Graph Algorithms

Andreas Klappenecker

Graphs

A graph is a set of vertices that are pairwise connected by edges.

We distinguish between directed and undirected graphs.

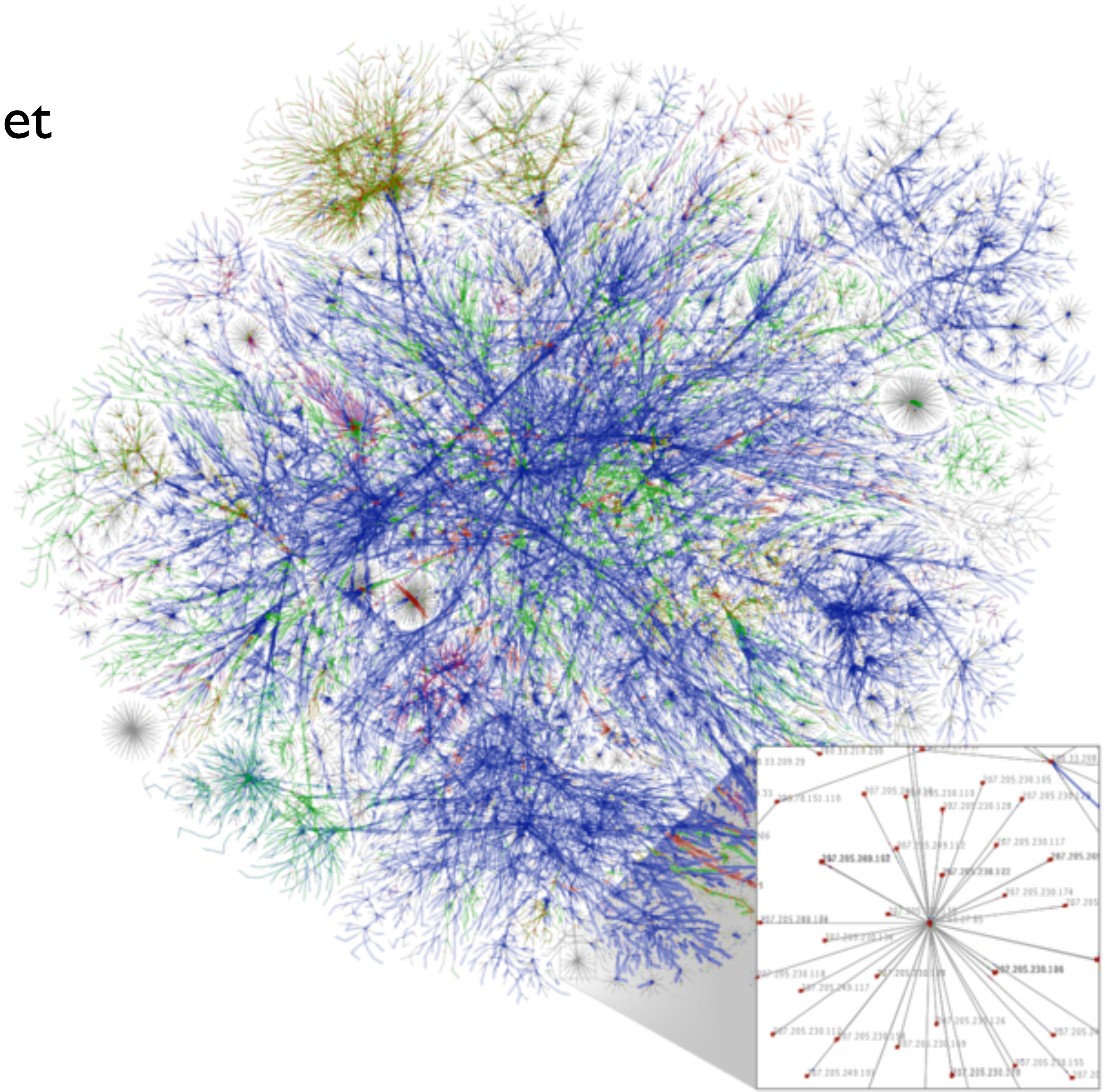
Why are we interested in graphs?

- Graphs are a very useful abstraction
- Graphs have many interesting applications
- Thousands of graph algorithms are known

Versatile Abstraction

Application	Vertices	Edges
Traffic	Intersections	Roads
Social Network	People	Friendship
Internet	Class C network	Connection
Game	Board Position	Legal Move
Erdos number	People	Coauthored Paper
CMOS Circuits	FET, Vdd, Vss, I/O	Wires
Financial	Stock, Currency	Transactions
Programs	Procedures	Procedure Call f->g

The Internet



Undirected Graphs

An undirected graph is a pair $G=(V,E)$, where

- V is a finite set
- E is a subset of $\{ e \mid e \subseteq V, |e|=2 \}$.

The elements in V are called vertices.

Elements in E are called edges, e.g. $e=\{u,v\}$, written $e=(u,v)$.

Self-loops are not allowed for undirected graphs, $e \neq \{u,u\}=\{u\}$.

Directed Graphs

An directed graph is a pair G=(V,E), where

- V is a finite set
- E is a subset of V x V

The set of edges does not need to be symmetric.

Thus, if (u,v) is an edge, then (v,u) does not need to be an edge.

We illustrate a directed edge often by an arrow u -> v.

Graph Terminology

If e=(u,v) is an edge in a graph, then v is called adjacent to u.

For undirected graphs, adjacency is a symmetric relation.

The edge e is said to be incident to u and v.

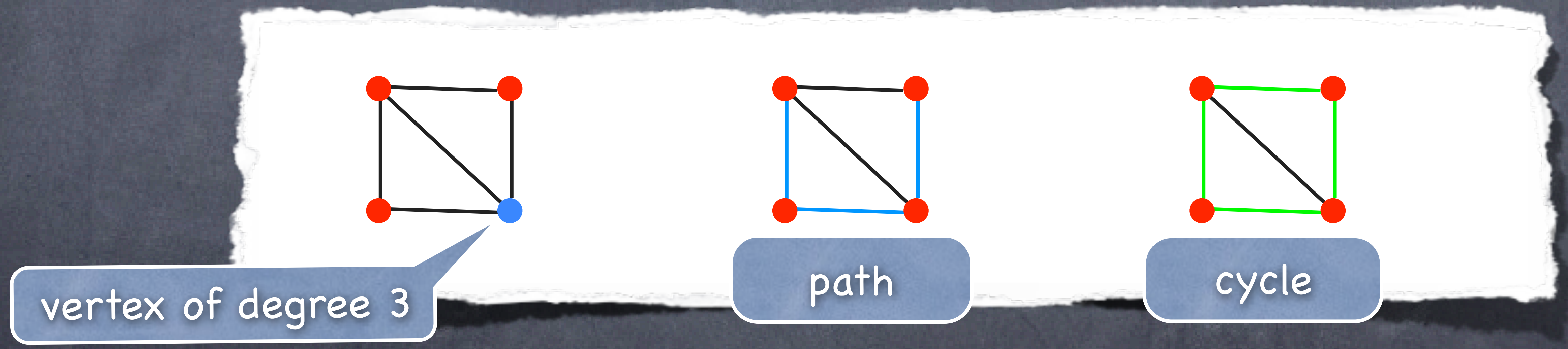
The number of edges incident to a vertex is called the degree of the vertex.

Graph Terminology

A path is a sequence of vertices that are connected by edges.

A cycle is a path whose first and last vertices are the same.

Two vertices are connected if and only if there is a path between them.



Breadth-First Search

Breadth First Search (BFS)

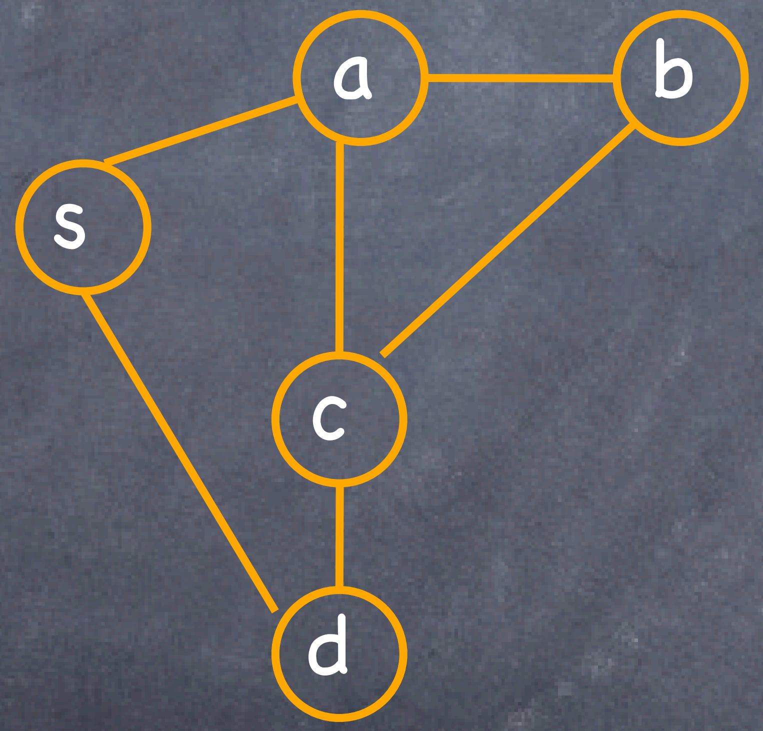
Input: A graph G = (V,E) and source node s in V

```
mark all nodes v in V as unvisited

mark source node s as visited

enq(Q,s)      // first-in first-out queue Q

while (Q is not empty) {
   u := deq(Q);
   for each unvisited neighbor v of u {
       mark v as visited; enq(Q,v);
   }
}
```

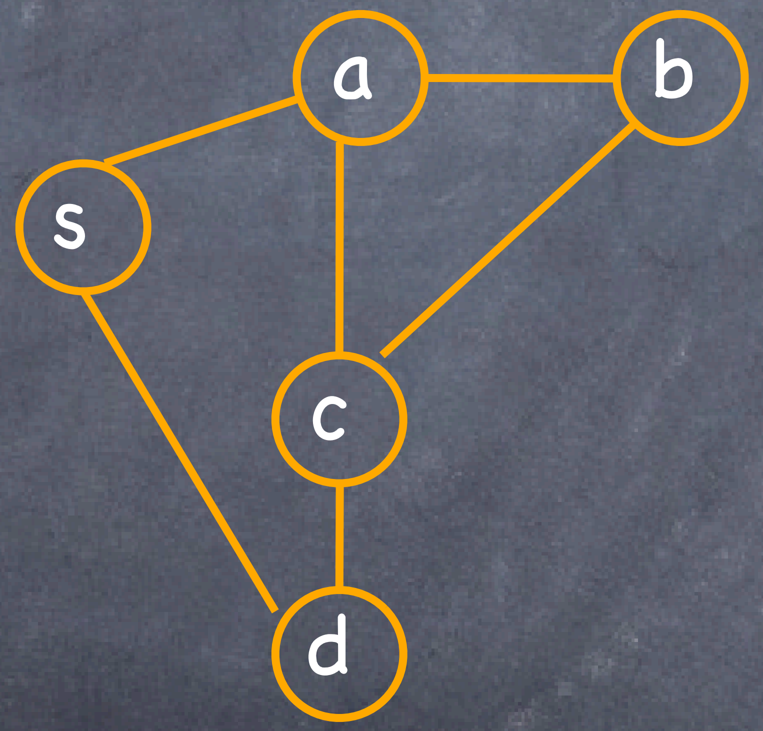
Example

www.cs.princeton.edu/courses/archive/spr10/cos226/…/51demo-bfs.ppt

BFS Example

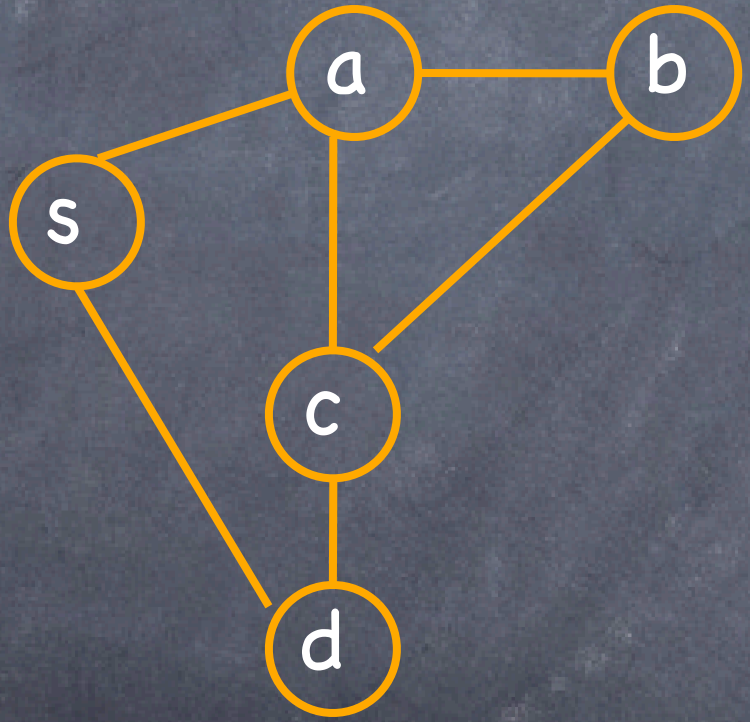


BFS Example



Visit the nodes in the order:

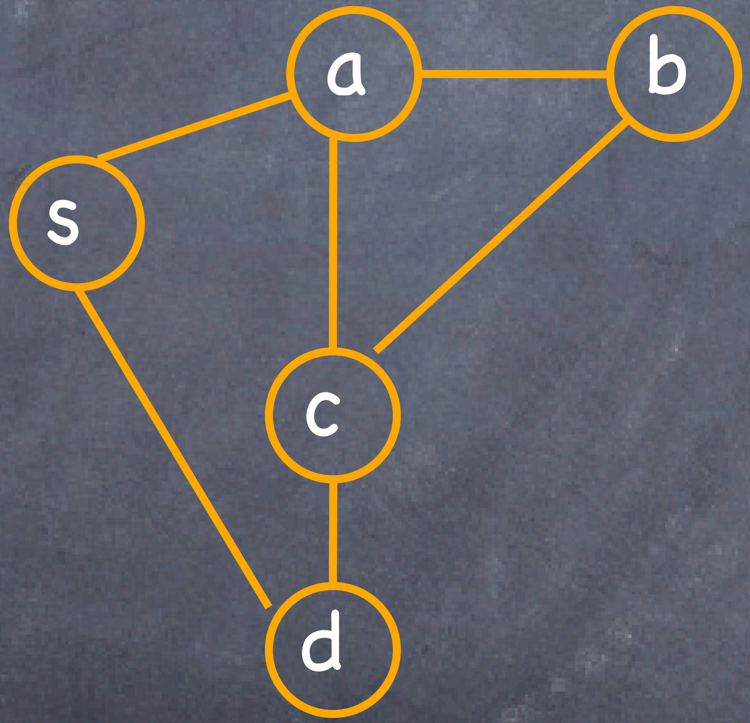
BFS Example



Visit the nodes in the order:

s

BFS Example

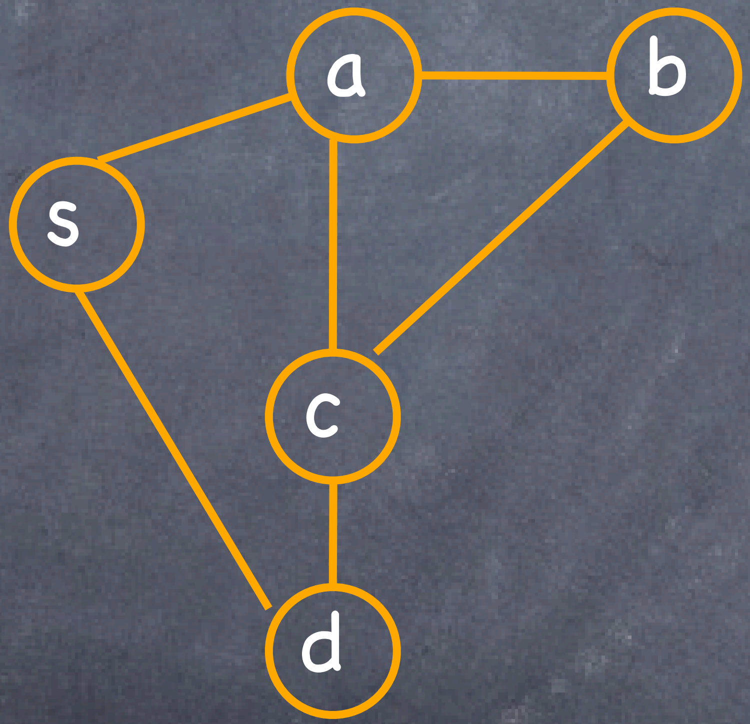


Visit the nodes in the order:

s

a, d

BFS Example



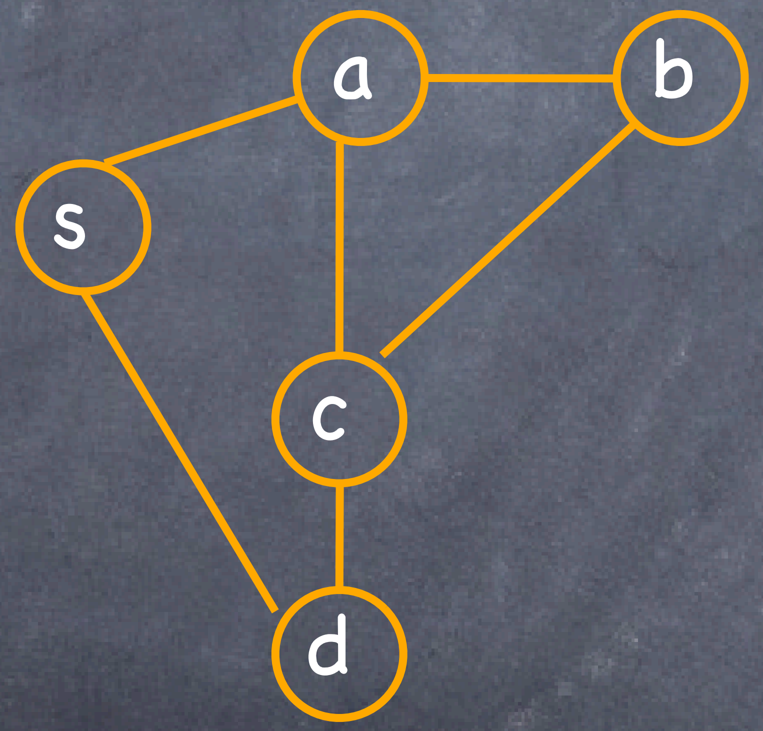
Visit the nodes in the order:

s

a, d

b, c

BFS Example



Visit the nodes in the order:

s

a, d

b, c

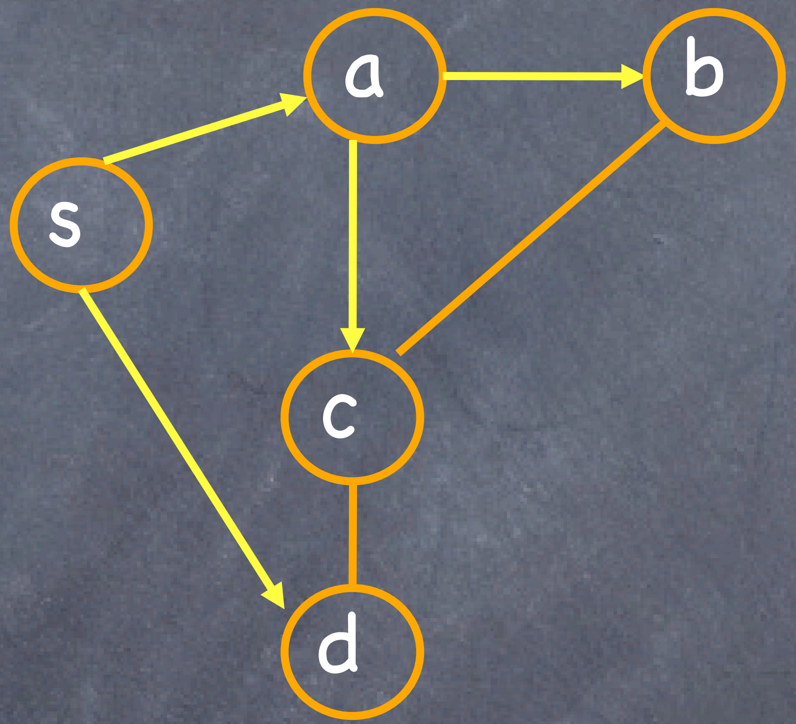
BFS Tree

We can make a spanning tree rooted at the source node s by remembering the parent of each node.

Breadth First Search (BFS)

```
Input:  A graph G = (V,E) and source node s in V
mark all nodes v in V as unvisited; set parent[v] := nil for all v in V
mark source node s as visited; parent[s] := s;
enq(Q,s)      // first-in first-out queue Q
while (Q is not empty) {
   u := deq(Q);
   for each unvisited neighbor v of u {
       mark v as visited; enq(Q,v); parent[v] := u
   }
}
```

BFS Tree Example



BFS Trees

The BFS tree is in general not unique for a given graph. It depends on the order in which neighboring nodes are processed.

