Graphs and Traversals
Reminders/ Check in

- **Assignment 01** due tonight at 10 pm (Gradescope Assignment 1)
- Assignment 02 will be released later today
- If you haven't done so already, check out Problem Set Advice
- Take advantage of office hours today:
  - Mine: 1.30-3 pm, TAs: 6-10 pm
- Questions?

- Announcements?
  - Winter Carnival - No class Friday
Today’s Outline

• Formal definitions of graph terms
• Review common approaches for graph representation
• Review breadth-first search
• Review depth-first search
• Search Proofs (runtime, correctness)
An undirected graph $G = (V, E)$

- $V$ is the set of nodes, $E$ is the set of edges
- Graph size parameters: $n = |V|$, $m = |E|$
- Sometimes we consider weighted graphs, where each edge $e$ has a weight $w(e)$

$$V = \{1,2,3,4,5,6,7,8\}$$

$$E = \{(1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (3,7), (3,8), (4,5), (5,6), (7,8)\}$$

$n = 8$, $m = 11$
Option 1a: **Adjacency matrix.**

- $n$-by-$n$ matrix where $A[u][v] = 1$ if $(u, v) \in E$

$n = |V|$, $m = |E|$
Option 1a: **Adjacency matrix.**

- $n$-by-$n$ matrix where $A[u][v] = 1$ if $(u, v) \in E$
- Space $O(n^2)$?
- Checking if $(u, v) \in E$ takes $O(1)$ time?

$$
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
7 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
$$

$n = |V|$, $m = |E|$
**Option 1b: Adjacency list.**

- Array of lists, where each list stores the neighbors of a given node.

\[ n = |V|, \quad m = |E| \]
**Representing Graphs (Review)**

**Option 1b: Adjacency list.**

- Array of lists, where each list stores the neighbors of a given node
- Space $O(n + m)$?
- Checking if $(u, v) \in E$ takes $O(\text{degree}(u))$ time?

$n = |V|, \ m = |E|$
Graph Terminology (Review)

- A walk in an undirected graph $G = (V, E)$ is a sequence of vertices $u_1, u_2, \ldots, u_k$ such that every consecutive pair $(u_{i-1}, u_i) \in E$.

- A walk is path if all vertices are distinct (no repeats!).

- The length of a path is the number of edges on the path.

- An undirected graph is connected if for every pair of nodes $u$ and $v$, there is a path between $u$ and $v$ (e.g., every node is reachable from all other nodes).
  - A connected component is the set of all vertices/edges reachable from some vertex $v$.
  - A connected graph has 1 connected component.

- A cycle is a walk $u_1, u_2, \ldots, u_k$ where $u_1 = u_k$ and where no other vertices repeat.
Trees (Review)

An undirected graph is a **tree** if it is **connected** and **acyclic** (i.e., it does not contain a cycle).

**Lemma.** Let $G$ be an undirected graph with $n$ nodes. Then any two of these conditions imply the third

- $G$ is connected
- $G$ does not contain a cycle
- $G$ has $n - 1$ edges
Graph Traversals

A few common questions we ask about a graph \( G = (V, E) \):

- **Connectivity.** How do we verify if a graph is connected?
- **Reachability.** Given \( s, t \in V \), is there a path between them?

Answers can be determined by “traversing the graph”

- Two classic graph traversal algorithms:
  - Breadth-first search (BFS)
  - Depth-first search (DFS)

- BFS & DFS are remarkably similar algorithms that differ in the data structure used
Breadth-first Search

Explore outwards in all possible directions from starting point, peeling “one layer after another”

- BFS algorithm: Initialize $L_0 = \{v\}$
  - $L_1 = \text{all neighbors of } L_0$
  - $L_2 = \text{all nodes that do not belong to } L_0 \text{ or } L_1 \text{ that are adjacent to a node in } L_1$
  - ...
  - $L_{i+1} = \text{all nodes that do not belong an earlier layer that are adjacent to a node in } L_i$
BFS Implementation

We need **data structures** to represent:

- Nodes that we have not encountered yet
- Nodes that we have encountered but not yet “explored”
- Nodes that have been “fully explored” (encountered all its neighbors as well)
BFS Implementation

Suppose we are currently exploring node $u$

- Its neighbors will be marked “encountered”, but when will they be explored compared to other encountered but unexplored nodes?

- **BFS Idea**: Explore all nodes at level $i$ (same distance from initial node) before moving on to level $i + 1$
  - **Rule**: first encountered node should be first node to be explored

- Which data structure should we use?
  - Queue! First-in-first-out
BFS Implementation: Queue

BFS (G, s):
Set status of all nodes to unmarked
Place s into the queue Q
While Q is not empty
    Extract v from Q
    If v is unmarked
        Mark v
        For each edge (v, w):
            Put w into the queue Q

Observations:

• Nodes that we have not encountered have never been added to Q

• When a node \( u \) is marked (after extraction from Q), all \( u \)'s neighbors are then enqueued, so the next time we see \( u \) we can ignore it —it's already been explored!

• We may enqueue some nodes multiple times, but we only explore them once (if a marked node is extracted, it is skipped)
BFS Example
Tracing the Traversal: BFS Tree

- We can remember parent nodes (the node at level $i$ that lead us to a given node at level $i + 1$)
- Keeping track of these relationships produces a tree rooted at $s$

**BFS-Tree(G, s):**
- Put $(\emptyset, s)$ in the queue $Q$
- While $Q$ is not empty
  - Extract $(p, v)$ from $Q$
  - If $v$ is unmarked
    - Mark $v$
    - $parent(v) = p$
  - For each edge $(v, w)$:
    - Put $(v, w)$ into the queue $Q$ (*
BFS Analysis

- Inserting and extracting an edge from a queue: $O(1)$ time
- For each marked node $v$, we run the for loop for its edges: $O(n)$ times
- Overall running time? $O(n^2)$
  - Can we tighten our analysis?
- Yes! We can improve our analysis to $O(n + m)$
  - Node $u$ has $\text{degree}(u)$ incident edges $(u, v)$
    - Total time processing edges: $\sum_{u \in V} \text{degree}(u) = 2m$
  
  each edge $(u, v)$ is counted exactly twice
  in sum: once in $\text{degree}(u)$ and once in $\text{degree}(v)$
Depth-First Search
Stack Instead of Queue

If we change how we store the visited vertices (the data structure we use), it changes how we traverse the graph.

**BFS** \((G, s)\):

- Set status of all nodes to unmarked
- Place \(s\) into the queue \(Q\)
- While \(Q\) is not empty
  - Extract \(v\) from \(Q\)
  - For each edge \((v, w)\):
    - If \(w\) is unmarked
      - Put \(w\) into the queue \(Q\)
Stack Instead of Queue

If we change how we store the visited vertices (the data structure we use), it changes how we traverse the graph.

**DFS (G, s):**
- Set status of all nodes to unmarked
- Place s into the stack S
- While S is not empty
  - Extract v from S
  - For each edge (v, w):
    - If w is unmarked
      - Put w into the stack S
Depth-First Search: Recursive

DFS is perhaps the more natural traversal algorithm to write.

- Can be written *iteratively* or *recursively*
- Both DFS versions are the same; can actually see the “recursion stack” in the iterative version

**Recursive-DFS(u):**
- Set status of u to marked
- For each edge (u, v):
  - If v's status is unmarked:
    - DFS(v)
- # done exploring neighbors of u
Example Graph
DFS Running Time

We can apply the same analysis as we did for BFS.

- Inserts and extracts to a stack: $O(1)$ time
- Setting status of each node to unmarked: $O(n)$
- Each node is set marked at most once; equivalently DFS($u$) is called at most once for each node
- For every node $v$, explore degree($v$) edges
  \[ \sum_v \text{degree}(v) = 2m \]
- Overall, running time $O(n + m)$
Depth-First Search Tree

DFS returns a spanning tree, similar to BFS

DFS-Tree(G, s):
    Put (∅, s) in the stack S
    While S is not empty
        Extract (p, v) from S
        If v is unmarked
            Mark v
            parent(v) = p
            For each edge (v, w):
                Put (v, w) into the stack S

The spanning tree formed by parent edges in a DFS are usually long and skinny
Proving Correctness
DFS Correctness

- DFS finds precisely the set of nodes reachable from start node $s$
- That is, DFS($s$) marks node $x$ iff node $x$ is reachable from $s$

Proof. ($\Rightarrow$)

- Since $x$ is marked, ($x$, parent($x$)) is an edge in the graph
- Claim. $x \rightarrow$ parent($x$) $\rightarrow$ parent(parent($x$)) $\rightarrow$ $\cdots$ leads to $s$
- Induction on the sequence of vertices marked by DFS
- Let $u_1, u_2, \ldots, u_k, \ldots, u_n$ denote the order in which vertices are marked, suppose claim holds all vertices with index less than $k$
- Consider $u_k$: parent($u_k$) must be discovered before $u_k$, and thus the claim holds for it, since ($u_k$, parent($u_k$)) is an edge, we have a path from $u_k$ to $s$
DFS Correctness

- DFS finds precisely the set of nodes reachable from start node $s$
- That is, $\text{DFS}(s)$ marks node $x$ iff node $x$ is reachable from $s$

**Proof.** ($\Leftarrow$)

- Suppose node $x$ is reachable from $s$ via path $P$, but $x$ is not marked by DFS
- Since $s$ is marked by DFS and $x$ is not, there must be a first node $v$ on $P$ that is not marked by DFS
- Thus, there is an edge $(u, v) \in P$ such that $u$ is marked and $v$ is not marked
- But this cannot happen, since when $u$ is marked, all its neighbors are also marked $\Rightarrow \Leftarrow$
BFS Correctness

- Breadth first search finds precisely the set of nodes reachable from $s$
- That is, $\text{BFS}(s)$ marks node $x$ iff node $x$ is reachable from $s$

**Proof.** ($\implies$)

- Since $x$ is marked, $(x, \text{parent}(x))$ is an edge in the graph
- **Claim.** $x \rightarrow \text{parent}(x) \rightarrow \text{parent}(\text{parent}(x)) \rightarrow \cdots$ leads to $s$
- Induction on the sequence of vertices marked by BFS
- Let $u_1, u_2, \ldots, u_k, \ldots, u_n$ denote the order in which vertices are marked, suppose claim holds all vertices with index less than $k$
- Consider $u_k$: $\text{parent}(u_k)$ must be discovered before $u_k$, and thus the claim holds for it, since $(u_k, \text{parent}(u_k))$ is an edge, we have a path from $u_k$ to $s$
BFS Correctness

- Breadth first search finds precisely the set of nodes reachable from $s$
- That is, BFS($s$) marks node $x$ iff node $x$ is reachable from $s$

**Proof.** ($\Leftarrow$)

- Suppose node $x$ is reachable from $s$ via path $P$, but $x$ is not marked by BFS
- Since $s$ is marked by BFS and $x$ is not, there must be a first node $v \neq s$ on $P$ that is not marked by BFS
- Thus, there is an edge $(u, v) \in P$ such that $u$ is marked and $v$ is not marked
- But this cannot happen, since when $u$ is marked, all its neighbors are also marked $\Rightarrow \Leftarrow$