \iff \exists \text{ topo. ordering}$$

TOPOSORT by Tarjan (probably), 1976

$$\nexists \text{ cycle} \implies \boxed{u \to v \iff \mathsf{f}[v] < \mathsf{f}[u]}$$

On digraphs:

$$\nexists \text{ back edge} \iff \text{DAG} \iff \exists \text{ topo. ordering}$$

Toposort by Tarjan (probably), 1976

$$\nexists \text{ cycle} \implies \boxed{u \to v \iff \mathsf{f}[v] < \mathsf{f}[u]}$$

Sort vertices in *decreasing* order of their *finish* times.

Kahn's $\textsc{Toposort}$ algorithm (1962; Problem 4.16)

- Queue Q for source vertices (in$[v] = 0$)
- Repeat: $\textsc{Dequeue}(\exists u \in Q)$, output u

  delete u and $u \to v$ from Q,

  $\textsc{Enqueue}(v)$ if in$[v] = 0$

Kahn's TOPOSORT algorithm (1962; Problem 4.16)

- Queue Q for source vertices ($\text{in}[v] = 0$)
- Repeat: $\textsc{Dequeue}(\exists u \in Q)$, output u

  delete u and $u \to v$ from Q,

  $\textsc{Enqueue}(v)$ if $\text{in}[v] = 0$

$$O(m + n)$$

Kahn's TOPOSORT algorithm (1962; Problem 4.16)

- Queue Q for source vertices (in$[v] = 0$)
- Repeat: $\text{DEQUEUE}(\exists u \in Q)$, output u

  delete u and $u \to v$ from Q,

  $\text{ENQUEUE}(v)$ if in$[v] = 0$

$$O(m+n)$$

Lemma (Correctness of Kahn's TOPOSORT)

Every DAG has at least one source (and at least one sink vertex).

Kahn's TOPOSORT algorithm (1962; Problem 4.16)

- Queue Q for source vertices ($\text{in}[v] = 0$)
- Repeat: $\textsc{Dequeue}(\exists u \in Q)$, output u

  delete u and $u \to v$ from Q,

  $\textsc{Enqueue}(v)$ if $\text{in}[v] = 0$

$$O(m+n)$$

Lemma (Correctness of Kahn's TOPOSORT)

Every DAG has at least one source (and at least one sink vertex).

Q : What if G is *not* a DAG?

Taking courses in few semesters (Problem 4.20)

- n courses
- m of $c_1 \to c_2$: prerequisite
- Goal: taking courses in few semesters

Taking courses in few semesters (Problem 4.20)

- n courses
- m of $c_1 \to c_2$: prerequisite
- Goal: taking courses in few semesters

Critical path *OR* Longest path using DFS in $O(n+m)$

Taking courses in few semesters (Problem 4.20)

- n courses
- m of $c_1 \to c_2$: prerequisite
- Goal: taking courses in few semesters

Critical path *OR* Longest path using DFS in $O(n+m)$

For general digraph, LONGEST-PATH is NP-hard.

Line up (Problem 4.22)

1. i hates j: $i \succ j$
2. i hates j: $\#i < \#j$

TOPOSORT

Critical path *OR* Longest path

Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

Q : $\exists$ HP in a DAG in $O(n+m)$

Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

Q : $\exists$ HP in a DAG in $O(n+m)$

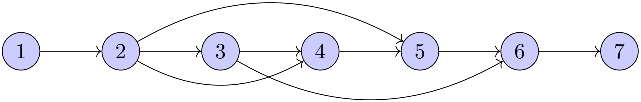
For general (di)graph, HP is NP-hard.

Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

Q : $\exists$ HP in a DAG in $O(n+m)$

For general (di)graph, HP is NP-hard.

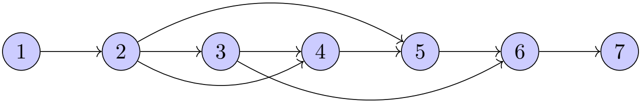


Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

Q : $\exists$ HP in a DAG in $O(n+m)$

For general (di)graph, HP is NP-hard.



