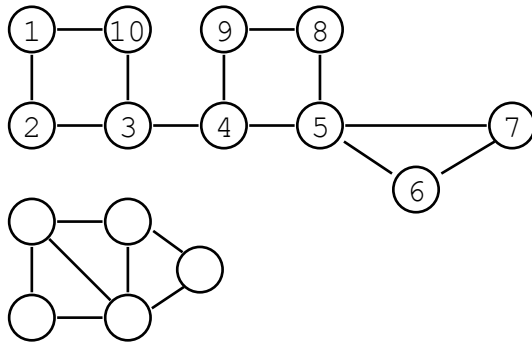


**Biconnectivity**

Let  $G = (N, E)$  be a connected

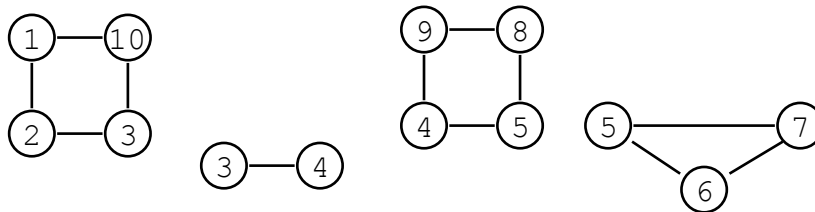
A node  $a \in N$  is an articulation point if there are  $v$  and  $w$  different from  $a$  such that every path from



**Biconnected Component**

Maximal subgraph that is still connected after

Express in terms of edges

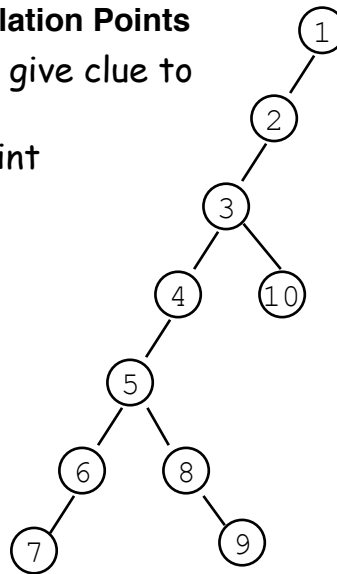


Articulation point must lie in two components

### DFS and Articulation Points

Tree, back edges in DFS give clue to articulation points

If  $a$  is an articulation point that splits  $v$  and  $w$ , then



### DFS and Articulation Points 2

Lemma:

$G = (N, E)$  is

$S = (N, T)$  is

Node  $a$  is an articulation point if and only if

1.  $a$  is the root of  $S$  and

or

2.  $a$  is not the root and  $a$  has some child  $c$  in  $S$  where there is no

## Low Numbers

For graph  $G = (N, E)$ , let

$T =$

$B =$

For simplicity, assume  $DFNum[v] =$

How high (lowest  $DFNum$ ) can we hop from a node on a back edge?

$LOW[v] =$

$\min(\{v\} \cup$

$\{w \mid \text{there is } \{x, w\} \text{ in } B \text{ where}$

## Calculating Low

If  $\{x, w\}$  in  $B$  and  $w < v$  then

Also,  $v$  is an articulation point if for some child  $c$  of  $v$ ,

$LOW[c]$

Calculate  $LOW$  during DFS

$LOW[v]$  is minimum of

1.

2.

3.

