Graphs are everywhere

- Transportation networks: hubs, links, routes
- Communication networks: routing, how many hops, latency/throughput?
- Information networks: WWW, what are important/authoritative pages?
- Social networks: study interaction dynamics, find influencers?

How do we build algorithms to answer these questions?
**Political blogosphere graph**

Node = political blog; edge = link.

More applications

- Network science
  - random graphs: various evolution models
  - scale-free, small world
- Analyzing graph evolution in time
  - fake news
  - botnets
- Analyzing programs
  - control flow graph, function call graph
  - state space search (also in games): compute reachable states (configurations)
    - is an error state reachable?

Graphs

A graph is a mathematical representation of a network

- Set of nodes (vertices) \( V \)
- Set of pairs of nodes (edges) \( E \) (a relation)

Graph \( G = (V, E) \)

Definitions: edge, path

- **Edge** \( e = \{u, v\} \) (for an undirected graph)
- but usually written \( e = (u, v) \)

\( u \) and \( v \) are neighbors, endpoints of \( e \)

A **path** is a sequence \( P = v_1, v_2, \ldots, v_{k-1}, v_k \) such that each consecutive pair \( v_i, v_{i+1} \) is joined by an edge in \( G \)

Called: path “from \( v_1 \) to \( v_k \)”. Or: a \( v_1 \rightleftharpoons v_k \) path

Clicker Question 1

Q: Which is not a path?

1. UCSB - SRI - UTAH
2. LINC - MIT - LINC - CASE
3. UCSB - SRI - STAN - UCLA - UCSB
4. None of the above

Simple path, distance, cycle

- **Simple path**: path where all vertices are distinct
  - Exercise. Prove: If there is a path from \( u \) to \( v \) then there is a simple path from \( u \) to \( v \).
- **Distance** from \( u \) to \( v \):
  - minimum number of edges in a \( u \rightleftharpoons v \) path
- **Cycle**: path \( v_1, \ldots, v_{k-1}, v_k \) where \( v_1 = v_k \) (\( k > 1 \))
  - **Simple cycle**: no repeated nodes (except first = last)
**Trees**

Tree = a connected graph with no cycles

- Q: Is this equivalent to trees you saw in Data Structures?
- A: More or less.
- Rooted tree: tree with parent-child relationship
  - Pick root \( r \) and "orient" all edges away from root
  - Parent of \( v \) = predecessor on path from \( r \) to \( v \)

**Directed Graphs**

- Directed graph \( G = (V, E) \)
  - Directed edge \( e = (u, v) \) is now an ordered pair
  - \( e \) leaves \( u \) (source) and enters \( v \) (sink)
- Directed path, cycle: same as before, but with directed edges
- Strongly connected: directed graph with directed path between every pair of vertices
- Note: graphs undirected if not otherwise specified

**Graph Traversal**

Thought experiment. World social graph.
- Is it connected?
- If not, how big is the largest connected component?
- Is there a path between you and <some famous person>?

"Six degrees of separation" (everyone connected in at most 6 links?)
Erdős number: coauthorship of scientific papers
How can you tell algorithmically?
Answer: graph traversal! (BFS/DFS)
### Breadth-First Search: Layers

Explore outward from starting node \( s \).

Define layer \( L_i \) = all nodes at distance exactly \( i \) from \( s \).

**Layers**

- \( L_0 = \{ s \} \)
- \( L_1 \) = nodes with edge to \( L_0 \)
- \( L_2 \) = nodes with an edge to \( L_1 \) that don’t belong to \( L_0 \) or \( L_1 \)
- \( \ldots \)
- \( L_{i+1} \) = nodes with an edge to \( L_i \) that don’t belong to any earlier layer.

**Observation:**
There is a path from \( s \) to \( t \) if and only if \( t \) appears in some layer.

### BFS Tree

**Exercise:** draw the BFS layers for a BFS starting from MIT

![BFS Tree Diagram](image)

We can use BFS to make a tree.

### BFS and non-tree edges

**Claim:** let \( T \) be the tree discovered by BFS on graph \( G = (V, E) \),
and let \((x, y)\) be any edge of \( G \).
Then the layers of \( x \) and \( y \) in \( T \) differ by at most 1.

**Proof**

- Let \((x, y)\) be an edge
- Suppose \( x \in L_i, y \in L_j \), and \( j > i + 1 \)
- When BFS visits \( x \), either \( y \) is already discovered or not.
  - If \( y \) is already discovered, then \( j \leq i + 1 \). Contradiction.
  - Otherwise since \((x, y) \in E\), \( y \) is added to \( L_{i+1} \). Contradiction.

