Graphs: Shortest Paths

Bijendra Nath Jain

bnjain@iiitd.ac.in

Outline of next 9 lectures

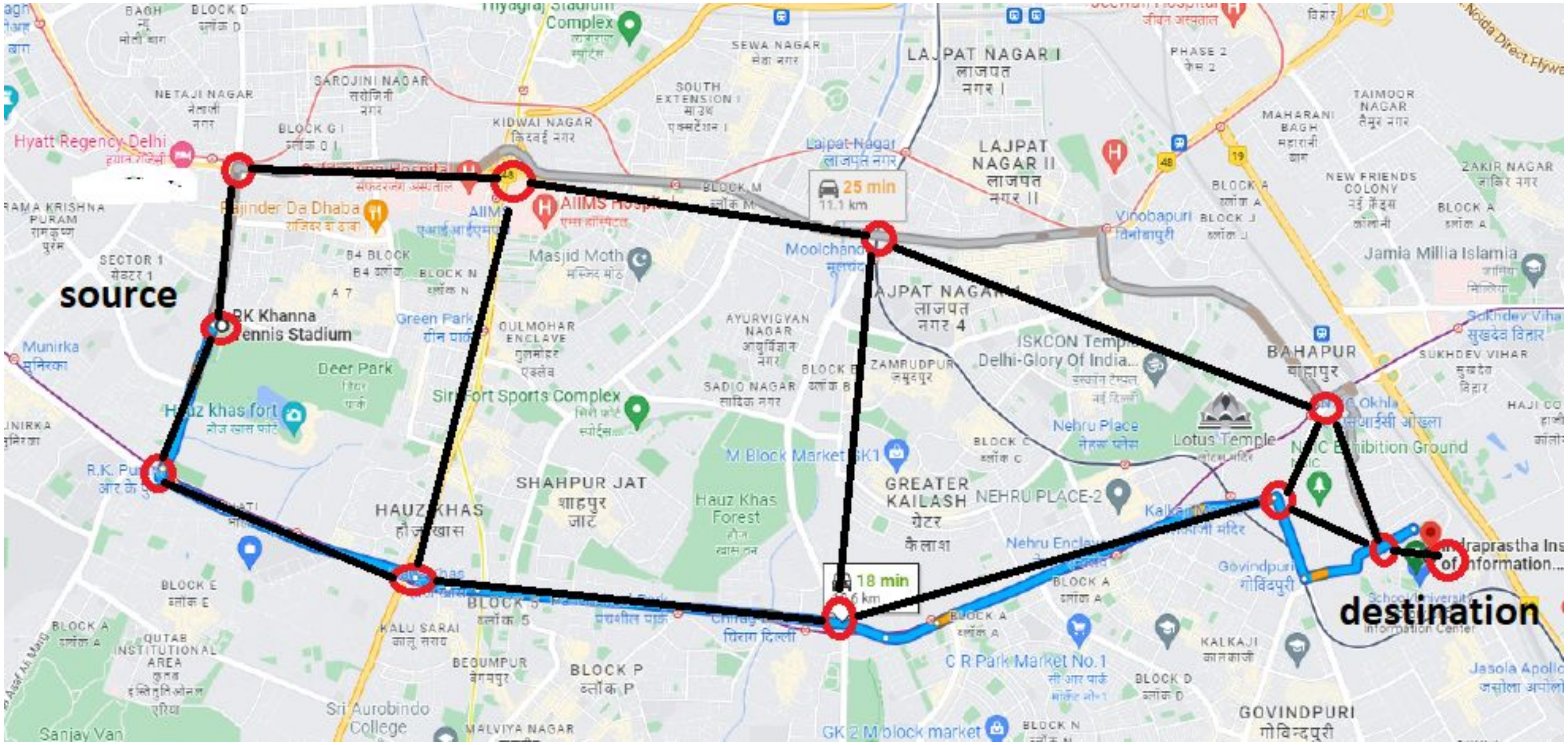
- Graphs:
  - Undirected graphs
  - Directed graphs
  - (Directed) acyclic graphs (or DAGs)
  - Sparse graphs
  - Weighted graphs
- Graph applications
- Representation of graphs:
  - Adjacency matrix
  - Linked lists
- Algorithms:
  - Traversal algorithms:
    - BFS
    - DFS
  - Topological sort
  - Minimum spanning trees
  - Dijkstra's Shortest path
    - One-to-one
    - One-to-many
    - Many-to-many

Applications of Shortest Path algorithms

- Applicable to any/every kind of networks
  - Road travel
  - Air travel
  - Internet
  - Speed-post/courier delivery
  - Etc.

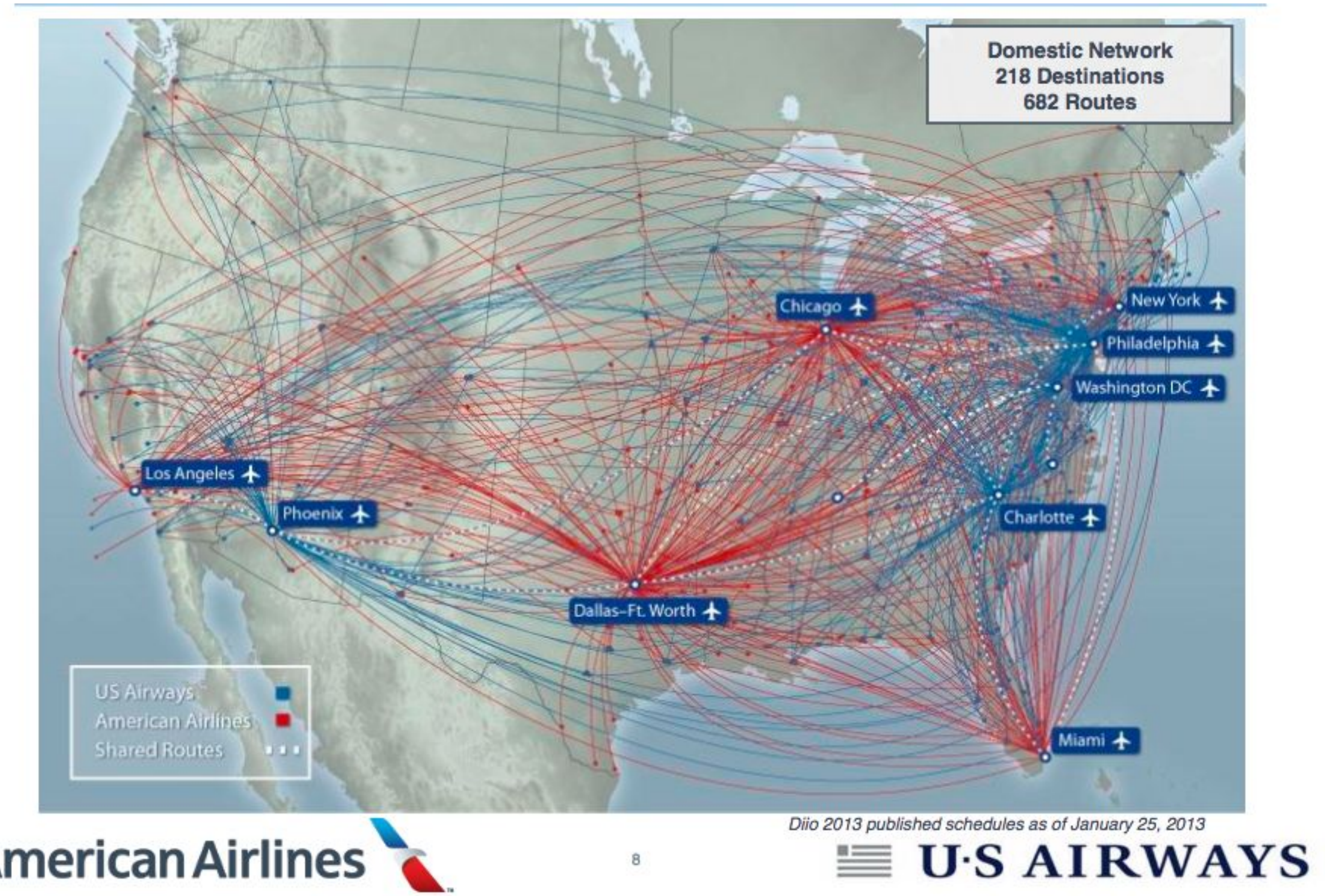
Applications of Shortest Path algorithms

- Applicable to any/every kind of networks
  - Road travel –optimization of travel time



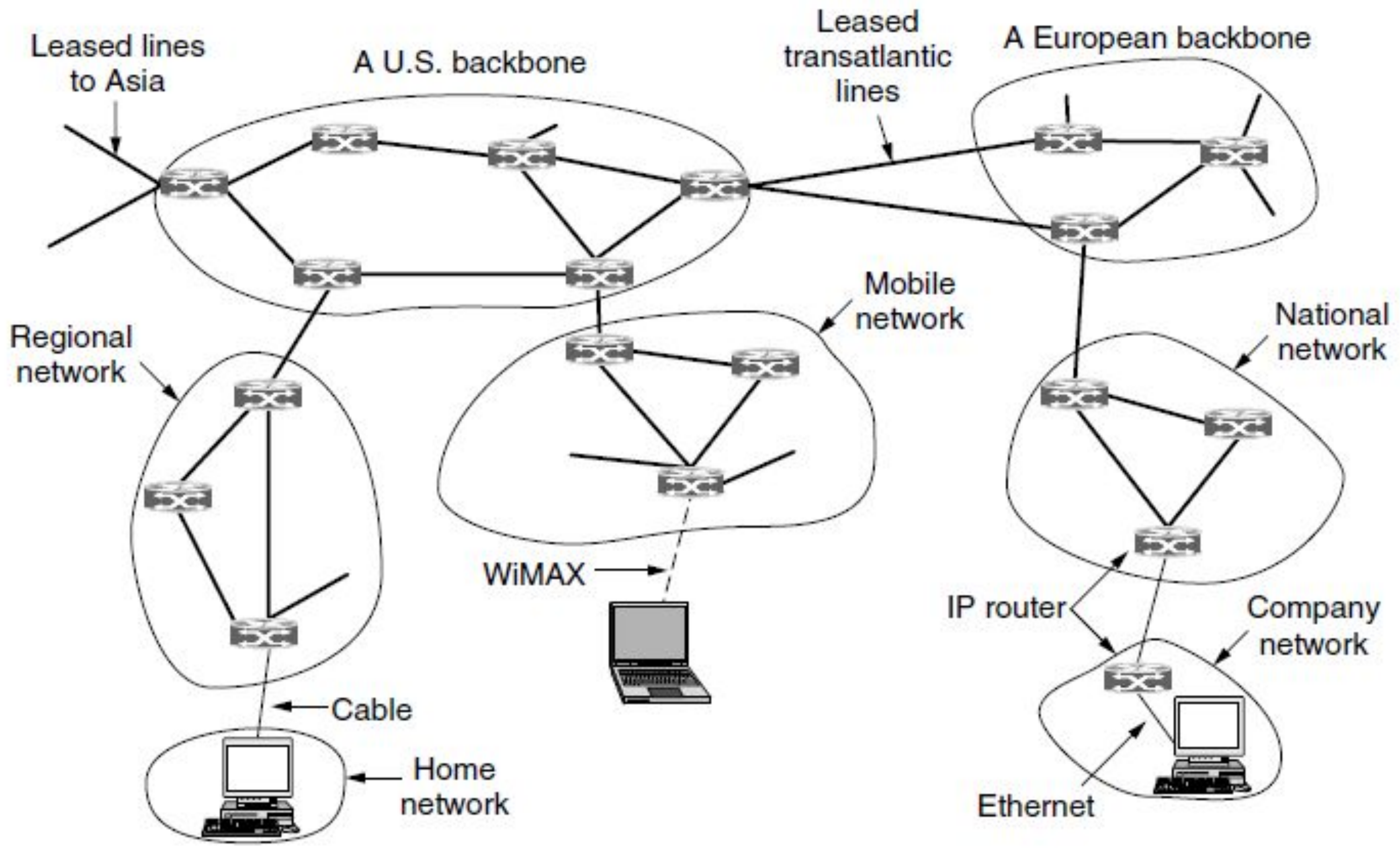
Applications of Shortest Path algorithms

