3-6 Decompositions of Graphs

(DFS, Cycle, DAG, Toposort, SCC)

Hengfeng Wei

hfwei@nju.edu.cn

October 29, 2018

Robert Tarjan

John Hopcroft

"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", 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.

"Depth-First Search And Linear Graph Algorithms", Robert Tarjan

"DFS is a powerful technique with many applications."

The Hammer of DFS

Power of DFS:

Graph Traversal $\Longrightarrow$ Graph Decomposition

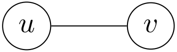
Power of DFS:

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 (Classification of Edges)

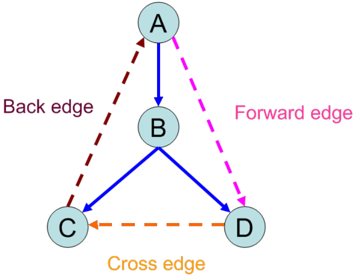
We can define four edge edges in terms of the depth-first forest G_π produced by a DFS on G:

Tree edge: edge in G_π

Back edge: $\rightarrow$ ancestor

Forward edge: $\rightarrow$ descendant (nontree edge)

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



DFS on Undirected Graphs (Problem 22.3-6)

Classifying an edge (u, v) as a tree edge or a back edge according to whether (u, v) or (v, u) is encountered first during the depth-first search

is equivalent to

classifying it according to the ordering of the four types in the classification scheme.

DFS on Undirected Graphs (Problem 22.3-6)

Classifying an edge (u, v) as a tree edge or a back edge according to whether (u, v) or (v, u) is encountered first during the depth-first search

is equivalent to

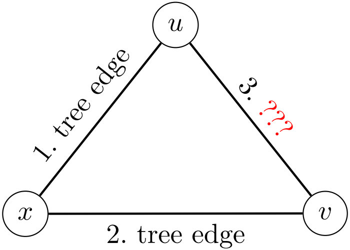
classifying it according to the ordering of the four types in the classification scheme.

DFS on Undirected Graphs (Problem 22.3-6)

Classifying an edge (u, v) as a tree edge or a back edge according to whether (u, v) or (v, u) is encountered first during the depth-first search

is equivalent to

classifying it according to the ordering of the four types in the classification scheme.



Theorem (Theorem 22.10)

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

Proof.

Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that $u.d < v.d$. Then the search must discover and finish v before it finishes u (while u is gray), since v is on u's adjacency list.

If the first time that the search explores edge (u, v), it is in the direction from u to v, then v is undiscovered (white) until that time, for otherwise the search would have explored this edge already in the direction from v to u. Thus, (u, v) becomes a tree edge.

If the search explores (u, v) first in the direction from v to u, then (u, v) is a back edge, since u is still gray at the time the edge is first explored. □

Theorem (Theorem 22.10)

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

Proof.

Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that $u.d < v.d$. Then the search must discover and finish v before it finishes u (while u is gray), since v is on u's adjacency list.

If the first time that the search explores edge (u, v), it is in the direction from u to v, then v is undiscovered (white) until that time, for otherwise the search would have explored this edge already in the direction from v to u. Thus, (u, v) becomes a tree edge.

If the search explores (u, v) first in the direction from v to u, then (u, v) is a back edge, since u is still gray at the time the edge is first explored. □


KEEP
CALM
AND
PROVE IT
KeepCalmAndPosters.com

DFS on Undirected Graphs (Problem 22.3-6)

Classifying an edge (u, v) as a tree edge or a back edge according to whether (u, v) or (v, u) is encountered first during the depth-first search

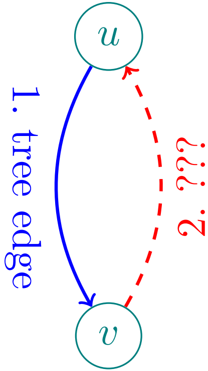
is equivalent to

classifying it according to the ordering of the four types in the classification scheme.

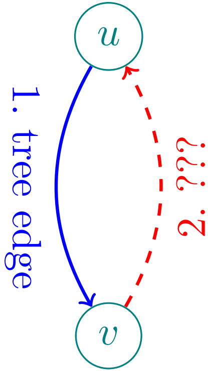
"First Types"	*vs.*	"First Time"
tree edge	$\iff$	tree edge
back edge	$\iff$	back edge

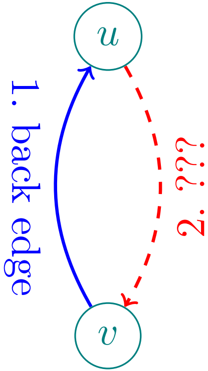
"First Types"	$\Longleftarrow$	"First Time"
tree edge	$\Longleftarrow$	tree edge
back edge	$\Longleftarrow$	back edge

"First Types"	$\Longleftarrow$	"First Time"
tree edge	$\Longleftarrow$	tree edge
back edge	$\Longleftarrow$	back edge



"First Types"	$\Longleftarrow$	"First Time"
tree edge	$\Longleftarrow$	tree edge
back edge	$\Longleftarrow$	back edge





"First Types"	$\Longrightarrow$	"First Time"
tree edge	$\Longrightarrow$	tree edge
back edge	$\Longrightarrow$	back edge

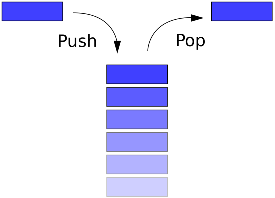
Theorem (Disjoint or Contained)

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

Theorem (Disjoint or Contained)

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

Proof.



□

Edge Types and Life Time of Vertices in DFS

$$\forall u \to v :$$

- 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

$$\forall u \to v :$$

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

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

Edge Types and Life Time of Vertices in DFS

$$\forall u \to v :$$

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

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

Edge Types and Life Time of Vertices in DFS

$$\forall u \to v :$$

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

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

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

Edge Types and Life Time of Vertices in DFS

$$\forall u \to v :$$

- 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

$$\forall u \to v:$$

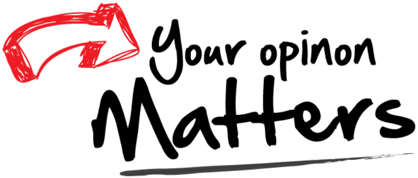
- tree/forward edge: $[_u\ [_v\ ]_v\ ]_u$
- back edge: $[_v\ [_u\ ]_u\ ]_v$
- cross edge: $[_v\ ]_v\ [_u\ ]_u$

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

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

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


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn