Announcements

Announcements

- HW2 is due today
  - Any questions on HW2?
- HW3 has been posted
- If you have not yet signed up for an oral exam slot, do that today! (Check BB for link)
- Exam questions will be posted on March 7 (Sunday), due March 14
- Exams start March 15

Graphs

Depth First Search

```
procedure explore(G, v)
Input:   G = (V, E) is a graph; v ∈ V
Output:  visited(u) is set to true for
         all nodes u reachable from v

 previsit(v)
 visited(v) = true
 for each edge (v, u) ∈ E:
     if not visited(u):  explore(u)
 postvisit(v)
```

Ignore for now

What is the running time of Depth First Search?

DFS on Directed Graphs

- We can run DFS on directed graphs, making sure to only follow edges in their correct direction
- The starting node is the root of the DFS tree
- u is an ancestor of v if there is a path from u to v in the DFS tree. v is a descendant of u.
- u is the parent of v if there is a directed edge from u to v in the DFS tree. v is the child of u.

DFS on Directed Graphs

can't happen

u → v

back

- We can run DFS on directed graphs, making sure to only follow edges in their correct direction

A is the **root**

B is a **child** of A

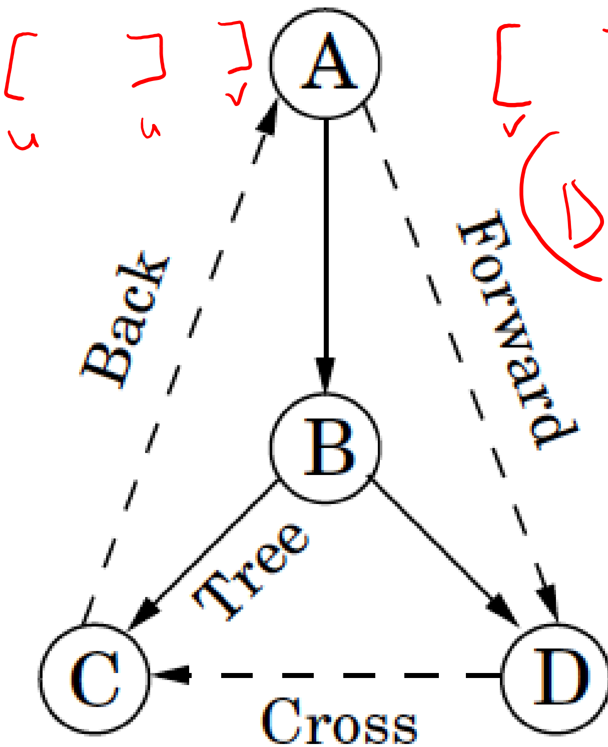
B is a **parent** of C

A is an **ancestor** of D

tree

forward

DFS tree



D → C

cross

A B C C D D B A

DFS on Directed Graphs

- How do we know whether u is a parent/child/ancestor/descendant of v?
- Use pre/post times!

In-Class Exercise

What types of edges are these?

pre/post *ordering for* (u, v)

$\underset{u}{[}\quad \underset{v}{[}\quad \underset{v}{]}\quad \underset{u}{]}$ – tree/forward

$\underset{v}{[}\quad \underset{u}{[}\quad \underset{u}{]}\quad \underset{v}{]}$ – back

$\underset{v}{[}\quad \underset{v}{]}\quad \underset{u}{[}\quad \underset{u}{]}$ – cross

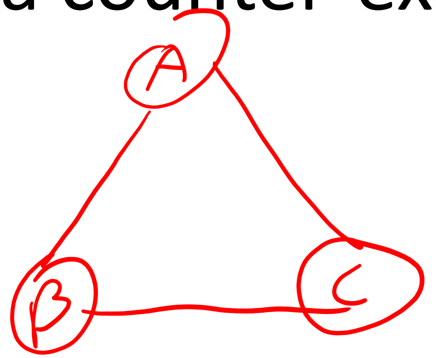
$\underset{u}{[}\quad \underset{u}{]}\quad \underset{v}{[}\quad \underset{v}{]}$ – can't happen

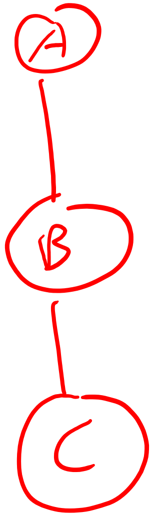
Finding Paths in Graphs

- Remember, a path between two nodes is a sequence of ~~of~~ adjacent edges connecting those nodes
- A shortest path is the path between two nodes with the fewest edges (unweighted graph)

DFS and Paths

- DFS finds paths between nodes… but does it always find shortest paths?
- Prove or give a counter-example





Breadth-first search (BFS)

N = # nodes
M = # edges

procedure bfs(G, s)
Input: Graph $G = (V, E)$, directed or undirected; vertex $s \in V$
Output: For all vertices u reachable from s, dist(u) is set to the distance from s to u.

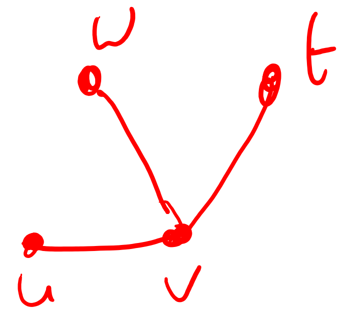
\# edges in s.p.

```
for all u ∈ V:            O(N)
    dist(u) = ∞
pred[s] = ∅
dist(s) = 0               O(1)
Q = [s] (queue containing just s)      O(1)
while Q is not empty:     O(N)
    u = eject(Q)          O(N)
    for all edges (u,v) ∈ E:           O(M)
        if dist(v) = ∞:   O(M)
            inject(Q, v)  O(N)
            dist(v) = dist(u) + 1
            pred(v) = u   O(N)
```

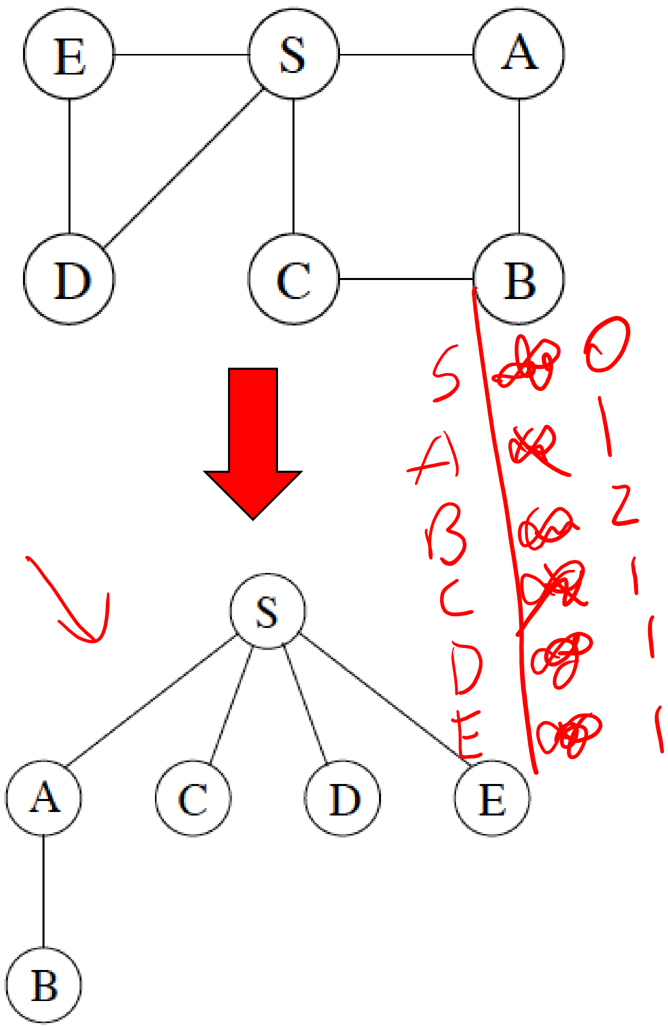
Put all neighbors on queue

$O(M+N)$

Distance to neighbor = 1 + distance to current node

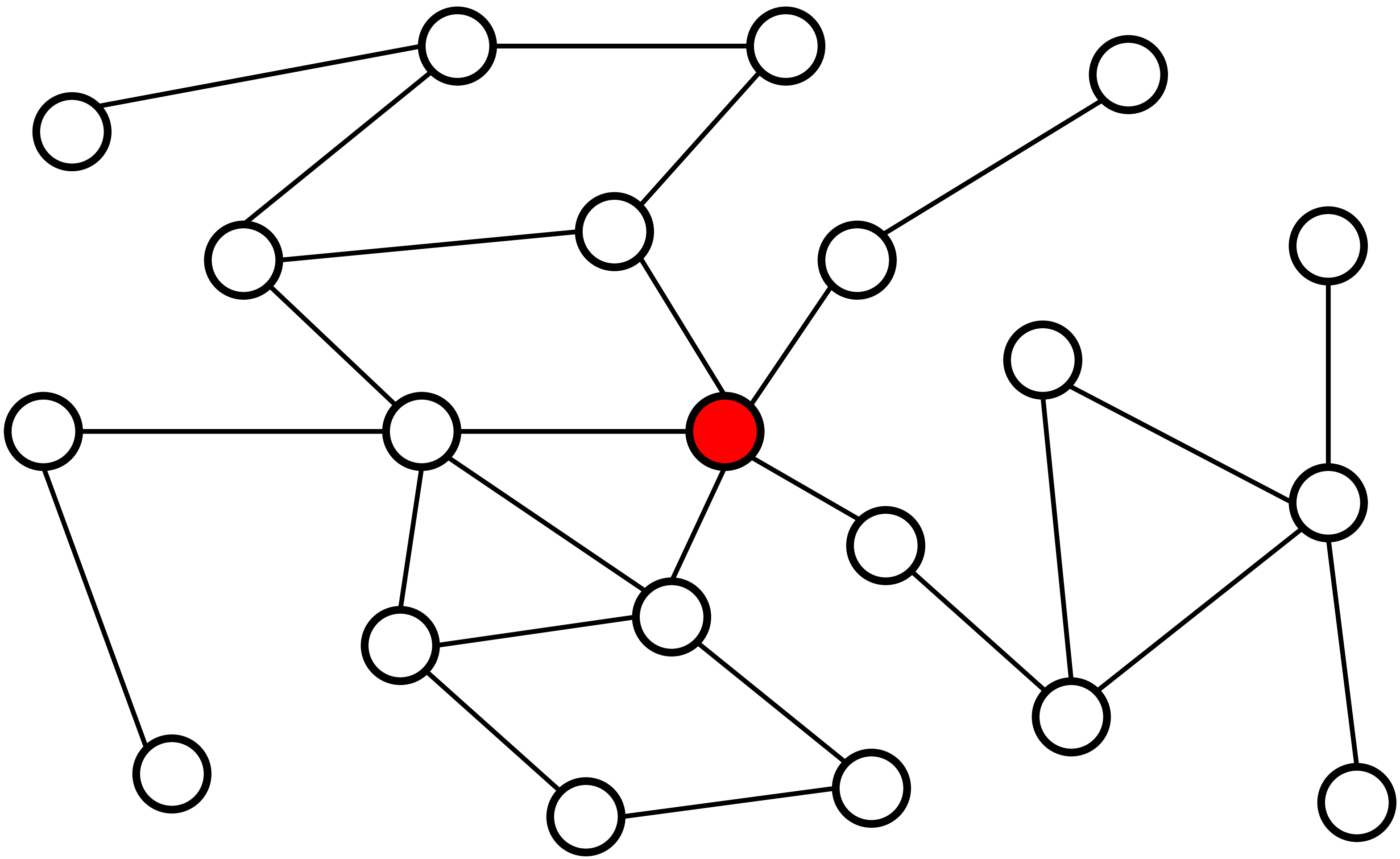


Breadth-first search (BFS)

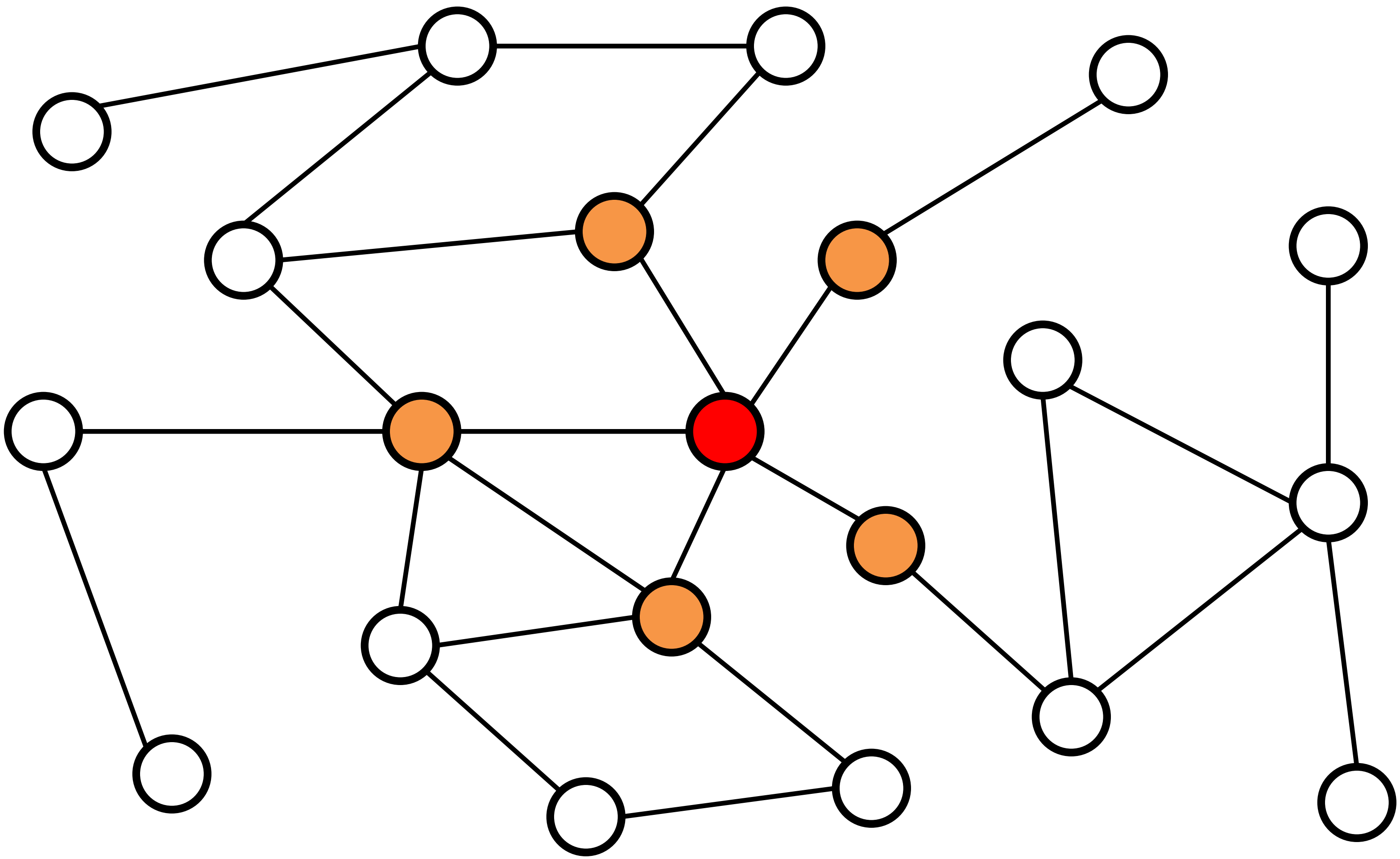


Order of visitation	Queue contents after processing node
	$[S]$
S	$[A\ C\ D\ E]$
A	$[C\ D\ E\ B]$
C	$[D\ E\ B]$
D	$[E\ B]$
E	$[B]$
B	$[\,]$

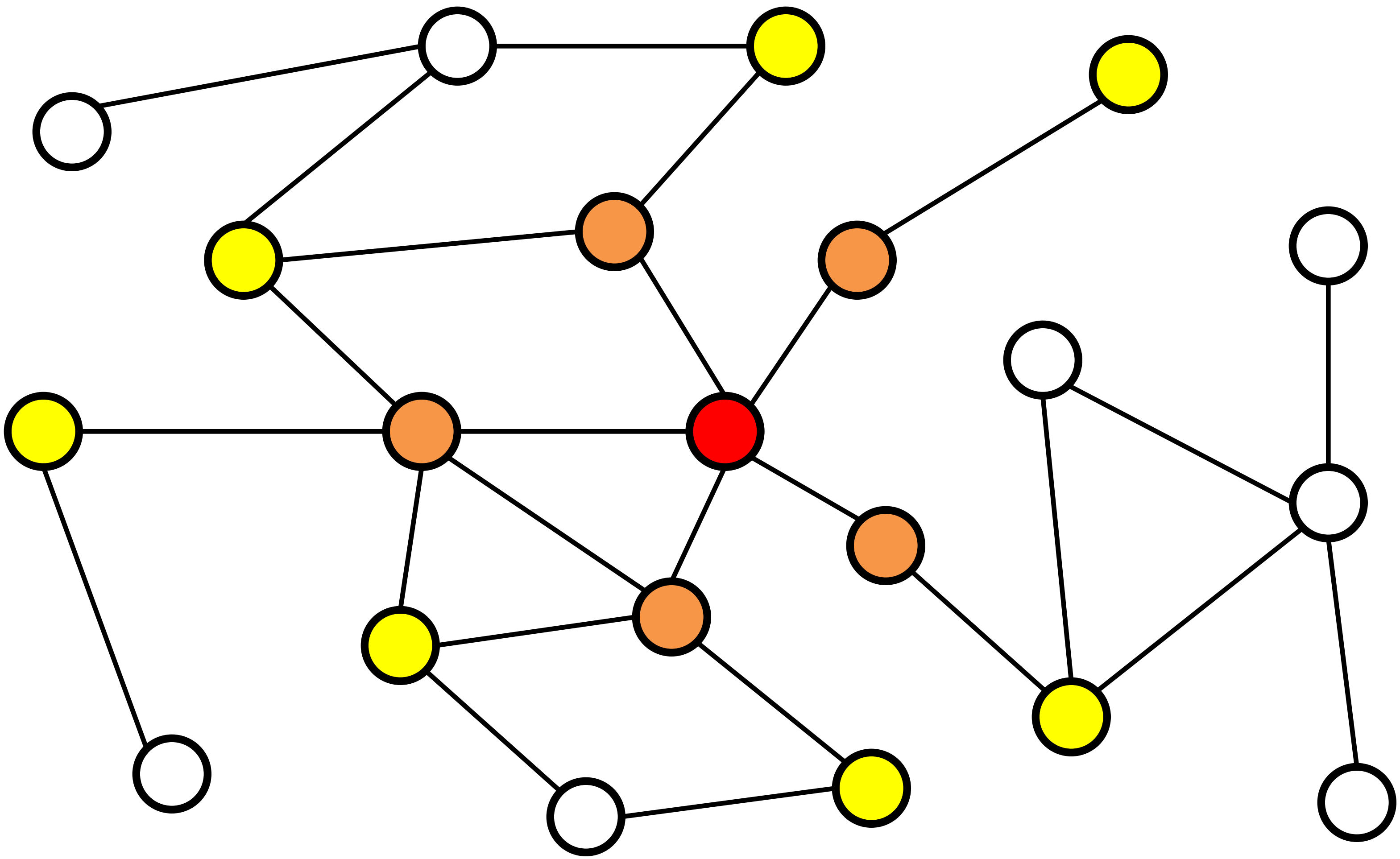
BFS



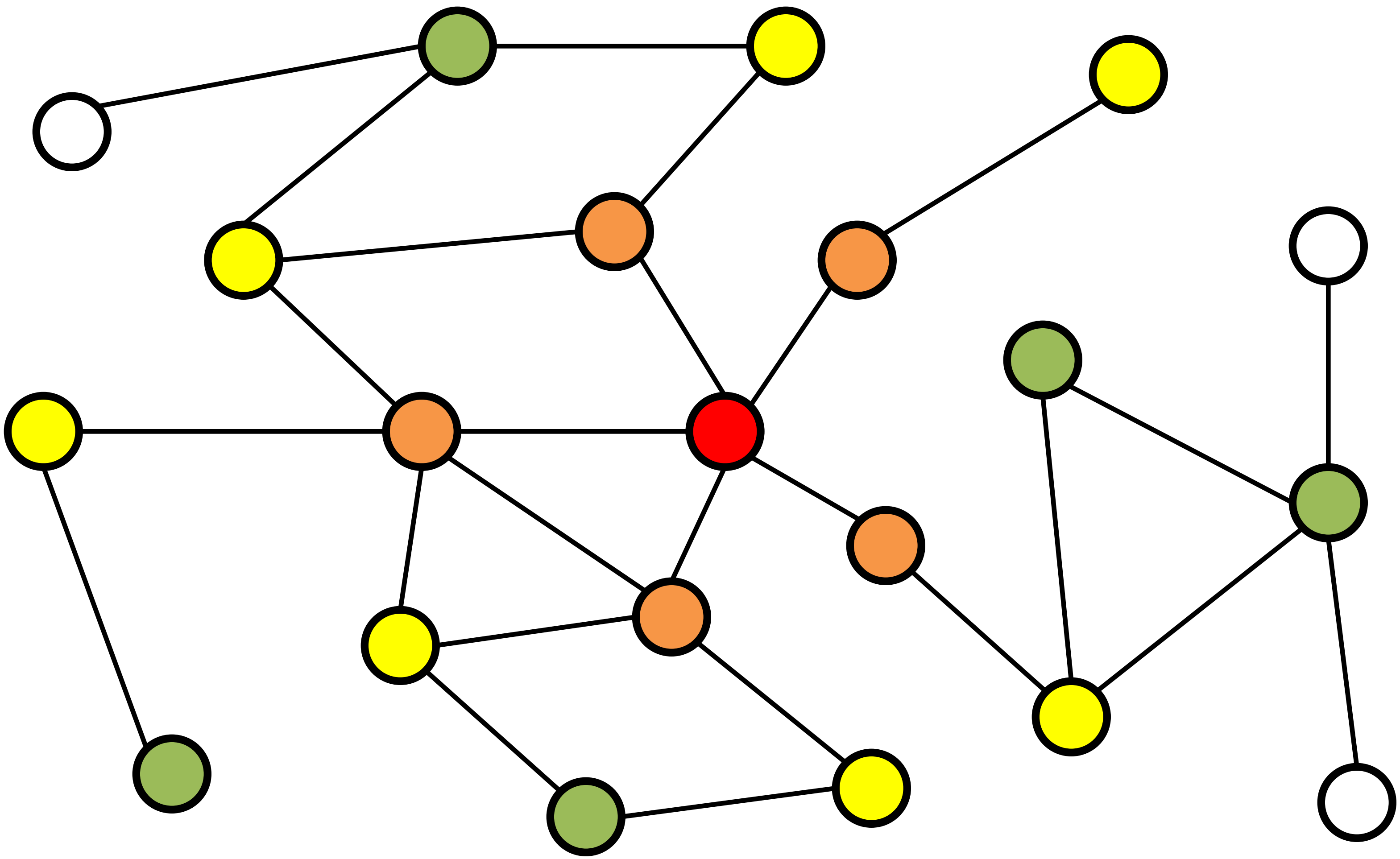
BFS



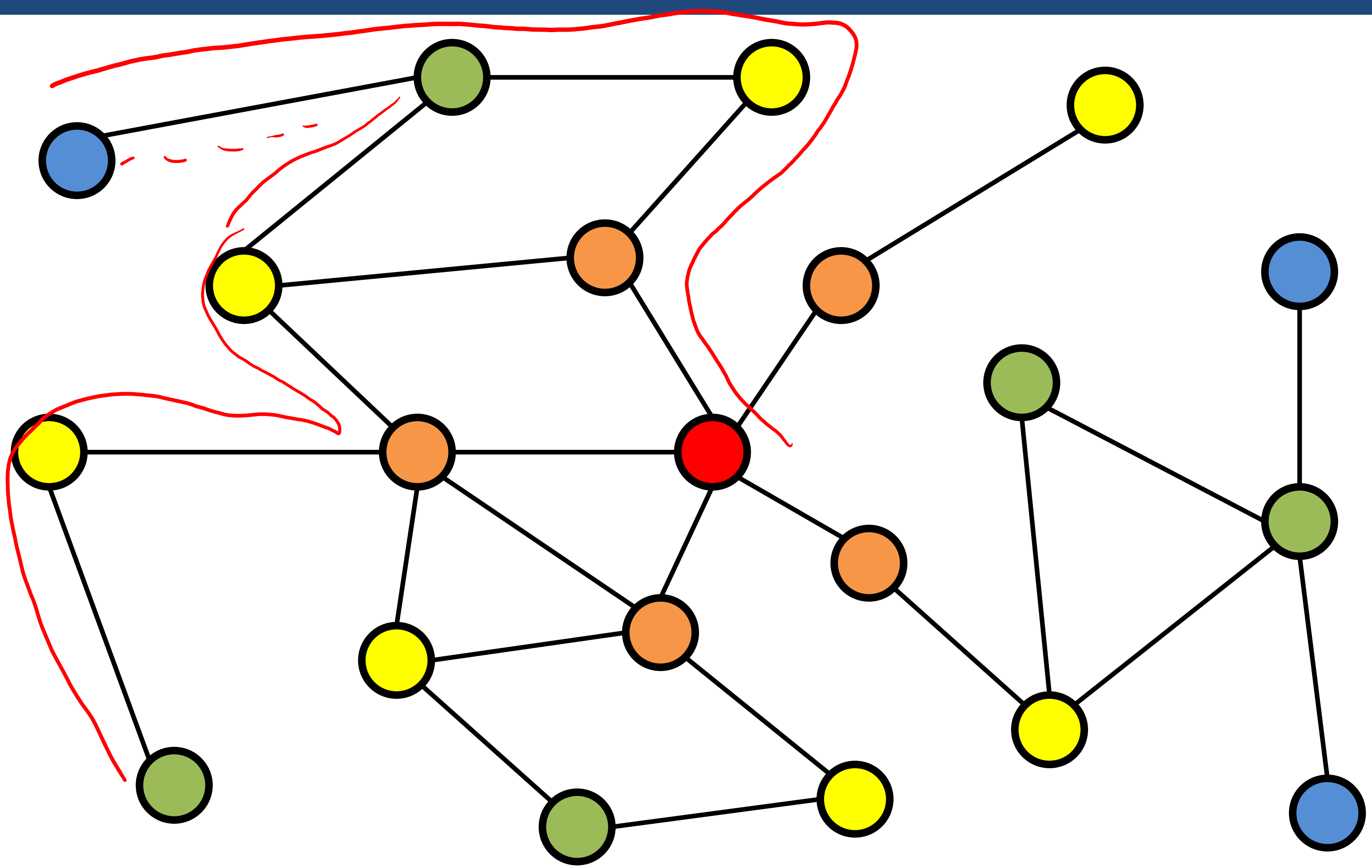
BFS



BFS



BFS



In-Class Exercise

- Does the ~~BFS tree~~ dist always give you the shortest path from the starting node to every other node? Prove or give a counter-example.

Let ρ_u denote the actual s.p. length from $s \to u$. Induction on ρ_u.

Base case: $\rho_u = 0$. This occurs when $u = s$. BFS sets $dist(s)$ before loop. ✓

I.H.: Assume that BFS is correct for all nodes u with $\rho_u \leq k$.

In-Class Exercise

I.S.: Show that if $\rho_u = k+1$, BFS sets $dist(u) = k+1$.

Suppose there is a path of length $k+1$ $s \to u$.

~~The s~~ Let w denote the node right before u on the s.p. This means that the s.p. $s \to w$ has length k. By I.H. $dist(w) = k$ correctly. When we set $dist(w)$, we look at all its neighbors. u is one of them, so $dist(u) = dist(w) + 1 = k+1$. $\square$

Running Time of BFS

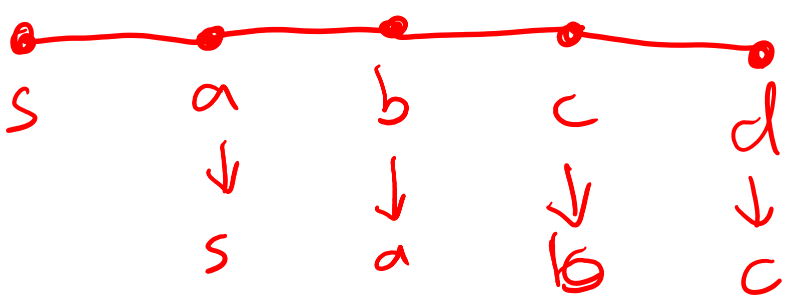
- What is the running time of BFS?

$O(\#\text{nodes} + \#\text{edges})$

Shortest Paths via BFS

- In an unweighted graph, the visit time of a BFS search gives the length of the shortest path from the source! store predecessor
- How can we modify the BFS code to keep track of what the edges in the shortest path are?

linked list ~~for each~~ node?



a: s → a
b: s → a → b
c: s → a → b → c
d: s → a → b → c → d

In-Class Exercise

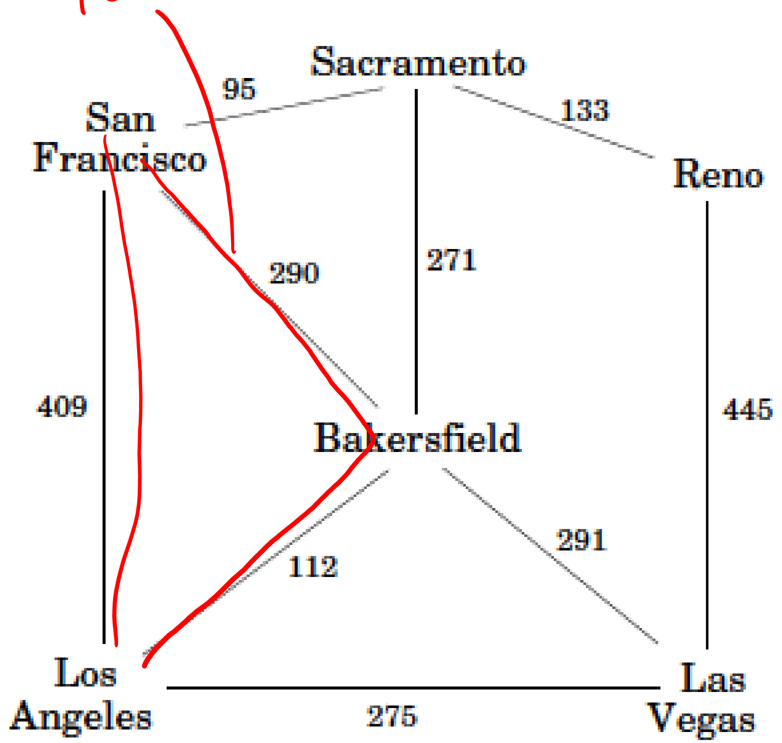
- How can we modify the code to keep track of what the edges on the shortest paths are?

```
for all u ∈ V:
    dist(u) = ∞
pred(s) = ∅
dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
    u = eject(Q)
    for all edges (u, v) ∈ E:
        if dist(v) = ∞:
            inject(Q, v)
            dist(v) = dist(u) + 1
            pred(v) = u
```

