Lecture 24
Shortest Path Algorithms

EECS 281: Data Structures & Algorithms

Shortest path examples

- Highway system
  - Distance
  - Travel time
  - Number of stoplights
  - Krispy Kreme locations
- Network of airports
  - Travel time
  - Fares
  - Actual distance

Weighted path length

- Consider an edge-weighted graph $G = (V, E)$.
- Let $C(v_i, v_j)$ be the weight on the edge connecting v_i to v_j.
- A path in G is a non-empty sequence of vertices $P = \{v_1, v_2, v_3, \ldots, v_k\}$.
- The weighted path length is given by

$$\sum_{i=1}^{k-1} C(v_i, v_{i+1})$$

The general problem

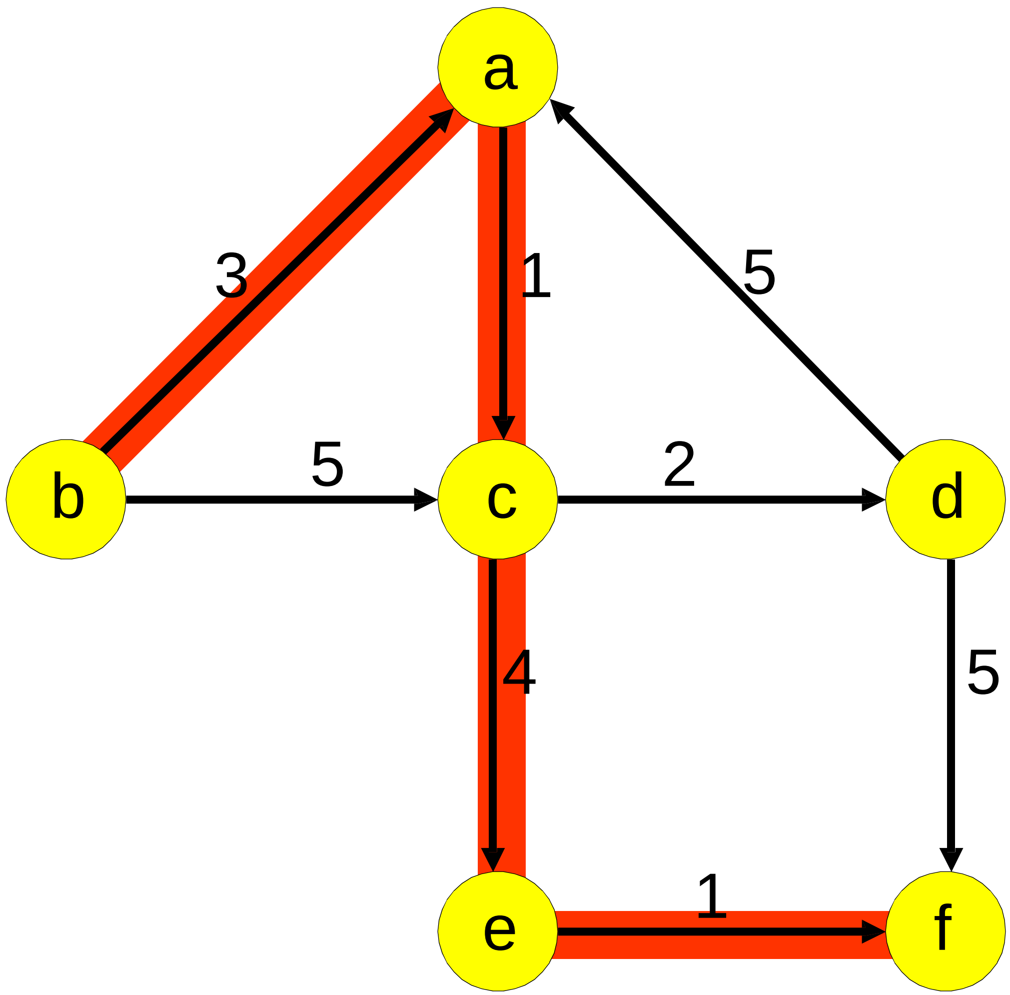
- Given an edge-weighted graph $G = (V, E)$ and two vertices, $v_s \in V$ and $v_d \in V$, find the path that starts at v_s and ends at v_d that has the smallest weighted path length

Single-source shortest path

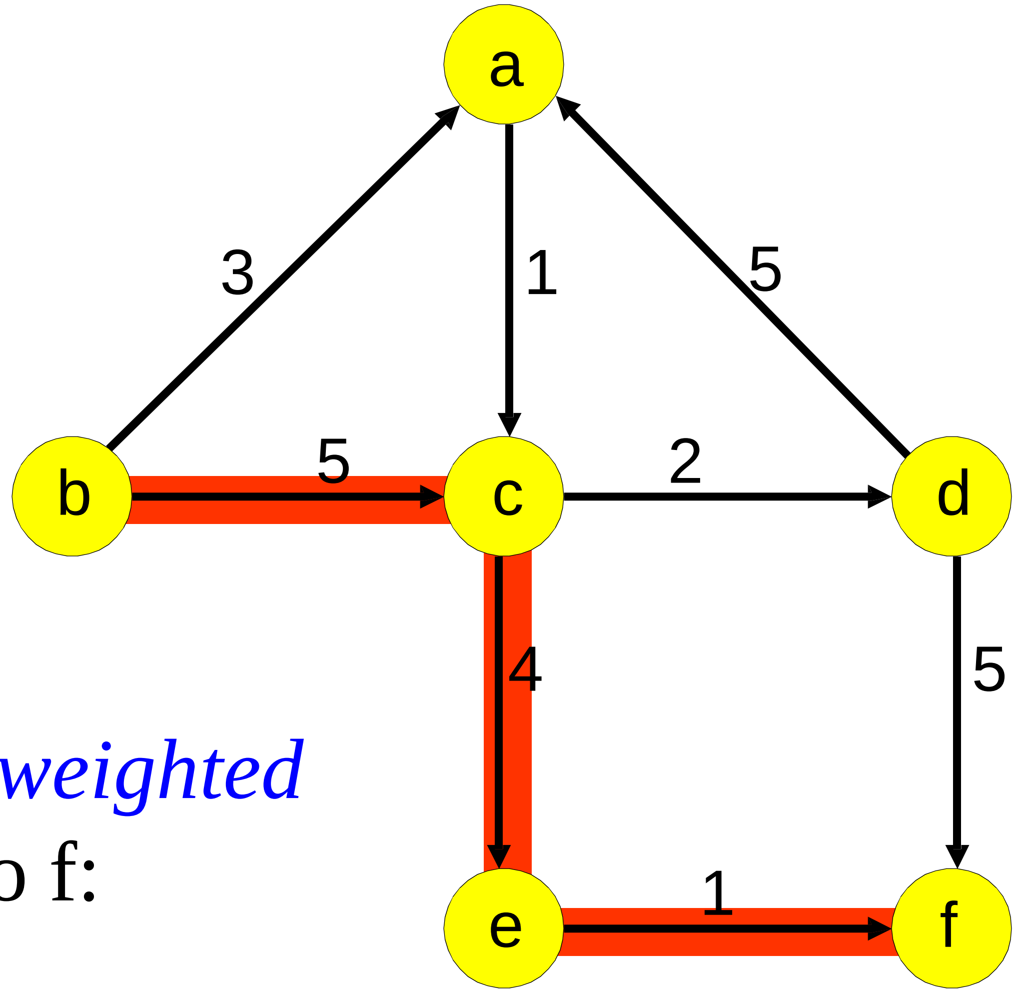
- Given an edge-weighted graph $G = (V, E)$ and a vertex, $v_s \in V$, find the shortest path from v_s to every other vertex in V

