Connectivity

- **s-t connectivity problem.** Given two node s and t, is there a path between s and t?

- **s-t shortest path problem.** Given two node s and t, what is the length of the shortest path between s and t?

**Applications.**
- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Fewest number of hops in a communication network.
Breadth First Search

- BFS intuition. Explore outward from s in all possible directions, adding nodes one "layer" at a time.

- BFS algorithm.
  - $L_0 = \{ s \}$.
  - $L_1$ = all neighbors of $L_0$.
  - $L_2$ = all nodes that do not belong to $L_0$ or $L_1$, and that have an edge to a node in $L_1$.
  - $L_{i+1}$ = all nodes that do not belong to an earlier layer, and that have an edge to a node in $L_i$.

For each \(i\), \(L_i\) consists of all nodes at distance exactly \(i\) from \(s\). There is a path from \(s\) to \(t\) iff \(t\) appears in some layer.
BFS in action

Graph

Result of BFS(s)

BFS in action

Q: w r
BFS in action

Q: \textbf{r tx}

BFS in action

Q: \textbf{txv}
BFS in action

Q: $x \, v \, u$

BFS in action

Q: $v \, u \, y$
BFS in action

Q: \textbf{u \ y}

\begin{itemize}
  \item \textbf{r} \rightarrow \textbf{u}
  \item \textbf{s} \rightarrow \textbf{w}
  \item \textbf{t} \rightarrow \textbf{x}
  \item \textbf{u} \rightarrow \textbf{v}
\end{itemize}

BFS in action

Q: \textbf{y}

\begin{itemize}
  \item \textbf{r} \rightarrow \textbf{s}
  \item \textbf{t} \rightarrow \textbf{w}
  \item \textbf{u} \rightarrow \textbf{x}
\end{itemize}
BFS in action

Q:

Let $T$ be a BFS tree of $G = (V, E)$, and let $(x, y)$ be an edge of $G$. Then the level of $x$ and $y$ differ by at most 1.
BFS algorithm using lists

• In this algorithm it seems that we are using the list to implement which data structure?

• What’s the worst case running time analysis of this algorithm?

---

**BFS**

```
BFS(s):
    Set Discovered[s] = true and Discovered[v] = false for all v
    Initialize L[0] to consist of the single element s
    Set the layer counter i = 0
    Set the current BFS tree T = ∅
    While L[i] is not empty
        Initialize an empty list L[i+1]
        For each node u ∈ L[i]
            Consider each edge (u, v) incident to u
            If Discovered[v] = false then
                Set Discovered[v] = true
                Add edge (u, v) to the tree T
                Add v to the list L[i+1]
            Endif
        Endfor
    Endwhile
```

---

Depth First Search (Traversal)
Depth First Search

- DFS intuition. Explore the graph as if you are wandering in a maze, going deeper and deeper with each exploration step.

In the DFS tree generated, for any edge \((x,y)\) in the graph but not in the tree, \(x\) or \(y\) is an ancestor of the other.

Depth First Traversal

Graph

Result of DFS(s)
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
Depth-First Search in Action

Depth-First Search in Action
DFS algorithm using Stacks

- The complexity of this algorithm is $O(m+n)$ as well.

DFS(s):
Initialize $S$ to be a stack with one element $s$
While $S$ is not empty
    Take a node $u$ from $S$
    If Explored[$u$] = false then
        Set Explored[$u$] = true
        For each edge $(u,v)$ incident to $u$
            Add $v$ to the stack $S$
    Endfor
Endif
Endwhile