CS140: Algorithms

Z Sweedyk
Lecture 14

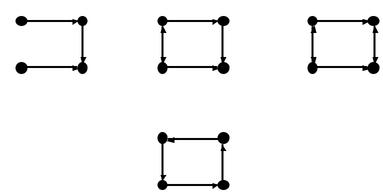
Graph Algorithms

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

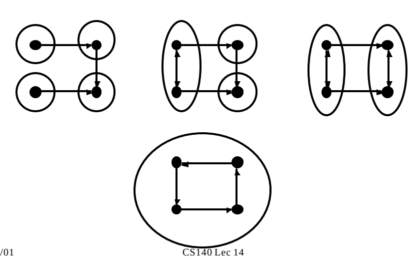
Digraph notions

- Vertex y is *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 *strongly connected* if x is reachable from y and y is reachable from x.

Strongly-connected vertices?



Strongly-connected vertices:



Digraph notions

- Vertex y is *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 *strongly connected* if x is reachable from y and y is reachable from x.
- Vertices form *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)
  While G has an unvisited
    vertex x:
      Depth-First(x)

Choose alphabetically

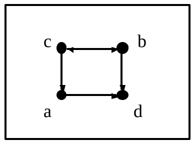
Depth-First(**x**)

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)

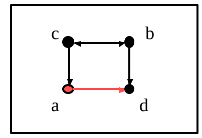


Call Stack:
DFS(G)

DFS(a)
Alphabetical priority

Visit a

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

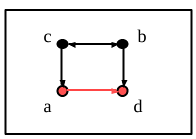


Call Stack:
DFS(a)
DFS(G)

DFS(d)
Alphabetical priority

Visit d

All out-edges checked so return



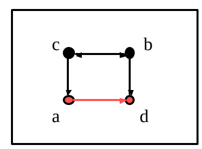
Call Stack:
DFS(d)
DFS(a)
DFS(G)

DFS(a)
Alphabetical priority

Visit a

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

All out-edges checked so return

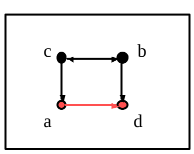


Call Stack:
DFS(a)
DFS(G)

DFS(G)
Alphabetical priority

a is unvisited so DFS(a)

b is unvisited so DFS(b)

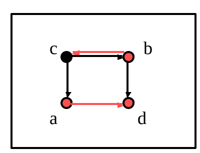


Call Stack:
DFS(G)

DFS(b)
Alphabetical priority

Visit b

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



Call Stack:
DFS(b)
DFS(G)

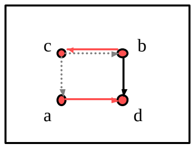
DFS(c)
Alphabetical priority

Visit c

Find edge <c,a> – no action

Find edge <c,b> – no action

All out-edges checked so return



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

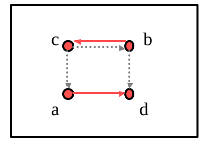
DFS(b)
Alphabetical priority

Visit b

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

Find edge <b,d> – no action

All out-edges checked so return



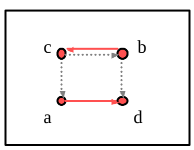
Call Stack:
DFS(b)
DFS(G)

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

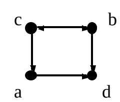
- DFS Forest
- Timestamps
- Reversal of G

DFS Forest

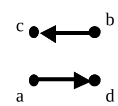
- The DFS Forest of G is the subgraph consisting of
  - Every vertex of G
  - Each edge traversed in DFS(G)

Different selection rules give different results

G:



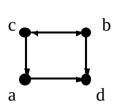
DFS tree of G:



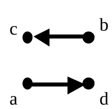
DFS Forest

- Selection rule: a,b,c,d

G:



DFS forest of G:



- What about b,d,a,c?
- What about d,c,b,a?
- What about b,c,d,a?

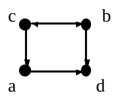
WARNING

- DFS Forests are sometimes

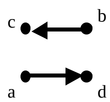
What is the connection?

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

G:

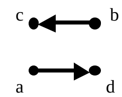


A DFS forest of G:



What can we say?

DFS forest of G



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

What can we say?

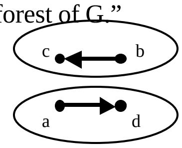
- If x and y are in the same strongly connected component of G then ________________________________.
- If x and y are in different strongly connected components of G then ________________________________.

What can we say?

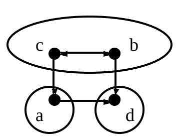
- If x and y are in the same tree in a DFS forest of G then ________________________________.
- If x and y are in different different trees in a DFS forest of G then ________________________________.

DFS Forest

- Strongly-connected in G is a refinements of the relation “in the same tree of a DFS forest of G.”



DFS forest of G



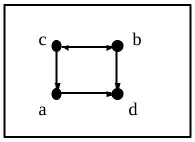
Strongly connected components of G

Three ideas

- DFS Forest of G
- **Timestamps**
- Reversal

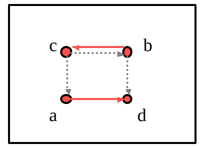
DFS(G)
Alphabetical order

Record first-arrival and last-departure times.



	First-arrival	Last-Departure
a		
b		
c		
d		

DFS(G)
Alphabetical order

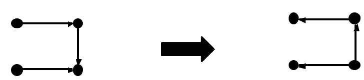


	First-arrival	Last-Departure
a	1	4
b	5	8
c	6	7
d	2	3

Three ideas

- DFS Forest
- Timestamps
- **Reversal of G**

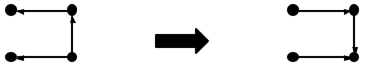
G^R: Reverse the edges of G



G

G^R

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



G^R

$(G^R)^R=G$

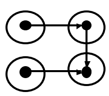
Reachability

G

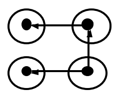
G^R

X is reachable from Y in G $\Leftrightarrow$ Y is reachable from X in G^T

Reachability



G



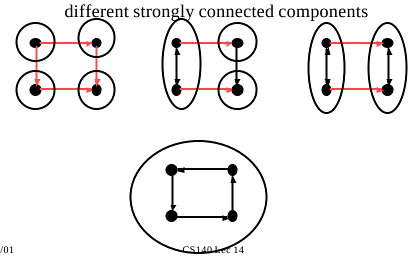
G^R

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

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

SCC

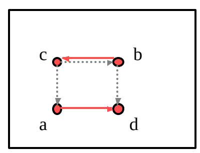
Reversal only affects edges between different strongly connected components



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

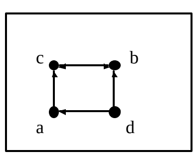
DFS(G)



	First-arrival	Last-Departure
a	1	4
b	5	8
c	6	7
d	2	3

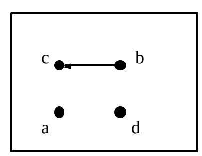
DFS(G^R)

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.

WLOG assume that r is the root the tree containing x and y in the DFS forest of G^R.

Hence r is reachable from x and from y in G.

We will show that x are reachable from r in G. (The same argument holds for y).

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 $\Rightarrow\Leftarrow$
- So First-arrival(r)<First-arrival(x) < Last-departure(x) < Last-departure(r) and therefore x is reachable from r in G.