Graphs + Shortest Paths

David Kauchak
cs302
Spring 2013

Admin

Tree BFS

```
TreeBFS($T$)
Enqueue($Q$, Root($T$))
while !Empty($Q$)
        $v \leftarrow$ Dequeue($Q$)
        Visit($v$)
        for all $c \in$ Children($v$)
                Enqueue($Q, c$)
```

Running time of Tree BFS

Adjacency list
- How many times does it visit each vertex?
- How many times is each edge traversed?
- O(|V|+|E|)

Adjacency matrix
- For each vertex visited, how much work is done?
- O(|V|^2)

```
TreeBFS($T$)
Enqueue($Q$, Root($T$))
while !Empty($Q$)
        $v \leftarrow$ Dequeue($Q$)
        Visit($v$)
        for all $c \in$ Children($v$)
                Enqueue($Q, c$)
```

BFS Recursively

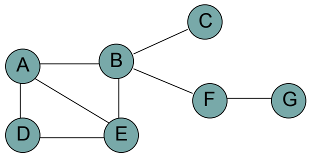
Hard to do!

```
TreeBFS($T$)
    Enqueue($Q$, Root($T$))
    while !Empty($Q$)
        $v \leftarrow$ Dequeue($Q$)
        Visit($v$)
        for all $c \in$ Children($v$)
            Enqueue($Q$, $c$)
```

BFS for graphs

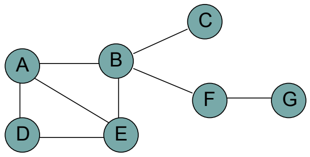
What needs to change for graphs?

Need to make sure we don't visit a node multiple times



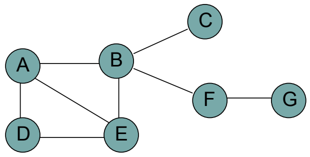
```
BFS($G, s$)
    for each $v \in V$
        $dist[v] = \infty$
    $dist[s] = 0$
    Enqueue($Q, s$)
    while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
            if $dist[v] = \infty$
                Enqueue($Q, v$)
                $dist[v] \leftarrow dist[u] + 1$
```

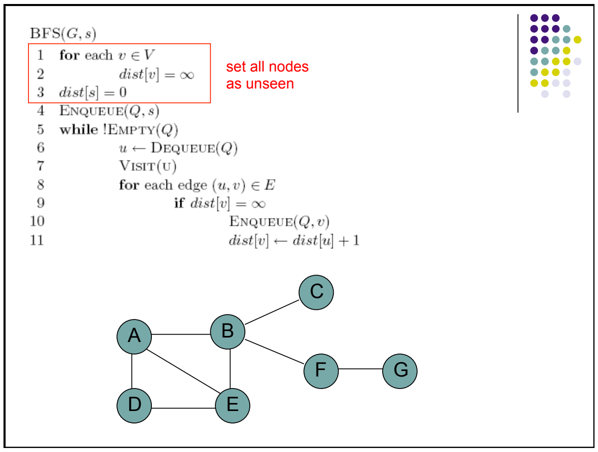
distance variable keeps track of how far from the starting node and whether we've seen the node yet



```
BFS($G, s$)
    for each $v \in V$
        $dist[v] = \infty$
    $dist[s] = 0$
    Enqueue($Q, s$)
    while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
            if $dist[v] = \infty$
                Enqueue($Q, v$)
                $dist[v] \leftarrow dist[u] + 1$
```

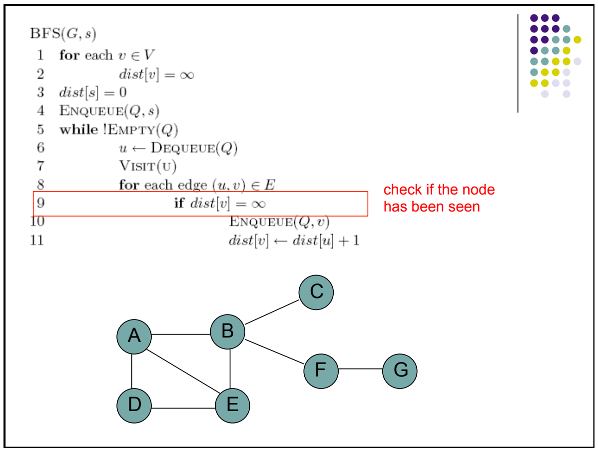
```
TreeBFS($T$)
    Enqueue($Q$, Root($T$))
    while !Empty($Q$)
        $v \leftarrow$ Dequeue($Q$)
        Visit($v$)
        for all $c \in$ Children($v$)
            Enqueue($Q$, $c$)
```





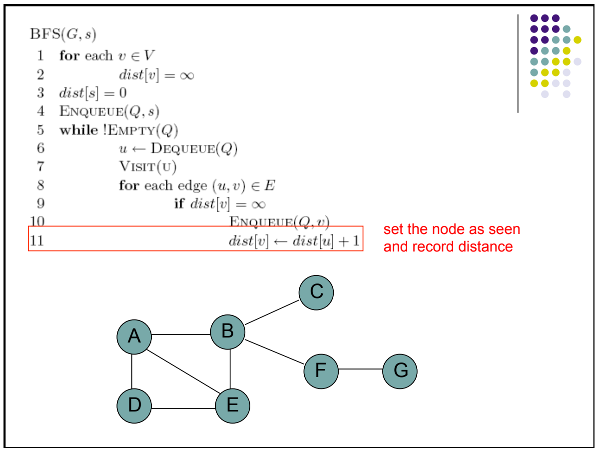
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

set all nodes as unseen



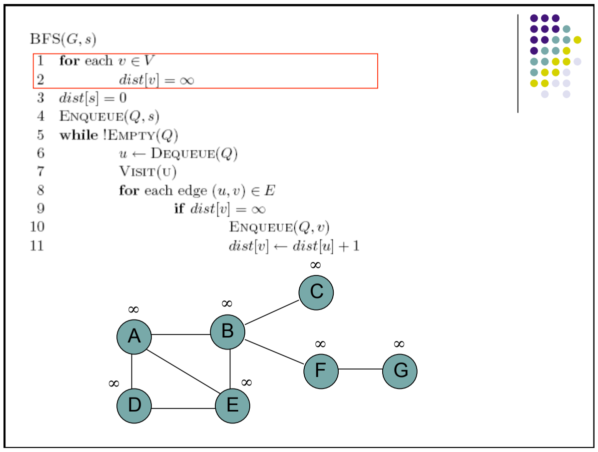
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

check if the node has been seen

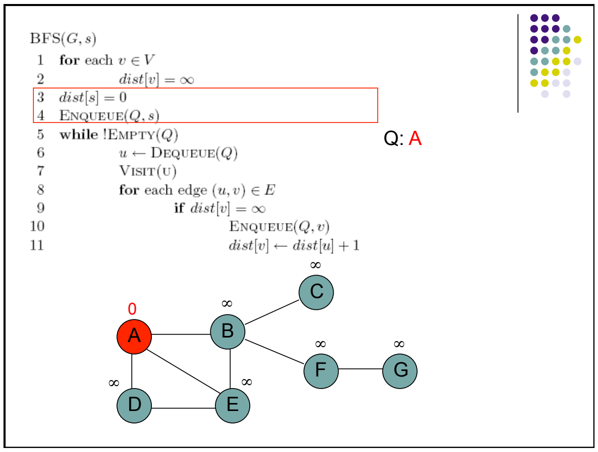


```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

set the node as seen and record distance

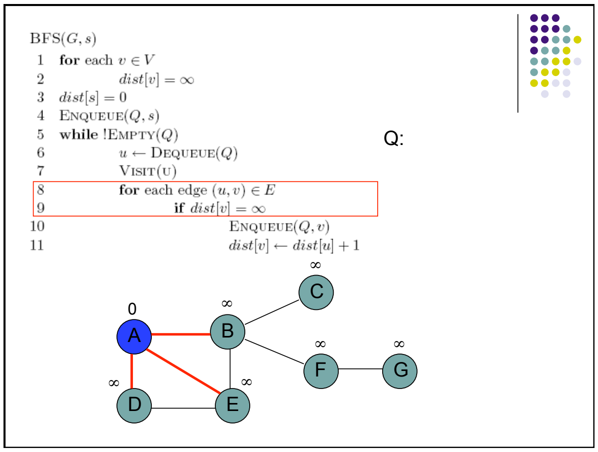


```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```



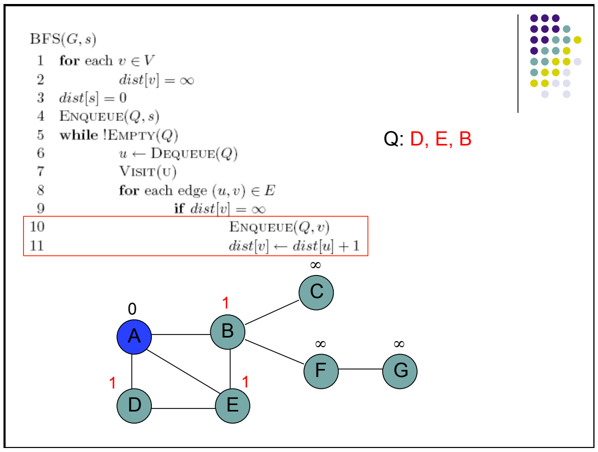
```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(U)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```

Q: A



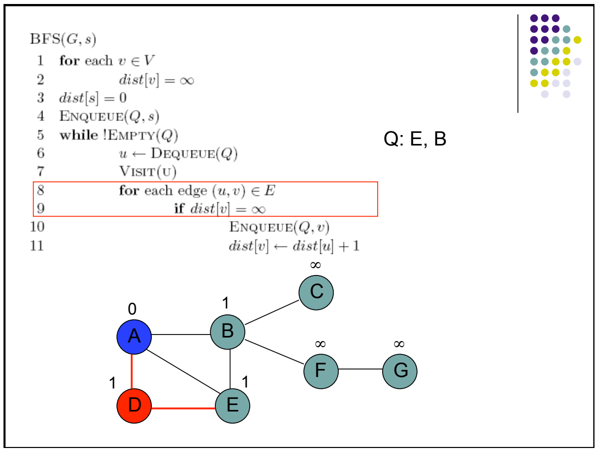
```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(U)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```

Q:



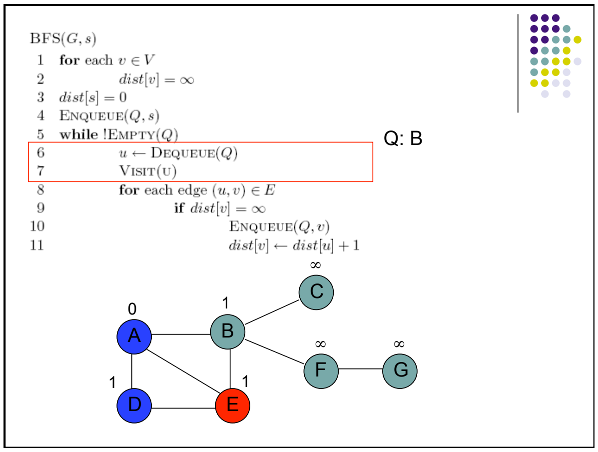
```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(U)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```

Q: D, E, B



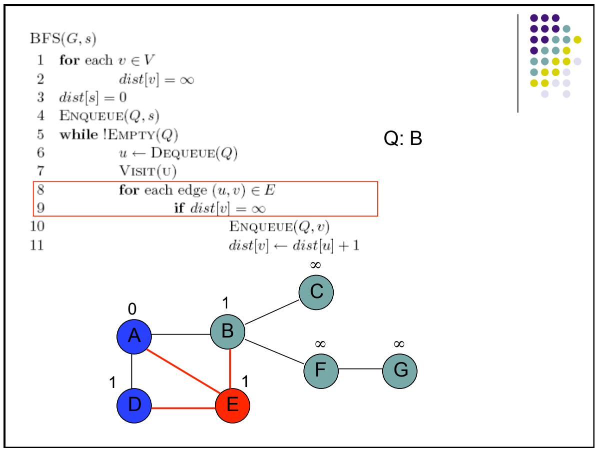
```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(U)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```

Q: E, B



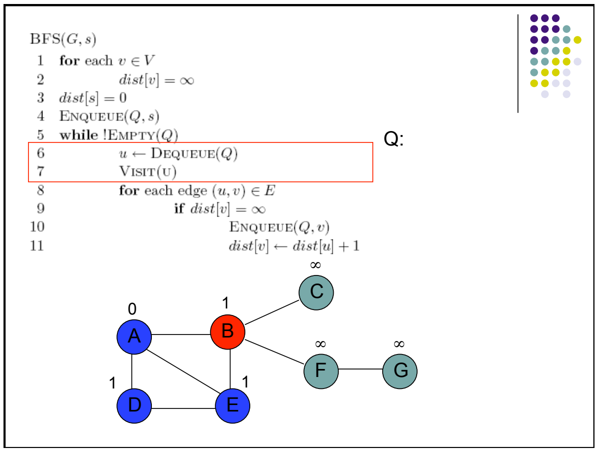
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
Enqueue($Q, s$)
while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        Enqueue($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Q: B



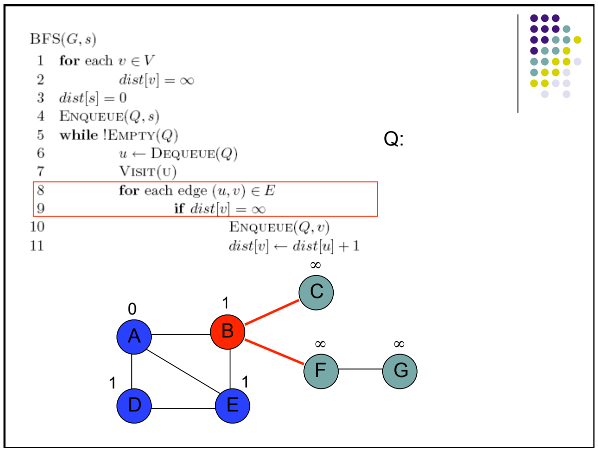
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
Enqueue($Q, s$)
while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        Enqueue($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Q: B



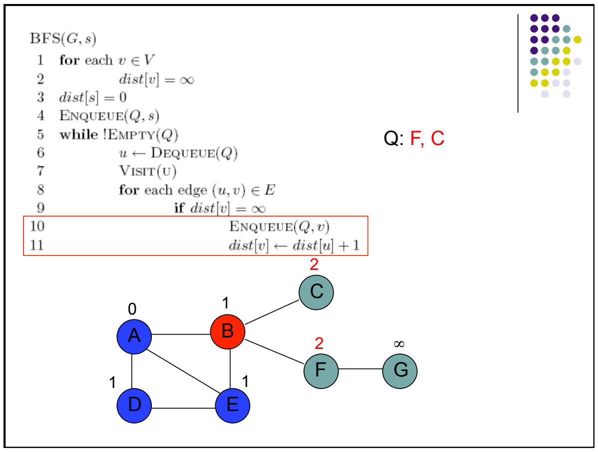
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
Enqueue($Q, s$)
while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        Enqueue($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Q:



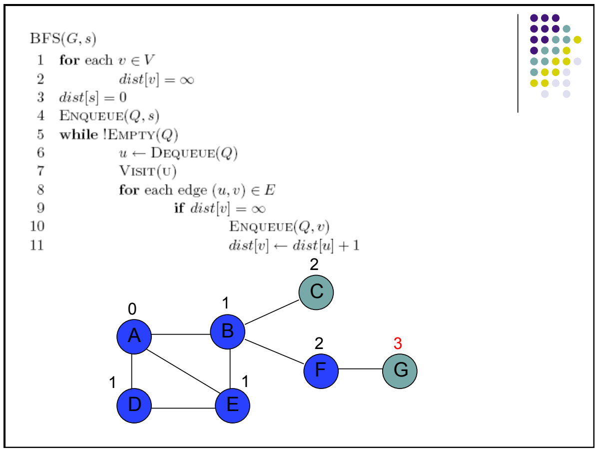
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
Enqueue($Q, s$)
while !Empty($Q$)
        $u \leftarrow$ Dequeue($Q$)
        Visit(u)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        Enqueue($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Q:

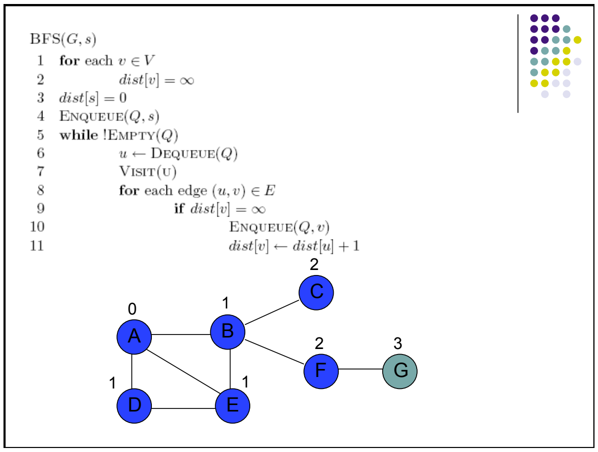


```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

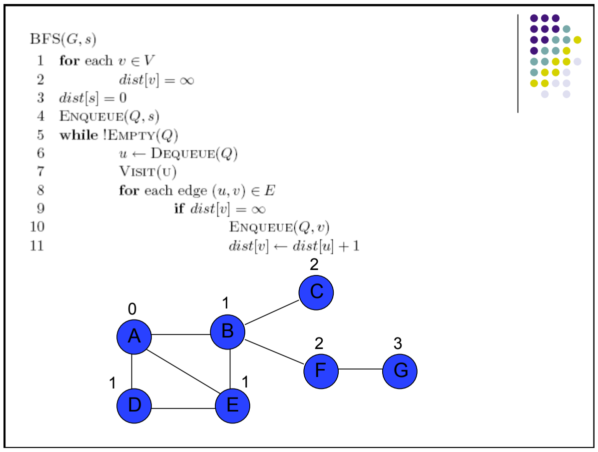
Q: F, C



```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```



```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

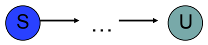


```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Is BFS correct?

Does it visit all nodes reachable from the starting node? Can you prove it?

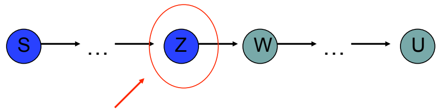
Assume we "miss" some node 'u', i.e. a path exists, but we don't visit 'u'



Is BFS correct?

Does it visit all nodes reachable from the starting node? Can you prove it?

Find the last node along the path to 'u' that was visited

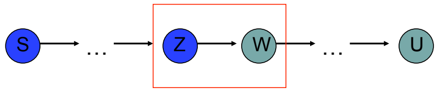


why do we know that such a node exists?

Is BFS correct?

Does it visit all nodes reachable from the starting node? Can you prove it?

We visited 'z' but not 'w', which is a contradiction, given the pseudocode



contradiction

Is BFS correct?

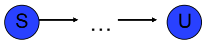
Does it correctly label each node with the shortest distance from the starting node?

```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT($u$)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Is BFS correct?

Does it correctly label each node with the shortest distance from the starting node?

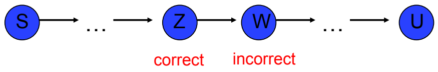
Assume the algorithm labels a node with a longer distance. Call that node 'u'



Is BFS correct?

Does it correctly label each node with the shortest distance from the starting node?

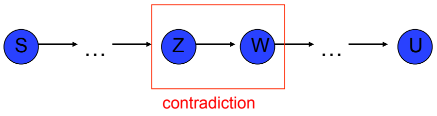
Find the last node in the path with the correct distance



Is BFS correct?

Does it correctly label each node with the shortest distance from the starting node?

Find the last node in the path with the correct distance



Runtime of BFS

Nothing changed over our analysis of TreeBFS

```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

```
TREEBFS($T$)
ENQUEUE($Q$, ROOT($T$))
while !EMPTY($Q$)
        $v \leftarrow$ DEQUEUE($Q$)
        VISIT($v$)
        for all $c \in$ CHILDREN($v$)
                ENQUEUE($Q, c$)
```

Runtime of BFS

Adjacency list: O(|V| + |E|)

Adjacency matrix: O($|V|^2$)

```
BFS($G, s$)
  for each $v \in V$
      $dist[v] = \infty$
  $dist[s] = 0$
  ENQUEUE($Q, s$)
  while !EMPTY($Q$)
      $u \leftarrow$ DEQUEUE($Q$)
      VISIT(u)
      for each edge $(u, v) \in E$
          if $dist[v] = \infty$
              ENQUEUE($Q, v$)
              $dist[v] \leftarrow dist[u] + 1$
```

Depth First Search (DFS)

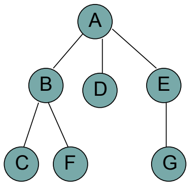
```
TREEDFS($T$)
  PUSH($S$, ROOT($T$))
  while !EMPTY($S$)
      $v \leftarrow$ POP($S$)
      VISIT($v$)
      for all $c \in$ CHILDREN($v$)
          PUSH($S, c$)
```

Depth First Search (DFS)

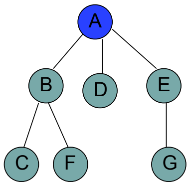
```
TREEDFS($T$)
  PUSH($S$, ROOT($T$))
  while !EMPTY($S$)
      $v \leftarrow$ POP($S$)
      VISIT($v$)
      for all $c \in$ CHILDREN($v$)
          PUSH($S, c$)
```

```
TREEBFS($T$)
  ENQUEUE($Q$, ROOT($T$))
  while !EMPTY($Q$)
      $v \leftarrow$ DEQUEUE($Q$)
      VISIT($v$)
      for all $c \in$ CHILDREN($v$)
          ENQUEUE($Q, c$)
```

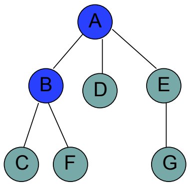
Tree DFS



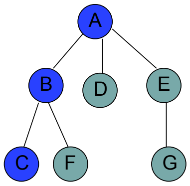
Tree DFS



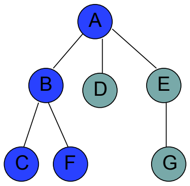
Tree DFS



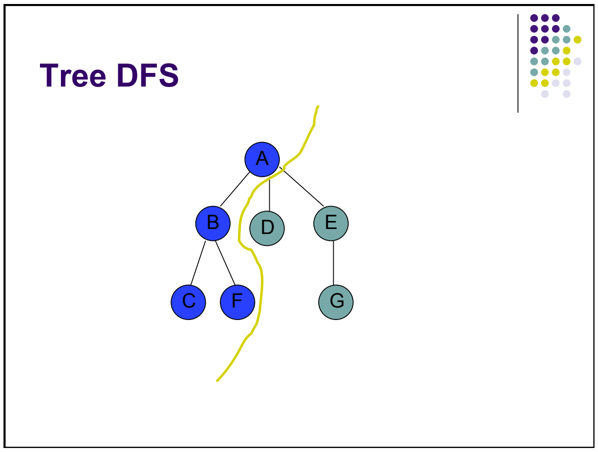
Tree DFS



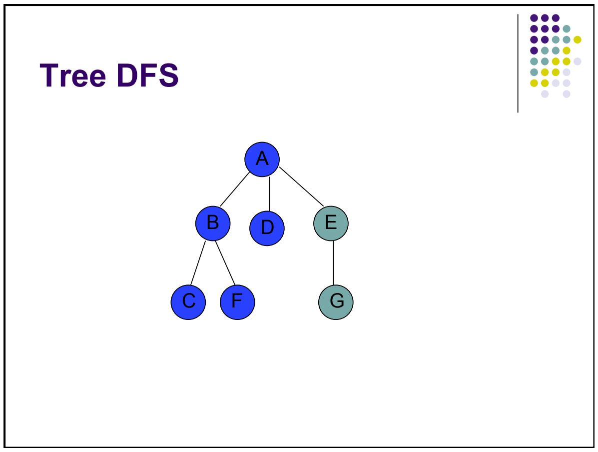
Tree DFS



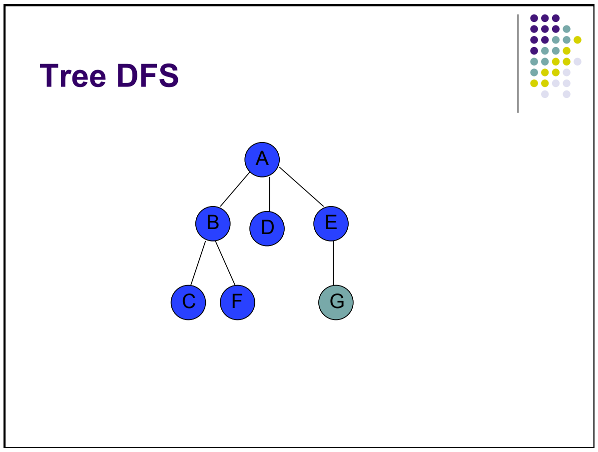
Frontier?



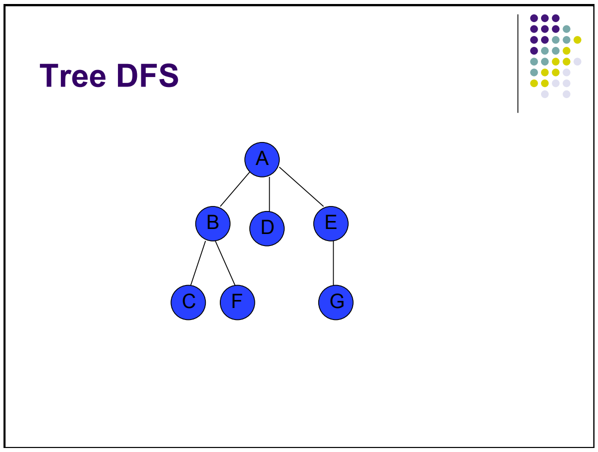
Tree DFS



Tree DFS



Tree DFS



Tree DFS

DFS on graphs

```
DFS($G$)
for all $v \in V$
        $visited[u] \leftarrow false$
for all $v \in V$
        if $!visited[v]$
                DFS-Visit($v$)

DFS-Visit($u$)
$visited[u] \leftarrow true$
PreVisit(u)
for all edges $(u,v) \in E$
        if $!visited[v]$
                DFS-Visit($v$)
PostVisit(u)
```

DFS on graphs

```
DFS($G$)
for all $v \in V$
        $visited[u] \leftarrow false$
for all $v \in V$
        if $!visited[v]$
                DFS-Visit($v$)

DFS-Visit($u$)
$visited[u] \leftarrow true$
PreVisit(u)
for all edges $(u,v) \in E$
        if $!visited[v]$
                DFS-Visit($v$)
PostVisit(u)
```

mark all nodes as not visited

DFS on graphs

```
DFS($G$)
for all $v \in V$
        $visited[u] \leftarrow false$
for all $v \in V$
        if $!visited[v]$
                DFS-Visit($v$)

DFS-Visit($u$)
$visited[u] \leftarrow true$
PreVisit(u)
for all edges $(u,v) \in E$
        if $!visited[v]$
                DFS-Visit($v$)
PostVisit(u)
```

until **all** nodes have been visited repeatedly call DFS-Visit

DFS on graphs

```
DFS($G$)
for all $v \in V$
        $visited[u] \leftarrow false$
for all $v \in V$
        if $!visited[v]$
                DFS-Visit($v$)

DFS-Visit($u$)
$visited[u] \leftarrow true$
PreVisit(u)
for all edges $(u,v) \in E$
        if $!visited[v]$
                DFS-Visit($v$)
PostVisit(u)
```

What happened to the stack?

```
TreeDFS($T$)
Push($S$, Root($T$))
while !Empty($S$)
        $v \leftarrow$ Pop($S$)
        Visit($v$)
        for all $c \in$ Children($v$)
                Push($S, c$)
```

What does DFS do?

Finds connected components

Each call to DFS-Visit from DFS starts exploring a new set of connected components

Helps us understand the structure/connectedness of a graph

Is DFS correct?

Does DFS visit all of the nodes in a graph?

```
DFS($G$)
for all $v \in V$
        $visited[u] \leftarrow false$
for all $v \in V$
        if $!visited[v]$
                DFS-VISIT($v$)
```

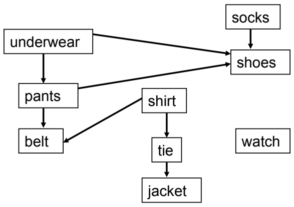
Running time?

Like BFS

- Visits each node exactly once
- Processes each edge exactly twice (for an undirected graph)
- O(|V|+|E|)

DAGs

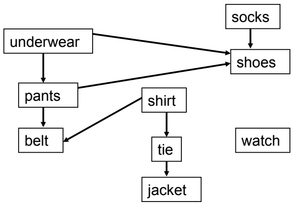
Can represent dependency graphs



Topological sort

A linear ordering of all the vertices such that for all edges (u,v) ∈ E, u appears before v in the ordering

An ordering of the nodes that “obeys” the dependencies, i.e. an activity can’t happen until it’s dependent activities have happened



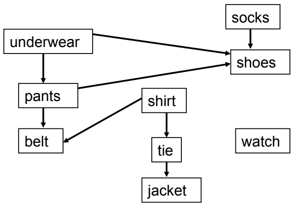
- watch
- underwear
- pants
- shirt
- belt
- tie
- socks
- shoes
- jacket

Topological sort

```
TOPOLOGICAL-SORT1(G)
    Find a node $v$ with no incoming edges
    Delete $v$ from $G$
    Add $v$ to linked list
    TOPOLOGICAL-SORT1(G)
```

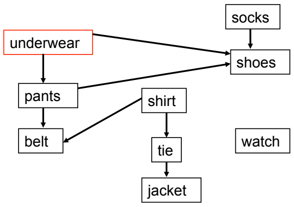
Topological sort

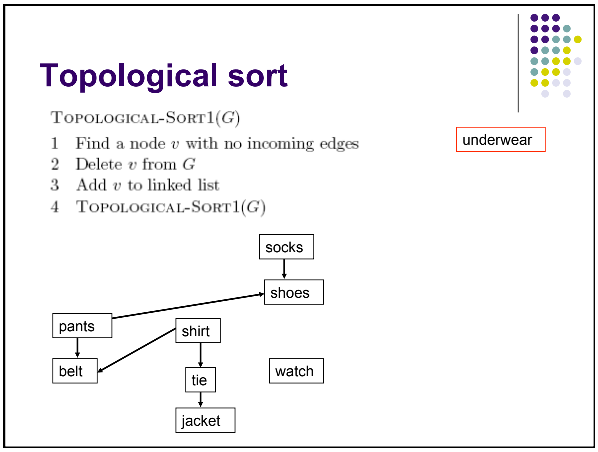
```
TOPOLOGICAL-SORT1(G)
    Find a node $v$ with no incoming edges
    Delete $v$ from $G$
    Add $v$ to linked list
    TOPOLOGICAL-SORT1(G)
```



Topological sort

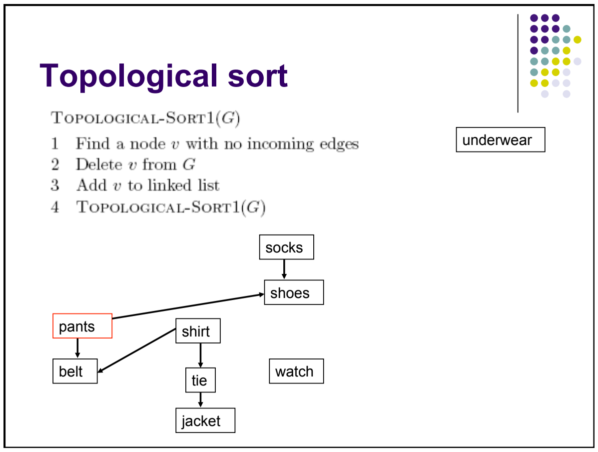
```
TOPOLOGICAL-SORT1(G)
    Find a node $v$ with no incoming edges
    Delete $v$ from $G$
    Add $v$ to linked list
    TOPOLOGICAL-SORT1(G)
```





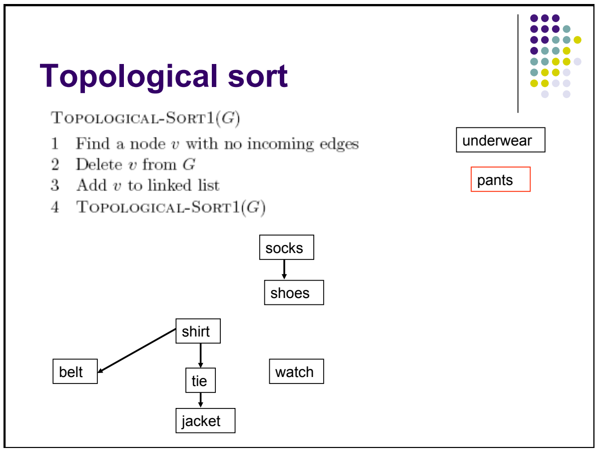
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



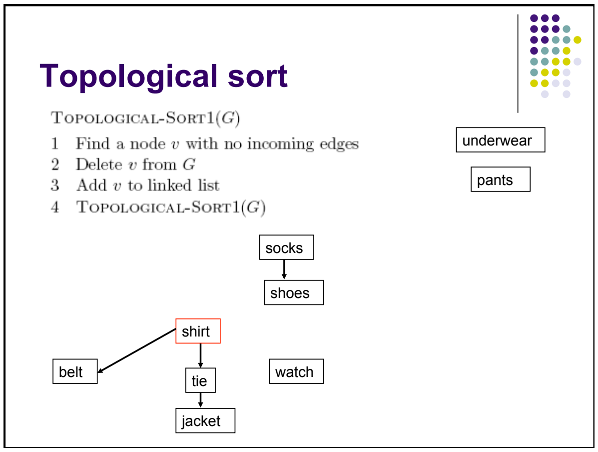
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



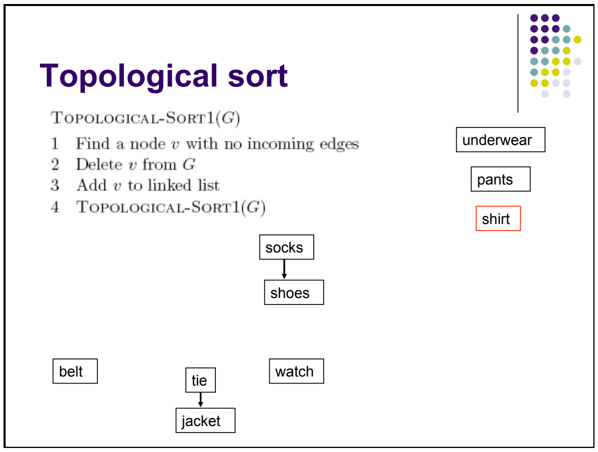
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



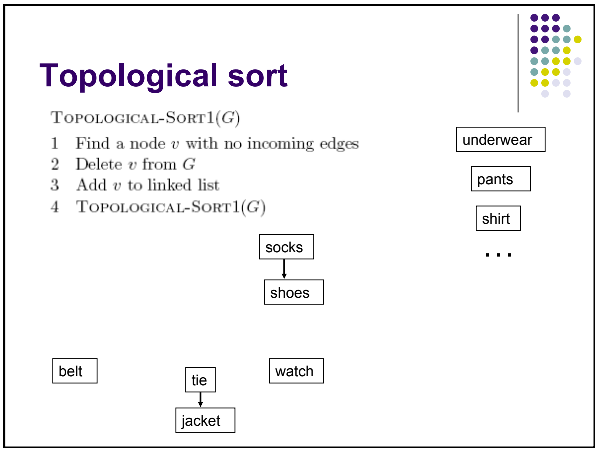
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



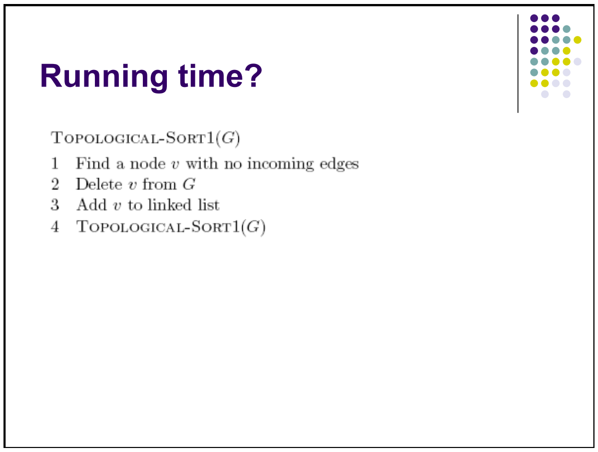
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



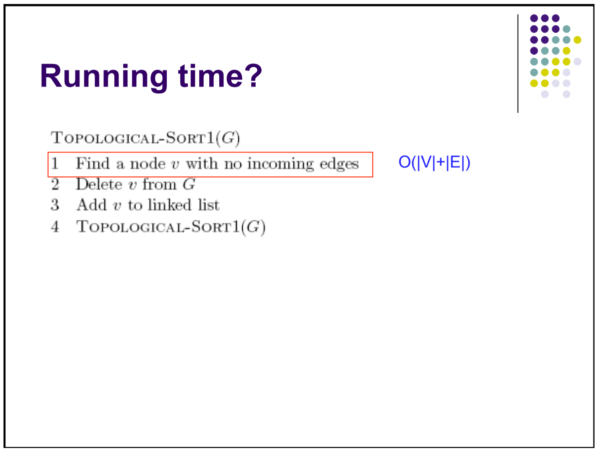
Topological sort

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



Running time?

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```



Running time?

```
TOPOLOGICAL-SORT1($G$)
Find a node $v$ with no incoming edges      O(|V|+|E|)
Delete $v$ from $G$
Add $v$ to linked list
TOPOLOGICAL-SORT1($G$)
```

Running time?

```
Topological-Sort1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
Topological-Sort1(G)
```

O(E) overall

Running time?

```
Topological-Sort1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
Topological-Sort1(G)
```

How many calls? |V|

Running time?

```
Topological-Sort1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
Topological-Sort1(G)
```

Overall running time?

$O(|V|^2+|V|\,|E|)$

Can we do better?

```
Topological-Sort1(G)
Find a node v with no incoming edges
Delete v from G
Add v to linked list
Topological-Sort1(G)
```

Topological sort 2

```
TOPOLOGICAL-SORT2($G$)
for all edges $(u, v) \in E$
    $active[v] \leftarrow active[v] + 1$
for all $v \in V$
    if $active[v] = 0$
        ENQUEUE($S, v$)
while !EMPTY($S$)
    $u \leftarrow$ DEQUEUE($S$)
    add $u$ to linked list
    for each edge $(u, v) \in E$
        $active[v] \leftarrow active[v] - 1$
        if $active[v] = 0$
            ENQUEUE($S, v$)
```

Topological sort 2

```
TOPOLOGICAL-SORT2($G$)
for all edges $(u, v) \in E$
    $active[v] \leftarrow active[v] + 1$
for all $v \in V$
    if $active[v] = 0$
        ENQUEUE($S, v$)
while !EMPTY($S$)
    $u \leftarrow$ DEQUEUE($S$)
    add $u$ to linked list
    for each edge $(u, v) \in E$
        $active[v] \leftarrow active[v] - 1$
        if $active[v] = 0$
            ENQUEUE($S, v$)
```

Topological sort 2

```
TOPOLOGICAL-SORT2($G$)
for all edges $(u, v) \in E$
    $active[v] \leftarrow active[v] + 1$
for all $v \in V$
    if $active[v] = 0$
        ENQUEUE($S, v$)
while !EMPTY($S$)
    $u \leftarrow$ DEQUEUE($S$)
    add $u$ to linked list
    for each edge $(u, v) \in E$
        $active[v] \leftarrow active[v] - 1$
        if $active[v] = 0$
            ENQUEUE($S, v$)
```

Topological sort 2

```
TOPOLOGICAL-SORT2($G$)
for all edges $(u, v) \in E$
    $active[v] \leftarrow active[v] + 1$
for all $v \in V$
    if $active[v] = 0$
        ENQUEUE($S, v$)
while !EMPTY($S$)
    $u \leftarrow$ DEQUEUE($S$)
    add $u$ to linked list
    for each edge $(u, v) \in E$
        $active[v] \leftarrow active[v] - 1$
        if $active[v] = 0$
            ENQUEUE($S, v$)
```

Running time?

How many times do we process each node?

How many times do we process each edge?

O(|V| + |E|)

```
TOPOLOGICAL-SORT2(G)
for all edges $(u, v) \in E$
        $active[v] \leftarrow active[v] + 1$
for all $v \in V$
        if $active[v] = 0$
                ENQUEUE$(S, v)$
while !EMPTY$(S)$
        $u \leftarrow$ DEQUEUE$(S)$
        add $u$ to linked list
        for each edge $(u, v) \in E$
                $active[v] \leftarrow active[v] - 1$
                if $active[v] = 0$
                        ENQUEUE$(S, v)$
```

Connectedness

Given an undirected graph, for every node u ∈ V, can we reach all other nodes in the graph?

Run BFS or DFS-Visit (one pass) and mark nodes as we visit them. If we visit all nodes, return true, otherwise false.

Running time: O(|V| + |E|)

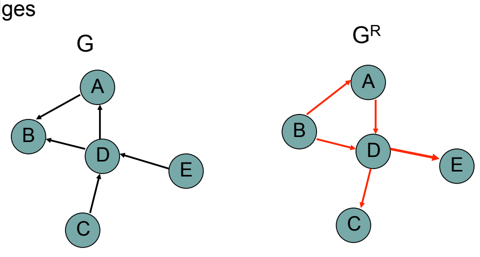
Strongly connected

Given a directed graph, can we reach any node v from any other node u?

Ideas?

Transpose of a graph

Given a graph G, we can calculate the transpose of a graph G^R by reversing the direction of all the edges



Running time to calculate G^R? O(|V| + |E|)

Strongly connected

```
STRONGLY-CONNECTED(G)
Run DFS or BFS from some node u
if not all nodes are visited
        return false
Create graph G^R by reversing all edge dirctions
Run DFS or BFS on G^R from node u
if not all nodes are visited
        return false
return true
```

Is it correct?

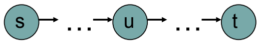
What do we know after the first pass?

- Starting at u, we can reach every node

What do we know after the second pass?

- All nodes can reach u. Why?
- We can get from u to every node in G^R, therefore, if we reverse the edges (i.e. G), then we have a path from every node to u

Which means that any node can reach any other node. Given any two nodes s and t we can create a path through u



Runtime?

```
STRONGLY-CONNECTED(G)
Run DFS or BFS from some node u                     O(|V| + |E|)
if not all nodes are visited                        O(|V|)
        return false
Create graph G^R by reversing all edge dirctions    O(|V| + |E|)
Run DFS or BFS on G^R from node u                   O(|V| + |E|)
if not all nodes are visited                        O(|V|)
        return false
return true
```

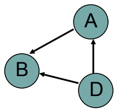
O(|V| + |E|)

Detecting cycles

Undirected graph

- BFS or DFS. If we reach a node we've seen already, then we've found a cycle

Directed graph



have to be careful

Detecting cycles

Undirected graph

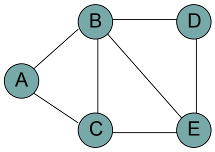
- BFS or DFS. If we reach a node we've seen already, then we've found a cycle

Directed graph

- Call TopologicalSort
- If the length of the list returned ≠ |V| then a cycle exists

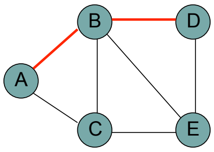
Shortest paths

What is the shortest path from a to d?



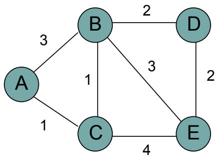
Shortest paths

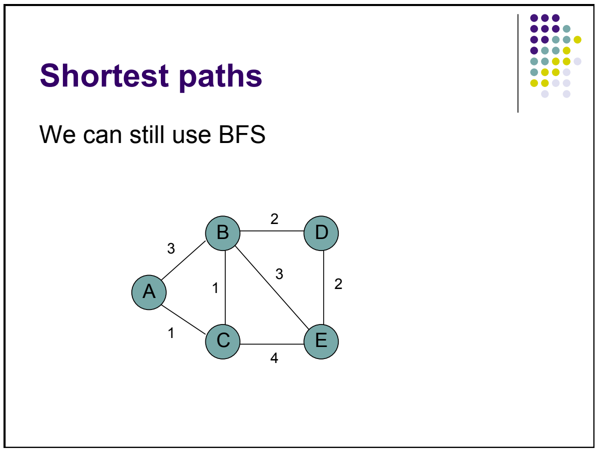
BFS



Shortest paths

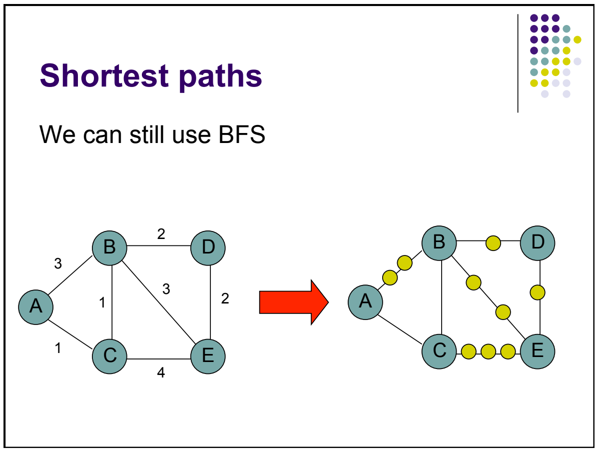
What is the shortest path from a to d?





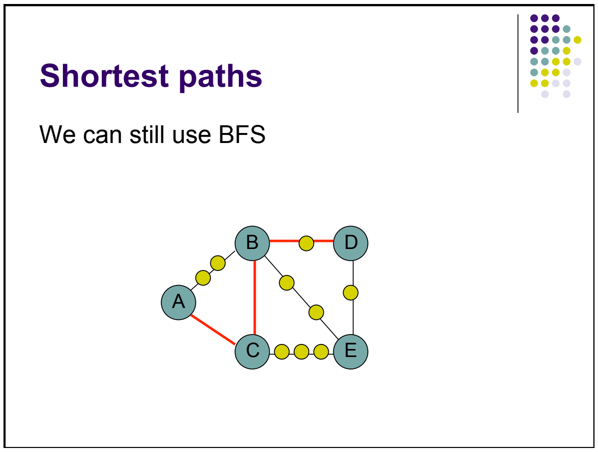
Shortest paths

We can still use BFS



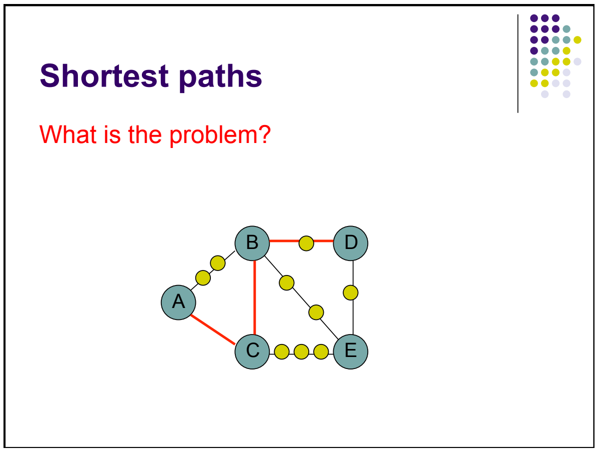
Shortest paths

We can still use BFS



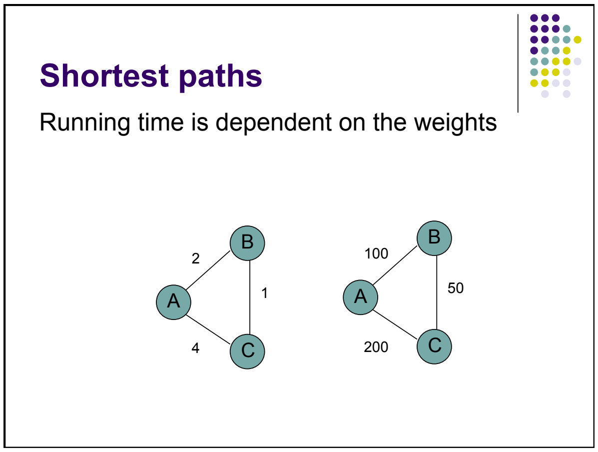
Shortest paths

We can still use BFS



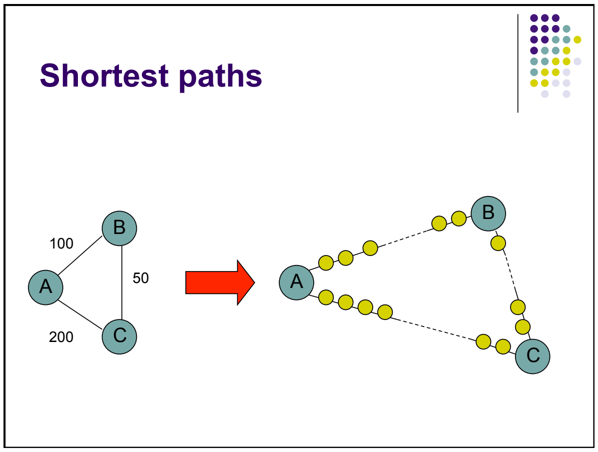
Shortest paths

What is the problem?

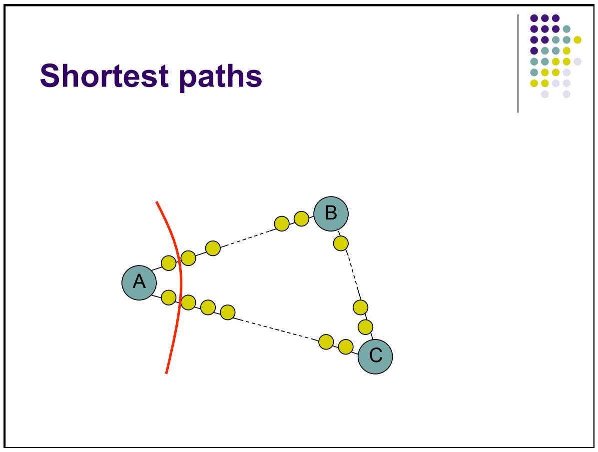


Shortest paths

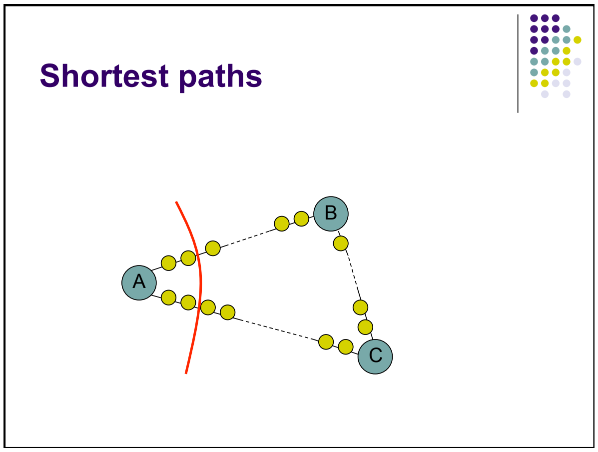
Running time is dependent on the weights



Shortest paths



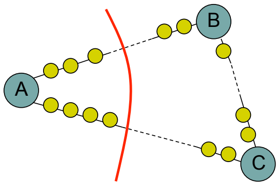
Shortest paths



Shortest paths

Shortest paths

Nothing will change as we expand the frontier until we've gone out 100 levels



Dijkstra's algorithm

```
DIJKSTRA($G, s$)
for all $v \in V$
        $dist[v] \leftarrow \infty$
        $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
        $u \leftarrow$ EXTRACTMIN($Q$)
        for all edges $(u, v) \in E$
                if $dist[v] > dist[u] + w(u, v)$
                        $dist[v] \leftarrow dist[u] + w(u, v)$
                        DECREASEKEY($Q, v, dist[v]$)
                        $prev[v] \leftarrow u$
```

Dijkstra's algorithm

```
DIJKSTRA($G, s$)
for all $v \in V$
        $dist[v] \leftarrow \infty$
        $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
        $u \leftarrow$ EXTRACTMIN($Q$)
        for all edges $(u, v) \in E$
                if $dist[v] > dist[u] + w(u, v)$
                        $dist[v] \leftarrow dist[u] + w(u, v)$
                        DECREASEKEY($Q, v, dist[v]$)
                        $prev[v] \leftarrow u$
```

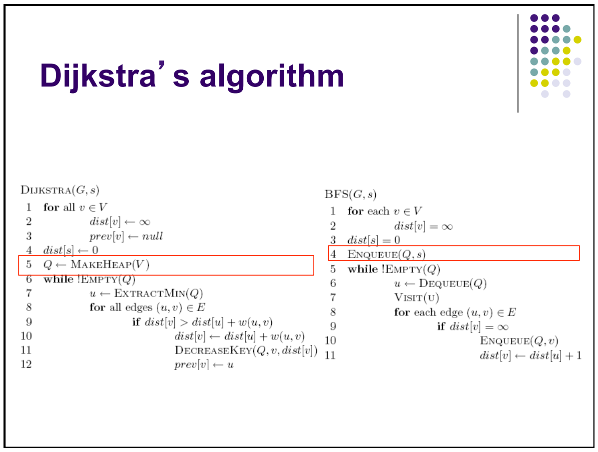
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```

Dijkstra's algorithm

prev keeps track of
the shortest path

```
DIJKSTRA($G, s$)
for all $v \in V$
        $dist[v] \leftarrow \infty$
        $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
        $u \leftarrow$ EXTRACTMIN($Q$)
        for all edges $(u, v) \in E$
                if $dist[v] > dist[u] + w(u, v)$
                        $dist[v] \leftarrow dist[u] + w(u, v)$
                        DECREASEKEY($Q, v, dist[v]$)
                        $prev[v] \leftarrow u$
```

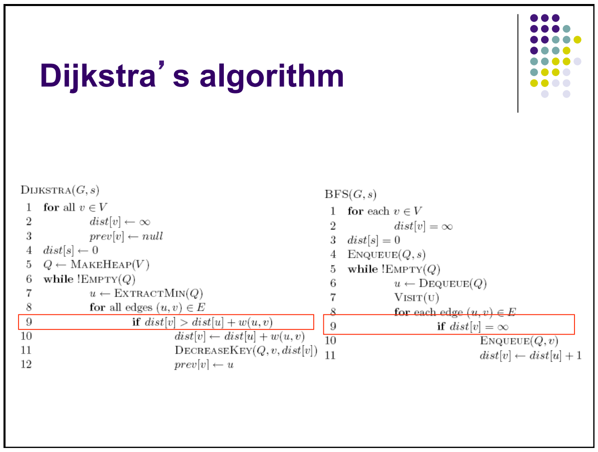
```
BFS($G, s$)
for each $v \in V$
        $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
        $u \leftarrow$ DEQUEUE($Q$)
        VISIT(U)
        for each edge $(u, v) \in E$
                if $dist[v] = \infty$
                        ENQUEUE($Q, v$)
                        $dist[v] \leftarrow dist[u] + 1$
```



Dijkstra’s algorithm

```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

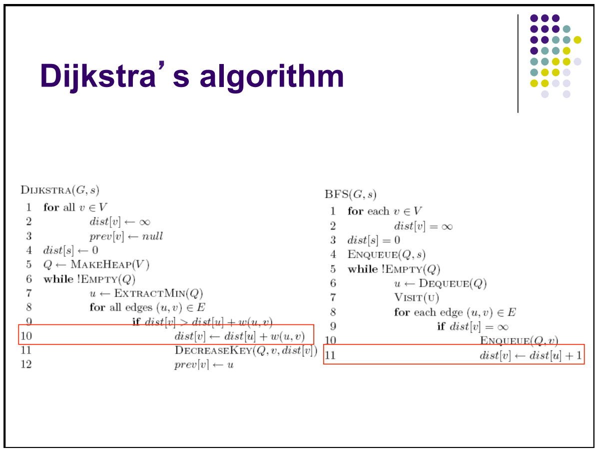
```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(u)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```



Dijkstra’s algorithm

```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(u)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```



Dijkstra’s algorithm

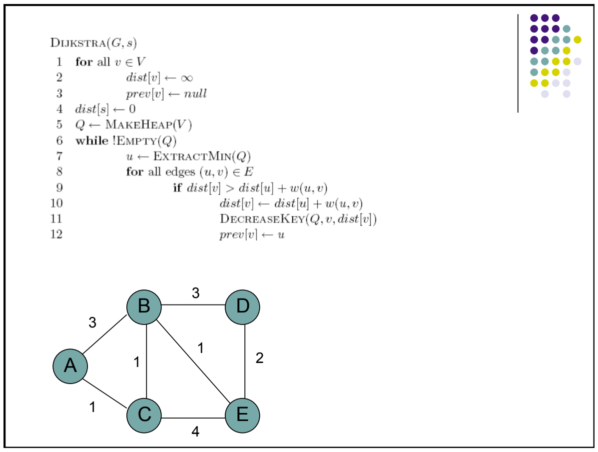
```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

```
BFS($G, s$)
for each $v \in V$
    $dist[v] = \infty$
$dist[s] = 0$
ENQUEUE($Q, s$)
while !EMPTY($Q$)
    $u \leftarrow$ DEQUEUE($Q$)
    VISIT(u)
    for each edge $(u, v) \in E$
        if $dist[v] = \infty$
            ENQUEUE($Q, v$)
            $dist[v] \leftarrow dist[u] + 1$
```

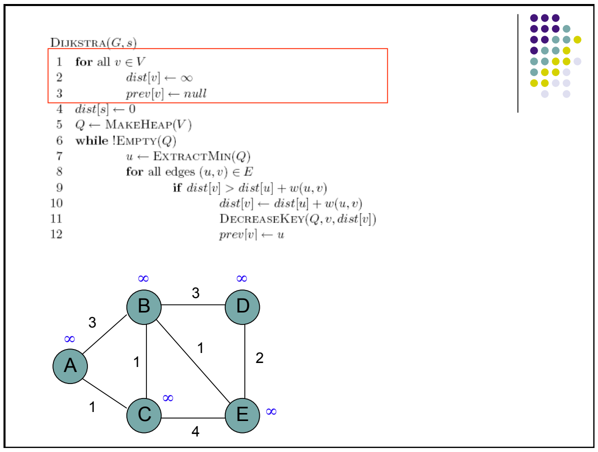
Single source shortest paths

All of the shortest path algorithms we’ll look at today are call “single source shortest paths” algorithms

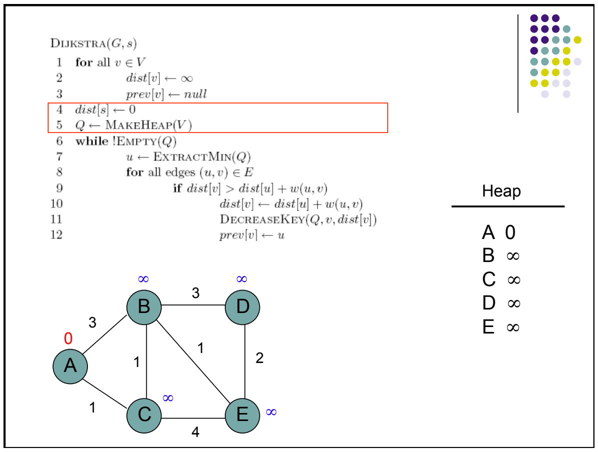
Why?



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```



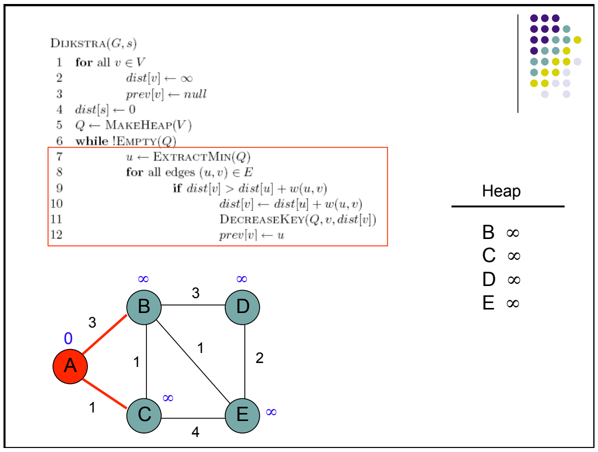
```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

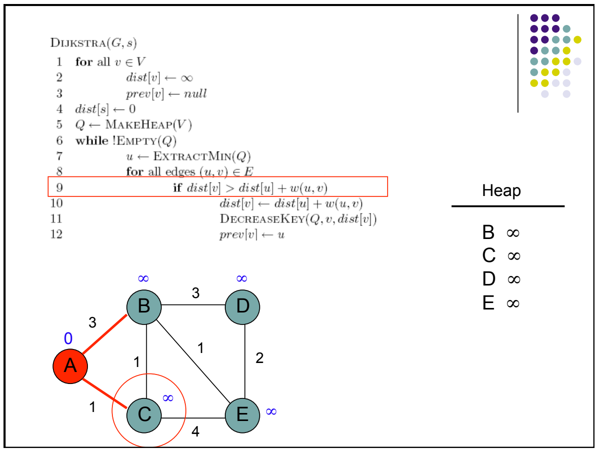
A	0
B	∞
C	∞
D	∞
E	∞



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

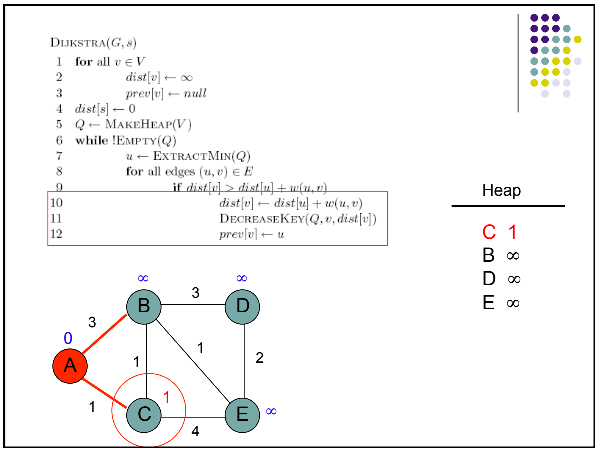
B	∞
C	∞
D	∞
E	∞



```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

Heap

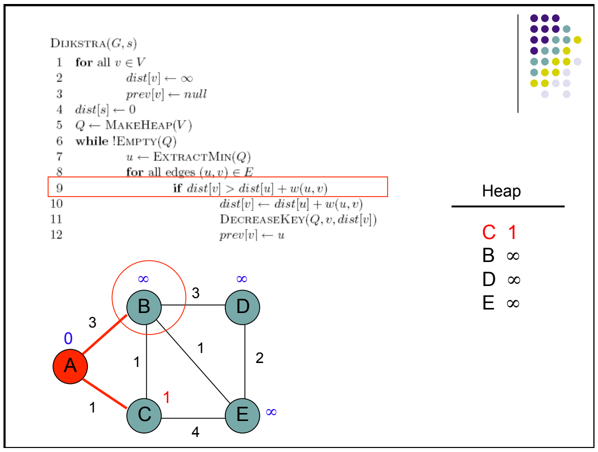
B ∞
C ∞
D ∞
E ∞



```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

Heap

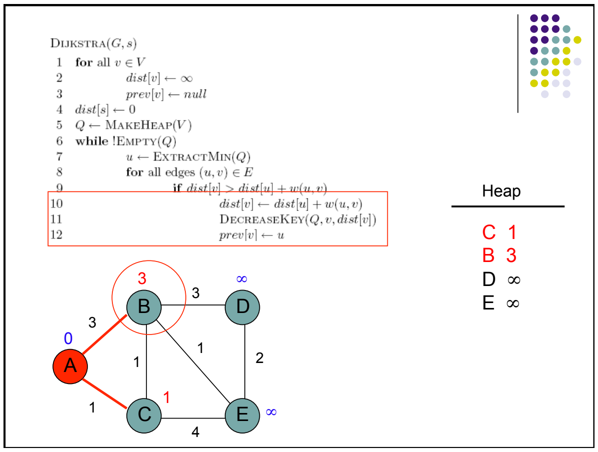
C 1
B ∞
D ∞
E ∞



```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

Heap

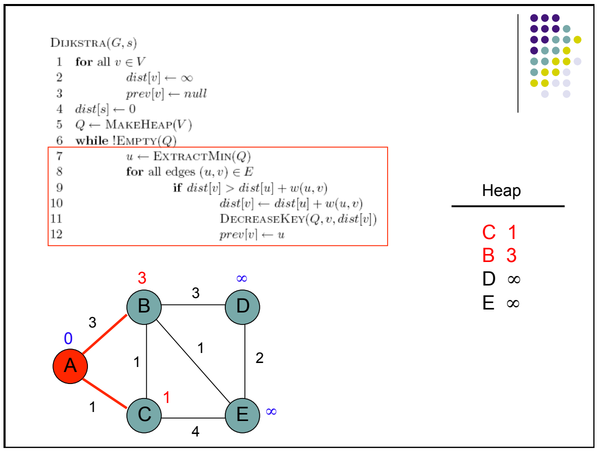
C 1
B ∞
D ∞
E ∞



```
DIJKSTRA(G, s)
for all v ∈ V
    dist[v] ← ∞
    prev[v] ← null
dist[s] ← 0
Q ← MAKEHEAP(V)
while !EMPTY(Q)
    u ← EXTRACTMIN(Q)
    for all edges (u, v) ∈ E
        if dist[v] > dist[u] + w(u, v)
            dist[v] ← dist[u] + w(u, v)
            DECREASEKEY(Q, v, dist[v])
            prev[v] ← u
```

Heap

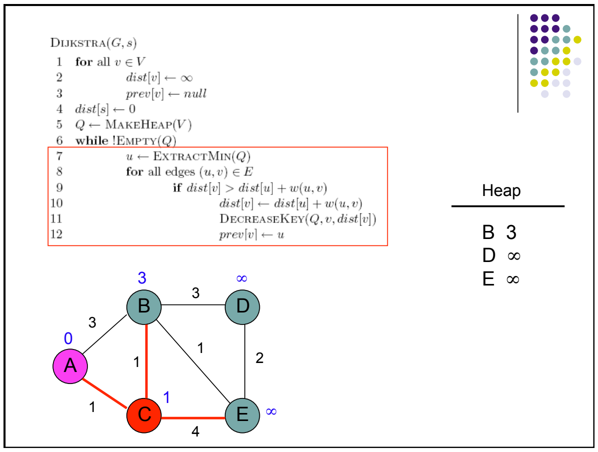
C 1
B 3
D ∞
E ∞



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

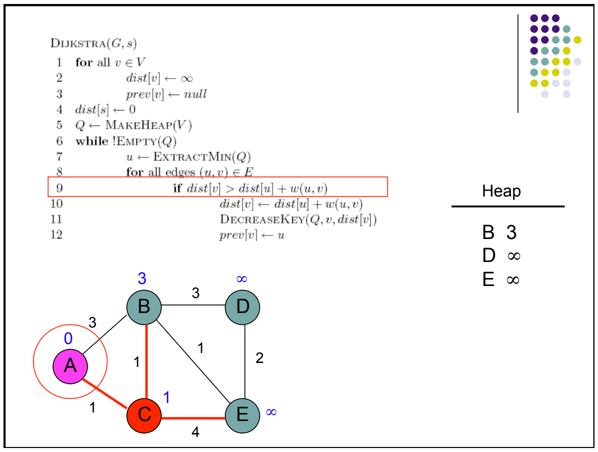
C 1
B 3
D ∞
E ∞



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

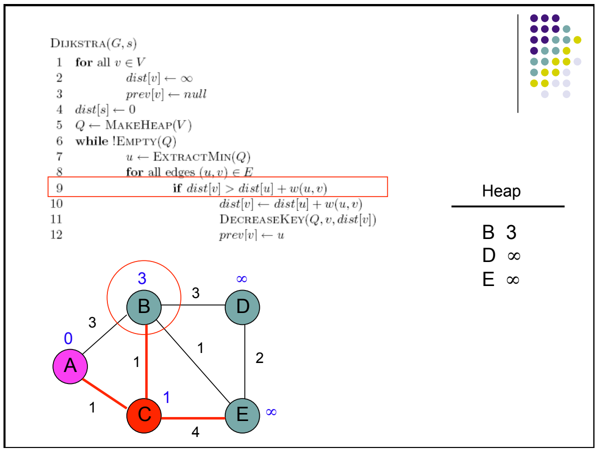
B 3
D ∞
E ∞



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

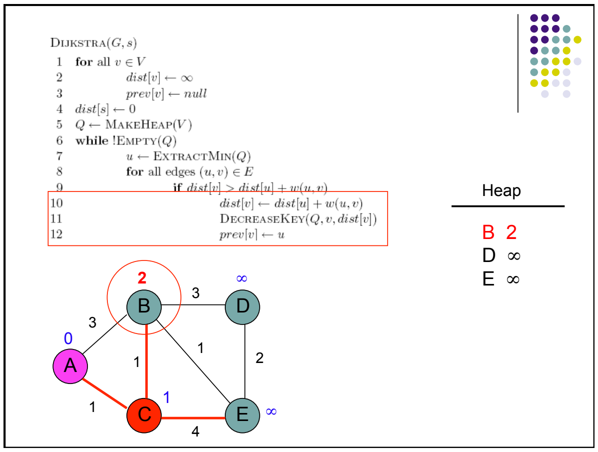
B 3
D ∞
E ∞



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

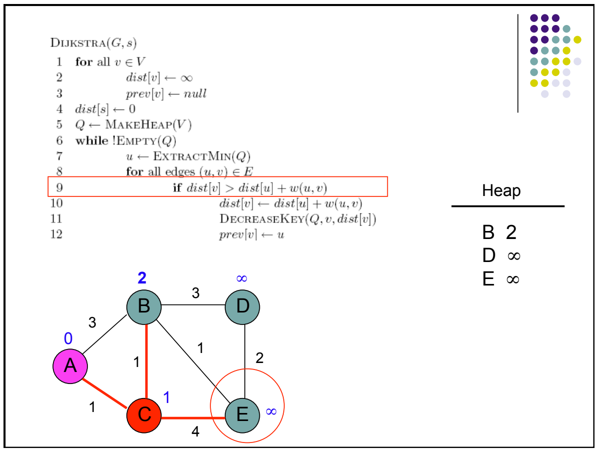
B 3
D ∞
E ∞



```
DIJKSTRA($G, s$)
  for all $v \in V$
      $dist[v] \leftarrow \infty$
      $prev[v] \leftarrow null$
  $dist[s] \leftarrow 0$
  $Q \leftarrow$ MAKEHEAP($V$)
  while !EMPTY($Q$)
      $u \leftarrow$ EXTRACTMIN($Q$)
      for all edges $(u, v) \in E$
          if $dist[v] > dist[u] + w(u, v)$
              $dist[v] \leftarrow dist[u] + w(u, v)$
              DECREASEKEY($Q, v, dist[v]$)
              $prev[v] \leftarrow u$
```

