Algorithmen II

Peter Sanders

Exercise:
Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:
http://algo2.iti.kit.edu/AlgorithmenII_WS20.php
4 Applications of DFS
DFS Schema for $G = (V, E)$

unmark all nodes;  init

foreach $s \in V$ do
  if $s$ is not marked then
    mark $s$
    root$(s)$
    $\text{DFS}(s, s)$

Procedure $\text{DFS}(u, v : \text{NodeId})$

  foreach $(v, w) \in E$ do
    if $w$ is marked then $\text{traverseNonTreeEdge}(v, w)$
    else $\text{traverseTreeEdge}(v, w)$
      mark $w$
      $\text{DFS}(v, w)$
  backtrack$(u, v)$  // return from $v$ along the incoming edge
DFS Ordering

init: \( \text{dfsPos} = 1 : 1..n \)

root(s): \( \text{dfsNum}[s] := \text{dfsPos}++ \)

traverseTreeEdge(v, w): \( \text{dfsNum}[w] := \text{dfsPos}++ \)

\[ u \prec v \iff \text{dfsNum}[u] < \text{dfsNum}[v] . \]

Observation:

Nodes on the recursion stack are sorted w.r.t. \( \prec \)

\[ \rightarrow \text{tree} \quad \rightarrow \text{backward} \quad \rightarrow \text{cross} \quad \rightarrow \text{forward} \]
Finishing Time

init: \( \text{finishingTime} = 1 : 1..n \)

\( \text{backtrack}(u,v) : \quad \text{finishTime}[v] := \text{finishingTime}++ \)
Strongly Connected Components

Consider the relation $\leftrightarrow^*$ where
$u \leftrightarrow^* v$ if $\exists$ path $\langle u, \ldots, v \rangle$ and $\exists$ path $\langle v, \ldots, u \rangle$.

**Observation:** $\leftrightarrow^*$ is an equivalence relation

The equivalence classes of $\leftrightarrow^*$ are called strongly connected components.
Strongly Connected Components –
Abstract Algorithm

\[ G_c := (V, \emptyset = E_c) \]

\textbf{foreach} edge \( e \in E \) \textbf{do}

\textbf{invariant} SCCs of \( G_c \) are known

\[ E_c := E_c \cup \{e\} \]
Shrunken Graph

\[ G_c^s = (V_c^s, E_c^s) \]

Nodes: SCCs of \( G_c \).

Edges: \((C, D) \in E_c^s \iff \exists (c, d) \in E_c : c \in C \land d \in D\)

Observation: the shrunken graph is acyclic
Effects of a New Edge $e$ on $G_c$, $G^s_c$

- **internal to an SCC:** Nothing changes

- **between two SCCs:**
  - **no cycle:** new edge in $G^s_c$
  - **closing a cycle:** SCCs on the cycle collapse.
More Concretely: Finding SCCs with DFS

[Cheriyan/Mehlhorn 96, Gabow 2000]

$V_c = \text{marked nodes}$

$E_c = \text{edges explored so far}$

**Active nodes**: marked but not yet finished.

SCCs of $G_c$:

- **not reached**: unmarked nodes
- **open**: contains active nodes
- **closed**: all nodes finished

component[w] is the representative of an SCC.

Nodes of open (closed) components are called open (closed)
Invariants of $G_c$

1. Edges from closed nodes lead to closed nodes

2. Open components $S_1, \ldots, S_k$ form a path in $G_c^s$.

3. Representatives partition the open components w.r.t. their dfsNum.

![Diagram showing open nodes ordered by dfsNum and their connection to representatives $r_1, r_2, \ldots, r_k$. The open components $S_1, S_2, \ldots, S_k$ are connected in a path with arrows indicating the direction of edges.]
**Lemma:** Finished SCCs of $G_c$ are SCCs of $G$

Consider a closed node $v$ and an arbitrary node $w$ in the SCC of $v$ w.r.t. $G$.

To prove: $w$ is closed and in the same SCC of $G_c$ as $v$.

Consider cycle $C$ containing $v$, $w$.

Inv. 1: nodes of $C$ are closed.

Closed nodes are finished.

Edges out of finished nodes have been explored.

Hence, all edges of $C$ are in $G_c$. $\square$
**Representation of Open Components**

Two stacks ordered by dfsNum ascendingly

- **oReps**: representatives of open components
- **oNodes**: all open nodes

![Diagram showing open nodes ordered by dfsNum]
init

component : NodeArray of Nodeld // SCC representatives
oReps=⟨⟩ : Stack of Nodeld // representatives of open SCCs
oNodes=⟨⟩ : Stack of Nodeld // all nodes in open SCCs