- Applicable to any/every kind of networks
  - Airline network – optimization of airfare



Applications of Shortest Path algorithms

- Applicable to any/every kind of networks
  - Internet – optimization of end-to-end delay, using “link-state routing” within a routing domain



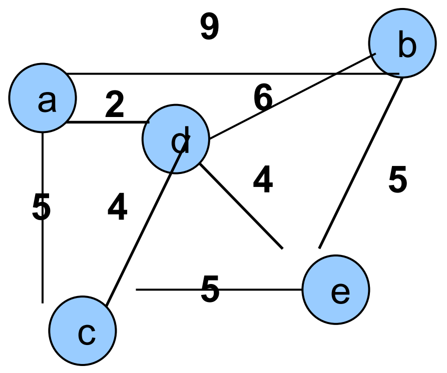
Applications of Shortest Path algorithms

- Shortest path problems
  - Single source-single destination routing
  - One-to-many or single source routing
  - Many-to-many or all pairs routing

Single-source-single-destination routing

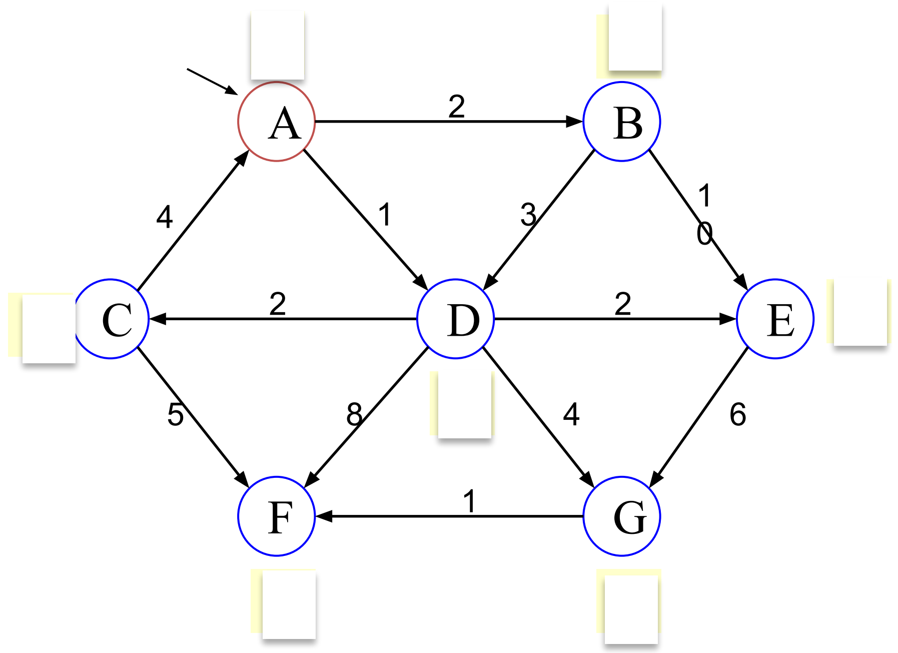
- Brute force technique
  - Consider a 5 node network with “distance” or weight associated with each edge
  - Compute “shortest” path between a b
  - To do so, list all possible paths a b, and compute the distance along the path
  - Example: for a b:

a-b	9
a-c-d-b	15
a-c-d-e-b	18
a-c-e-b	…
a-c-e-d-b	…
a-d-b	8
a-d-c-e-b	…
etc.	



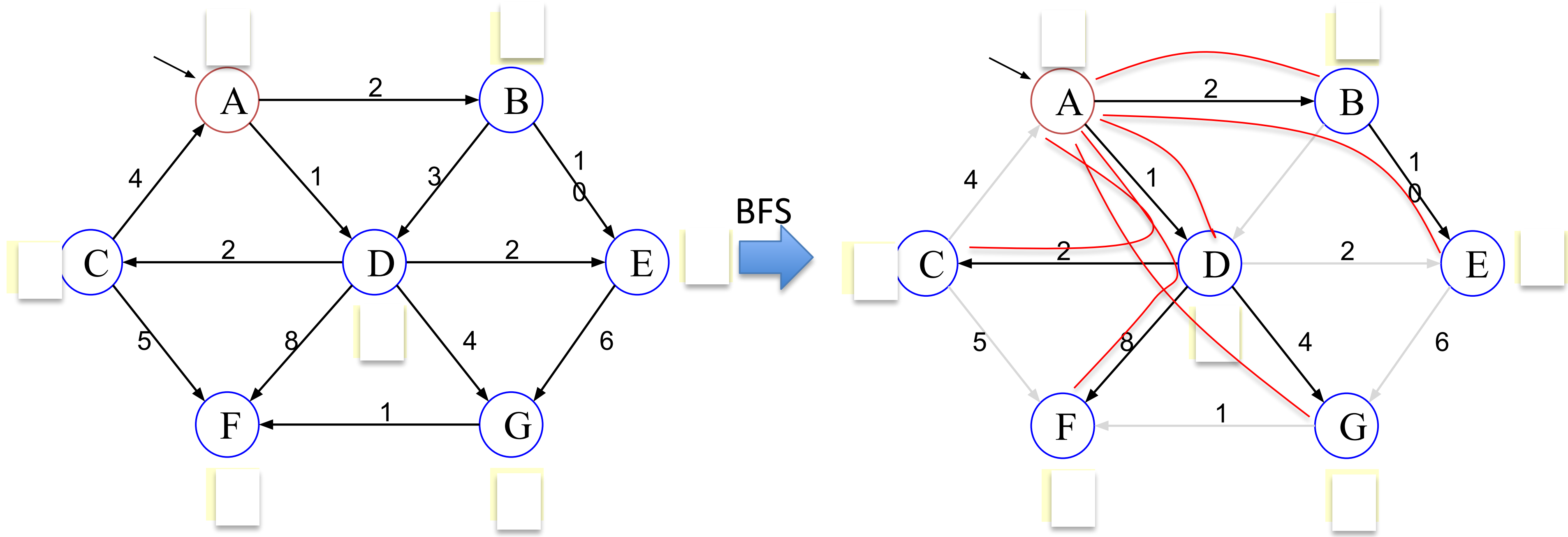
Single-source-all-destination routing

- Consider the 7 vertex network, with distances associated with each edge



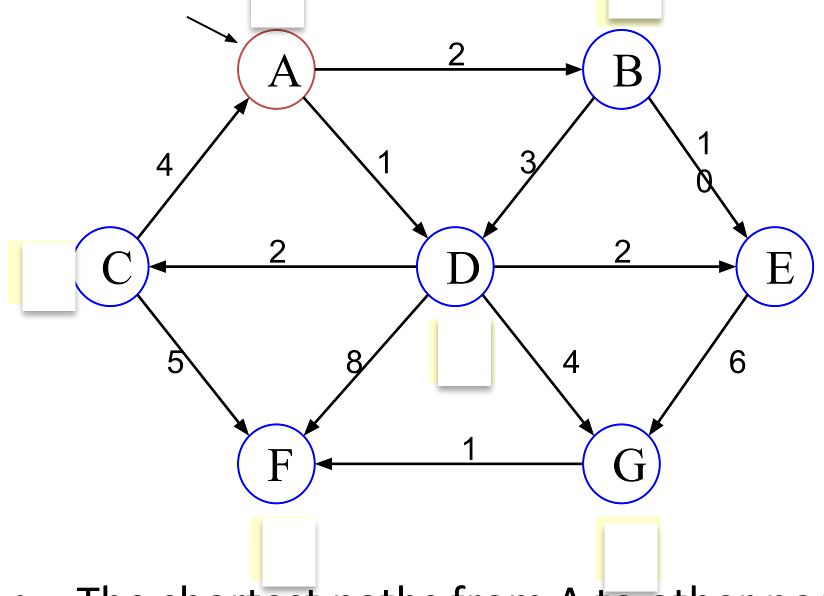
Single-source-all-destination routing

- Run the BFS algorithm, and identify routes, compute path lengths

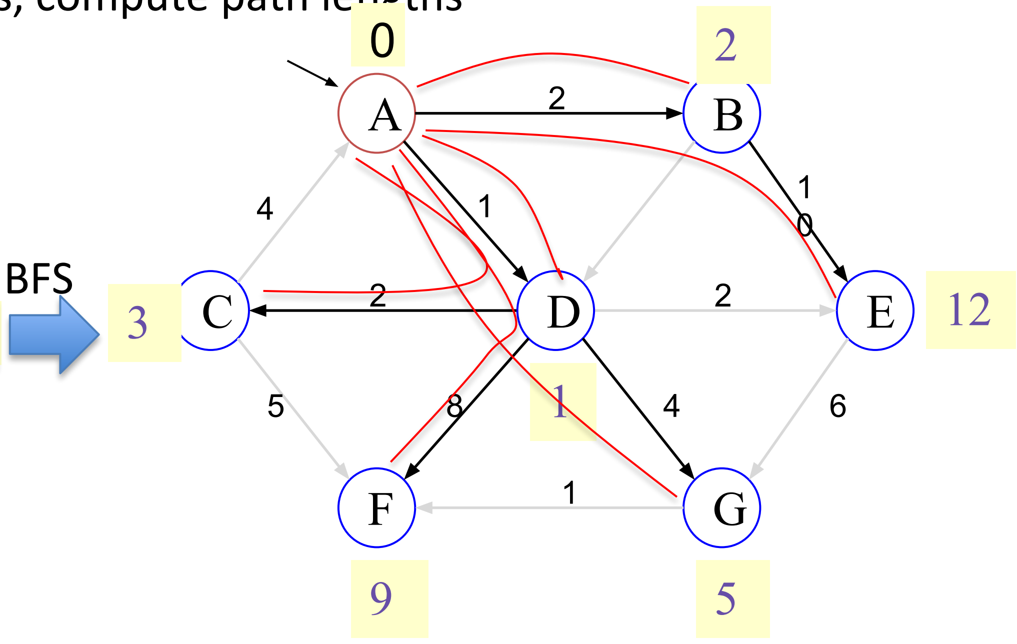


Single-source-all-destination routing

- Run the BFS algorithm, and identify routes, compute path lengths

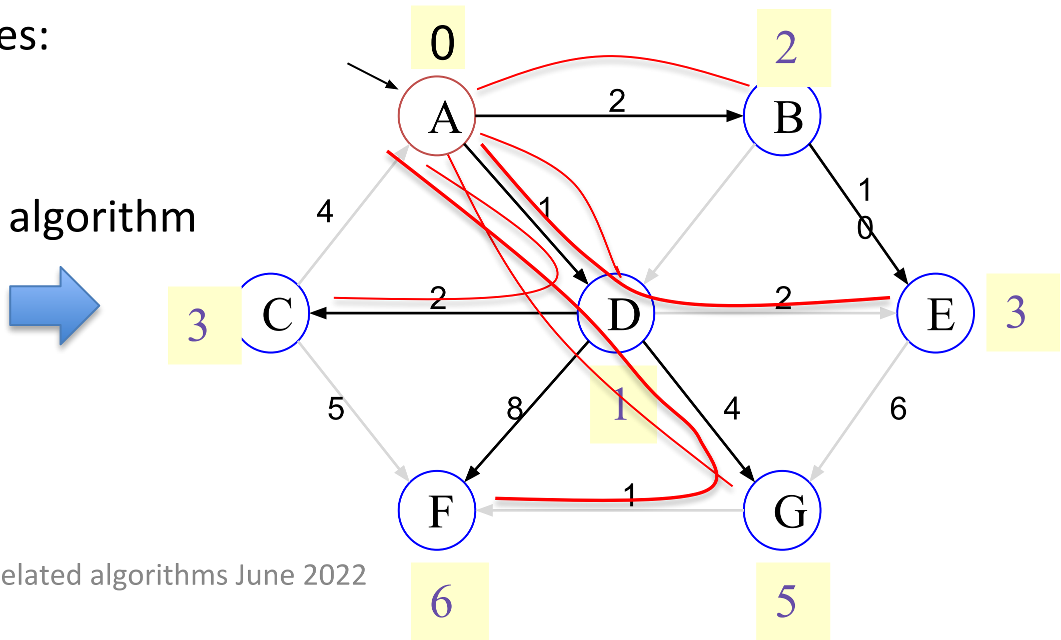


BFS



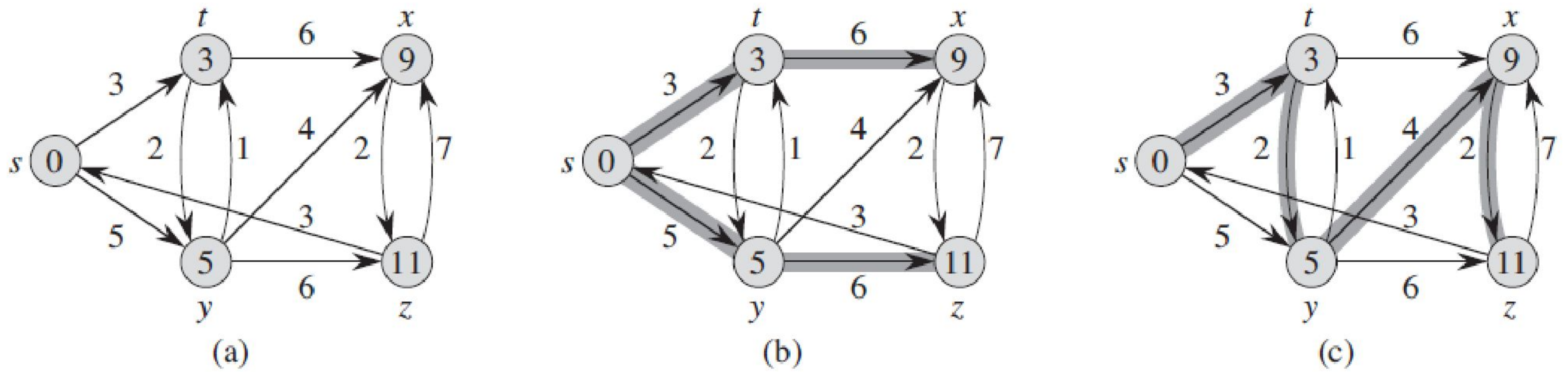
- The shortest paths from A to other nodes:

Dijkstra's algorithm



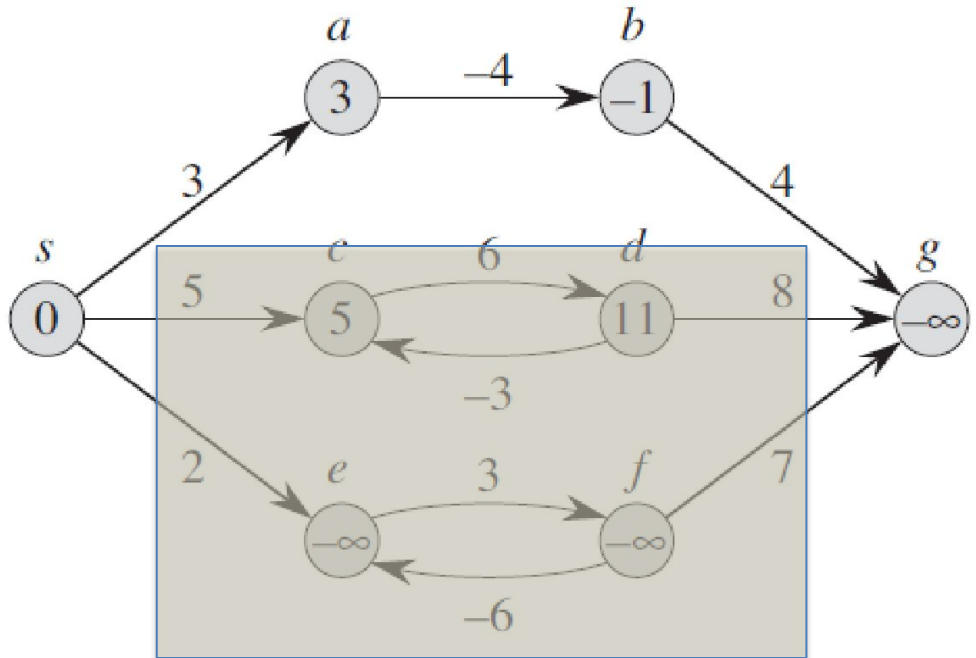
Single-source-all-destination routing

- Works on both directed and undirected graphs, but with nonnegative weights



Single-source-all-destination routing: Dijkstra's algorithm

- Works on both directed and undirected graphs, but with nonnegative weights
- However consider following graphs with edges that have negative weight
  - What is the shortest path:
    - s a
    - s b
    - s g



Single-source-all-destination routing: Dijkstra's algorithm

- Works on both directed and undirected graphs, but with nonnegative weights
- However consider following graphs with edges that have negative weight
  - What is the shortest path:

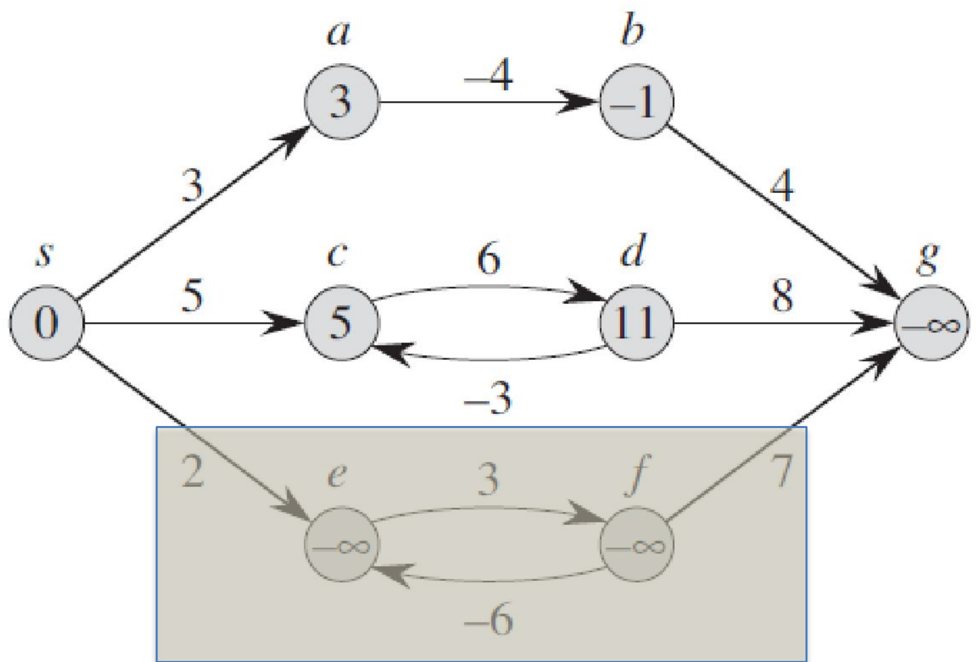
    s □ a

    s □ b

    s □ c

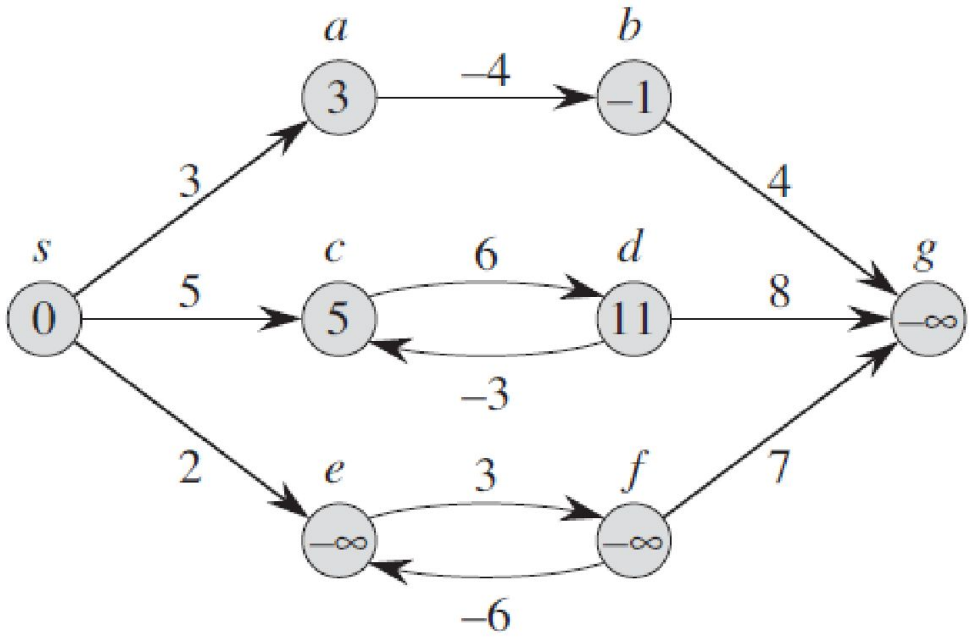
    s □ d

    s □ g
  - The problem is not so much with negative weights



Single-source-all-destination routing: Dijkstra’s algorithm

- Works on both directed and undirected graphs, but with nonnegative weights
- However consider following graphs with edges that have negative weight
  - What is the shortest path:
    - s □ a
    - s □ b
    - s □ c
    - s □ d
    - s □ e
    - s □ f
    - s □ g
  - The problem is not so much with negative weights, but with cycles that have negative weights



Single-source-all-destination routing: Dijkstra's algorithm

Dijkstra's algorithm:

- Input: Weighted graph G = (E,V), weights W, source vertex *s* ∈ V
- Output: Shortest paths from vertex *s* ∈ V to all other vertices

Mimics Prim's algorithm:

1a. Call s = u_0. Identify shortest path s → u_0, with path weight/distance = 0
1b. Set distance to all other vertices as ∞
2a. Compute (update) shortest path from s to all other vertices reachable from u_0, but going through vertex s or u_0
2b. Sort all vertices, other than u_0, in non-decreasing order of their path lengths
2c. Identify vertex u_1 with the shortest path length, and freeze the shortest path
3 Repeat steps 2a-2c n-1 times, viz. there are no more vertices to be considered

Put differently:

- Let $\delta(u)$: shortest path from s to u
- The algorithm runs in n iterations: in iteration i, it finds vertex u_i and $\delta(u_i)$ in $\mathrm{non-decresing}$ order of $\delta(u_i)$

Single-source-all-destination routing: Dijkstra’s algorithm

Dijkstra’s algorithm:

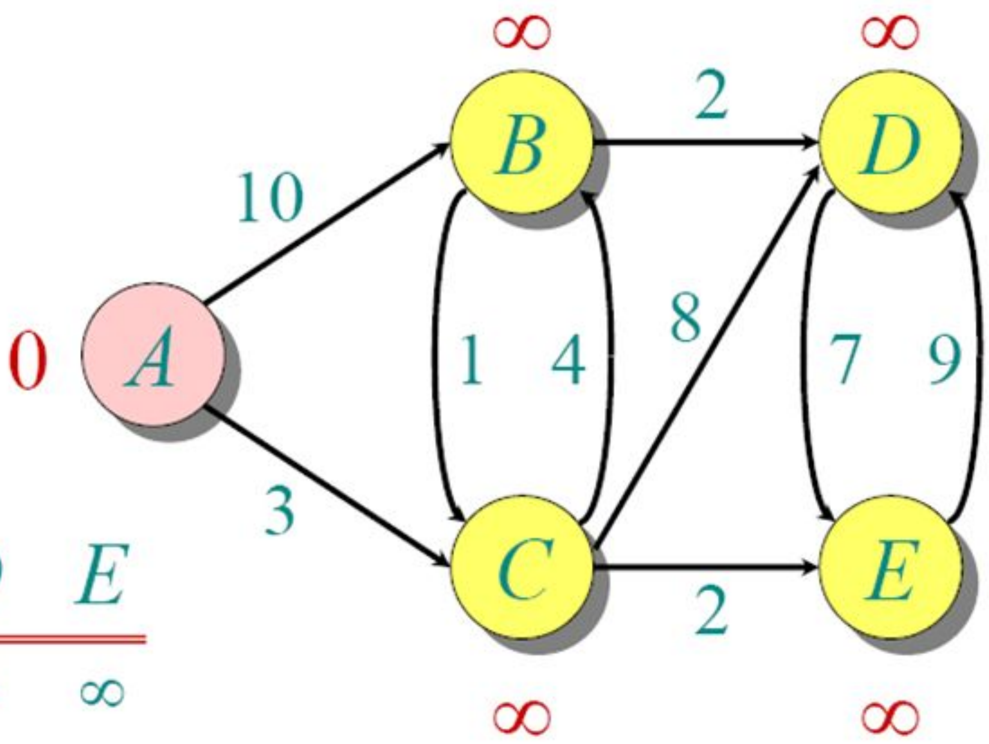
```
Initialize $S = \emptyset, D[s] = 0, D[u] = \infty, u \neq s$

for i = 1, ..., n
    Let u* be the vertex with min $D[u]$
    Add u* to S
    For every $u \notin S, (u^*, u) \in E$
        $D[u] = \min(D[u], D[u^*] + w(u^*, u))$
        parent(u) = u* if D[u] was updated
```

Vertex u is presently estimated to be D[u] away

Maintain a min binary heap of $u \notin S$ with u.key = D[u]

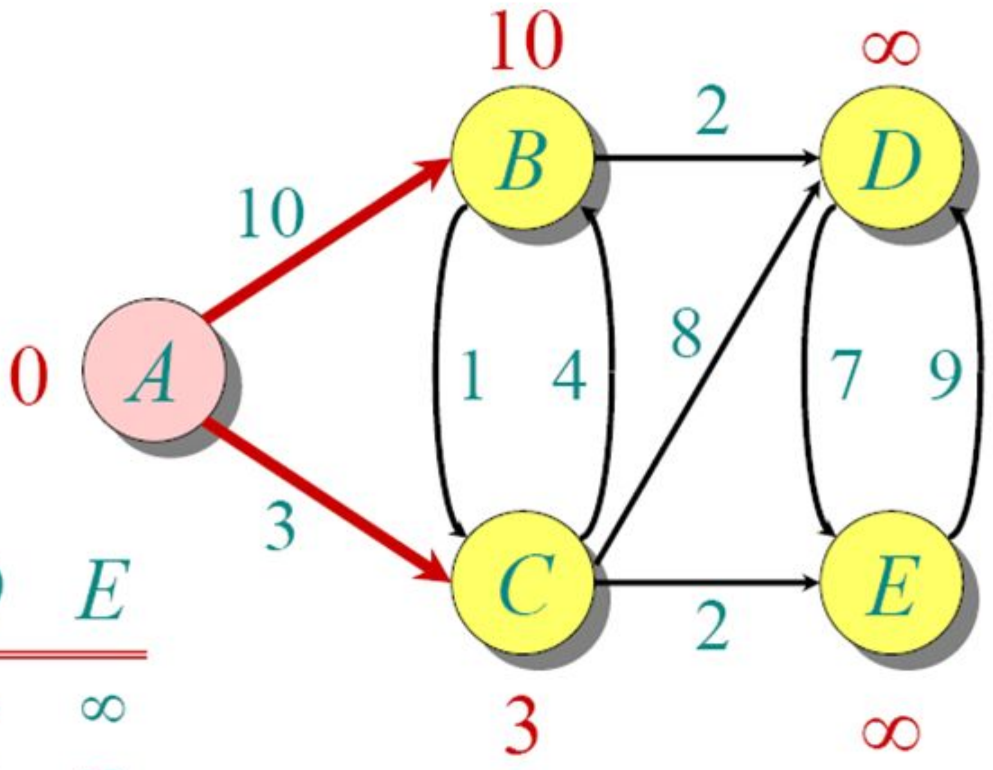
whether a vertex is in S is determined by setting a bit in array S

Single-source-all-destination routing: Dijkstra’s algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

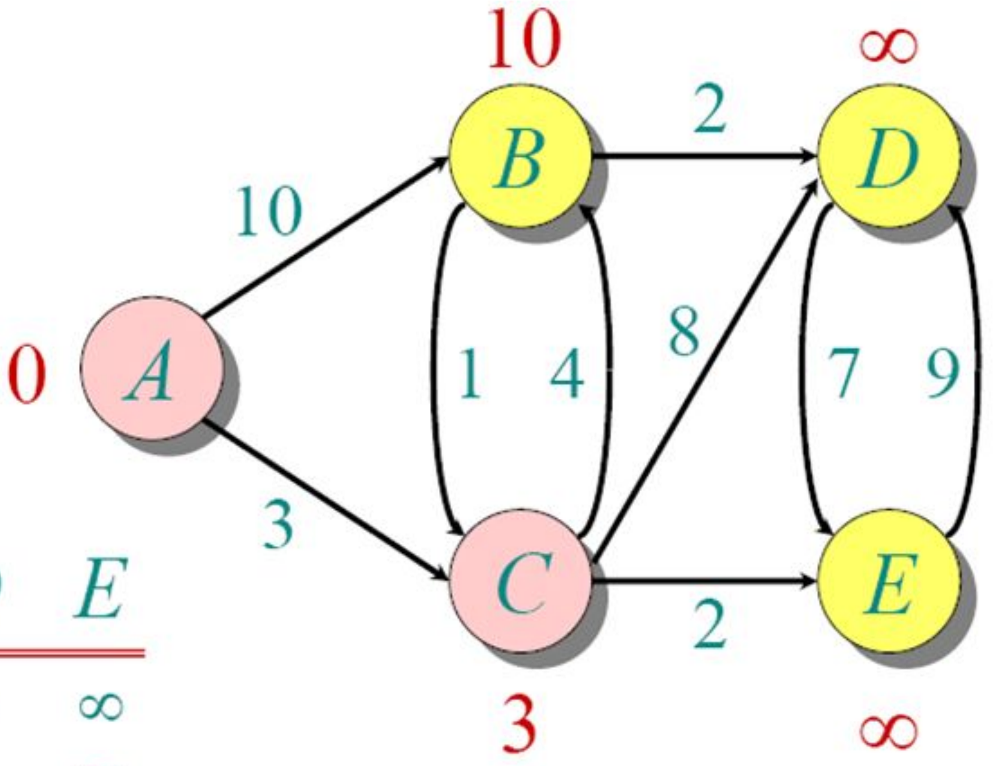
Single-source-all-destination routing: Dijkstra's algorithm



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

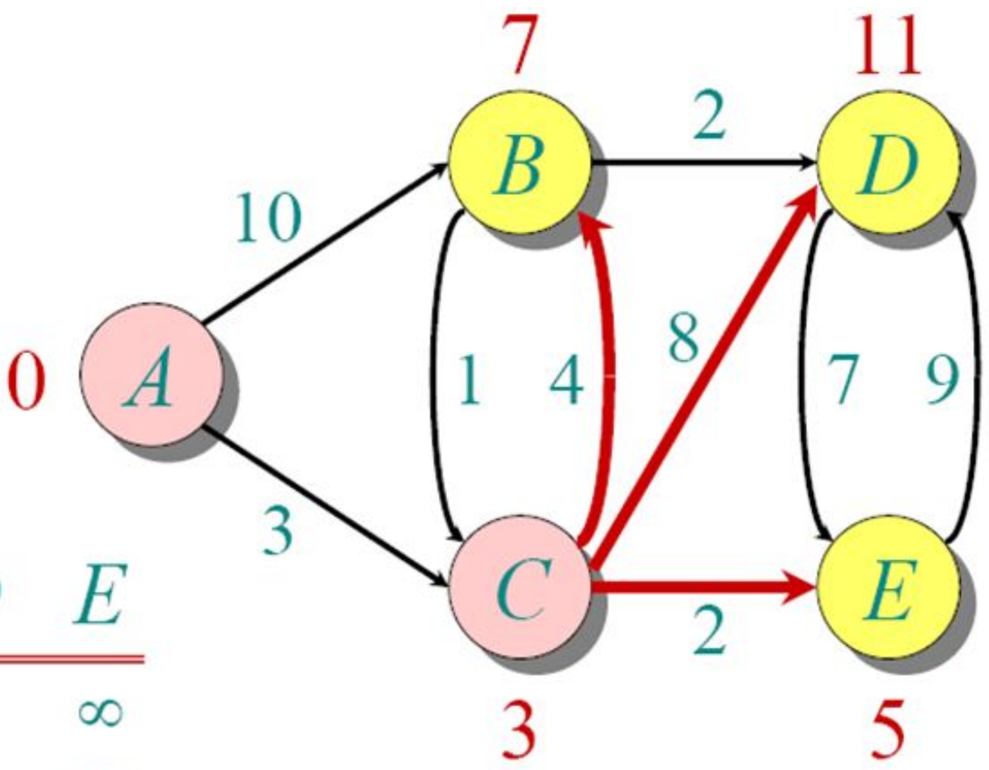
Single-source-all-destination routing: Dijkstra’s algorithm



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }

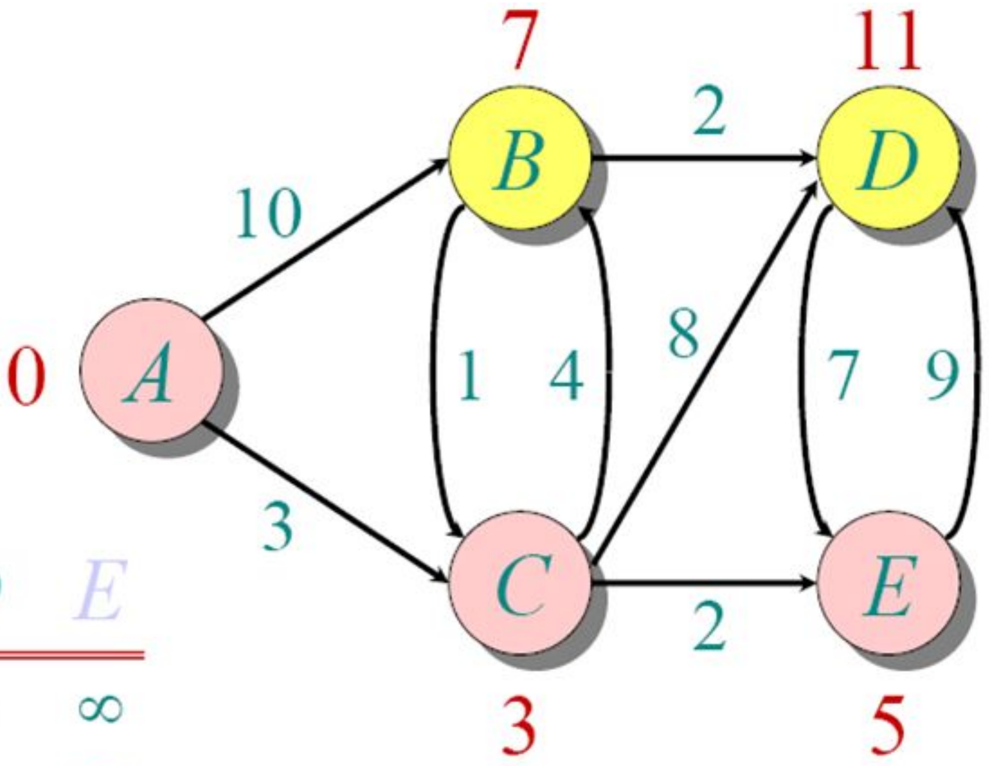
Single-source-all-destination routing: Dijkstra's algorithm



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

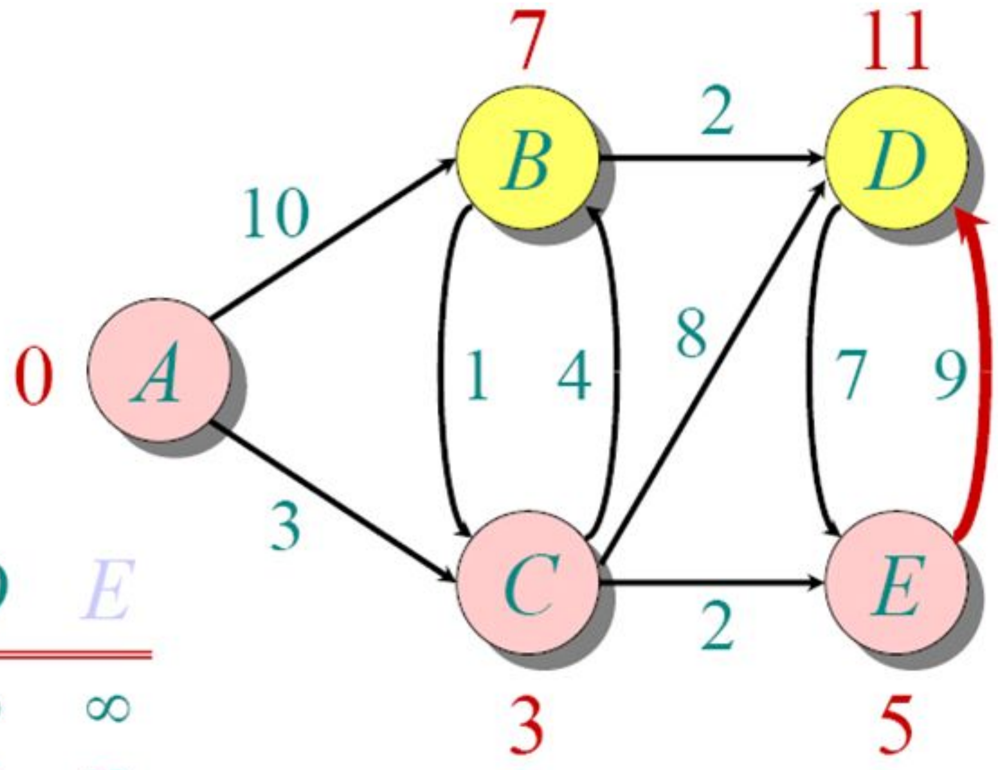
Single-source-all-destination routing: Dijkstra's algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

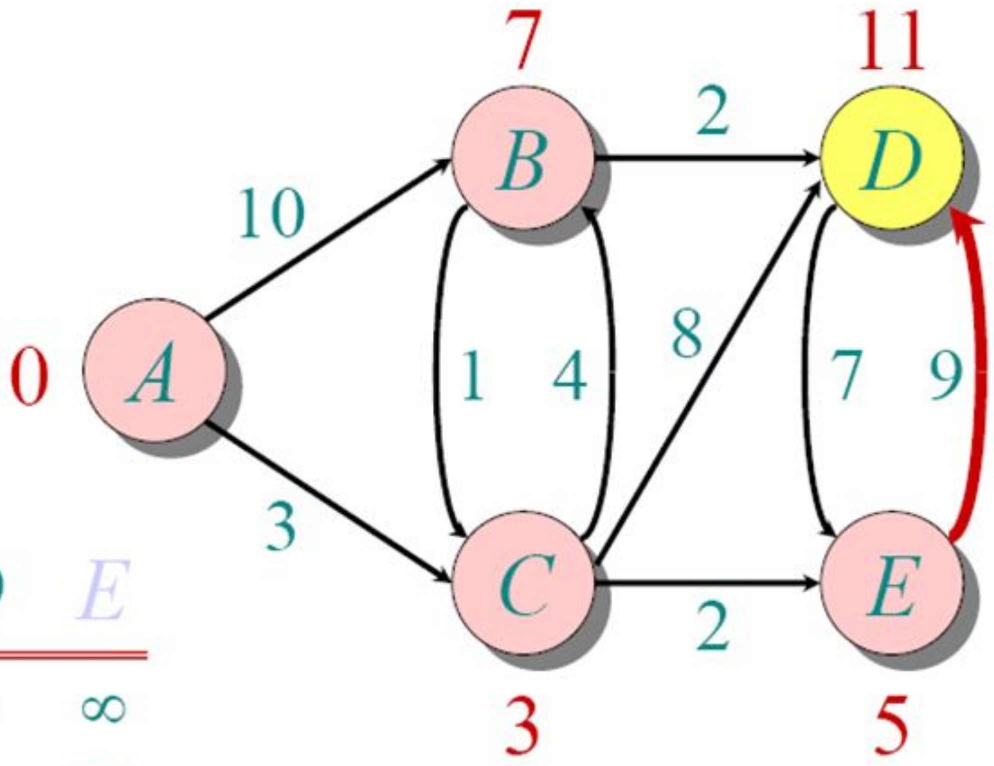
Single-source-all-destination routing: Dijkstra's algorithm



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Single-source-all-destination routing: Dijkstra’s algorithm

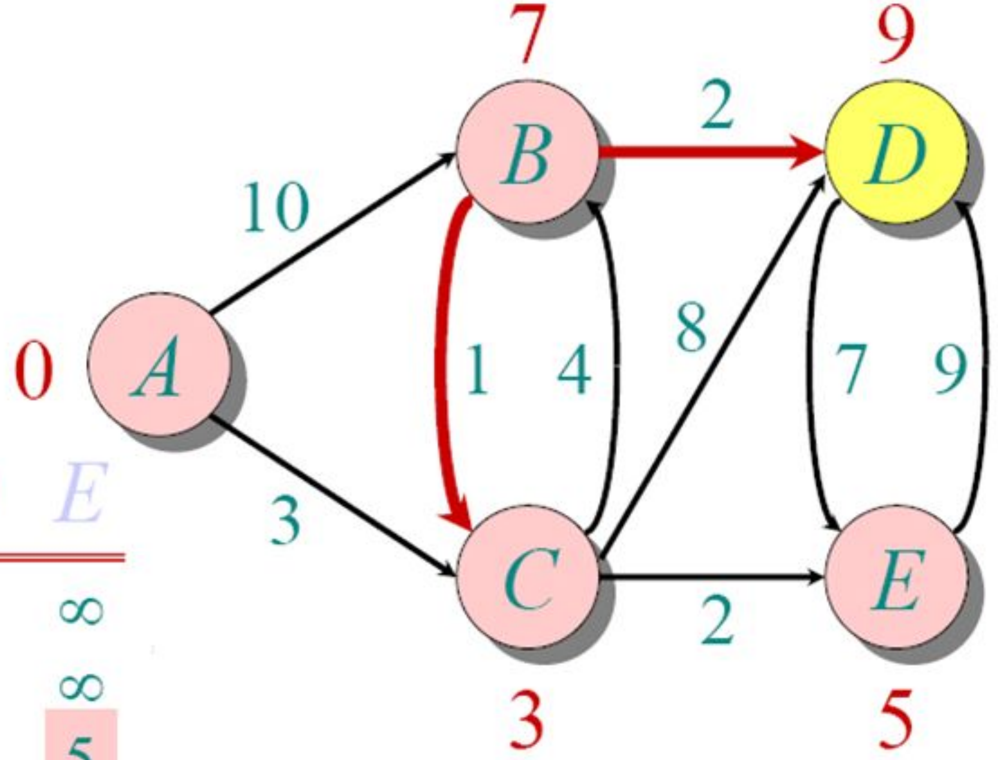


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E*, *B* }

Single-source-all-destination routing: Dijkstra's algorithm

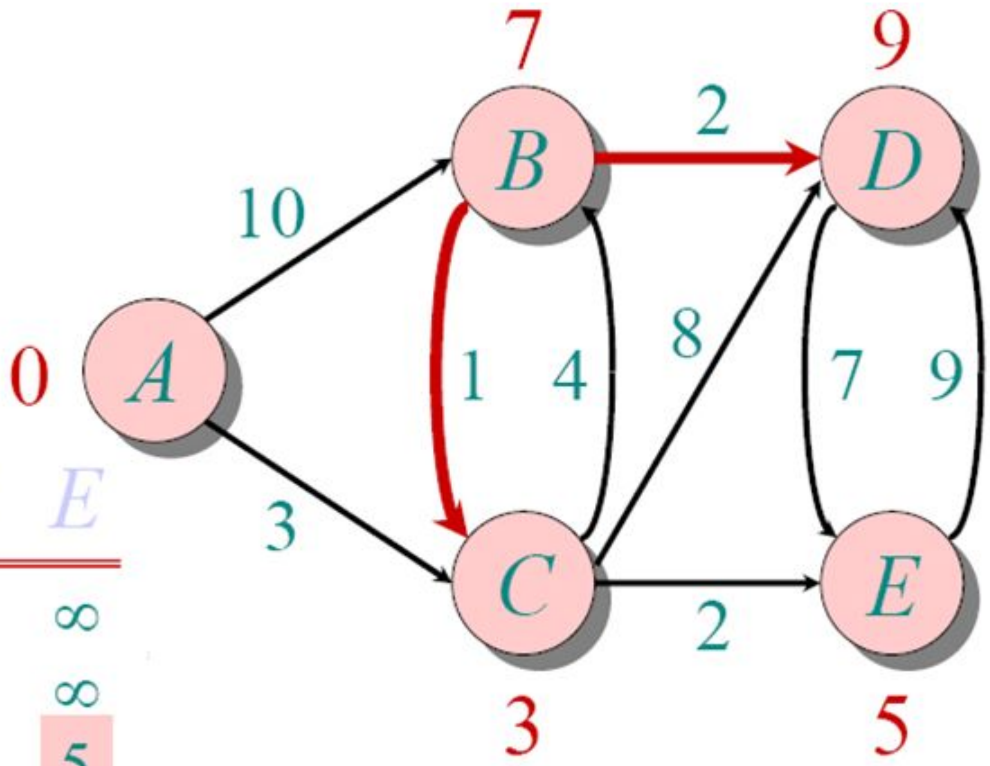
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	



S: { *A*, *C*, *E*, *B* }

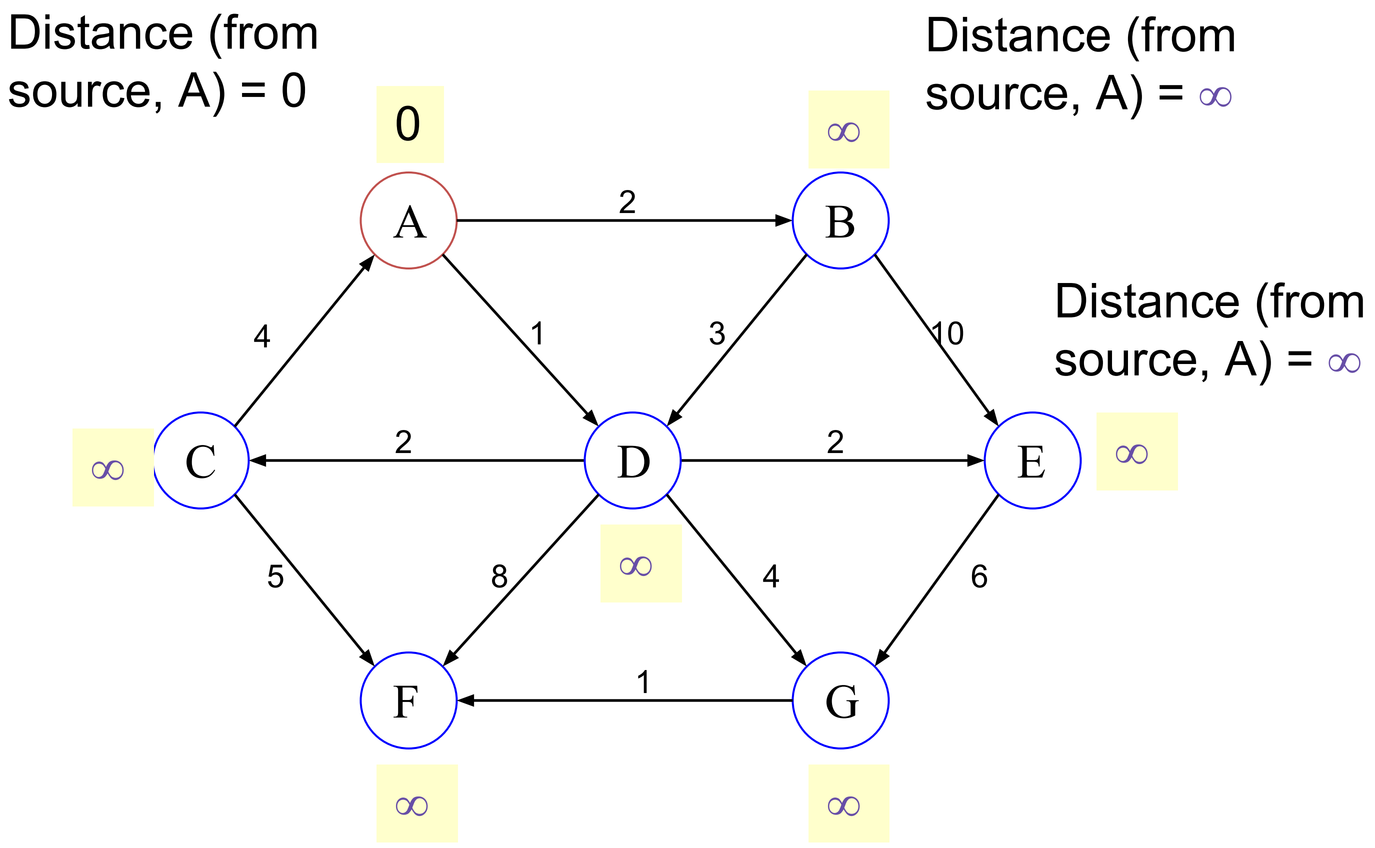
Single-source-all-destination routing: Dijkstra's algorithm

Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				$\boxed{9}$	



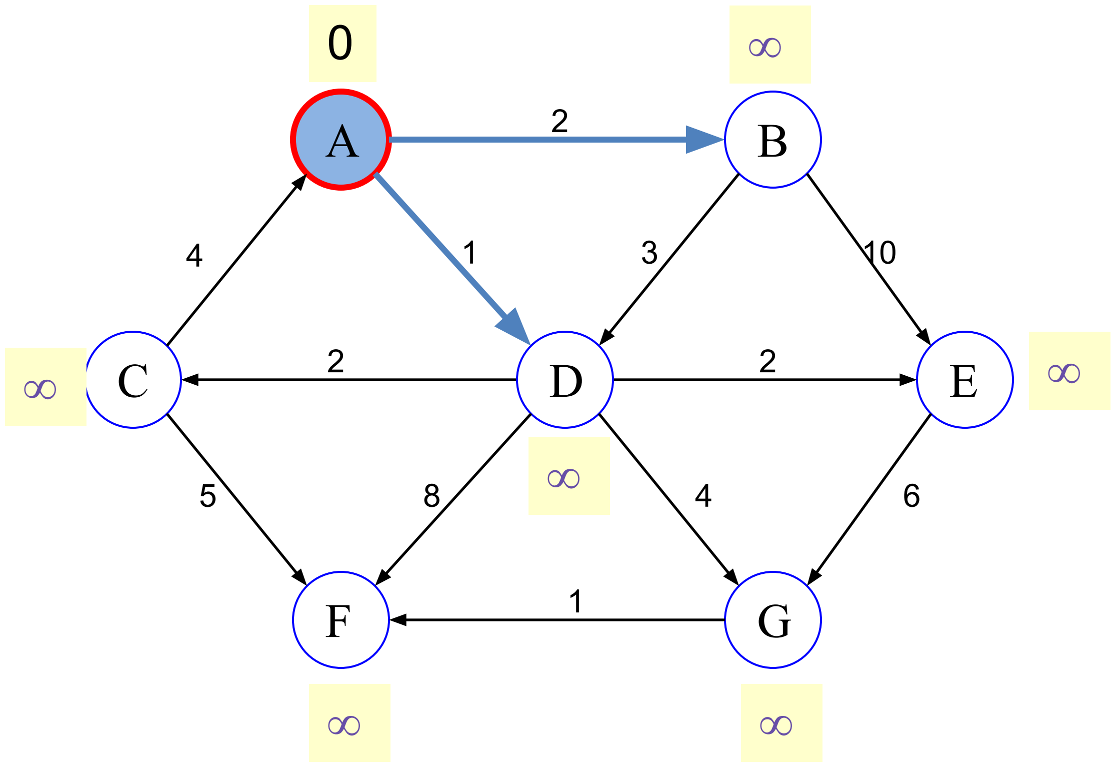
S: { A, C, E, B, D }

Single-source-all-destination routing: Dijkstra's algorithm


Distance (from source, A) = 0
Distance (from source, A) = ∞
Distance (from source, A) = ∞
0
∞
∞
∞
∞
∞
∞
A
B
C
D
E
F
G
2
4
1
3
10
2
2
5
8
4
6
1

Single-source-all-destination routing: Dijkstra's algorithm

S = {A}
D(A) = 0

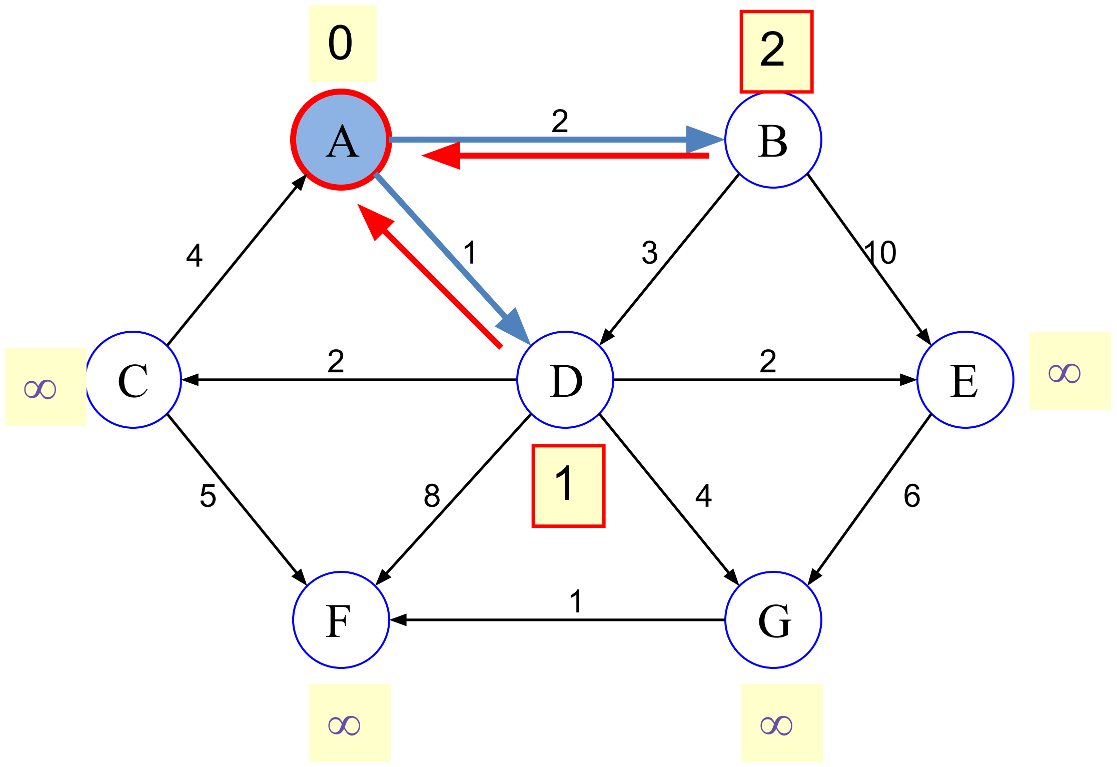


Single-source-all-destination routing: Dijkstra's algorithm

S = {A}

D(A) = 0

Update D(B), D(D), parent(B), parent(D)



Single-source-all-destination routing: Dijkstra's algorithm

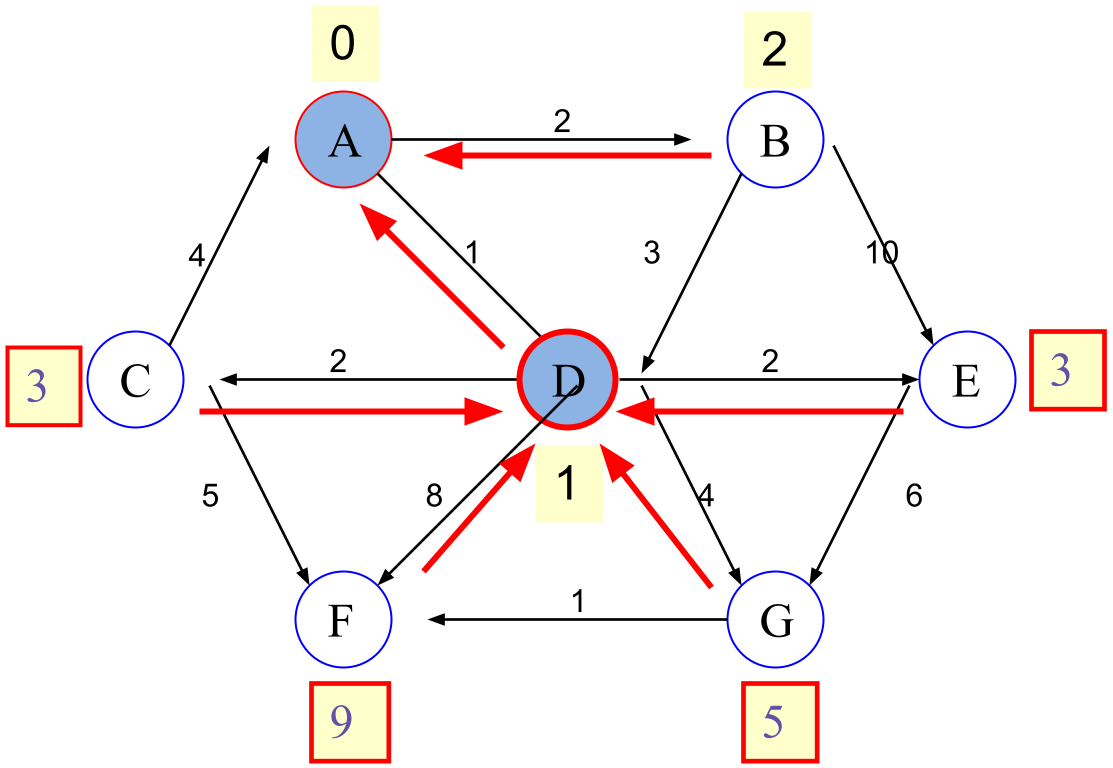
S = {A}

D(A) = 0

Update D(B), D(D), parent(B), parent(D)

S = S U {D}

Update D(C), D(F), D(G), D(E), parent(…)



Single-source-all-destination routing: Dijkstra's algorithm

S = {A}

D(A) = 0

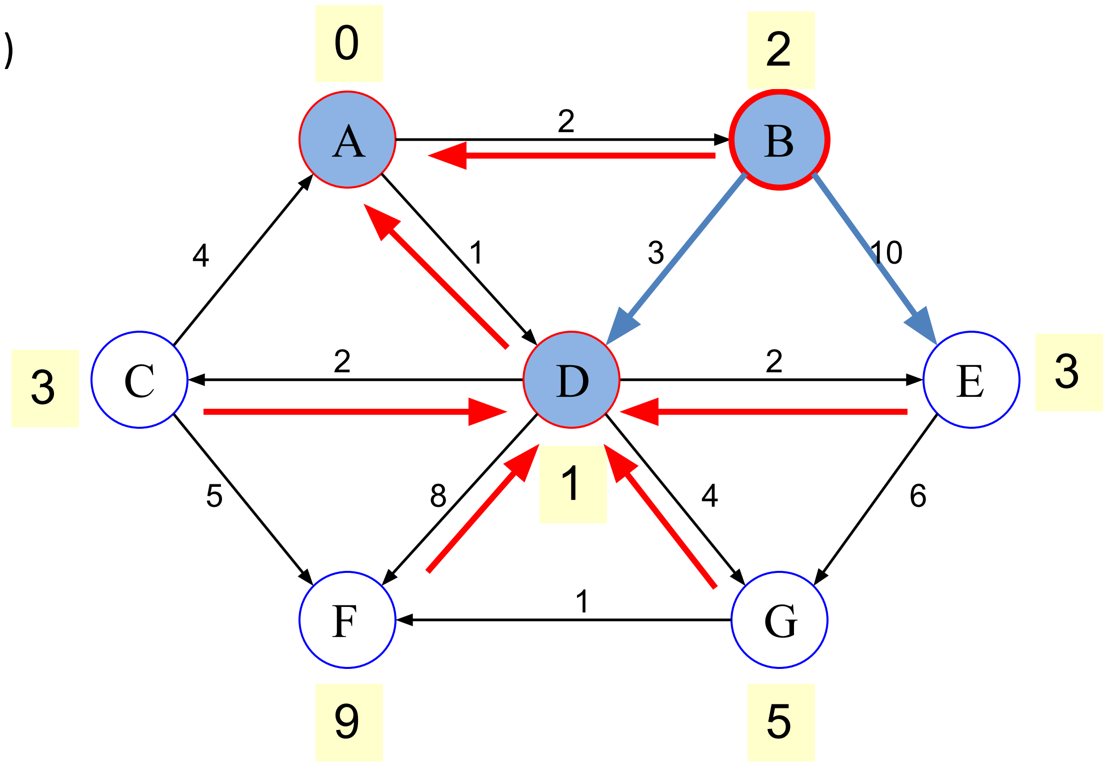
Update D(B), D(D), parent(B), parent(D)

S = S U {D}

Update D(C), D(F), D(G), D(E), parent(…)

S = S U {B}

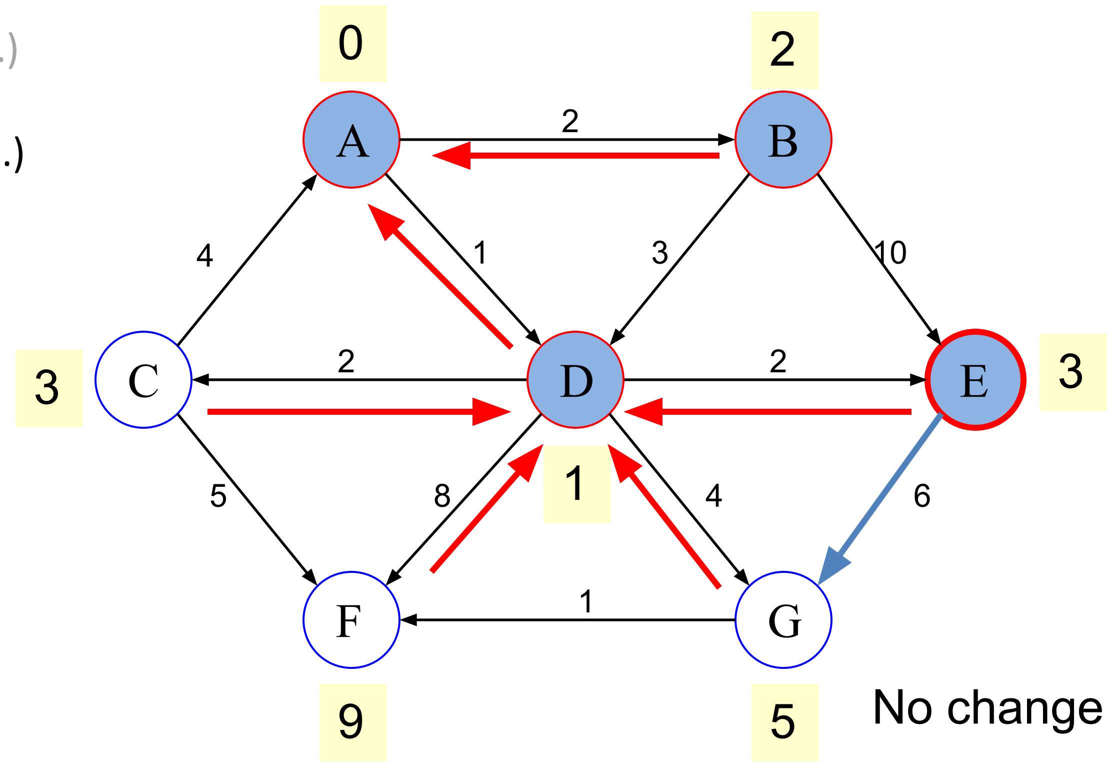
Update D(E), parent(…)



No change

Single-source-all-destination routing: Dijkstra's algorithm

S = {A}
D(A) = 0
Update D(B), D(D), parent(B), parent(D)
S = S U {D}
Update D(C), D(F), D(G), D(E), parent(…)
S = S U {B}
Update D(E), parent(…)
S = S U {E}
Update D(G), parent(…)



Single-source-all-destination routing: Dijkstra’s algorithm

S = {A}

D(A) = 0

Update D(B), D(D), parent(B), parent(D)

S = S U {D}

Update D(C), D(F), D(G), D(E), parent(…)

S = S U {B}

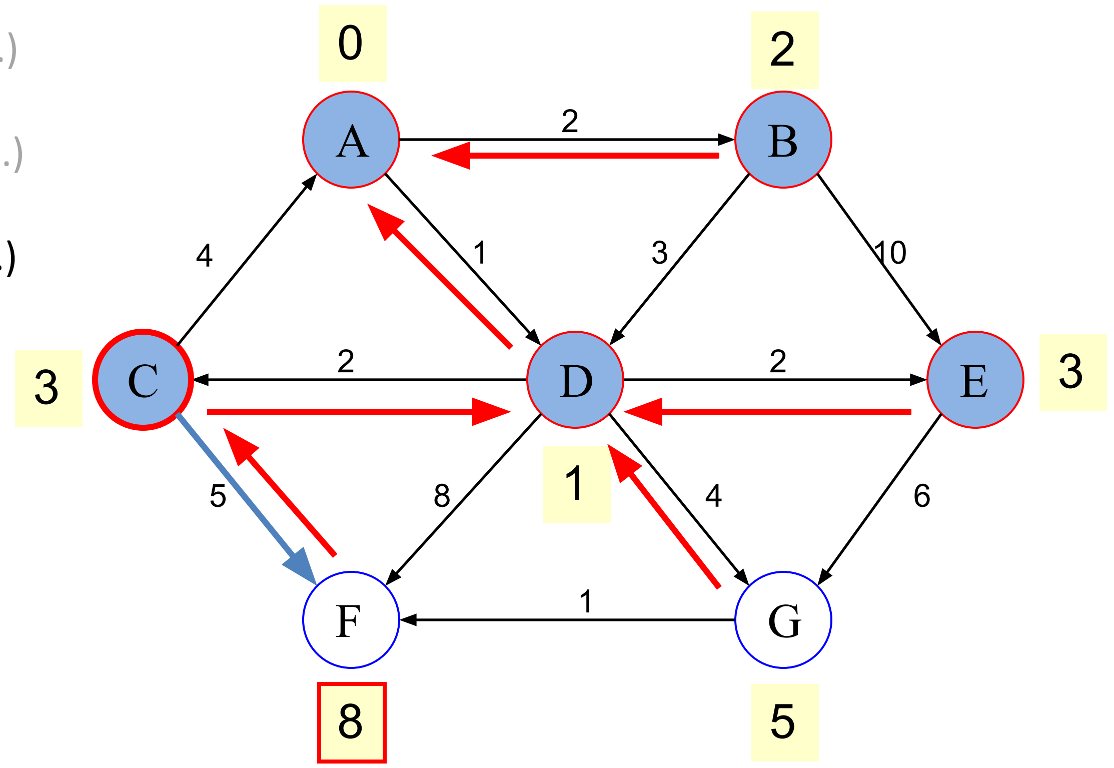
Update D(E), parent(…)

S = S U {E}

Update D(G), parent(…)