- To find the shortest path from v_s to v_d, we must find the shortest path from v_s to every vertex in G

The shortest weighted path from b to f: {b, a, c, e, f}

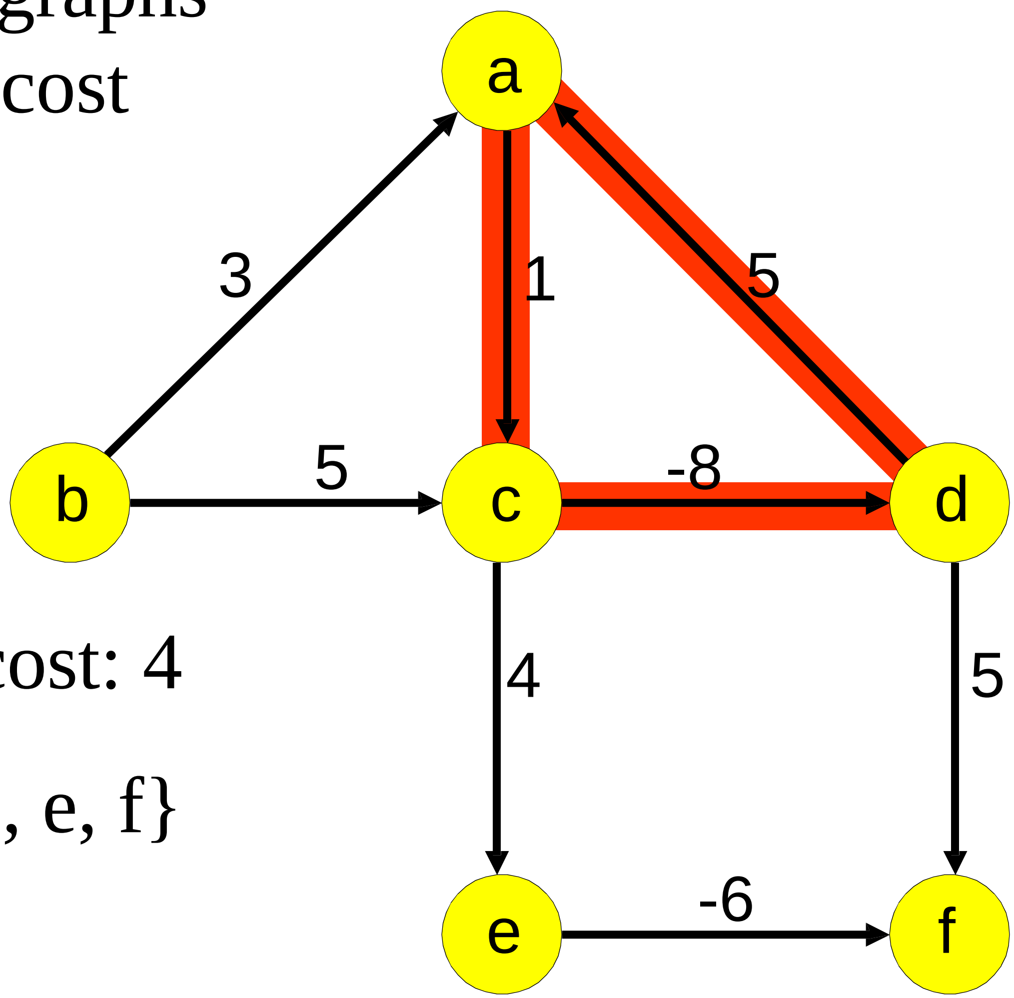


The shortest *weighted* path from b to f:
{b, a, c, e, f}



A shortest *unweighted* path from b to f:
{b, c, e, f}

Shortest path problem undefined for graphs with negative-cost cycles



{d, a, c, e, f} cost: 4

{d, a, c, d, a, c, e, f} cost: 2

{d, a, c, d, a, c, d, a, c, e, f} cost: 0

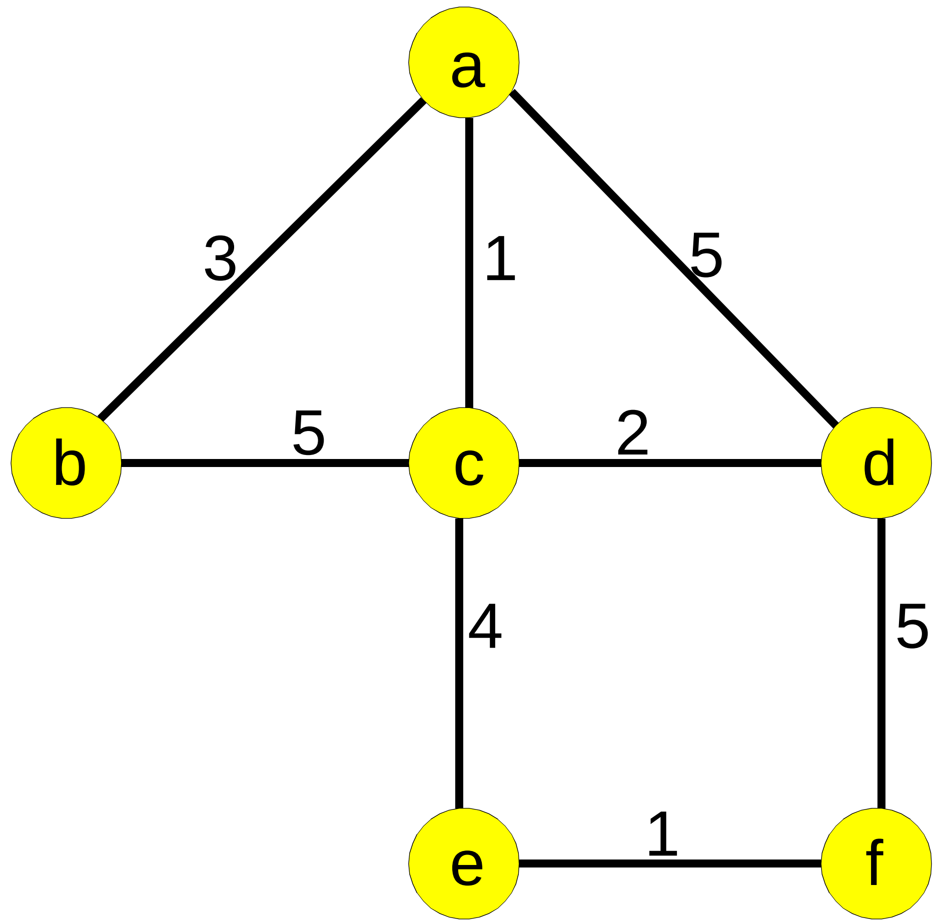
Dijkstra's Algorithm

- Greedy algorithm for solving shortest path problem
- Assume non-negative weights
- Find shortest path from v_s to every other vertex

Dijkstra's Algorithm

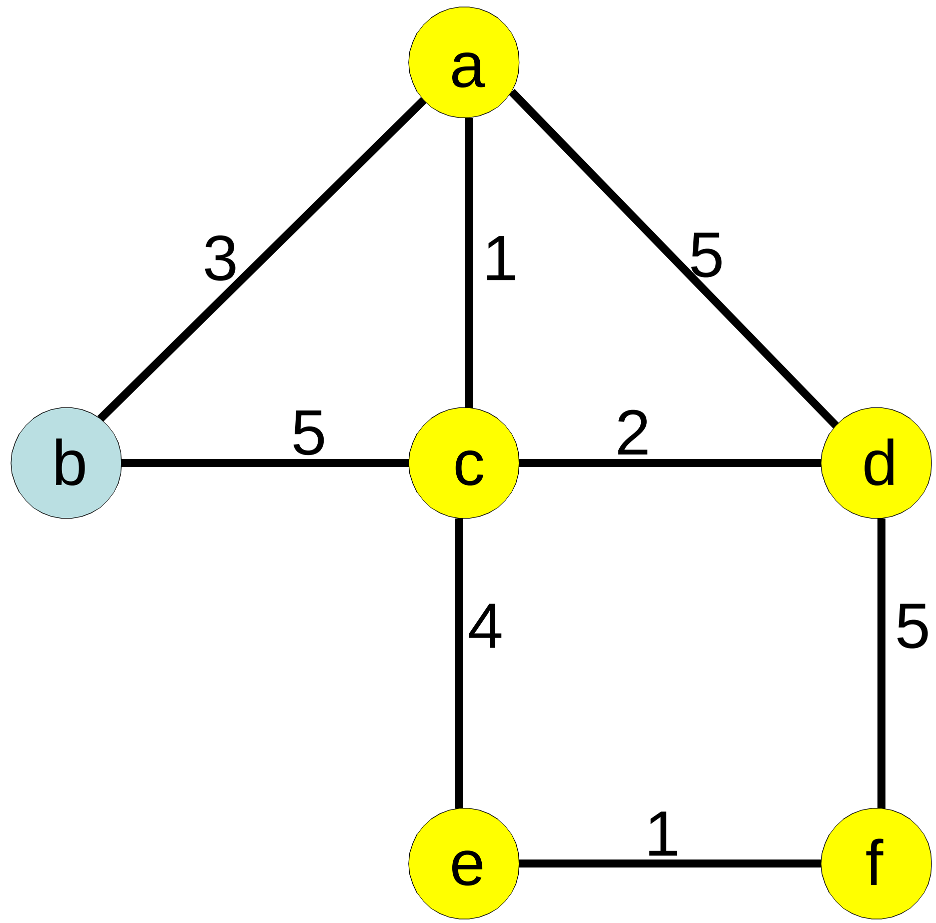
- For each vertex v, need to know:
  - k_v: Is the shortest path from v_s to v known? (initially false for all $v \in V$)
  - d_v: What is the length of the shortest path from v_s to v? (initially ∞ for all $v \in V$, except $v_s = 0$)
  - p_v: What vertex precedes (is parent of) v on the shortest path from v_s to v? (initially unknown for all $v \in V$)

v	k_v	d_v	p_v
a	***F***	∞	
b	***F***	*0*	
c	***F***	∞	
d	***F***	∞	
e	***F***	∞	
f	***F***	∞	

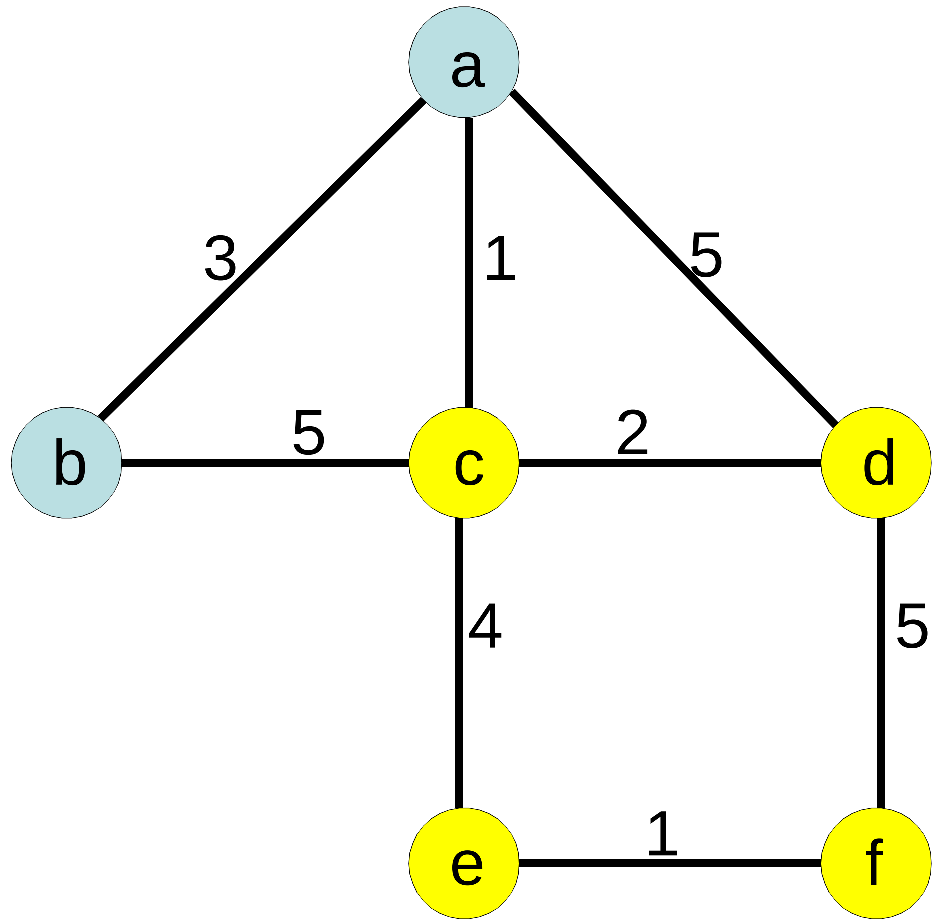


Find shortest paths to *b*

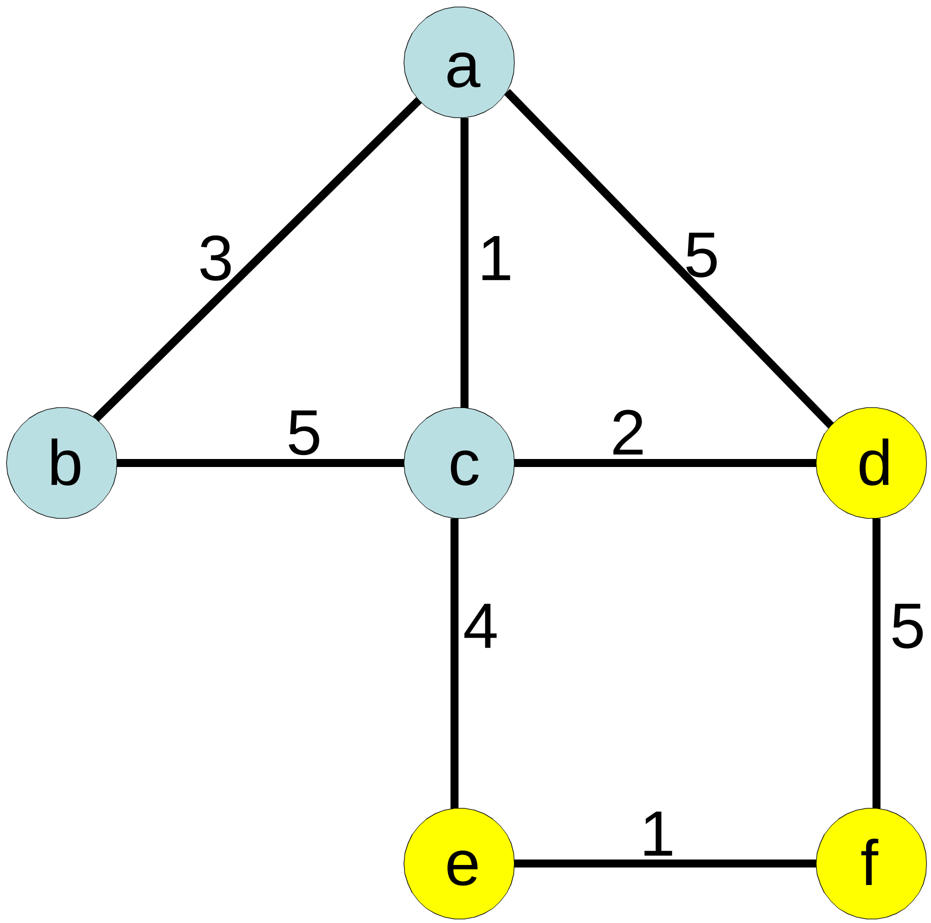
v	k_v	d_v	p_v
a	***F***	3	b
b	***T***	0	--
c	***F***	5	b
d	***F***	∞	
e	***F***	∞	
f	***F***	∞	



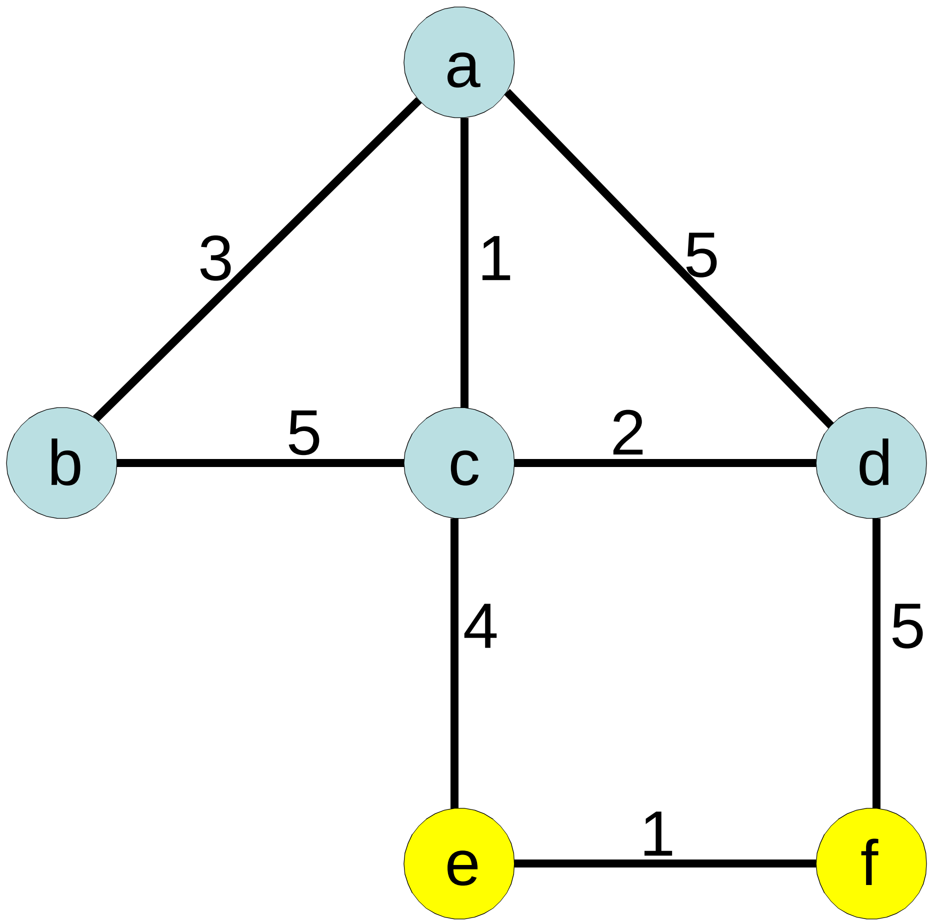
v	k_v	d_v	p_v
a	***T***	3	*b*
b	***T***	0	--
c	***F***	4	*a*
d	***F***	8	*a*
e	***F***	∞	
f	***F***	∞	



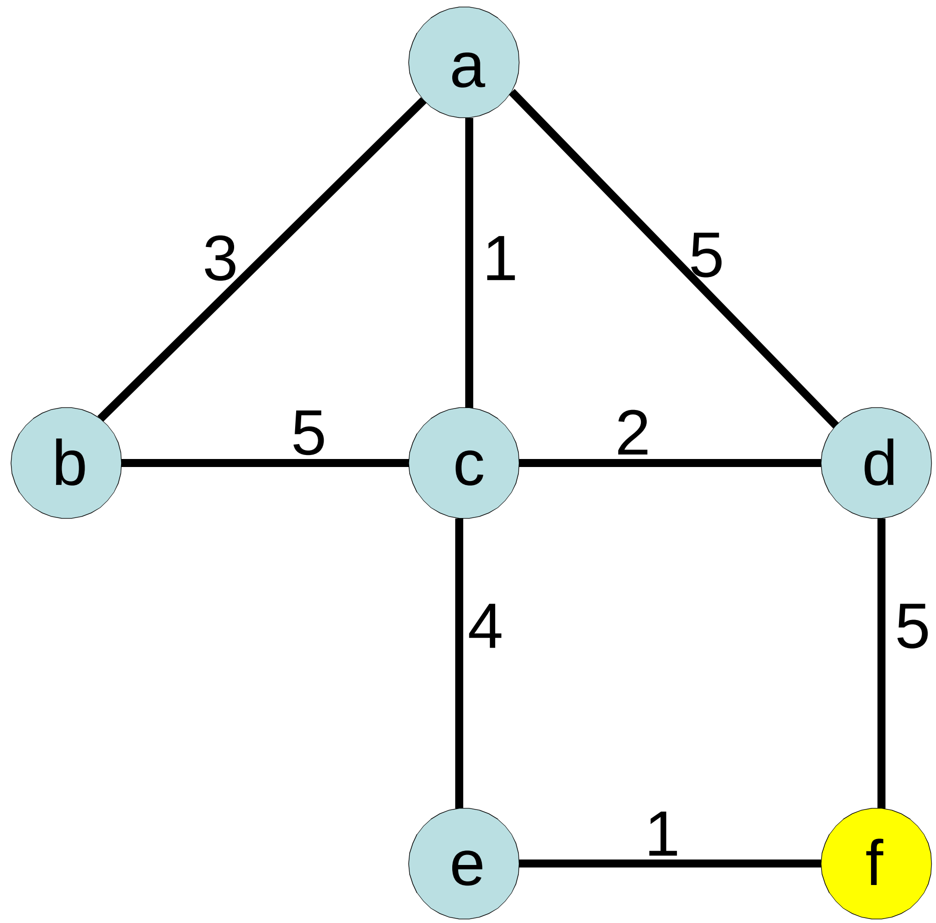
v	k_v	d_v	p_v
a	***T***	3	b
b	***T***	0	--
c	***T***	4	a
d	***F***	6	c
e	***F***	8	c
f	***F***	∞	



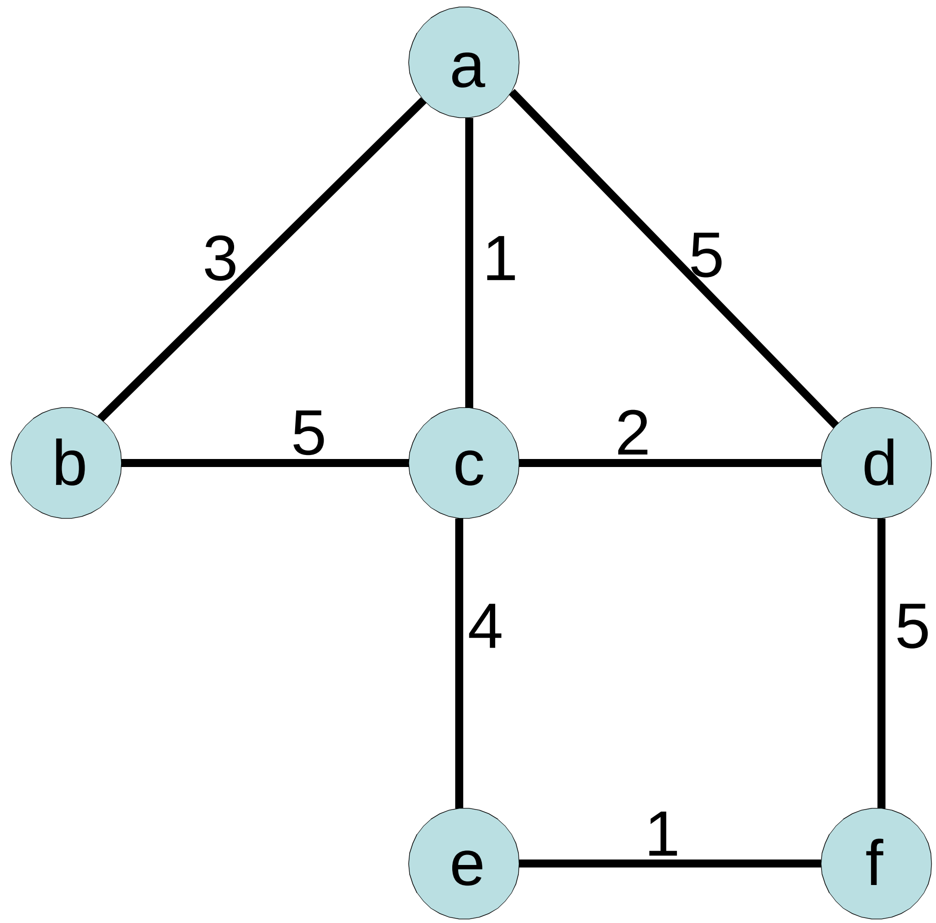
v	k_v	d_v	p_v
a	***T***	3	b
b	***T***	0	--
c	***T***	4	a
d	***T***	6	c
e	***F***	8	c
f	***F***	11	d



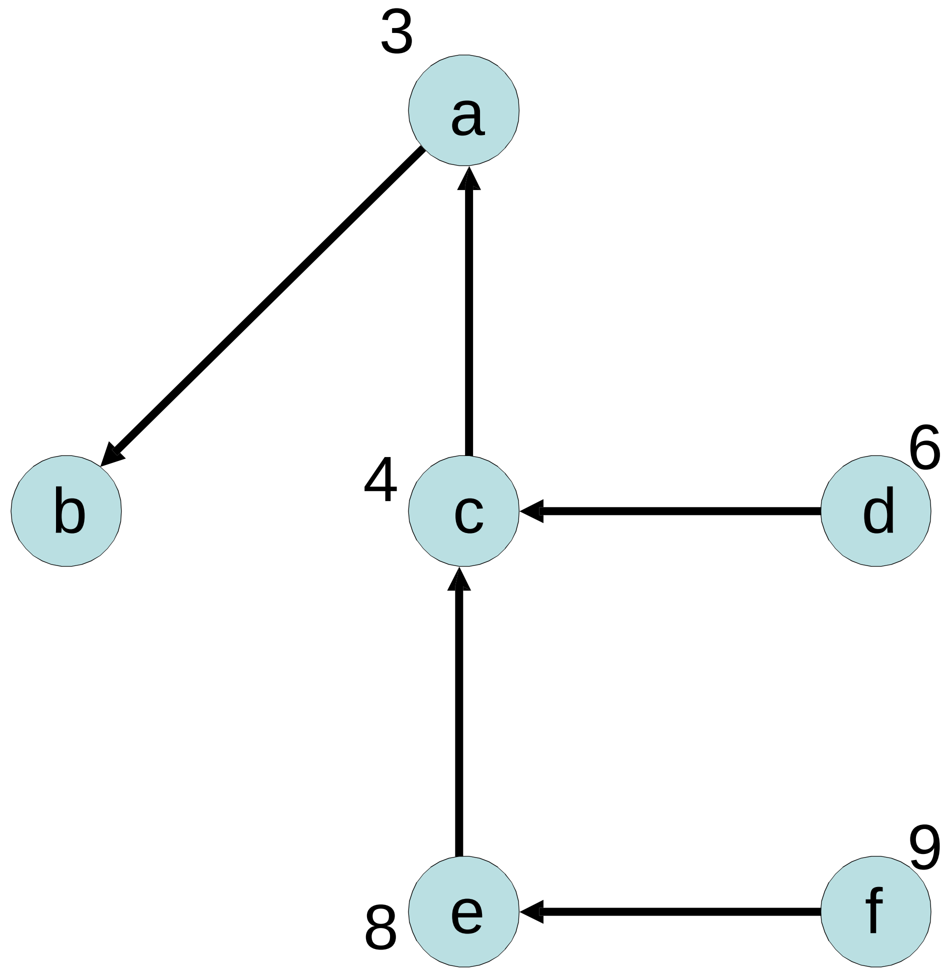
v	k_v	d_v	p_v
a	***T***	3	*b*
b	***T***	0	--
c	***T***	4	*a*
d	***T***	6	*c*
e	***T***	8	*c*
f	***F***	9	*e*



v	k_v	d_v	p_v
a	$\boldsymbol{T}$	3	b
b	$\boldsymbol{T}$	0	--
c	$\boldsymbol{T}$	4	a
d	$\boldsymbol{T}$	6	c
e	$\boldsymbol{T}$	8	c
f	$\boldsymbol{T}$	9	e



v	k_v	d_v	p_v
a	$\boldsymbol{T}$	3	b
b	$\boldsymbol{T}$	0	--
c	$\boldsymbol{T}$	4	a
d	$\boldsymbol{T}$	6	c
e	$\boldsymbol{T}$	8	c
f	$\boldsymbol{T}$	9	e



Dijkstra Complexity

- $O(V^2)$ for a simple nested loop implementation, a lot like Prim's
  - Intuition: for each vertex, find the min using linear search
- $O(E \log V)$ for sparse graphs, using heaps
  - E for considering every edge
  - $\log E = O(\log V^2) = O(\log V)$ for finding the shortest edge in heap
- CLRS 24.3 has a good explanation

