3-6 Decompositions of Graphs
(DFS, DAG, Toposort, Cycle, 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
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 undirected 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 $k_1, k_2, \text{and } k_3$, 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†

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 undirected 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 $\implies$ Graph Decomposition

*Structure!* *Structure!* *Structure!*
Graph *structure* induced by DFS:

- **States of** $v$
- **Types of** $u ightarrow v$
Graph *structure* induced by DFS:

- **states of** $v$
- **types of** $u \rightarrow v$

**lifetime of** $v$:

- $v : d[v], f[v]$
- $f[v]$: **TOPSORT, SCC**
- $d[v]$: **BICOMP** (Problem 22-2)
Definition (Classification of Edges)

We can **define** four edge types in **terms of the depth-first forest** $G_\pi$ produced by a DFS on $G$:

- **Tree edge**: edge in $G_\pi$
- **Back edge**: $\rightarrow$ ancestor
- **Forward edge**: $\rightarrow$ descendant (nontree edge)
- **Cross edge**: $\rightarrow (\neg\text{ancestor}) \land (\neg\text{descendant})$
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.

![Diagram of DFS on Undirected Graphs]

- 1. tree edge
- 2. tree edge
- 3. ???
Thanks. However, I am still confused. I have added an example to explain my confusion. Could you please have a look at it? – hengxin 3 hours ago

I am checking ... It looks like the answer is clear to me. – Apass.Jack 3 hours ago

I will let you try following the procedure in the book step by step for the next few minutes. Or tell me if you have already tried. (Hopefully, I will visit your university...) (this comment will be removed later.) – Apass.Jack 3 hours ago

I am going to update my answer now. It may take 5 minutes to half an hour. – Apass.Jack 2 hours ago

:) I am waiting (both on the Internet and in my university). – hengxin 2 hours ago

add a comment
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.
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 Type” vs. “First Time”

<table>
<thead>
<tr>
<th>Tree edge</th>
<th>Back edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇐⇒</td>
<td>⇐⇒</td>
</tr>
</tbody>
</table>
“First Type” ⇐ “First Time”

tree edge ⇐ tree edge

back edge ⇐ back edge
“First Type” ⇐ “First Time”

tree edge ⇐ tree edge

back edge ⇐ back edge
“First Type” ⇐ “First Time”

tree edge ⇐ tree edge

back edge ⇐ back edge

Hengfeng Wei (hfwei@nju.edu.cn)
“First Type” $\implies$ “First Time”

tree edge $\implies$ tree edge

back edge $\implies$ back edge
“First Type” $\implies$ “First Time”

tree edge $\implies$ tree edge

back edge $\implies$ back edge
“First Type” $\implies$ “First Time”

- tree edge $\implies$ tree edge
- back edge $\implies$ back edge
tree edge
tree edge

back edge

forward edge

cross edge
Edge Types and Lifetime of Vertices in DFS

∀u → v:

- tree/forward edge: [u [v ]v ]u
- back edge: [v [u ]u ]v
- cross edge: [v ]v [u ]u
Edge Types and Lifetime of Vertices in DFS

∀u → v:

- tree/forward edge: \([u \leftarrow v \rightarrow u]\)
- back edge: \([v \leftarrow u \rightarrow v]\)
- cross edge: \([v \rightarrow u \leftarrow v]\)

\[f[v] < d[u] \iff \text{cross edge}\]
Edge Types and Lifetime of Vertices in DFS

\[ \forall u \rightarrow v : \]

- tree/forward edge: \([u \leftarrow v \rightarrow u]\)
- back edge: \([v \leftarrow u \rightarrow v]\)
- cross edge: \([v \rightarrow u \rightarrow v]\)

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

\[ f[u] < f[v] \iff \text{back edge} \]
Edge Types and Lifetime of Vertices in DFS

\[ \forall u \rightarrow v : \]

- tree/forward edge: \([u \rightarrow v \rightarrow u]\)
- back edge: \([v \leftarrow u \rightarrow v]\)
- cross edge: \([v \rightarrow u \rightarrow v]\)

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

\[ \not\exists \text{ cycle} \implies u \rightarrow v \iff f[v] < f[u] \]
On digraphs:

\[ \not\exists \text{ back edge} \iff \text{DAG} \iff \exists \text{ topo. ordering} \]
On digraphs:

\[ \neg \text{back edge} \iff \text{DAG} \iff \exists \text{topo. ordering} \]

**Toposort** by Tarjan (probably), 1976

\[ \neg \text{cycle} \implies u \to v \iff f(v) < f(u) \]
On digraphs:

\[ \# \text{ back edge} \iff \text{DAG} \iff \exists \text{ topo. ordering} \]

**Toposort** by Tarjan (probably), 1976

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

Sort vertices in *decreasing* order of their *finish* times.
Cycle Detection (Problem 22.4-3)

Whether an undirected graph $G$ contains a cycle?

