Given a graph $G(V,E)$, it can be represented in two ways:

1. **Adjacency-List Representation**
   - Consists of:
     - An array $A$ of $|V|$ lists, one for each vertex in $V$
     - $\forall u \in V$, $A[u]$ contains pointers to all vertices $v$ such that $(u,v) \in E$
   - The vertices in $A$ are typically stored in an arbitrary order.

2. **Adjacency-Matrix Representation**
   - Given $G(V,E)$, assume numbering of vertices as $1, 2, \ldots, |V|$ in an arbitrary manner.
   - The adjacency matrix representation is given by a matrix $Adj = (a_{ij})$ of size $|V| \times |V|$
   - $a_{ij} = 1$ if $(i,j) \in E$
   - $a_{ij} = 0$ otherwise
Example

Adjacency List

For a directed graph
Properties

- If \( G \) is a directed graph
  - sum of lengths of all adjacency lists is \( |E| \)

- If \( G \) is an undirected graph
  - sum is \( 2|E| \)

- Amount of memory required by the adjacency list is \( O(V + E) \)

- Search presence of edge \((u, v)\) in Adjacency list representation requires (assuming we know, \( u, u \in V \))
  - searching the adjacency list of \( u \) (or \( v \))

- In an adjacency matrix, this can be done in \( O(1) \) time

- Adjacency matrix requires \( O(V^2) \) memory, independent of the number of edges.

- Adjacency matrix is wasteful for sparsely connected graphs.
Questions

Defn
- In-degree: Number of edges pointing to a given vertex
- Out-degree: Number of edges pointing out of a given vertex.

Given a directed graph \( G(V, E) \) and its adjacency list representation:

1. How long does it take to compute the out-degree of every vertex? \( O(E) \)

2. Compute the in-degree of every vertex.
   - Simple: in-degree of 1 vertex is \( O(E) \) of \( V \) vertices \( O(VE) \)
   - Refined: Maintain an array of size \( |V| \). Traverse the adjacency list representation and update the array whenever the corresponding vertex is in the adjacency list of any vertex. \( O(E) \)
Computing the square of a directed graph $G(V, E)$ is $G^2 = (V, E^2)$:

$(u, w) \in E^2$ iff for some $v \in V$, both $(u, v) \in E$ and $(v, w) \in E$.

That is, $G^2$ contains an edge between $u$ and $w$ whenever $G$ has a path with exactly two edges between $u$ and $w$.

Give an efficient algorithm for computing $G^2$ from $G$ for both adjacency list and adjacency matrix representations.

For each vertex $v$ in $G$:

- Do for each element $e$ in the connectivity list of $v$:
  - Do scan the connectivity list of $e$:
    - Update $E^2$.

$O(V^3)$ for connectivity matrix

$O(V) + O(E) + O(E^2)$?
Breadth-First Search (BFS)

- One of the most fundamental traversal algorithms
- Used in shortest path algorithms
- Used in MST algorithms

Idea: Given G(V, E) and a "source" vertex s, BFS searches G to find every vertex reachable from s. It also computes the shortest path (in terms of number of edges) from s to all reachable vertices. It represents this information in a "breadth-first tree" rooted at s.

BFS works on both directed and undirected graphs.

(a) The algorithm is called "breadth-first" because it discovers all vertices at distance k from s before discovering any vertex at distance k+1.

(b) BFS uses a coloring scheme for each vertex to track progress (W, G, B): white, gray, black
   - All vertices start with color W (white)
   - When a vertex is encountered for the first time, it is considered "discovered." Once a vertex is discovered, it becomes non-white.
- G and B vertices are therefore discovered.

- To ensure that the search progresses in a breadth-first manner, G and B vertices are distinguished:
  - If $(u, v) \in E$ and $u$ is black, then $v$ is either gray or black.
    - All vertices adjacent to B-vertices have been discovered.
    - G-vertices may have some W-vertices as neighbors and represent the boundary between discovered and undiscovered vertices.

BFS constructs a breadth-first tree, mutually containing only $s$. When a white vertex $v$ is discovered, the edge while scanning the adjacency list of a discovered vertex $u$, the vertex $v$ and edge $(u, v)$ are added to the tree.
  - $u$ is called the predecessor of $v$
  - Similarly: if $u$ is on a path from $s$ to $v$
    - $u$ is an ancestor of $v$
    - $v$ is a descendant of $u$
Let $G(V, E)$ be represented using adjacency list.

Let the color of each vertex $u$ be stored in color[$u$] $\forall u \in V$

Let the predecessor of $u$ be stored in $\pi [u]$. If $u$ has no predecessor, then $\pi [u] = \text{NIL}$.

The distance from source $s$ to vertex $u$ is stored in $d[u]$

The algorithm also uses a FIFO queue $(Q)$ to manage the gray vertices

```
BFS(G, s)
```

1. For each vertex $u \in V[G] - \{s\}$
   - do $\text{color}[u] \leftarrow \text{W}$
   - $d[u] \leftarrow \infty$
   - $\pi [u] \leftarrow \text{NIL}$

2. $\text{color}[s] \leftarrow \text{G}$
3. $d[s] \leftarrow 0$
4. $\pi [s] \leftarrow \text{NIL}$
5. $Q \leftarrow \{s\}$
6. while $Q \neq \emptyset$
   - do $u \leftarrow \text{head}[Q]$
     - for each $v \in \text{Adj}[u]$
       - do if $\text{color}[v] = \text{W}$
         - then $\text{color}[v] \leftarrow \text{G}$
         - $d[v] \leftarrow d[u] + 1$
         - $\pi [v] \leftarrow u$
       - $\text{Enqueue}(Q, v)$
   - $\text{Dequeue}(Q)$

Paint every vertex whose
initialize
$d[u]$ and $\pi [u]$

$s$ is gray.

$s$ start with $s$ in the $Q$
Hence for $P$, all vertices in
$Q$ will be gray

The loop continued, till there are vertices (Gray) which have not
had their adjacency lists examined.
\( \emptyset = \emptyset s_3 \)
\( d[s_3] = 0 \)
\( n_i[s_3] = Nil \)

head = s
\( \emptyset = \emptyset w_2, s_3, \) head = s

\( \Pi(w_2) = s \)
\( \Pi(s_3) = s \)

head = w_0
\( \emptyset = \emptyset w_0, t, x_3 \) head = w_0

\( \Pi(t) = 0. \)
\( \Pi(x_3) = 0. \)

head = r
\( \emptyset = \emptyset s_3, t, x_3, v_3 \)

\( \Pi(u) = 2 \)

head = t
\( \emptyset = \emptyset \) head = t

\( \Pi(u) = t \)

head = x
\( \emptyset = \emptyset u_3, v_3 \)

\( \Pi(y) = x \)
\[ \text{head} = V \]
\[ S = \emptyset \]

\[ \text{head} = U \]
\[ S = \emptyset \]

\[ \text{head} = U \]
\[ S = \emptyset \]

\[ a \quad s \quad t \quad u \]
\[ v \quad n \quad x \quad y \]
Analysis

- After initialization, no vertex is ever whited.
- The test in line #12 thus ensures that every vertex is enqueued once and (therefore dequeued once).

Enqueue and dequeue is $O(1)$
so queue operations is $O(V)$

- Adjacency list of a vertex is scanned only when it is dequeued.
  
  Sum of lengths of all adjacency lists is $O(E)$
  Therefore total time spent in scanning adjacency lists is $O(E)$

- Initialization is $O(V)$

- Total : $O(V + E)$