Lecture 15: Breadth-First Search
Outline

- Breadth First Search (BFS)
- Applications of BFS
BFS Algorithm

- Explores vertices and edges in systematic order, go for breadth
  - start at some vertex $v$, $\text{currVertex} = v$
  - explore all edges from $\text{currVertex} = v$
  - $v$ is now fully explored, who should become $\text{currVertex}$ next?
    - From $v$, we first visited $u$, then $w$, then $z$
    - $\text{currVertex}$ should be the earliest visited but not yet explored vertex
    - To keep track of visited, but not fully explored vertices, use queue $Q$ (FIFO)
    - During exploration, put not fully explored vertices at the queue end
    - After done exploring vertex, get next $\text{currVertex}$ from front of queue
    - Mark vertices VISITED to avoid cycles
BFS Algorithm

Q = \{v\}
currVertex = v ; Q = {}  
currVertex = v ; Q=\{u\}  
currVertex = v ; Q=\{u, w\}  
currVertex = v ; Q=\{u, w, z\}  
currVertex = u ; Q ={w, z }  
currVertex = u ; Q ={w, z , x }  
currVertex = w ; Q ={z , x }  
currVertex = w ; Q ={z , x , t }  
currVertex = w ; Q =\{z , x , t , y\}  
currVertex = z ; Q =\{x , t , y\}  
currVertex = x ; Q =\{t , y\}  
currVertex = t ; Q =\{y\}  
currVertex = y ; Q = \{\}\n
Done!
BFS Algorithm

- As in DFS, need 2 methods, in case the graph is not connected
  - BFS\((G,s)\) performs BFS in the connected component containing \(s\)
  - Marks edges as DISCOVERY and CROSS

**Algorithm \(BFS(G)\)**

**Input:** graph \(G\)

**Output:** labeling of edges and partition of vertices of \(G\)

```
for all \(e \in G\).edges()
    setLabel\((e, UNVISITED)\)
for all \(u \in G\).vertices()
    setLabel\((u, UNVISITED)\)
for all \(v \in G\).vertices()
    if getLabel\((v) = UNVISITED\)
        BFS\((G,v)\)
```

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

- \(Q \leftarrow \) new empty queue
- \(Q\).insertLast\((s)\)
- setLabel\((s, VISITED)\)

```
while \(\neg Q\).isEmpty()
    \(v \leftarrow Q\).getFirst()
    for all \(e \in G\).incidentEdges\((v)\)
        \(w \leftarrow \) opposite\((v,e)\)
        if getLabel\((w) = UNVISITED\)
            setLabel\((w, VISITED)\)
            setLabel\((e, DISCOVERY)\)
            \(Q\).insertLast\((w)\)
        else
            if getLabel\((e) = UNVISITED\)
                setLabel\((e, CROSS)\)
```

BFS Algorithm: Levels

- Identify levels in graph after BFS finishes
- Level 0 consists of the start vertex
- Level 1 consists of vertices connected by discovery edge with a vertex in Level 0
- Level 2 consists of vertices connected by a discovery edge with vertex in Level 1
- ... 

- Level \(i\) consists of vertices connected by discovery edge with a vertex in Level \(i - 1\)

To compute levels explicitly
- in the beginning of \(\text{BFS}(G,s)\) put statement \(\text{setLevel}(s,0)\)
- after \(\text{setLabel}(w, \text{VISITED})\) add \(\text{setLevel}(w,\text{getLevel}(v)+1)\)
Algorithm \textit{BFS}(G)

\textbf{Input:} graph G

\textbf{Output:} labeling of edges and partition of vertices of G

\begin{itemize}
  \item for all e ∈ G.edges()
    \begin{itemize}
      \item setLabel(e,UNVISITED)
    \end{itemize}
  \item for all u ∈ G.vertices()
    \begin{itemize}
      \item setLabel(u,UNVISITED)
    \end{itemize}
  \item for all v ∈ G.vertices()
    \begin{itemize}
      \item if \ getLabel(v) = UNVISITED
        \begin{itemize}
          \item BFS(G,v)
        \end{itemize}
    \end{itemize}
\end{itemize}

Algorithm \textit{BFS}(G,s)

\begin{itemize}
  \item Q ← new empty queue
  \item Q.insertLast(s)
  \item setLevel(s,0)
  \item setLabel(s,VISITED)
  \item while ¬Q.isEmpty()
    \begin{itemize}
      \item v ← Q.getFirst()
      \item for all e ∈ G.incidentEdges(v)
        \begin{itemize}
          \item w ← opposite(v,e)
          \item if getLabel(w) = UNVISITED
            \begin{itemize}
              \item setLabel(w,VISITED)
              \item setLevel(w,getCellLevel(v)+1)
              \item setLabel(e,DISCOVERY)
              \item Q.insertLast(w)
            \end{itemize}
          \end{itemize}
        \end{itemize}
      \end{itemize}
  \end{itemize}
\end{itemize}
Example

- **A** unexplored vertex
- **A** visited vertex
- **---** unexplored edge
- **→** discovery edge
- **→ →** cross edge
Example (cont.)
Example (cont.)
Properties

Notation

\( G_s \): connected component of \( s \)

Property 1

\( \text{BFS}(G, s) \) visits all the vertices and edges of \( G_s \)

Property 2

Edges labeled as discovery form a spanning tree \( T_s \) of \( G_s \)

Property 3

For any edge \((v,w)\), the difference in level of \( v \) and level of \( w \) is at most 1

Property 4

For each vertex \( v \) in \( L_i \)
- the path of \( T_s \) from \( s \) to \( v \) has \( i \) edges
- every path from \( s \) to \( v \) in \( G_s \) has at least \( i \) edges

Property 4 says that a path from \( s \) to \( v \) in \( T_s \) is the shortest path (in the number of edges) from \( s \) to \( v \) in \( G \)
Algorithm **BFS**\((G,s)\)

\[
\begin{align*}
Q & \leftarrow \text{new empty queue} \\
Q.\text{insertLast}(s) \\
\text{setLevel}(s,0) \\
\text{setLabel}(s,\text{VISITED}) \\
\text{while } ¬Q.\text{isEmpty}() \\
\quad v & \leftarrow Q.\text{getFirst}() \\
\quad \text{for all } e \in G.\text{incidentEdges}(v) \\
\quad \quad w & \leftarrow \text{opposite}(v,e) \\
\quad \quad \text{if getLabel}(w)=\text{UNVISITED} \\
\quad \quad \quad \text{setLabel}(w,\text{VISITED}) \\
\quad \quad \quad \text{setLabel}(e,\text{DISCOVERY}) \\
\quad \quad \quad w.\text{setParent}(v) \\
\quad \quad \quad Q.\text{insertLast}(w) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{if getLabel}(e) = \text{UNVISITED} \\
\quad \quad \quad \quad \text{setLabel}(e,\text{CROSS})
\end{align*}
\]
Analysis

- Assume adjacency list structure

- \(BFS(G)\) takes \(O(n+m)\) time, go over vertices and edges twice

- Let \(n_s\) be number of vertices and \(m_s\) number of edges in the connected component of \(s\)

- In \(BFS(G,s)\) while loop executes \(n_s\) times, once for each vertex in connected component of \(s\)
  - each vertex inserted once into \(Q\)
  - at each iteration of while loop, one vertex is removed from the queue

- Operations inside while loop take \(O(\text{deg}(v))\)
  - recall that \(\Sigma_v \text{deg}(v) = 2m\)

- Total time for \(BFS(G,s)\) is \(O(n_s + m_s)\)

- \(BFS(G,s)\) is called one time for each connected component
  - total time spent in \(BFS(G,s)\) is \(O(n + m)\)

- BFS runs in \(O(n + m)\) time

Algorithm \(BFS(G)\)

Input: graph \(G\)

