Data Structures

CSCI 2270-202: REC 10

Sanskar Katiyar

Logistics

Office Hours (Zoom ID on Course Calendar)

Wednesday: 3 pm – 5 pm

Thursday: 5 pm – 6 pm

Friday: 3 pm – 5 pm

Recitation Materials *(Notes, Slides, Code, etc.)*

sanskarkatiyar.github.io/CSCI2270

Recitation Outline

1. Graph: Depth First Search
2. Graph: Breadth First Search
3. Exercise

Graph: Traversal

Previously, we discussed 3 traversals in a tree

How should one go about **traversing a graph**?

- Start at a node
- Process neighbor nodes in *some* order

Popular Graph Traversals

- Breadth First Search (BFS)
- Depth First Search (DFS)

Depth First Search (DFS)

Depth First Search (DFS)

Don’t visit any vertex more than once

Keep track, mark each visited vertex as *visited*

Depth first:

Visit currentNode’s *neighbor’s neighbor’s neighbor’s* ….

If no more unvisited vertices on this path, backtrack on the path until you find an unvisited neighbor, and start DFS there

Depth First Search (DFS)

Need a source vertex to start DFS at

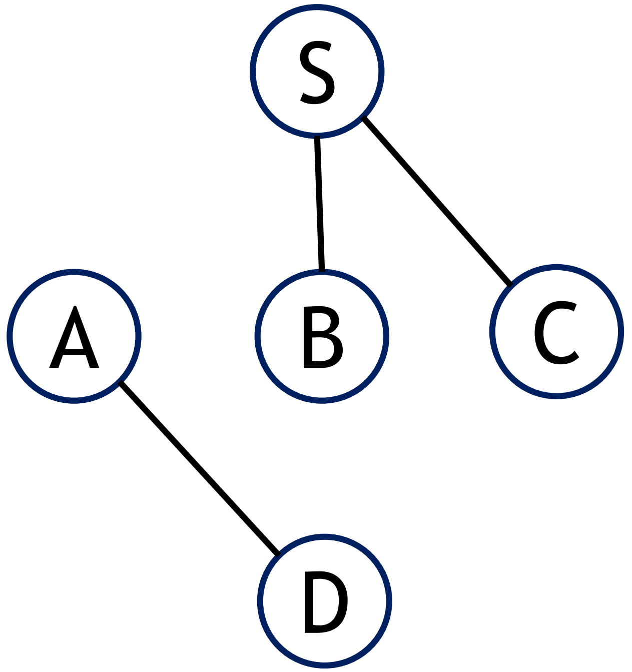
Approaches

- Recursive: recall tree traversals
- Iterative: utilize a Stack

Applications

- Connected components in an undirected Graph, Web Crawlers*
- Path planning, Maze solving

DFS: Pseudocode (Recursive)

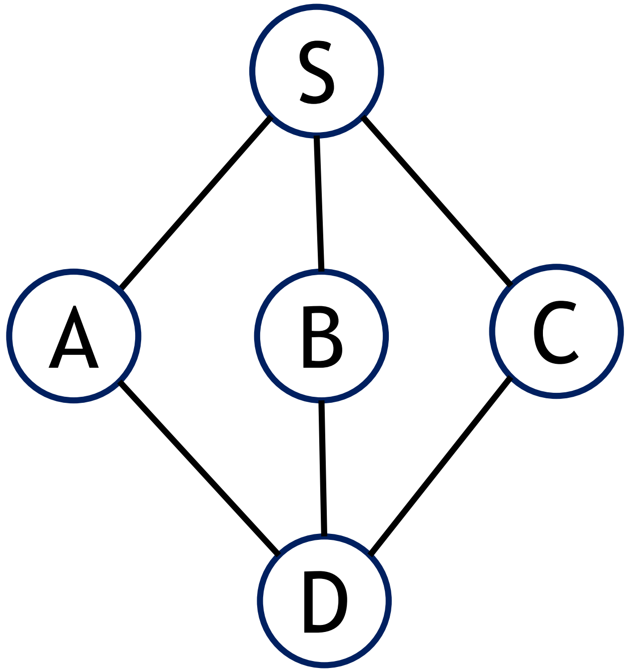


```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}

for each u ∈ G
    u.visited = false
for each u ∈ G
    if u.visited == false
        DFS(G, u)
```

Initialize all nodes as unvisited

Loop: If there is more than one component

DFS: Example



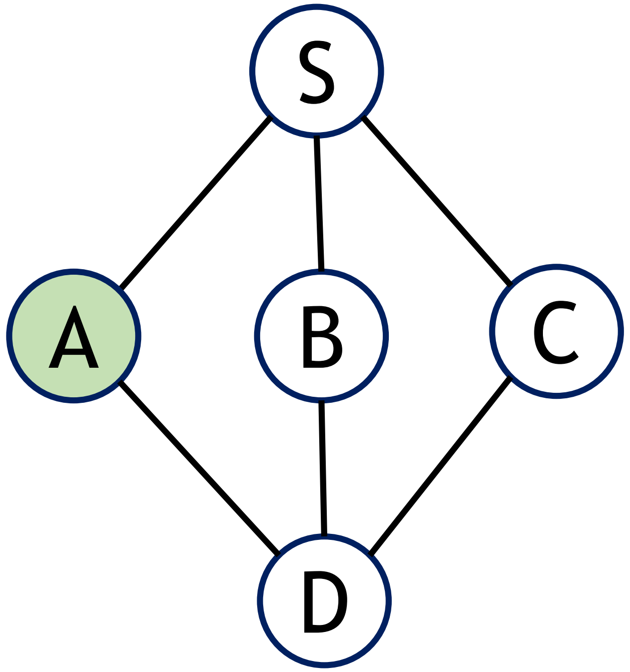
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

Function Call Stack

>

DFS: Example



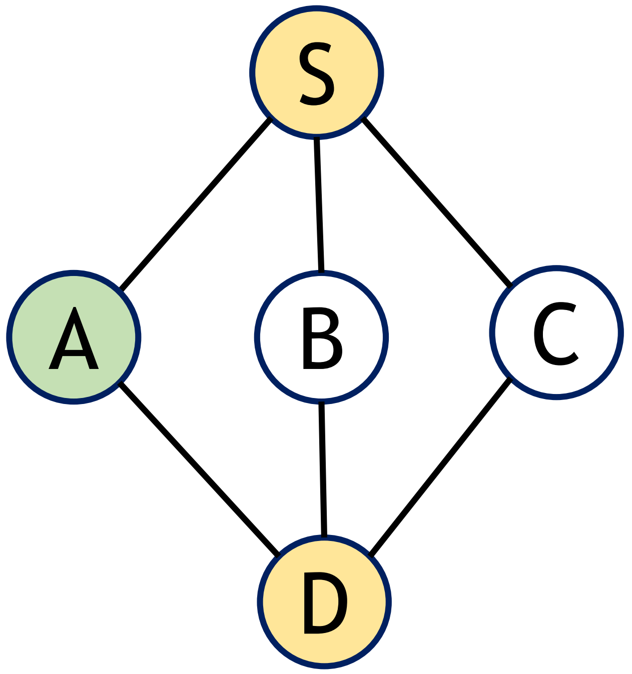
> A

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

A					

DFS: Example



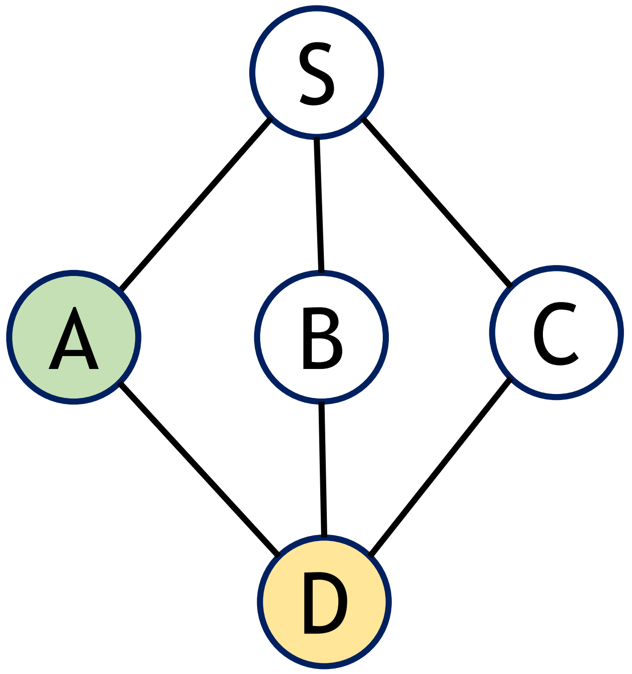
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

A				

> A

DFS: Example



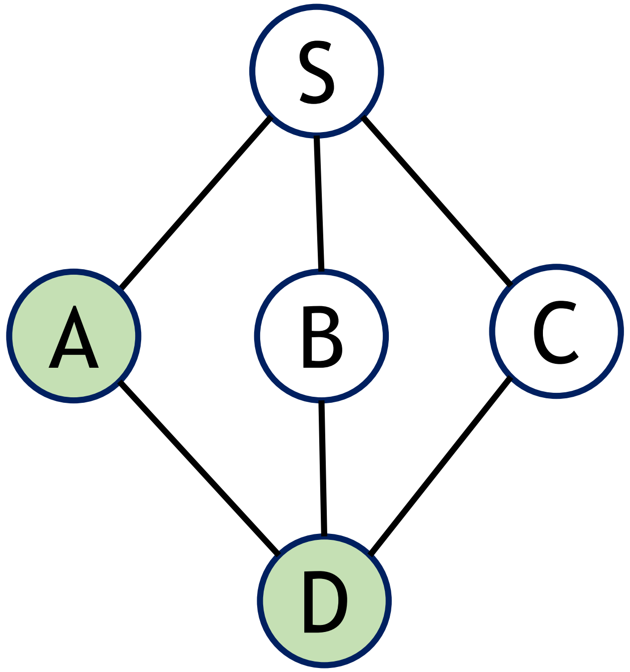
> A

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

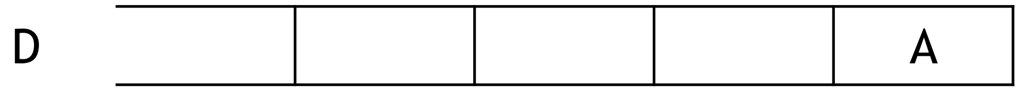
A					

DFS: Example



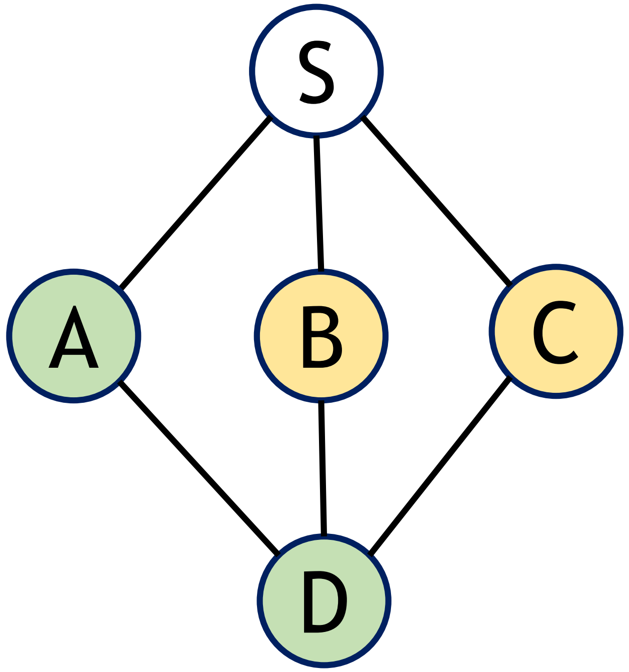
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)



> A, D

DFS: Example



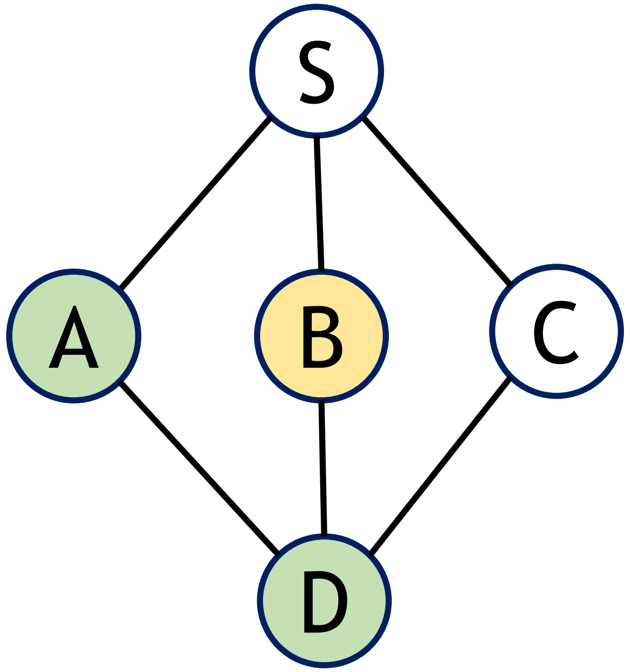
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

D					A

> A, D

DFS: Example



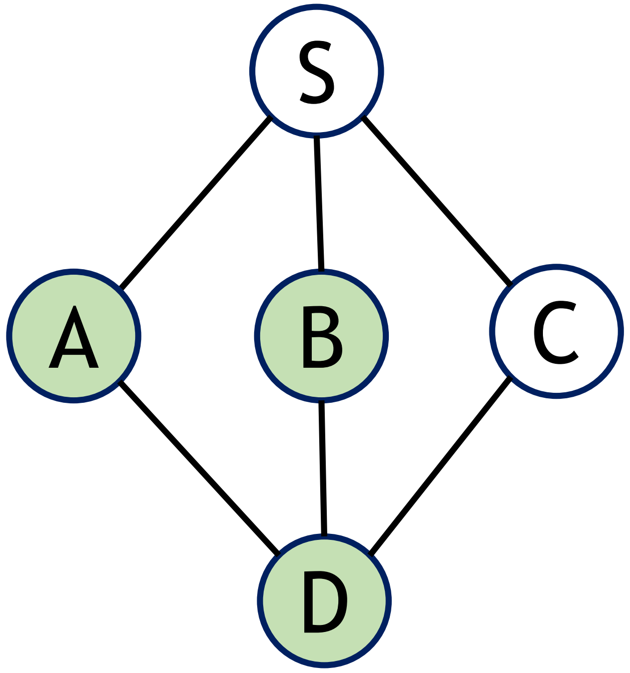
> A, D

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

D					A

DFS: Example



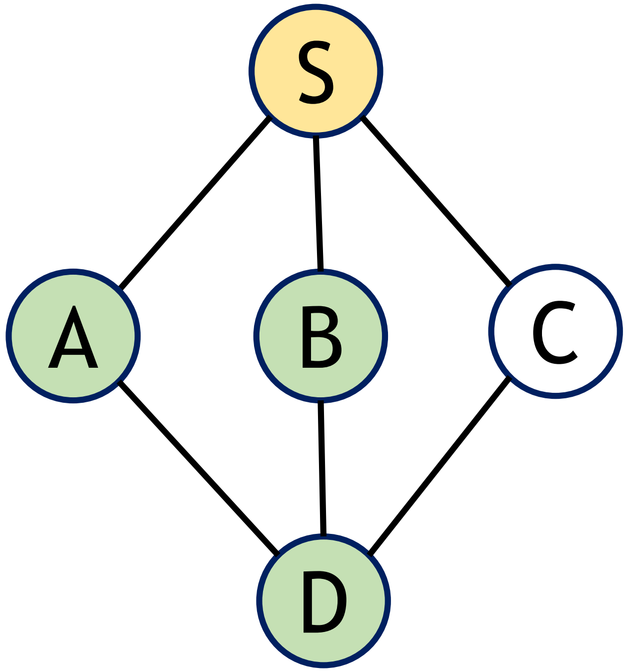
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

B				D	A

> A, D, B

DFS: Example



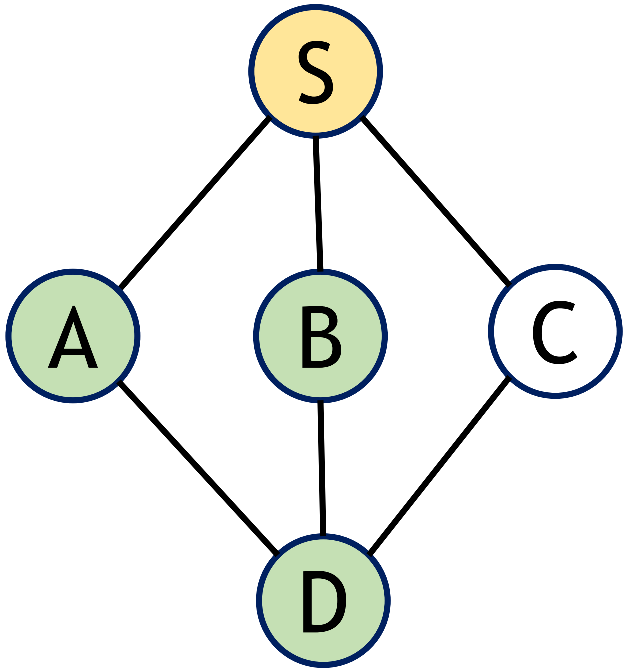
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

