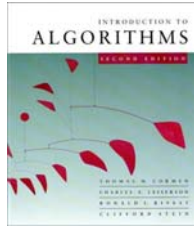




CS 5633 -- Spring 2008



Single Source Shortest Paths

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

4/8/08

CS 5633 Analysis of Algorithms

1

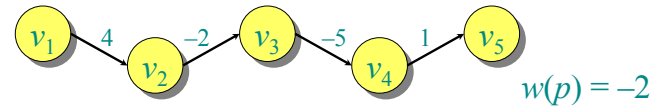


Paths in graphs

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

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

Example:



4/8/08

CS 5633 Analysis of Algorithms

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.

4/8/08

CS 5633 Analysis of Algorithms

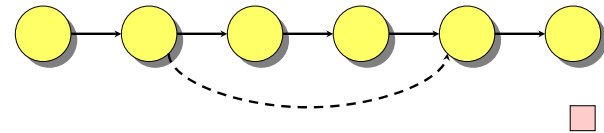
3



Optimal substructure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:



4/8/08

CS 5633 Analysis of Algorithms

4

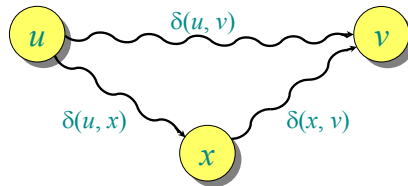


Triangle inequality

Theorem. For all $u, v, x \in V$, we have $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$.

Proof.

- $\delta(u, v)$ minimizes over **all** paths from u to v
- Concatenating two shortest paths from u to x and from x to v yields **one** specific path from u to v



4/8/08

CS 5633 Analysis of Algorithms

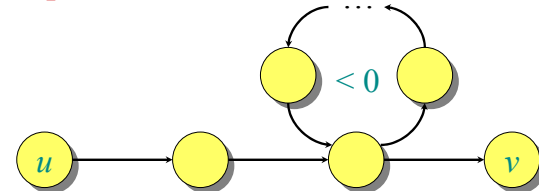
5



Well-definedness of shortest paths

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

Example:



4/8/08

CS 5633 Analysis of Algorithms

6



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.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path weights 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 .

4/8/08

CS 5633 Analysis of Algorithms

7



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

```

relaxation step

Implicit DECREASE-KEY

4/8/08

CS 5633 Analysis of Algorithms

8

Dijkstra

```

    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
    
```

PRIM'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)
    
```

It suffices to only check $v \in Q$, but it doesn't hurt to check all v

relaxation step

Implicit DECREASE-KEY

4/8/08 CS 5633 Analysis of Algorithms 9

Example of Dijkstra's algorithm

Graph with nonnegative edge weights:

```

    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)
    
```

4/8/08 CS 5633 Analysis of Algorithms 10

Example of Dijkstra's algorithm

Initialize:

S: {}

Q:

A	B	C	D	E
0	∞	∞	∞	∞

```

    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)
    
```

4/8/08 CS 5633 Analysis of Algorithms 11

Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

S: {A}

Q:

A	B	C	D	E
0	∞	∞	∞	∞

```

    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)
    
```

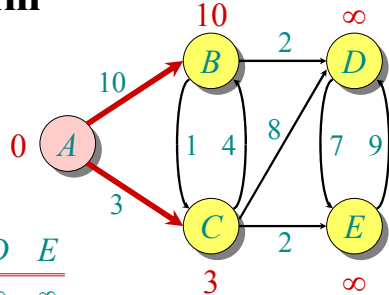
4/8/08 CS 5633 Analysis of Algorithms 12



Example of Dijkstra's algorithm

Relax all edges leaving A:

S: {A}



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	-	-

```

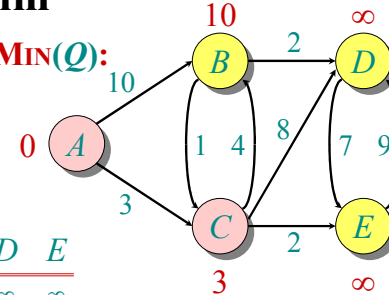
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):

S: {A, C}



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	-	-

```

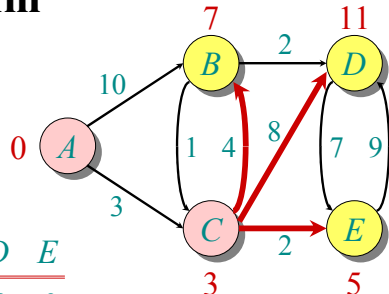
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

Relax all edges leaving C:

S: {A, C}



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

```

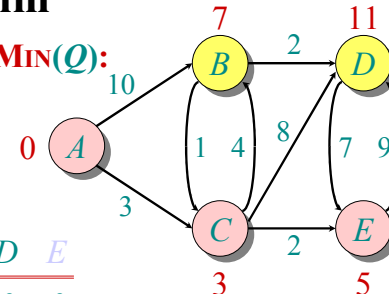
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

"E" ← EXTRACT-MIN(Q):

S: {A, C, E}



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

```

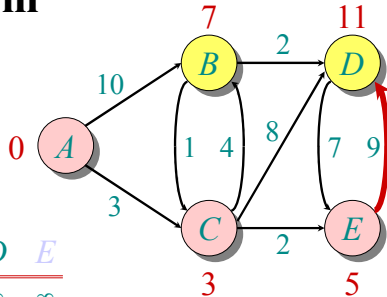
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

Relax all edges leaving **E**:

$S: \{A, C, E\}$



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

```

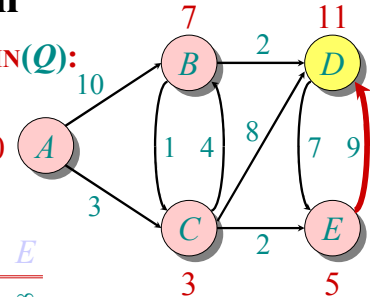
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

"B" ← EXTRACT-MIN(Q):

$S: \{A, C, E, B\}$



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

```

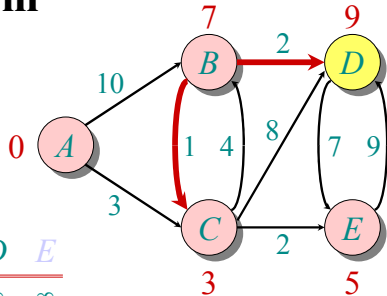
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

Relax all edges leaving **B**:

$S: \{A, C, E, B\}$



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

```

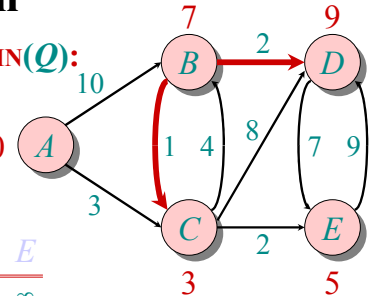
while Q ≠ ∅ do
  u ← EXTRACT-MIN(Q)
  S ← S ∪ {u}
  for each v ∈ Adj[u] do
    if d[v] > d[u] + w(u, v) then
      d[v] ← d[u] + w(u, v)
  
```



Example of Dijkstra's algorithm

"D" ← EXTRACT-MIN(Q):

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



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

```

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)
  
```



Analysis of Dijkstra

$|V|$ times $\left\{ \begin{array}{l} \text{while } Q \neq \emptyset \text{ do} \\ \quad u \leftarrow \text{EXTRACT-MIN}(Q) \\ \quad S \leftarrow S \cup \{u\} \\ \quad \text{for each } v \in \text{Adj}[u] \text{ do} \\ \quad \quad \text{if } d[v] > d[u] + w(u, v) \text{ then} \\ \quad \quad \quad d[v] \leftarrow d[u] + w(u, v) \end{array} \right.$

$\left. \begin{array}{l} \text{degree}(u) \\ \text{times} \end{array} \right\}$

Handshaking Lemma $\Rightarrow \Theta(|E|)$ implicit DECREASE-KEY's.

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

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



Analysis of Dijkstra (continued)

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

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



Correctness

Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S .

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



Correctness

Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S .

Proof. By induction.

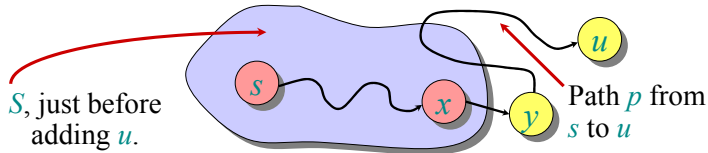
- Base: Before the while loop, $d[s]=0$ and $d[v]=\infty$ for all $v \neq s$, so (i) and (ii) are true.
- Step: Assume (i) and (ii) are true before an iteration; now we need to show they remain true after another iteration. Let u be the vertex added to S , so $d[u] \leq d[v]$ for all other $v \notin S$.



Correctness

Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S .

- (i) Need to show that $d[u] = \delta(s, u)$. Assume the contrary.
 \Rightarrow There is a path p from s to u with $w(p) < d[u]$. Because of (ii) that path uses vertices $\notin S$, in addition to u .
 \Rightarrow Let y be first vertex on p such that $y \notin S$.



4/8/08

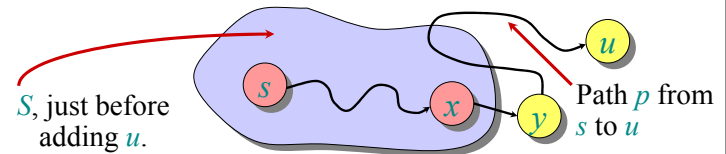
CS 5633 Analysis of Algorithms

25



Correctness

Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S .



$\Rightarrow d[y] \leq w(p) < d[u]$. Contradiction to the choice of u .

weights are nonnegative

assumption about path

4/8/08

CS 5633 Analysis of Algorithms

26



Correctness

Theorem. (i) For all $v \in S$: $d[v] = \delta(s, v)$
(ii) For all $v \notin S$: $d[v]$ = weight of shortest path from s to v that uses only (besides v itself) vertices in S .

- (ii) Let $v \notin S$. Let p be a shortest path from s to v that uses only (besides v itself) vertices in S .
 - p does not contain u : (ii) true by inductive hypothesis
 - p contains u : p consists of vertices in $S \setminus \{u\}$ and ends with an edge from u to v .
 $\Rightarrow w(p) = d[u] + w(u, v)$, which is the value of $d[v]$ after adding u . So (ii) is true.

4/8/08

CS 5633 Analysis of Algorithms

27



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

```

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

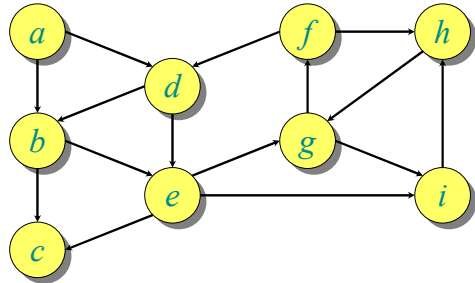
4/8/08

CS 5633 Analysis of Algorithms

28



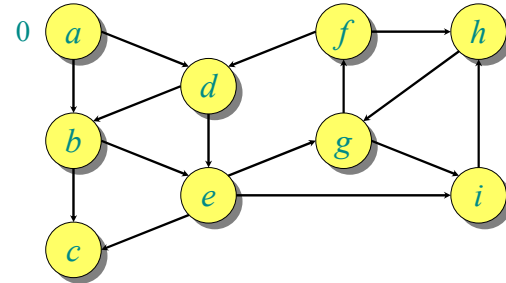
Example of breadth-first search



Q:



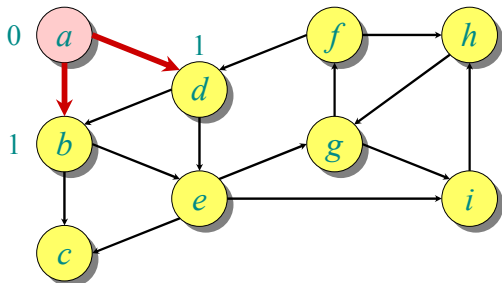
Example of breadth-first search



Q: a



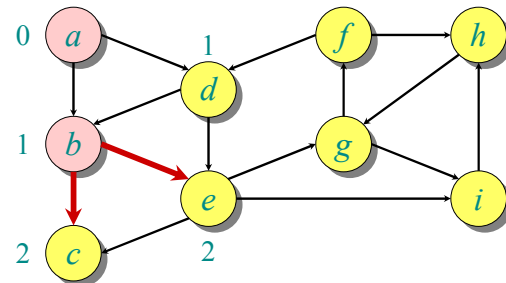
Example of breadth-first search



Q: a b d



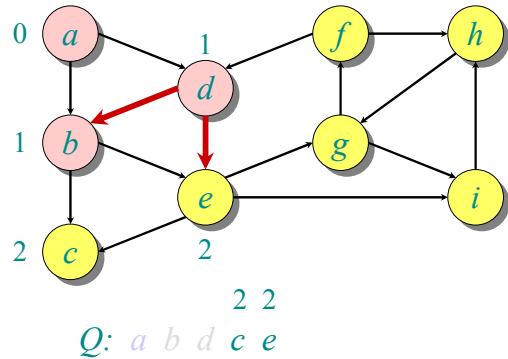
Example of breadth-first search



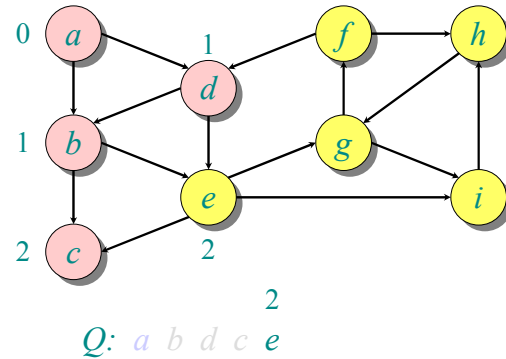
Q: a b d c e



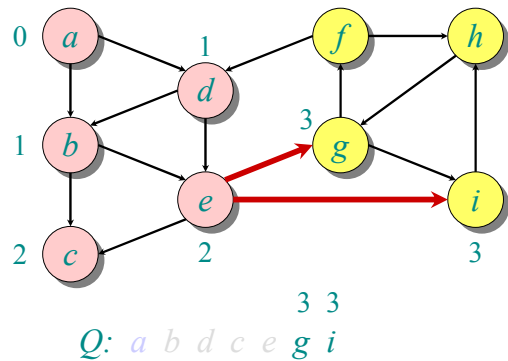
Example of breadth-first search



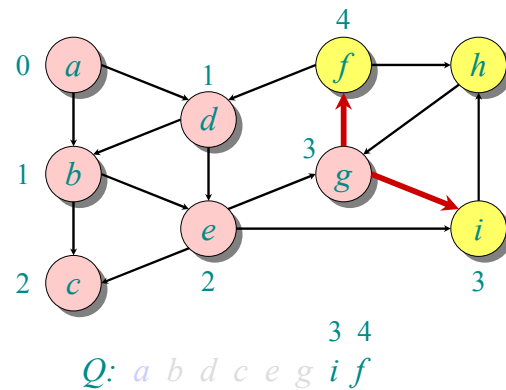
Example of breadth-first search



Example of breadth-first search

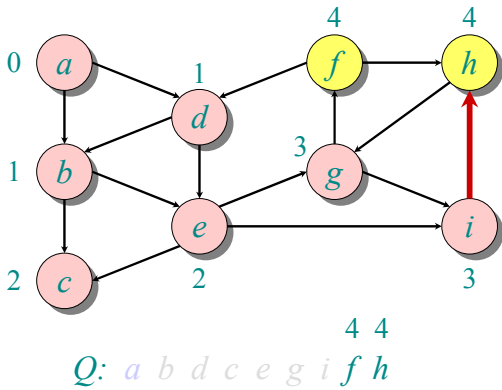


Example of breadth-first search

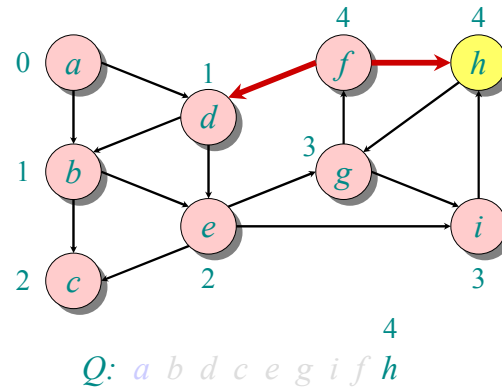




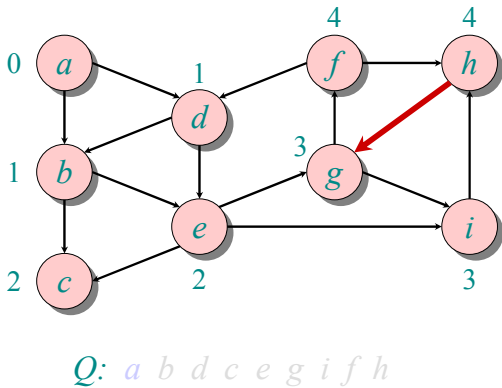
Example of breadth-first search



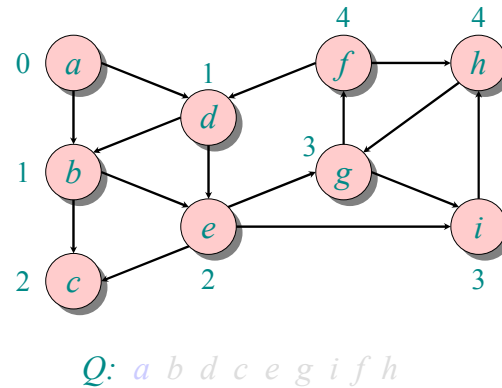
Example of breadth-first search



Example of breadth-first search



Example of breadth-first search





Correctness of BFS

