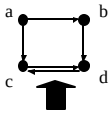
Graph Algorithms

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

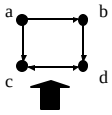
Graph and Digraph Traversal

- Breadth-first
- Depth-first

Digraph Notation

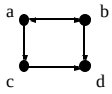


This is hard to draw.



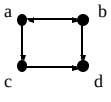
So we'll use this to mean <c,d> and <d,c> are both edges of the digraph.

Digraph Representation Adjacency Matrix



	a	b	c	d
a	0	1	1	0
b	1	0	0	1
c	0	0	0	1
d	0	0	0	0

Digraph Representation Adjacency Matrix

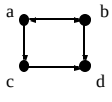


in-edges

out-edges

	a	b	c	d
a	0	1	1	0
b	1	0	0	1
c	0	0	0	1
d	0	0	0	0

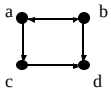
Digraph Representation Adjacency List



Vertex	Out-edges	In-edges (optional)
a	b→c	
b	a→d	a
c	d	a
d		b→c

Breadth-First(**c**)

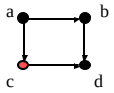
- For i=0,1,…
  Visit each unvisited vertex that is distance i from **c**



Breadth-First(**c**)

- For i=0,1,…
  Visit each unvisited vertex that is distance i from **c**

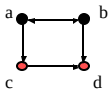
i=0



Breadth-First(**c**)

- For i=0,1,…
  Visit each unvisited vertex that is distance i from **c**

i=0,1

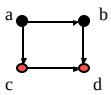


Digraph notions

- Vertex y is *reachable* from x if there is a directed path in G from x to y.
- Note that x is reachable from x by a directed path of length 0.

Breadth-First(x)

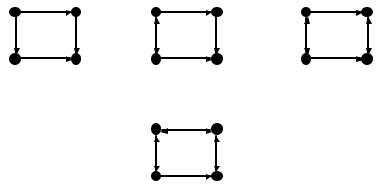
- Breadth-First(x) visits all vertices *reachable* from x.



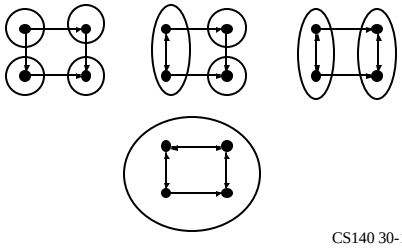
Digraph notions cont.

- 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.

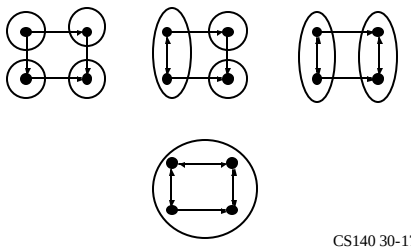
- Strongly-connected is an equivalence relation:
  - x is strongly connect to x.
  - If x is strongly connected to y then y is strongly connected to x.
  - If x is strongly connected to y and y is strongly connected to z then x is strongly connected to z.
- The vertices are partitioned into equivalence classes.

Strongly Connected Component

A strongly connected components of G is a *maximal* subgraph of G in which each pair of vertices is strongly connected.

The strongly connected components are exactly the equivalence classes under the strongly connected relation

Strongly-connected components:



BFS Application

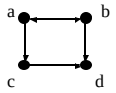
- Identify the strongly connected components of a digraph G.

DFS Application

- Identify the strongly connected components of a digraph G.

Depth-First(**c**)

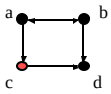
Depth-First(x)
Mark x visited
For each edge <x,y>
If y is unvisited then
DFS(y)



Depth-First(**c**)

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

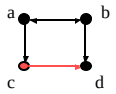
x = c



Depth-First(**c**)

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

x = c y=d

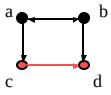


Depth-First(**d**)

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

x=d

x = c y=d

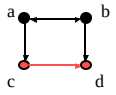


Depth-First(**d**)

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

x=d

x = c y=d

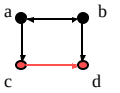


Depth-First(**d**)

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

x=d

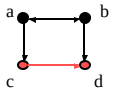
x = c y=d



Depth-First(**c**)

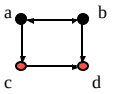
Depth-First(x)
- **Mark x visited**
- **For each edge <x,y>**
  - **If y is unvisited then**
    - **DFS(y)**

x = c



Depth-First(**c**)

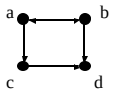
Depth-First(x)
- **Mark x visited**
- **For each edge <x,y>**
  - **If y is unvisited then**
    - **DFS(y)**



DFS(G)

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

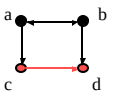
x=c



DFS(G)

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

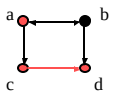
x=a



Depth-First(**a**)

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

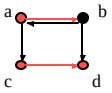
x = a



Depth-First(**a**)

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

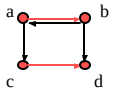
x = a y=b



Depth-First(**b**)

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

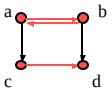
x=b
x = a y=b



Depth-First(**b**)

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

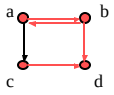
x=b y=a
x = a y=b



Depth-First(**b**)

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

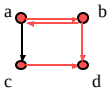
x=b y=d
x = a y=b



Depth-First(**b**)

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

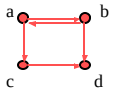
x = a y=b



Depth-First(**a**)

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

x = a y=c

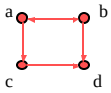


DFS(G)

DFS(G)

While G has an unvisited vertex x:

Depth-First(x)



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.