$O(|V|)$
Cycle Detection (Problem 22.4-3)

Whether an undirected graph $G$ contains a cycle?

$O(|V|)$

tree: $|E| = |V| - 1 \implies$ check $|E| \geq |V|$
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>back edge $\iff$ cycle</td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hengfeng Wei (hfwei@nju.edu.cn)
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>back edge \iff cycle</td>
<td>back edge \iff cycle</td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hengfeng Wei (hfwei@nju.edu.cn)
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>back edge $\iff$ cycle</td>
<td>back edge $\iff$ cycle</td>
</tr>
<tr>
<td>BFS</td>
<td>cross edge $\iff$ cycle</td>
<td></td>
</tr>
</tbody>
</table>
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS</strong></td>
<td>back edge $\iff$ cycle</td>
<td>back edge $\iff$ cycle</td>
</tr>
<tr>
<td><strong>BFS</strong></td>
<td>back edge $\implies$ cycle</td>
<td>cross edge $\iff$ cycle</td>
</tr>
<tr>
<td></td>
<td>cycle $\not\iff$ back edge</td>
<td></td>
</tr>
</tbody>
</table>
## Cycle Detection

<table>
<thead>
<tr>
<th></th>
<th>Digraph</th>
<th>Undirected graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>back edge $\iff$ cycle</td>
<td>back edge $\iff$ cycle</td>
</tr>
<tr>
<td>BFS</td>
<td>back edge $\implies$ cycle</td>
<td>cross edge $\iff$ cycle</td>
</tr>
</tbody>
</table>
Theorem (Digraph as DAG)

*Every digraph is a dag of its SCCs.*
Theorem (Digraph as DAG)

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
digraph $\equiv$ a dag of SCCs

Kosaraju’s SCC algorithm, 1978

“SCCs can be topo-sorted in decreasing order of their highest finish time.”
digraph $\equiv$ a dag of SCCs

Kosaraju’s SCC algorithm, 1978

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

The vertex with the highest finish time is in a source SCC.
digraph \equiv a \ dag \ of \ SCCs

Kosaraju’s SCC algorithm, 1978

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

The vertex with the highest finish time is in a source SCC.

(I) DFS on $G$; DFS on $G^T$
digraph \equiv a \; dag \; of \; SCCs

Kosaraju’s SCC algorithm, 1978

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

The vertex with the highest finish time is in a source SCC.

(I) DFS on \( G \); DFS on \( G^T \)

(II) DFS on \( G^T \); DFS on \( G \)
digraph \equiv a \text{ dag of SCCs}

Kosaraju’s SCC algorithm, 1978

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

The vertex with the highest finish time is in a source SCC.

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

(II) DFS on $G^T$; DFS/BFS on $G$
Semiconnected Digraph (Problem 4.14)

$$\forall u, v \in V : u \leadsto v \lor v \leadsto u$$
Semiconnected Digraph (Problem 4.14)

\[ \forall u, v \in V : u \sim v \lor v \sim u \]

\[ \text{digraph} \equiv \text{a dag of SCCs} \]
Semiconnected Digraph (Problem 4.14)

\[ \forall u, v \in V : u \sim v \lor v \sim u \]

\text{digraph} \equiv \text{a dag of SCCs}
Semiconnected Digraph (Problem 4.14)

\[ \forall u, v \in V : u \sim v \lor v \sim u \]

digraph \equiv a dag of SCCs

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

Tarjan’s TOPOSORT + Check edges \((v_i, v_{i+1})\)
DAG: Semiconnected $\iff \exists!$ topo. ordering

Tarjan’s Toposort + Check edges $(v_i, v_{i+1})$
Office 302
Mailbox: H016
hfwei@nju.edu.cn