Review and Outlook

- Graph traversal by BFS/DFS
  - Different versions of general exploration strategy
  - $O(m + n)$ time
  - Produce trees with useful properties (for other problems)
  - Basic algorithmic primitive — used in many other algorithms (path from $s$ to $t$, connected components)
- Bipartite testing
- Directed graphs
  - Traversal
  - Topological sorting
  - Strong connectivity

Bipartite Graphs

Definition: A graph $G = (V, E)$ is bipartite if $V$ can be partitioned into sets $X, Y$ such that every edge has one end in $X$ and one in $Y$.

Can color nodes red/blue so that no edges connect nodes of the same color.

Examples
- Bipartite: student-college graph in stable matching
- Bipartite: client-server connections
- Not bipartite: "odd cycle" (cycle with an odd number of nodes)
- Not bipartite: any graph containing an odd cycle

Claim (easy): If $G$ contains an odd cycle, it is not bipartite.

Bipartite Testing

Question: Given $G = (V, E)$, is $G$ bipartite?

Algorithm: Idea: run BFS from any node $s$

- $L_0 = \text{red}$
- $L_1 = \text{blue}$
- $L_2 = \text{red}$
- ... (even layers red, odd layers blue)

What could go wrong? Edge between two nodes at the same layer.

Algorithm

- Run BFS from any node $s$
  - if there is an edge between two nodes in the same layer then Output "not bipartite"
  - else
    - $X =$ even layers
    - $Y =$ odd layers
  - end if

Correctness? Recall: all edges between same or adjacent layers.

1. No edges between nodes in the same layer $\Rightarrow$ correct labeling, $G$ bipartite.
2. Edge between two nodes in the same layer $\Rightarrow G$ has an odd cycle, not bipartite.

Proof

- Let $T$ be BFS tree of $G$ and suppose $(x, y)$ is an edge between two nodes in the layer $j$
  - Let $z \in L_i$ be the least common ancestor of $x$ and $y$ (Useful in proofs: take least/greatest item with some property)
    - Let $P_{zx}$ = path from $z$ to $x$ in $T$
    - Let $P_{zy}$ = path from $z$ to $y$ in $T$
    - The path that follows $P_{zx}$ then edge $(x, y)$ then $P_{zy}$ is a cycle of length $2(j - i) + 1$, which is odd
  - Therefore $G$ is not bipartite.
Which of the following is true?

A. If $G$ is bipartite, then $G$ does not have an odd cycle
B. If $G$ does not have an odd cycle, then $G$ is bipartite
C. Both A and B
D. Neither A nor B

**Directed Graph Definitions**

Most definitions extend naturally to directed graphs by mapping the word “edge” to “directed edge”

- **Directed path**: sequence $P = v_1, v_2, \ldots, v_k$ such that each consecutive pair $v_i, v_{i+1}$ is joined by a directed edge in $G$. A $v_1 \rightarrow v_k$ path.

- **Directed cycle**: directed path with $v_1 = v_k$

- When referring to a directed graph, the words “path” and “cycle” mean “directed path” and “directed cycle”

- **Connected? Connected component?** More subtle, because now there can be a path from $s$ to $t$ but not vice versa. More later.

**Directed Graph Traversal**

BFS/DFS naturally extend to directed graphs.

BFS($s$):

- mark $s$ as "discovered"
- $L[0] \leftarrow \{s\}$, $i \leftarrow 0$
- while $L[i]$ is not empty do
  - $L[i+1] \leftarrow$ empty list
  - for all nodes $v$ in $L[i]$ do
    - for all edges $(v, w)$ leaving $v$ do
      - if $w$ is not marked "discovered" then
        - mark $w$ as "discovered"
        - put $w$ in $L[i+1]$
      - end if
    - end for
  - end for
  - $i \leftarrow i + 1$
- end while

Find all nodes $v$ with $v \rightarrow t$ path? BFS following edges in reverse direction.

Useful to keep adjacency lists for both outgoing and incoming edges.
Clicker

Suppose $G$ is a directed path on $n$ vertices and BFS is called repeatedly starting from any unexplored vertex until all nodes are explored. What is the maximum number of times BFS may be called?

A. $n - 1$
B. $n$
C. 1
D. $m$

Directed Acyclic Graphs

**Definition:** A directed acyclic graph (DAG) is a directed graph with no cycles.

Models dependencies, e.g. course prerequisites:

Math132
CS187
CS220
CS240
CS250
CS311
CS383

Topological Sorting

**Definition:** A topological ordering of a directed graph is an ordering of the nodes such that all edges go “forward” in the ordering

- Label nodes $v_1, v_2, \ldots, v_n$ such that
- For all edges $(v_i, v_j)$ we have $i < j$
- A way to order the classes so all prerequisites are satisfied

Q: Is a topological ordering possible for any directed graph?

Exercise

1. Find a topological ordering.
2. Devise an algorithm to find a topological ordering.

Topological Ordering

**Claim** If $G$ has a topological ordering, then $G$ is a DAG.

Problem Given DAG $G$, compute a topological ordering for $G$.

topo-sort($G$)

while there are nodes remaining do
    Find a node $v$ with no incoming edges
    Place $v$ next in the order
    Delete $v$ and all of its outgoing edges from $G$
end while

Running time? $O(n^2 + m)$ easy. $O(m + n)$ more clever
Topological Sorting Analysis

- In a DAG, there is always a node \( v \) with no incoming edges. Try to prove. (contradiction, pigeonhole principle)
- Removing a node \( v \) from a DAG, produces a new DAG.
- Any node with no incoming edges can be first in topological ordering.

**Theorem**: \( G \) is a DAG if and only if \( G \) has a topological ordering.

**Clicker**

The maximum number of edges in a DAG with \( n \) nodes is

A. \( 2(n - 1) \)
B. \( 2n - 1 \)
C. \( n(n - 1)/2 \)
D. \( n(n - 1) \)

**Topological Sorting in \( O(m + n) \)**

topo-sort(\( G \))

\[ \text{while there are nodes remaining do} \]

- Find a node \( v \) with no incoming edges
- Place \( v \) next in the order
- Delete \( v \) and all of its outgoing edges from \( G \)

\[ \text{end while} \]

Optimization: don’t search every time for nodes w/o incoming edges

- Keep and update incoming edge count for each node (setup in \( O(m + n) \), each update constant-time)
- Keep set of nodes of nodes with incoming edges; add node when its count becomes zero
- Running time: \( O(m + n) \)

**Directed Graph Connectivity**

- Strongly connected graph. Directed path between any two nodes.
- Strongly connected component (SCC). Maximal subset of nodes with directed path between any two.
- SCCs can be found in time \( O(m + n) \). (Tarjan, 1972)

**Clicker**

Consider the graph \( G' \) whose nodes are SCCs and there is an edge from \( C \) to \( D \) if any node in \( C \) has an edge to \( D \). Which of the following is always true?

A. \( G' \) is strongly connected
B. \( G' \) has a cycle
C. \( G' \) has at least \( n/2 \) nodes
D. \( G' \) is a DAG

**BFS in Directed Graphs: Non-Tree Edges**

With respect to BFS tree, graph edges can go

- one level down (tree or non-tree edge)
- why not > 1? same reason, would add to next level
- same level (non-tree)
- any levels up (non-tree)
DFS in Directed Graphs: Non-Tree Edges

3 → 1 is a back edge (to ancestor)
2 → 5 is a forward edge (to descendant)
4 → 5 is a cross edge (node in another subtree)