Shortest Paths via BFS

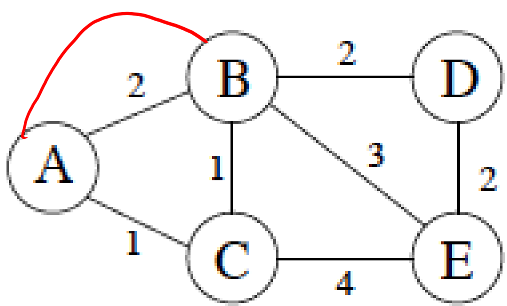
- What about in weighted graphs?

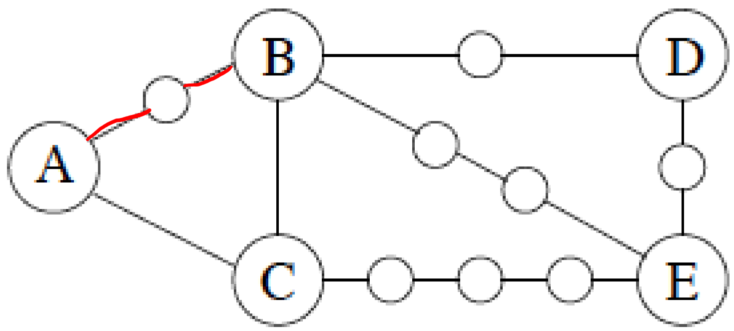
402



Dijkstra's Algorithm

- One way to deal with edges with positive, integer weights is to use dummy nodes

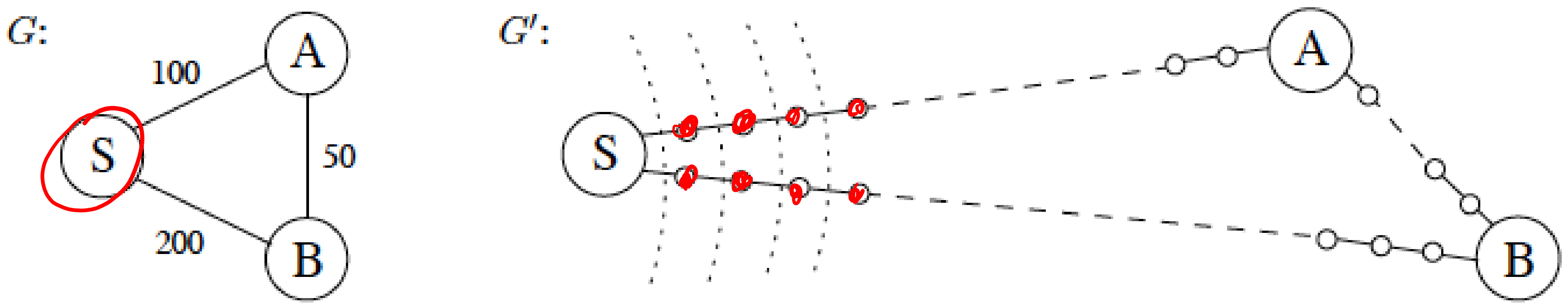




- What does this do to the running time? vastly increases!

Dijkstra's Algorithm

- With dummy nodes, most of the search process is uninteresting



- Idea: set alarm clocks to go off when we expect something interesting to happen

Dijkstra's Algorithm

- To implement this, we're going to use a priority queue

from G

- Each element (node) has a key value
- Added operations:

with value

  - Insert (add new element to set)
  - Decrease-key (reduce the value of the key)
  - Delete-min (find the element with the smallest key value, return it and delete from priority queue)
  - Make queue (build queue)

Dijkstra's Algorithm

assumes that all weights ≥ 0

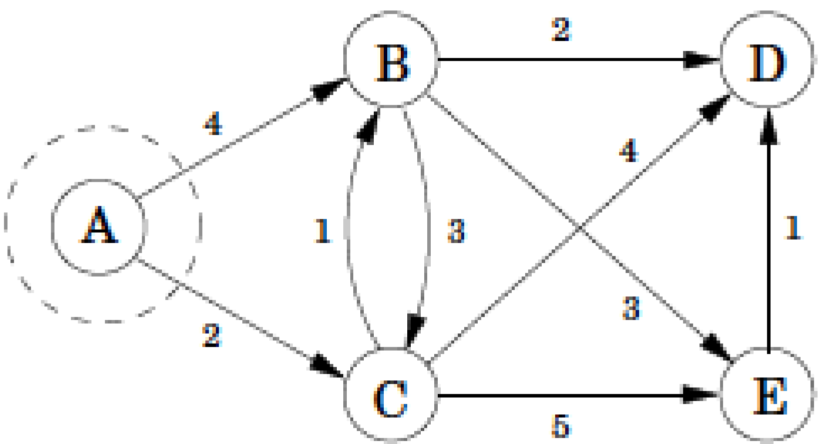
weight 0 = no edge

lower is better

```
for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u,v) ∈ E:
      if dist(v) > dist(u) + l(u,v):
         dist(v) = dist(u) + l(u,v)
         prev(v) = u
         decreasekey(H,v)
```

