Graph Algorithms

- Strongly connected components
- Topological sort
- Single-source shortest path

Digraph notions

- Vertex \( y \) is \textit{reachable} from \( x \) if there is a directed path in \( G \) from \( x \) to \( y \).
  (By convention \( x \) is reachable from \( x \) by a directed path of length 0.)
- Vertices \( x \) and \( y \) are \textit{strongly connected} if \( x \) is reachable from \( y \) and \( y \) is reachable from \( x \).

Strongly-connected vertices?

- Vertices form \textit{strongly connected components}.
  (Equivalence classes)
DFS Application

- Identify the strongly connected components of a digraph G.

Depth-First(x)

Depth-First(x)
Mark x visited
For each edge <x,y>
  If y is unvisited then
    DFS(y)

DFS(G)

DFS(G)
While G has an unvisited vertex x:
  Depth-First(x)

Selection rule

• We’ll use alphabetical priority

DFS(G)

DFS(G)
Choose alphabetically
While G has an unvisited vertex x:
  Depth-First(x)

Depth-First(x)

Depth-First(x)
Mark x visited
For each edge <x,y>
  If y is unvisited then
    DFS(y)
Choose alphabetically
DFS(G)
Alphabetical priority

a is unvisited so DFS(a)

Visit a
Find <a,d> edge and call
DFS(d)

DFS(a)
Alphabetical priority

Visit a
Find <a,d> edge and call
DFS(d)

DFS(d)
Alphabetical priority

Visit d
All out-edges checked so
return

DFS(b)
Alphabetical priority

Visit b
Find edge <b,c> and call
DFS(c)

DFS(G)
Alphabetical priority

a is unvisited so DFS(a)
b is unvisited so DFS(b)

Call Stack:
DFS(G)
DFS(c)
Alphabetical priority

Visit c
Find edge \langle c,a \rangle – no action
Find edge \langle c,b \rangle – no action
All out-edges checked so return

Call Stack:
DFS(c)
DFS(b)
DFS(G)

DFS(b)
Alphabetical priority

Visit b
Find edge \langle b,c \rangle and call DFS(c)
Find edge \langle b,d \rangle – no action
All out-edges checked so return

Call Stack:
DFS(b)
DFS(G)

DFS(G)
Alphabetical priority

a is unvisited so DFS(a)
b is unvisited so DFS(b)
All nodes checked so return

Call Stack:
DFS(G)

What is the running time of DFS?
• O(m+n)
  • Every vertex is pushed onto the stack once and popped from the stack once.
  • Each out-edge is inspected once.

Strongly Connected Components
• Input: Digraph G
• Output: The strongly connected components of G.

Naïve Algorithm
• Are x and y in the same connected component?
  • Mark all vertices unvisited and call DFS(x)
  • If y unvisited return no
  • Mark all vertices unvisited and call DFS(y)
  • If x unvisited return no
  • Return yes
Naïve algorithm

- Worst case: $n^2$ calls to DFS(x)

All little more sophistication please…

- We can find the strongly connected components of $G$ with two calls to DFS($G$)

Three ideas

- DFS Forest
- Timestamps
- Reversal of $G$

DFS Forest

- The DFS Forest of $G$ is the subgraph consisting of
  - Every vertex of $G$
  - Each edge traversed in DFS($G$)

Different selection rules give different results

WARNING

- DFS Forests are sometimes
What is the connection?

- What can we say about strongly connected components of $G$ vs. trees in a DFS forest of $G$?

```
G:  
  a    d
 /    /  
|     |   
 a    d
```

A DFS forest of $G$:
```
  c    b  
  c    b
```

What can we say?

- What is the relationship between the trees of a DFS forest and the strongly connected components of the graph?

```
DFS forest of $G$
```

What can we say?

- If $x$ and $y$ are in the same strongly connected component of $G$ then
- If $x$ and $y$ are in different strongly connected components of $G$ then

What can we say?

- If $x$ and $y$ are in the same tree in a DFS forest of $G$ then
- If $x$ and $y$ are in different different trees in a DFS forest of $G$ then

What can we say?

- If $x$ and $y$ are in the same strongly connected component of $G$ then
- If $x$ and $y$ are in different strongly connected components of $G$ then

Three ideas

- DFS Forest of $G$
- Timestamps
- Reversal
DFS(G)
Alphabetical order
Record first-arrival and last-departure times.

<table>
<thead>
<tr>
<th></th>
<th>First-arrival</th>
<th>Last-Departure</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DFS(G)
Alphabetical order

<table>
<thead>
<tr>
<th></th>
<th>First-arrival</th>
<th>Last-Departure</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Three ideas
• DFS Forest
• Timestamps
• Reversal of G

(G^R)^R: Reverse the edges of G^R

(G^R)^R = G

Reachability
X is reachable from Y in G ⇔ Y is reachable from X in G^T
Reachability

X is reachable from Y in G as Y is reachable from X in \(G^T\)

So the Strongly Connected Components of G and \(G^R\) are the same!

SCC

- DFS(G) with timestamp (alphabetical or other order)
- DFS(G\(^R\)) using last-departure time decreasing order
- The trees in the DFS forest of G\(^R\) correspond to the connected components of G

DFS(G)

<table>
<thead>
<tr>
<th></th>
<th>First-arrival</th>
<th>Last-Departure</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

DFS(G\(^R\))

Order: b,c,a,d

DFS Forest

Order: b,c,a,d
Why does this work?

1. If x and y are in the same strongly connected component of G then they are in the same tree of the DFS forest of \( G^R \).

2. If x and y are in the same tree of the DFS forest of \( G^R \) then they are in the same strongly connected component of G.

Claim 1 (Easy)

1. If x and y are in the same strongly connected component of G then they are in the same tree of the DFS forest of \( G^R \).
   - If x and y are in the same SCC of G then x and y are in the same SCC of \( G^R \).
   - If x and y are in the same SCC of \( G^R \) then they are in the same tree of the DFS forest of \( G^R \).

Claim 2

2. If x and y are in the same tree of the DFS forest of \( G^R \) then they are in the same strongly connected component of G.

Proof of Claim 2

- We know that Last-departure(x) < Last-departure(r).
- If Last-departure(x) < First-arrival(r) then r is not reachable from x in G =>
- So First-arrival(r) < First-arrival(x) < Last-departure(x) < Last-departure(r) and therefore x is reachable from r in G.