Output: labeling of edges and partition of vertices of \(G\)

\[
\begin{align*}
&\text{for all } e \in G.\text{edges}() \\
&\quad \text{setLabel}(e, \text{UNVISITED}) \\
&\text{for all } u \in G.\text{vertices}() \\
&\quad \text{setLabel}(u, \text{UNVISITED}) \\
&\text{for all } v \in G.\text{vertices}() \\
&\quad \text{if getLabel}(v) = \text{UNVISITED} \quad \text{BFS}(G,v)
\end{align*}
\]

Algorithm \(BFS(G,s)\)

\[
\begin{align*}
&Q \leftarrow \text{new empty queue} \\
&Q.\text{insertLast}(s) \\
&\text{setLabel}(s, 0) \\
&\text{setLabel}(s, \text{VISITED}) \\
&\text{while } \neg Q.\text{isEmpty}() \\
&\quad v \leftarrow Q.\text{getFirst}() \\
&\quad \text{for all } e \in G.\text{incidentEdges}(v) \\
&\quad \quad w \leftarrow \text{opposite}(v,e) \\
&\quad \quad \text{if getLabel}(w) = \text{UNVISITED} \\
&\quad \quad \quad \text{setLabel}(w, \text{VISITED}) \\
&\quad \quad \quad \text{setLabel}(w, \text{getLevel}(v)+1) \\
&\quad \quad \quad \text{setLabel}(e, \text{DISCOVERY}) \\
&\quad \quad Q.\text{insertLast}(w) \\
&\quad \text{else} \\
&\quad \quad \text{if getLabel}(e) = \text{UNVISITED} \\
&\quad \quad \quad \text{setLabel}(e, \text{CROSS})
\end{align*}
\]
Applications

- Can specialize BFS to solve the following problems in $O(n + m)$ time
  - Compute the connected components of $G$
  - Compute a spanning forest of $G$
  - Find a simple cycle in $G$, or report that $G$ is a forest
  - **Given two vertices of $G$, find a path in $G$ between them with the minimum number of edges, or report that no such path exists**
    - Perform BFS starting at vertex $v$
    - For any vertex $u$, if the level of $u$ is $d$, then the shortest distance from $v$ to $u$ in the number of edges is $d$
## DFS vs. BFS

### Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>DFS</th>
<th>BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanning forest, connected components, paths, cycles</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Shortest paths (in number of edges)</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Biconnected components (did not study this)</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

DFS and BFS are two fundamental algorithms in graph theory. DFS explores as far as possible down the tree, while BFS explores breadth-first. The diagrams illustrate the exploration order of DFS (left) and BFS (right) starting from node A.

**DFS:**
- Begins at A and explores its children first.
- Moves to B, then to E, and back to A.
- Continues to C, D, F, and back to A.
- Ends when no more children are available.

**BFS:**
- Begins at A and explores its children next.
- Moves to B, then to F, and back to A.
- Continues to C, D, E, and back to A.
- Ends when all children are explored.

### Summary
- DFS is useful for finding connected components, cycles, and spanning forests.
- BFS is useful for finding shortest paths in number of edges.
- Biconnected components are not typically studied in the context of DFS and BFS.
DFS vs. BFS

Back edge \((v,w)\)
- \(w\) is an ancestor of \(v\) in the tree of discovery edges

Cross edge \((v,w)\)
- \(w\) is in the same level as \(v\) or in the next level in the tree of discovery edges (\(u\) is not an ancestor of \(v\) and \(v\) is not an ancestor of \(u\))

DFS

BFS
Breadth-First Search Summary

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
  - Visits all the vertices and edges of G
  - Determines whether G is connected
  - Computes the connected components of G
  - Computes a spanning forest of G
- BFS on a graph with $n$ vertices and $m$ edges takes $O(n + m)$ time with adjacency list data structure
- BFS can be further extended to solve other graph problems
  - Find and report path with minimum number of edges between two given vertices
  - Find a simple cycle, if there is one