Decompositions of Graphs

Hengfeng Wei

hfwei@nju.edu.cn

June 12, 2018

John Hopcroft

Robert Tarjan

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 un-direct 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.

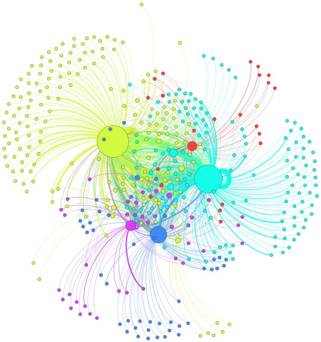
Power of DFS:

Graph Traversal $\Longrightarrow$ Graph Decomposition

Power of DFS:

Graph Traversal $\Longrightarrow$ Graph Decomposition

Structure! Structure! Structure!



Graph *structure* induced by DFS:

states of v

types of $u - v$

Graph *structure* induced by DFS:

states of v

types of u — v

life time of v:

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

$\mathsf{f}[v]$: DAG, SCC

$\mathsf{d}[v]$: biconnectivity

Definition (Classifying edges)

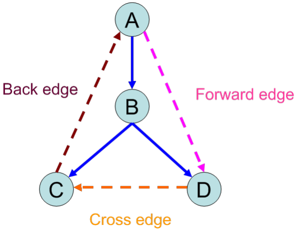
Given a DFS traversal $\Longrightarrow$ DFS tree:

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)

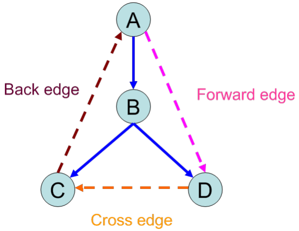
Given a DFS traversal $\implies$ DFS tree:

Tree edge: $\to$ child

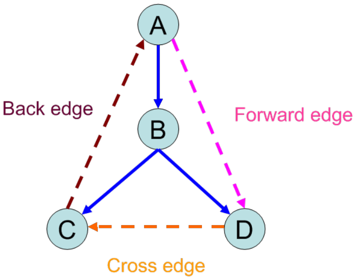
Back edge: $\to$ ancestor

Forward edge: $\to$ *nonchild* descendant

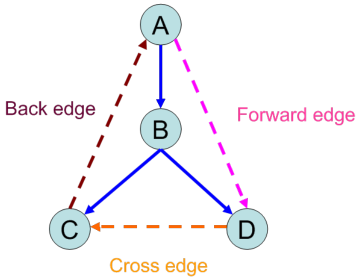
Cross edge: $\to$ $(\neg\text{ancestor}) \wedge (\neg\text{descendant})$



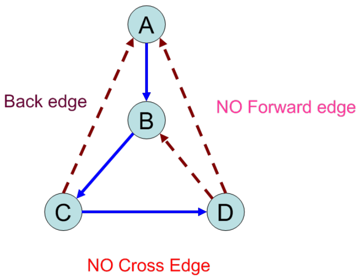
- also applicable to BFS
- w.r.t. DFS/BFS trees