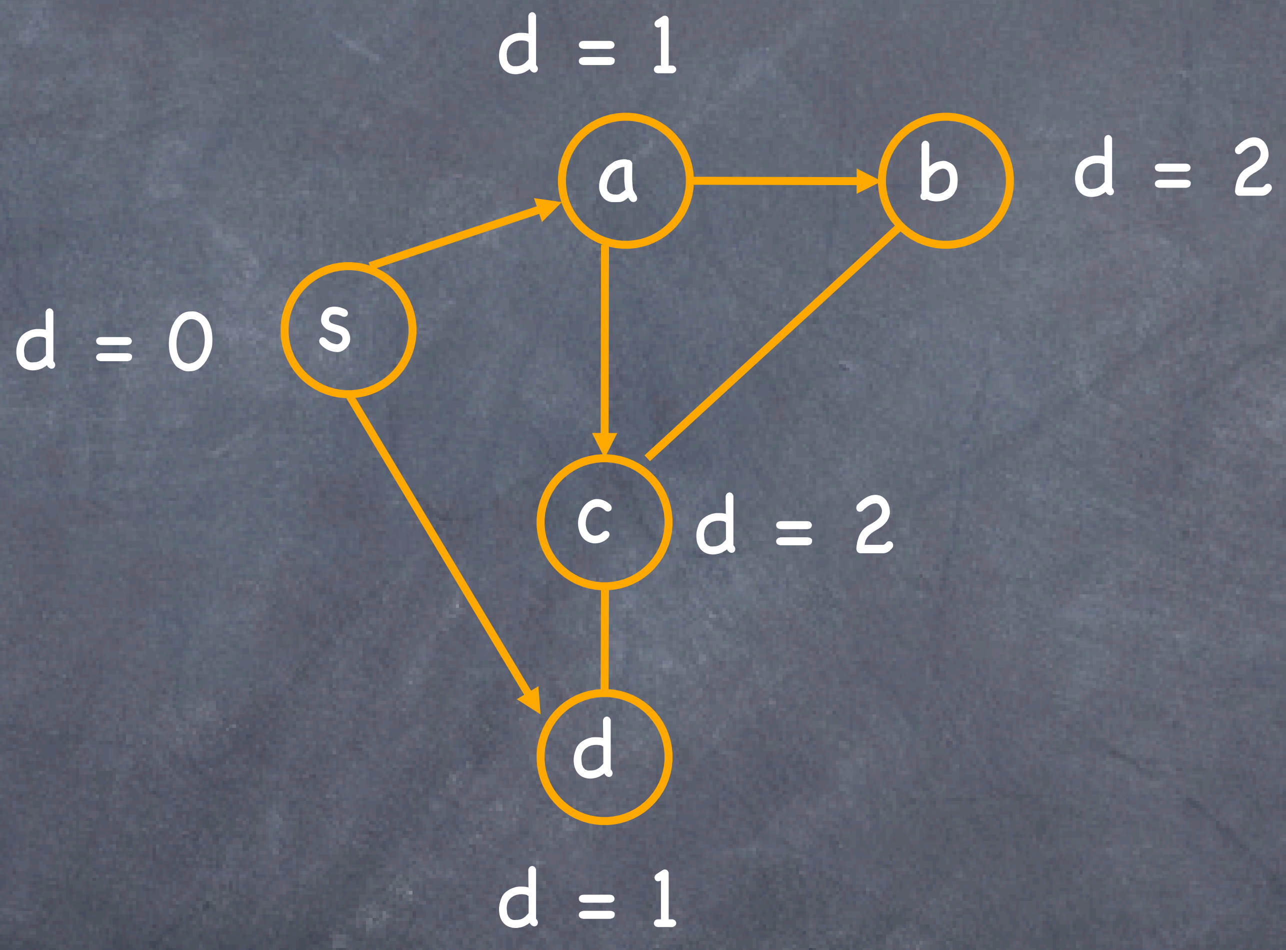
BFS Numbering

During the breadth-first search, assign to each node v its distance d[v] from the source.

Breadth First Search (BFS)

```
Input:  A graph G = (V,E) and source node s in V

mark all nodes v in V as unvisited; set parent[v] := nil; d[v] = ∞ for all v in V

mark source node s as visited; parent[s] := s; d[v] = 0

enq(Q,s)      // first-in first-out queue Q

while (Q is not empty) {
   u := deq(Q);
   for each unvisited neighbor v of u {
       mark v as visited; enq(Q,v); parent[v] := u; d[v] = d[u]+1
   }
}
```

BFS Numbering Example



Shortest Path Tree

Theorem: The BFS algorithm

- visits all and only nodes reachable from s
- for all nodes v sets d[v] to the shortest path distance from s to v
- sets parent variables to form a shortest path tree

