Breadth First Search & Depth First Search

Problem: Given a graph $G(V,E)$, and a starting node $s \in V$, find all nodes reachable from $s$ (i.e., in the connected component of $s$).

Representing graphs

1. Adjacency Matrix

   $a_{ij} = \begin{cases} 
   1 & \text{if } \{i,j\} \in E \\ 
   0 & \text{otherwise} 
   \end{cases}$

   - $O(n^2)$ storage space
   - Given $i$, find all neighbors of $i$: $O(n)$ time.
   - Is $\{i,j\}$ an edge?: $O(1)$ time.

2. Adjacency list representation.

   ![Diagram of a graph with nodes 0, 1, 2, 3, 4, 5 and edges connecting them.]

   - $0: 2$
   - $2: 1-4-5-3$
   - $3: 2-4-5$
   - $4: 2-3$
   - $5: 2-3$
For every node, list of its neighbors is given.
- $O(m)$ space where $m = \#\text{edges}$
- Find all neighbors of $i$ : $O(d)$ time
  IS $\{ij\}$ an edge ? : $O(d)$ time
  $d = \text{maximum degree of any node}$

---

$x$

Theorem. Given a graph $G(V,E)$ in adjacency list rep., connected component of $s$ can be found in $O(m+n)$ time using B.F.S. on D.F.S.

Breadth First Search
- let layer $L_0$ be single node $s$: $L_0 = \{s\}$
- let $L_{i+1} = \{v \mid v \notin L_0 \cup L_1 \cup L_2 \ldots \cup L_i, \exists (u,v) \in E \text{ for some } u \in L_i\}$
  for $i = 0, 1, 2, \ldots, k$ so that $L_{k+1} = \emptyset$
  then stop
- Vertices reachable from $s$ are
decomposed into layers

\[ L_0, L_1, L_2, \ldots, L_K. \]

Claim

1. There are no edges between layers \( L_i \) and \( L_j \) if \( i+1 < j \). I.e., all edges are either inside some layer \( L_i \) or between adjacent layers \( L_i, L_{i+1} \).

2. \( L_i \) is precisely the set of vertices at distance \( i \) from \( S \).

3. \( L_0 \cup L_1 \cup \ldots \cup L_K \) is the connected component of \( S \).

Proof. Exercise.
Theorem BFS can be performed in $O(m+n)$ time.

Proof Here is the algorithm.

Maintain a marker for all vertices indicating whether they have been visited.

Initialize $L_0 = \{ s \}$ and mark $s$. All other vertices are unmarked.

For $i = 0, 1, 2, 3, \ldots$

- Go over all $u \in L_i$.

- For every $u \in L_i$, go over all edges $(u, v)$ incident on $u$. If $v$ hasn’t been marked, then mark it and add it to list $L_{i+1}$. Also add the edge $(u, v)$ to “BFS tree”.

The shortest paths from $s$ are given by the unique paths in the BFS tree.
Depth first search \( G(V,E), S \).

- Attempt to visit a new/undiscovered node if possible.
- Backtrack if necessary.

Example

**DFS tree**

**BFS tree**
Algorithm

$\text{Explored}(v) = F \ \forall v \in V.$

$\text{DFS}(s)$:

$\text{Explored}(v) = T.$

Let $u_1, u_2, \ldots, u_k$ be neighbors of $v$.

For $i=1,2,\ldots,k$ {

    If $\text{Explored}(u_i) = F$, $\text{Add edge } (v, u_i) \text{ to DFS tree.}$

}

Theorem $\text{DFS}$ runs in $O(m+n)$ time.

Proof: each edge is examined $O(1)$ times.
Stack implementation of DFS

- \( \text{Explored}(v) = \text{F} \quad \forall \ v \in V \).
- Initialize stack to \( \{ S \} \).

\[
\text{While} \{ \text{stack} \neq \emptyset \} \{
\quad \text{Take topmost node} \ u \ \text{from the stack}.
\quad \text{If} \ \text{Explored}(u) = \text{F} \ \text{then} \{
\quad \quad \text{set} \ \text{Explored}(u) = \text{T}.
\quad \quad \forall (u,v) \in E, \ \text{push} \ v \ \text{to stack}.
\}\}
\]

Applications of BFS

Def \( G(V,E) \) is bipartite if \( V = U \cup W \) and every edge has one endpoint each in \( U \) and \( W \).
Fact: G is bipartite if and only if G has no odd cycle.

Problem: Given G(V,E) in adjacency list representation, decide if G is bipartite.

Algorithm: O(m+n) time.
- Do BFS starting at any node.
- Let L₀, L₁, L₂, ..., Lₖ be layers.
- If there is no edge inside any layer Lᵢ, declare G to be bipartite.
- Else declare G to be non-bipartite.

Proof of correctness:
1. If no edge inside any layer Lᵢ, then let
   \[ U = L₀ \cup L₂ \cup L₄ \cup L₆ \cdots \]
   \[ W = L₁ \cup L₃ \cup L₅ \cup L₇ \cdots \]
   \( U \cup W \) is a bipartition as all edges are across layers Lᵢ and Lᵢ₊₁.
Suppose there is an edge $u,v$ inside $L_i$.

Then $s \leadsto v \leadsto u \leadsto i \leadsto s$ is a closed walk of length $2i+1$ (odd). Hence $G$ is not bipartite.

**Note.** BFS, DFS can also be performed on directed graphs.

- i.e. - Given directed graph $G(V,E)$, node $s$, all nodes reachable from $s$ can be found in $O(m+n)$ time.
- Let $G_{\text{reverse}}$ be the graph $G$ with all edge directions reversed. By performing BFS/DFS on $G_{\text{reverse}}$, one can find
in $O(m+n)$ time, all nodes from which $s$ can be reached.

Def Let $G(v, E)$ be a directed graph. $G$ is called strongly connected if
$\forall u, v \in V$, there is $u \leftrightarrow v$ path (and also $v \leftrightarrow u$ path).

Problem Given dir. $G(v, E)$, decide if $G$ is strongly connected.

Algorithm $O(m+n)$ time:
Fix some $s \in V$. Note that $G(v, E)$ is strongly connected iff
(1) All nodes are reachable from $s$.
(2) $s$ can be reached from all nodes.

Both tasks (1), (2) can be checked in $O(m+n)$ time by BFS/DFS as described.