Week 9: Lecture Notes

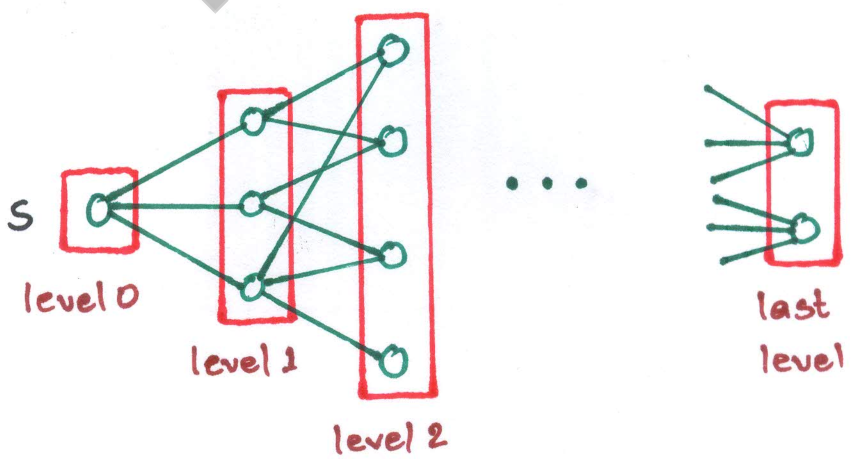
Topics:

- BFS and DFS
- Shortest path problem
- Dijktra's Algorithm
- Example of Dijktra
- Bellman Ford

Breadth-First Search

Explore graph level by level from s (start vertex)

- level 0 = {s}
- level i = vertices reachable by path of i edges but not fewer.
- build level i>0 from i-1 by trying all outgoing edges, but ignoring vertices from previous levels

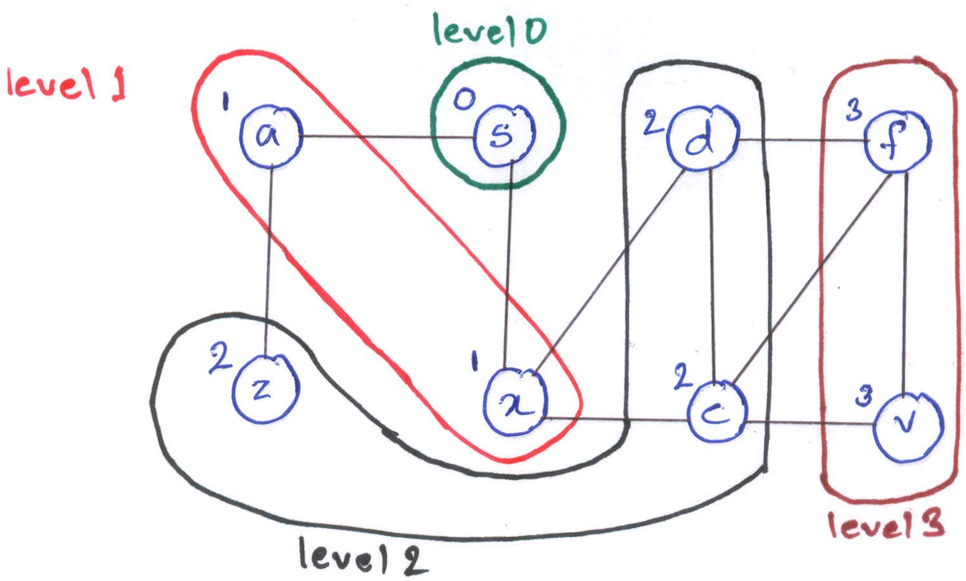


Breadth-First-Search Algorithm

BFS (V, Adj, s):

```
level = {s : 0}
parent = {s: None}
i = 1
frontier = [s]
while frontier
    next = []
    for u in frontier :
        for v in Adj [u]:
            if u not in level:
                level [v]= i
                parent [v]= u
                next.append (v)
    frontier = next
    i = i+1
```


Example



$frontier_0 = \{s\}$

$frontier_1 = \{a, x\}$

$frontier_2 = \{z, d, c\}$

$frontier_3 = \{f, v\}$

(not x, c, d)

Analysis

- vertex V enters next (and then frontier) only once

  (because level[s] then set)

  base case : $v = s$

- $\Rightarrow$ Adj[v] looped through only once.

$$\text{time} = \sum_{v \in V} |\text{Adj}[v]| = \begin{cases} |E| & \text{for directed graphs} \\ 2|E| & \text{for undirected graphs} \end{cases}$$

- $\Rightarrow O(E)$ time

- $O(V+E)$ ("linear time") to also list vertices

  unreachable from v (those still not assigned level)

Shortest Paths

- for every vertex v, fewest edges to get from s to v

is

$$\begin{cases} \text{level}[v] & \text{if } v \text{ is assigned level} \\ \infty & \text{else (no path)} \end{cases}$$

- parent pointers from shortest-path tree

  = union of such shortest path for each v.

  $\Rightarrow$ to find shortest path, take v, parent[v], parent[parent[v]], etc., until s (or None)

Depth-First Search (DFS)

This is like exploring a maze.

- follow a path until you get stuck.
- backtrack along breadcrumbs until reach unexplored neighbour
- recursively explore
- careful not to repeat a vertex.

Depth-First-Search Algorithm

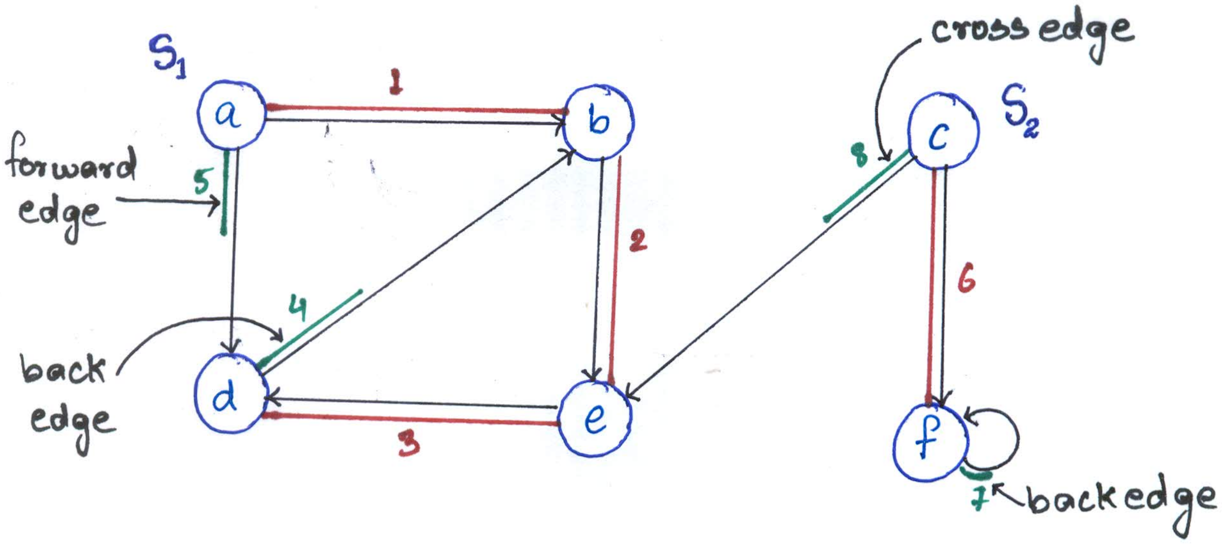
```
parent = {s: None}
DFS-visit (V, Adj, s):
    for v in Adj[s]:
        if v not in parent:
            parent [v] = s
            DFS-visit (V, Adj, v)

DFS (V, Adj)
    parent = {}
    for s in V:
        if s not in parent:
            parent [s] = None
            DFS-Visit (V, Adj, s)
```

search from start vertex s (only see stuff reachable from s)

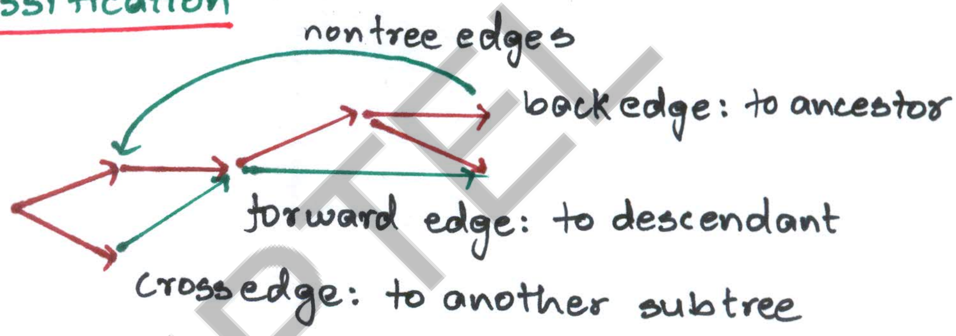
explore entire graph

(could do same to extend BFS)

Example of DFS



Edge Classification



- to compute this classification (back or not), mark nodes for duration they are "on the stack"

- only tree and back edges in undirected graph.