```

while  $Q \neq \emptyset$ 
  do  $u \leftarrow \text{DEQUEUE}(Q)$ 
    for each  $v \in \text{Adj}[u]$ 
      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 $d[v] = d[u] + 1$.

4/8/08

CS 5633 Analysis of Algorithms

41



How to find the actual shortest paths?

Store a predecessor tree:

```

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

```

4/8/08

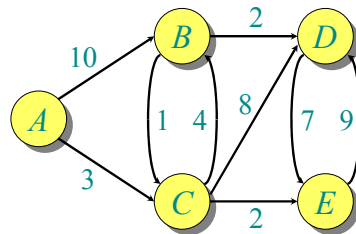
CS 5633 Analysis of Algorithms

42



Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



```

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

```

4/8/08

CS 5633 Analysis of Algorithms

43

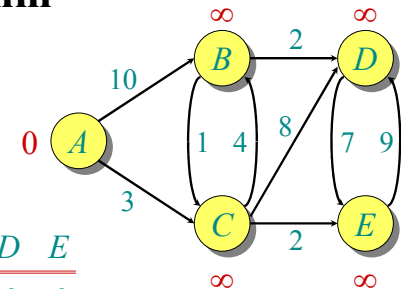


Example of Dijkstra's algorithm

Initialize:

