Greedy Algorithms: Dijkstra's Algorithm

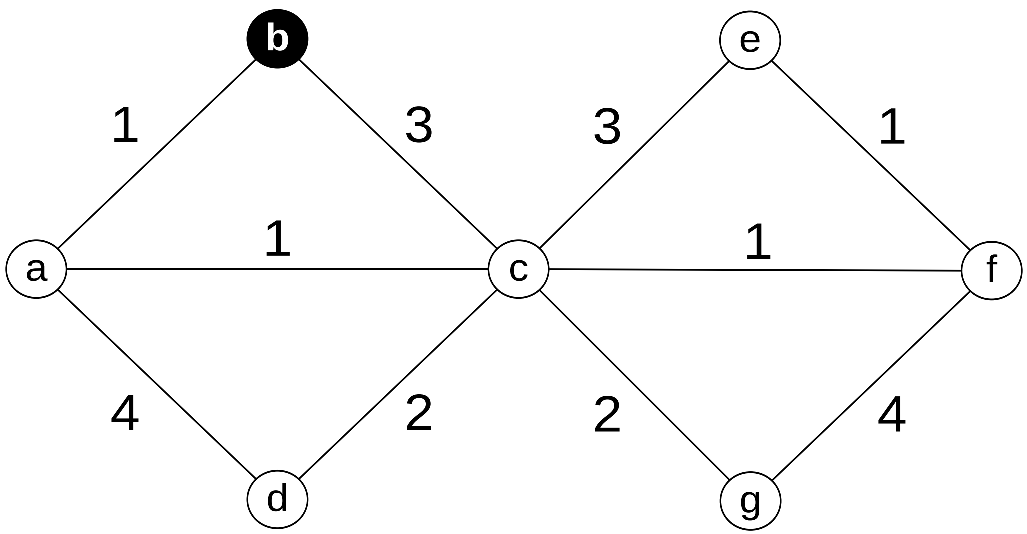
Textbook: Chapter 9.3

Context

- This is one of several “greedy algorithms” we will examine:
  - Minimum Spanning Tree of a graph
    - Prim’s algorithm
    - Kruskal’s algorithm
  - Shortest Paths from a Single Source in a graph
    - Dijkstra’s algorithm
  - Graph coloring

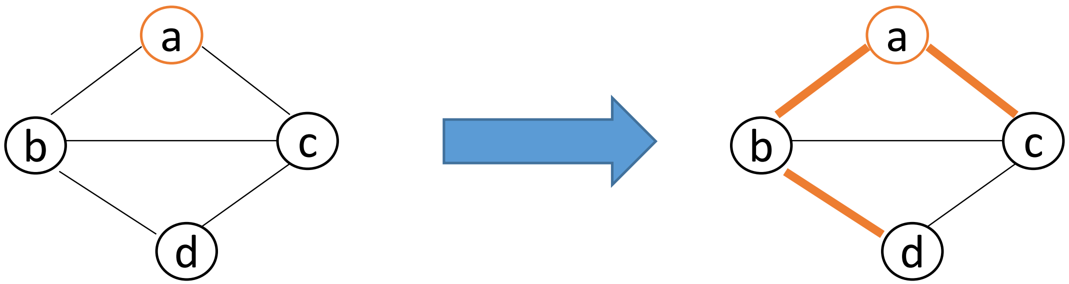
Problem:
Single-source Shortest Paths

- Find the shortest path from a chosen vertex (the *source*) to every other vertex

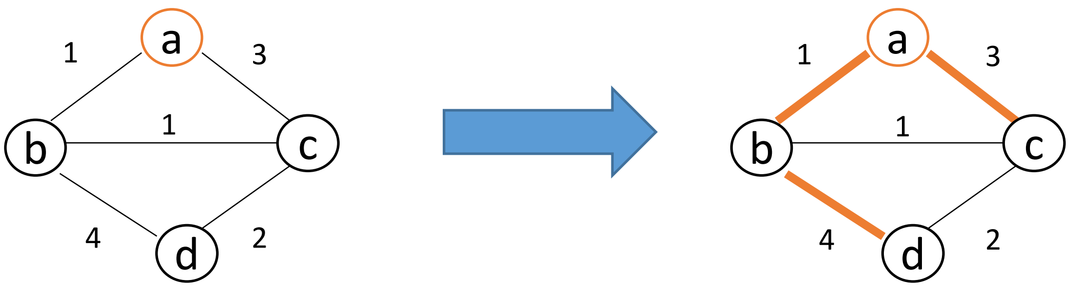


What about BFS?

- Simple/basic BFS already does this for an unweighted graph:



- … but not for weighted graphs. Consider the distance between a and d:



a-b-d has length 5 in BFS tree, but shortest path is 4 (a-b-c-d)

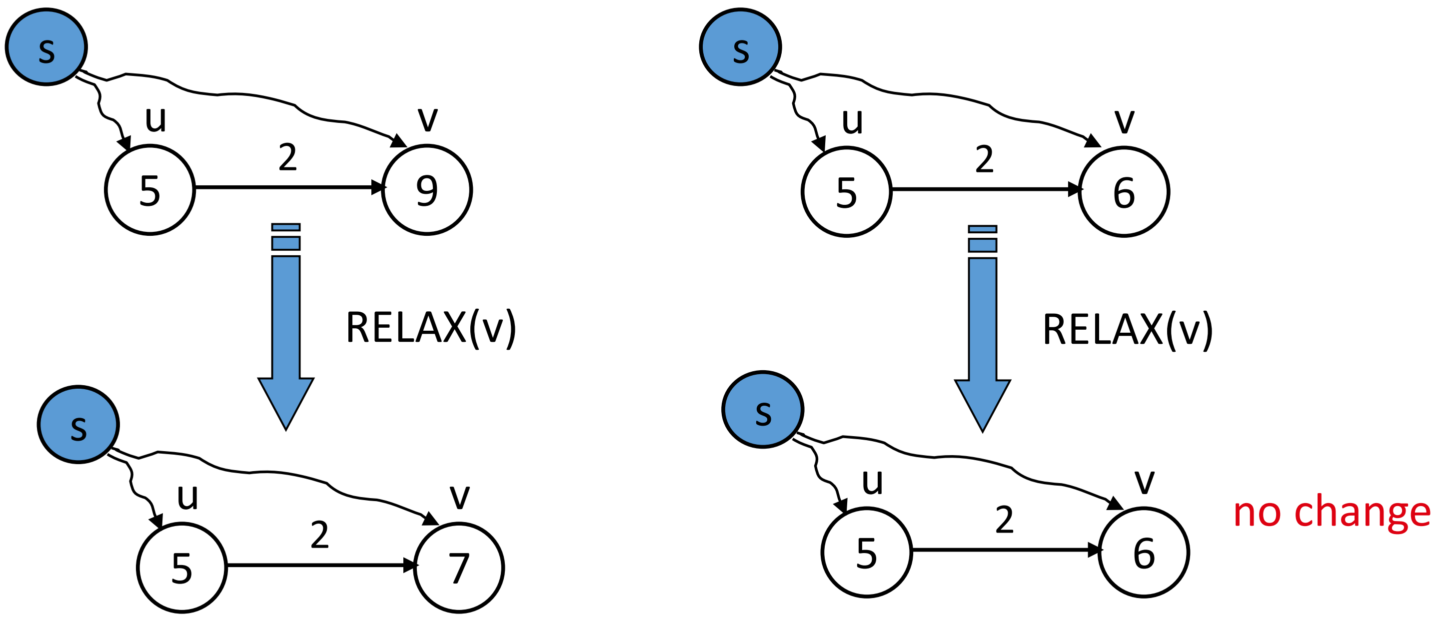
- *Algorithm to find shortest paths in weighted graphs needs to consider the weight on the edge before including it in the solution*

Idea of Dijkstra’s algorithm

- Remember the best-known shortest distances for all vertices
  - Initially “infinity” for all
- Choose the nearest unprocessed vertex
  - Definition of “nearest” tbd
- Look at all of its neighbors
- Update their known shortest distances (“Relax”)
- Repeat

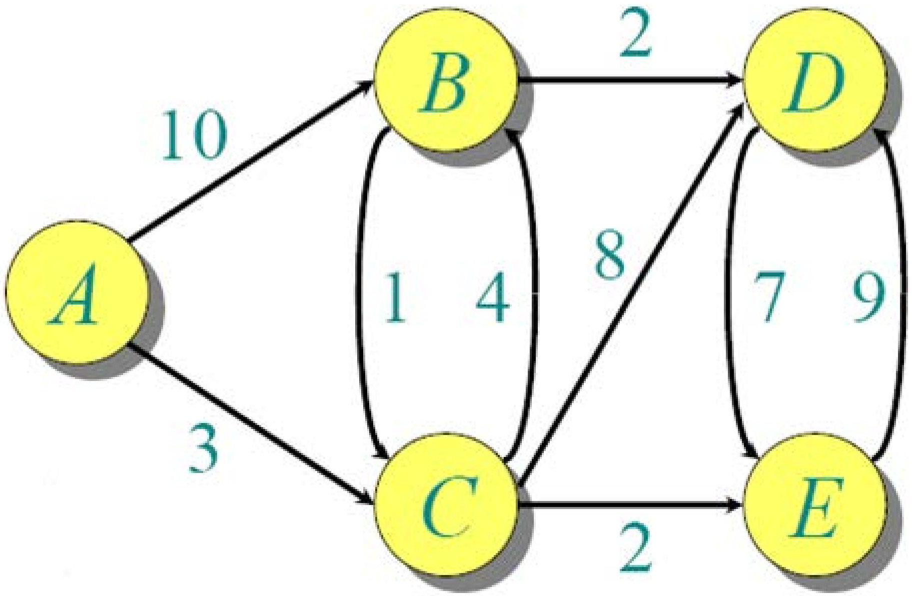
Relaxation

- Dijkstra refers to "relaxing" a vertex
- Meaning: update the best known shortest path to v



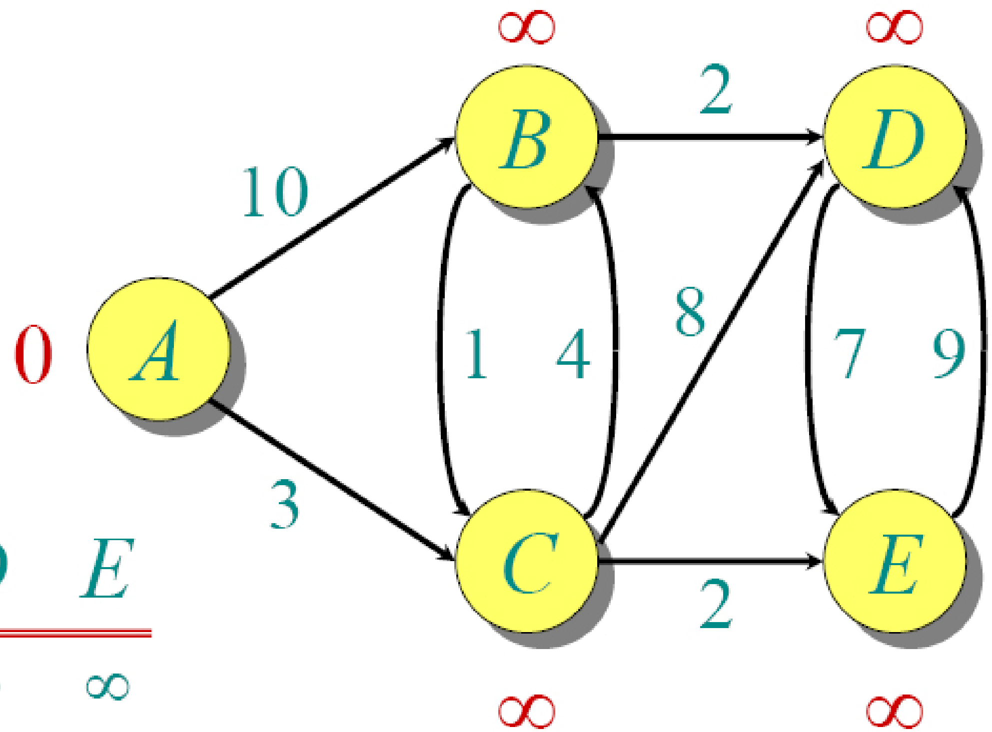
Dijkstra Example

Find the shortest paths from A to all other vertices



Dijkstra Example

Initialize:

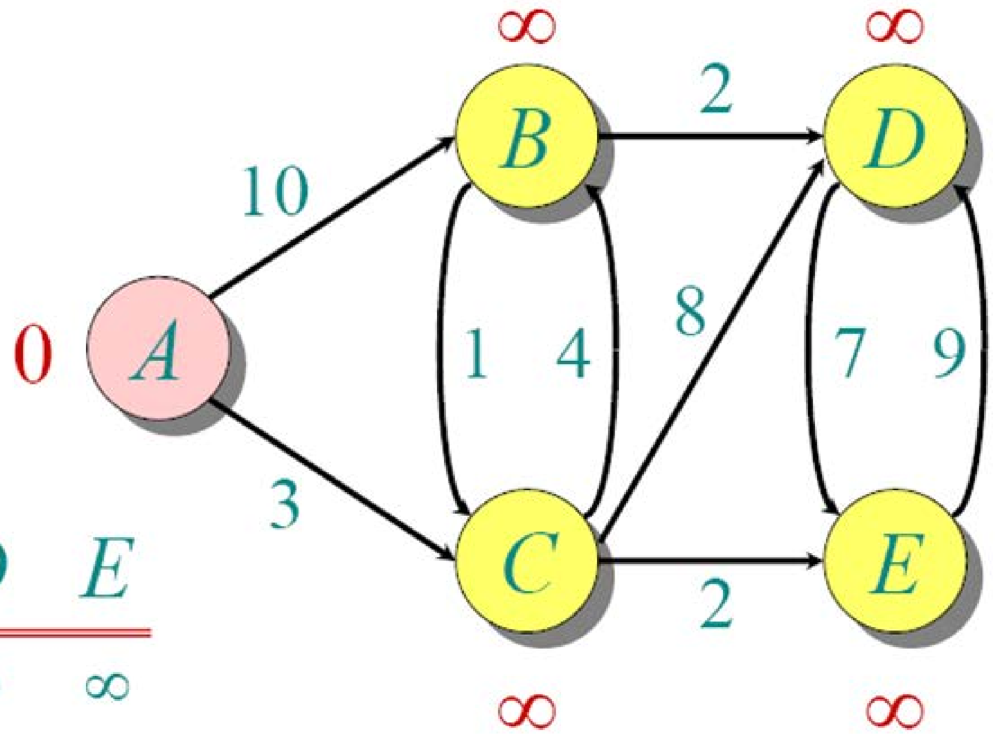


(dist)

Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Dijkstra Example

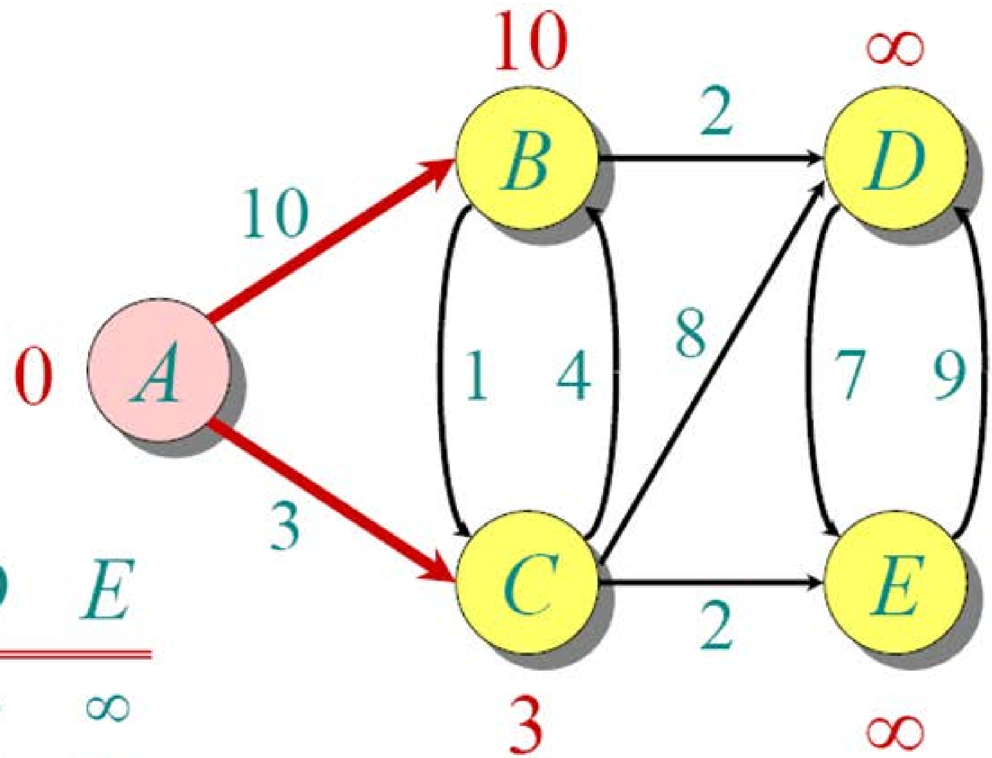


Q:	A	B	C	D	E
(dist)	0	∞	∞	∞	∞

Dijkstra Example

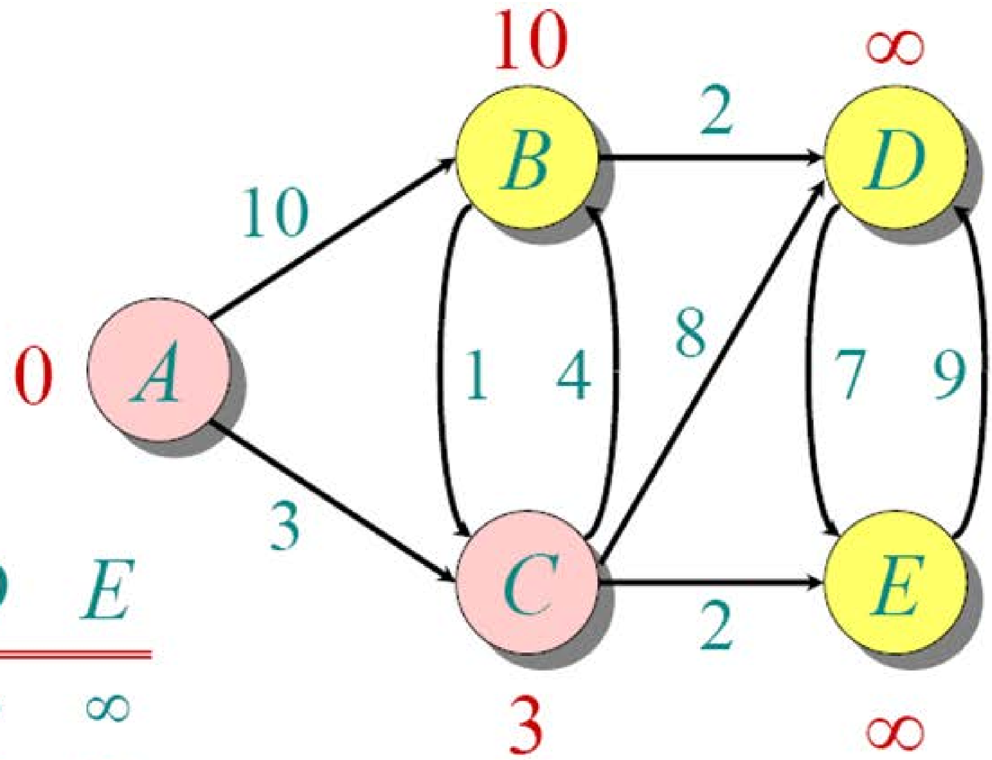
(dist)

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞



S: { A }

Dijkstra Example

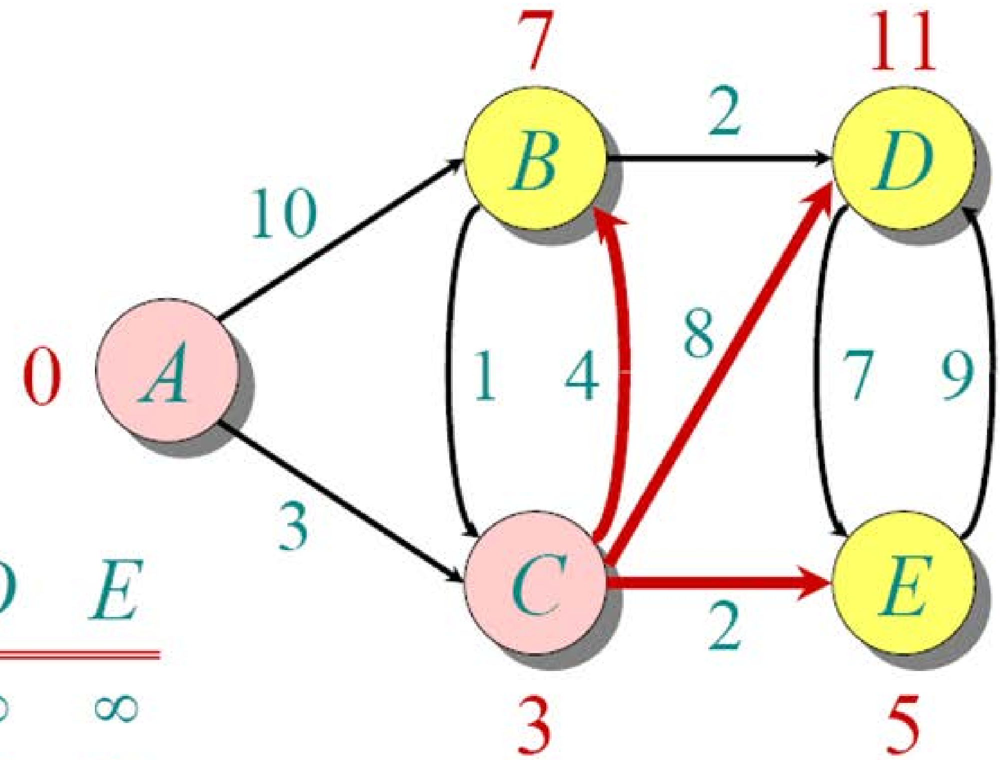


(dist)

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

S: { A, C }

Dijkstra Example

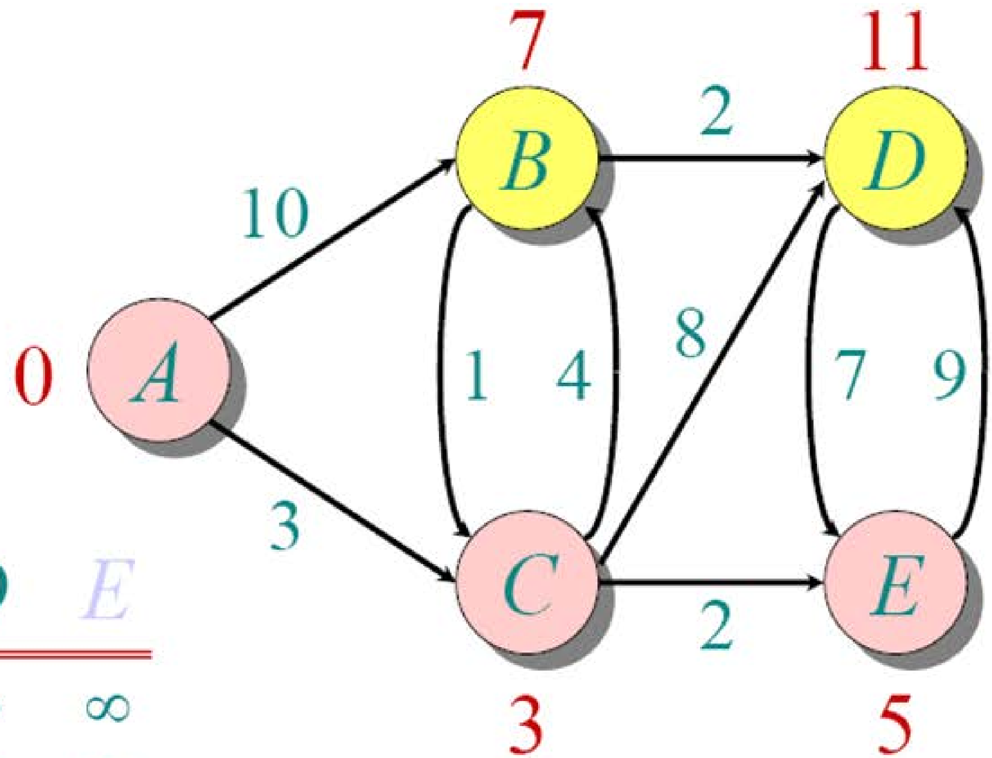


(dist)

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

$S: \{ A, C \}$

Dijkstra Example

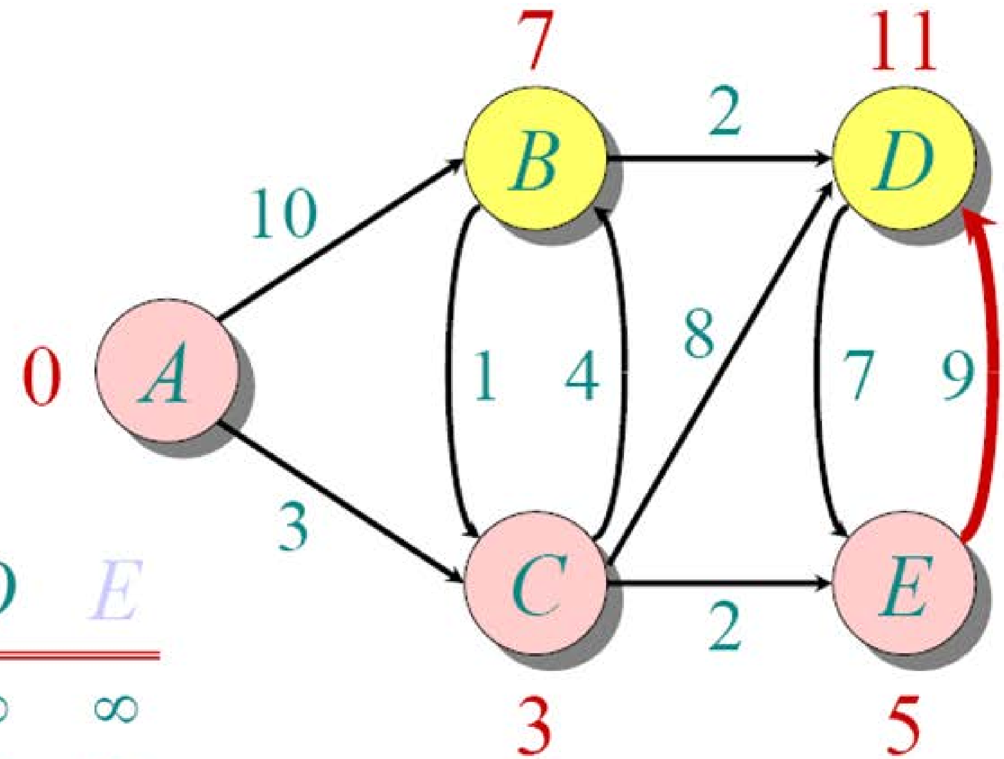


(dist)

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

S: { A, C, E }

Dijkstra Example

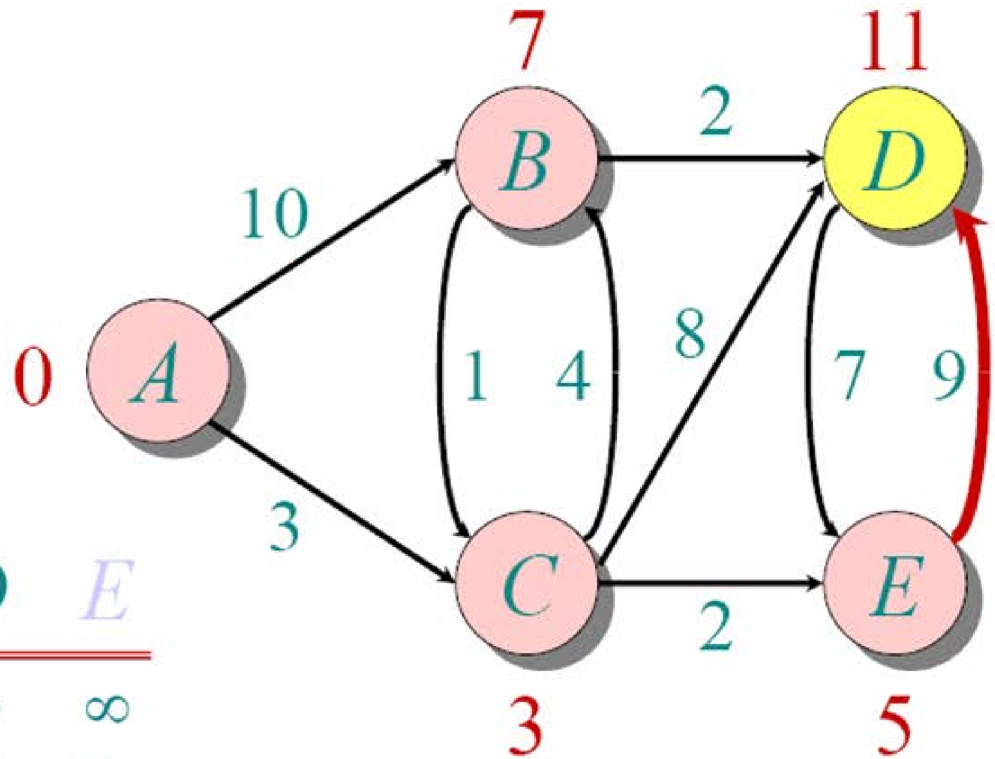


(dist)

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

S: { A, C, E }

Dijkstra Example



(dist)

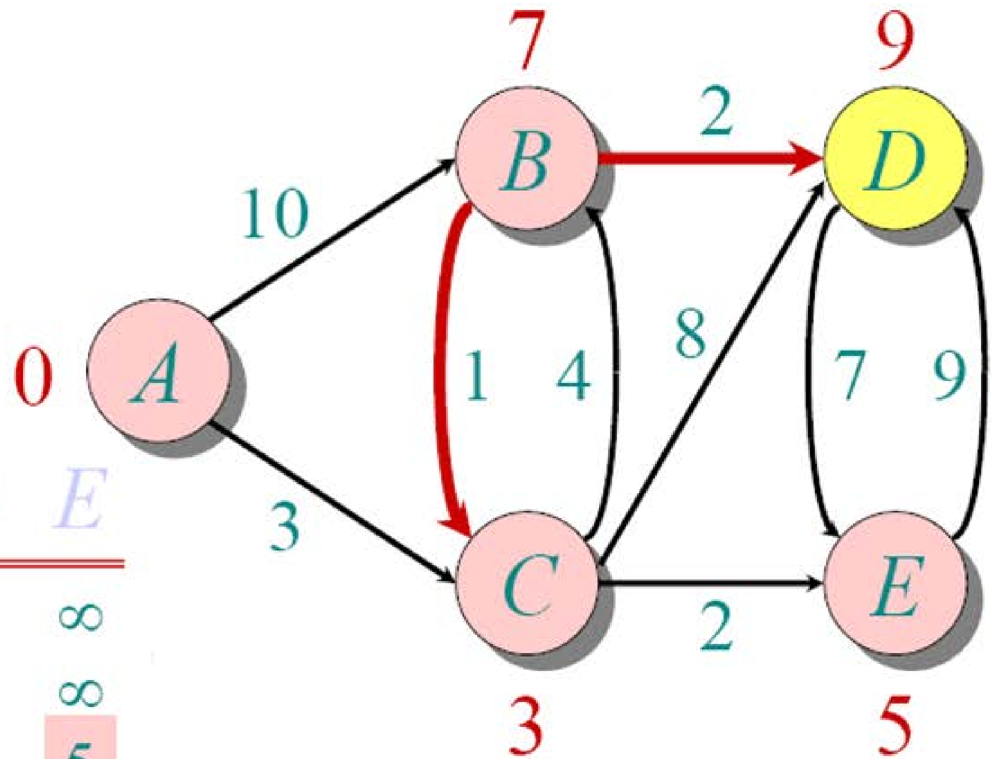
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra Example

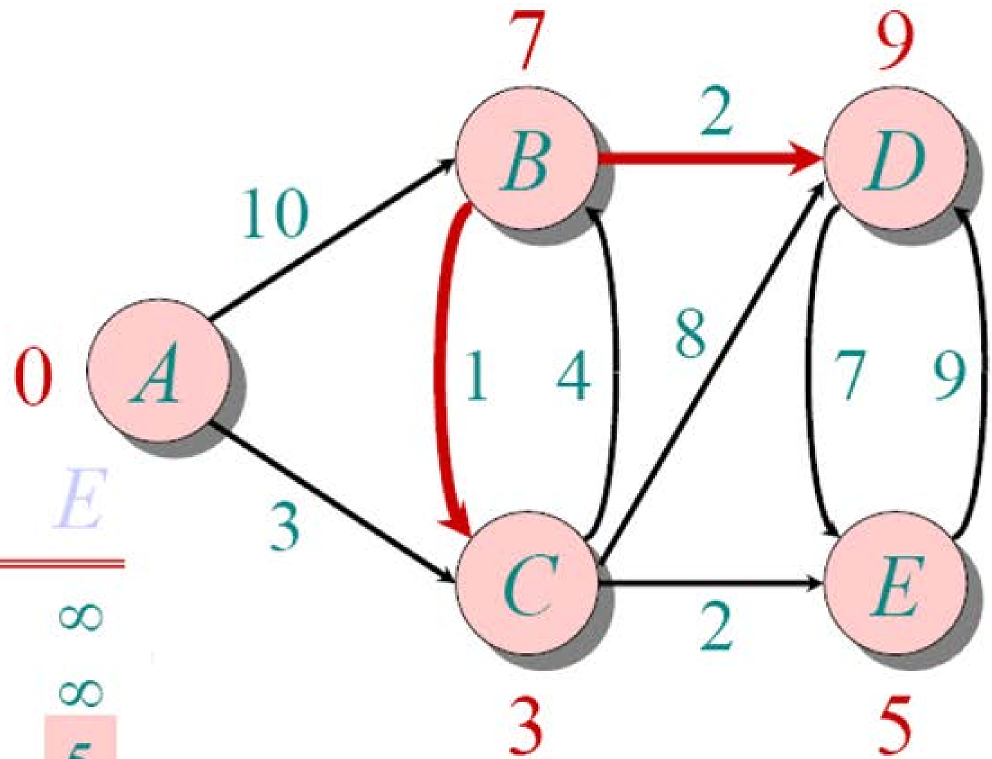
(dist)

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



S: { A, C, E, B }

Dijkstra Example



(dist)

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

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

Dijkstra's Algorithm

- Builds a tree of shortest paths rooted at the starting vertex
- This is a greedy algorithm: it adds the closest vertex, then the next closest, and so on (until all vertices have been added)

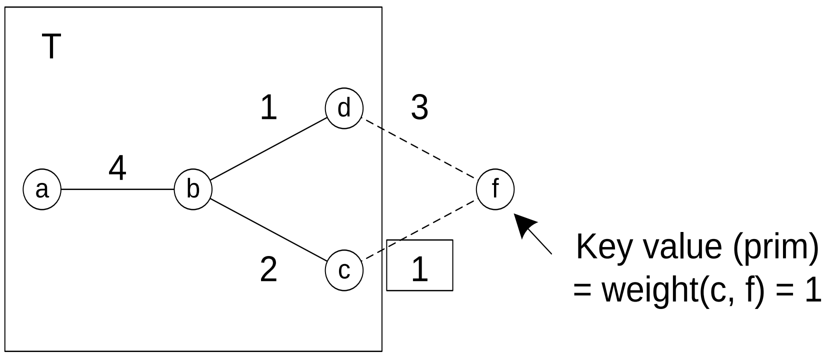
<u>High-level pseudocode:</u>

```
Initialise d and prev
Add all vertices to a PQ with distance from source as the key
While there are still vertices in PQ
    Get next vertex u from the PQ
    For each vertex v adjacent to u
        If v is still in PQ, relax v

Relax(v):
    if d[u] + w(u,v) < d[v]
        d[v] ← d[u] + w(u,v)
        prev[v] ← u
        PQ.updateKey(d[v], v)
```

Similarity of Dijkstra to Prim

- Both accumulate a tree T of edges from G
- Each iteration: select the minimum priority edge adjacent to the tree that has been built so far
- In Prim's the priority of an edge is simply the weight of the edge



- In Dijkstra's the "priority" is the weight of the edge *(u, v)* <u>plus</u> the distance from the start to the parent of *v*

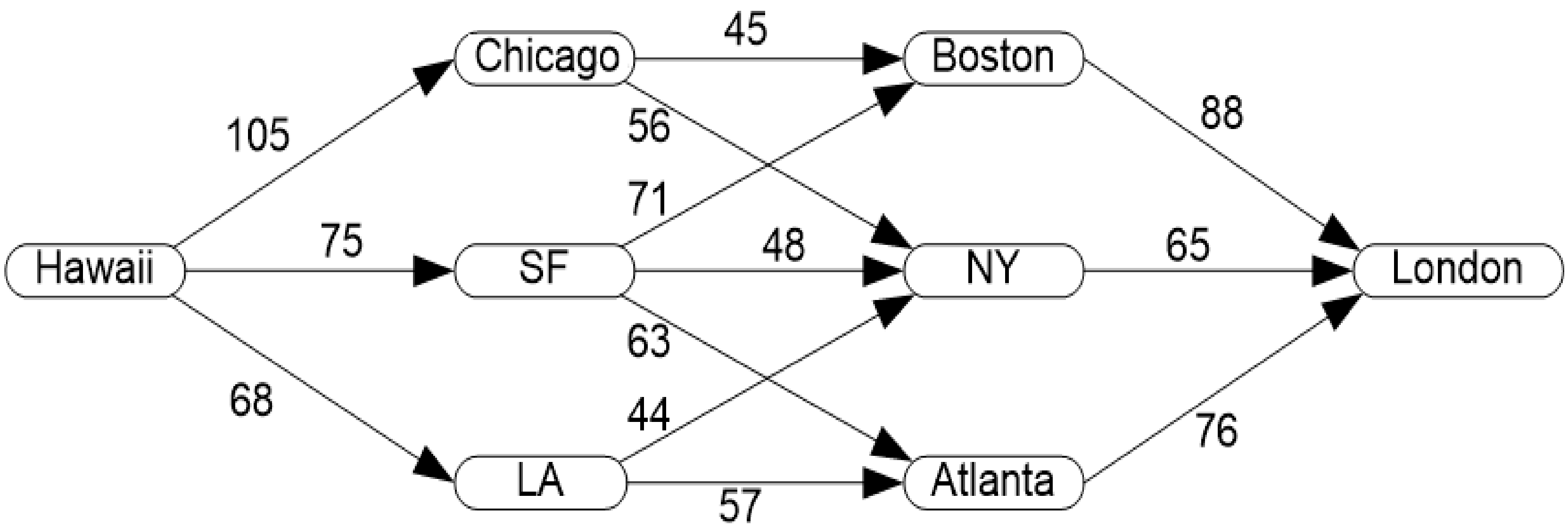
Sample application of Dijkstra’s

- Suppose London wants fresh pineapples from Hawaii.
- There are no direct flights, but many possible connections.
- What is the best possible route to minimize overall shipping cost?

Input: Shipping costs, city to city

- Honolulu to Chicago 105
- Honolulu to San Francisco 75
- Honolulu to Los Angeles 68
- Chicago to Boston 45
- Chicago to New York 56
- San Francisco to Boston 71
- San Francisco to New York 48
- San Francisco to Atlanta 63
- Los Angeles to New York 44
- Los Angeles to Atlanta 57
- Boston to London 88
- New York to London 65
- Atlanta to London 76

Graph model of the problem



Apply Dijkstra’s algorithm to find the cheapest cost from Hawaii to London
(bonus: cheapest cost to all the other cities, too)

Dijkstra limitation: negative weight edges

- Dijkstra's algorithm doesn't work with negative weight edges
- If we added a new edge to T, and it had a negative weight, then there could exist a shorter path (through this new vertex) to vertices already in T

- For example, consider graph A below.
  - Graph B is the result of running Dijkstra's algorithm on A.
  - But clearly there exists a path such as a-c-e in graph C that is shorter than the path found in B. Therefore Dijkstra's algorithm did not work on this graph that has a negative edge weight.