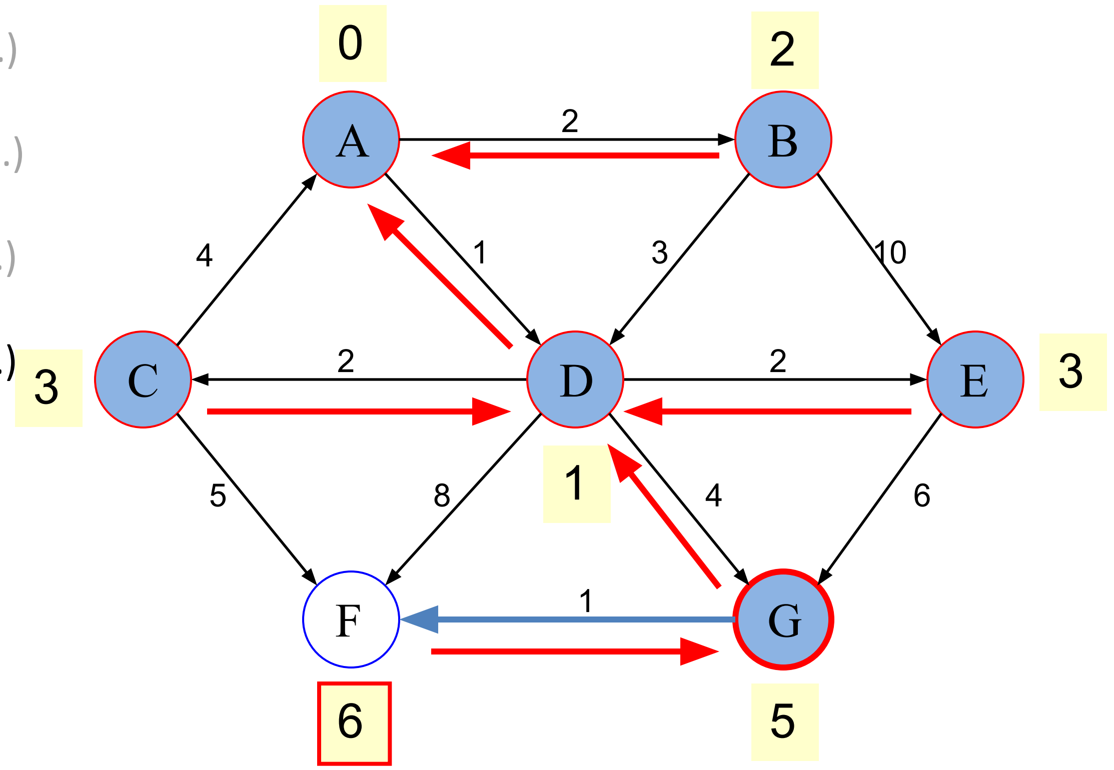
S = S U {C}

Update D(F), parent(…)



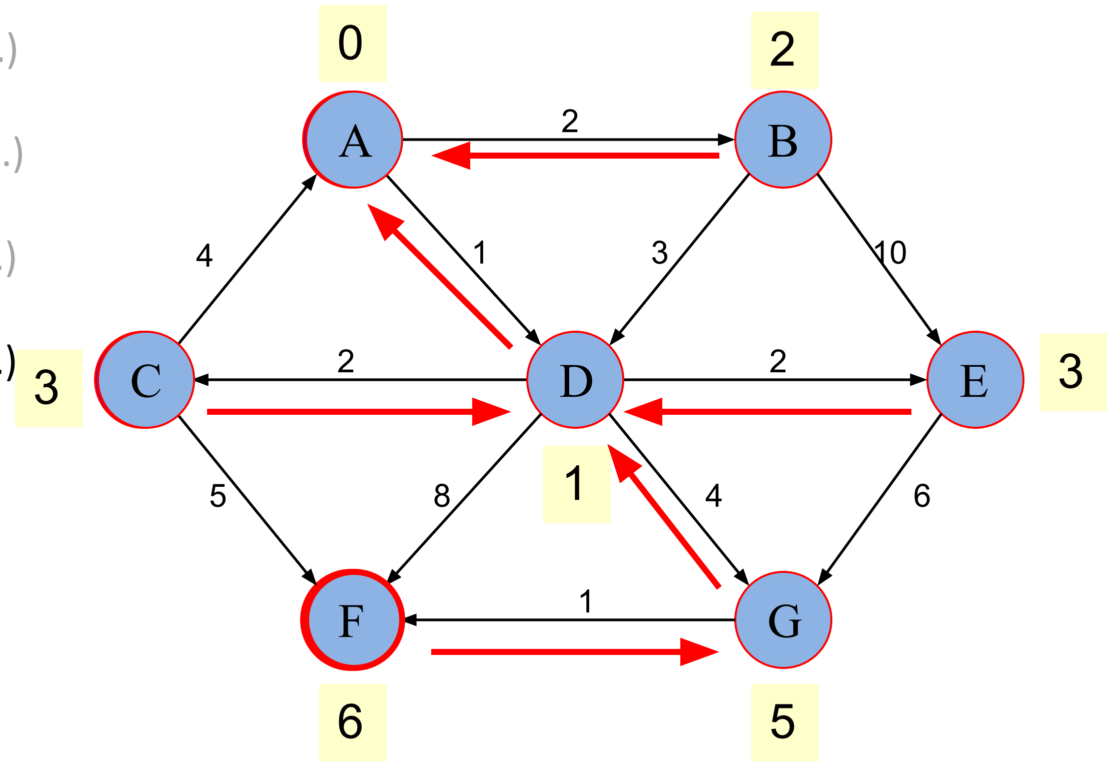
Single-source-all-destination routing: Dijkstra's algorithm

S = {A}
D(A) = 0
Update D(B), D(D), parent(B), parent(D)
S = S U {D}
Update D(C), D(F), D(G), D(E), parent(…)
S = S U {B}
Update D(E), parent(…)
S = S U {E}
Update D(G), parent(…)
S = S U {C}
Update D(F), parent(…)
S = S U {G}
Update D(F), parent(…)



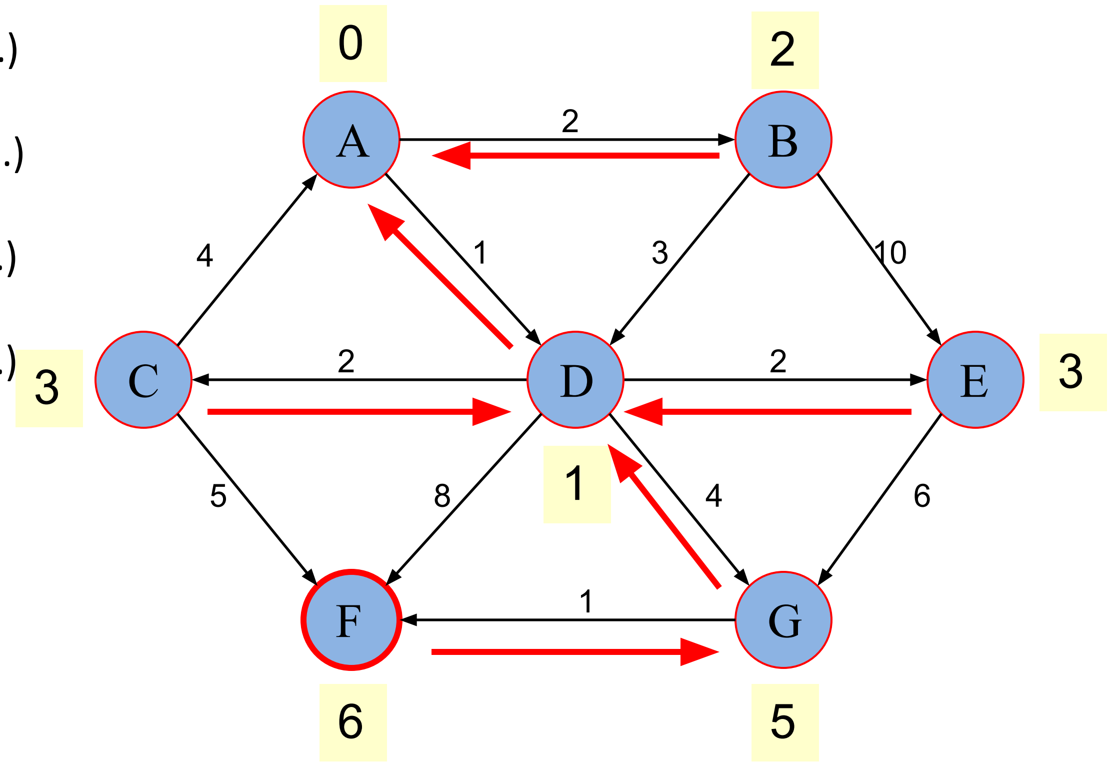
Single-source-all-destination routing: Dijkstra's algorithm

S = {A}
D(A) = 0
Update D(B), D(D), parent(B), parent(D)
S = S U {D}
Update D(C), D(F), D(G), D(E), parent(…)
S = S U {B}
Update D(E), parent(…)
S = S U {E}
Update D(G), parent(…)
S = S U {C}
Update D(F), parent(…)
S = S U {G}
Update D(F), parent(…)
S = S U {F}
Update ----



Single-source-all-destination routing: Dijkstra's algorithm

S = {A}
D(A) = 0
Update D(B), D(D), parent(B), parent(D)
S = S U {D}
Update D(C), D(F), D(G), D(E), parent(…)
S = S U {B}
Update D(E), parent(…)
S = S U {E}
Update D(G), parent(…)
S = S U {C}
Update D(F), parent(…)
S = S U {G}
Update D(F), parent(…)
S = S U {F}
Update ----



Single-source-all-destination routing: Dijkstra’s algorithm

Dijkstra’s algorithm: Running time: O(m log n)

```
Initialize $S = \emptyset, D[s] = 0, D[u] = \infty, u \neq s$          ← O(1)

for i = 1, …, n                                                          ← O(n)
    Let u* be the vertex with min $D[u]$                                 ← O(n log n)
    Add u* to S                                                          ← O(1)
    For every $u \notin S, (u^*, u) \in E$                               ┐
        $D[u] = \min\big(D[u], D[u^*] + w(u^*, u)\big)$                  ├ ← O(m log n)
        parent(u) = u* if D[u] was updated                               ┘
```

Q&A

Single-source-all-destination routing: Dijkstra’s algorithm

- Dijkstra’s algorithm:

```
Initialize $S_1 = \{s\}, \delta(s) = 0$
for i = 1, 2, …, n-1
    for every u $\notin S_i$, $D_i[u] = \min_{v \in S_i}(\delta(v) + w(v,u))$
    Let u* be the vertex with min $D_i[u]$
    Set $\delta(u^*) = D_i[u^*]$ and $S_{i+1} = S_i \cup \{u^*\}$
```

Vertex u is presently estimated to be at most $D_i[u]$ away

Single-source-all-destination routing: Dijkstra’s algorithm

- Dijkstra’s algorithm:

```
Initialize $S_1 = \{s\}, \delta(s) = 0.$
For I = 1, 2, …, n-1
    For every u $\notin$ $S_i$, $D_i[u] = \min_{v \in S_i}(\delta(v) + w(v,u))$
    Let u* be the vertex with min $D_i[u]$
    Set $\delta(u^*) = D_i[u^*]$ and $S_{i+1} = S_i \cup \{u^*\}$
```

$$D_i[u] = \min(D_{i-1}[u], \delta(u_i) + wt(v,u))$$

Recognize that in i-th iteration u_i has been added. Improved algorithm since change in path length can only happen because u_i has since been added