3-6 Decompositions of Graphs

(Part II: DFS, SCC, Bicomponent)

Hengfeng Wei

hfwei@nju.edu.cn

November 05, 2018

The Power of the Hammer of DFS

Graph Traversal $\Longrightarrow$ Graph Decomposition

Structure! Structure! Structure!

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.

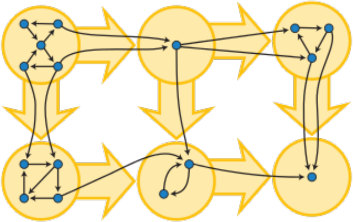
Key words. Algorithm, backtracking, biconnectivity, connectivity, depth-first, graph, search, spanning tree, strong-connectivity.

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

Tarjan's SCC Bicomponent

Theorem (Digraph as DAG)

Every digraph is a dag of its SCCs.



Two tiered structure of digraphs:

digraph $\equiv$ a dag of SCCs

SCC: equivalence class over reachability

Semiconnected Digraph (Problem 22.5-7)

$$G = (V, E)$$

$$\forall u, v \in V : u \leadsto v \lor v \leadsto u$$

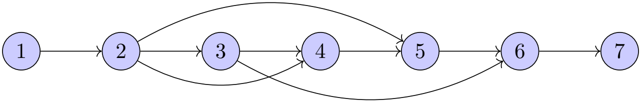
$$\boxed{\text{digraph} \equiv \text{a dag of SCCs}}$$

G is semiconnected $\iff$ The dag of SCCs is semiconnected

Is a DAG semiconnected?

A DAG is semiconnected $\Longrightarrow$ $\exists!$ topo. ordering

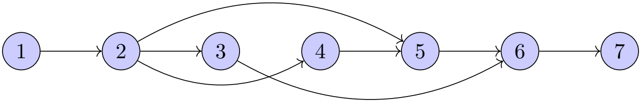
A DAG is semiconnected $\Longleftarrow$ $\exists!$ topo. ordering



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

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

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



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

Lemma (22.14)

Let C and C' be two SCCs in a digraph G.

$$u \in C \to v \in C' \implies f[C] > f[C']$$

$$d[U] = \min_{u \in U} d[u] \qquad f[U] = \max_{u \in U} f[u]$$

$$\text{DAG} \implies \boxed{u \to v \iff f[u] > f[v]}$$

Kosaraju's SCC algorithm, 1978

SCCs can be topo-sorted in decreasing order of their $f[\cdot]$.

The vertice v with the highest $f[v]$ is in a source SCC.

(I) DFS on G; DFS/BFS on G^T ($f[\cdot] \downarrow$)

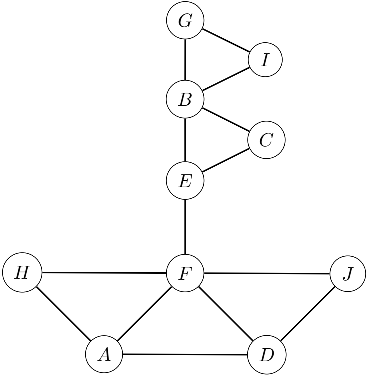
(II) DFS on G^T; DFS/BFS on G ($f[\cdot] \downarrow$)

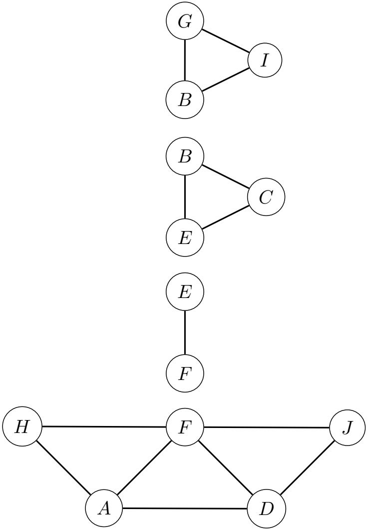
Definition (Biconnected Graph)

A connected undirected graph is *biconnected* if it contains no "cut-nodes".

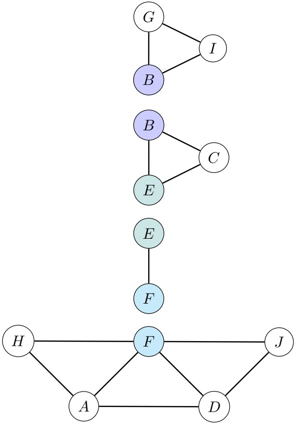
Definition (Biconnected Component (Bicomponent))

A *bicomponent* of an undirected graph is a maximal biconnected subgraph.





Paritition of edges (not of nodes)



Theorem (Cut-nodes and Bicomponents)

Let $G_i = (V_i, E_i)$ be the bicomponents of a connected undirected graph $G = (V, E)$.