Dijkstra's Algorithm

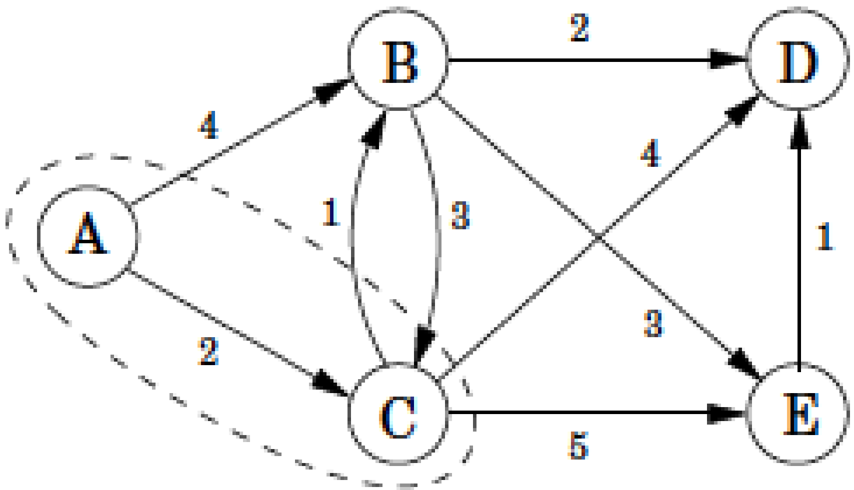


A: 0	D: ∞
B: ∞	E: ∞
C: ∞	

A: 0	D: ∞
B: 4	E: ∞
C: 2	

~~A~~	B	C	D	E
~~0~~	~~∞~~	~~∞~~	∞	∞
	4	2		

Dijkstra's Algorithm



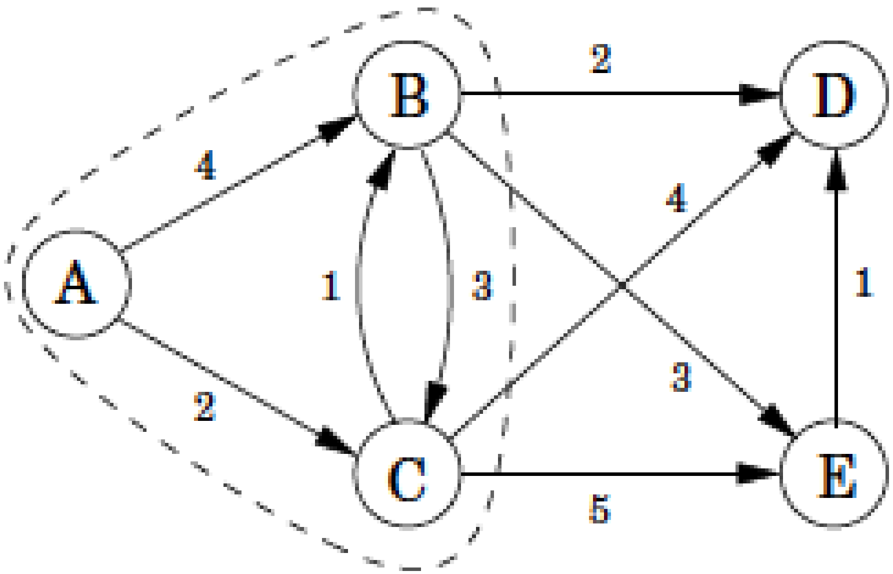
A: 0	D: 6
B: 3	E: 7
C: 2	

B ~~C~~ D E

4 ~~A~~ ~~∞~~ ~~∞~~

6 7

Dijkstra’s Algorithm



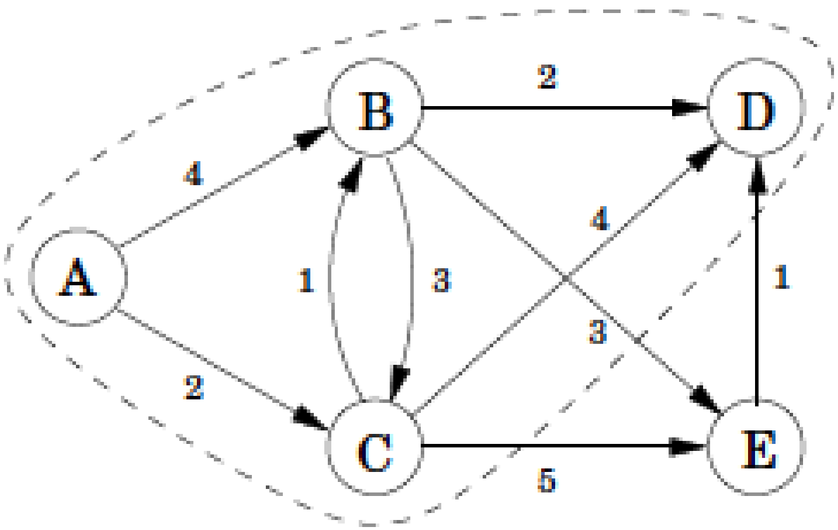
A: 0 B: 3 C: 2	D: 5 E: 6

~~B~~ D E

~~4~~ ~~6~~ ~~7~~

5 6

Dijkstra’s Algorithm



A: 0	D: 5
B: 3	E: 6
C: 2	

~~D 5~~ ~~E 6~~