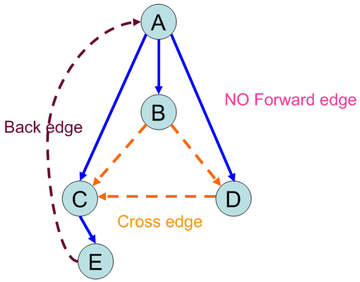
DFS on directed graph



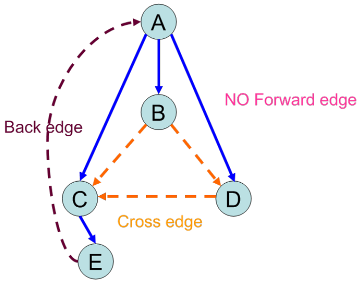
DFS on directed graph



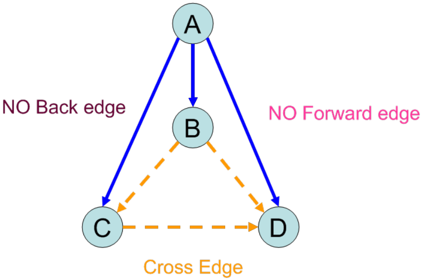
DFS on undirected graph



BFS on directed graph



BFS on directed graph



BFS on undirected graph

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

$\square$

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

$$G \equiv T$$

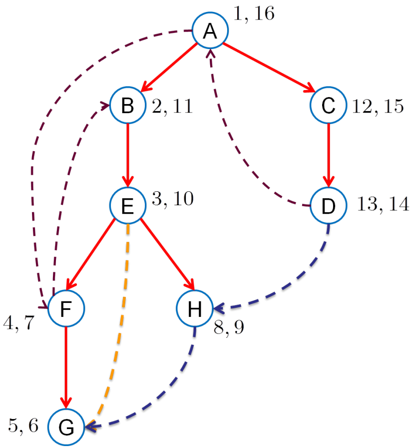
Proof.

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

□

Q : What if G is a digraph?

Lift time of vertices in DFS



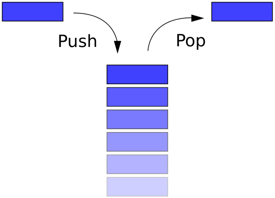
Theorem (Disjoint or Contained (Problem 4.2: (1) & (2)))

$$\forall u, v : [_u\ ]_u \cap [_v\ ]_v = \emptyset \bigvee \left([_u\ ]_u \subsetneq [_v\ ]_v \vee [_v\ ]_v \subsetneq [_u\ ]_u \right)$$

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

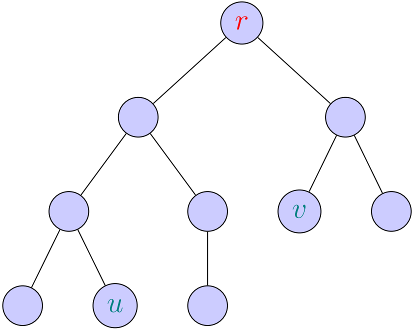
$$\forall u, v : [_u\ ]_u \cap [_v\ ]_v = \emptyset \bigvee \left([_u\ ]_u \subsetneqq [_v\ ]_v \vee [_v\ ]_v \subsetneqq [_u\ ]_u \right)$$

Proof.



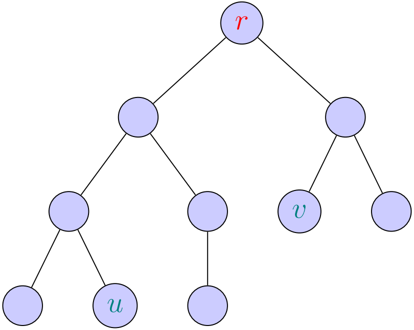
□

Preprocessing for ancestor/descendant relation (Problem 5.23)



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

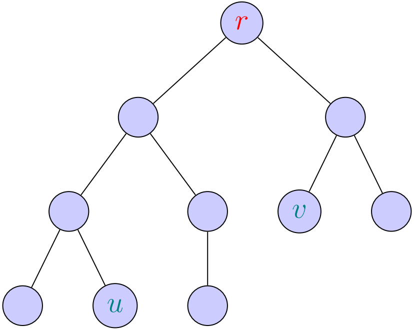
Preprocessing for ancestor/descendant relation (Problem 5.23)



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

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

Preprocessing for ancestor/descendant relation (Problem 5.23)



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: $[_v\ ]_v\ [_u\ ]_u$

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

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

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: $[_v \ ]_v \ [_u \ ]_u$

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

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

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

Height and diameter of tree (Problem 5.4)

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

- height $H(T)$ in $O(n)$
- 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:

- height $H(T)$ in $O(n)$
- 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:

- height $H(T)$ in $O(n)$
- 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:

- height $H(T)$ in $O(n)$
- diameter $D(T)$ in $O(n)$

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

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

Height and diameter of tree (Problem 5.4)

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

- height $H(T)$ in $O(n)$
- diameter $D(T)$ in $O(n)$

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

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

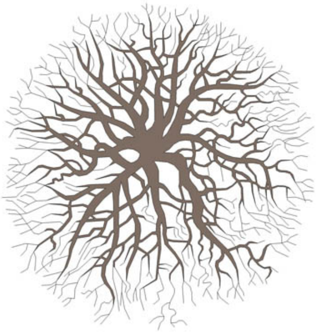
Binary tree $T = (V, E)$ with $|V| = n$ and the root r

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

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

Your Job: Prove it!

