Graph Algorithms

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

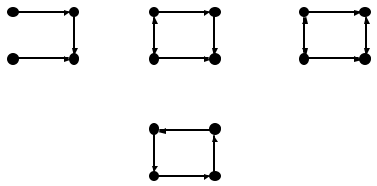
Digraph notions

- 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.

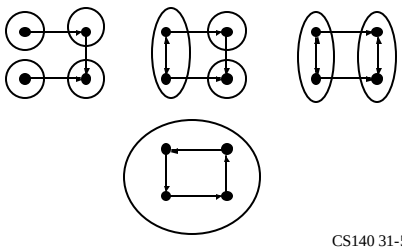
Digraph notions cont.

- Vertex y is *reachable* from x if there is a directed path in G from x to y.
- Vertices x and y are *strongly connected* if x is reachable from y and y is reachable from x.

Strongly-connected vertices?



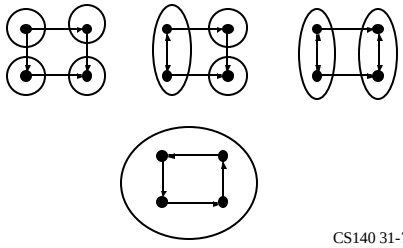
Strongly-connected vertices:



Digraph notions cont.

- Strongly-connected is an equivalence relation.
- The vertices are partitioned into equivalence classes called strongly connected components.

Strongly connected components SCC



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)

WARNING

- Order matters
- We'll use alphabetical priority

DFS(G)

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

Choose alphabetically

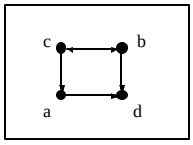
Depth-First(**x**)

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

Choose alphabetically

DFS(G)
Alphabetical priority

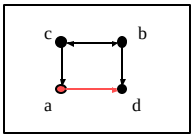
a is unvisited so DFS(a)



DFS(a)
Alphabetical priority

Visit a

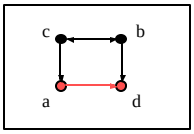
Find <a,d> edge and call DFS(d)



DFS(d)
Alphabetical priority

Visit d

All out-edges checked so return

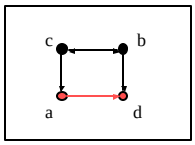


DFS(a)
Alphabetical priority

Visit a

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

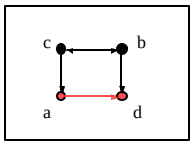
All out-edges checked so return



DFS(G)
Alphabetical priority

a is unvisited so DFS(a)

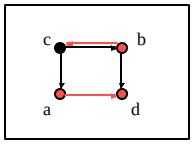
b is unvisited so DFS(b)



DFS(b)
Alphabetical priority

Visit b

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



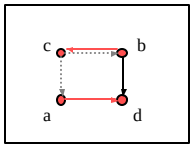
DFS(c)
Alphabetical priority

Visit c

Find edge <a,c> – no action

Find edge <b,c> – no action

All out-edges checked so return



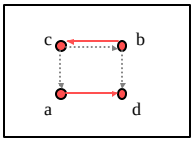
DFS(b)
Alphabetical priority

Visit b

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

Find edge <b,d> – no action

All out-edges checked so return

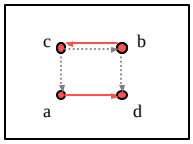


DFS(G)
Alphabetical priority

a is unvisited so DFS(a)

b is unvisited so DFS(b)

All nodes checked so return



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 otherwise 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

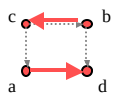
- **DFS Forest of G**
- Timestamps
- Reversal of G

DFS Forest of G

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

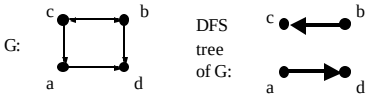
"Traversed"

- Walked across to visit



DFS Forest

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

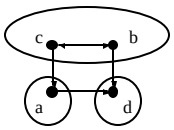


WARNING

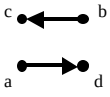
- DFS Forests are sometimes

DFS Forest

- If x is reachable from y … can't say much



Strongly connected components of G

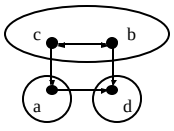


DFS forest of G

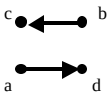
DFS Forest

- 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.
- Why?

