CS 372 Lecture #29

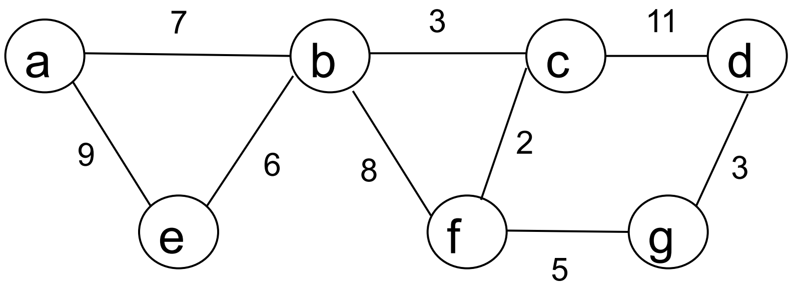
Routing

- **Modeling the network core**
- **Djikstra's Algorithm**

Note: Many of the lecture slides are based on presentations that accompany *Computer Networking: A Top Down Approach,* 6th edition, by Jim Kurose & Keith Ross, Addison-Wesley, 2013.

Optimal routes

- Router software computes optimal routes
- Many algorithms
  - Find shortest path
  - Find path with least traffic
  - Etc.



- Model the network core (group of routers) as a weighted undirected graph
  - "Nodes" represent routers
  - "Edges" model direct connections between routers
  - “Weights” represent costs
    - Costs are determined by speed, distance, additional hardware, traffic, bottlenecks, etc.

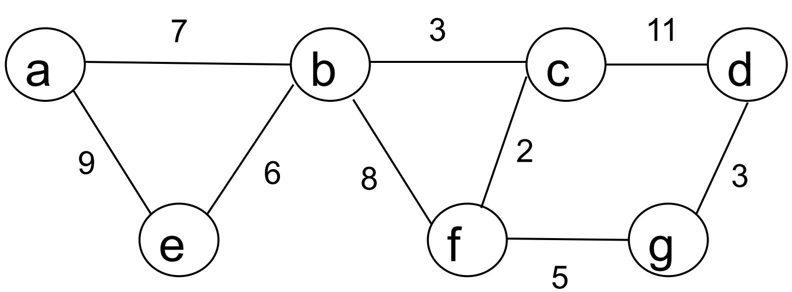
Shortest path

- Shortest path is the path with lowest total weight (sum of weights of all edges in the path)
  - Router groups collaborate to keep cost information current
- Shortest path is not necessarily fewest edges or fewest hops
- First node in shortest path is "next-hop"
  - Insert next-hop information into routing tables

- Djikstra's Algorithm
  - Sometimes called "Link-State Algorithm"
    - … but Link-State <u>uses</u> Djikstra's

Djikstra's Algorithm

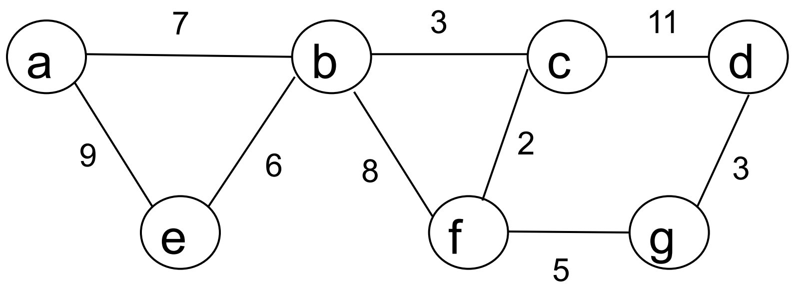
- Data structures:
  - choose *source* node and *destination* node
  - S = {all nodes except source}
  - variables u, v represent nodes
  - D is array of weights of edges
    - Initially, D[v] = edge weight ("cost") if edge from source to v exists
    - Use ∞ to represent the "cost" of a node for which a path has not yet been computed
  - R is an array of nodes
    - Initially, *R*[*v*] = *v* (if edge from *source* to *v* exists) or zero (otherwise)
  - P is an array of nodes
    - Initially, *P* [*v*] = *source* (if an edge from *source* to *v* exists), or zero (otherwise)



Djikstra's Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a "smallest element"
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: "no path"; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	∞	0	0
b	∞	0	0
c	11	c	d
d	*	*	*
e	∞	0	0
f	∞	0	0
g	3	g	d

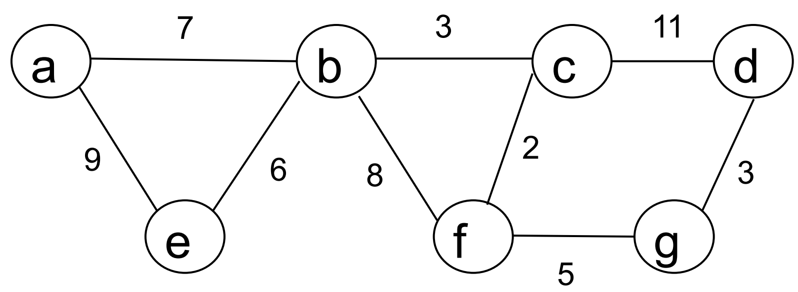
Initialization

S = {a,b,c,e,f,g}

Djikstra's Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
      u = node in S with D[u] a "smallest element"
                ... if tied, take smallest u;
      if(D[u] == ∞) {
           error: "no path"; exit;}
      S = S - {u};
      for (each v such that edge (u,v) exists) {
           if(v in S) {
                cost = D[u] + weight (u,v);
                if(c < D[v]) {
                     D[v] = cost;
                     R[v] = R[u];
                     P[v] = u;
                }
           }
      }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	∞	0	0
b	∞	0	0
c	11	c	d
d	*	*	*
e	∞	0	0
f	~~∞~~ 8	~~0~~g	~~0~~g
g	3	g	d

Iteration #1   S = {a,b,c,e,f,g}

u = g   (smallest D[u], u in S)

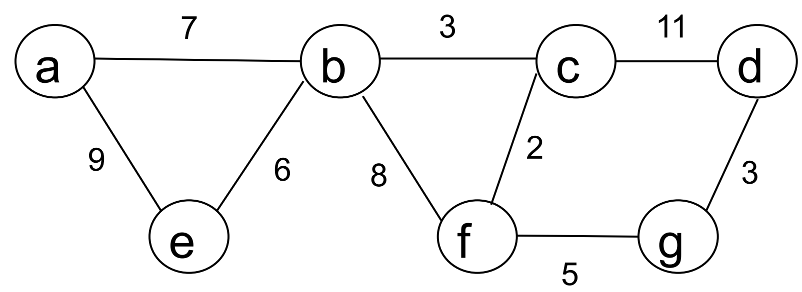
S = {a,b,c,e,f,~~g~~}

v = f   cost = 3 + 5 = 8

Djikstra’s Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a “smallest element”
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: “no path”; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	∞	0	0
b	~~∞~~16	~~0~~g	~~0~~f
c	~~11~~10	~~c~~g	~~d~~f
d	*	*	*
e	∞	0	0
f	~~∞~~8	~~0~~g	~~0~~g
g	3	g	d

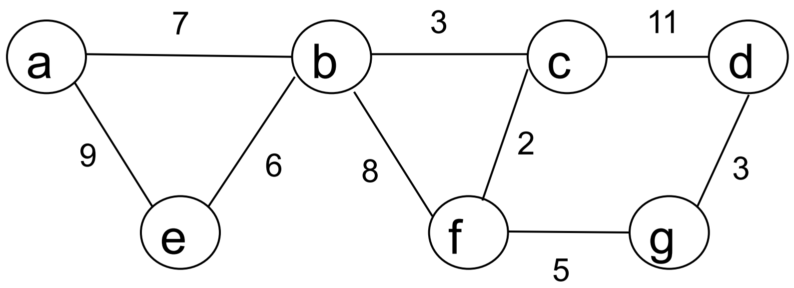
Iteration #2 S = {a,b,c,e,f}

u = f (smallest D[u], u in S)
S = {a,b,c,e,~~f~~}
v = c cost = 8 + 2 = 10
v = b cost = 8 + 8 = 16

Djikstra's Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a "smallest element"
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: "no path"; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	∞	0	0
b	~~∞~~ ~~16~~13	~~0~~ ~~g~~ g	~~0~~ ~~f~~ c
c	~~11~~10	~~c~~g	~~d~~f
d	*	*	*
e	∞	0	0
f	~~∞~~ 8	~~0~~g	~~0~~g
g	3	g	d

Iteration #3 S = {a,b,c,e}

u = c (smallest D[u], u in S)

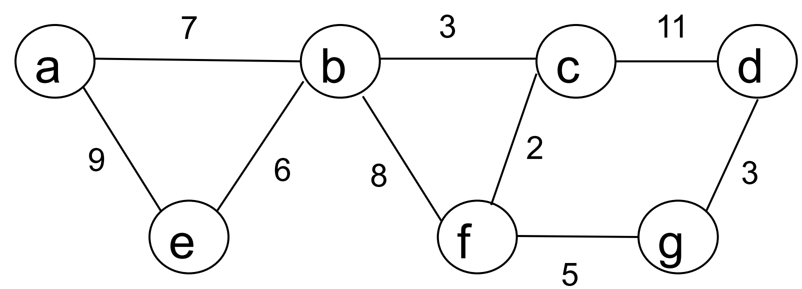
S = {a,b,~~c~~,e}

v = b cost = 10 + 3 = 13

Djikstra’s Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a "smallest element"
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: "no path"; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	~~∞~~ 20	~~0~~g	~~0~~b
b	~~∞~~ ~~16~~13	~~0g~~ g	~~0f~~ c
c	~~11~~10	~~c~~g	~~d~~f
d	*	*	*
e	~~∞~~ 19	~~0~~g	~~0~~b
f	~~∞~~ 8	~~0~~g	~~0~~g
g	3	g	d

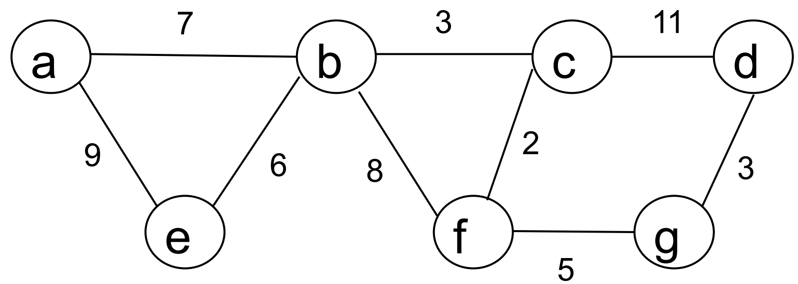
Iteration #4 S = {a,b,e}

u = b (smallest D[u], u in S)
S = {a,~~b~~,e}
v = a cost = 13 + 7 = 20
v = e cost = 13 + 6 = 19

Djikstra’s Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a “smallest element”
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: “no path”; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	~~∞~~ 20	~~0~~g	~~0~~b
b	~~∞~~ ~~16~~13	~~0~~ ~~g~~ g	~~0~~ ~~f~~ c
c	~~11~~10	~~c~~g	~~d~~f
d	*	*	*
e	~~∞~~ 19	~~0~~g	~~0~~b
f	~~∞~~ 8	~~0~~g	~~0~~g
g	3	g	d