$S: \{\}$

$Q: \underline{A \ B \ C \ D \ E}$
 $\quad 0 \ \infty \ \infty \ \infty \ \infty$



```

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

```

4/8/08

CS 5633 Analysis of Algorithms

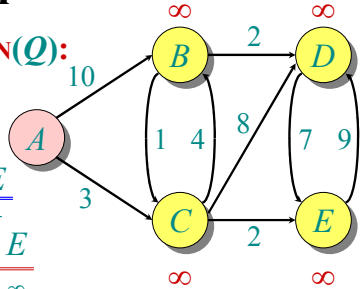
44



Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

S:	{ A }	0				
π :	A	B	C	D	E	
	-	-	-	-	-	
Q:	A	B	C	D	E	
	0	∞	∞	∞	∞	



```

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)
       $\pi[v] \leftarrow u$ 

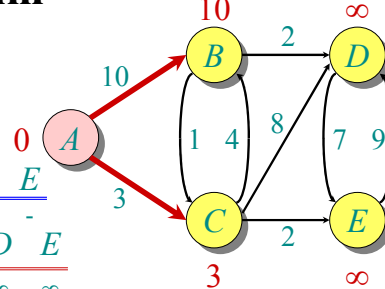
```



Example of Dijkstra's algorithm

Relax all edges leaving A:

S:	{ A }	0				
π :	A	B	C	D	E	
	-	-	-	-	-	
Q:	A	B	C	D	E	
	0	∞	∞	∞	∞	
		10	3	-	-	



```

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)
       $\pi[v] \leftarrow u$ 

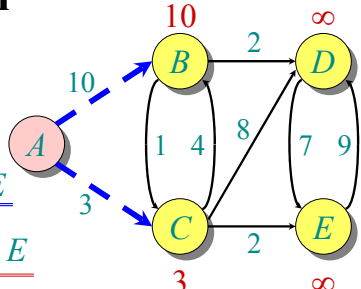
```



Example of Dijkstra's algorithm

Relax all edges leaving A:

S:	{ A }	0				
π :	A	B	C	D	E	
	-	A	A	-	-	
Q:	A	B	C	D	E	
	0	∞	∞	∞	∞	
		10	3	-	-	



```

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)
       $\pi[v] \leftarrow u$ 

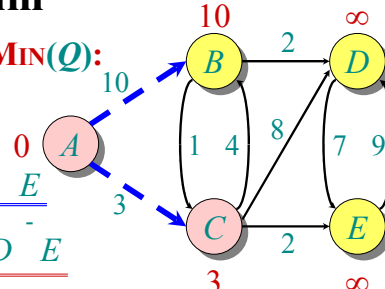
```



Example of Dijkstra's algorithm

"C" ← EXTRACT-MIN(Q):

S:	{ A, C }	0				
π :	A	B	C	D	E	
	-	A	A	-	-	
Q:	A	B	C	D	E	
	0	∞	3	∞	∞	
		10		-	-	



```

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)
       $\pi[v] \leftarrow u$ 

```



Example of Dijkstra's algorithm

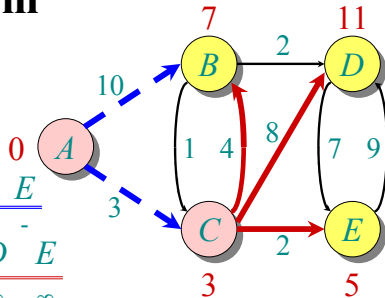
Relax all edges leaving C:

S: {A, C}

π : A B C D E

Q: A B C D E

0	∞	∞	∞	∞
	10	3	-	-
	7		11	5



```

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)
      π[v] ← u
  
```



Example of Dijkstra's algorithm

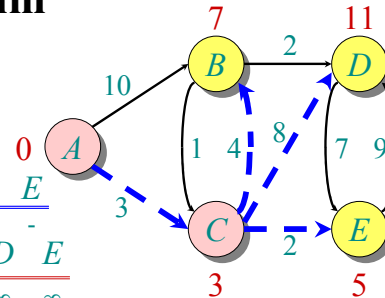
Relax all edges leaving C:

S: {A, C}

π : A B C D E

Q: A B C D E

0	∞	∞	∞	∞
	10	3	-	-
	7		11	5



```

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)
      π[v] ← u
  
```



Example of Dijkstra's algorithm

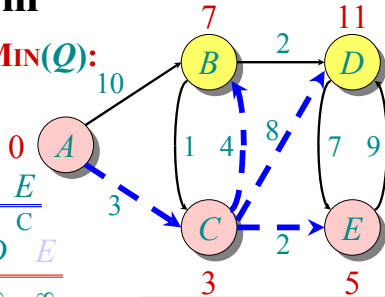
"E" ← EXTRACT-MIN(Q):

S: {A, C, E}

π : A B C D E

Q: A B C D E

0	∞	∞	∞	∞
	10	3	-	-
	7		11	5



```

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)
      π[v] ← u
  
```



Example of Dijkstra's algorithm

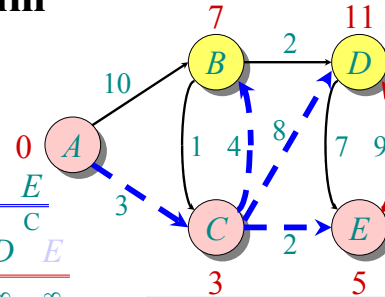
Relax all edges leaving E:

S: {A, C, E}

π : A B C D E

Q: A B C D E

0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	



```

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)
      π[v] ← u
  
```



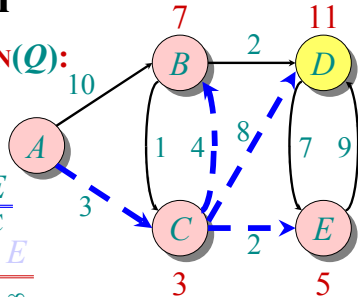
Example of Dijkstra's algorithm

"B" ← EXTRACT-MIN(Q):

S: {A, C, E, B} 0

π : $\begin{matrix} A & B & C & D & E \\ - & C & A & C & C \end{matrix}$

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



```

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)
      π[v] ← u
  
```



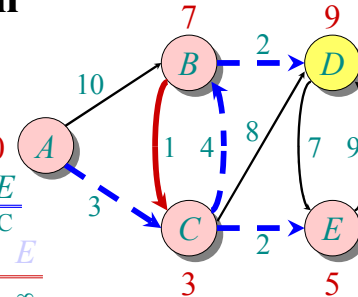
Example of Dijkstra's algorithm

Relax all edges leaving B:

S: {A, C, E, B} 0

π : $\begin{matrix} A & B & C & D & E \\ - & C & A & B & C \end{matrix}$

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



```

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)
      π[v] ← u
  
```



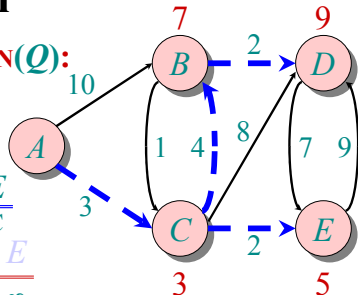
Example of Dijkstra's algorithm

"D" ← EXTRACT-MIN(Q):

S: {A, C, E, B, D} 0

π : $\begin{matrix} A & B & C & D & E \\ - & C & A & C & C \end{matrix}$

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



```

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)
      π[v] ← u
  
```