csci 210: Data Structures

Graph Traversals
Graph traversal (BFS and DFS)

- G can be undirected or directed
- We think about coloring each vertex
  - WHITE before we start
  - GRAY after we visit a vertex but before we visited all its adjacent vertices
  - BLACK after we visit a vertex and all its adjacent vertices

- We store all GRAY vertices---these are the vertices we have seen but we are not done with

- Depending on the structure (queue or list), we get BFS or DFS

- We remember from which vertex a given vertex w is colored GRAY ---- this is the vertex that discovered w, or the parent of w
**BFS**

- G can be undirected or directed
- Initialize:
  - for each v in V
    - color(v) = WHITE
    - parent(v) = NULL

**Traverse(v)**
- color(v) = GRAY
- create an empty set S
- insert v in S
- while S not empty
  - delete node u from S
  - for all adjacent edges (u, w) of e in E do
    - if color(w) = WHITE
      - color(w) = GRAY
      - parent(w) = u
      - insert w in S
  - color(u) = BLACK
Breadth-first search (BFS)

- How it works:

- BFS(v)
  - start at v and visit first all vertices at distance =1
  - followed by all vertices at distance=2
  - followed by all vertices at distance=3
  - ...

- BFS corresponds to computing the shortest path (in terms of number of edges) from v to all other vertices
BFS

- G can be undirected or directed
- BFS-initialize:
  - for each v in V
    - color(v) = WHITE
    - d[v] = infinity
    - parent(v) = NULL

BFS(v)
- color(v) = GRAY
- d[v] = 0
- create an empty queue Q
- Q.enqueue(v)
- while Q not empty
  - Q.dequeue(u)
  - for all adjacent edges (u,w) of e in E do
    - if color(w) = WHITE
      » color(w) = GRAY
      » d[w] = d[u] + 1
      » parent(w) = u
      » Q.enqueue(w)
  - color(u) = BLACK
BFS

- We can classify edges as
  - discovery (tree) edges: edges used to discover new vertices
  - non-discovery (non-tree) edges: lead to already visited vertices
- The distance $d(u)$ corresponds to its “level”
- For each vertex $u$, $d(u)$ represents the shortest path from $v$ to $u$
  - justification: by contradiction. If $d[u]=k$, assume there exists a shorter path from $v$ to $u$.
- Assume $G$ is undirected (similar properties hold when $G$ is directed).
  - connected components are defined undirected graphs (note: on directed graphs: strong connectivity)
- As for DFS, the discovery edges form a tree, the BFS-tree
- BFS($v$) visits all vertices in the connected component of $v$
- If $(u,w)$ is a non-tree edges, then $d(u)$ and $d(w)$ differ by at most 1.
- If $G$ is given by its adjacency-list, BFS($v$) takes $O(|V|+|E|)$ time.
BFS

Putting it all together:

Proposition: Let $G=(V,E)$ be an undirected graph represented by its adjacency-list. A BFS traversal of $G$ can be performed in $O(|V|+|E|)$ time and can be used to solve the following problems:

- testing whether $G$ is connected
- computing the connected components (CC) of $G$
- computing a spanning tree of the CC of $v$
- computing a path between 2 vertices, if one exists
- computing a cycle, or reporting that there are no cycles in $G$
- computing the shortest paths from $v$ to all vertices in the CC of $v$
Depth-first search (DFS)

- G can be directed or undirected
- use Traverse(v) with S = stack
- or recursively

DFS(v)
  - mark v visited
  - for all adjacent edges (v,w) of v do
    - if w is not visited
      - parent(w) = v
      - (v,w) is a discovery (tree) edge
      - DFS(w)
    - else (v,w) is a non-discovery (non-tree) edge
\begin{itemize}
  \item Assume \(G\) is undirected (similar properties hold when \(G\) is directed).
  \item DFS(\(v\)) visits all vertices in the connected component of \(v\).
  \item The discovery edges form a tree: the DFS-tree of \(v\).
    \begin{itemize}
      \item justification: never visit a vertex again\(\Rightarrow\) no cycles
      \item we can keep track of the DFS tree by storing, for each vertex \(w\), its parent
    \end{itemize}
  \item The non-discovery (non-tree) edges always lead to a parent.
  \item If \(G\) is given as an adjacency-list of edges, then DFS(\(v\)) takes \(O(|V|+|E|)\) time.
\end{itemize}
Putting it all together:

Proposition: Let $G=(V,E)$ be an undirected graph represented by its adjacency-list. A DFS traversal of $G$ can be performed in $O(|V|+|E|)$ time and can be used to solve the following problems:

- testing whether $G$ is connected
- computing the connected components (CC) of $G$
- computing a spanning tree of the CC of $v$
- computing a path between 2 vertices, if one exists
- computing a cycle, or reporting that there are no cycles in $G$