Decompositions of Graphs

(DFS/BFS, DAG, SCC, Bicomp)

Hengfeng Wei

hfwei@nju.edu.cn

June 12, 2019

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 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.

"Depth-First Search And Linear Graph Algorithms"

—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 $k_1 V + k_2 E + k_3$ 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"

—Robert Tarjan

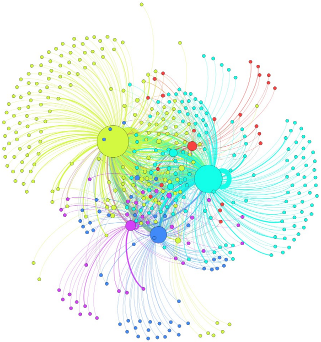
Power of DFS:

Graph Traversal $\Longrightarrow$ Graph Decomposition

Power of DFS:

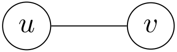
Graph Traversal $\Longrightarrow$ Graph Decomposition

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)

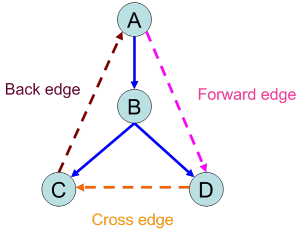
Given a DFS traversal $\implies$ DFS tree:

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)

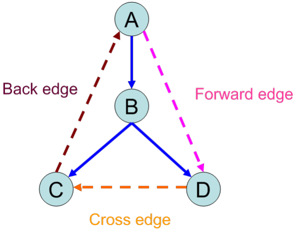
Given a DFS traversal $\implies$ DFS tree:

Tree edge: $\rightarrow$ child

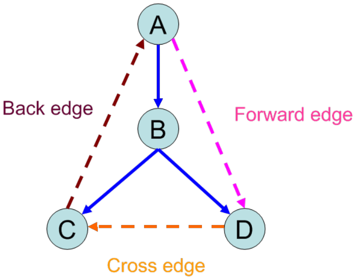
Back edge: $\rightarrow$ ancestor

Forward edge: $\rightarrow$ *nonchild* descendant

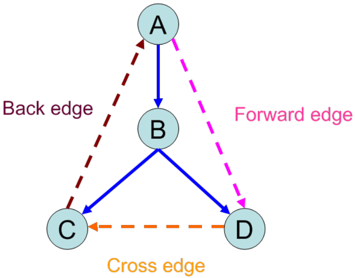
Cross edge: $\rightarrow (\neg\text{ancestor}) \land (\neg\text{descendant})$



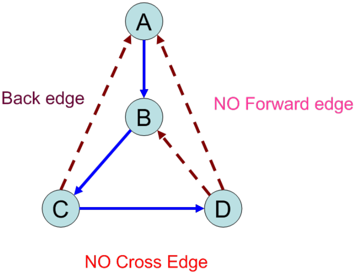
Also applicable to BFS



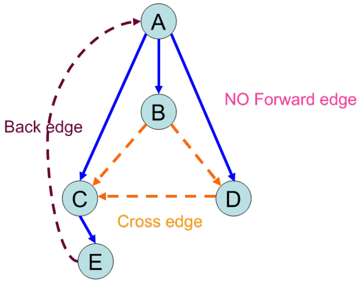
DFS on directed graph



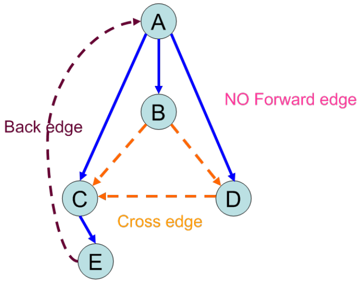
DFS on directed graph



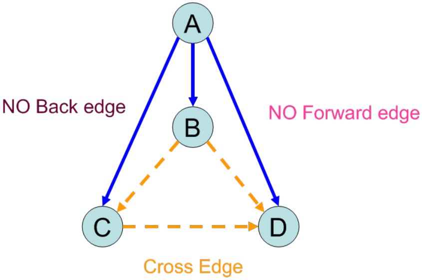
DFS on undirected graph



BFS on directed graph



BFS on directed graph



BFS on undirected graph

Q : How to identify the types of edges in DFS?

Q : How to identify the types of edges in DFS?

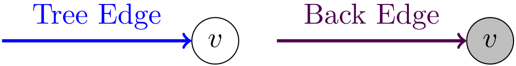
Coloring

Q : How to identify the types of edges in DFS?

Coloring

Q : How to identify the types of edges in DFS?

Coloring



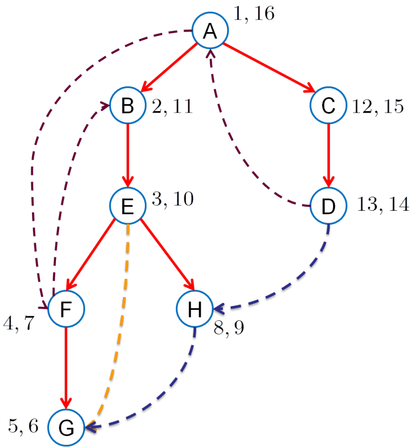
Q : How to identify the types of edges in DFS?

Coloring

Q : How to identify the types of edges in DFS?

Coloring

Life time of vertices in DFS



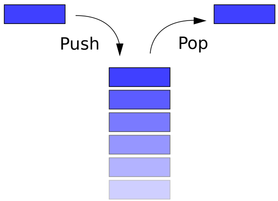
Theorem (Disjoint or Contained (Problem 4.2 : (1)&(2)))

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

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

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

Proof.



□

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

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

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

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

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

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

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

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

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

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)

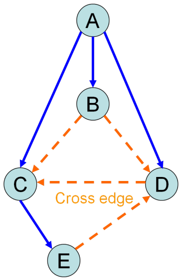
	Digraph	Undirected graph
DFS	back edge $\iff$ cycle	back edge $\iff$ cycle
BFS		cross edge $\iff$ cycle

Cycle detection (Problem 5.8 – 1)

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

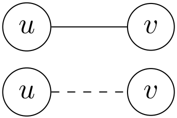
Evasiveness of acyclicity of undirected graphs (Problem 5.8 – 2)

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

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Algorithm $\mathbb{A}$:

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

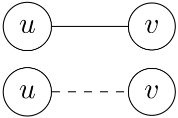
Evasiveness of acyclicity of undirected graphs (Problem 5.8 – 2)

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

Q : Is acyclicity evasive?

By Adversary Argument.

Adversary $\mathcal{A}$:



Hint: Kruskal

Algorithm $\mathbb{A}$:

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

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

$$\Longleftrightarrow$$

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

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

$$\Longleftrightarrow$$

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

Q : Why adjacency matrix?

On digraphs:

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

On digraphs:

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

TOPOSORT by Tarjan (probably), 1976

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

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

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 $O(n+m)$

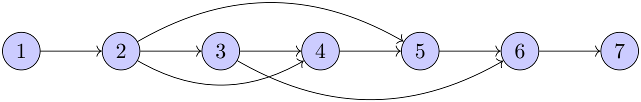
For general (di)graph, HP is NP-hard.

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.

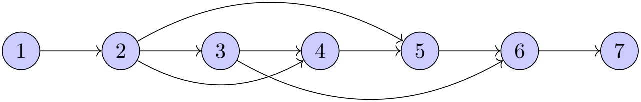


Hamiltonian path in DAG (Problem 4.14)

HP: path visiting each vertex once

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

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



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

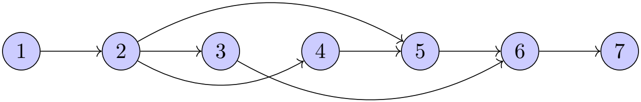
$$\boxed{\text{DAG: } \exists \text{ HP} \iff \exists! \text{ topo. ordering}}$$

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

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

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

Tarjan's TOPOSORT + Check edges (v_i, v_{i+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:

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

SCC: equivalence class over reachability

One-to-all reachability in a digraph (Problem 4.17)

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

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

One-to-all reachability in a digraph (Problem 4.17)

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

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

SCC

One-to-all reachability in a digraph (Problem 4.17)

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

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

SCC

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

One-to-all reachability in a digraph (Problem 4.17)

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

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

SCC

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

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

One-to-all reachability in a digraph (Problem 4.17)

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

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

SCC

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

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

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

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

Impacts of vertices in a digraph (Problem 4.18)

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

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

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)

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

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

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

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

Impacts of vertices in a digraph (Problem 4.18)

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

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

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

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

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

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)

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

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

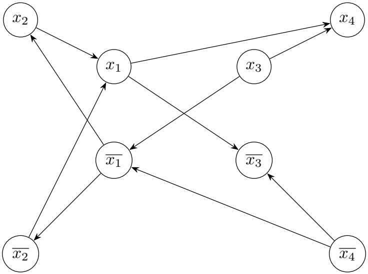
Implication graph G_I.

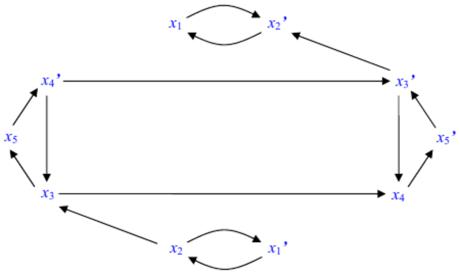
2SAT (Problem 4.23)

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

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

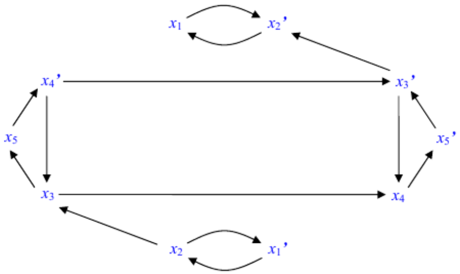
Implication graph G_I.





Theorem (2SAT)

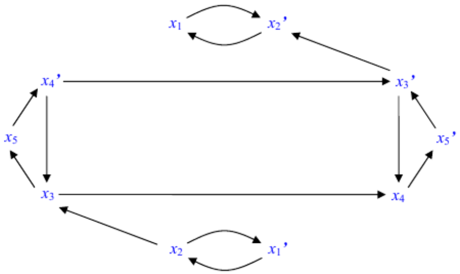
$$\exists\ SCC\ \exists x : v_x \in SCC \wedge v_{\overline{x}} \in SCC \iff I \text{ is not satisfiable.}$$



Theorem (2SAT)

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

$\perp$ for source SCC & $\top$ for sink SCC



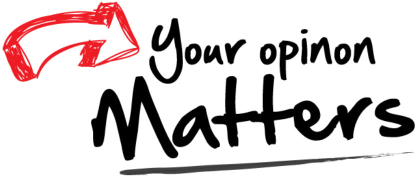
Theorem (2SAT)

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

$\perp$ for source SCC & $\top$ for sink SCC

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


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn