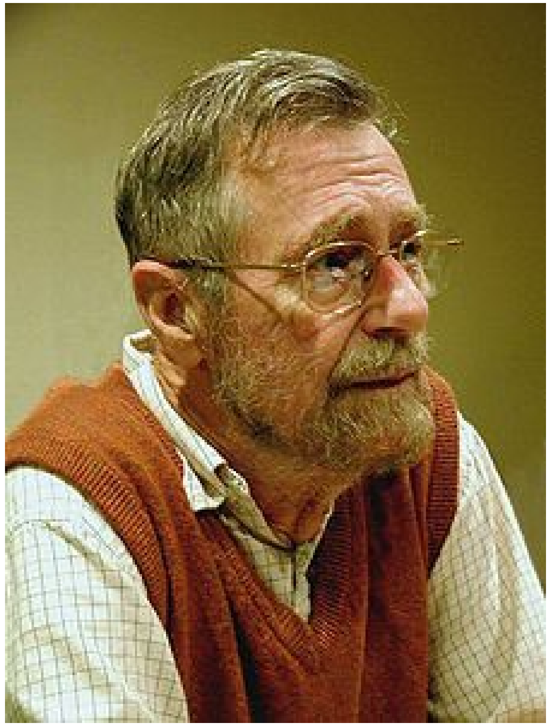
Shortest Path Algorithms

DIJKSTRA'S ALGORITHM

By Laksman Veeravagu and Luis Barrera

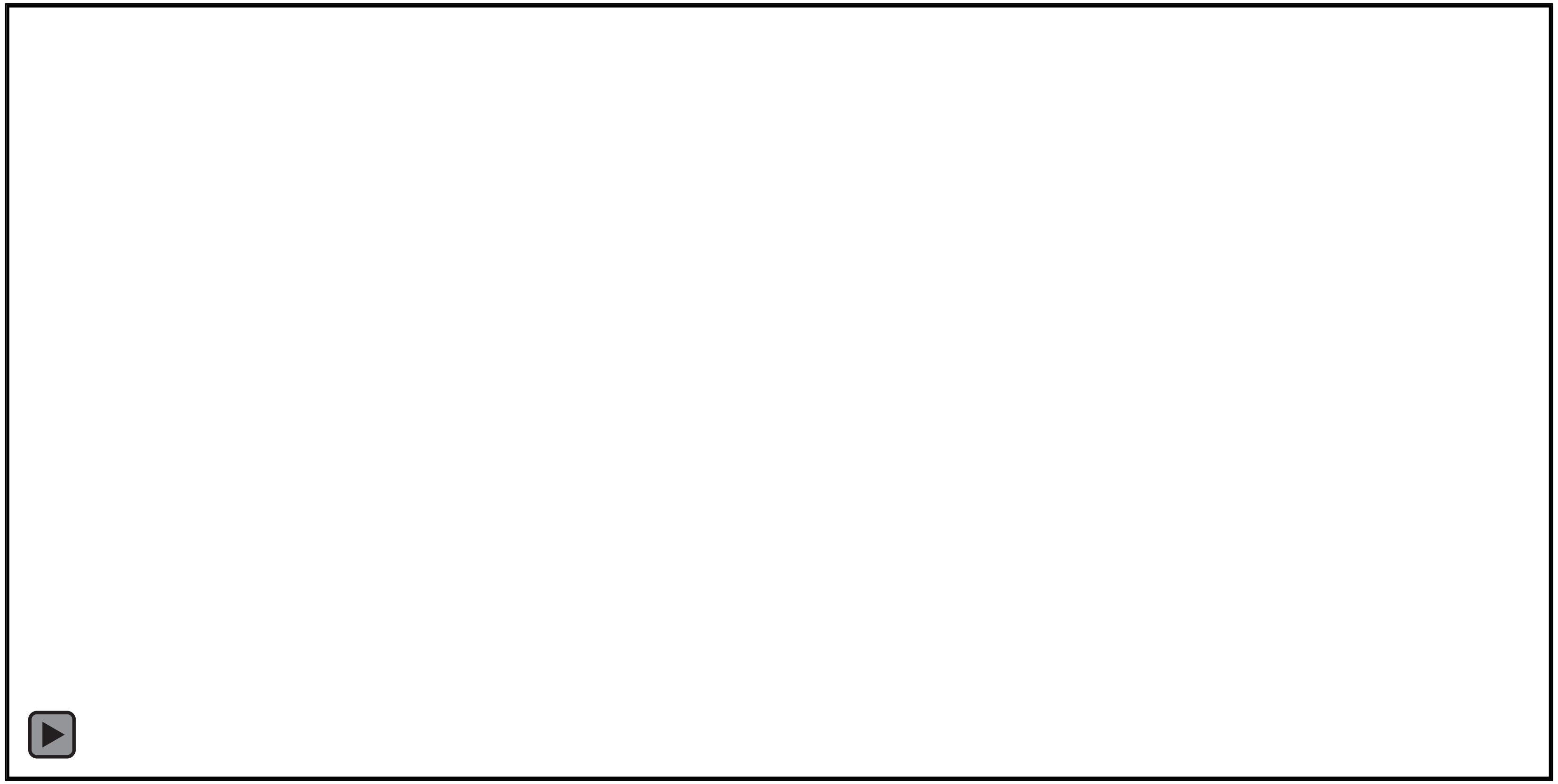
The author: Edsger Wybe Dijkstra



"Computer Science is no more about computers than astronomy is about telescopes."

http://www.cs.utexas.edu/~EWD/

A* VISUALIZATION



Visualization of A*

Vis Credit : https://qiao.github.io/PathFinding.js/visual/

Edsger Wybe Dijkstra

- May 11, 1930 – August 6, 2002

- Received the 1972 A. M. Turing Award, widely considered the most prestigious award in computer science.

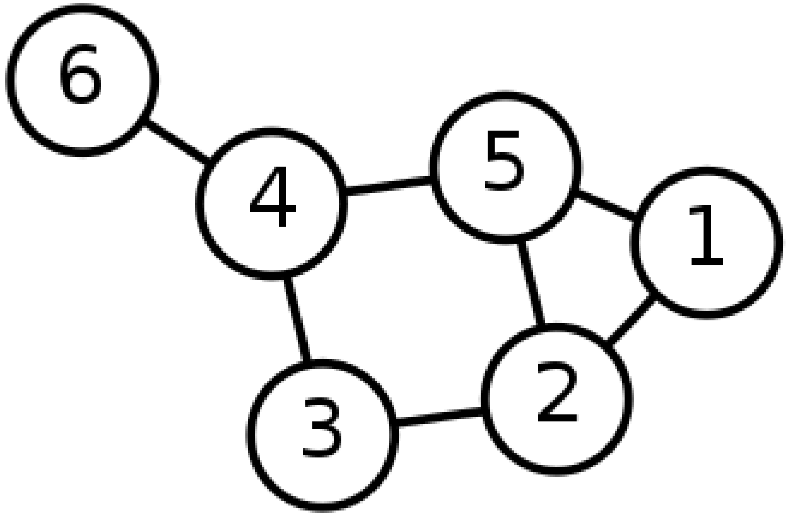
- The Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until 2000

- Made a strong case against use of the GOTO statement in programming languages and helped lead to its deprecation.

- Known for his many essays on programming.

Single-Source Shortest Path Problem

<u>Single-Source Shortest Path Problem</u> - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Preliminary Concepts

- Shortest Path (If exists)

$$Shortest\ Path = \ min\{\ w(\pi)|\ paths\ \pi\ s \rightarrow t\}$$

- DAG Relaxation

  This maintains an upper estimate $d(s,v)$ estimates upperbound and then gradually lowered until equal to $\delta(s,v)$

  - If $d(s,v) > d(s,u) + w(s,v)$, then "relax" by changing d(s,v) to $d(s,u) + w(s,v)$

- Triangle Inequality

If δ(u,v) is the shortest path length between u and v,

$$\delta(u,v) \leq \delta(u,x) + \delta(x,v)$$

Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Approach: Greedy

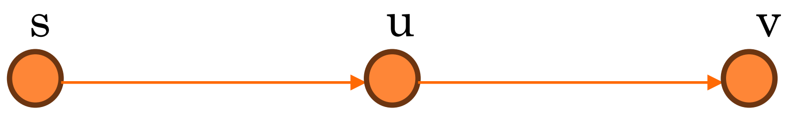
Input: Weighted graph $G=\{E,V\}$ and source vertex $v \in V$, such that all edge weights are nonnegative

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

DIJKSTRA

- If $w \geq 0$, then distance increases along shortest path

$$\delta(s, u) \leq \delta(s, v)$$



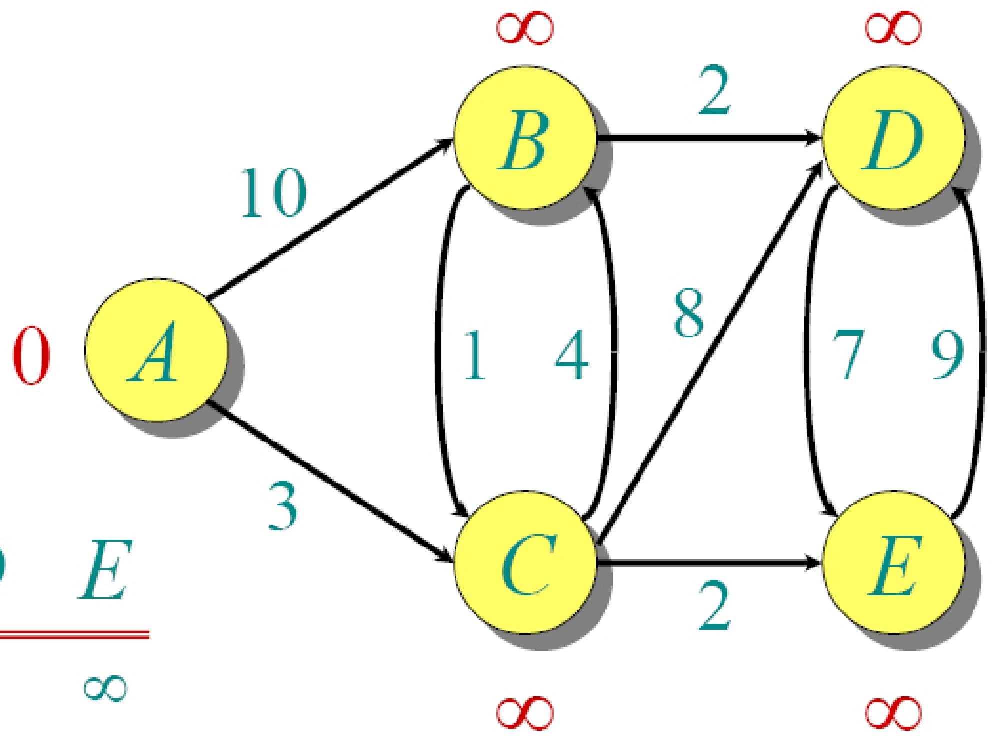
- Relax edges from vertices in increasing order of distance from s
- Find next vertex efficiently using a Data Structure
  - We will use a priority queue over here (priority is the d value)

Dijkstra's algorithm - Pseudocode

```
dist[s] ←o
for  all v ∈ V–{s}
        do  dist[v] ←∞
S←∅
Q←V
while Q ≠∅
do   u ← mindistance(Q,dist)
      S←S∪{u}
       for all v ∈ neighbors[u]
              Relax {u,v,w}
return dist
```

DIJKSTRA ANIMATED EXAMPLE

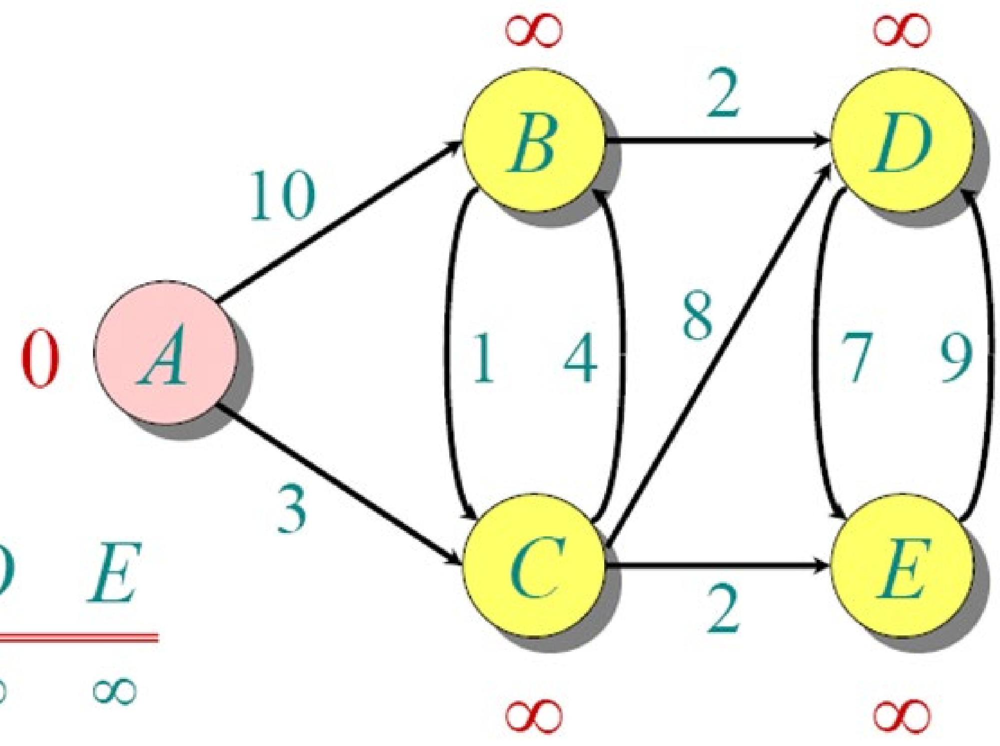
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

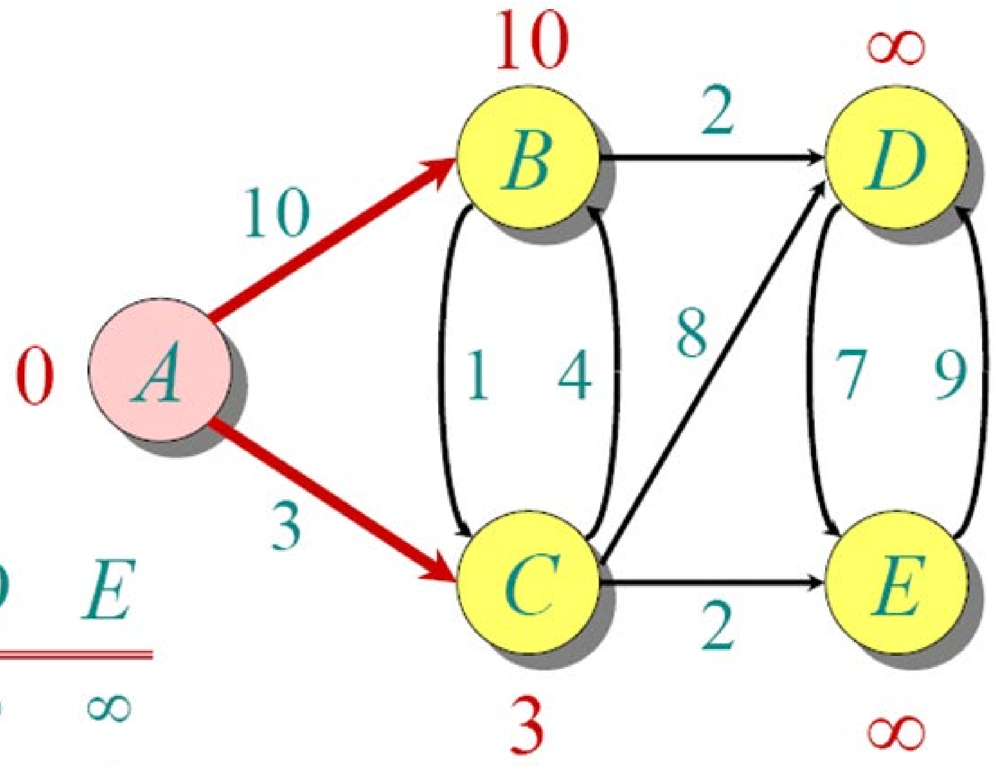
S: {}

Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

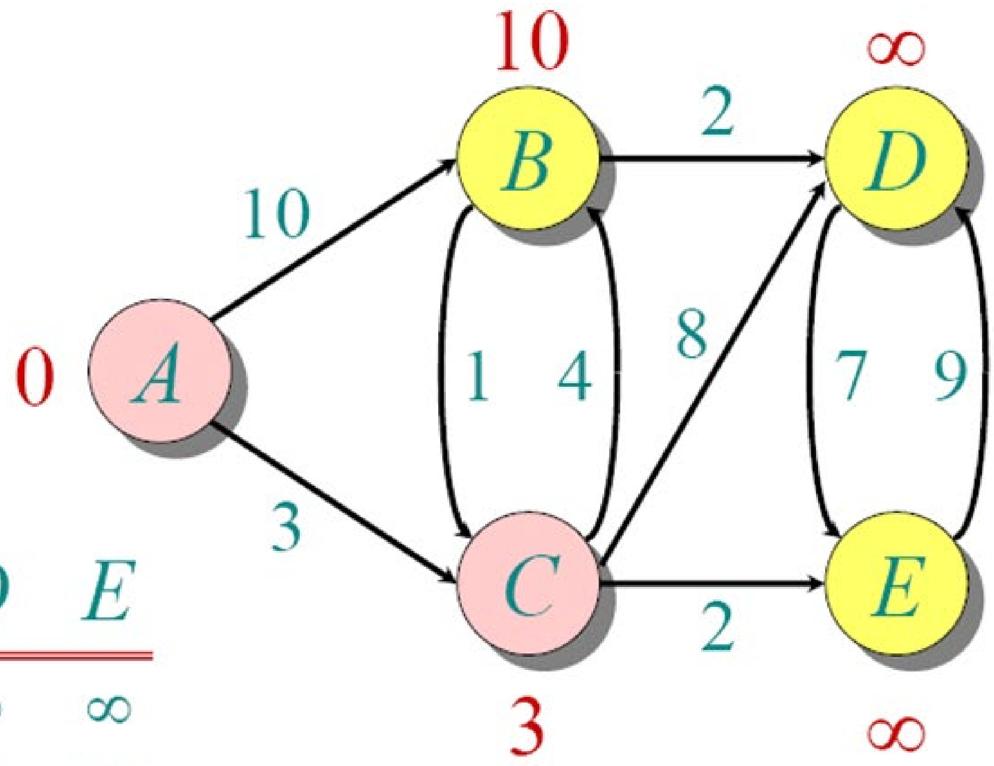
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

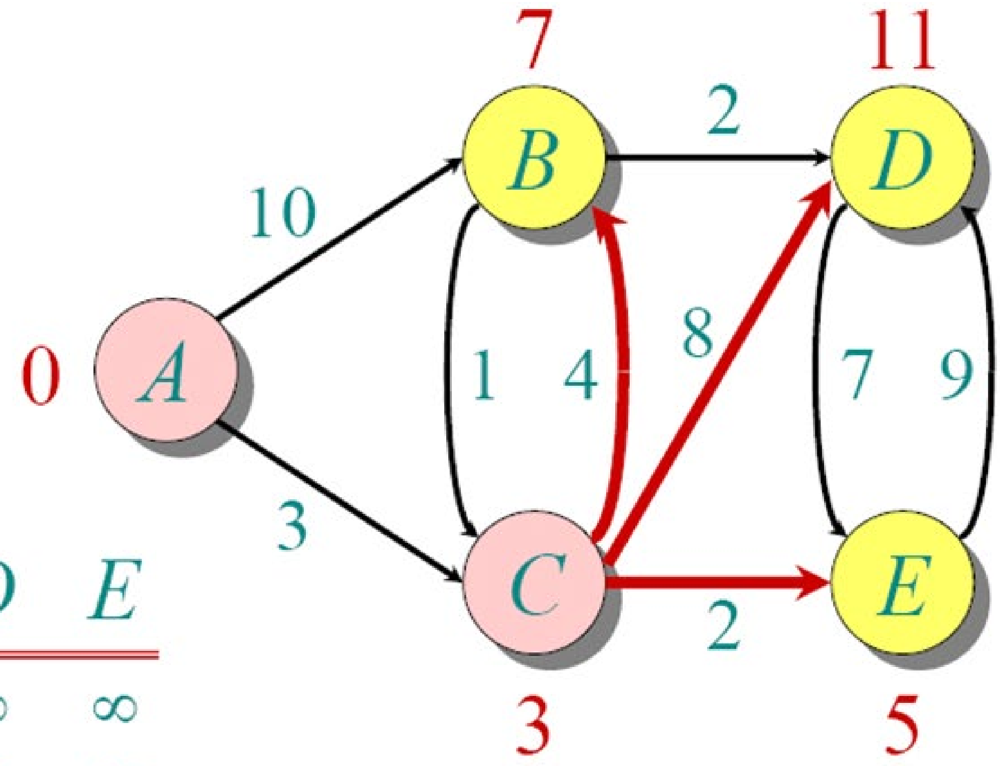
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

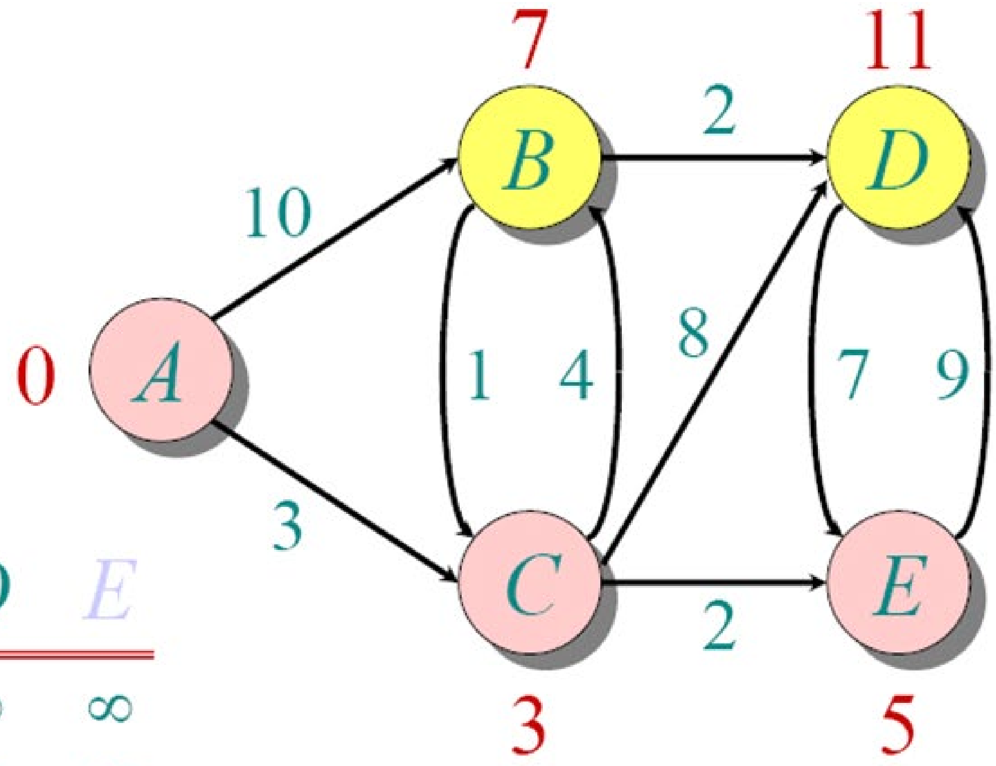
Dijkstra Animated Example



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

S: { A, C }

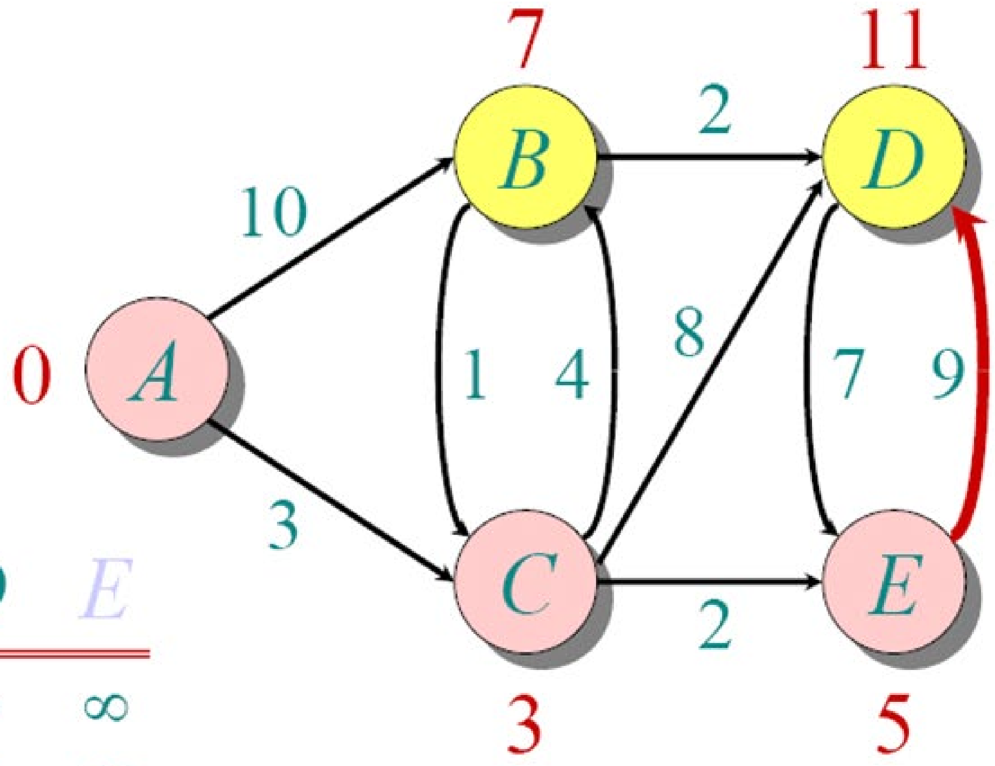
Dijkstra Animated Example



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

S: { A, C, E }

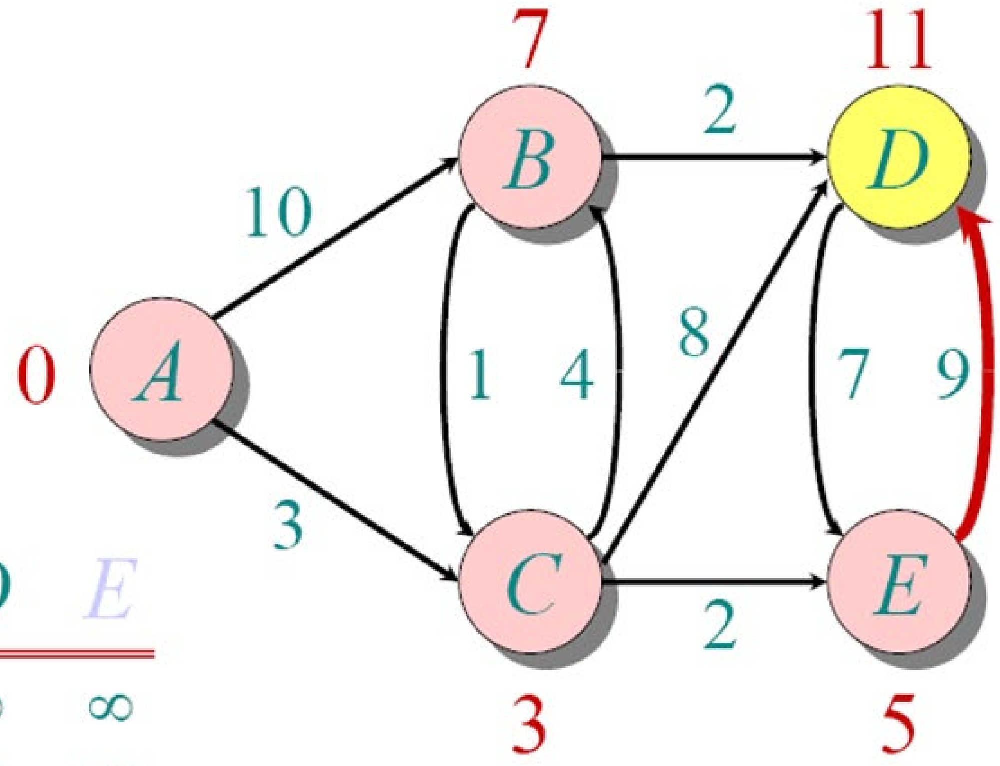
Dijkstra Animated Example



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

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

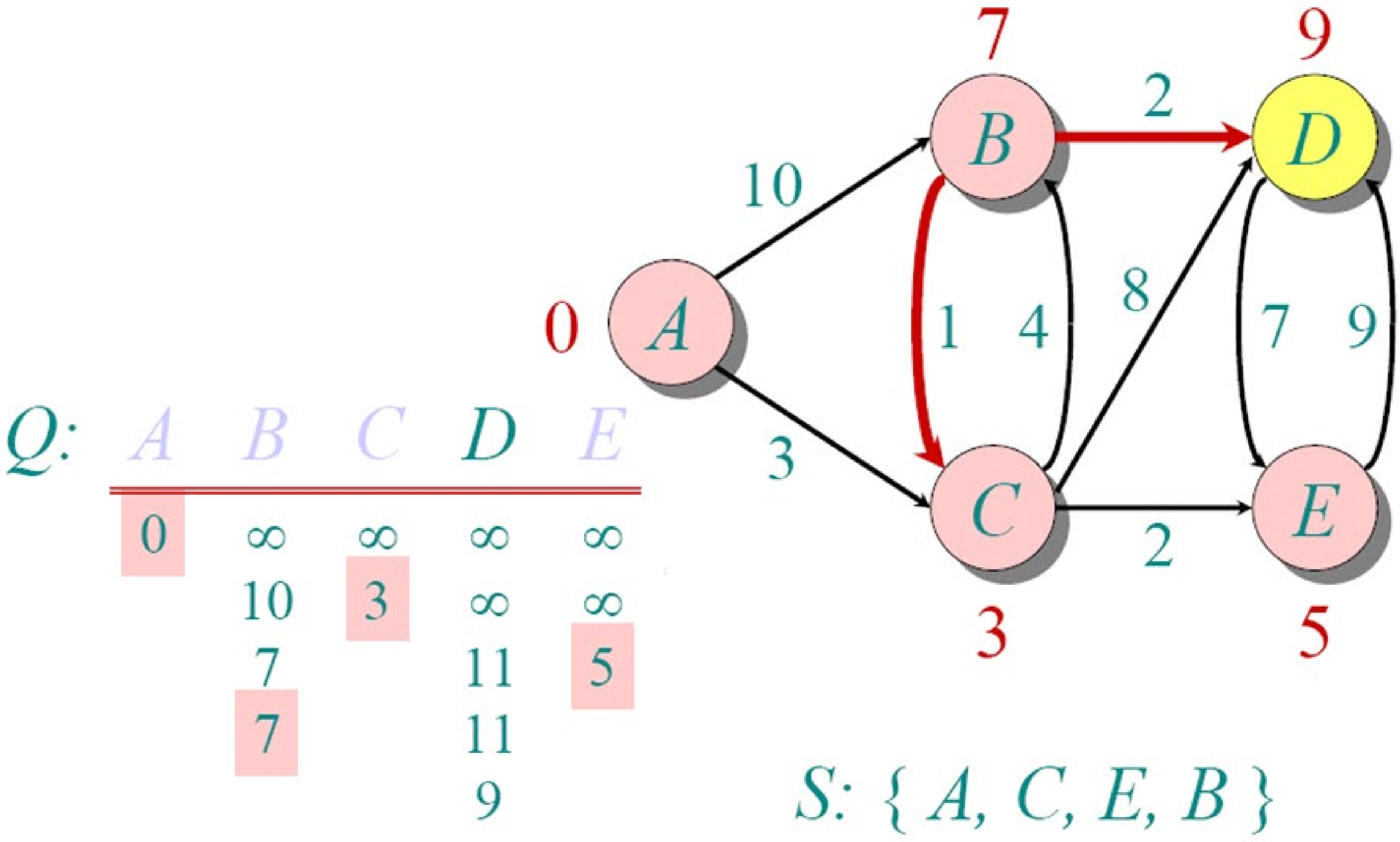
Dijkstra Animated Example



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

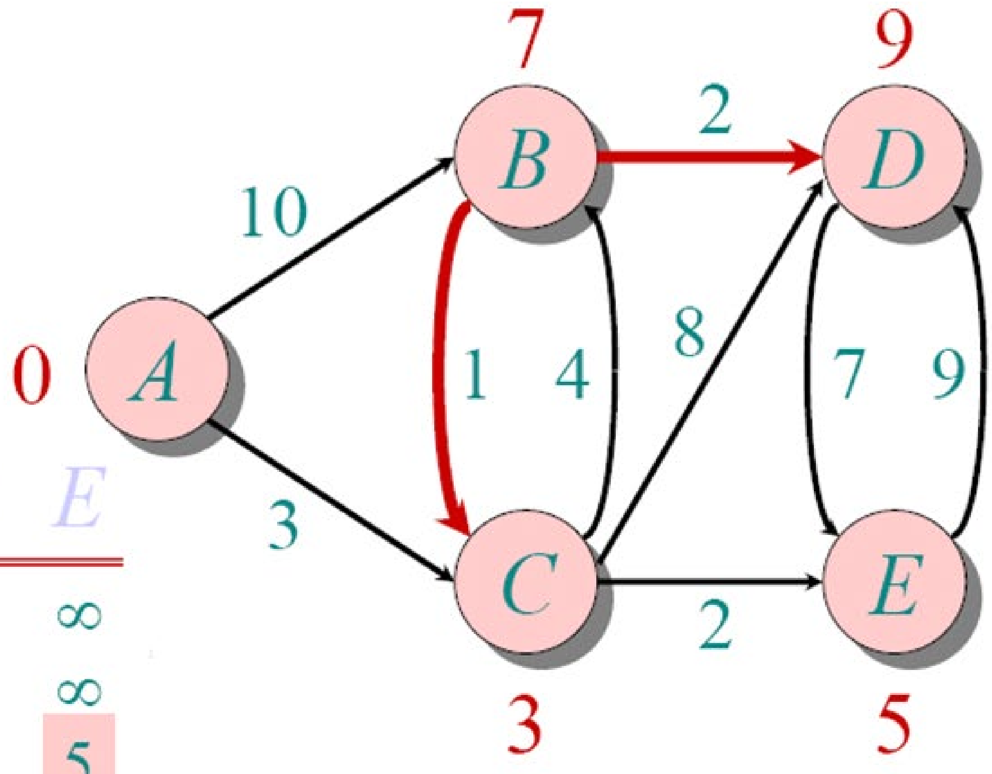
S: { A, C, E, B }

DIJKSTRA ANIMATED EXAMPLE



Dijkstra Animated Example

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



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

Dijkstra's Algorithm - Correctness

- Because of time constraint we will not go deep into correctness proof
- The algorithm's correctness can be proved via induction
- Hints:
  - Optimal Substructure Property
    - Optimal solution can be constructed from optimal solutions of its subproblems
  - Relaxation is safe
  - Non-negative weights

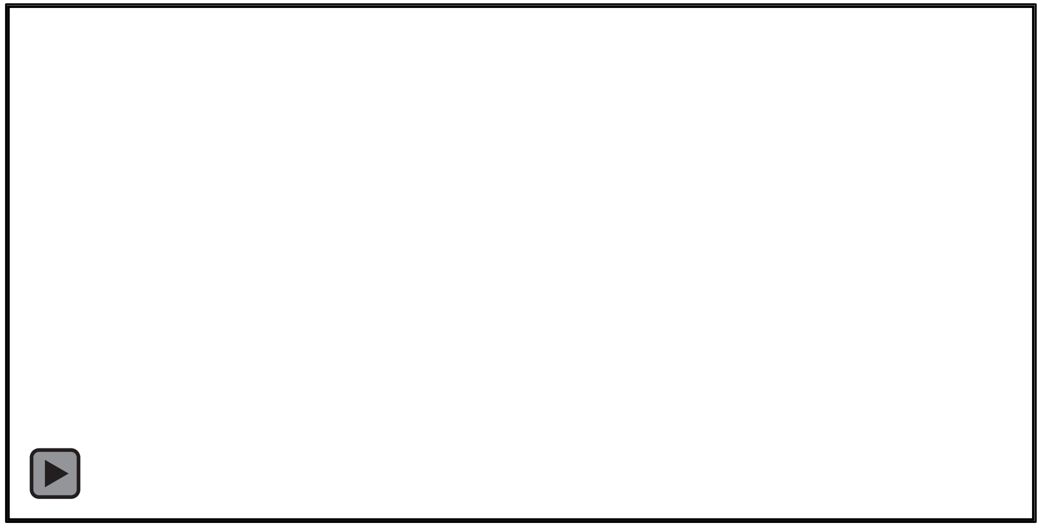
Implementations and Running Times

- For the given priority queue we are performing three different operation
  - Build
  - Extract minimum
  - Decrease key

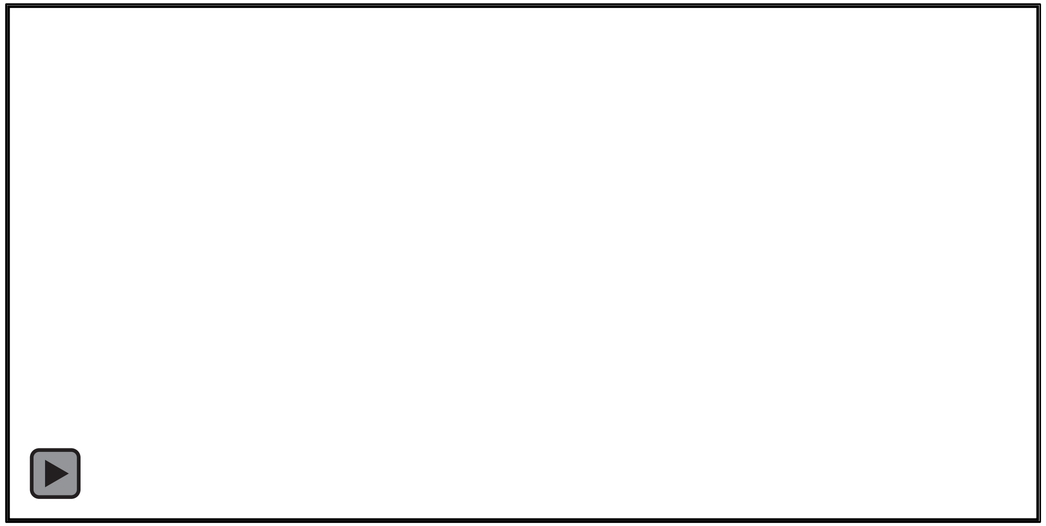
Priority Queue Q' on n items	Q Operations $O(\cdot)$ build(X)	delete_min()	decrease_key(id, k)	Dijkstra $O(\cdot)$ $n = \|V\| = O(\|E\|)$
Array	n	n	1	$\|V\|^2$
Binary Heap	n	$\log n_{(a)}$	$\log n$	$\|E\| \log \|V\|$
Fibonacci Heap	n	$\log n_{(a)}$	$1_{(a)}$	$\|E\| + \|V\| \log \|V\|$

Pic Credit : https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring 2020/d819e7f4568aced8d5b59e03db6c7b67_MIT6_006S20_lec13.pdf. MIT Introduction to algorithms

DIJKSTRA VS A*



Visualization of A*



Visualization of Dijkstra

Why Dijkstra?

No Heuristic Dependency: Dijkstra's algorithm does not require a heuristic function, making it more suitable when no good heuristic is available or when you need guaranteed correctness without the risk of an inadmissible/inconsistent heuristic.

Guaranteed Exploration: Dijkstra explores all reachable vertices, making it ideal for applications like finding the shortest path to all nodes (e.g., in network routing) rather than a single destination.

Vis Credit : https://qiao.github.io/PathFinding.js/visual/

The Bellman-Ford Shortest Path Algorithm

Neil Tang
03/11/2010

Shortest Simple Path

- Shortest Path (If exists)

$$Shortest\ Path = \ min\{\ w(\pi) |\ paths\ \pi\ s \to t\}$$

- If δ(s, v) is finite then there is a shortest path from s-v is simple
- **Simple path** in a graph is a path that does not revisit any vertex
  - At most |v| vertex
  - At most |v|-1 edges
- $\delta_{|V|}(s, v) < \delta_{|V|-1}(s, v)$ then $\delta(s, v) = -\infty$

The Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

1. Initialize-Single-Source(G, s)
2. **for** i := 1 to |V| - 1 **do**
3. **for** each edge (u, v) $\in$ E **do**
4. Relax(u, v, w)
5. **for** each vertex v $\in$ u.adj **do**
6. if d[v] > d[u] + w(u, v)
7. **then return** False // there is a negative cycle
8. **return** True

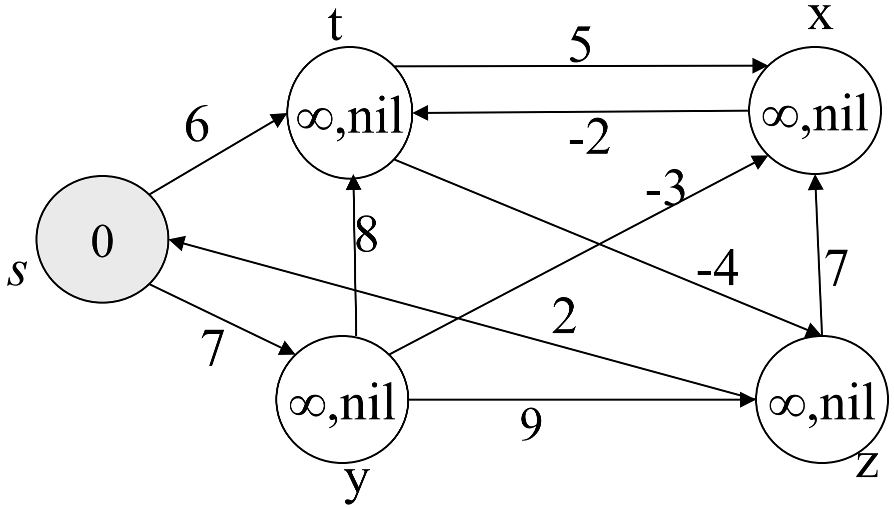
Relax(u, v, w)

if d[v] > d[u] + w(u, v)

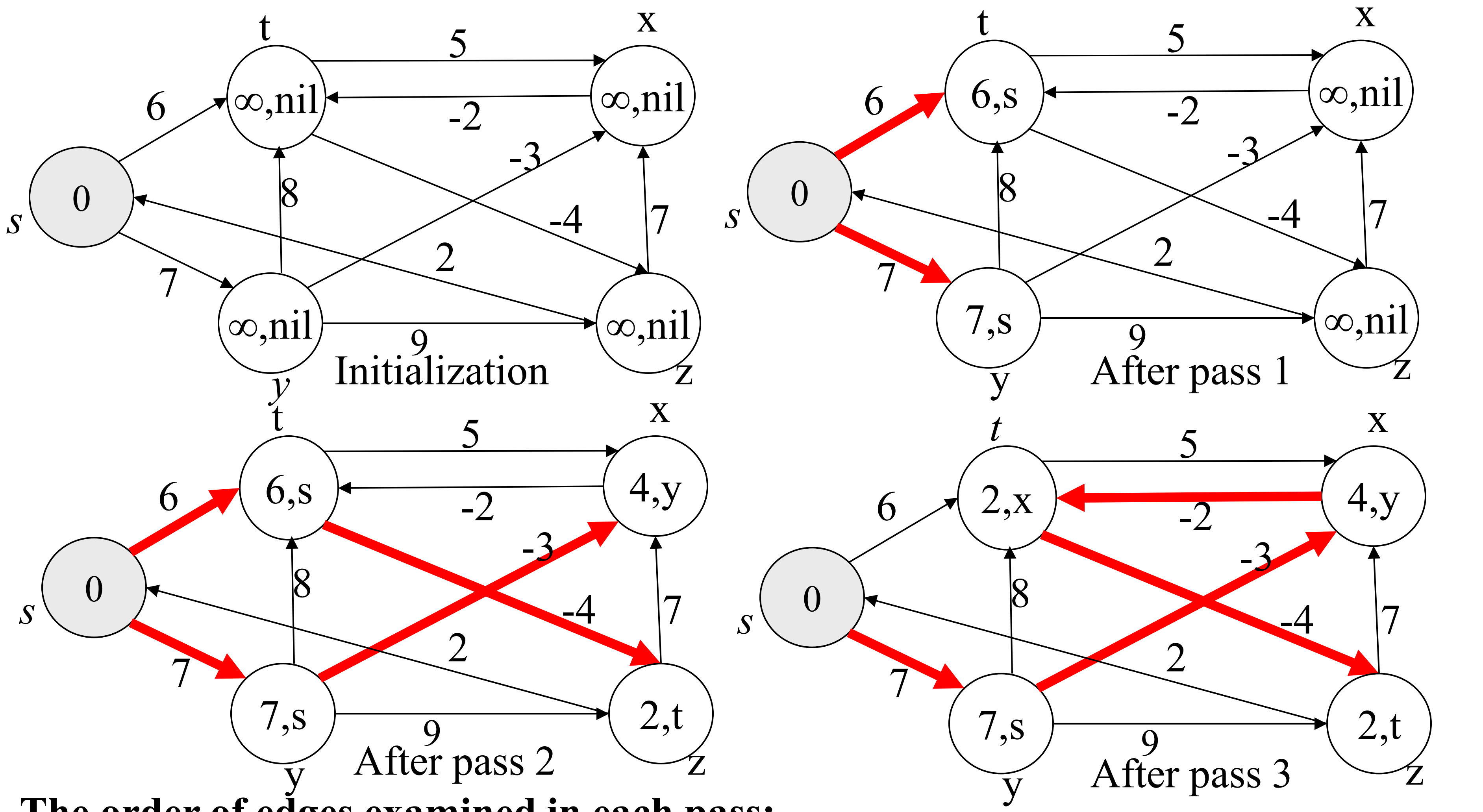
then d[v] := d[u] + w(u, v)

parent[v] := u

The Bellman-Ford Algorithm



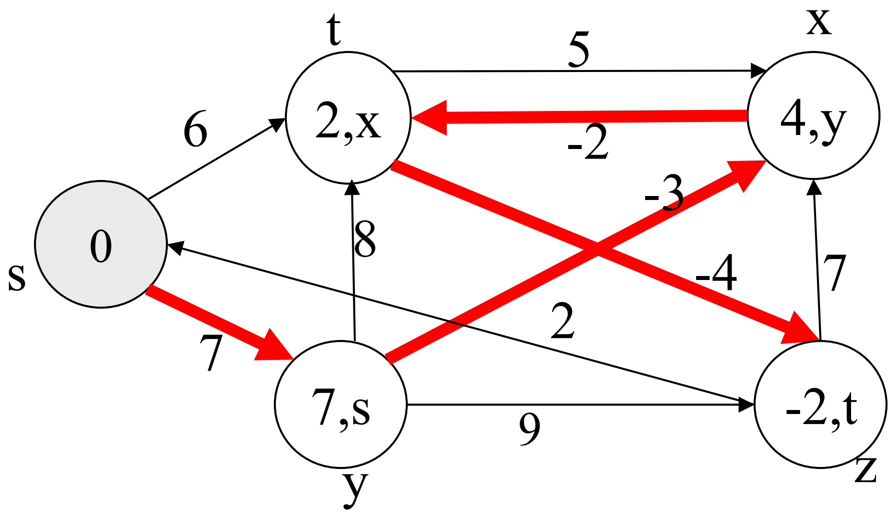
Graph Duplication



The order of edges examined in each pass:

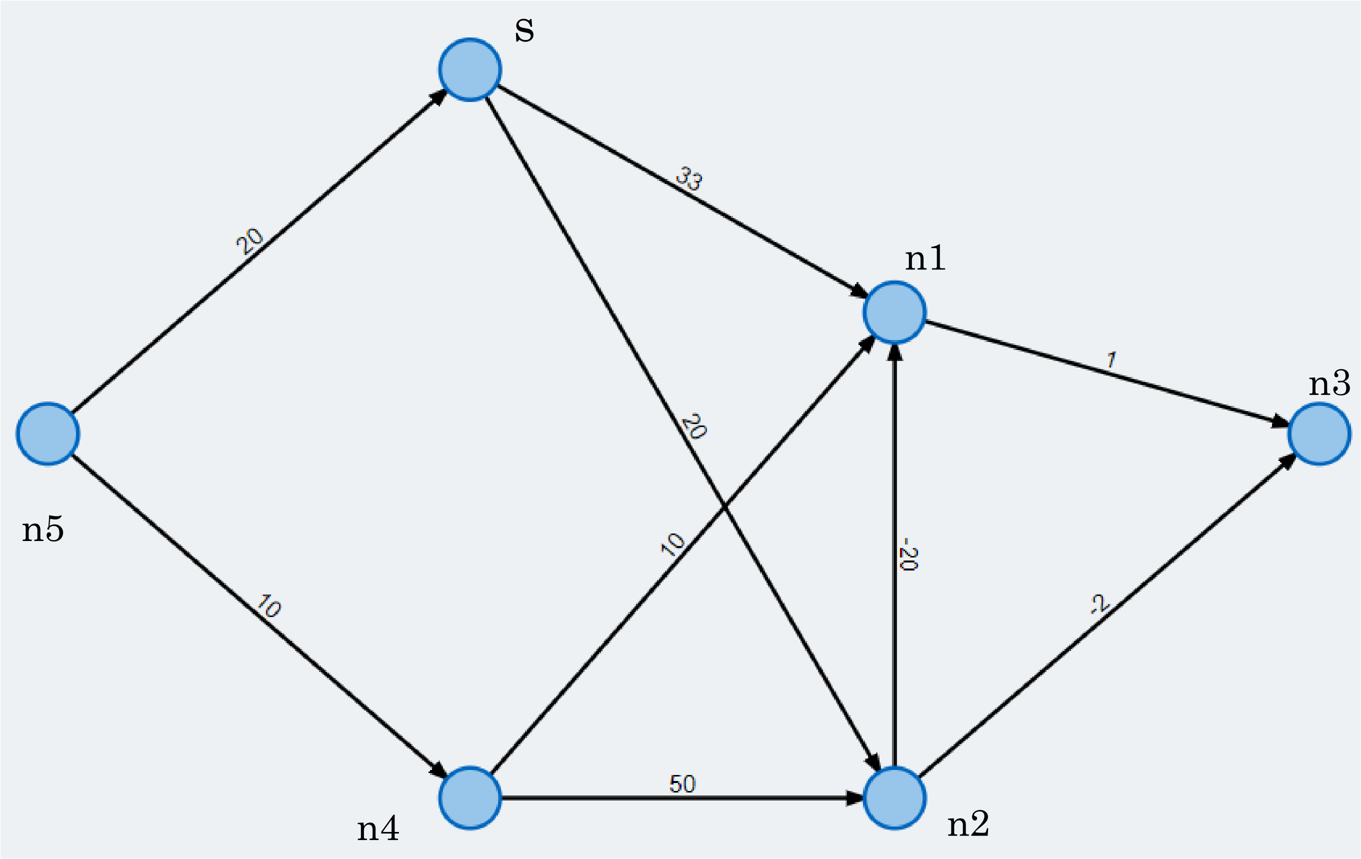
(s, t), (s, y),(t, x), (t, z), (x, t), (y, x), (y, t), (y, z), (z, x), (z, s),

The Bellman-Ford Algorithm



After pass 4

THE BELLMAN-FORD ALGORITHM VISUALIZATION



The order of edges examined in each pass:

(s, n1), (n2, n3),(n2, n1), (n1, n3), (s, 2), (n4, n1), (n4, n2), (n5, s), (n5, n4)

Vis Credit : TUM Bellman Ford : https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html

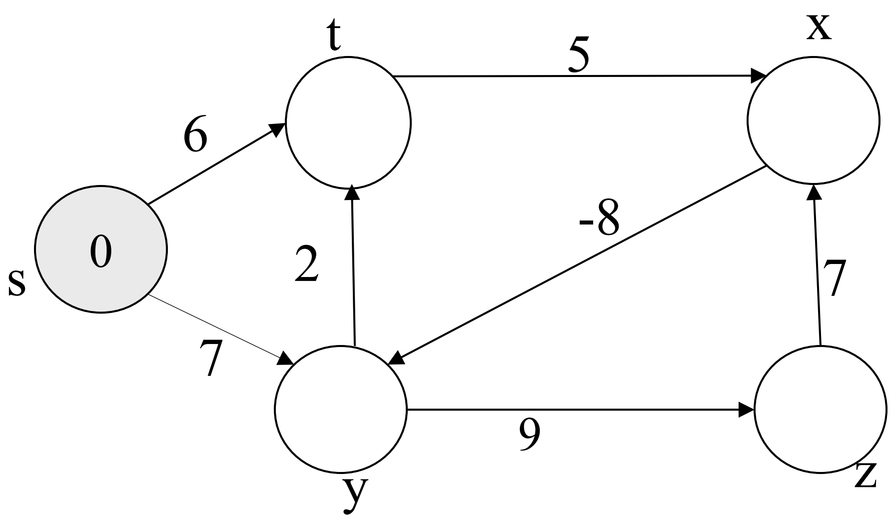
The Bellman-Ford Algorithm Visualization

The order of edges examined in each pass:

(s, n1), (n2, n3),(n2, n1), (n1, n3), (s, n2), (n4, n1), (n4, n2), (n5, s), (n5, n4)

Vis Credit : TUM Bellman Ford : https://algorithms.discrete.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html

THE BELLMAN-FORD ALGORITHM



The order of edges examined in each pass:

(s, t), (s, y),(t, x), (x,y), (y, t), (y, z), (z, x),
(s, t), (s, y),(t, x), (x,y), (y, t), (y, z), (z, x),
(s, t), (s, y),(t, x), (x,y), (y, t), (y, z), (z, x),
(s, t), (s, y),(t, x), (x,y), (y, t), (y, z), (z, x),

Time Complexity

```
Bellman-Ford(G, w, s)
Initialize-Single-Source(G, s)                      → O(|V|)
for i := 1 to |V| - 1 do
    for each edge (u, v) ∈ E do
        Relax(u, v, w)                              → O(|V||E|)
for each vertex v ∈ u.adj do                        → O(|E|)
    if d[v] > d[u] + w(u, v)
        then return False  // there is a negative cycle
return True
```

Time complexity: O(|V||E|)

DIFFERENCES

- Negative link weight: The Bellman-Ford algorithm works; Dijkstra's algorithm doesn't.

- Time complexity: The Bellman-Ford algorithm is higher than Dijkstra's algorithm.

References

- Dijkstra's original paper:
  E. W. Dijkstra. (1959) *A Note on Two Problems in Connection with Graphs*. Numerische Mathematik, 1. 269-271.
- MIT OpenCourseware, 6.046J Introduction to Algorithms.
  < http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/> Accessed 4/25/09
- Meyers, L.A. (2007) Contact network epidemiology: Bond percolation applied to infectious disease prediction and control. *Bulletin of the American Mathematical Society* **44**: 63-86.
- Department of Mathematics, University of Melbourne. *Dijkstra's Algorithm*.
  <http://www.ms.unimelb.edu.au/~moshe/620-261/dijkstra/dijkstra.html > Accessed 4/25/09