B				D	A

> A, D, B

DFS: Example



```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

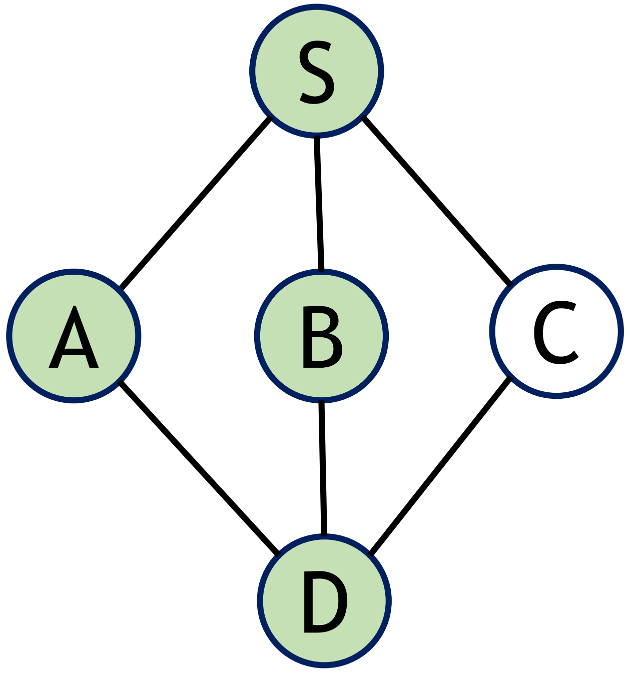
DFS(G, A)

B

			D	A

> A, D, B

DFS: Example



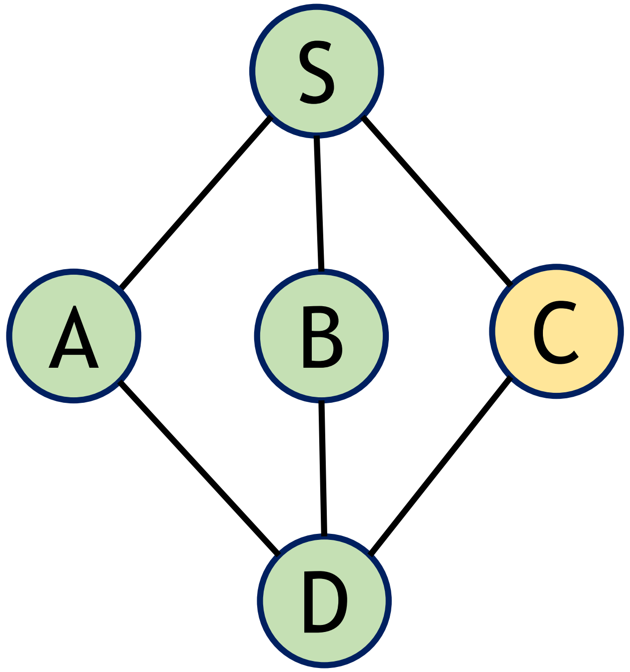
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

S			B	D	A

> A, D, B, S

DFS: Example



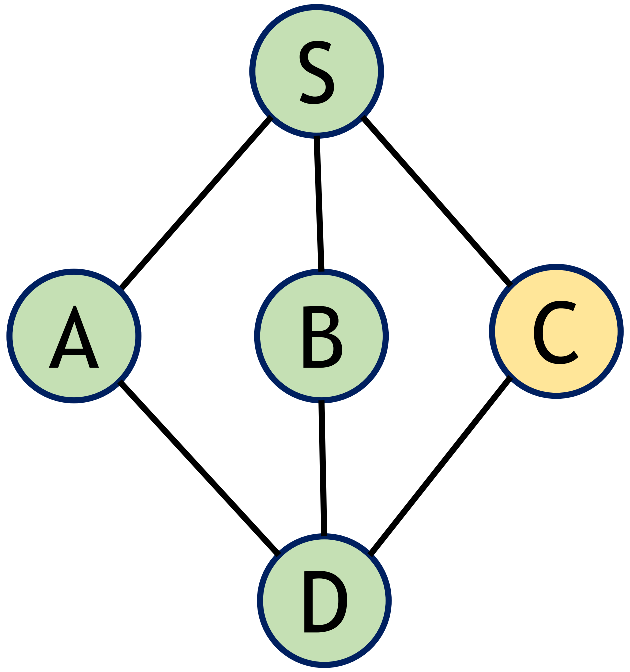
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

S			B	D	A

> A, D, B, S

DFS: Example



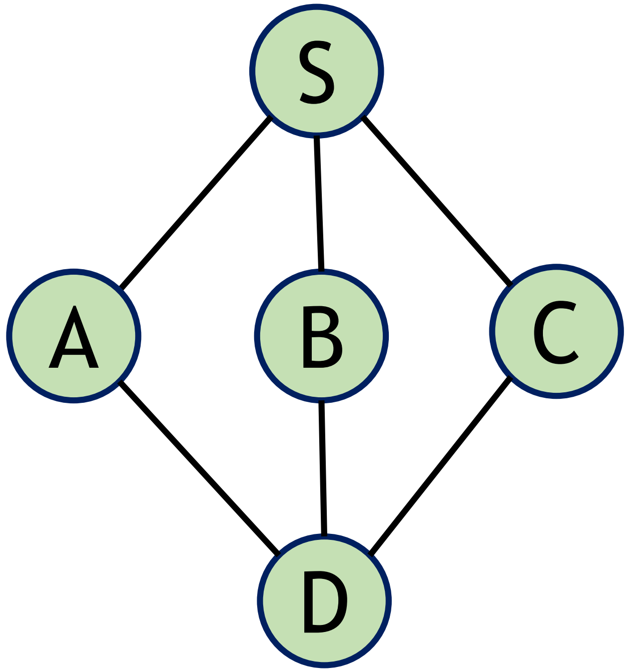
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

S			B	D	A

> A, D, B, S

DFS: Example



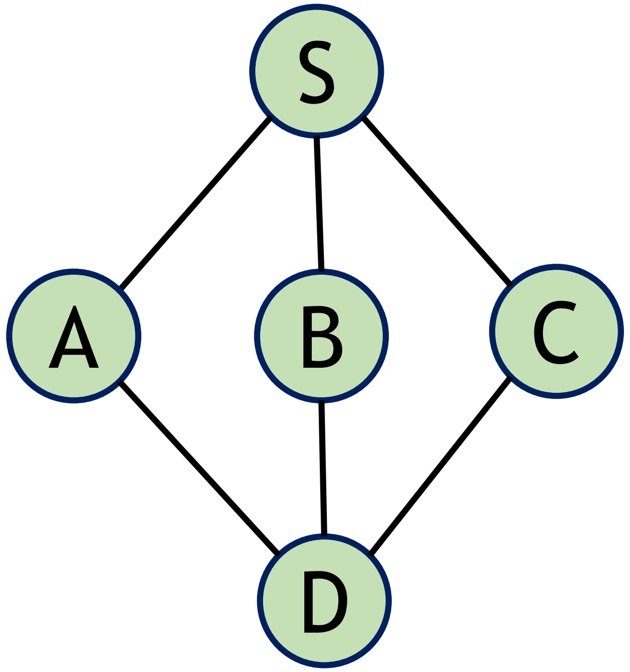
> A, D, B, S, C

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

C		S	B	D	A

DFS: Example



```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

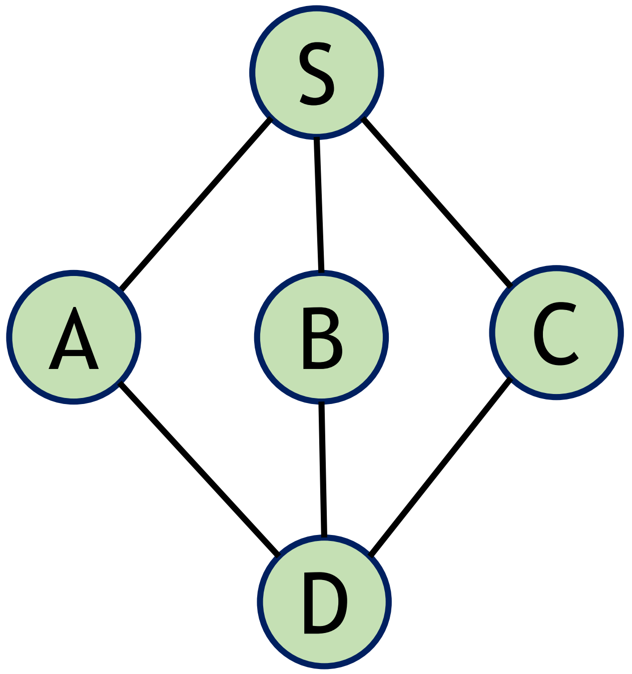
> A, D, B, S, C

C

	S	B	D	A

No unvisited nodes, function calls pop

DFS: Example



```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

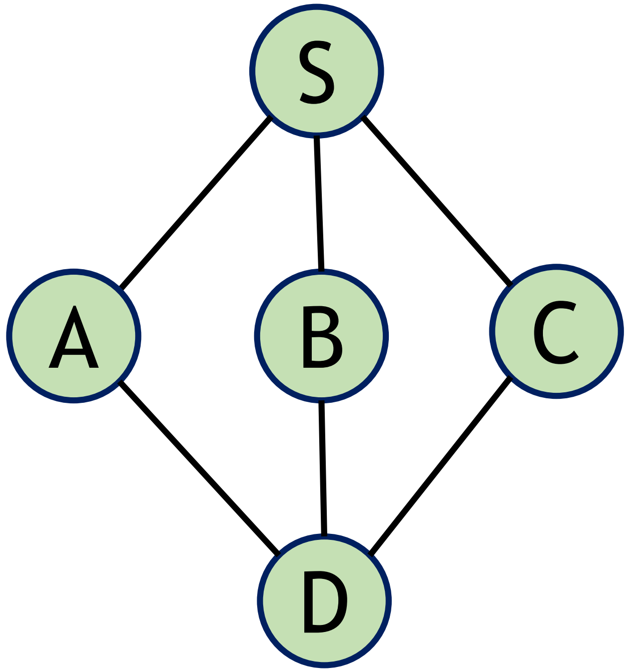
```
DFS(G, A)
```

> A, D, B, S, C

S			B	D	A

No unvisited nodes, function calls pop

DFS: Example



> A, D, B, S, C

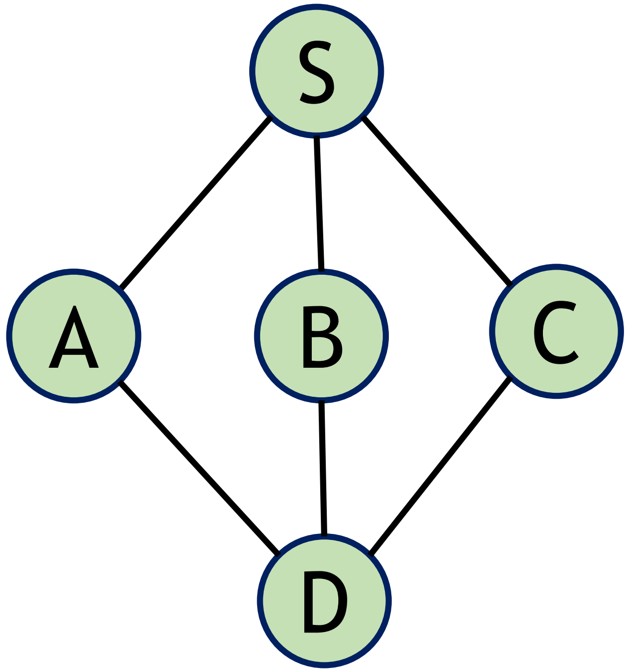
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

B				D	A

No unvisited nodes, function calls pop

DFS: Example



> A, D, B, S, C

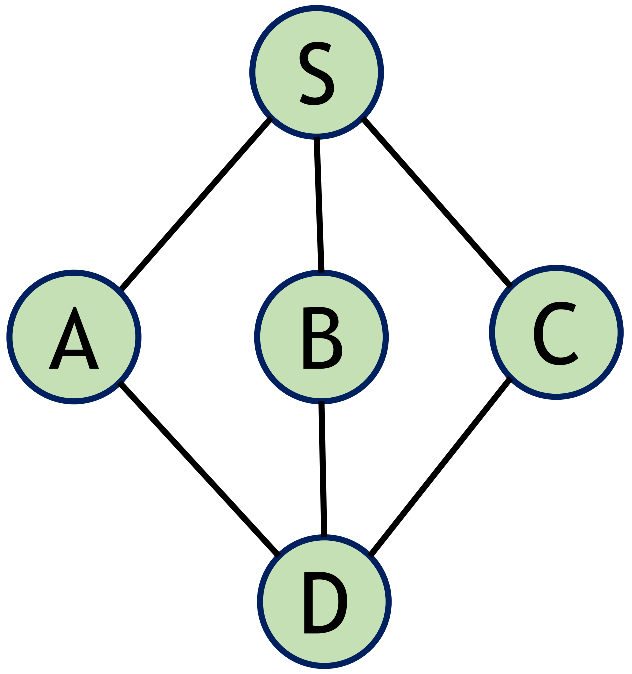
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

D					A

No unvisited nodes, function calls pop

DFS: Example



> A, D, B, S, C

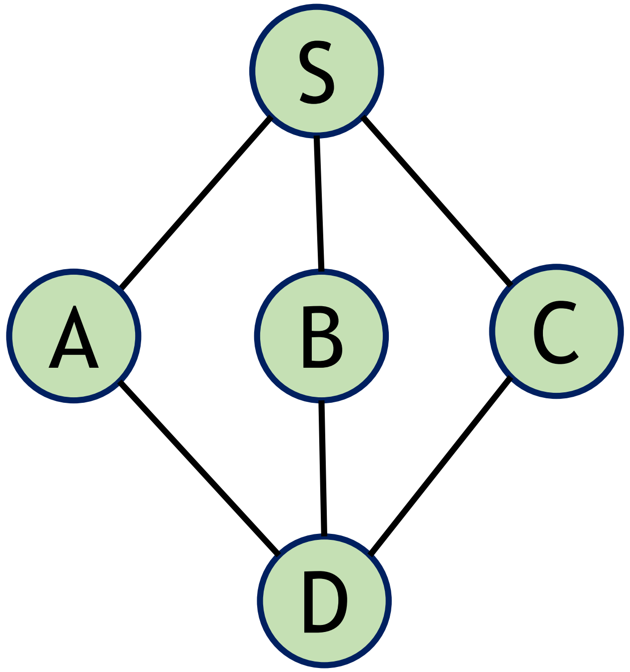
```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

DFS(G, A)

A

No unvisited nodes, function calls pop

DFS: Example



> A, D, B, S, C

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}
```

⬅

DFS(G, A)

Complete!

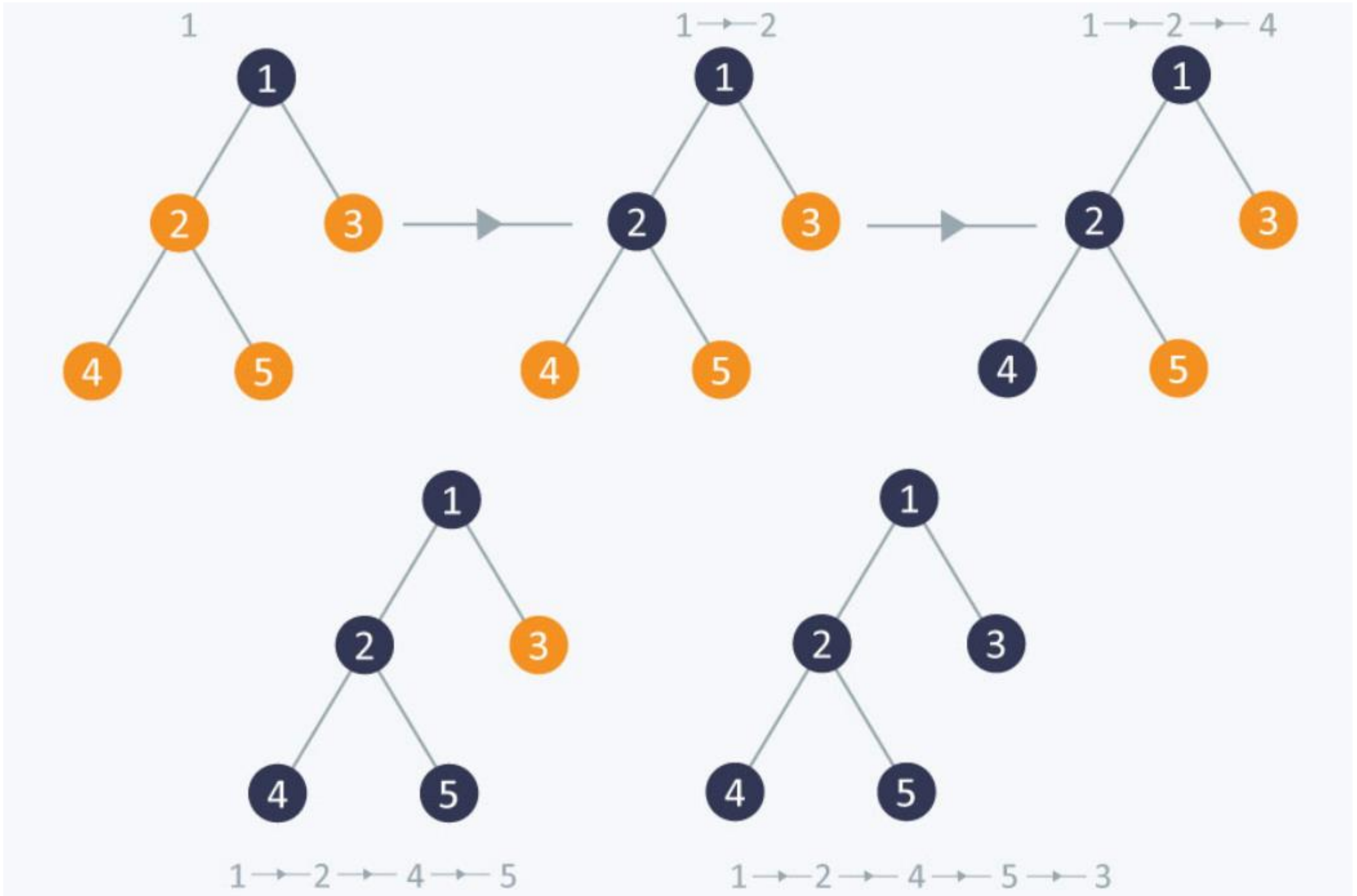
DFS: Iterative

Explicitly declare a Stack (Can use STL Stack)

Push the nodes as you visit them

Github > Recitation 10 > Code

DFS: Tree



Which tree traversal is this equivalent to?

Preorder

https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

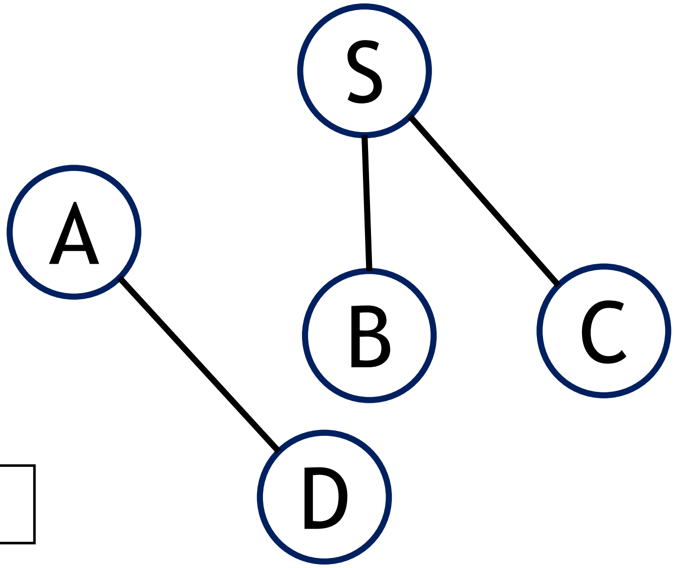
DFS: Finding Number of Components