Dijkstra's Algorithm

Algorithm Dijsktra(G, s_0)

//Initialize

n = |V| O( )

create_table(n) //stores k,d,p O( )

create_pq() //empty heap O( )

table[s_0].d = 0 O( )

insert_pq(0, s_0) O( )

Dijkstra’s Algorithm (cont.)

while (!pq.isempty)	O( )
v_0 = getMin() //heap top() & pop()	O( )
if (!table[v_0].k) //not known	O( )
table[v_0].k = true	O( )
for each $v_i \in$ Adj[v_0]	O( )
distance = table[v_0].d + weight(v_i, v_0)	O( )
if (distance < table[v_i].d)	O( )
table[v_i].d = distance	O( )
table[v_i].p = v_0	O( )
insert_pq(distance, v_i)	O( )

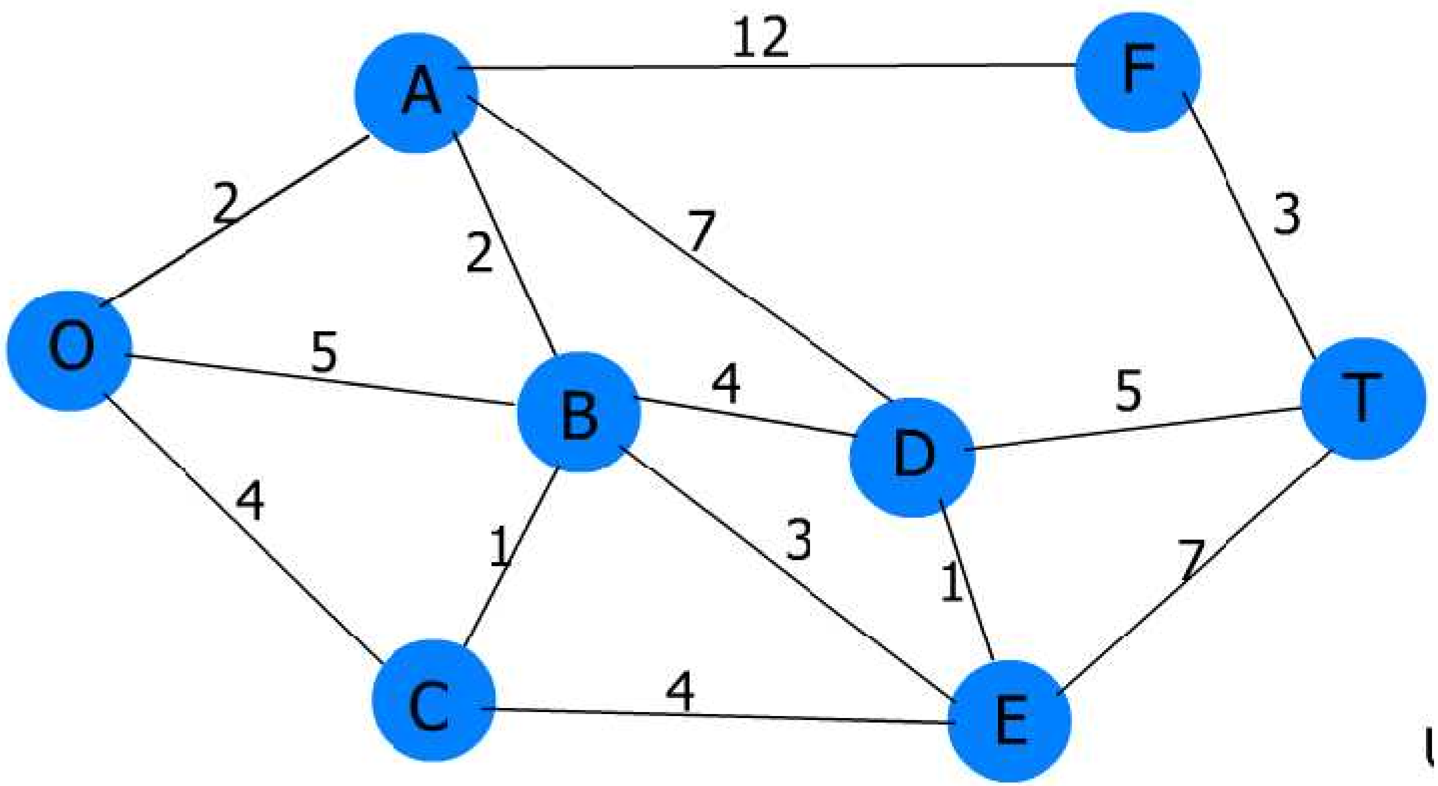
Dijkstra's Algorithm (cont.)

for each $v \in G(V,E)$
    //build vertex set in T
    $v \in T(V, E')$ O( )

for each $v \in G(V,E)$
    //build edge set in T
    $(v, table[v].p) \in T(V, E')$ O( )

Exercise

Find the shortest path from O to T



All-pairs shortest path problem

- Given an edge-weighted graph $G = (V, E)$, for each pair of vertices in V find the length of the shortest weighted path between the two vertices

Solution:

Run Dijkstra V times

Use Floyd's Algorithm (dense graphs)

Use Johnson's Algorithm (sparse graphs)

Solution 2: Floyd's Algorithm

- Floyd-Warshall Algorithm
- Dynamic programming method for solving all-pairs shortest path problem on a dense graph
- Uses an adjacency matrix
- $O(V^3)$ (best, worst, average)

Weighted path length

- Consider an edge-weighted graph $G = (V,E)$, where $C(v,w)$ is the weight on the edge (v,w).
- Vertices numbered from 1 to $|V|$ (i.e. $V = \{v_1, v_2, v_3, \ldots, v_{|V|}\}$)

Weighted path length

- Consider the set $V_k = \{v_1, v_2, v_3, \ldots, v_k\}$ for $0 \leq k \leq |V|$
- $P_k(i,j)$ is the shortest path from i to j that passes only through vertices in V_k if such a path exists
- $D_k(i,j)$ is the length of $P_k(i,j)$

$$D_k(i,j) = \begin{cases} |P_k(i,j)| & \text{if } P_k(i,j) \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