$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$

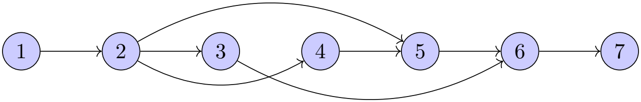
$$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$$

$$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$$

Tarjan's TOPOSORT + Check edges (v_i, v_{i+1})

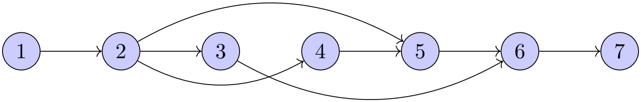
$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$

Tarjan's $\textsc{Toposort}$ + Check edges (v_i, v_{i+1})



$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$

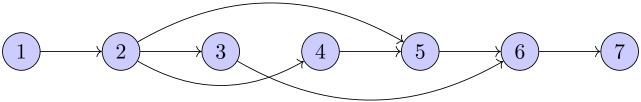
Tarjan's $\textsc{Toposort}$ + Check edges (v_i, v_{i+1})



Kahn's $\textsc{Toposort}$ (Problem 4.16)

$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$

Tarjan's $\textsc{Toposort}$ + Check edges (v_i, v_{i+1})



Kahn's $\textsc{Toposort}$ (Problem 4.16)

$$|Q| \leq 1$$

Theorem (Digraph as DAG (Problem 4.6))

Every digraph is a dag of its SCCs.

Theorem (Digraph as DAG (Problem 4.6))

Every digraph is a dag of its SCCs.

Two tiered structure of digraphs:

digraph $\equiv$ a dag of SCCs

SCC: equivalence class over reachability

$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$

Kosaraju's SCC algorithm, 1978

"SCCs can be topo-sorted

in decreasing order of their highest finish time."

$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$

Kosaraju's SCC algorithm, 1978

"SCCs can be topo-sorted

in decreasing order of their highest finish time."

The vertice with the highest finish time is in a source SCC.

$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$

Kosaraju's SCC algorithm, 1978

"SCCs can be topo-sorted

in decreasing order of their highest finish time."

The vertice with the highest finish time is in a source SCC.

(I) DFS on G; DFS/BFS on G^T

$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$

Kosaraju's SCC algorithm, 1978

"SCCs can be topo-sorted

in decreasing order of their highest finish time."

The vertice with the highest finish time is in a source SCC.

(I) DFS on G; DFS/BFS on G^T

(II) DFS on G^T; DFS/BFS on G

Kosaraju's SCC algorithm, 1978 (Problem 4.7)

$$\text{1st DFS} \stackrel{?}{\Longrightarrow} \text{BFS}$$

$$\text{2nd DFS} \stackrel{?}{\Longrightarrow} \text{BFS}$$

Kosaraju's SCC algorithm, 1978 (Problem 4.7)

$$1\text{st DFS} \overset{?}{\Longrightarrow} \text{BFS}$$

$$2\text{nd DFS} \overset{?}{\Longrightarrow} \text{BFS}$$

1st DFS: toposort between SCCs

2nd DFS: reachability within an SCC

Kosaraju's SCC algorithm, 1978 (Problem 4.7)

$$\text{1st DFS} \overset{?}{\Longrightarrow} \text{BFS}$$

$$\text{2nd DFS} \overset{?}{\Longrightarrow} \text{BFS}$$

1st DFS: toposort between SCCs

2nd DFS: reachability within an SCC

$$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$$

One-to-all reachability in a digraph (Problem 5.12)

$$v : v \rightsquigarrow^{?} \forall u$$

$$\exists ?\ v : v \rightsquigarrow \forall u$$

One-to-all reachability in a digraph (Problem 5.12)

$$v : v \leadsto^{?} \forall u$$

$$\exists ?\ v : v \leadsto \forall u$$

SCC

$$\exists!\ \text{source vertex } v \iff v \leadsto \forall u$$

One-to-all reachability in a digraph (Problem 5.12)

$$v : v \leadsto^{?} \forall u$$

$$\exists ?\ v : v \leadsto \forall u$$

SCC

$$\exists!\ \text{source vertex}\ v \iff v \leadsto \forall u$$

$$\Longleftarrow : \exists!\ \text{source}$$

One-to-all reachability in a digraph (Problem 5.12)

$$v : v \rightsquigarrow^{?} \forall u$$

$$\exists ?\ v : v \rightsquigarrow \forall u$$

SCC

$$\exists !\ \text{source vertex } v \iff v \rightsquigarrow \forall u$$

$$\Longleftarrow : \exists !\ \text{source}$$

$$\Longrightarrow : \text{By contradiction.}$$

$$\exists u : v \not\rightsquigarrow u \land \text{in}[u] > 0 \implies \exists \text{ cycle}$$

