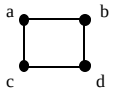
Graph Algorithms

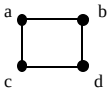
- Graph traversal
- Topological sort
- Strongly connected components
- Single-source shortest path

Graph Representation Adjacency Matrix



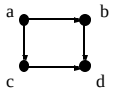
	a	b	c	d
a	0	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

Graph Representation Adjacency List



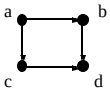
Vertex	Adj. List
a	b→c
b	a→d
c	a→d
d	b→c

Digraph Representation Adjacency Matrix



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

Digraph Representation Adjacency Matrix

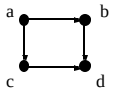


in-edges

out-edges

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

Digraph Representation Adjacency List



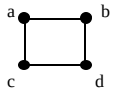
Vertex	Out-edges	In-edges (optional)
a	b→c	
b	d	a
c	d	a
d		b→c

Graph Traversal

- Breadth-first
- Depth-first

Breadth-First(**a**)

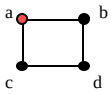
- For i=0,1,…
  Visit each unvisited vertex that is i edges away from **a**



Breadth-First(**a**)

- For i=0,1,…
  Visit each unvisited vertex that is i edges away from **a**

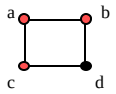
i=0



Breadth-First(**a**)

- For i=0,1,…
  Visit each unvisited vertex that is i edges away from **a**

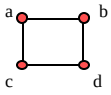
i=0,1



Breadth-First(**a**)

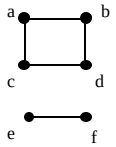
- For i=0,1,…
  Visit each unvisited vertex that is i edges away from **a**

i=0,1,2



Breadth-First Search(G)

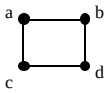
- While there is an unvisited vertex x
  - Breadth-First(x)



Breadth-First(**a**) Implementation

Visit a.

Put a in Q.

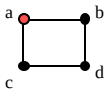


Breadth-First(**a**) Implementation

Visit a.

Put a in Q.

Q = a



Breadth-First(**a**) Implementation

LOOP:

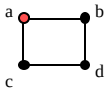
➡ curr = de-head of Q

For each unvisited neighbor of curr

Visit and add to Q

Q =

curr=a



Breadth-First(**a**) Implementation

LOOP:

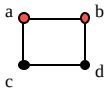
curr = de-head of Q

➡ For each unvisited neighbor of curr

Visit and add to Q

Q = b

curr=a



Breadth-First(**a**) Implementation

LOOP:

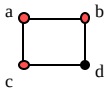
curr = de-head of Q

➡ For each unvisited neighbor of curr

Visit and add to Q

Q = b,c

curr=a



Breadth-First(**a**) Implementation

LOOP:

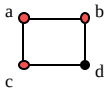
➡ curr = de-head of Q

For each unvisited neighbor of curr

Visit and add to Q

Q = c

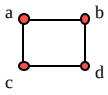
curr=b



Breadth-First(**a**) Implementation

LOOP:
- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q

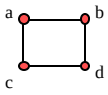
Q = c,d
curr=b



Breadth-First(**a**) Implementation

LOOP:
- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q

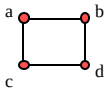
Q = d
curr=c



Breadth-First(**a**) Implementation

LOOP:
- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q

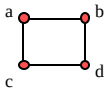
Q =
curr=d



Breadth-First(**a**) Implementation

LOOP:
- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q
- Until Q is empty

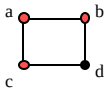
Q =
curr=d



Breadth-First(**a**) Implementation More Details

- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q

Q = c
curr=b

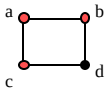


How is G represented?
- Adjacency matrix
- Adjacency list

Breadth-First(**a**) Implementation More Details

- curr = de-head of Q
- For each unvisited neighbor of curr
  - Visit and add to Q

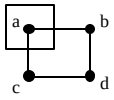
Q = c
curr=b



How is unvisited/visited represented?
- Bit vector

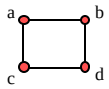
Connected(G)

- **Choose vertex x of G**
- **DFS(G,x)**
- If every vertex visited
  - Return YES
- Else return No



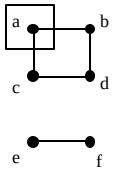
Connected(G)

- Choose vertex x of G
- DFS(G,x)
- **If every vertex visited**
  - **Return YES**
- Else return No



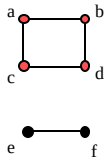
Connected(G)

- **Choose vertex x of G**
- **DFS(G,x)**
- If every vertex visited
  - Return YES
- Else return No



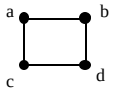
Connected(G)

- Choose vertex x of G
- Breadth-first(x)
- If every vertex visited
  - Return YES
- **Else return No**



Depth-First(**a**)

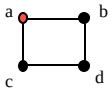
Depth-First(x)
  Visit x
  While x has an unvisited
    neighbor y: Depth-First(y)



Depth-First(**a**) Implementation

Depth-First(x)
  Visit x
  While x has an unvisited
    neighbor y: Depth-First(y)

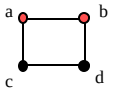
a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

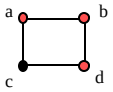
b
a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

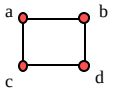
d
b
a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

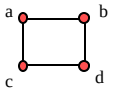
c
d
b
a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

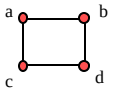
d
b
a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

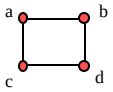
b
a



Depth-First(**a**) Implementation

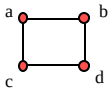
```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```

a



Depth-First(**a**) Implementation

```
Depth-First(x)
    Visit x
    While x has an unvisited
        neighbor y:  Depth-First(y)
```



CS140 28-37