Suppose k = 0

- $V_0 = \emptyset$, so P_0 paths are the edges in G:

$$P_0(i,j) = \begin{cases} \{i,j\} & if\ (i,j) \in E \\ undefined & otherwise \end{cases}$$

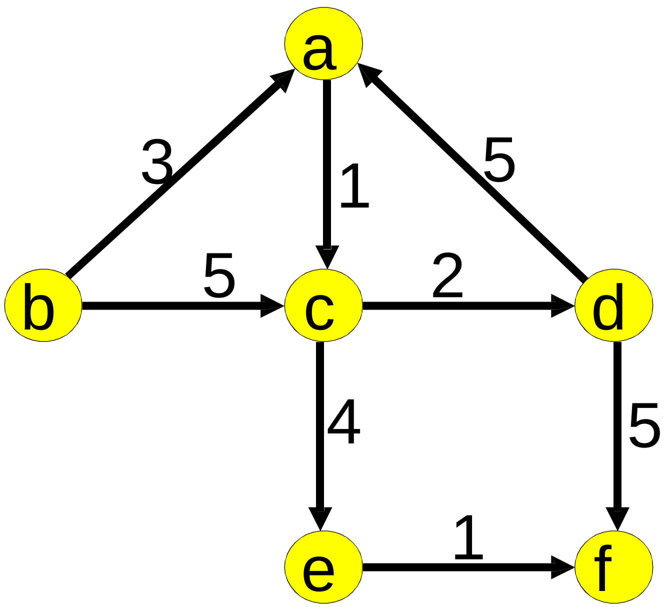
- Therefore D_0 path lengths are:

$$D_0(i,j) = \begin{cases} |C(i,j)| & if\ (i,j) \in E \\ \infty & otherwise \end{cases}$$

to

D_0	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	5	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	∞	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

from



Floyd's Algorithm

- Add vertices to V_k one at a time
- For each new vertex v_k, consider whether it improves each possible path
  - Compute $D_k(i,j)$ for each i,j in V
  - Minimum of:
    - $D_{k-1}(i,j)$
    - $D_{k-1}(i,k) + D_{k-1}(k,j)$

$$D_k(i,j) = min(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_1 = \{a\}$$

to

from

D_0	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	5	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	∞	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

D_1	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	**4**	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	**6**	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

$$D_k(i,j) = min(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_2 = \{a, b\}$$

to

D_1	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	4	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	6	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

from

D_2	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	4	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	6	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

$$D_k(i,j) = min(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_3 = \{a, b, c\}$$

to

D_2	a	b	c	d	e	f
a	∞	∞	1	∞	∞	∞
b	3	∞	4	∞	∞	∞
c	∞	∞	∞	2	4	∞
d	5	∞	6	∞	∞	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

from

D_3	a	b	c	d	e	f
a	∞	∞	1	**3**	**5**	∞
b	3	∞	4	**6**	**8**	∞
c	∞	∞	∞	2	4	∞
d	5	∞	6	∞	**10**	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

$$D_k(i,j) = min(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_4 = \{a, b, c, d\}$$

to

D_3	a	b	c	d	e	f
a	∞	∞	1	3	5	∞
b	3	∞	4	6	8	∞
c	∞	∞	∞	2	4	∞
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

from

D_4	a	b	c	d	e	f
a	∞	∞	1	3	5	**8**
b	3	∞	4	6	8	**11**
c	**7**	∞	∞	2	4	**7**
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

$$D_k(i,j) = min(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_5 = \{a, b, c, d, e\}$$

to

from

D_4	a	b	c	d	e	f
a	∞	∞	1	3	5	8
b	3	∞	4	6	8	11
c	7	∞	∞	2	4	7
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

D_5	a	b	c	d	e	f
a	∞	∞	1	3	5	**6**
b	3	∞	4	6	8	**9**
c	7	∞	∞	2	4	**5**
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

$$D_k(i,j) = min(D_{k-1}(i,j),\ D_{k-1}(i,k) + D_{k-1}(k,j))$$

$$V_6 = \{a, b, c, d, e, f\}$$

to

D_5	a	b	c	d	e	f
a	∞	∞	1	3	5	8
b	3	∞	4	6	8	11
c	7	∞	∞	2	4	7
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

from

D_6	a	b	c	d	e	f
a	∞	∞	1	3	5	6
b	3	∞	4	6	8	9
c	7	∞	∞	2	4	5
d	5	∞	6	∞	10	5
e	∞	∞	∞	∞	∞	1
f	∞	∞	∞	∞	∞	∞

Floyd's Algorithm

```
Floyd(G)
{
   // Initialize
   n = |V|;                                          O( )
   for (k = 0; k <= n; k++)
      for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
            d[k][i][j] = infinity;                   O( )

   for (all (v,w) ∈ E)
      d[0][v][w] = C(v,w)                            O( )
```

Floyd’s Algorithm

```
// Compute next distance matrix
for (k = 1; k <= n; k++)
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      d[k][i][j] = min(d[k-1][i][j],                O( )
                       d[k-1][i][k] + d[k-1][k][j]);
```

What About the Paths?

- Can't simply reconstruct them at end
- Add initialization:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        // If edge doesn't exist,      no path
        if (C(i,j) == infinity)
            p[0][i][j] = NIL;
        else
            p[0][i][j] = j;
```

Updating Paths

- When going through the triple-nested loop of the algorithm, if you ever update a weight, you must also update the path
- See code next page

Paths: Primary Loops

```
for (k = 1; k < n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            // Compute next distance matrix
            d[k][i][j] = min(d[k-1][i][j],                                 O( )
                             d[k-1][i][k] + d[k-1][k][j]);
            // Compute next paths matrix
            if (d[k-1][i][j]
                <= d[k-1][i][k] + d[k-1][k][j])
                p[k][i][j] = p[k-1][i][j];
            else                                                           O( )
                p[k][i][j] = p[k-1][k][j];
```

Worst Case Running Time

- Add vertices to V_k one at a time
  - Outer loop executes $|V|$ times
- For each new vertex, consider whether it improves each possible path
  - Inner loops execute $|V|^2$ times
- Overall O($|V|^3$)
- Better than running Dijkstra $|V|$ times?