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** 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 s.t. no edges between nodes of same color.

**Examples**
- Bipartite: student-college graph in stable matching
- Bipartite: client-server connections
- Not bipartite: “odd cycle” (cycle with odd \# of nodes)

**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 \(s\)

- \(L_0\) = red
- \(L_1\) = blue
- \(L_2\) = red
- ...
- Even layers red, odd layers blue

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

Algorithm

Run BFS from any node \(s\)
if there is an edge between two nodes in same layer then
  Output "not bipartite"
else
  Output "bipartite" with \(X\) = even layers and \(Y\) = odd layers

Correctness

Remember the fact about BFS: every edge connects nodes in the same layer or in adjacent layers (i.e., one even, one odd).

Proof structure:
1. If the algorithm outputs “bipartite”, then all edges connect nodes in an even layer \((X)\) and an odd layer \((Y)\), so \(G\) is bipartite. ✓
2. If the algorithm outputs “not bipartite”, then there is an edge between two nodes in the same layer. We will show this implies that \(G\) has an odd cycle, so \(G\) is not bipartite.
Proof

Claim: if there is an edge between two nodes in the same layer, then \( G \) has an odd cycle.

- 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 \( P_{yz} \) is a cycle of length \( 2(j - i) + 1 \), which is odd
- The claim is proved, which completes the proof of the algorithm

Clicker

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 Graphs

\[ G = (V, E) \]

- \((u, v) \in E\) is a directed edge
- \( u \) points to \( v \)
- \( e = (u, v) \) leaves \( u \), enters \( v \), is an outgoing edge from \( u \), incoming edge to \( v \)

Examples
- Facebook: undirected
- Twitter: directed
- Web: directed
- Road network: directed (discuss)

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 \( v_1 \to v_k \) 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

Reachability. Find all nodes reachable from some node \( s \). All nodes \( v \) with \( s \to v \) path.

\( s \to t \) shortest path. What is the length of the shortest directed path from \( s \) to \( t \)?

Algorithm?

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
  - \( L[i + 1] \leftarrow \text{empty list} \)
  - 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 \( L[i + 1] \)
      - \( i \leftarrow i + 1 \)
Directed Graph Traversal

Find all nodes $v$ with $v \to t$ 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:

<table>
<thead>
<tr>
<th>Math132</th>
<th>CS187</th>
<th>CS220</th>
<th>CS311</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS187</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS220</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS240</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS383</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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.
Topological Sorting

**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 $G$

Running time? Can show it’s $O(m + n)$

Topological Sorting Analysis

What to prove:
1. Algorithm can always find an ordering
2. The ordering is a topological ordering.

Sketch of analysis:
- In a DAG $G$, there is always a node $v$ with no incoming edges. **Try to prove.**
- Any such node $v$ can be first in the topological ordering.
- Removing $v$ from $G$ produces a new DAG $G'$.
- The node $v$ followed by a topological ordering for $G'$ is a topological ordering for $G$.

DAGs and Topological Orderings

**Theorem:** $G$ is a DAG if and only if $G$ has a topological ordering.

**Proof:**
1. If $G$ is a DAG, the algorithm finds a topological ordering.
2. If $G$ is not a DAG then $G$ does not have a topological ordering.

Directed Graph Connectivity

**Strongly connected graph:** graph with directed path between any pair of nodes.

**Strongly connected component (SCC):** maximal subset of nodes with directed path between any pair.

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