Analysis of DFS

- DFS-visit gets called with a vertex s only once
  $\Rightarrow$ time in DFS-visit $= \sum_{s \in V} |Adj[s]| = O(E)$

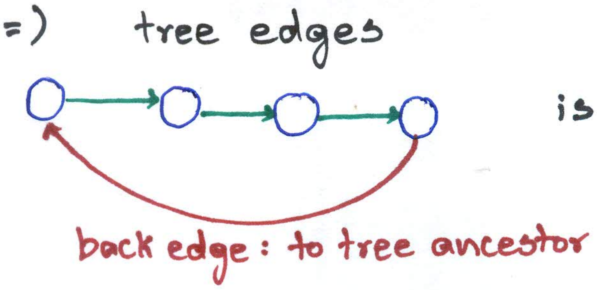
- DFS outer loop adds just $O(V)$
  $\Rightarrow$ $O(V+E)$ time (linear time)

Cycle Detection

Graph G has a cycle ⟺ DFS has a back edge

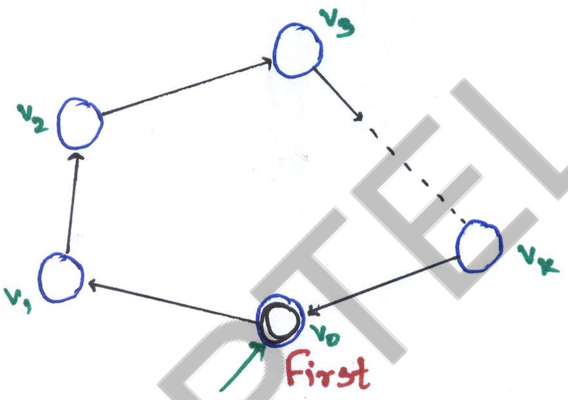
Proof:

(⇐)



is a cycle

(⇒) consider first visit to cycle:



- before visit to v_i finishes,
  will visit v_{i+1} (and finish):
  will consider edge (v_i, v_{i+1})
  ⇒ visit v_{i+1} now or already did
- ⇒ before visit to v_0 finishes
  will visit v_k (and didn't before)
- ⇒ before visit to v_k (or v_0) finishes
  will see (v_k, v_0) as back edge.

Job Scheduling

Given Directed Acyclic Graph (DAG), where vertices represents task and edges represent dependencies, order tasks without violating dependencies.

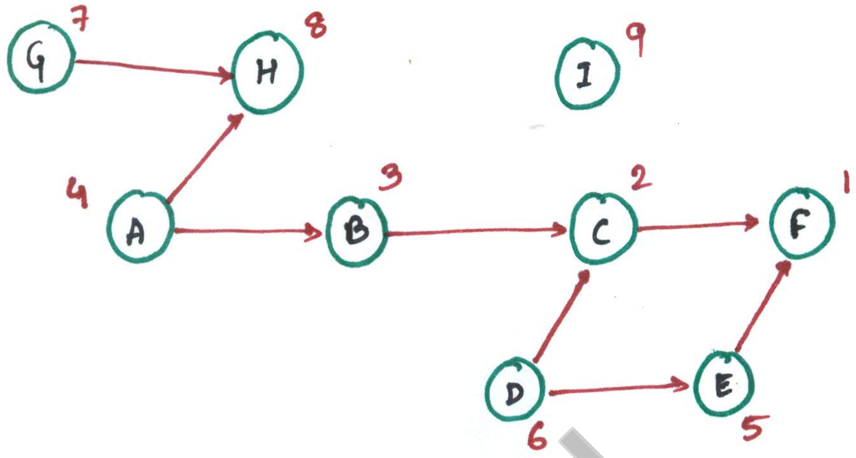


Fig: Dependence graph: DFS finishing times.

Source:

Source = vertex with no incoming edges
= scheduling at begining (A, G, I).

Attempt:

BFS from each source

- from A finds A, BH, C, F
- from D finds D, BE, CF ← slow... and wrong!.
- from G finds G, H
- from I finds I.

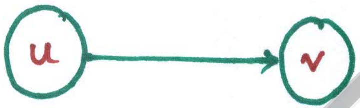
Topological Sort

Reverse of DFS finishing times
(time at which DFS-visit(v) finishes)

```
DFS-Visit(v)
  ...
    order.append(v)
order.reversed()
```

Correctness

For any edge (u,v) - u ordered before v, i.e. v finished before u



- if u visited before v:
  - before visit to u finishes, will visit v (via (u,v) or otherwise)

  $\Rightarrow$ v finishes before u

if v visited before u

graph is acyclic

$\Rightarrow$ u cannot be reached from v

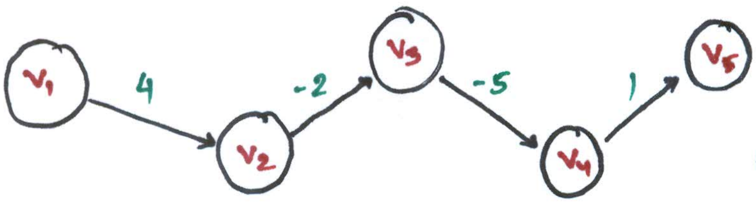
$\Rightarrow$ visit to v finishes before visiting u.

Paths in graphs

Consider a diagraph $G=(V,E)$ with edge-weight function $w: E \to R$. The weight of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Example:



$w(p) = \underline{\underline{-2}}$

Shortest Paths

A "shortest path" from u to v is a path of minimum weight from u to v.

The "shortest path weight" from u to v is defined as:

$$\delta(u,v) = \min \{ w(p) : p \text{ is a path from } u \text{ to } v \}$$

Note: $\delta(u,v) = \infty$ if no path from u to v exists.

Optimal Substructure

Theorem:

A subpath of a shortest path is a shortest path.

Triangle Inequality

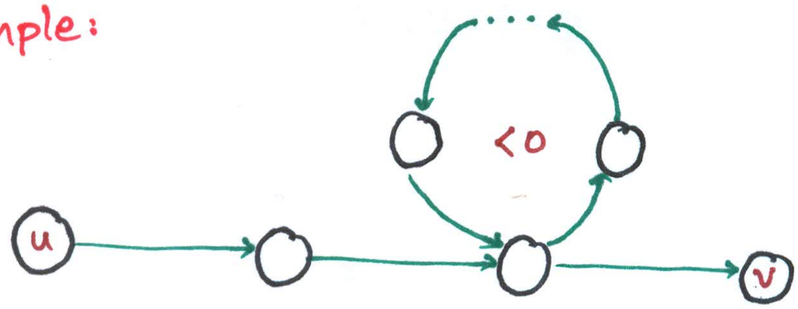
For all $u, v, x \in V$, we have

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

Well-definedness of shortest paths

If a graph G contains a negative-weight cycle, then some shortest path may not exist.

Example:



Single-source shortest paths

Problem: From a given source vertex $s \in V$, find the shortest-path weights $\delta(u,v)$ for all $v \in V$

If all edge weights $w(u,v)$ are non-negative, all shortest-path weights must exist.

Idea: Greedy

1. Maintain a set S of vertices whose shortest-path distance from s are known.
2. At each step add to S the vertex $v \in V-S$ whose distance estimate from s is minimal.
3. Update the distance estimates of vertices adjacent to v.

Dijkstra's Algorithm

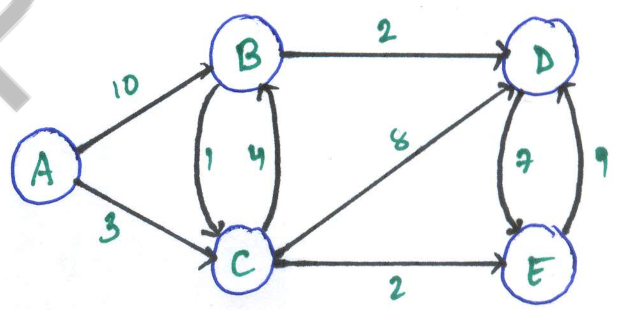
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \phi$
$Q \leftarrow V$ ▶ Q is a priority queue maintaining V-S
while $Q \neq \phi$
    do $u \leftarrow$ EXTRACT-MIN (Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u,v)$
                then $d[v] \leftarrow d[u] + w(u,v)$

} relaxation step

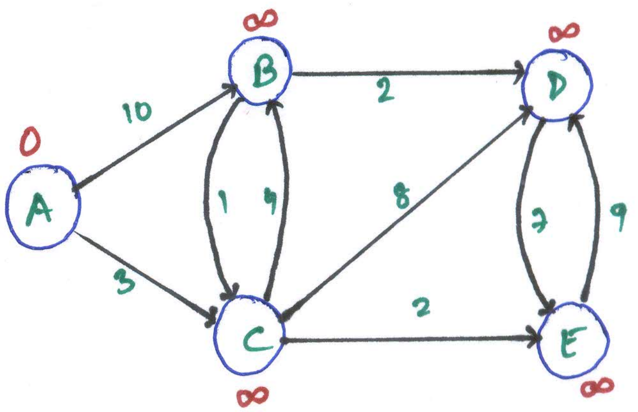
Implicit DECREASE-KEY

Example of Dijkstra's Algorithm

Graph with non-negative edge weights

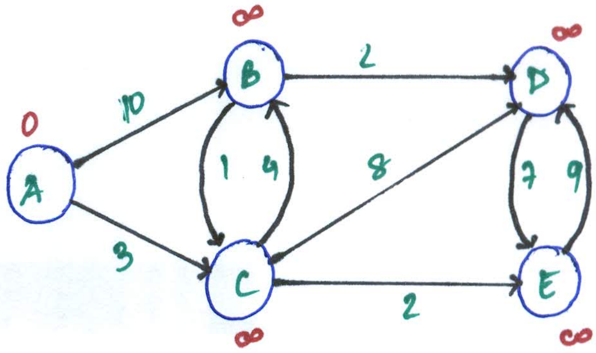


Initialize



Q: A B C D E
0 ∞ ∞ ∞ ∞

A ← EXTRACT-MIN(Q)

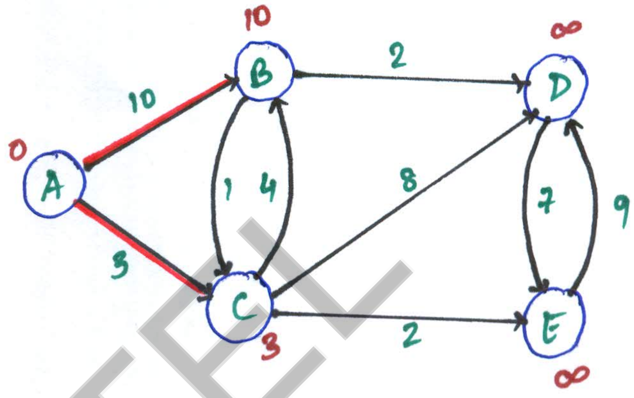


Q:

A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞

S: {A}

Relax all edges leaving A

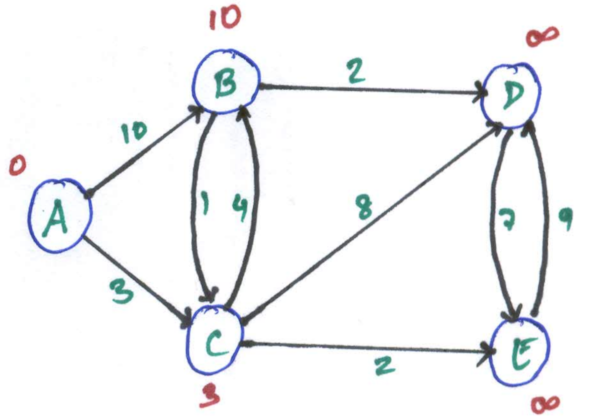


Q:

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

S: {A}

C ← EXTRACT-MIN(Q)

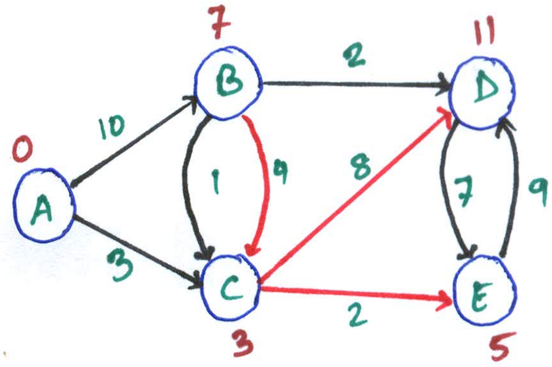


Q:

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

S: {A, C}

Relax all edges leaving C

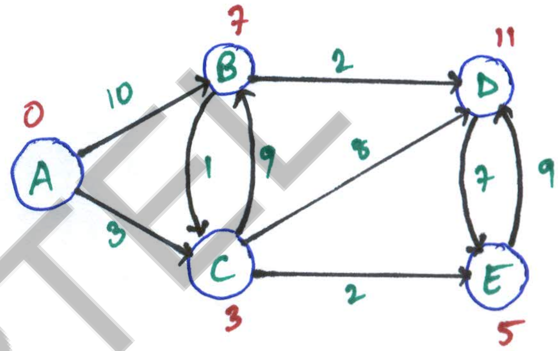


Q:

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

S: {A, C}

E ← EXTRACT-MIN (Q)

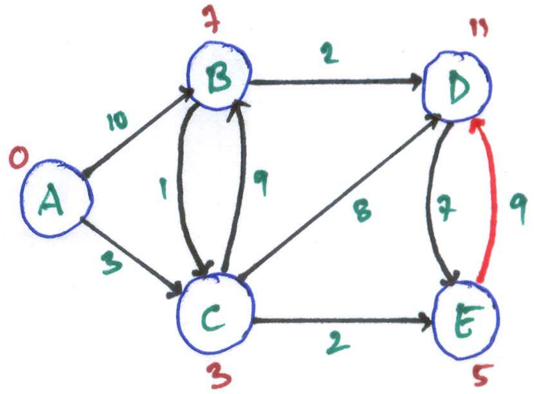


Q:

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

S: {A, C, E}

Relax all edges leaving E

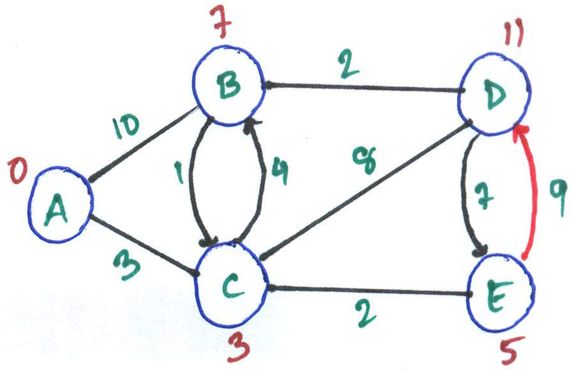


Q

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

S: {A, C, E}

B ← EXTRACT-MIN(Q):

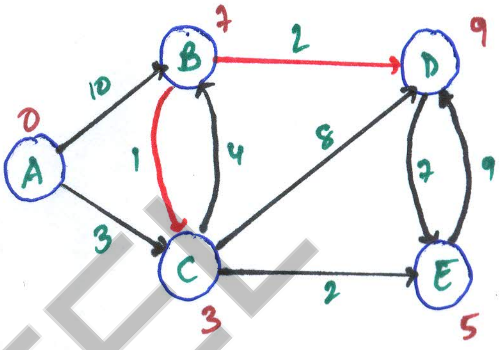


Q:

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

S: {A, C, E, B}

Relax all edges leaving B

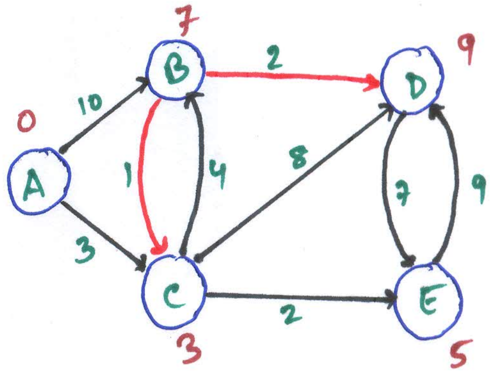


Q:

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

S: {A, C, E, B}

D ← EXTRACT-MIN(Q)



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}

Correctness - Part I

Lemma: Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s,v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof:

Suppose not. Let v be the first vertex for which $d[v] < \delta(s,v)$, and let u be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u,v)$. Then

$$
\begin{aligned}
d[v] &< \delta(s,v) && \text{supposition} \\
&\leq \delta(s,u) + \delta(u,v) && \text{triangle inequality} \\
&\leq \delta(s,u) + w(u,v) && \text{sh. path} \leq \text{specific path} \\
&\leq d[u] + w(u,v) && v \text{ is first violation.}
\end{aligned}
$$

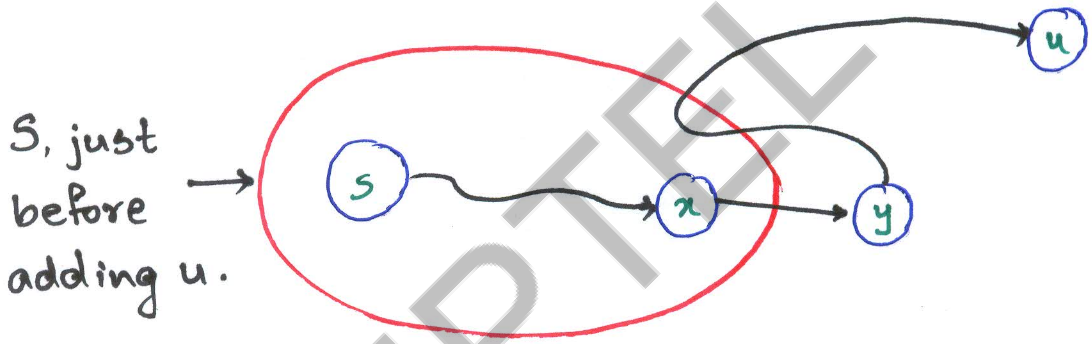
which is a contradiction.

Correctness - Part II

Theorem: Dijkstra's algorithm terminates with $d[v] = \delta(s,v)$ for all $v \in V$

Proof:

It suffices to show that $d[v] = \delta(s,v)$ for every $v \in V$ when v is added to S. Suppose u is the first vertex added to S for which $d[u] \neq \delta(s,u)$. Let y be the first vertex in $V-S$ along a shortest path from s to u, and x be it's predecessor:



Since u is the first vertex violating the claimed invariant, we have $d[x] = \delta(s,x)$. Since subpaths of shortest paths are shortest paths, it follows that $d[y]$ was set to $\delta(s,x) + w(x,y) = \delta(s,y)$ when (x,y) was relaxed just after x was added to S.

Consequently, we have $d[y] = \delta(x,y) \leq \delta(s,u) \leq d[u]$.

But $d[u] \leq d[y]$ by our choice of u, and hence

$$d[y] = \delta(x,y) = \delta(s,u) = d[u]$$

which is a contradiction.

Analysis of Dijkstra

while $Q \neq \phi$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u,v)$
        then $d[v] \leftarrow d[u] + w(u,v)$

Handshaking Lemma

$\Rightarrow \Theta(E)$ implicit DECREASE-KEY's

Time $= \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

Note: Same formula as in the analysis of Prim's minimum spanning tree algorithm.

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\log V)$	$O(\log V)$	$O(E \log V)$
fibonacci heap	$O(\log V)$ amortized	$O(1)$ amortized	$O(E + V \log V)$ worst case

Negative Weight Cycle

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Bellman-Ford Algorithm:

Finds all shortest-path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative cycle exists.

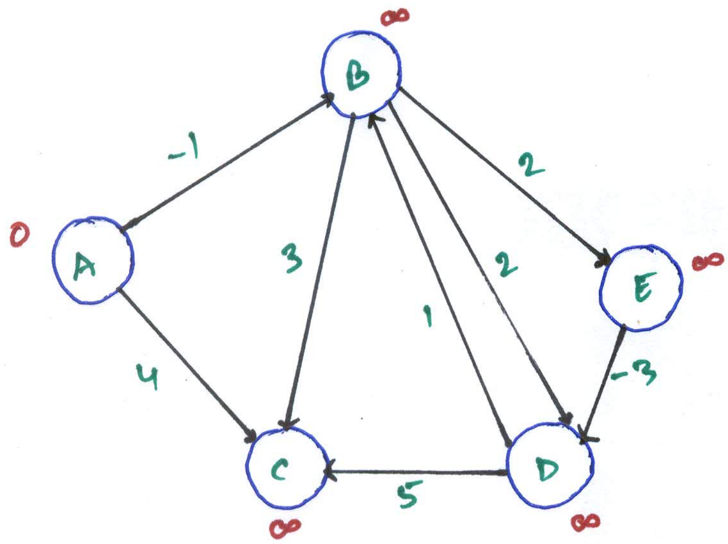
Bellman-Ford Algorithm

```
d[s] ← 0                                       }
for each v ∈ V - {s}                           }  initialization
    do d[v] ← ∞                                }

for i ← 1 to |V| - 1
    do for each edge (u,v) ∈ E
        do if d[v] > d[u] + w[u,v]             }  relaxation
            then d[v] ← d[u] + w(u,v)          }  step

for each edge (u,v) ∈ E
    do if d[v] > d[u] + w(u,v)
        then report that a negative-weight
             cycle exists.
```

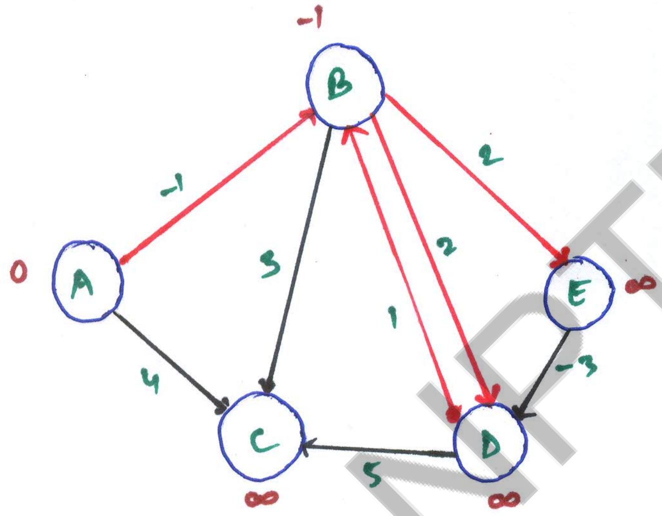
At the end, $d[v] = \delta(s, v)$

Time $= O(VE)$

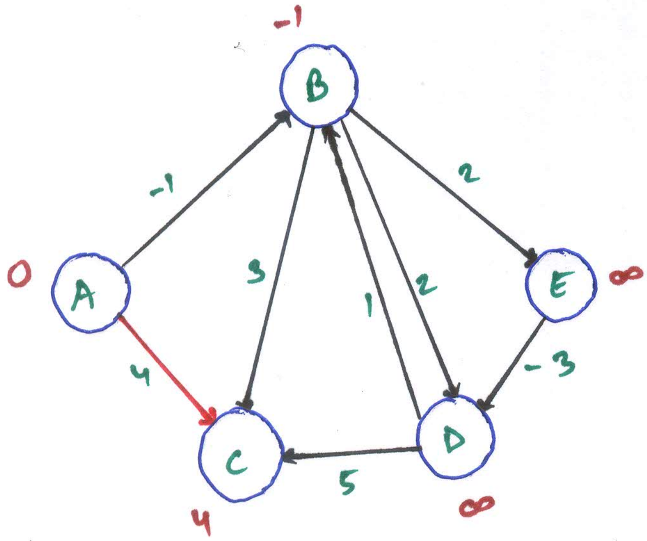
Example of Bellman-Ford



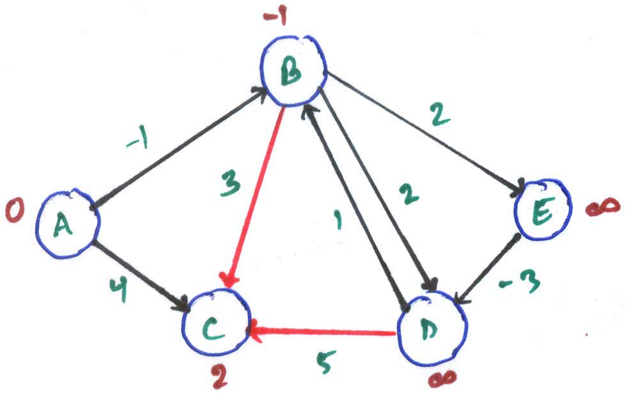
A	B	C	D	E
0	∞	∞	∞	∞



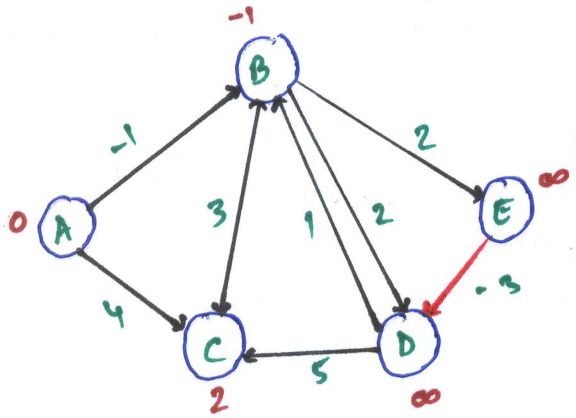
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞



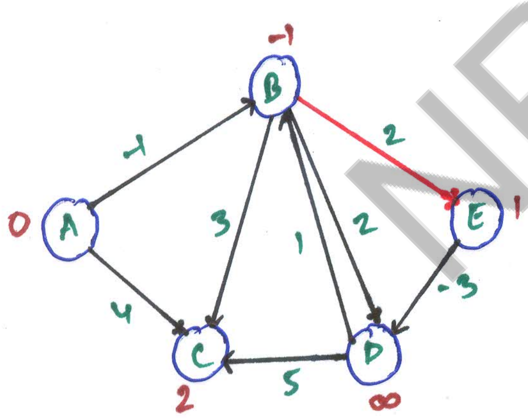
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞



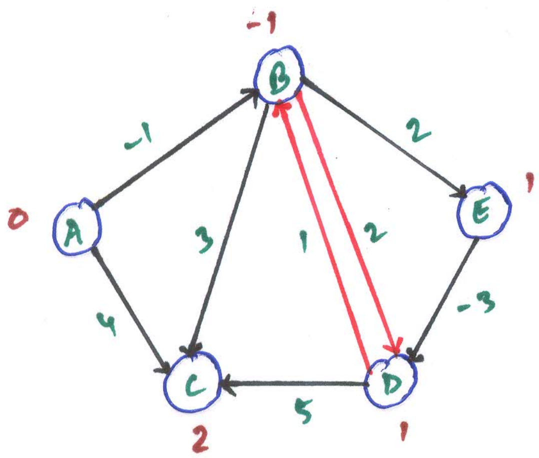
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



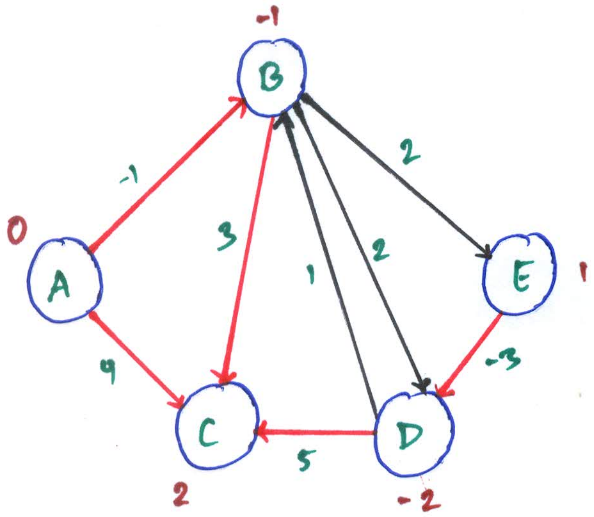
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞



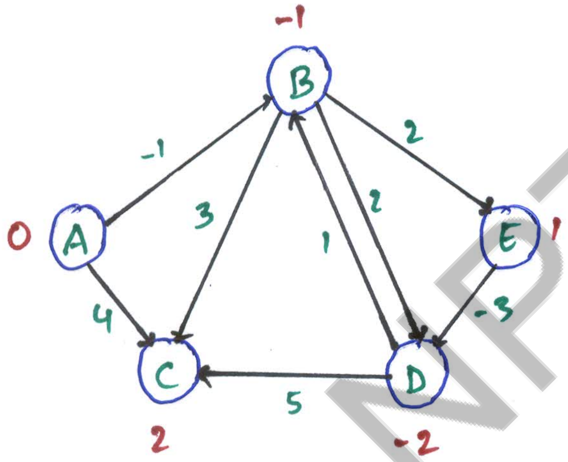
A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1



A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1



A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1