### A More General Exploration Strategy

To explore the connected component containing \( s \):

![Connected Component](image)

Add any node \( v \) for which

- \((u, v)\) is an edge
- \( u \) is explored, but \( v \) is not
**Depth-First Search**

Depth-first search (DFS): keep exploring from the most recently added node until you have to backtrack.

**Example.**

**Recursive DFS**

DFS(u)
Mark u as "explored"
for each edge (u, v) incident to u do
if v is not marked "explored" then
    Recursively invoke DFS(v)
end if
end for

**Exercise:** do an example

---

**DFS Tree**

*Can also extract tree T from DFS.*

- \((u, v) \in T\) if \(v\) explored from \(u\)—i.e., DFS(u) calls DFS(v)

**Claim:** let T be a depth-first search tree for graph \(G = (V, E)\), and let \((x, y)\) be an edge that is in \(G\) but not \(T\) (a "non-tree edge"). Then either \(x\) is an ancestor of \(y\) or \(y\) is an ancestor of \(x\) in \(T\).

**Proof?**

---

**Exploring all Connected Components**

How to explore entire graph even if it is disconnected?

```
while there is some unexplored node \(s\) do
    BFS(s) \(\triangleright\) Run BFS starting from \(s\).
    Extract connected component containing \(s\)
end while
```

Usually OK to assume graph is connected.
State if you are doing so and why it does not trivialize the problem.

**Running time?** What’s the running time of BFS?

---

**DFS and Non-tree edges**

**Claim:** let T be a depth-first search tree for graph \(G = (V, E)\), and let \((x, y)\) be an edge that is in \(G\) but not \(T\) (a "non-tree edge"). Then either \(x\) is an ancestor of \(y\) or \(y\) is an ancestor of \(x\) in \(T\).

**Proof**

- Suppose not and suppose that \(x\) is reached first by DFS.
- Before leaving \(x\), we must examine \((x, y)\).
- Since \((x, y) \not\in T\), \(y\) must have been explored by this time.
- But \(y\) was not explored when we arrived at \(x\) by assumption.
- Thus \(y\) was explored during the execution of DFS(x).
- Implies \(x\) is ancestor of \(y\).

**Implementation**

- How do we implement graph traversal?
  What is the running time?

- Preliminaries
  - Let \(m = |E|\) be the number of edges
  - Let \(n = |V|\) be the number of nodes
  - Data structure to represent graph? …
Graph representation: adjacency matrix

$n$-by-$n$ matrix with $A_{uv} = 1$ if $(u, v)$ is an edge

Space proportional to $n^2$

Clicker Question 3

An adjacency matrix representation for graph $(V, E)$ with $|V| = n$
takes time

A) $\Theta(n)$ to check if $(u, v)$ is an edge, $\Theta(|E|)$ to traverse all edges

B) $\Theta(n)$ to check if $(u, v)$ is an edge, $\Theta(n^2)$ to traverse all edges

C) $\Theta(1)$ to check if $(u, v)$ is an edge, $\Theta(|E|)$ to traverse all edges

D) $\Theta(1)$ to check if $(u, v)$ is an edge, $\Theta(|E|)$ to traverse all edges

Graph representation: adjacency lists

Adjacency lists. Each node keeps list of neighbors

- Each edge stored twice
- Space? $\Theta(m + n)$
- Checking if $(u, v)$ is an edge?
  $O(\text{degree}(u))$ time (degree = number of neighbors)

Traversal Implementations

Generic approach: maintain set of explored nodes and discovered nodes

- Exploring = have seen this node and explored its outgoing edges

- Discovered = the “frontier”. Have seen the node, but not explored its outgoing edges.

Generic Graph Traversal

Let $A$ = data structure of discovered nodes

Traverse($s$)

Put $s$ in $A$

while $A$ is not empty do

  Take a node $v$ from $A$

  if $v$ is not marked “explored” then

    Mark $v$ as “explored”

    for each edge $(v, w)$ incident to $v$ do

      if $w$ is discovered then

        Put $w$ in $A$

      end if

    end for

  end if

end while

Note: one part of this algorithm seems wasteful. Why?
Can put multiple copies of a single node in $A$.

BFS: $A$ is a queue (FIFO)  DFS: $A$ is a stack (LIFO)