```
DFS(G, u) {
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
}

for each u ∈ G
    u.visited = false
for each u ∈ G
    if u.visited == false
        DFS(G, u)
```

```
#components++;
```

Initialize all nodes as unvisited

Loop: If there is more than one component



Breadth First Search (BFS)

Breadth First Search (BFS)

Don’t visit any vertex more than once

Keep track, mark each visited vertex as *visited*

Breadth first:

Visit all of currentNode’s neighbors, *followed by their neighbors*

Proceed in hops (print all the items at distance *i*) `printLevelNodes?`

Use Queue to maintain the order

Breadth First Search (BFS)

Need a source vertex to start BFS at

Approaches

- Recursive: recall `printLevelNodes` from Assignment 6 (similar)
- Iterative: utilize a Queue

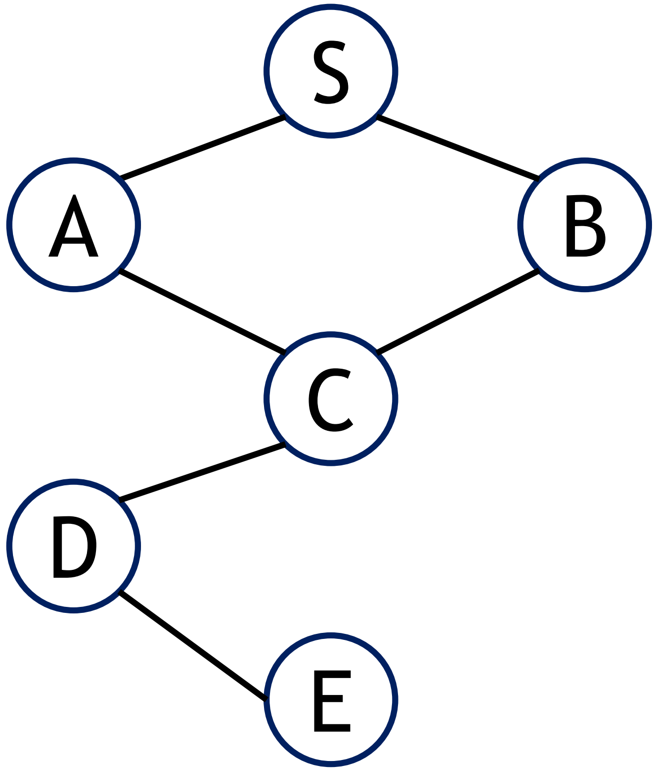
Applications

- Connected components in an undirected Graph, Web Crawlers*
- Shortest path between two nodes

BFS: Pseudocode (Iterative)

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

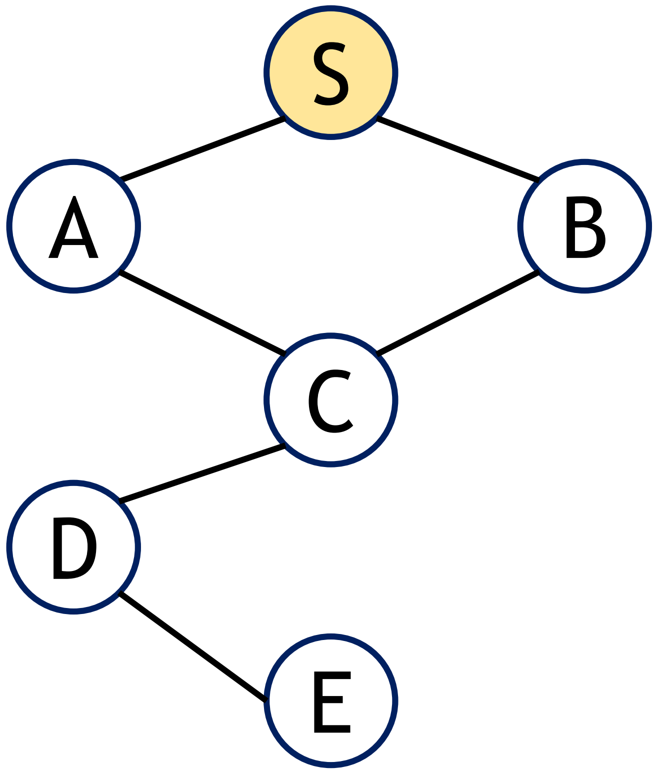
BFS: Example



>

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

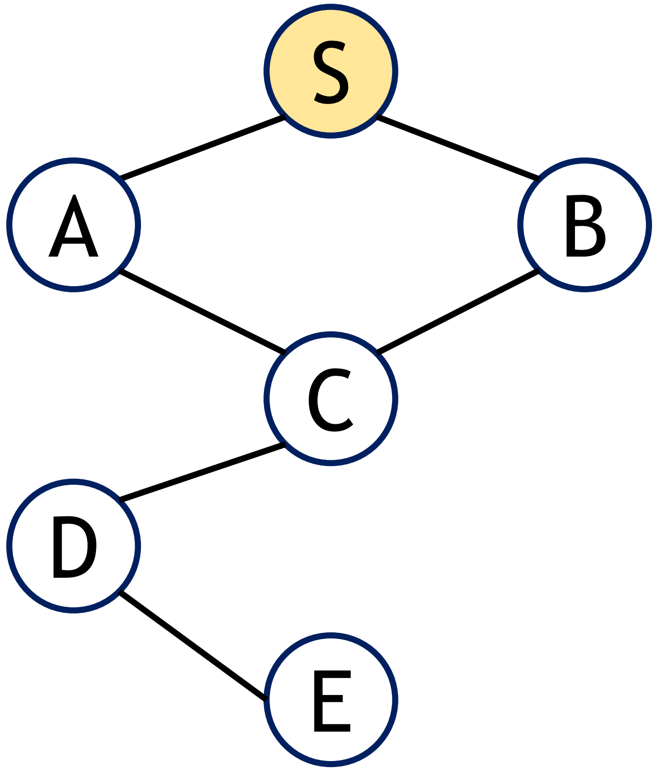
BFS: Example



>

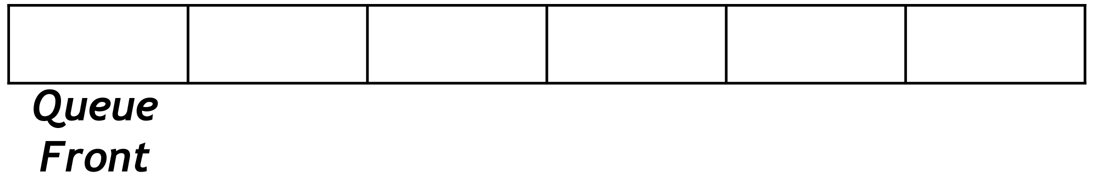
```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

BFS: Example

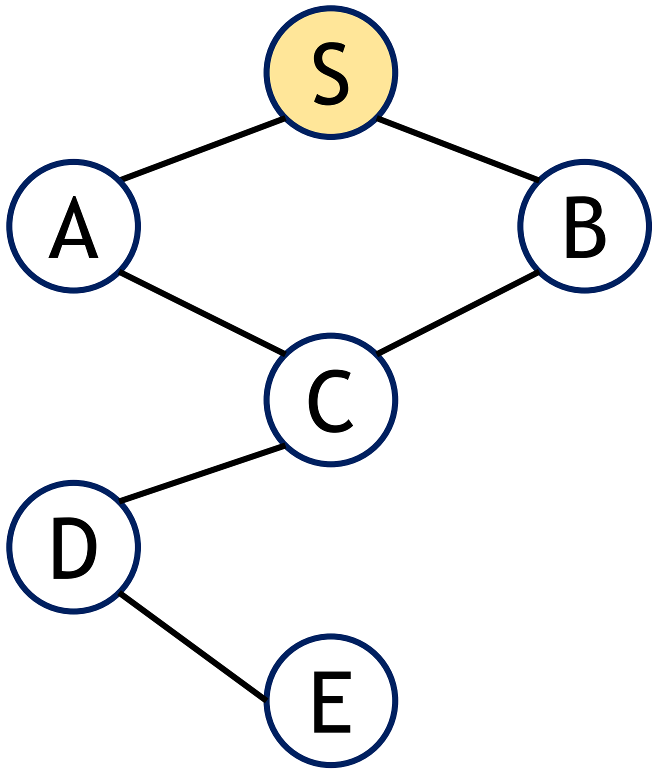


>

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



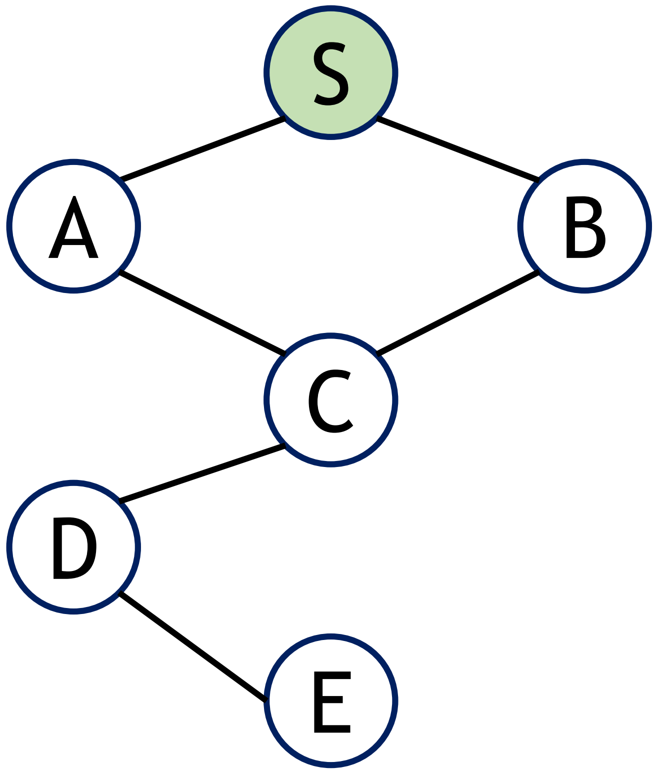
```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

S					

Queue Front

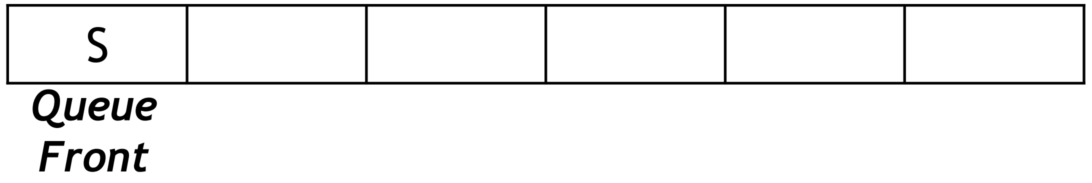
>

BFS: Example

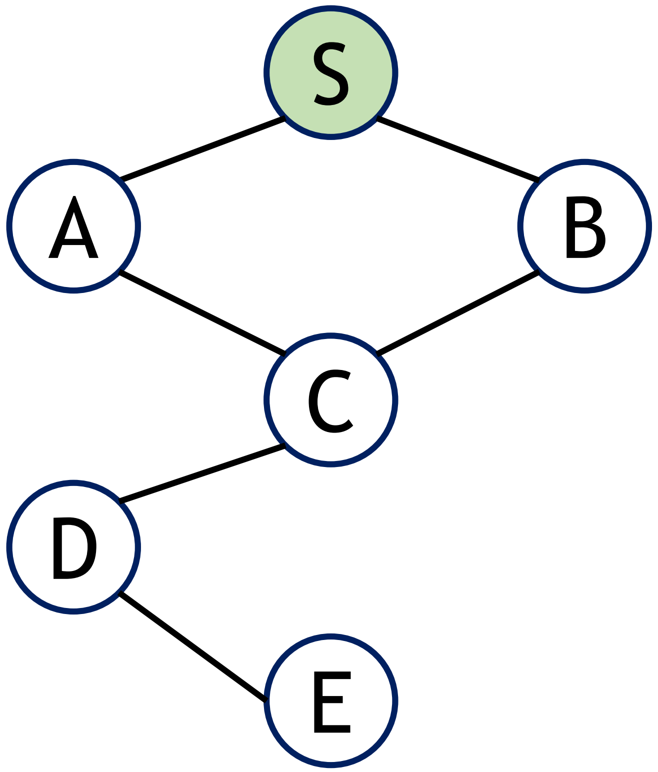


> S

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

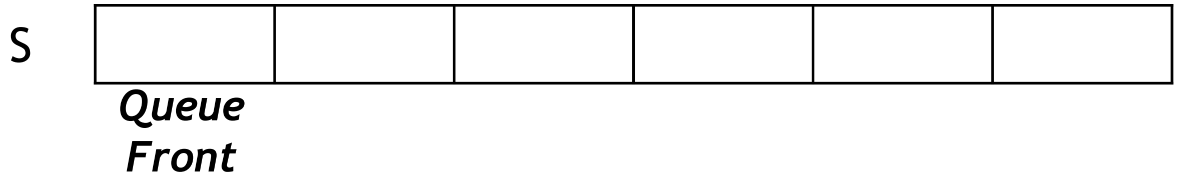


BFS: Example

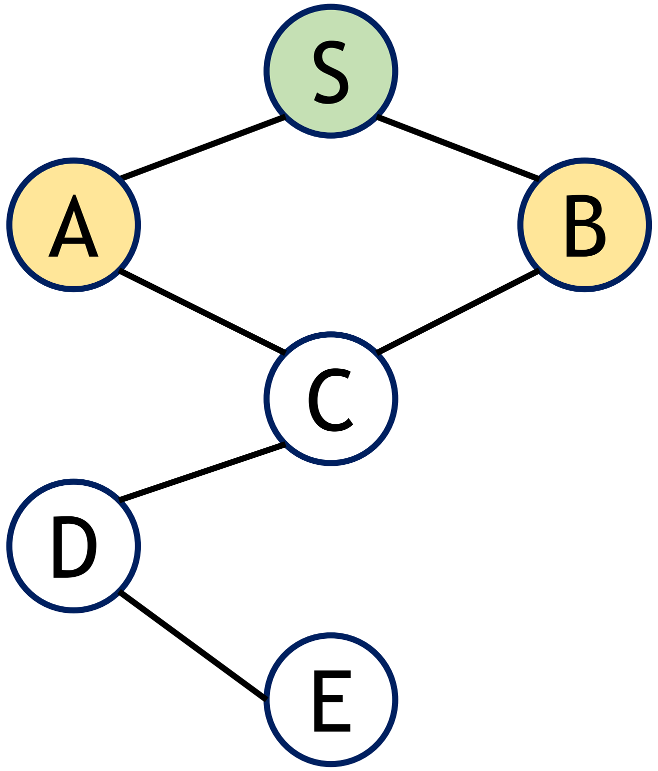


> S

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

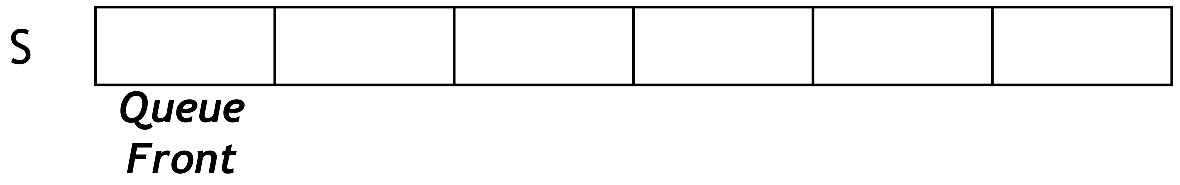


BFS: Example

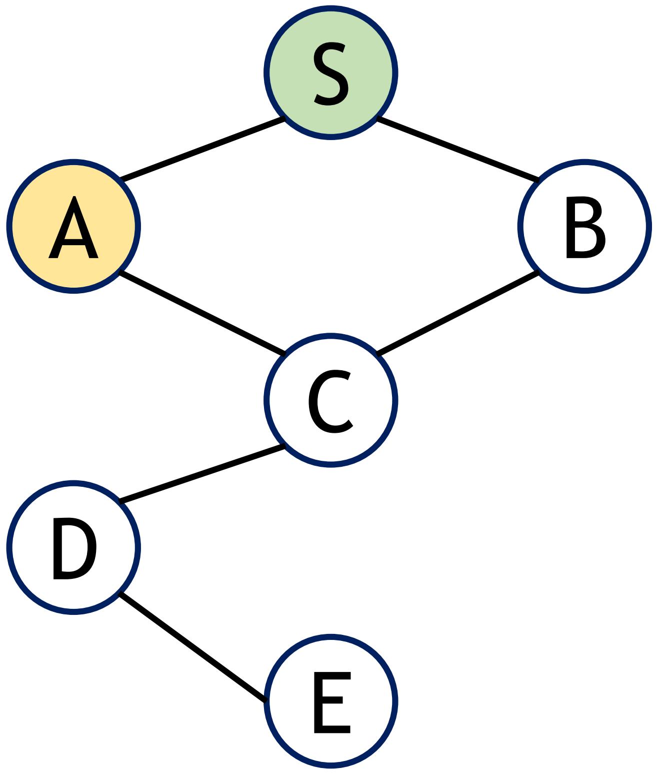


> S

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

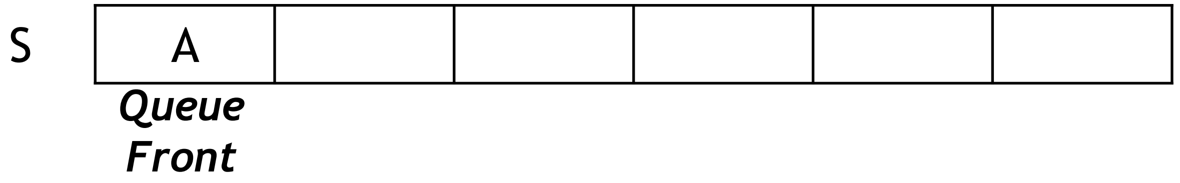


BFS: Example

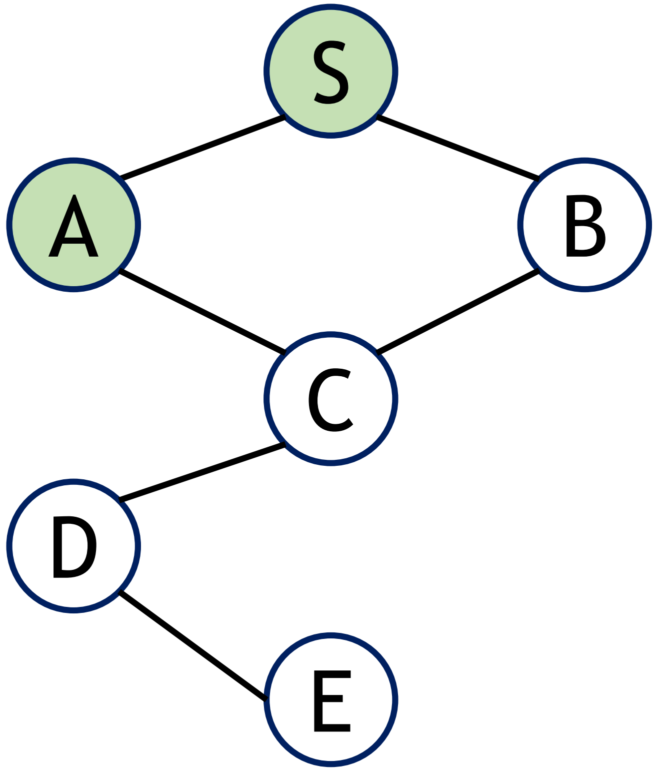


> S

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

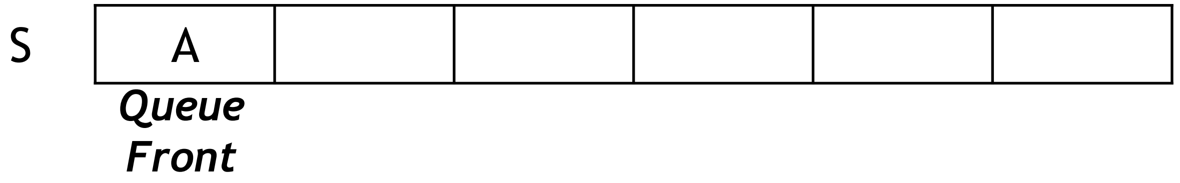


BFS: Example

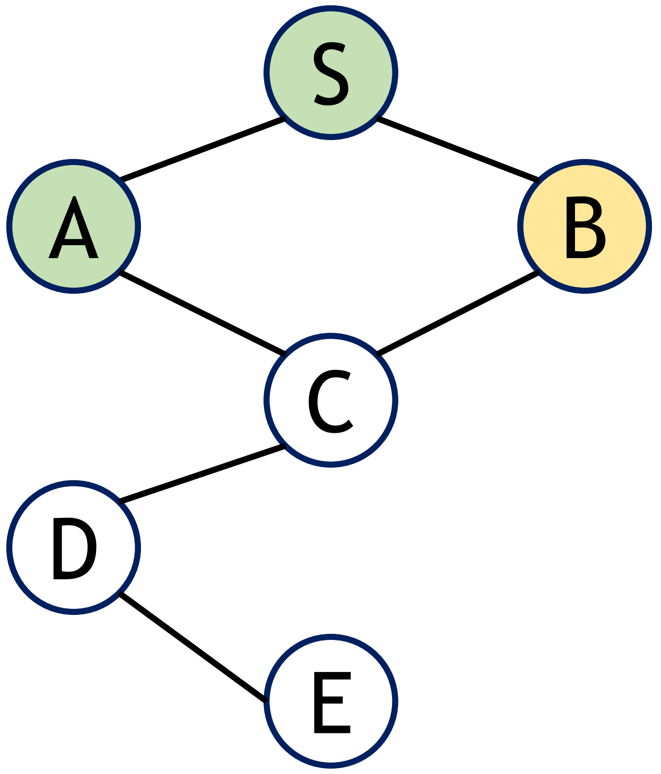


> S, A

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



> S, A

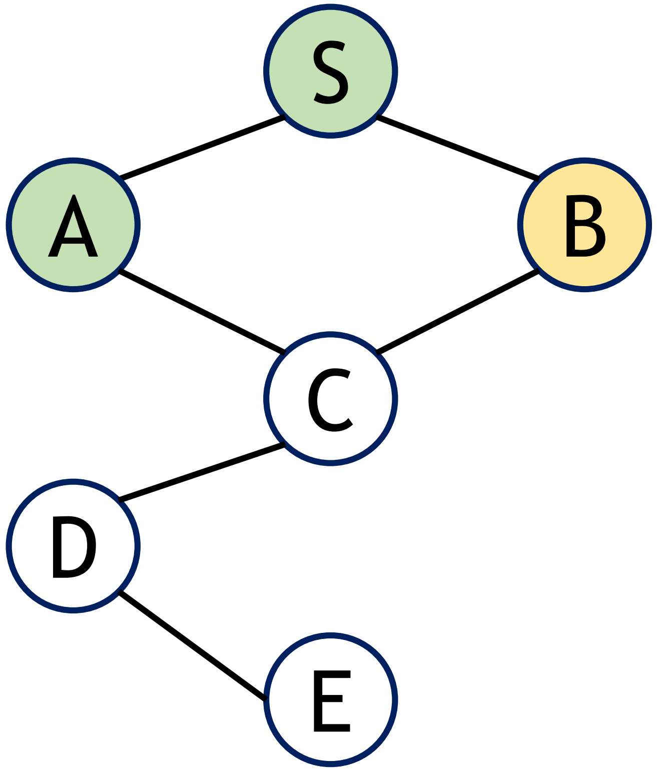
```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

S

A					

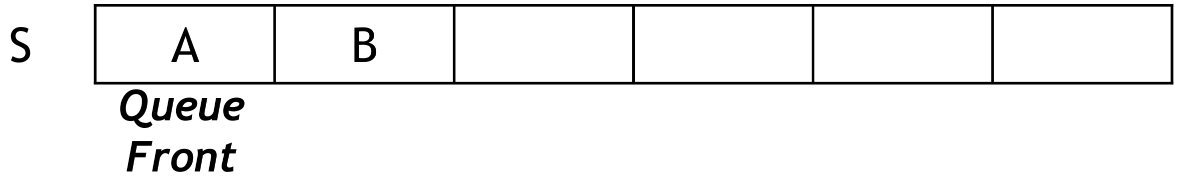
Queue Front

BFS: Example

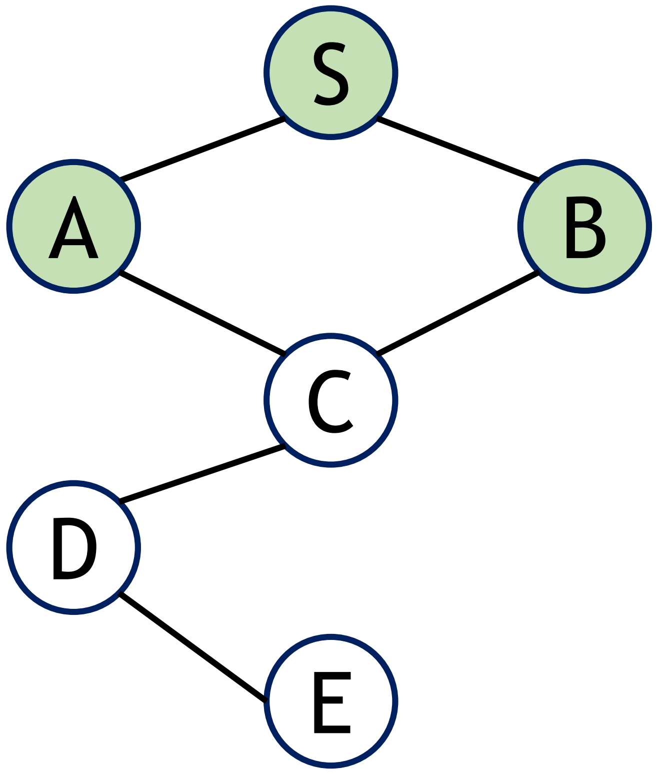


> S, A

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

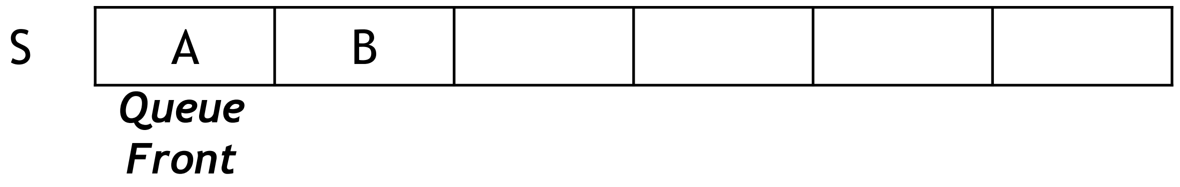


BFS: Example

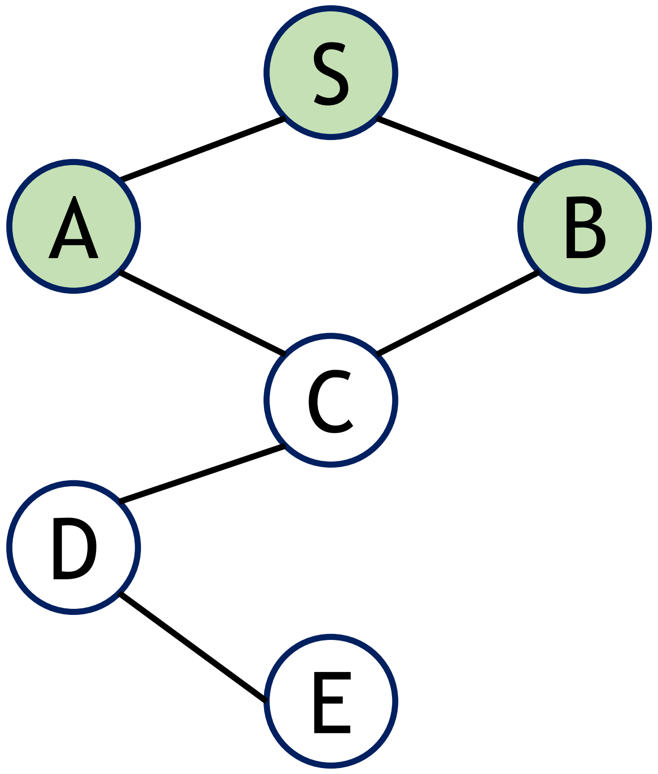


> S, A, B

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



> S, A, B

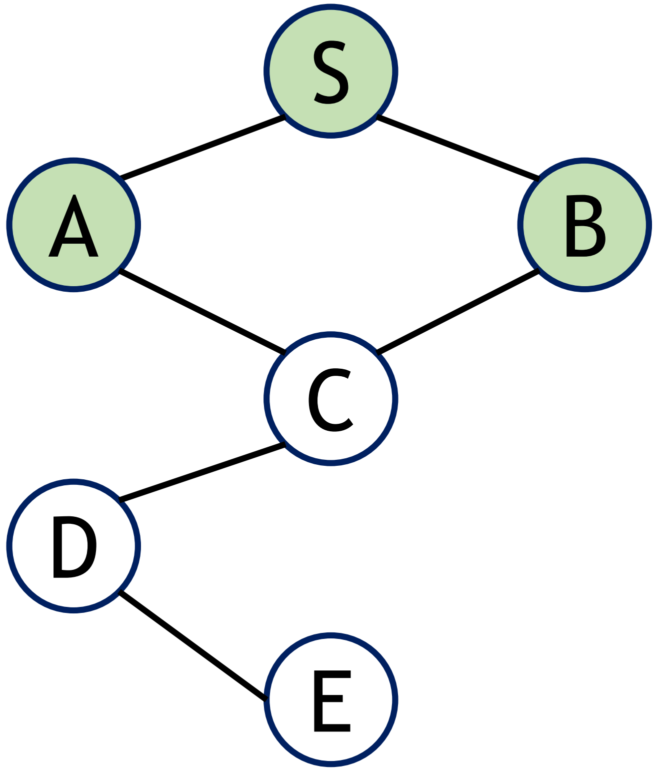
```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

S

A	B				

Queue Front

BFS: Example



> S, A, B

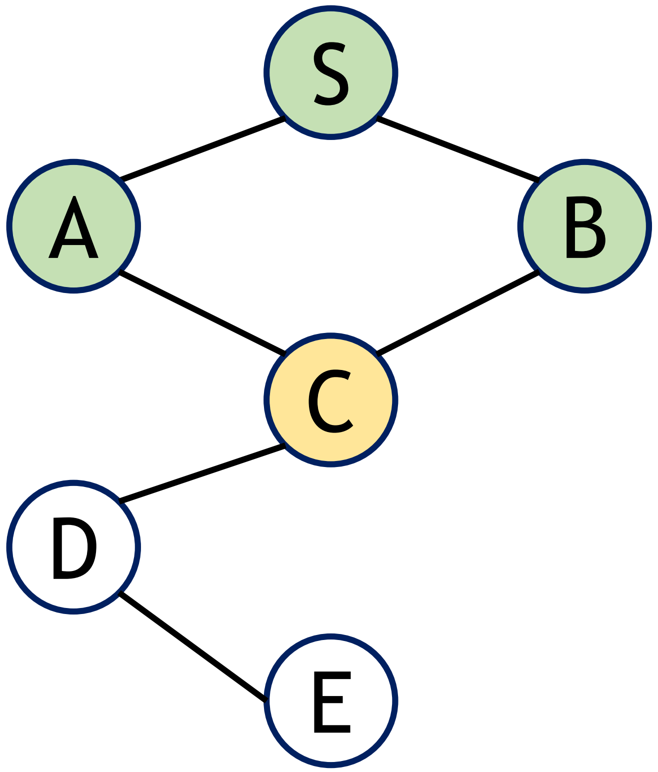
```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

A

B					

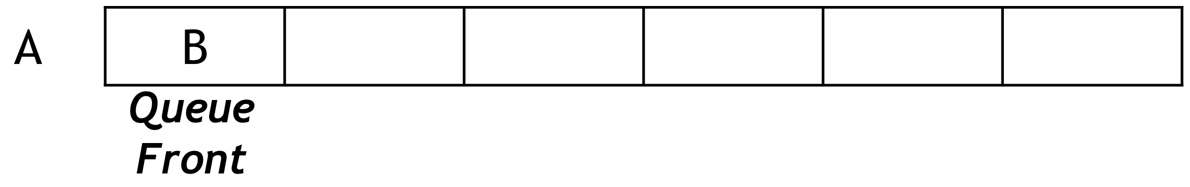
Queue Front

BFS: Example

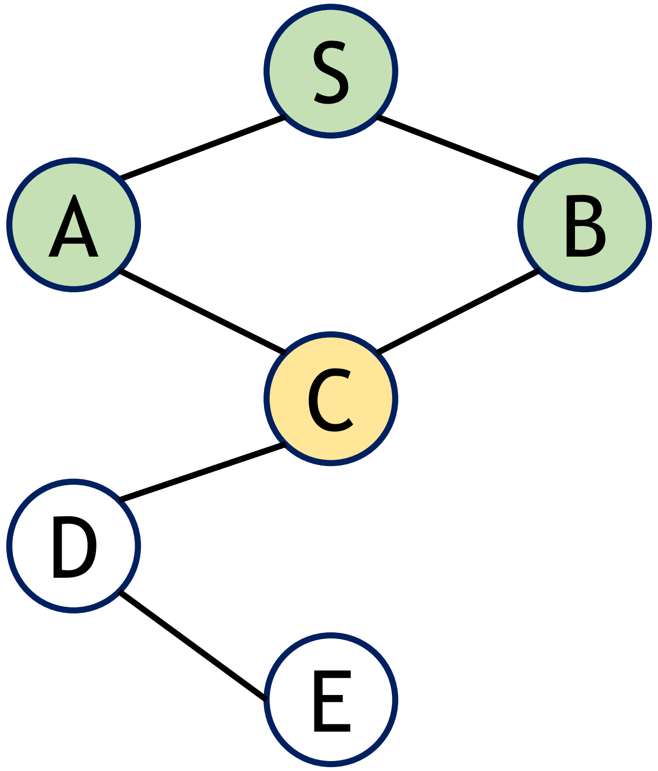


> S, A, B

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



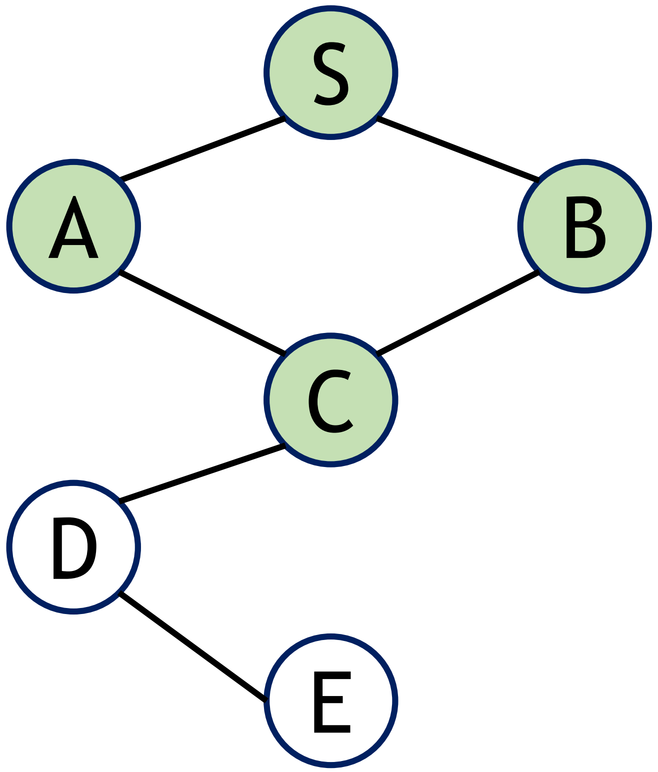
> S, A, B

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

A	B	C				

Queue Front

BFS: Example

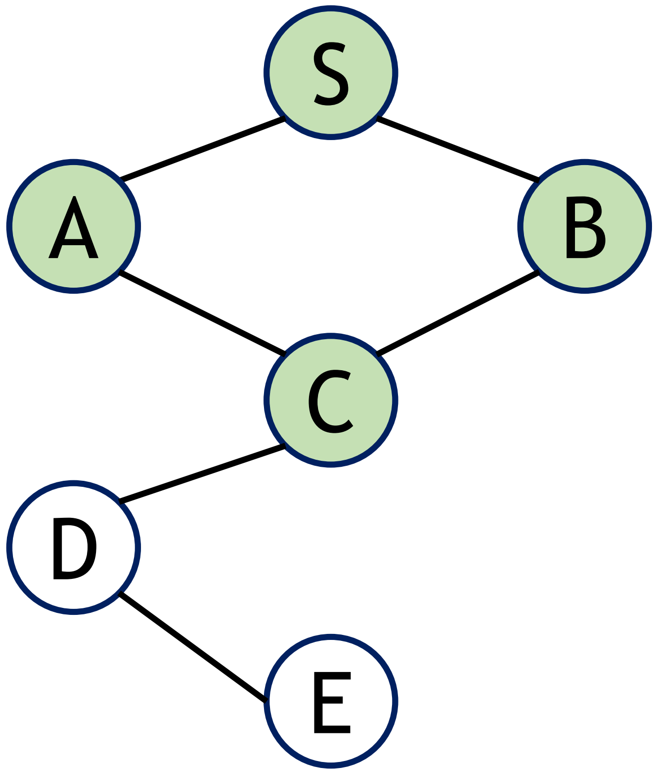


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

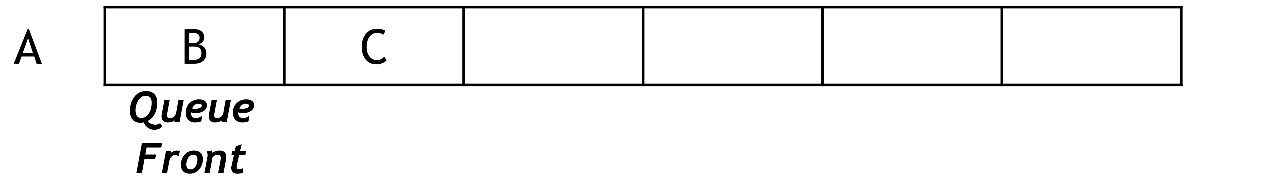
A	B	C				
	Queue Front					

BFS: Example

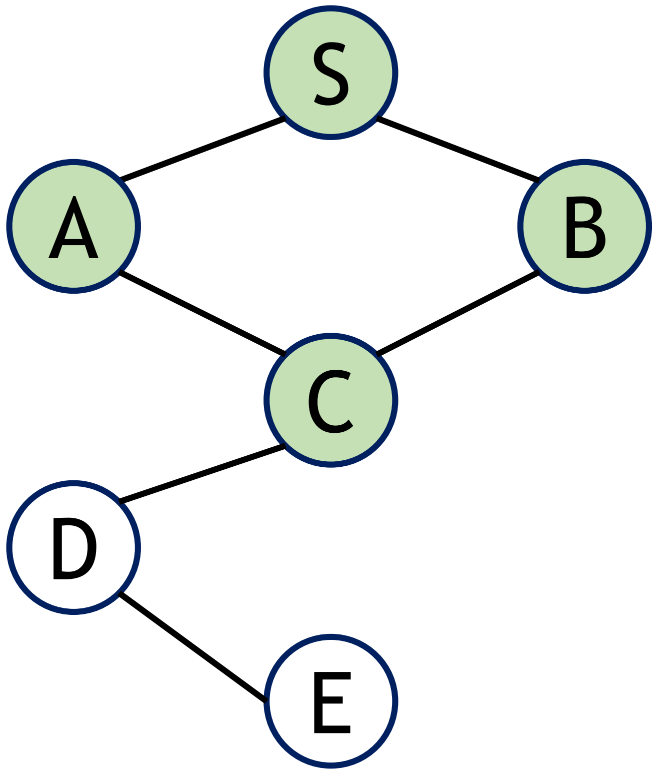


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

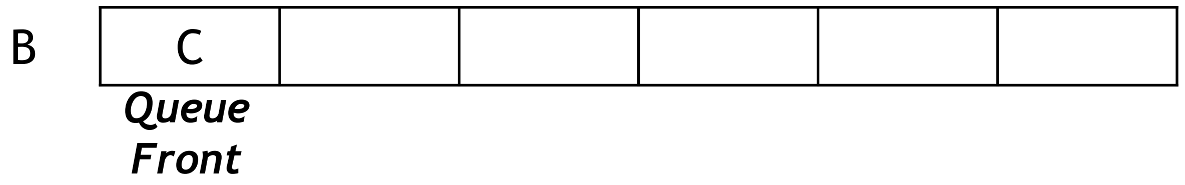


BFS: Example

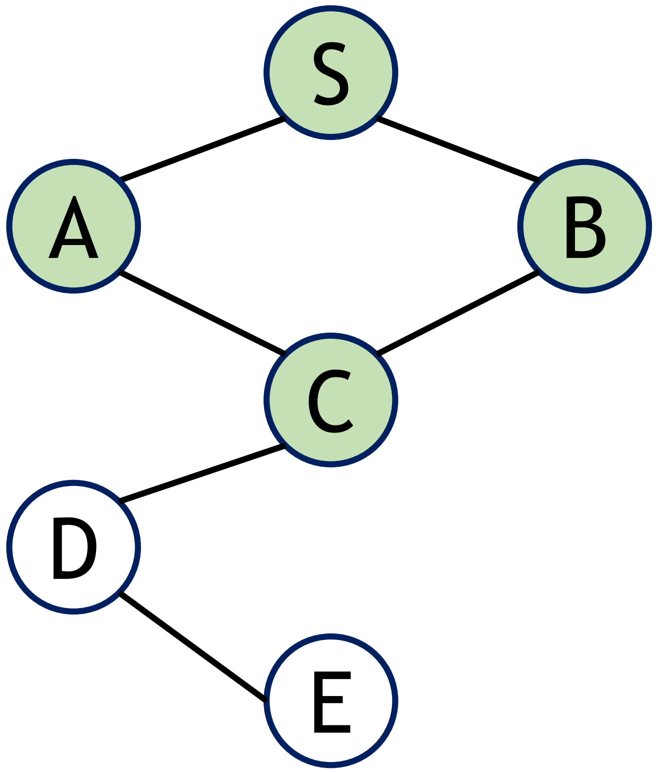


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

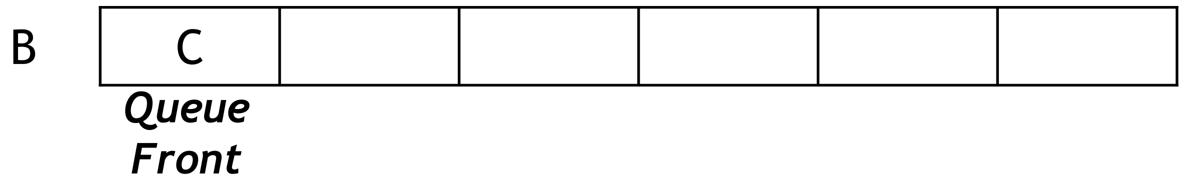


BFS: Example

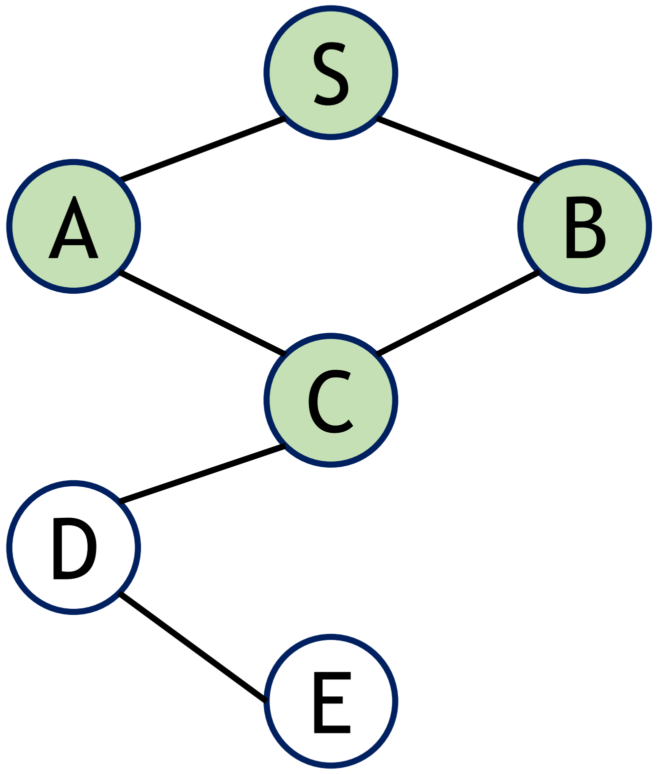


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

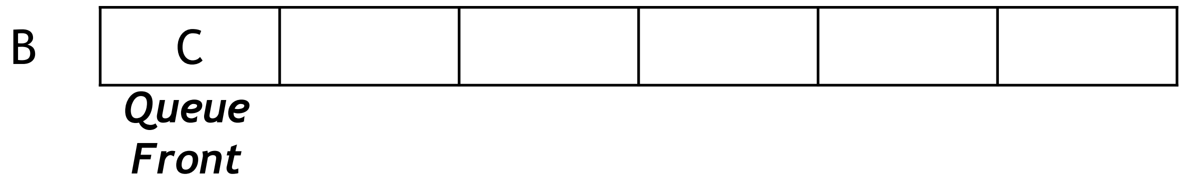


BFS: Example

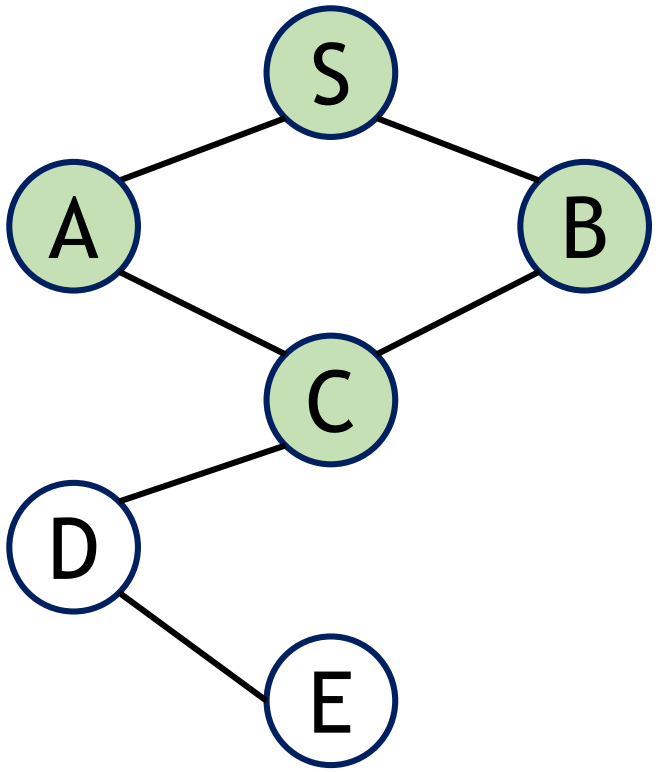


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

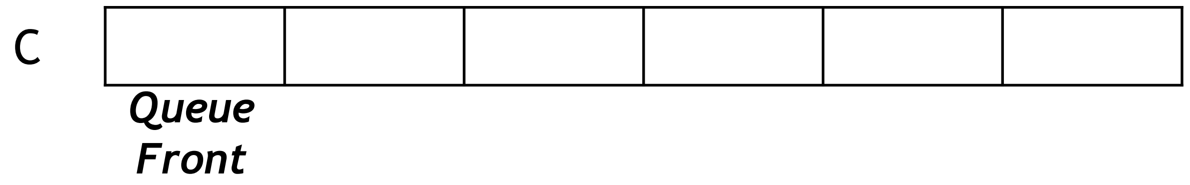


BFS: Example

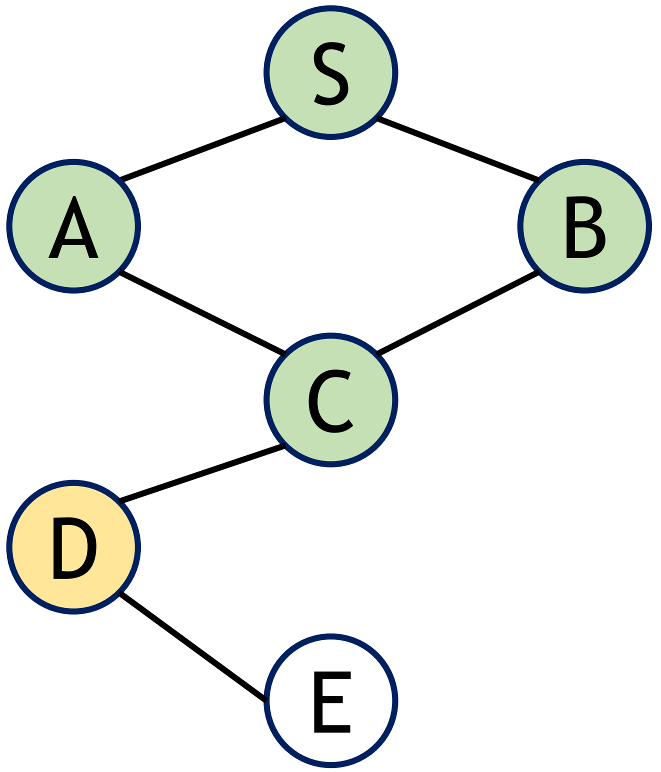


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

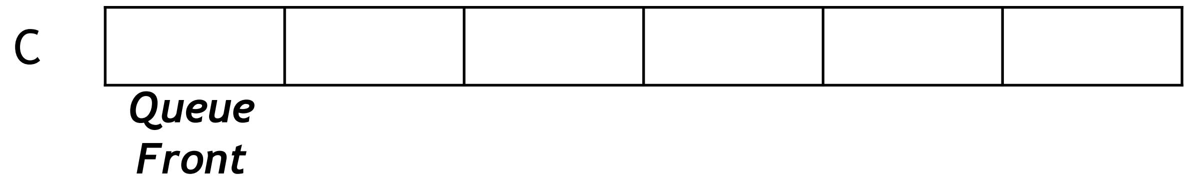


BFS: Example

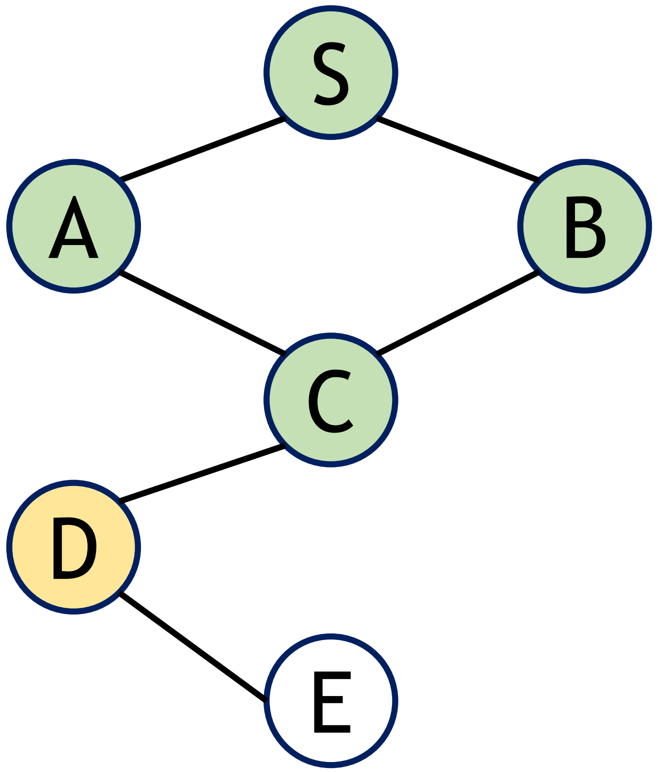


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

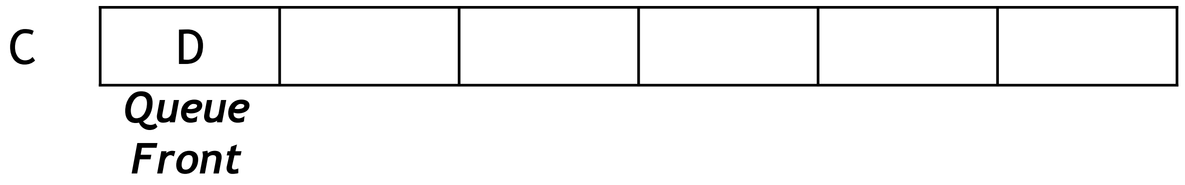


BFS: Example

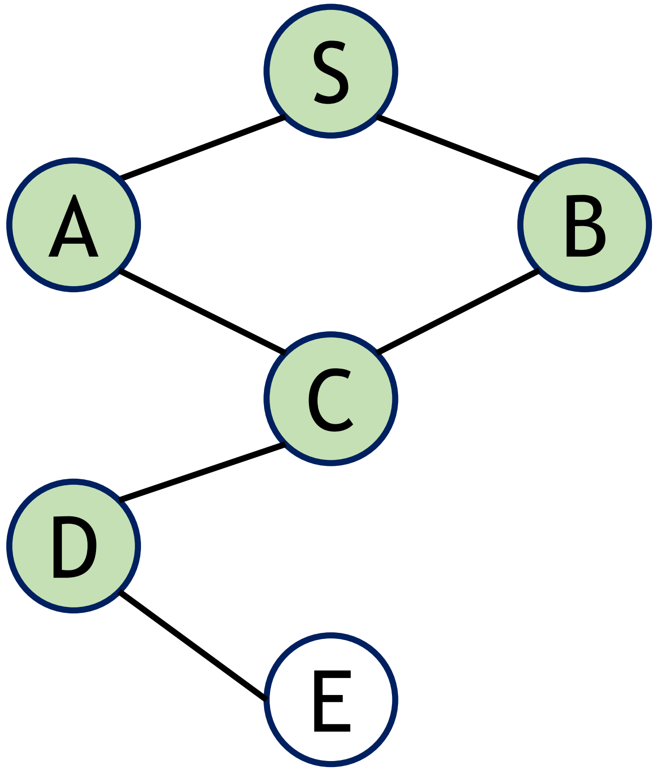


> S, A, B, C

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

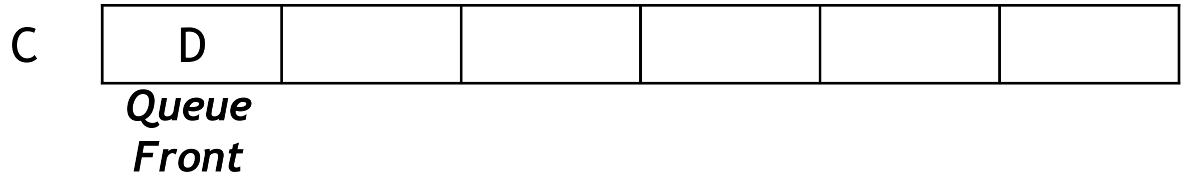


BFS: Example

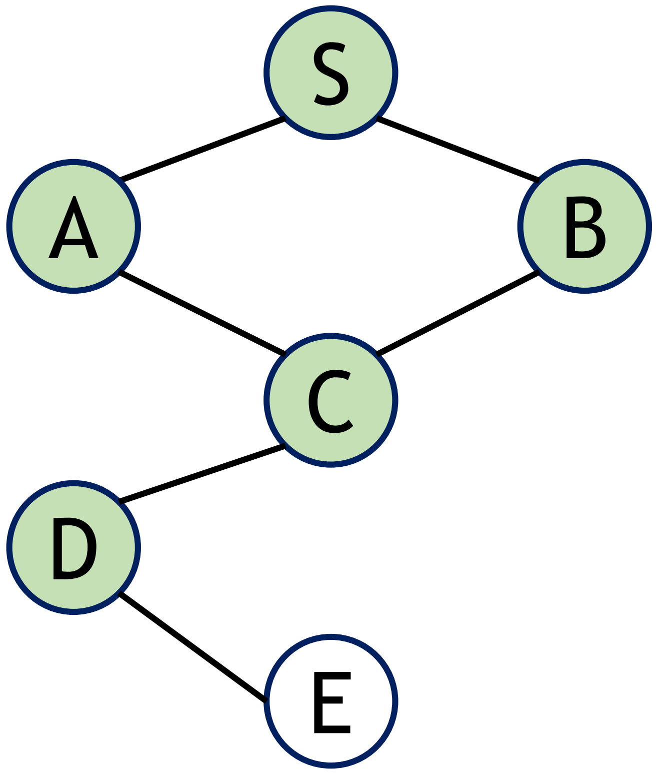


> S, A, B, C, D

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

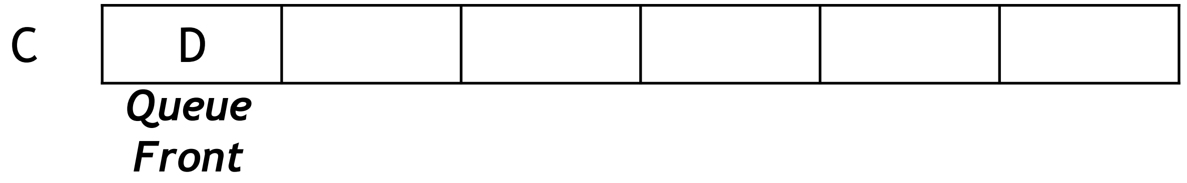


BFS: Example

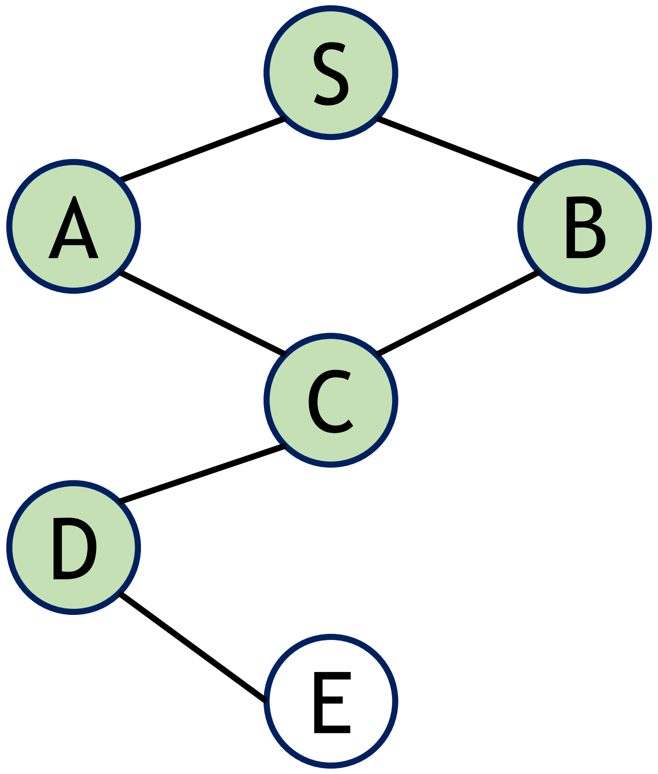


> S, A, B, C, D

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



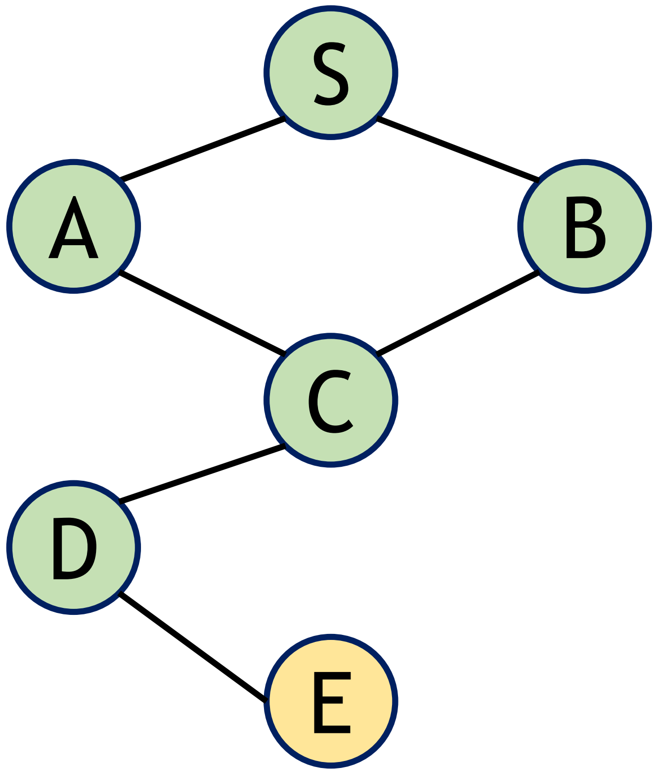
> S, A, B, C, D

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

D

Queue Front

BFS: Example



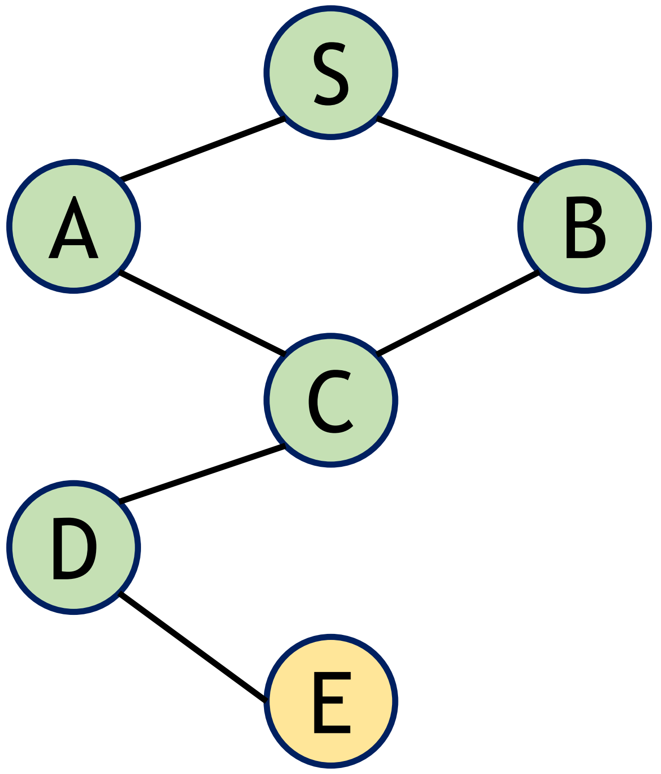
> S, A, B, C, D

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

D

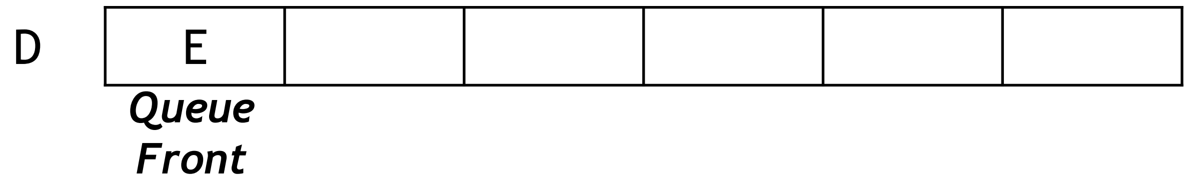
Queue Front

BFS: Example

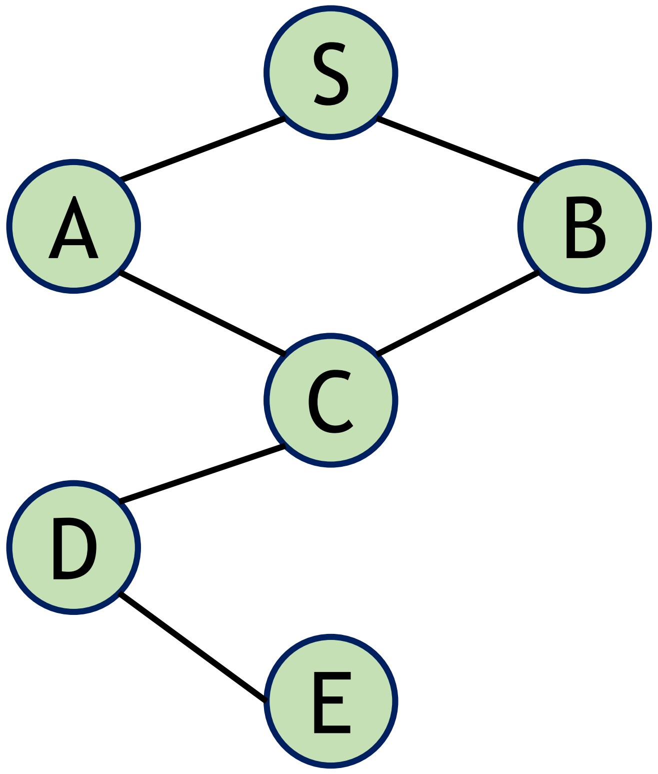


> S, A, B, C, D

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

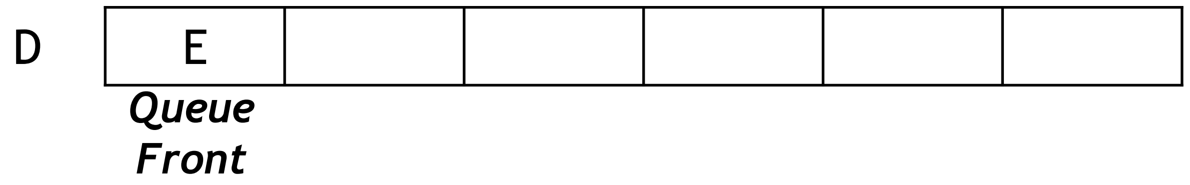


BFS: Example

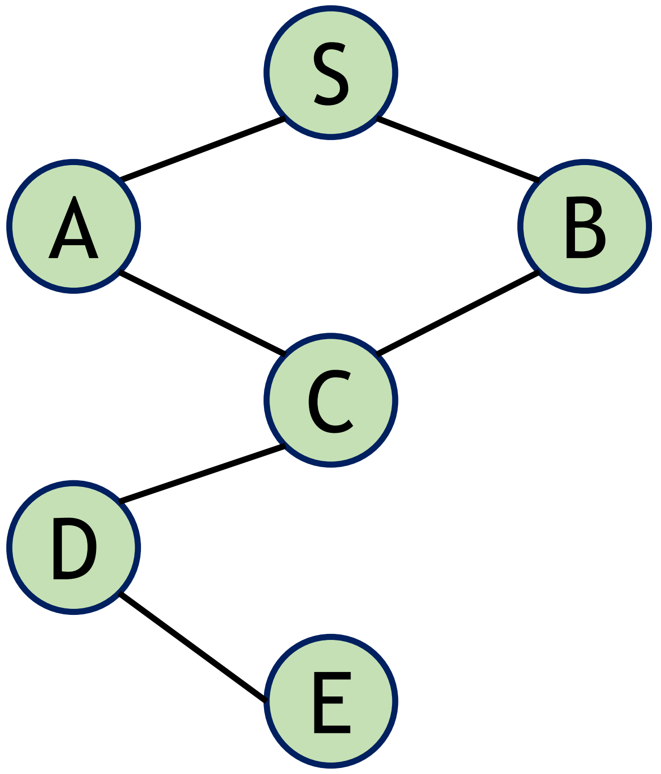


> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



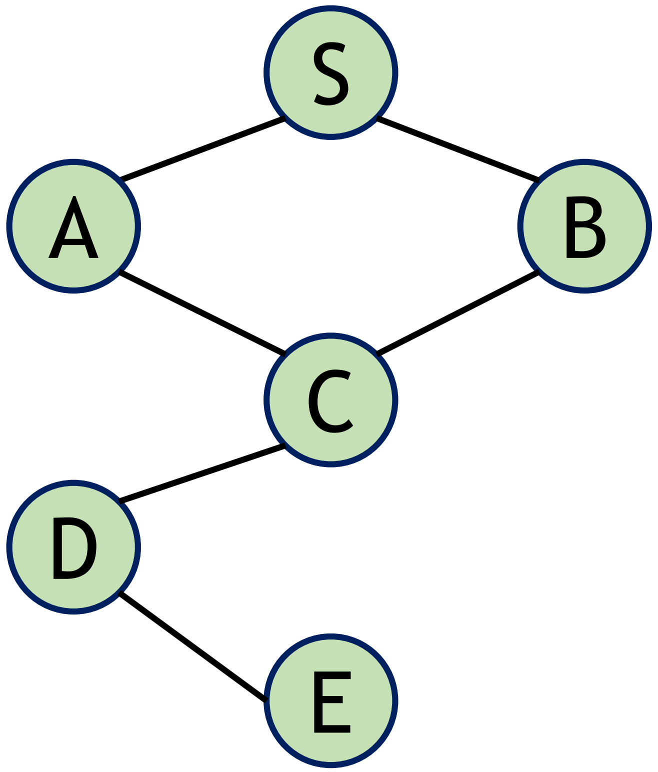
> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

D	E					

Queue Front

BFS: Example



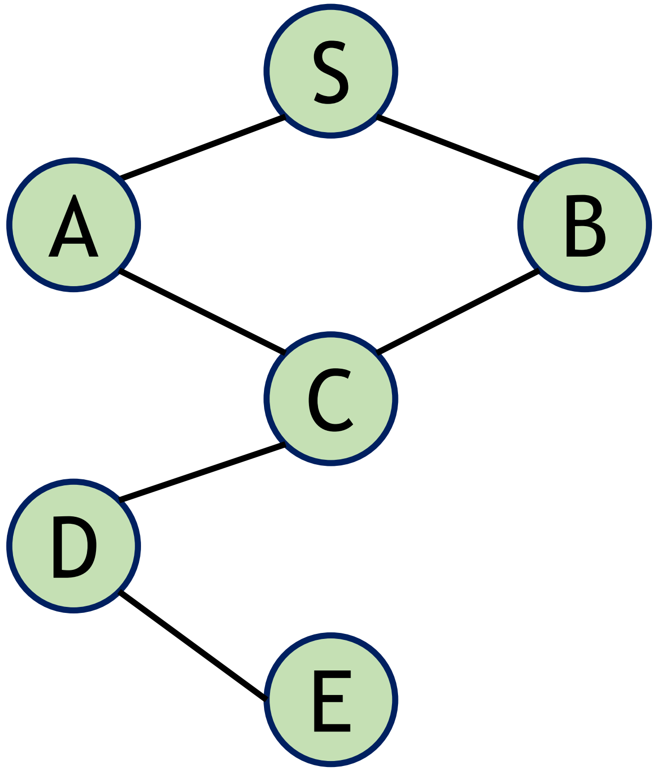
> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

E

Queue Front

BFS: Example



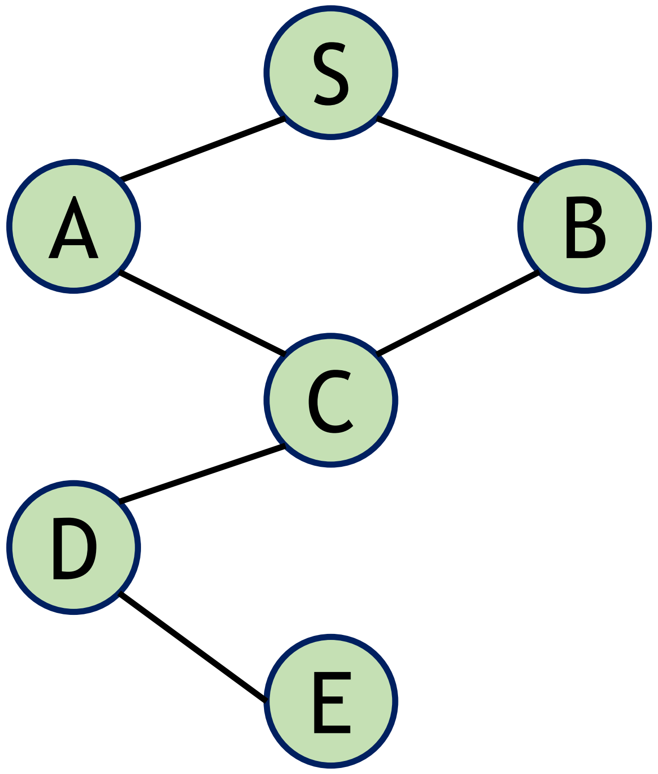
> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

E

Queue Front

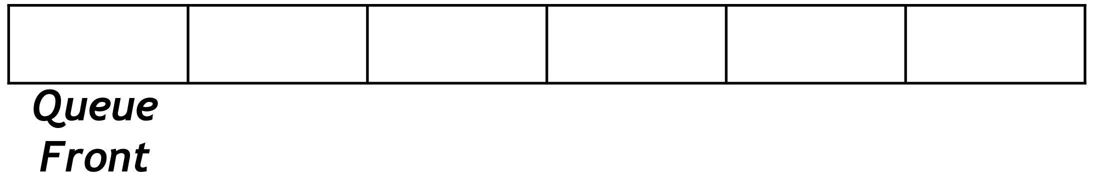
BFS: Example



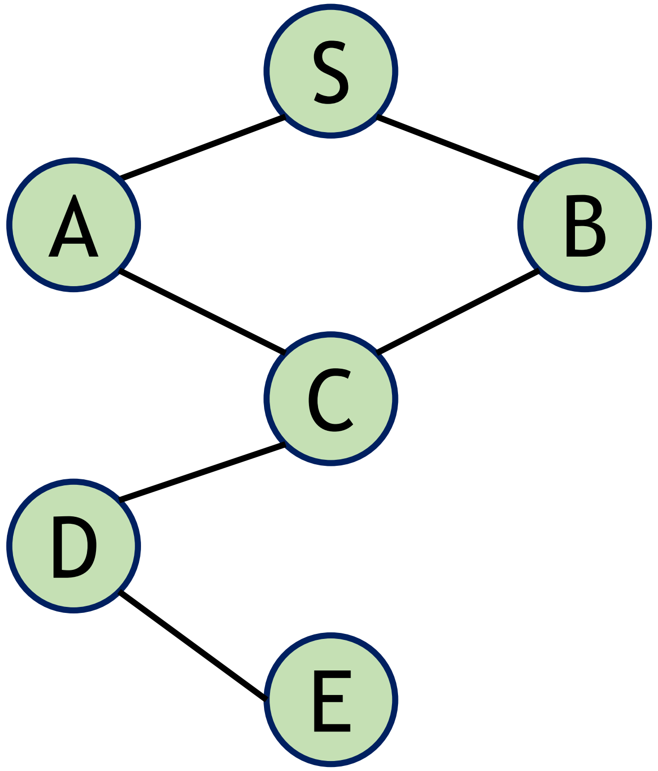
> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

E

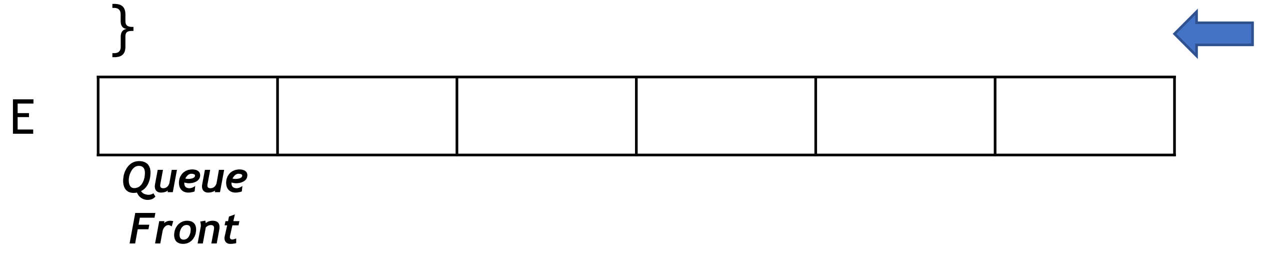


BFS: Example

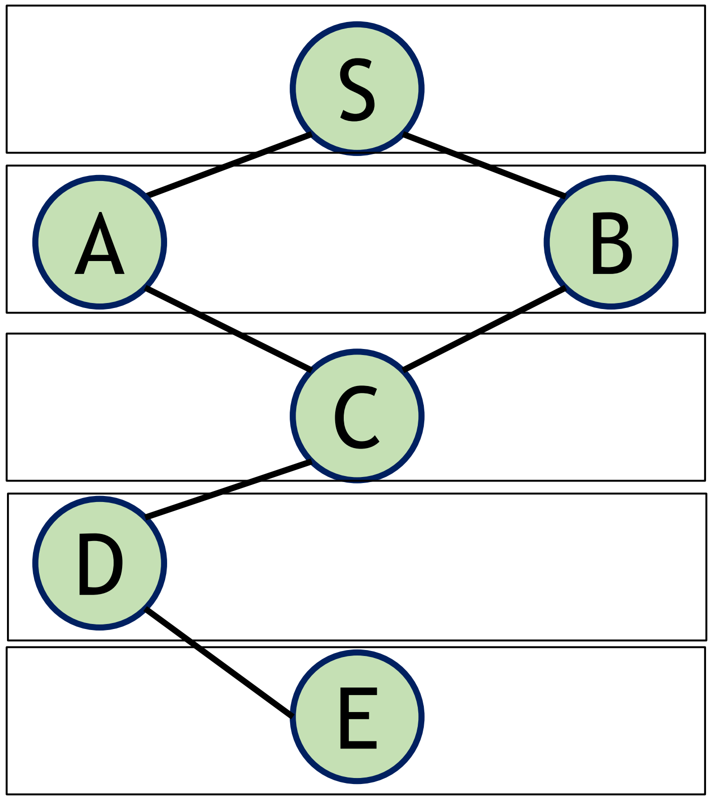


> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```



BFS: Example



> S, A, B, C, D, E

```
BFS(G, u) {
  Q = Queue()
  Q.enqueue(u)
  u.visited = true
  while !Q.isEmpty()
    v = Q.peek(); Q.dequeue();

    for each w ∈ G[v]
      if w.visited == false
        Q.enqueue(w)
        w.visited = true
}
```

BFS: Shortest Path

Use BFS to find the **shortest path** ***(number of edges)*** **between two vertices**: *[Source, Destination]*

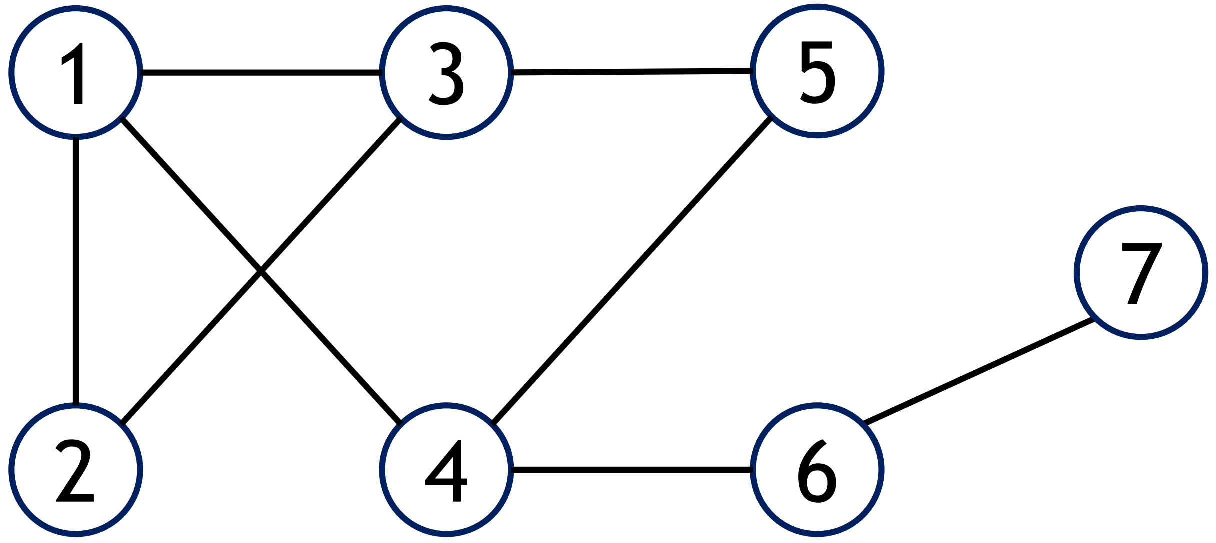
Key points for reasoning:

Can only visit a node once

Each node visits ALL of its neighbors first (1 hop at a time)

Proof by example 🙊

BFS: Shortest Path



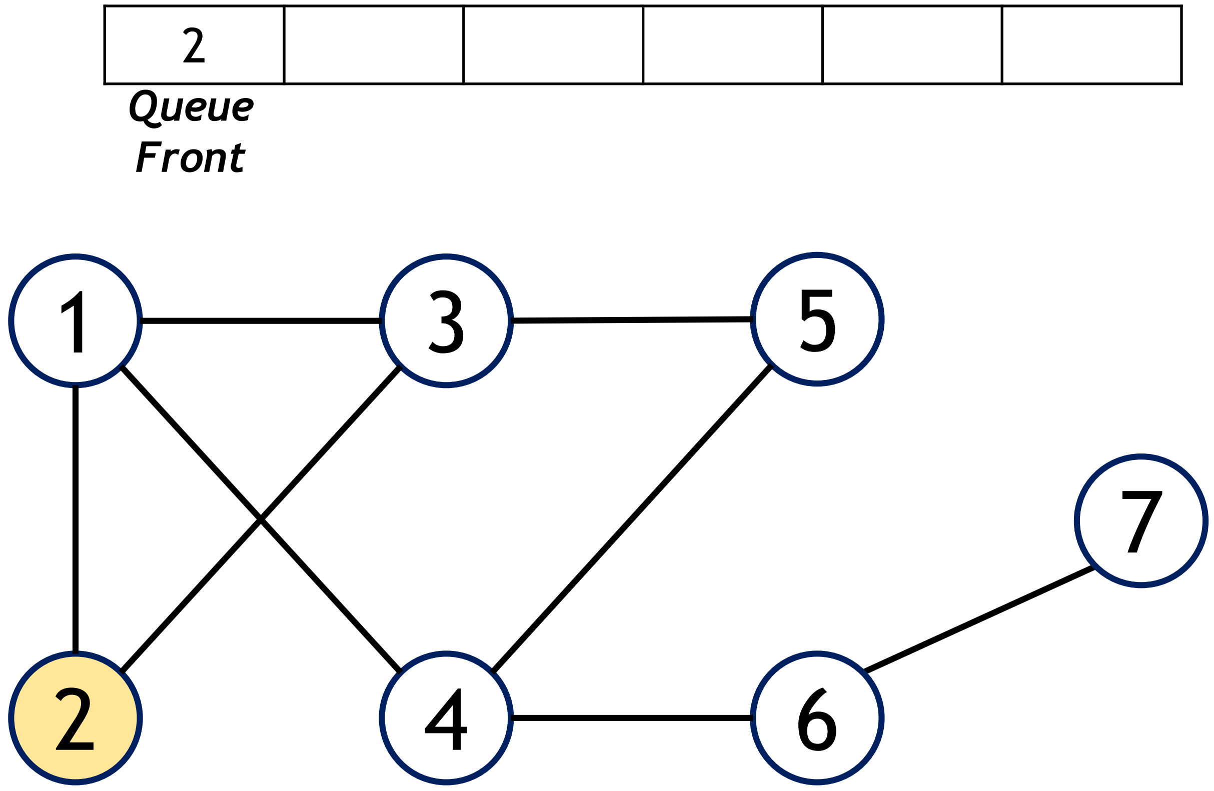
Paths between 2 and 5

2 -> 1 -> 3 -> 5

2 -> 1 -> 4 -> 5

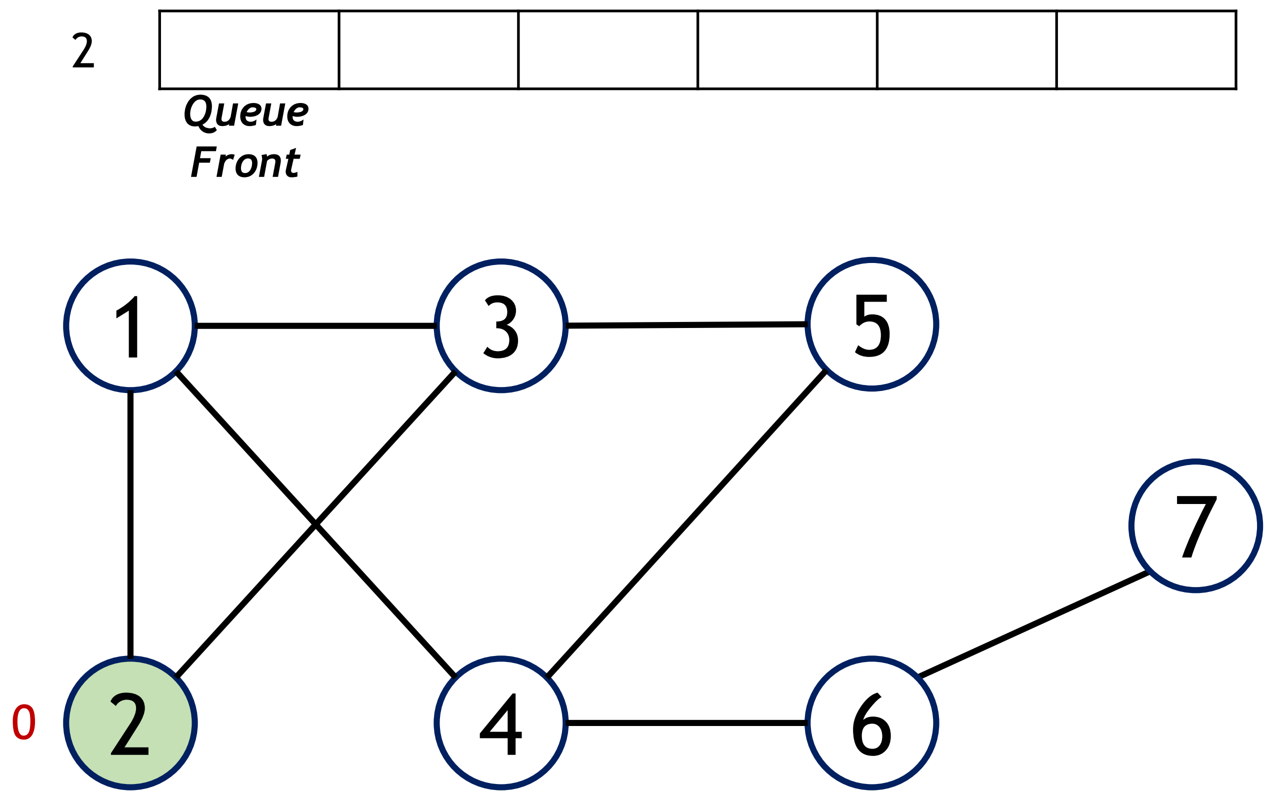
2 -> 3 -> 5

BFS: Shortest Path



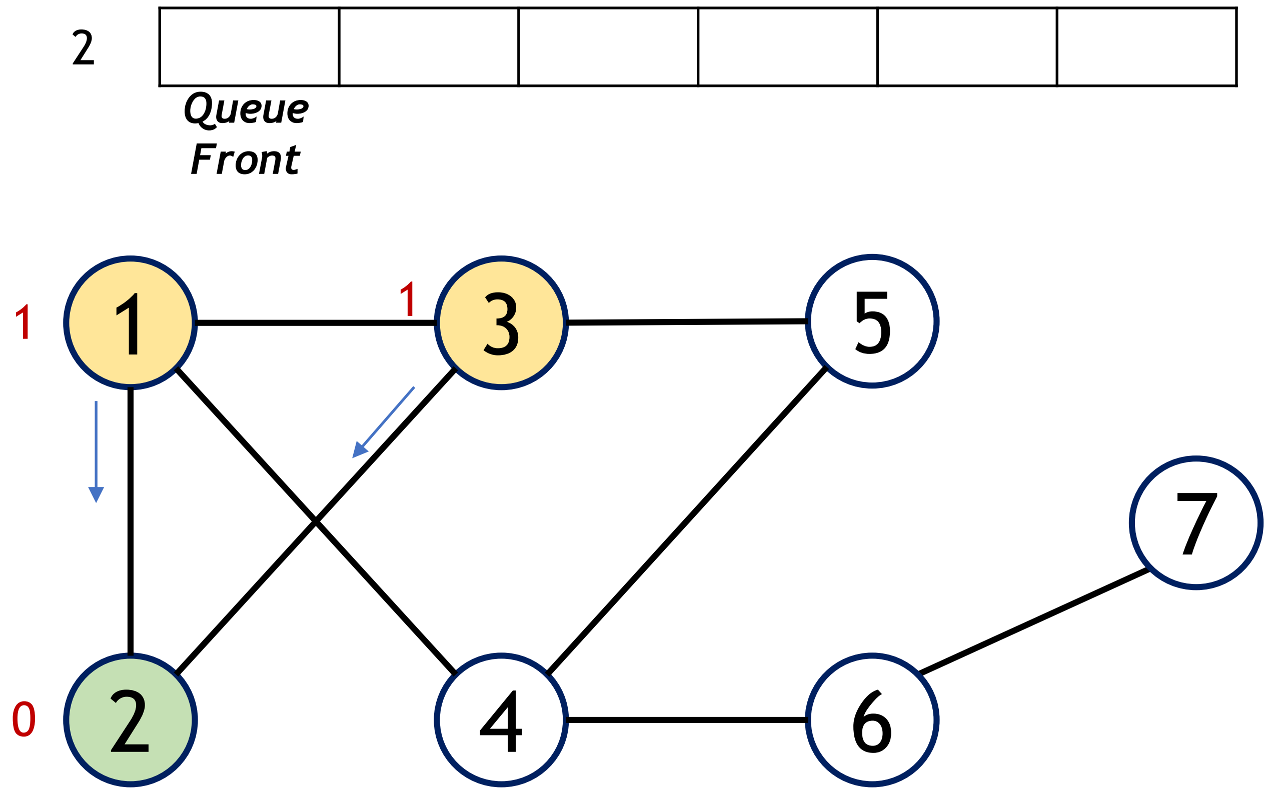
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
          u.prev = v; u.dist = v.dist + 1;
          u.visited = true;
          Q.enqueue(u)

          if u == destination
            // destination found
            // how will you trace path
// Not found
```

BFS: Shortest Path



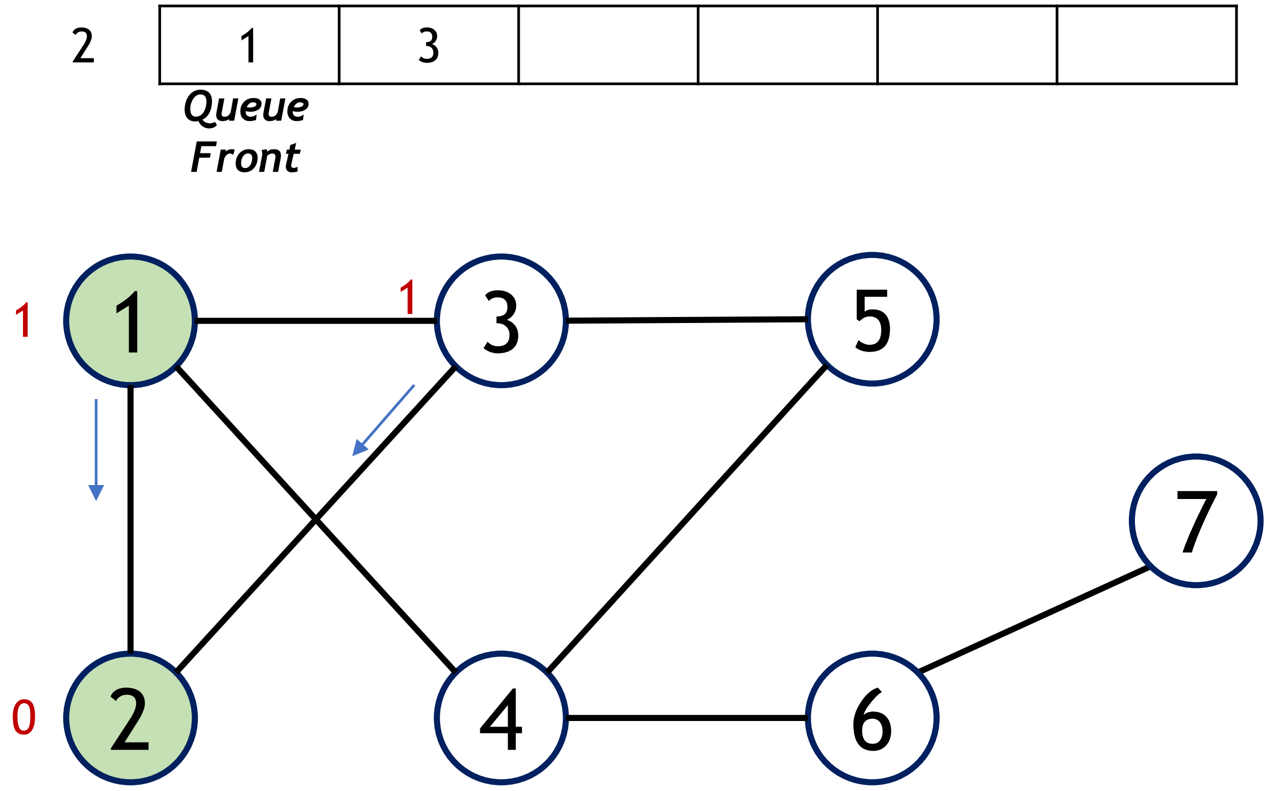
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
        u.prev = v; u.dist = v.dist + 1;
        u.visited = true;
        Q.enqueue(u)

        if u == destination
          // destination found
          // how will you trace path
// Not found
```

BFS: Shortest Path



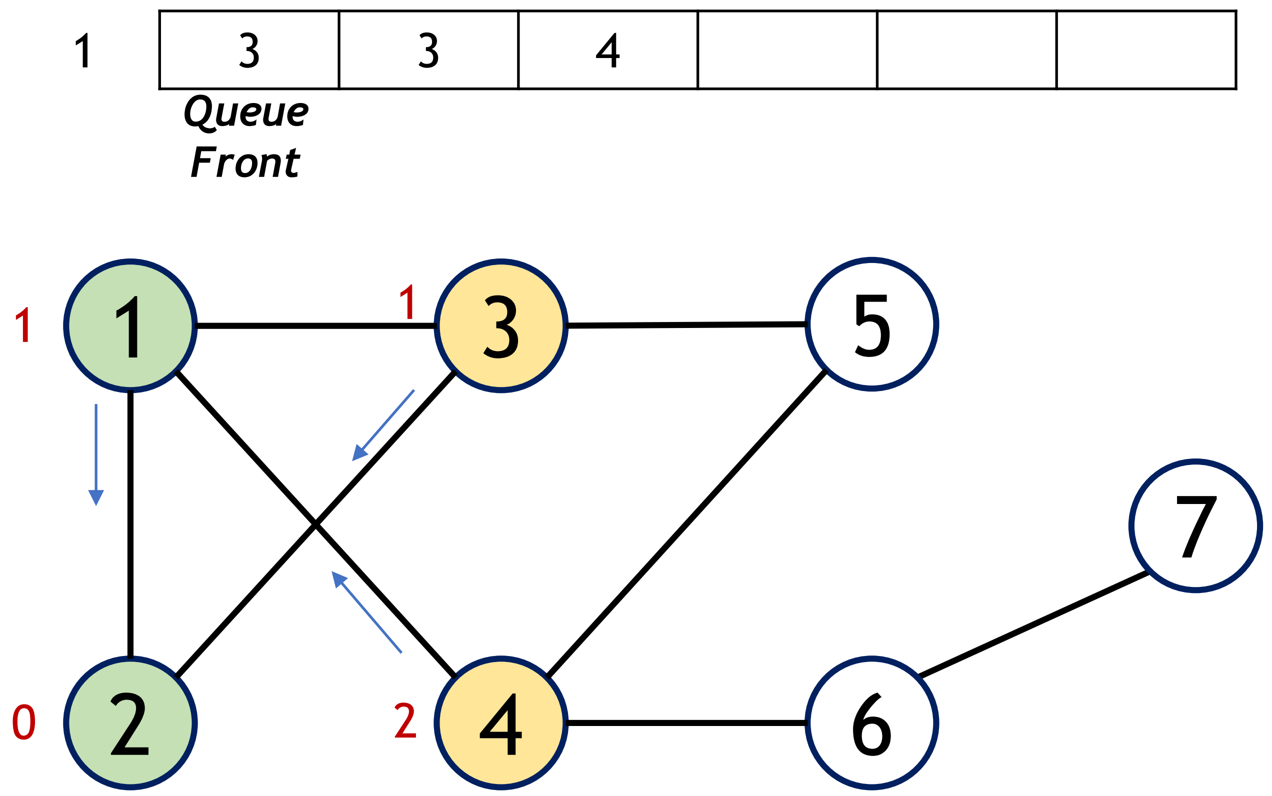
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
        u.prev = v; u.dist = v.dist + 1;
        u.visited = true;
        Q.enqueue(u)

        if u == destination
          // destination found
          // how will you trace path
// Not found
```

BFS: Shortest Path



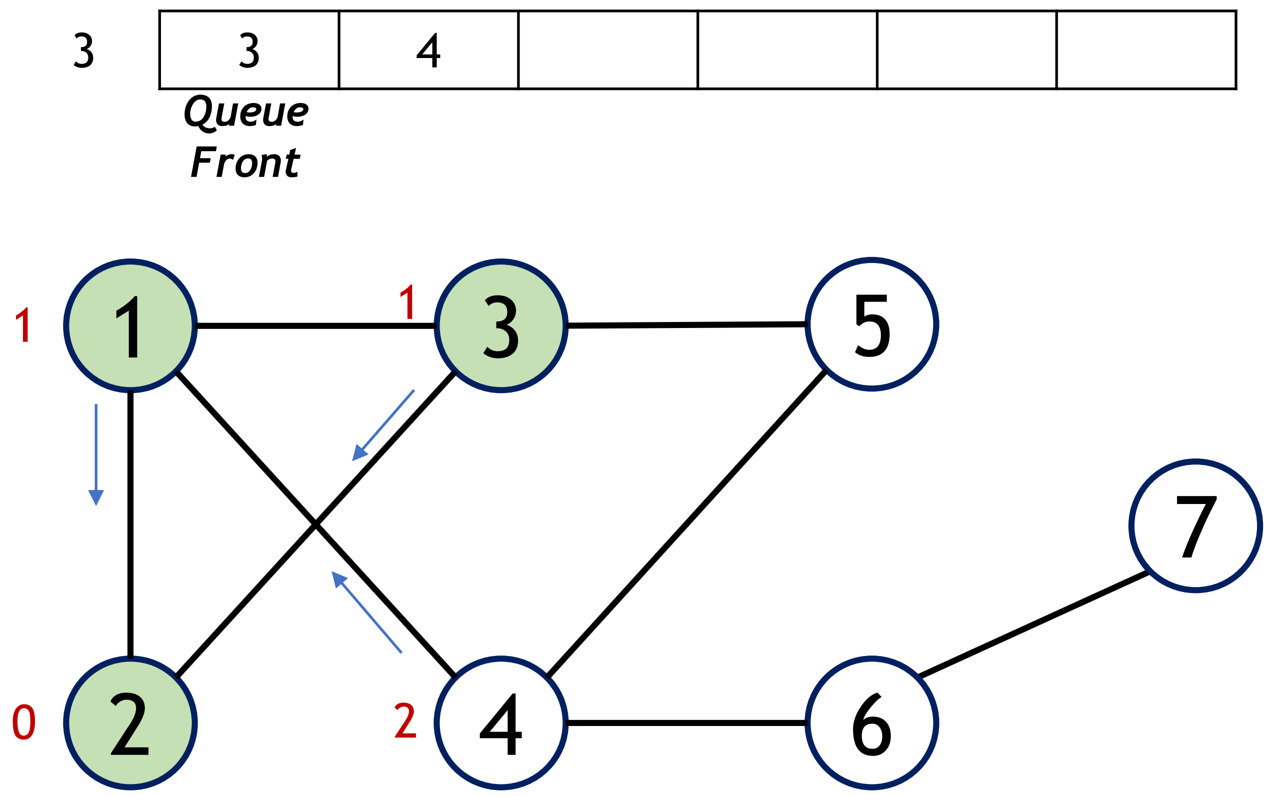
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
          u.prev = v; u.dist = v.dist + 1;
          u.visited = true;
          Q.enqueue(u)

          if u == destination
            // destination found
            // how will you trace path
// Not found
```

BFS: Shortest Path



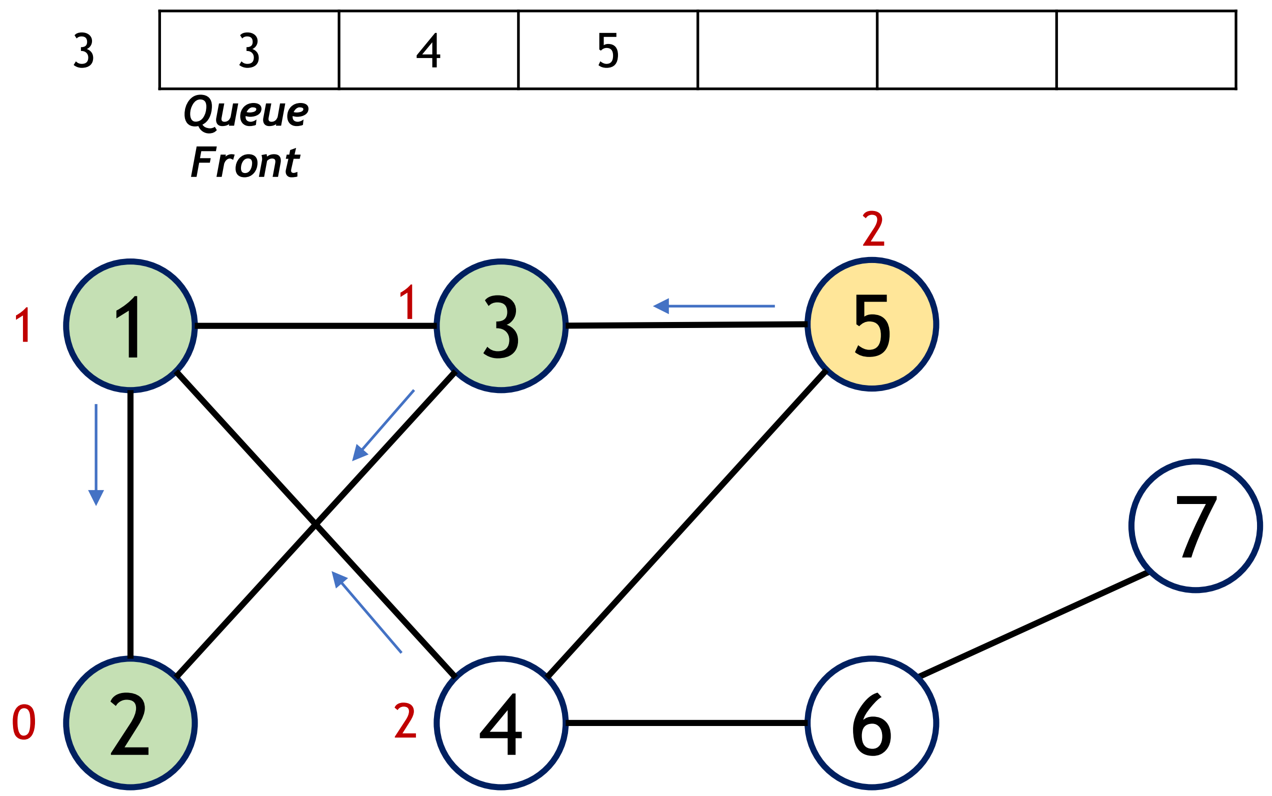
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
        u.prev = v; u.dist = v.dist + 1;
        u.visited = true;
        Q.enqueue(u)

        if u == destination
          // destination found
          // how will you trace path
// Not found
```

BFS: Shortest Path



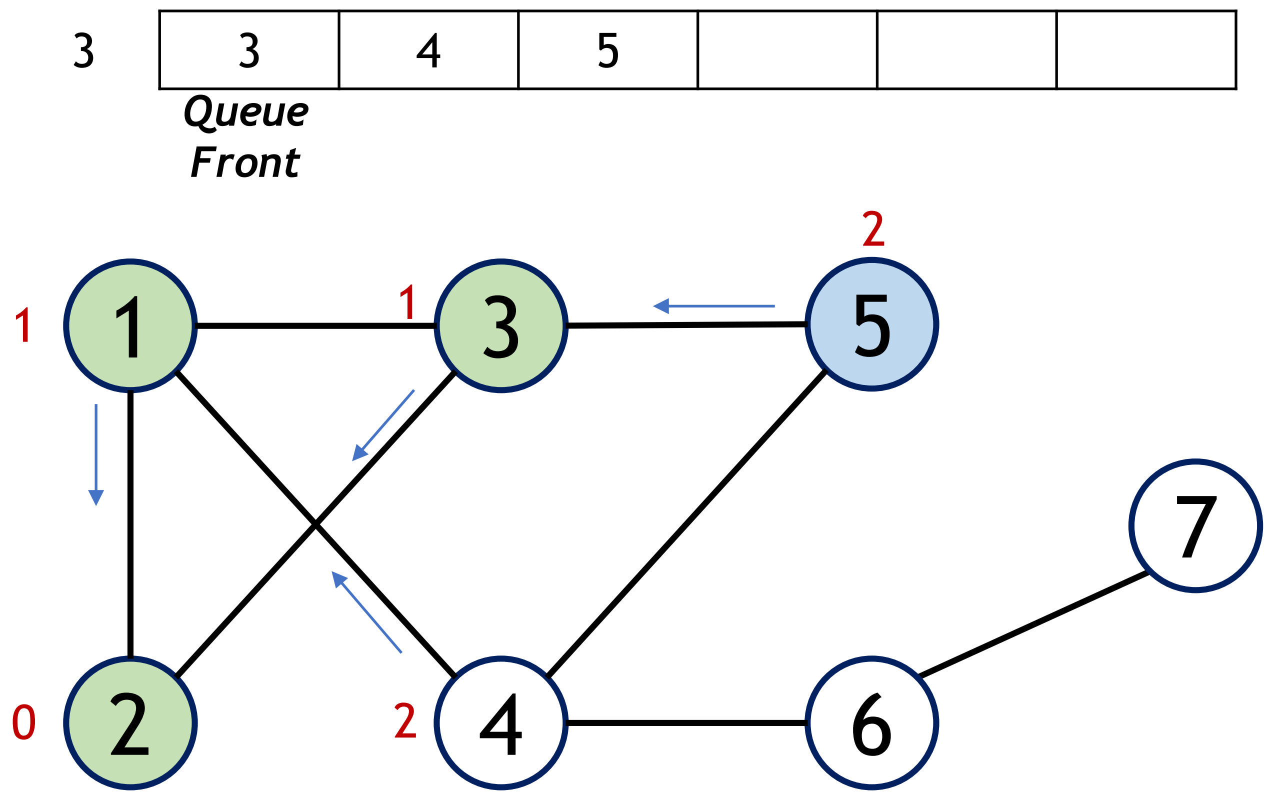
```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
          u.prev = v; u.dist = v.dist + 1;
          u.visited = true;
          Q.enqueue(u)

          if u == destination
            // destination found
            // how will you trace path
// Not found
```

BFS: Shortest Path



```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
          u.prev = v; u.dist = v.dist + 1;
          u.visited = true;
          Q.enqueue(u)

          if u == destination
            // destination found
            // how will you trace path
// Not found
```

BFS: Shortest Path



```
Q.enqueue(s)
s.dist = 0; s.prev = NULL;
while !Q.isEmpty()
    v = Q.peek(); Q.pop();
    v.visited = true;
    for each u in G[v]
      if u.visited == false
         u.prev = v; u.dist = v.dist + 1;
         u.visited = true;
         Q.enqueue(u)

         if u == destination
           // destination found
           // how will you trace path
// Not found
```

Exercise

BFS, Shortest Path

Exercise: Note

```
struct vertex;

struct adjVertex{
  vertex *v;
};

struct vertex{
  std::string name;
  bool visited = false;
  std::vector<adjVertex> adj;
};
```

Type of **`vertices`**: `vector<vertex*>`

Vertex at some index:
  `vertices[i]` : type vertex*

Dereferencing a pointer (vertex*): `->`

Dereferencing a struct (adjVertex): `.`

Careful with dereferencing!

Exercise: Silver

```
Expected Output: 2
```

Implement:

```
void Graph::findShortestPath(int src, int dest)
```

Notes

Why does the question specify "unweighted, undirected"?

Use the pseudocode for "BFS: Shortest Path"

Observe `Graph.hpp` *for the fields that vertex has*

Exercise: Gold

```
Expected Output: 1 4 6
```

Implement:

```
void Graph::printPath(int src, int dest)
```

Notes

Assume that `findShortestPath` *has been called already*

Print the path from source to destination

Which data member of the vertex is of interest?