(a)

$$\forall i \neq j : \left| V_i \cap V_j \right| \leq 1$$

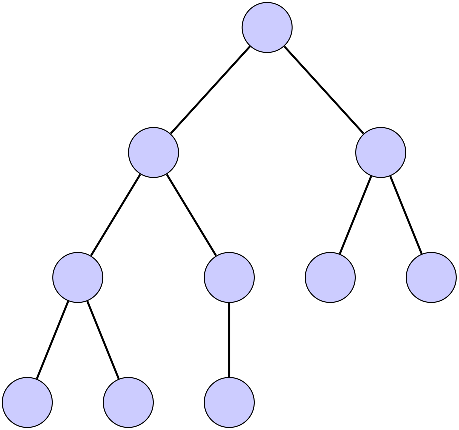
(b)

$$v \text{ is a cut-node } \iff \exists i \neq j : v \in V_i \cap V_j$$

The Power of the Hammer of DFS on Undirected Graphs

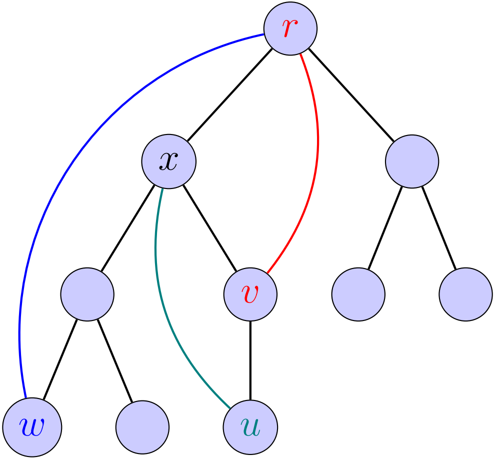
Theorem (Theorem 22.10)

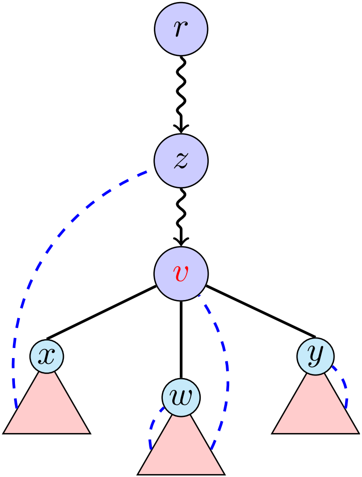
In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.



Cut-nodes? Bicomponents?

Bicomp: Back!




r
z
v
x
w
y

Theorem (Characterization of Cut-nodes)

In a DFS tree, v is a cut-node

$$\Longleftrightarrow$$

(a) *v is the root and $deg(v) \geq 2$*

(b) *v is not the root and some subtree of v has no back edge to a proper ancestor of v*

(I) When and how to identify a bicomponent?

back$[v]$:

The earliest ancestor v can get

by following tree edges $\mathbb{T}$ and back edges $\mathbb{B}$.

(II) When and how to initialize&update back$[v]$?

$\text{back}[v]$:

The earliest ancestor v can get

by following tree edges $\mathbb{T}$ and back edges $\mathbb{B}$.

$$\text{back}[v] = \min \begin{cases} \{v\} \\ \{w \mid (v, w) \in \mathbb{B}\} \\ \{\text{back}[w] \mid (v, w) \in \mathbb{T}\} \end{cases}$$

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

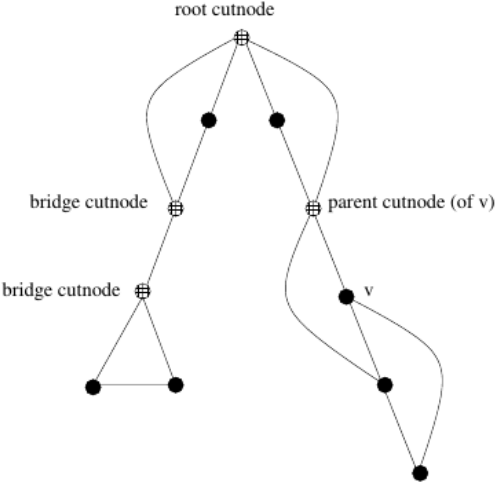
back edge $(v \rightarrow w)$: $\text{back}[v] = \min\{\text{back}[v], d[w]\}$

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

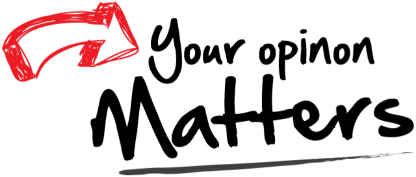
Backtracking from w to v : $\text{back}[w] \geq d[v]$

After-class Exercise: Bicomp

procedure BICOMP(G)
    Here: Your Code Based on the DFS Framework


root cutnode
bridge cutnode
parent cutnode (of v)
bridge cutnode
v


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn