CSc 445

Shortest Paths in Graphs

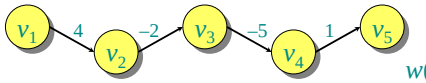
Slides courtesy of Erik Demaine and Charles E. Leiserson with a few modifications by Carola Wenk and Alon Efrat

Paths in Graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbf{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \ldots \to v_k$ is defined to be

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

Example:



$w(p) = -2$

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

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

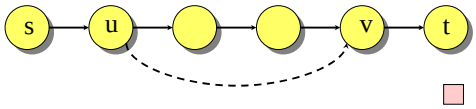
Also called ***distance*** of ***u*** from ***v***

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.

Proof. Cut and paste:

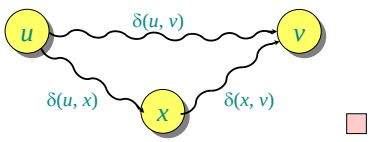


Triangle Inequality

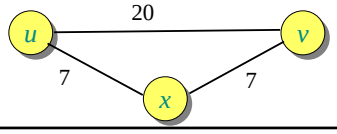
Theorem. For all $u, v, x \in V$, we have

$\delta(u, v) \le \delta(u, x) + \delta(x, v)$.

Proof.



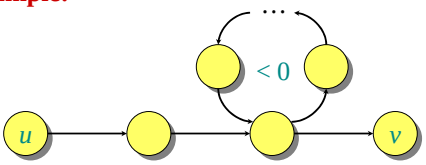
Note: This does not imply that $w(u,v) \le w(u,x)+w(x,v)$.



Well-Definedness of Shortest Paths

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

Example:



Single-Source Shortest Paths

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

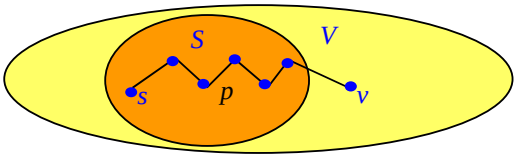
If all edge weights $w(u, v)$ are *nonnegative*, all shortest-path weights must exist. We'll use **Dijkstra**'s algorithm.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path distances from s are known. Also maintain **distance estimates** to the other vertices.
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.

Internal Paths - Definition

1. Let S be a set of vertices (that contains s).
2. We say that path p is **internal** to S if all its vertices, excluding maybe the last one, are in S.
3. Distance estimation: The algorithm maintains for every vertex v the value $d[v]$, which is the length of the shortest path from s to v, which is internal to S.
4. Will show: If v is in S, then $d[v] = \delta(s, v)$.



Dijkstra's Algorithm

```
d[s] ← 0
for each v ∈ V – {s}
    do d[v] ← ∞
S ← ∅
Q ← V          ▷ Q is a priority queue maintaining V – S
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each v ∈ Adj[u]      /* all nbrs of u */
           do if    d[v] > d[u] + w(u, v)
              then d[v] ← d[u] + w(u, v)      relaxation step
```

Implicit DECREASE-KEY

Dijkstra's Algorithm

```
d[s] ← 0
for each v ∈ V – {s}
    do d[v] ← ∞
S ← ∅
Q ← V
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each v ∈ Adj[u]
           do if    d[v] > d[u] + w(u, v)
              then d[v] ← d[u] + w(u, v)      relaxation step
```

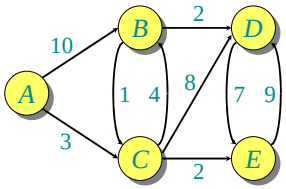
Implicit DECREASE-KEY

PRIM's algorithm

```
Q ← V
key[v] ← ∞ for all v ∈ V
key[s] ← 0 for some arbitrary s ∈ V
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
              then key[v] ← w(u, v)
                   π[v] ← u
```

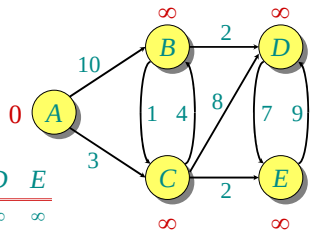
Example of Dijkstra's Algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's Algorithm

Initialize:

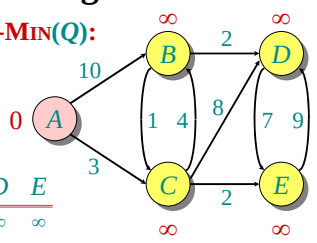


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Example of Dijkstra's Algorithm

"*A*" ← EXTRACT-MIN(*Q*):

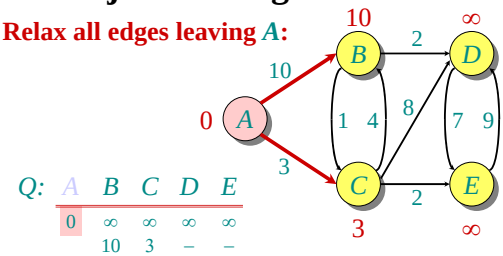


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

S: { *A* }

Example of Dijkstra's Algorithm

Relax all edges leaving *A*:

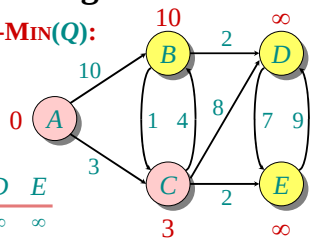


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–

S: { *A* }

Example of Dijkstra's Algorithm

"*C*" ← EXTRACT-MIN(*Q*):

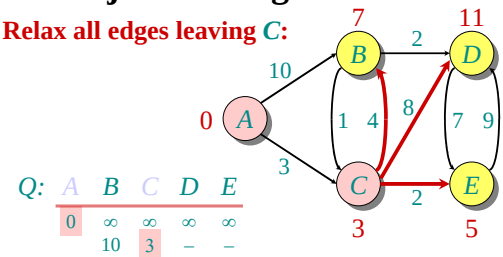


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–

S: { *A*, *C* }

Example of Dijkstra's Algorithm

Relax all edges leaving *C*:

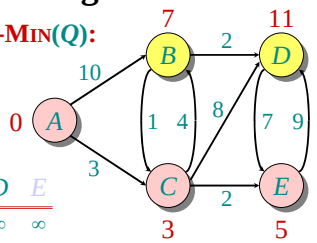


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

S: { *A*, *C* }

Example of Dijkstra's Algorithm

"*E*" ← EXTRACT-MIN(*Q*):

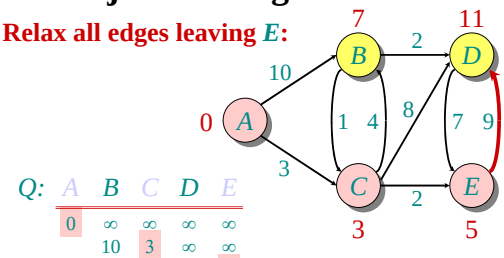


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

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

Example of Dijkstra's Algorithm

Relax all edges leaving *E*:

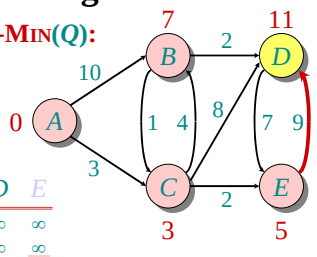


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Example of Dijkstra's Algorithm

"*B*" ← EXTRACT-MIN(*Q*):

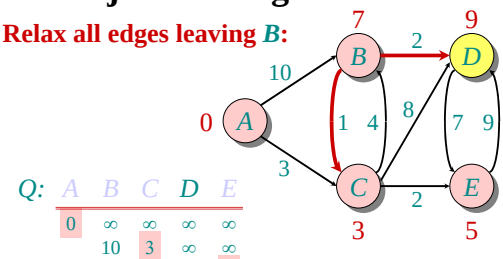


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Example of Dijkstra's Algorithm

Relax all edges leaving *B*:

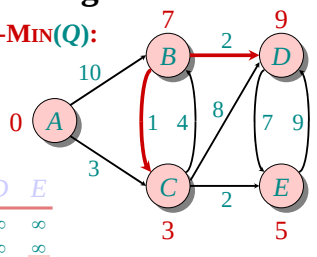


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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

Example of Dijkstra's Algorithm

"*D*" ← EXTRACT-MIN(*Q*):



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

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

Correctness — Part I

Lemma. At any stage of the algorithm, and for every vertex v, it is always true that $d[v] \geq \delta(s, v)$.

Proof. It is true after initialization (trivially).

Suppose not. Let v be the first (chronologically) 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,

$d[v] < \delta(s, v)$	supposition
$\leq \delta(s, u) + \delta(u, v)$	triangle inequality
$\leq \delta(s,u) + w(u, v)$	shortest path ≤ specific path
$\leq d[u] + w(u, v)$	v is first violation

Contradiction. □

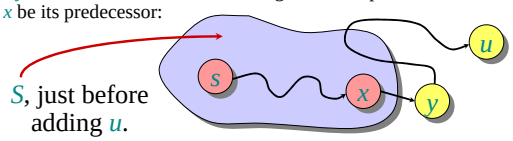
Handwaving: $d[v]$ is the length of **a** path to v, while $\delta(s, v)$ is the **shortest** path to v.

Correctness — Part II

Theorem. When the algorithm terminates, $d[v] = \delta(s, v)$, $\forall v \in V$.

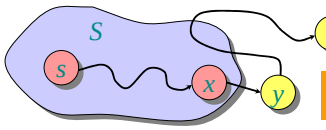
Proof. It suffices to show that $d[v] = \delta(s, v)$, when v is added to S.

- Suppose u is the first vertex added to S for which $d[u] > \delta(s, u)$.
  - Recall: $d[u] \geq \delta(s, u)$ always.
  - Recall $d[u] \leq d[w]$, $\forall w \in V\text{-}S$
- Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:



S, just before adding u.

Correctness — Part II (cont.)



Assumption: $d[u] > \delta(s, u)$.

- Since u is the first vertex violating the claimed invariant, $d[x] = \delta(s, x)$.
- Since subpaths of shortest paths are shortest paths, $\delta(s, y) = \delta(s, x) + w(x, y)$
- When x joined S, we perform a relaxation step: $d[y] = \min\{d[y], d[x] + w(x,y)\}$ so $d[y] = \delta(s, y)$
- If u is y we are done. So assume u is **not** y.
- We have $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$.
- But, $d[u] \leq d[y]$ by our choice of u, a contradiction.

□

Analysis of Dijkstra

```
|V| times:
    while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           S ← S ∪ {u}
           degree(u) times:
               for each v ∈ Adj[u]
                   do if d[v] > d[u] + w(u, v)
                         then d[v] ← 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.

Analysis of Dijkstra (cont.)

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

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

Unweighted Graphs

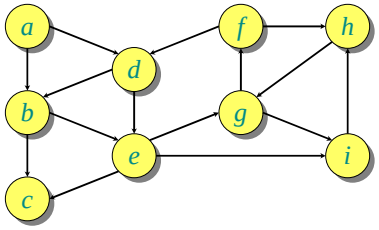
Suppose $w(u, v) = 1$ for all $(u, v) \in E$. Can the code for Dijkstra be improved?

- Use a simple FIFO queue instead of a priority queue.
- ***Breadth-first search***

```
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
                 then d[v] ← d[u] + 1
                      ENQUEUE(Q, v)
```

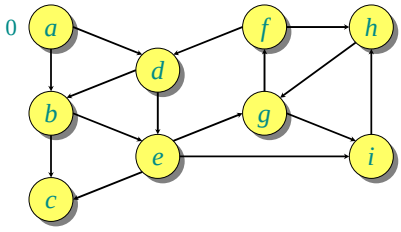
Analysis: Time = $O(V + E)$.

Example of Breadth-First Search



Q:

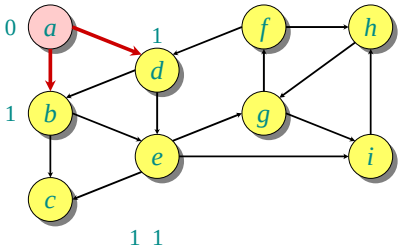
Example of Breadth-First Search



0

Q: a

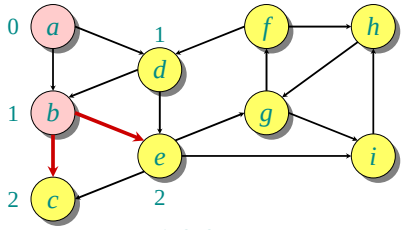
Example of Breadth-First Search



1 1

Q: a b d

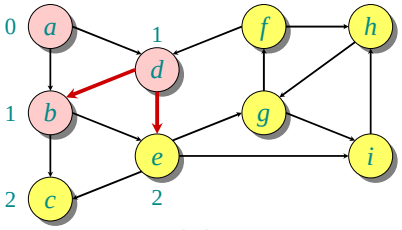
Example of Breadth-First Search



1 2 2

Q: a b d c e

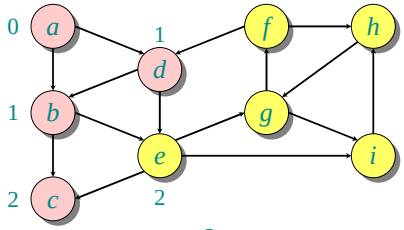
Example of Breadth-First Search



2 2

Q: a b d c e

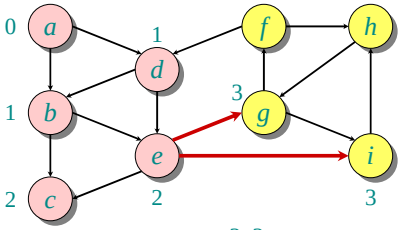
Example of Breadth-First Search



2

Q: a b d c e

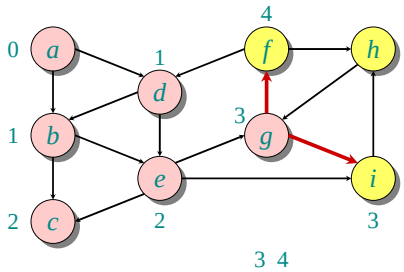
Example of Breadth-First Search



3 3

Q: a b d c e g i

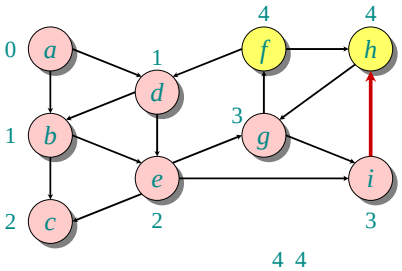
Example of Breadth-First Search



3 4

Q: a b d c e g i f

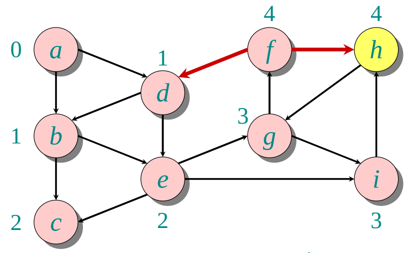
Example of Breadth-First Search



4 4

Q: a b d c e g i f h

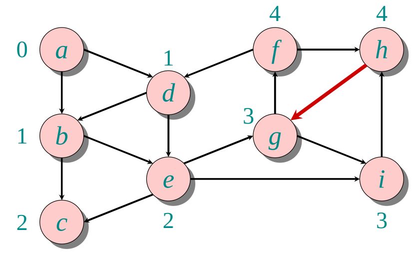
Example of Breadth-First Search



4

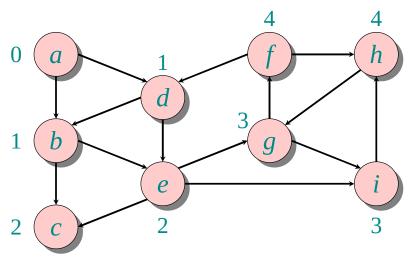
Q: *a b d c e g i f h*

Example of Breadth-First Search



Q: *a b d c e g i f h*

Example of Breadth-First Search



Q: *a b d c e g i f h*

Correctness of BFS

while $Q \neq \emptyset$
  do $u \leftarrow$ DEQUEUE(Q)
    for each $v \in Adj[u]$
      do if $d[v] = \infty$
        then $d[v] \leftarrow d[u] + 1$
          ENQUEUE(Q, v)

Key idea:

The FIFO Q in breadth-first search mimics the priority queue Q in Dijkstra.

- **Invariant:** v comes after u in Q implies that $d[v] = d[u]$ or $d[v] = d[u] + 1$.

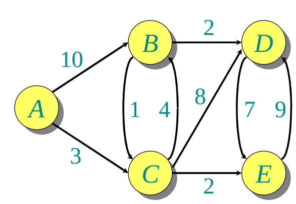
How to find the actual shortest paths?

Store a predecessor tree:

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \emptyset$
  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 $d[v] \leftarrow d[u] + w(u, v)$
          $\pi[v] \leftarrow u$ /* *Producing edges of the shortest paths tree* */

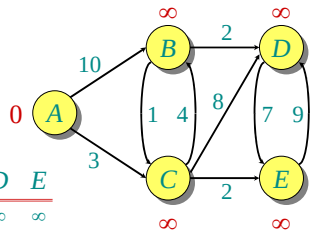
Example of Dijkstra's Algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's Algorithm

Initialize:

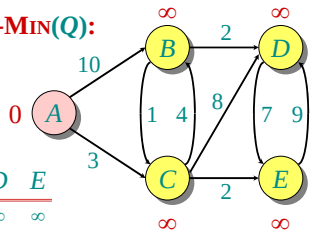


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { }

Example of Dijkstra's Algorithm

"*A*" ← EXTRACT-MIN(*Q*):



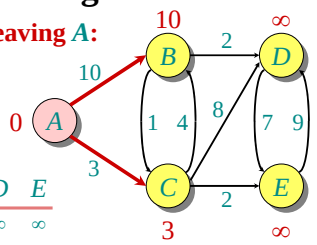
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { *A* }

π:	*A*	*B*	*C*	*D*	*E*
	–	–	–	–	–

Example of Dijkstra's Algorithm

Relax all edges leaving *A*:



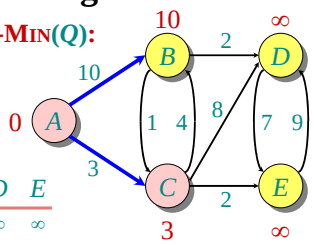
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–

S: { *A* }

π:	*A*	*B*	*C*	*D*	*E*
	–	A	A	–	-

Example of Dijkstra's Algorithm

"*C*" ← EXTRACT-MIN(*Q*):



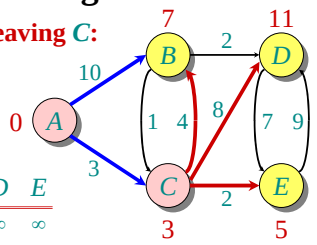
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–

S: { *A*, *C* }

π:	*A*	*B*	*C*	*D*	*E*
	–	A	A	–	-

Example of Dijkstra's Algorithm

Relax all edges leaving *C*:



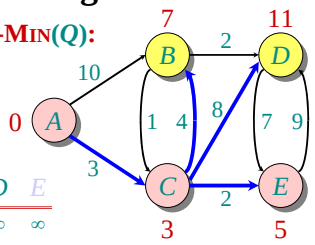
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

S: { *A*, *C* }

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

Example of Dijkstra's Algorithm

"*E*" ← EXTRACT-MIN(*Q*):



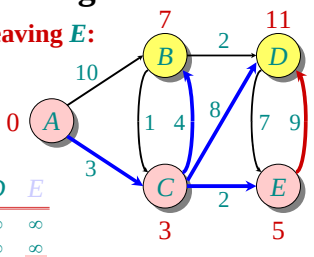
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

Example of Dijkstra's Algorithm

Relax all edges leaving *E*:



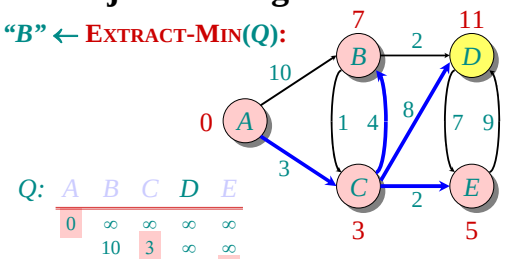
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

Example of Dijkstra's Algorithm

"*B*" ← EXTRACT-MIN(*Q*):



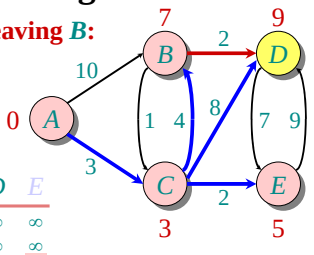
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	C	C

Example of Dijkstra's Algorithm

Relax all edges leaving *B*:



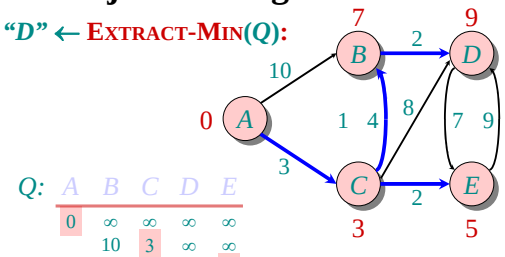
Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	B	C

Example of Dijkstra's Algorithm

"*D*" ← EXTRACT-MIN(*Q*):



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

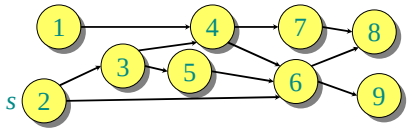
S: { *A*, *C*, *E*, *B*, *D* }

π:	*A*	*B*	*C*	*D*	*E*
	–	C	A	B	C

DAG Shortest Paths

If the graph is a ***directed acyclic graph*** (***DAG***), we first ***topologically sort*** the vertices:

- Determine $f : V \to \{1, 2, \ldots, |V|\}$ such that $(u, v) \in E \Rightarrow f(u) < f(v)$ (will describe later how).
- $O(V + E)$ time using depth-first search.



Walk through the vertices $u \in V$ in this order, relaxing the edges in $Adj[u]$, thereby obtaining the shortest paths from s in a total of $O(V + E)$ time.