

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*

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.”

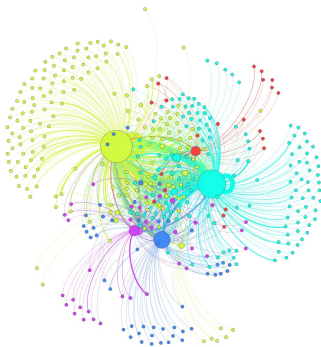
“Depth-First Search And Linear Graph Algorithms”

—Robert Tarjan

Power of DFS:


Graph Traversal \implies Graph Decomposition

Structure! Structure! Structure!



Graph *structure* induced by DFS:

states of 

types of 

life time of :

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

$d[v]$: BICOMP

$f[v]$: TOPOSORT, SCC

Definition (Classifying edges)

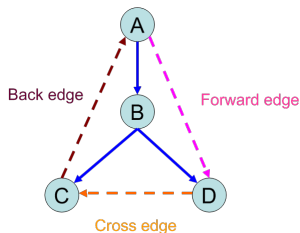
Given a DFS traversal \implies DFS tree:

Tree edge: \rightarrow child

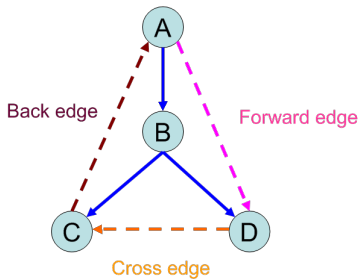
Back edge: \rightarrow ancestor

Forward edge: \rightarrow *nonchild* descendant

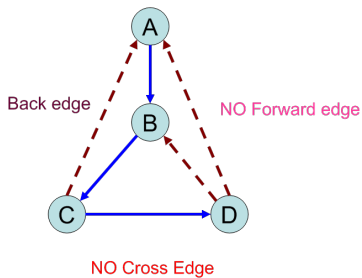
Cross edge: $\rightarrow (\neg \text{ancestor}) \wedge (\neg \text{descendant})$



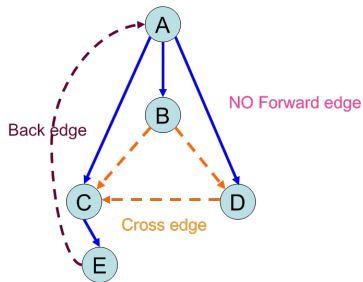
Also applicable to BFS



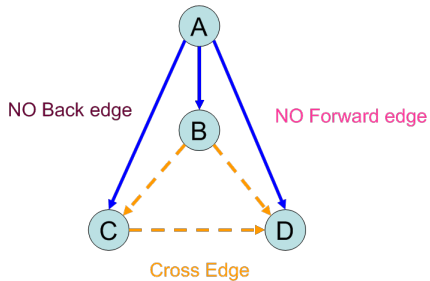
DFS on directed graph



DFS on undirected graph



BFS on directed graph



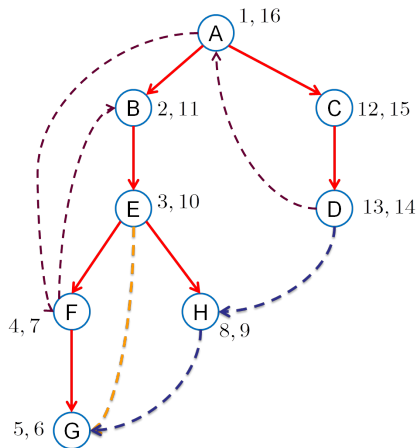
BFS on undirected graph

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

Coloring



Life time of vertices in DFS



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

$$\forall u \rightarrow v :$$

- ▶ tree/forward edge: $[u [v]v]u$
- ▶ back edge: $[v [u]u]v$
- ▶ cross edge: $[v]v [u]u$

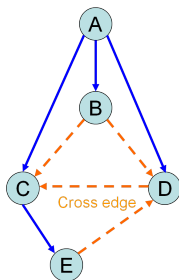
$$f[v] < d[u] \iff \text{cross edge}$$

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

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

Cycle detection (Problem 5.8 – 1)

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



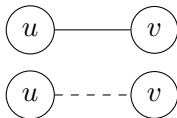
Evasiveness of acyclicity of **undirected** graphs (Problem 5.8 – 2)

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

Q : Is **acyclicity** evasive?

By Adversary Argument.

Adversary \mathcal{A} :



Algorithm \mathcal{A} :

CHECKEDGE(u, v)

Hint: Kruskal



$$\begin{aligned}
 \mathbb{A} : \text{CHECKEDGE}(u, v) &\leftarrow \mathcal{A} : \text{---} \begin{array}{c} (u) \text{---} (v) \end{array} \\
 &\iff \\
 \mathcal{A} : \nexists \text{ cycle} \in G + &\begin{array}{c} (u) \text{---} (v) \end{array}
 \end{aligned}$$

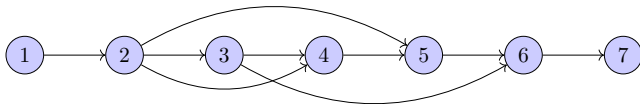
\mathcal{Q} : Why adjacency matrix?

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.

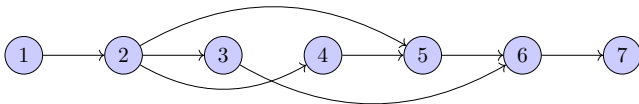


DAG: \exists HP $\iff \exists!$ topo. ordering

DAG: \exists HP $\iff \exists!$ topo. ordering



By contradiction; Perform Toposort; Swap



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.

Two tiered structure of digraphs:

digraph \equiv a dag of SCCs

SCC: equivalence class over reachability

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

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

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

SCC

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

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

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

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

Impacts of vertices in a digraph (Problem 4.18)

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

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

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

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

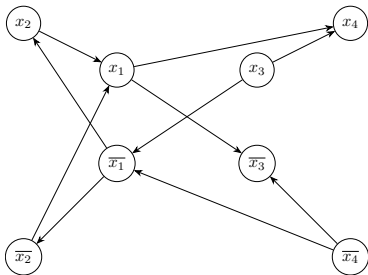
$Q : \forall v, \text{ computing } \text{impact}(v)$

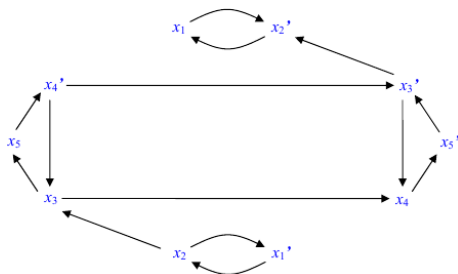
2SAT (Problem 4.23)

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

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

Implication graph G_I .





Theorem (2SAT)

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

“F” for source SCC & “T” for sink SCC

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





Office 302

Mailbox: H016

hfwei@nju.edu.cn