Heap

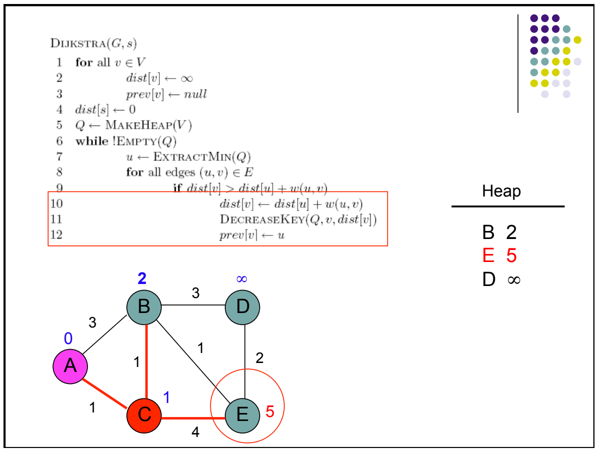
B 2
D ∞
E ∞



```
DIJKSTRA($G, s$)
  for all $v \in V$
      $dist[v] \leftarrow \infty$
      $prev[v] \leftarrow null$
  $dist[s] \leftarrow 0$
  $Q \leftarrow$ MAKEHEAP($V$)
  while !EMPTY($Q$)
      $u \leftarrow$ EXTRACTMIN($Q$)
      for all edges $(u, v) \in E$
          if $dist[v] > dist[u] + w(u, v)$
              $dist[v] \leftarrow dist[u] + w(u, v)$
              DECREASEKEY($Q, v, dist[v]$)
              $prev[v] \leftarrow u$
```

Heap

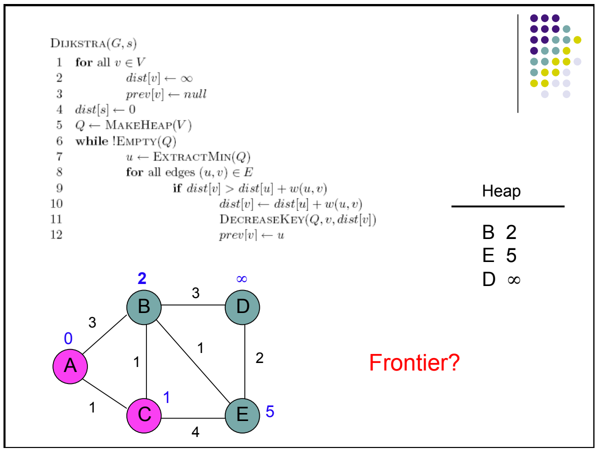
B 2
D ∞
E ∞



```
DIJKSTRA($G, s$)
  for all $v \in V$
      $dist[v] \leftarrow \infty$
      $prev[v] \leftarrow null$
  $dist[s] \leftarrow 0$
  $Q \leftarrow$ MAKEHEAP($V$)
  while !EMPTY($Q$)
      $u \leftarrow$ EXTRACTMIN($Q$)
      for all edges $(u, v) \in E$
          if $dist[v] > dist[u] + w(u, v)$
              $dist[v] \leftarrow dist[u] + w(u, v)$
              DECREASEKEY($Q, v, dist[v]$)
              $prev[v] \leftarrow u$
```

Heap

B 2
E 5
D ∞

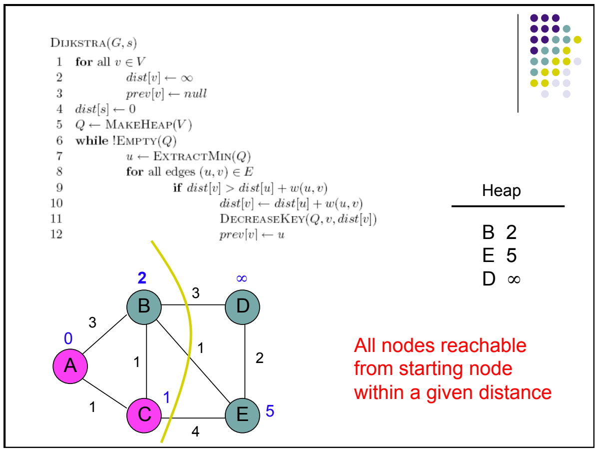


```
DIJKSTRA($G, s$)
  for all $v \in V$
      $dist[v] \leftarrow \infty$
      $prev[v] \leftarrow null$
  $dist[s] \leftarrow 0$
  $Q \leftarrow$ MAKEHEAP($V$)
  while !EMPTY($Q$)
      $u \leftarrow$ EXTRACTMIN($Q$)
      for all edges $(u, v) \in E$
          if $dist[v] > dist[u] + w(u, v)$
              $dist[v] \leftarrow dist[u] + w(u, v)$
              DECREASEKEY($Q, v, dist[v]$)
              $prev[v] \leftarrow u$
```

Heap

B 2
E 5
D ∞

Frontier?

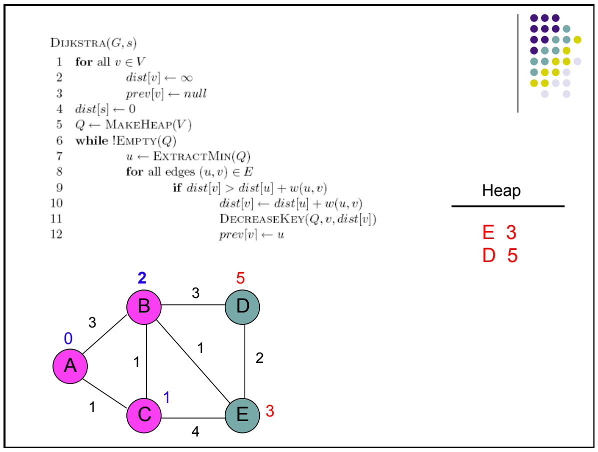


```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

B 2
E 5
D ∞

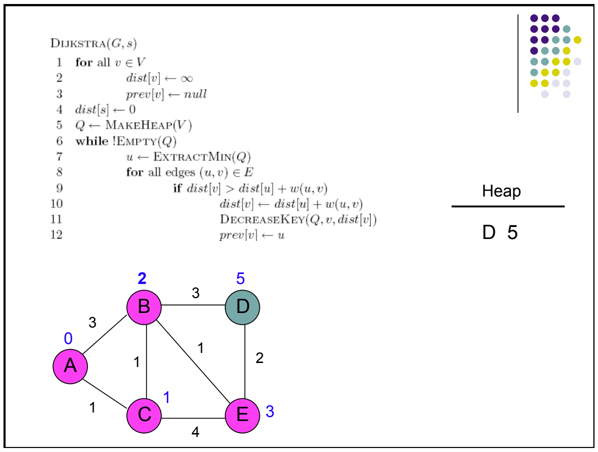
All nodes reachable from starting node within a given distance



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap

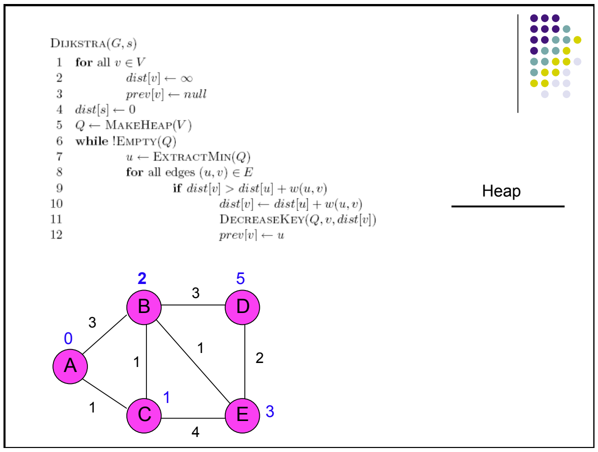
E 3
D 5



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

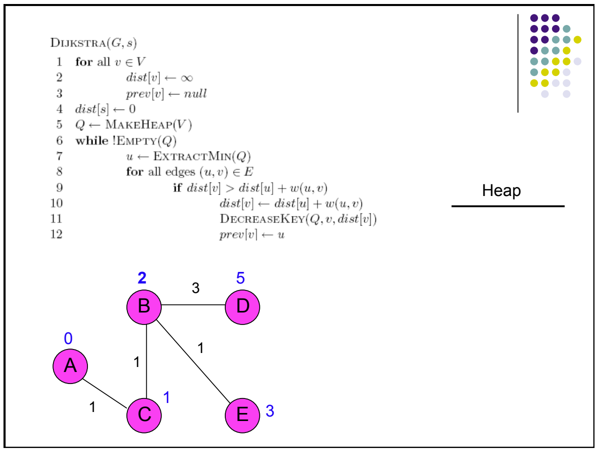
Heap

D 5



```
DIJKSTRA($G, s$)
for all $v \in V$
    $dist[v] \leftarrow \infty$
    $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MAKEHEAP($V$)
while !EMPTY($Q$)
    $u \leftarrow$ EXTRACTMIN($Q$)
    for all edges $(u, v) \in E$
        if $dist[v] > dist[u] + w(u, v)$
            $dist[v] \leftarrow dist[u] + w(u, v)$
            DECREASEKEY($Q, v, dist[v]$)
            $prev[v] \leftarrow u$
```

Heap



```
Dijkstra($G, s$)
for all $v \in V$
        $dist[v] \leftarrow \infty$
        $prev[v] \leftarrow null$
$dist[s] \leftarrow 0$
$Q \leftarrow$ MakeHeap($V$)
while !Empty($Q$)
        $u \leftarrow$ ExtractMin($Q$)
        for all edges $(u, v) \in E$
                if $dist[v] > dist[u] + w(u, v)$
                        $dist[v] \leftarrow dist[u] + w(u, v)$
                        DecreaseKey($Q, v, dist[v]$)
                        $prev[v] \leftarrow u$
```

Heap