Converse is not true



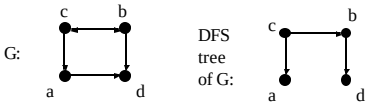
Strongly connected components of G



DFS forest of G

The problem

- The DFS forest of G depends on the order in which we consider vertices and edges.
- Example: ordering c,b,a,d or c,b,d,a

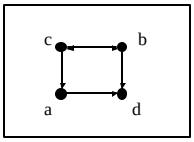


Three ideas

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

DFS(G)

a is unvisited so DFS(a)

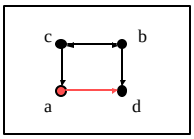


	First-arrival	Last-Departure
a		
b		
c		
d		

DFS(a)

Visit a

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

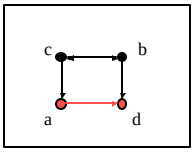


	First-arrival	Last-Departure
a	1	
b		
c		
d		

DFS(d)

Visit d

All out-edges checked so return



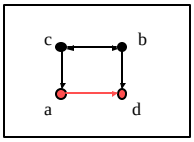
	First-arrival	Last-Departure
a	1	
b		
c		
d	2	3

DFS(a)

Visit a

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

All out-edges checked so return

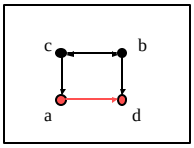


	First-arrival	Last-Departure
a	1	4
b		
c		
d	2	3

DFS(G)

a unvisited so Call DFS(a)

b unvisited so Call DFS(b)

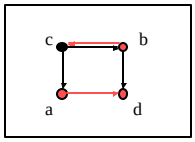


	First-arrival	Last-Departure
a	1	4
b		
c		
d	2	3

DFS(b)

Visit b

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



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

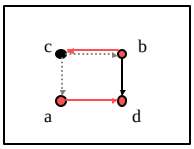
DFS(c)

Visit c

Find edge <a,c> – no action

Find edge <b,c> – no action

All out-edges checked so return



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

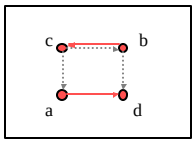
DFS(b)

Visit b

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

Find edge <b,d> – no action

All out-edges checked so return



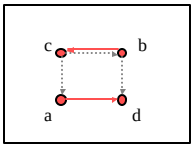
	First-arrival	Last-Departure
a	1	4
b	5	8
c	6	7
d	2	3

DFS(G)

Call DFS(a)

Call DFS(b)

All nodes visited so return



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

Three ideas

- DFS Forest of G
- Timestamp
- **Reversal of G**

G^R: Reverse the edges of G

G G^R

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

Reachability

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

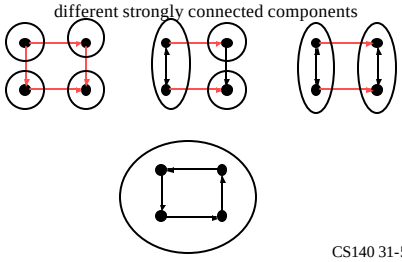
Reachability

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

So the SCC of G and G^R are the same!

SCC

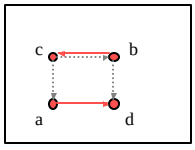
Reversal only affects edges between different strongly connected components



SCC

- DFS(G) with timestamp
- 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

Last-Departure time decreasing order: b,c,a,d

Claim

- If x is reachable from y then at least one of the following holds:
  A. Last-Departure(y)>Last-Departure(x)
  B. x and y are in the same SCC

Proof

When DFS(y) is called:

1. We haven't yet visited X – then x is visited during DFS(y)
2. We have visited X
   a. We haven't departed X
   b. We have departed X

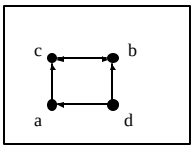
Proof

When DFS(y) is called:

1. We haven't yet visited X → then x is visited during DFS(y) ➡ A holds
2. We have visited X
   - We haven't departed X ➡ B holds
   - We have departed X ➡ A holds

DFS(G^R)

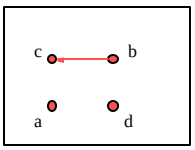
Order: b,c,a,d



DFS(G^R)

Order: b,c,a,d

Each DFS tree corresponds to a connected component.



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

- 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.
- The SCC of G are identical to those of G^R.

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 2

- Let x and y be in the same tree constructed in the DFS forest of G^R. WLOG assume x is the root.
- Since y is reachable from x in G^R, x is reachable from y in G. By a previous claim one of the following conditions holds:
  - Last-Departure(y)>Last-Departure(x) in DFS(G), or
  - x and y are in the same SCC in G

Claim 2

- Let x and y be in the same tree constructed in the DFS forest of G^R. WLOG assume x is the root.
- Since y is reachable from x in G^R, x is reachable from y in G. By a previous claim one of the following conditions holds
  - Last-Departure(y)>Last-Departure(x) in DFS(G)
    ➡ Contradiction
  - x and y are in the same SCC in G

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.

Graph Algorithms

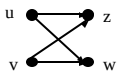
- Graph traversal
- **Topological sort**
- Strongly connected components
- Single-source shortest path

Topological Sort

- Input: Acyclic digraph
- Output: List of the vertices of G ordered to satisfy the following: If there is a path from x to y in G then x precedes y in the list.

Example

- Input:



- Output: u, v, z, w

Algorithm

- DFS(G) with timestamps
- Output vertices by decreasing last-departure time.