Counting shortest paths (Problem 5.26)

Counting # of shortest paths in (un)directed graphs using BFS.

Counting shortest paths (Problem 5.26)

Counting # of shortest paths in (un)directed graphs using BFS.

Maybe in the next class...

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 $5.8 - 1$)

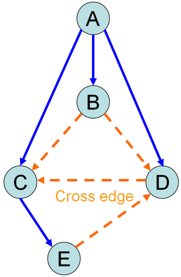
	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		cross edge $\iff$ cycle

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.

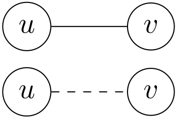
Evasiveness of acyclicity of undirected graphs (Problem $5.8 - 2$)

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

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Algorithm $\mathbb{A}$:

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

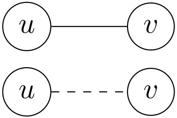
Evasiveness of acyclicity of undirected graphs (Problem $5.8 - 2$)

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

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Algorithm $\mathbb{A}$:

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

Hint: Kruskal

$$\textsc{CheckEdge}(u, v) \leftarrow \textcircled{u} \text{———} \textcircled{v}$$

$$\Longleftrightarrow$$

$$\nexists \ \text{cycle} \ \in G + \textcircled{u} \text{———} \textcircled{v}$$

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

$$\iff$$

$\nexists$ cycle $\in G + (u) - (v)$

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 $\Longleftrightarrow$ $\exists$ 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 $v \in C$

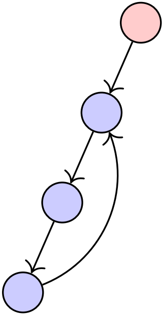
Orientation of undirected graph (Problem 4.13)

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

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

orientation $\iff \exists$ cycle C

DFS from $v \in C$



Shortest cycle of undirected graph (Problem 4.12)

A WRONG DFS-based algorithm:

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

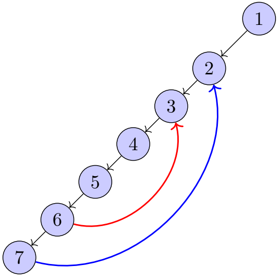
$$\text{Back edge } u \to v : \mathsf{level}[u] - \mathsf{level}[v] + 1$$

Shortest cycle of 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:

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

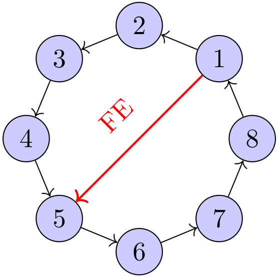
$$\text{Back edge } u \to v : \mathsf{level}[u] - \mathsf{level}[v] + 1$$

Shortest cycle of digraph (Problem 4.12)

A WRONG DFS-based algorithm:

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

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



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:

$\nexists$ back edge $\iff$ DAG $\iff$ $\exists$ 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 TOPOSORT algorithm (1962; Problem 4.16)

- Queue Q for source vertices (in$[v] = 0$)
- Repeat: $\textsc{Dequeue}(u \in Q)$, delete u and $u \to v$ from Q,
  output u, $\textsc{Enqueue}(v)$ if in$[v] = 0$

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

- Queue Q for source vertices (in$[v] = 0$)
- Repeat: $\textsc{Dequeue}(u \in Q)$, delete u and $u \to v$ from Q,
  output u, $\textsc{Enqueue}(v)$ if in$[v] = 0$

Lemma (Correctness of Kahn's $\textsc{Toposort}$)

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

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

- Queue Q for source vertices (in$[v] = 0$)
- Repeat: $\textsc{Dequeue}(u \in Q)$, delete u and $u \to v$ from Q, output u, $\textsc{Enqueue}(v)$ if in$[v] = 0$

Lemma (Correctness of Kahn's $\textsc{Toposort}$)

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 5.14)

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

Taking courses in few semesters (Problem 5.14)

- 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 5.14)

- 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

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 $O(n+m)$

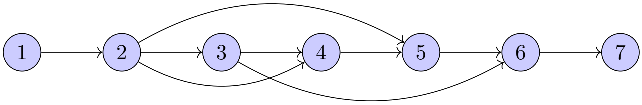
For general (di)graph, HP is NP-hard.

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.