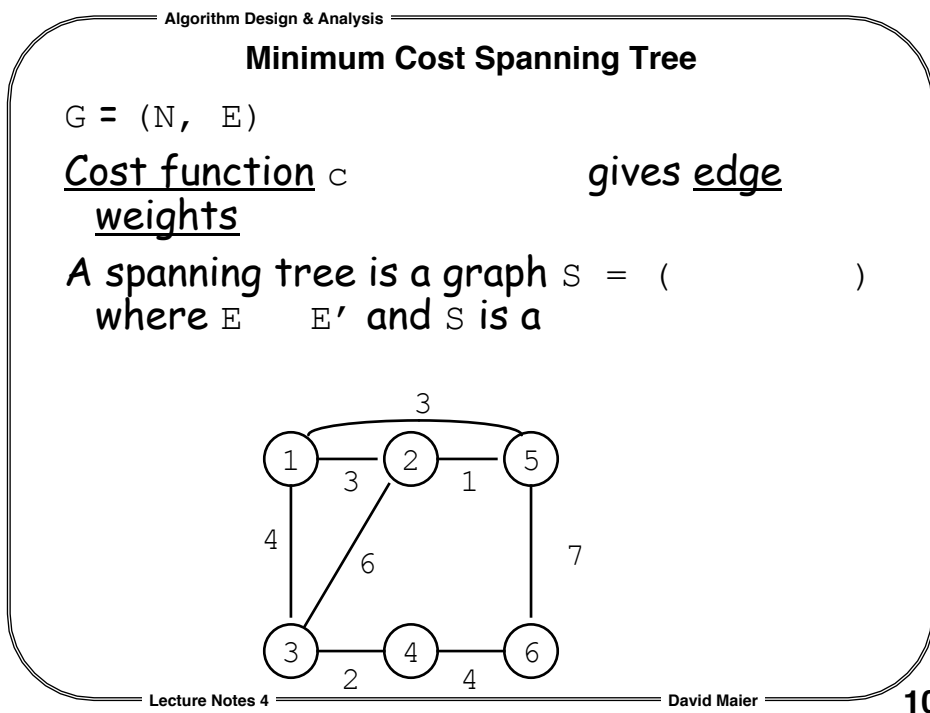
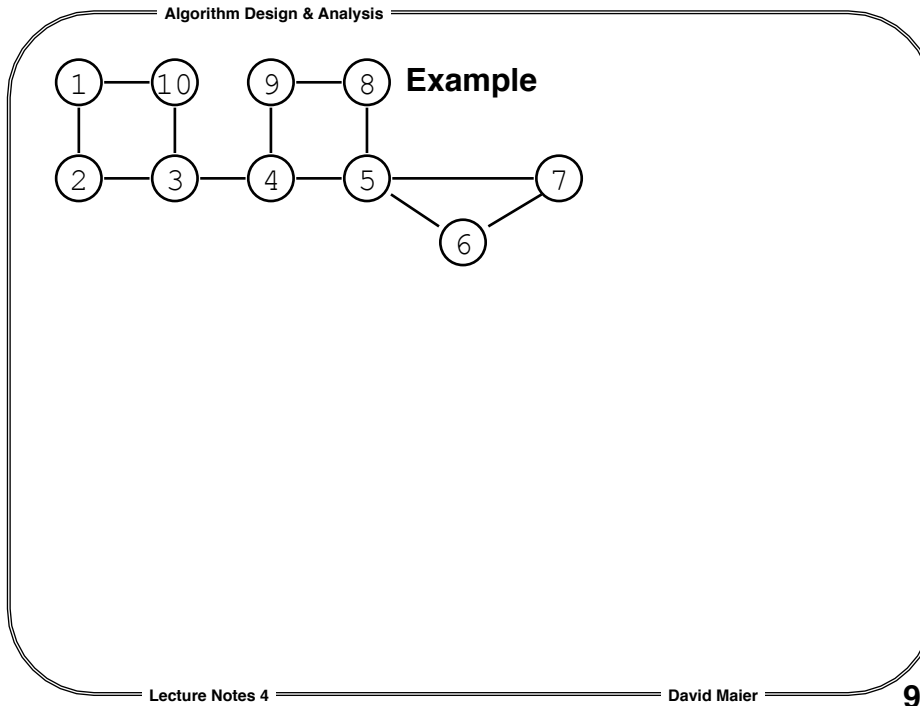
## Biconnected Components Algorithm

```
Count ←  
Parent[v] ←  
BI(v)  
  mark v  
  DFNum[v] ← Count, increment Count  
  Low[v] ← DFNum[v]
```

## Algorithm Cont.

```
for each edge {v,w} do  
  if w not marked then  
  
    if LOW[w] ≥ DFNum[v]  
      and v not root then  
  
        LOW[v] ←  
    else if PARENT[v] ≠ w then  
      LOW[v] ←
```

Need a check at end if root is an articulation point.



### Cost of a Spanning Tree

Cost of a spanning tree  $S = (N, E')$  is

$$\sum_{e \in E'}$$

Want minimum cost

Lemma: Let  $S = (N, E')$  be a spanning tree for  $G = (N, E)$ . Then

- a. for  $v_1, v_2$  in  $N$
- b. adding an edge from  $E - E'$  to  $S$

### Spanning Forest

A set of trees

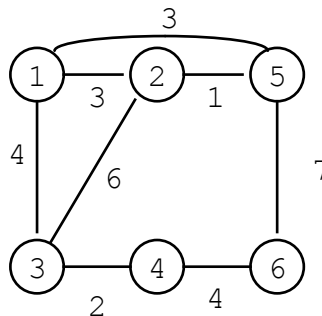
$$\{ (N_1, E_1), (N_2, E_2), \dots, (N_k, E_k) \}$$

Where

$$N =$$

$$\subseteq E$$

$$E_i \quad E_j$$



**Expanding a Spanning Forest**

Lemma: Suppose we have a spanning forest of  $k > 1$  trees. Let  $E' =$

Let  $e = \{v, w\}$  be such that  $v \in N_1, w \notin N_1$ .

Then there is a spanning tree for  $G$  that includes  $E' \cup$  and has minimal cost among trees that include  $E'$ .

*What kind of algorithm is this leading us towards?*

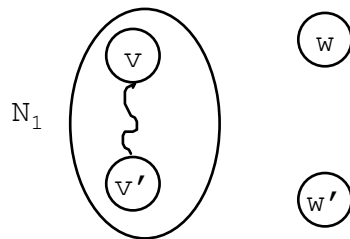
**Proof of Lemma**

Suppose not. Let

$$S'' = (N, E''), E' \subseteq E'',$$

have lower cost than any spanning tree

Adding  $e$  to  $S''$  causes



**Proof of Lemma 2**

$S''$  must have an edge  $e' = \{v', w'\}$   
with

We know  $c(e) < c(e')$

Let  $S = S''$

$S$  has no cycles

$S$  is still a tree

**Greedy Strategy**

Take least cost edge  $e_1$  in  $E$ . By last lemma,

[Consider an initial spanning forest

$\{(\{ \}, \emptyset), (\{ \}, \emptyset), \dots, (\{ \}, \emptyset)\}$ ]



## Example

Q	c	NS
{2, 5}	1	{1} {2} {3} {4} {5} {6}
{3, 4}	2	
{1, 2}	3	
{1, 5}	3	
{1, 3}	4	
{4, 6}	4	
{2, 3}	6	
{5, 6}	7	

## Complexity

Priority Queue: sort, make a list

Union/Find

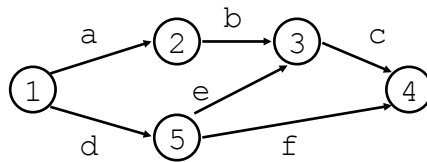
# Union's =

# Find's =

Can do a little better—  
keep a heap of edges by cost

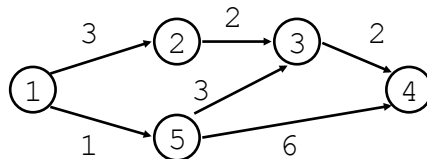
## Path Problems

Directed graphs with labels on edges



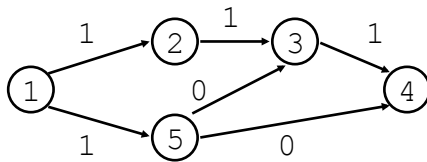
## Shortest Path

Labels could be non-negative integers  
Use for



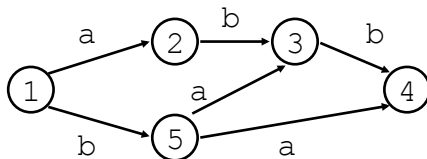
## Connectivity

Labels could be Boolean values  
Use for



## Finite Automata

Labels could be symbols from an  
alphabet  
(as in a finite automaton). Use for



## Path Algorithms

Can produce generic algorithms, just need definitions for

Can take advantage of identities

Two kinds of algorithms

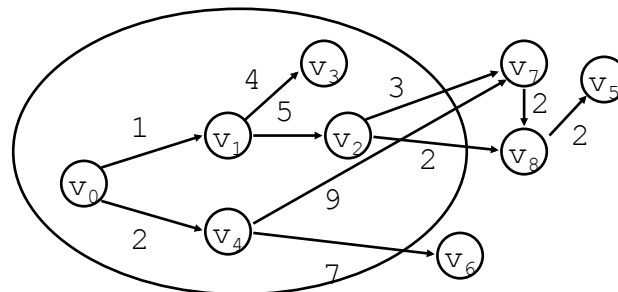
1.

2.