All invariants are satisfied.
( Neither open nor closed nodes)
root(s)

oReps.push(s)  // new open
oNodes.push(s)  // component

\{s\} is the only open component.
All invariants remain valid

\begin{center}
\begin{tikzpicture}
  \node (s) at (0,0) {s};
  \node (s1) at (1,1) {$S_1$};
  \node {current node};
  \draw[->] (s) -- (s1);
\end{tikzpicture}
\end{center}

open nodes ordered by dfsNum
traverseTreeEdge(\(v, w\))

\begin{align*}
\text{oReps.push}(w) & \quad \text{// new open component} \\
\text{oNodes.push}(w) & \quad \text{// component}
\end{align*}

\{w\} is a new open component.

dfsNum(w) > \text{all others}.

\[ \Rightarrow \text{All invariants remain valid} \]

open nodes ordered by dfsNum
traverseNonTreeEdge\( (v, w) \)

**if** \( w \in \text{oNodes} \) **then**

**while** \( w \prec oReps.\text{top} \) **do** oReps.pop

\( w \not\in \text{oNodes} \Rightarrow \text{w is closed} \Rightarrow \text{Lemma(\(*\)} \Rightarrow \text{edge is not interesting} \)

\( w \in \text{oNodes}: \text{collapse open SCCs on the cycle} \)
backtrack\((u, v)\)

\[
\text{if } v = \text{oReps.top then}
\]
\[
\text{oReps.pop} \\
\text{repeat}
\]
\[
w := \text{oNodes.pop} \\
\text{component}[w] := v
\]
\[
\text{until } w = v
\]

To prove: invariants remain valid…
backtrack\((u, v)\)

\[
\text{if } v = \text{oReps.top then} \\
\text{oReps.pop} \\
\text{repeat} \\
\text{ } w := \text{oNodes.pop} \\
\text{component}[w] := v \\
\text{until } w = v
\]

\textbf{Inv. 1:} edges from closed nodes lead to closed nodes.
backtrack\((u, v)\)

\[\text{if } v = \text{oReps.top} \text{ then} \]
\[\text{oReps.pop} \]

\[\text{repeat} \]
\[w := \text{oNodes.pop} \]
\[\text{component}[w] := v \]
\[\text{until } w = v \]

\textbf{Inv. 2:} open components } S_1, \ldots, S_k \text{ form a path in } G_s^c \]

OK. (\(S_k\) may be removed)
backtrack\((u, v)\)

\[ \text{if } v = \text{oReps.top then} \]
\[ \text{oReps.pop} \]
\[ \text{repeat} \]
\[ w := \text{oNodes.pop} \]
\[ \text{component}[w] := v \]
\[ \text{until } w = v \]

**Inv. 3:** representatives partition the open components w.r.t. their dfsNum.

OK. \((S_k \text{ may be removed})\)
Example

root(a) traverse(a,b) traverse(b,c)

unmarked marked finished
nonrepresentative node
representative node
nontraversed edge closed SCC
traversed edge open SCC
unmarked marked finished
● nonrepresentative node
□ representative node
→ nontraversed edge
→ traversed edge

closed SCC
open SCC
backtrack(a,a)

unmarked marked finished

nontraversed edge traversed edge

closed SCC open SCC
root(d) traverse(d,e) traverse(e,f) traverse(f,g)

unmarked marked finished
nonrepresentative node
representative node
nontraversed edge closed SCC
traversed edge open SCC
unmarked marked finished

nonrepresentative node

representative node

nontraversed edge

closed SCC

traversed edge
open SCC
traverse(e, g)  traverse(e, h)  traverse(h, i)

unmarked marked finished
○ ○ ○ nonrepresentative node
■ ■ ■ representative node
→ nontraversed edge  closed SCC
→ traversed edge  open SCC
traverse(i,e)

unmarked marked finished

nontraversed edge

traversed edge

nonrepresentative node

representative node

closed SCC

open SCC
traverse(i,j)  traverse(j,c)  traverse(j,k)

unmarked  marked  finished

nonrepresentative node  representative node

nontraversed edge  closed SCC  traversed edge  open SCC
traverse(k,d)

unmarked marked finished

nonrepresentative node

representative node

nontraversed edge
closed SCC

traversed edge
open SCC
backtrack(j,k) backtrack(i,j) backtrack(h,i) backtrack(e,h) backtrack(d,e)

unmarked marked finished

nonrepresentative node

representative node

nontraversed edge closed SCC

traversed edge open SCC
backtrack(d,d)

unmarked marked finished

nonrepresentative node

representative node

nontraversed edge

closed SCC

traversed edge

open SCC
Summary: Computing SCCs

- Simple instantiation of the DFS template
- Nontrivial correctness proof
- Running time $O(m + n)$: (at most $n$ push/pop operations, resp.)
- A single iteration

Implementation details:
Mehlhorn, Näher, Sanders
Engineering DFS-Based Graph Algorithms
arxiv.org/abs/1703.10023
2-Connected Components (Undirected)

Components remain connected when removing a single node.

(Partitioning of the edges)

Possible in $O(m + n)$ time with an algorithm similar to that for SCCs.
More DFS-based Linear Time Algorithms

- 3-connected components
- Planarity testing
- Embedding of planar graphs