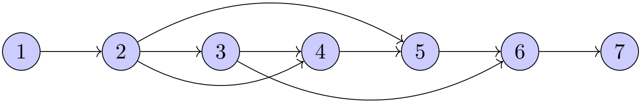


Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

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

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



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

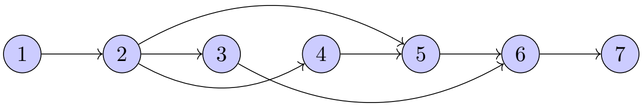
$$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$$

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

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

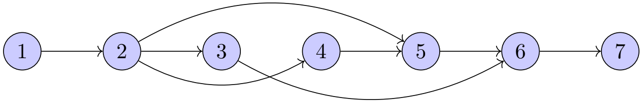
$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$

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



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

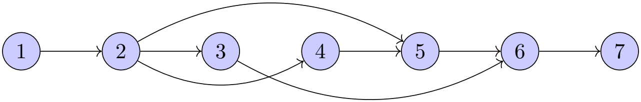
Tarjan's TOPOSORT + Check edges (v_i, v_{i+1})



Kahn's 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 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 SCC algorithm, 1978

"SCCs can be topo-sorted
in decreasing order of their highest finish time."

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

Kosaraju SCC algorithm, 1978 (Problem 4.7)

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

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

Kosaraju SCC algorithm, 1978 (Problem 4.7)

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

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

1st DFS: toposort between SCCs

2nd DFS: reachability within an SCC

Kosaraju 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)

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

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

SCC

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

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

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

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

SCC

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

$\Longleftarrow$: $\exists!$ source

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

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

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

SCC

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

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

$\Longrightarrow$: 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)

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

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

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

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

Impacts of vertices in a digraph (Problem 4.18)

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

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

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

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

$$Q : \forall v : \text{computing impact}(v)$$

2SAT (Problem 4.23)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

2SAT (Problem 4.23)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \to \beta \equiv \overline{\beta} \to \alpha$$

2SAT (Problem 4.23)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \to \beta \equiv \overline{\beta} \to \alpha$$

Implication graph G_I.

2SAT (Problem 4.23)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \to \beta \equiv \overline{\beta} \to \alpha$$

Implication graph G_I.

Theorem (2SAT)

$\exists$ *SCC* $\exists x : v_x \in$ *SCC* $\wedge\, v_{\overline{x}} \in$ *SCC* $\iff$ I *is not satisfiable.*

2SAT (Problem 4.23)

$$I : (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_4)$$

$$\alpha \vee \beta \equiv \overline{\alpha} \to \beta \equiv \overline{\beta} \to \alpha$$

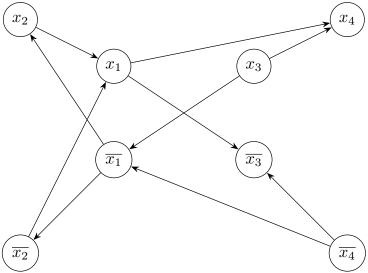
Implication graph G_I.

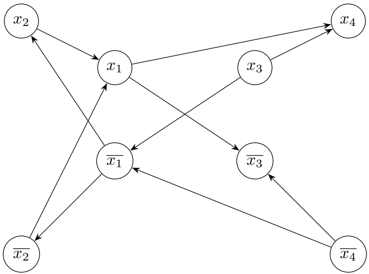
Theorem (2SAT)

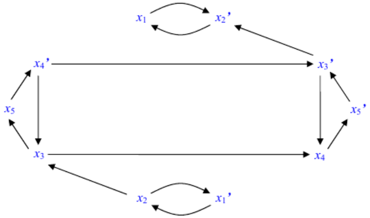
$\exists$ SCC $\exists x : v_x \in$ SCC $\wedge\, v_{\overline{x}} \in$ SCC $\iff I$ is not satisfiable.

Reference:

- *"A Linear-time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas"* by Bengt Aspvall, Michael Plass, and Robert Tarjan, 1979.


x_2
x_4
x_1
x_3
$\overline{x_1}$
$\overline{x_3}$
$\overline{x_2}$
$\overline{x_4}$


x_2
x_4
x_1
x_3
$\overline{x_1}$
$\overline{x_3}$
$\overline{x_2}$
$\overline{x_4}$


x_1
x_2'
x_4'
x_3'
x_5
x_5'
x_3
x_4
x_2
x_1'


THANK YOU