Impacts of vertices in a digraph (Problem 4.18)

$$\mathsf{impact}(v) = |\{w \neq v : v \rightsquigarrow w\}|$$

- $\arg\min_v \mathsf{impact}(v)$
- $\arg\max_v \mathsf{impact}(v)$

Impacts of vertices in a digraph (Problem 4.18)

$$\text{impact}(v) = |\{w \neq v : v \rightsquigarrow w\}|$$

- $\arg\min_v \text{impact}(v)$
- $\arg\max_v \text{impact}(v)$

$$\arg\min_v \text{impact}(v) \in \text{sink SCC of smallest cardinality}$$

Impacts of vertices in a digraph (Problem 4.18)

$$\text{impact}(v) = |\{w \neq v : v \leadsto w\}|$$

- $\arg\min_v \text{impact}(v)$
- $\arg\max_v \text{impact}(v)$

$$\arg\min_v \text{impact}(v) \in \text{sink SCC of smallest cardinality}$$

$$\arg\max_v \text{impact}(v) \in \text{source SCC}$$

Impacts of vertices in a digraph (Problem 4.18)

$$\text{impact}(v) = |\{w \neq v : v \leadsto w\}|$$

- $\arg\min_v \text{impact}(v)$
- $\arg\max_v \text{impact}(v)$

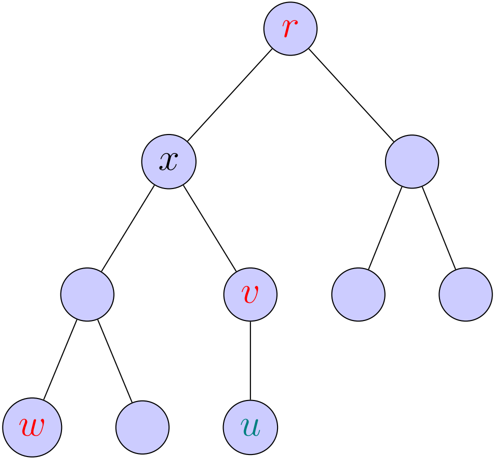
$$\arg\min_v \text{impact}(v) \in \text{sink SCC of smallest cardinality}$$

$$\arg\max_v \text{impact}(v) \in \text{source SCC}$$

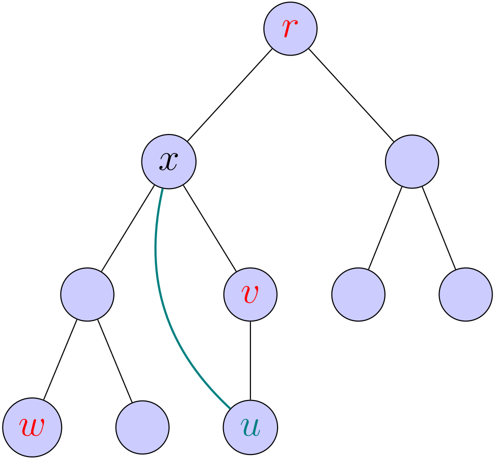
Q : $\forall v$, computing impact(v)

BICOMP: Back!

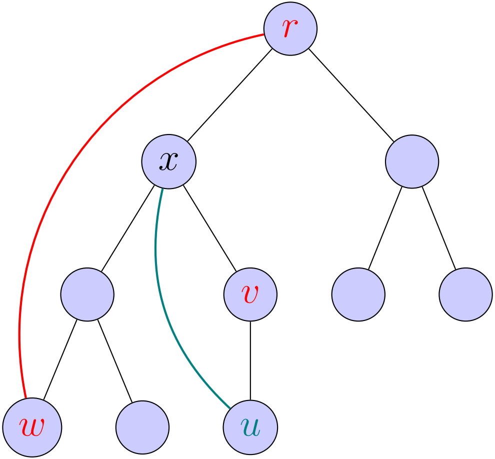
Bicomp: Back!



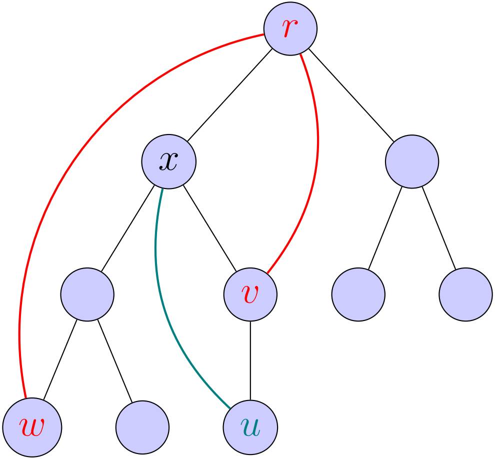
BICOMP: Back!



BICOMP: Back!



BICOMP: Back!



back[v]: the earliest reachable ancestor of v

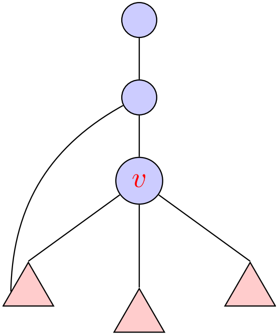
(I) When and how to update back[v]?

(II) When and how to identify a bicomponent?

back$[v]$: the earliest reachable ancestor of v

(I) When and how to update back$[v]$?

(II) When and how to identify a bicomponent?



Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \lor 2(n+1)$$

Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \vee 2(n+1)$$

tree edge $(\to v)$: $\text{back}[v] = d[v]$

back edge $(v \to w)$: $\text{back}[v] = \min\{\text{back}[v], d[w]\}$

backtracking from w: $\text{back}[v] = \min\{\text{back}[v], \text{back}[w] = \text{wBack}\}$

Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \vee 2(n+1)$$

tree edge $(\to v)$: $\text{back}[v] = d[v]$

back edge $(v \to w)$: $\text{back}[v] = \min\{\text{back}[v], d[w]\}$

backtracking from w: $\text{back}[v] = \min\{\text{back}[v], \text{back}[w] = \text{wBack}\}$

Proof.

if ever updated

Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \lor 2(n+1)$$

$$\begin{aligned} \text{tree edge } (\to v)\text{:} \quad & \text{back}[v] = d[v] \\ \text{back edge } (v \to w)\text{:} \quad & \text{back}[v] = \min\{\text{back}[v], d[w]\} \\ \text{backtracking from } w\text{:} \quad & \text{back}[v] = \min\{\text{back}[v], \text{back}[w] = \text{wBack}\} \end{aligned}$$

Proof.

if never updated:

if ever updated

Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \lor 2(n+1)$$

tree edge ($\to v$): back$[v] = d[v]$

back edge ($v \to w$): back$[v] = \min\{\text{back}[v], d[w]\}$

backtracking from w: back$[v] = \min\{\text{back}[v], \text{back}[w] = \text{wBack}\}$

Proof.

if never updated:

if ever updated

wBack $= \infty > d[v]$ *vs.*

Initialization of back$[v]$ (Problem 4.9)

$$\text{back}[v] = d[v] \textit{ vs. } \text{back}[v] = \infty \vee 2(n+1)$$

tree edge $(\to v)$: $\text{back}[v] = d[v]$

back edge $(v \to w)$: $\text{back}[v] = \min\{\text{back}[v], d[w]\}$

backtracking from w: $\text{back}[v] = \min\{\text{back}[v], \text{back}[w] = \text{wBack}\}$

Proof.

if never updated:

if ever updated $\quad \text{wBack} = \infty > d[v] \textit{ vs. } \text{wBack} = d[w] > d[v]$

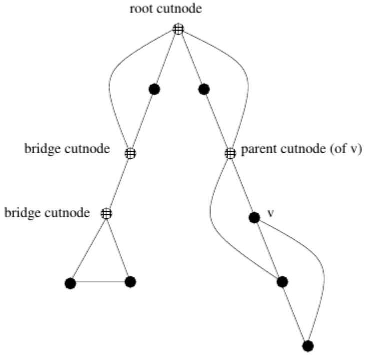
$\square$

Root cutnode v (Problem 4.8)

$$v \text{ is a cutnode } \iff \text{out}[v] \geq 2$$

Root cutnode v (Problem 4.8)

$$v \text{ is a cutnode } \iff \text{out}[v] \geq 2$$




THANK YOU