Can do 2. with

## Dijkstra's Algorithm

Single-source, shortest path in a digraph



S—

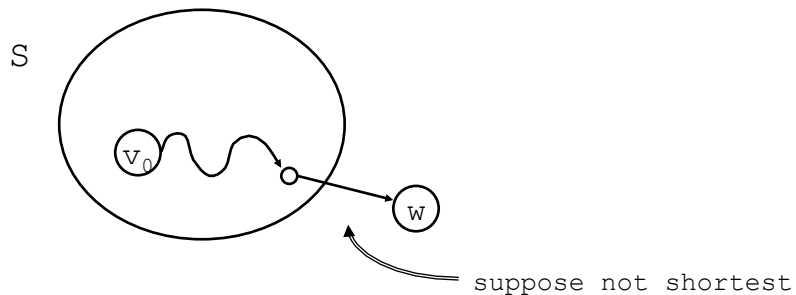
D—array of distances

## General Step

Add node not in  $S$  with minimum  $D$  value to  $S$ .

Update distances to nodes in  $N-S$

Is  $S$  still correct?



## Full Algorithm

$G = (N, E)$

$\text{lab}(v, w)$  - label  
represent with matrix

$S \leftarrow \{v_0\}$

$D[v_0] \leftarrow 0$

$D[v] \leftarrow \text{lab}(v_0, v)$  for rest

**while**  $S \neq N$  **do**

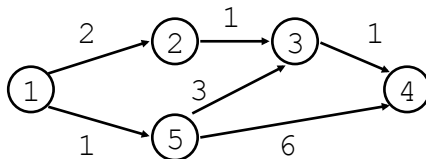
  choose  $w$

$S \leftarrow$

**for** all  $v$  in  $N-S$  **do**

$D[v] \leftarrow$

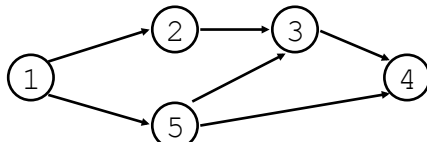
Example



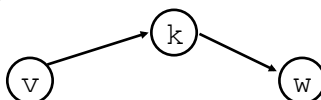
S	D[1]	D[2]	D[3]	D[4]	D[5]
{1}	0	2	$\infty$	$\infty$	1

Warshall's Algorithm

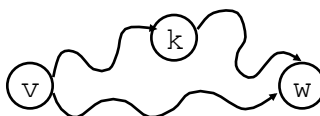
Transitive Closure



Basic op



Consider each  $k$  only once



## Algorithm

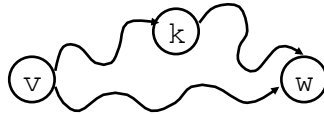
```
for k = 1 to n do
  for v = 1 to n do
    if a[v,k] then
      for w = 1 to n do
        if a[k,w] then
```

## Example

	1	2	3	4	5
1	1	1	0	0	1
2	0	1	1	0	0
3	0	0	1	1	0
4	0	0	0	1	0
5	0	0	1	1	1

### Floyd's Algorithm

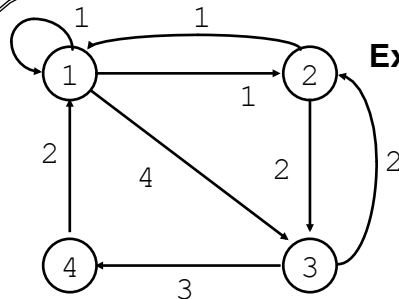
All-pairs shortest path



```

for k = 1 to n do
  for v = 1 to n do
    if d[v,k] ≠ ∞ then
      for w = 1 to n do
        d[v,w] ←
    
```

### Example



	1	2	3	4
1	0	1	4	∞
2	1	0	2	∞
3	∞	2	0	3
4	2	∞	∞	0

$d[i,j]^0$

**Example 2**

		1	2	3	4	$d[i, j]^0$
1	(	0	1	4	$\infty$	)
2		1	0	2	$\infty$	
3		$\infty$	2	0	3	
4	)	2	$\infty$	$\infty$	0	)

		1	2	3	4	$d[i, j]^1$
1	(					)
2						
3						
4	)					)

**Example 3**

		1	2	3	4	$d[i, j]^1$
1	(	0	1	4	$\infty$	)
2		1	0	2	$\infty$	
3		$\infty$	2	0	3	
4	)	2	3	6	0	)

		1	2	3	4	$d[i, j]^2$
1	(	0	1	3	$\infty$	)
2		1	0	2	$\infty$	
3		3	2	0	3	
4	)	2	3	5	0	)

**Example 3**

	1	2	3	4	$d[i, j]^2$
1	( 0	1	3	$\infty$	)
2	1	0	2	$\infty$	
3	3	2	0	3	
4	( 2	3	5	0	)

	1	2	3	4	$d[i, j]^3$
1	(				)
2					
3					
4	(				)