Proof Ideas

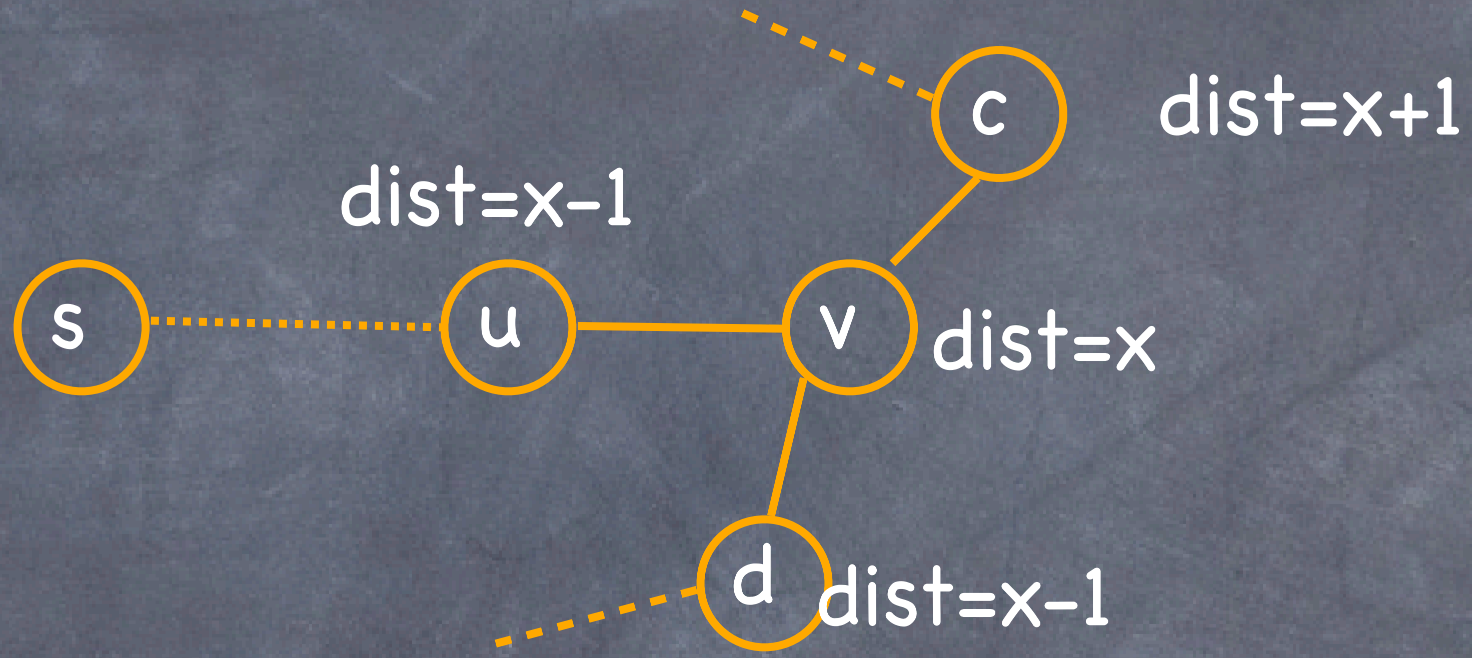
We use induction on the distance from the source node s to show that a node v at distance x from s has has correct d[v].

Basis: Distance 0. d[s] is set to 0.

Induction: Assume that all nodes u at distance x-1 from s satisfy d[u]=x-1. Our goal is to show that every node v at distance x satisfies d[v]=x as well.

Since v is at distance x, it has at least one neighbor at distance x-1. Let u be the first of these neighbors that is enqueued.

Proof Ideas



A key property of shortest path distances: If v has distance x,

- it must have a neighbor with distance x-1,
- no neighbor has distance less than x-1, and
- no neighbor has distance more than x+1

Proof Ideas

Claim: When the node u is dequeued, then v is still unvisited.

Indeed, this follows from behavior of the queue and the fact that d never underestimates the distance.

By induction, d[u] = x-1.

When v is enqueued, d[v] is set to d[u] + 1 = x.

BFS Running Time

Initialization of each node takes O(V) time

Every node is enqueued once and dequeued once, taking O(V) time

When a node is dequeued, all its neighbors are checked to see if they are unvisited, taking time proportional to number of neighbors of the node, and summing to O(E) over all iterations

Total time is O(V+E)

Credits

In the preparation of these slides, I got inspired by slides by Robert Sedgewick. The slides on BFS are based on slides by Jennifer Welch.