Iteration #5 S = {a,e}

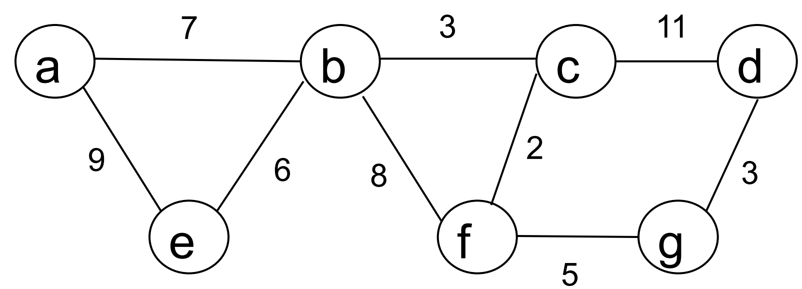
u = e (smallest D[u], u in S)
S = {a,~~e~~}
v = a cost = 19 + 9 = 28

Note: cost is <u>not less than D[a]</u>, so don’t replace.

Djikstra's Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a "smallest element"
              ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: "no path"; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	~~∞~~ **20**	~~0~~g	~~0~~ b
b	~~∞~~ ~~16~~13	~~0~~~~g~~ g	~~0~~~~f~~ c
c	~~11~~10	~~c~~g	~~d~~ f
d	*	*	*
e	~~∞~~19	~~0~~g	~~0~~ b
f	~~∞~~8	~~0~~g	~~0~~ g
g	3	g	d

Iteration #6   S = {a}

u = a   (smallest D[u], u in S)
S = {~~a~~}

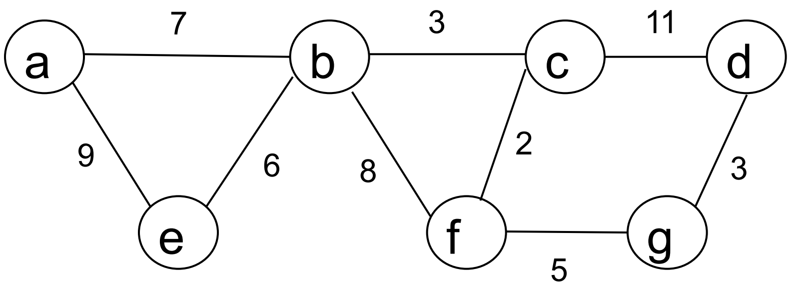
Note: S is empty, so the for loop does nothing, and the while loop terminates.

Now, start from destination and trace backwards in P:
a ⇦ b ⇦ c ⇦ f ⇦ g ⇦ d
Shortest path is d-g-f-c-b-a (cost = 20)

Djikstra's Algorithm

```
initialize S, D, R, P;
while (!empty(S)) {
     u = node in S with D[u] a "smallest element"
               ... if tied, take smallest u;
     if(D[u] == ∞) {
          error: "no path"; exit;}
     S = S - {u};
     for (each v such that edge (u,v) exists) {
          if(v in S) {
               cost = D[u] + weight (u,v);
               if(c < D[v]) {
                    D[v] = cost;
                    R[v] = R[u];
                    P[v] = u;
               }
          }
     }
}
```

Example: Find shortest path from d to a



Dest	D	R	P
a	~~∞~~ 20	~~∅~~ g	~~∅~~ b
b	~~∞~~ ~~16~~13	~~∅~~ ~~g~~ g	~~∅~~ ~~f~~ c
c	~~11~~10	~~c~~ g	~~d~~ f
d	*	*	*
e	~~∞~~ 19	~~∅~~ g	~~∅~~ b
f	~~∞~~ 8	~~∅~~ g	~~∅~~ g
g	3	g	d

Shortest path is d-g-f-c-b-a
Note: P is not stored by the router.
What does R represent?

It's the complete routing table for router d

Further study

- Graph theory important in advanced computer research
  - major topic
    - networking
    - artificial intelligence
- Other optimal path algorithms
  - Link-state
  - Distance-vector

Summary Lecture #29

- Graph representation of router group
- Shortest path
  - Computed by weight (<u>not</u> by minimum hops)
- Djikstra's algorithm
  - several forms exist
    - use at least one form to compute shortest path