### Basic Definitions

Let $G = (V, E)$, be a graph, $|V| = n$, $|E| = m$. For any $x \in V$, the degree of $x$, or $\deg(x)$, is the number of vertices adjacent to $x$. Maximum and minimum degree in $G$, are denoted, respectively, by $\Delta(G)$ and $\delta(G)$.

### Data Structures for Graphs

- **Adjacency Matrix:** an $n \times n$ matrix $M$, so that $M[i, j] = 1$ if and only if $ij \in E$, requiring $O(n^2)$ storage.
- **Adjacency List:** A Collection of $n$ linked lists, so that list $i$ contains all vertices adjacent to vertex $i$, requiring $O(n + m)$ storage.
- **Adjacency List:** In the implementation, a one dimension array $L$, can be used, so that, $L[i]$ is list containing vertices adjacent to $i$. Alternatively, $L$ can be linked list itself.
Which Data Structure is Better?

- How fast $\delta(G)$ and $\Delta(G)$ can be computed?
- How fast $G$ can be updated, if a node $x$ is deleted?
- Given $x, y \in V$, how fast one can determine, if $xy \in E$?
- How effective we can simulate a dynamic network?
Breadth First Search (BFS)

BFS(s)

1. \( \text{InQ}(s); \ visit[s] \leftarrow 1 \)
2. While \( Q \neq \emptyset \) do
3.   \( x \leftarrow \text{DeQ}() \)
4.   For all \( y \in G[x] \) do
5.     If \( visit[y] = 0 \) then \( visit[y] \leftarrow 1; \ parent[y] \leftarrow x; \ \text{InQ}(y) \) endif
6. EndWhile
Graph Traversals

### BFS

**Theorem**

*BFS can be implemented in* \(O(|V| + |E|)\) *time and requires* \(O(|V| + |E|)\) *storage, provided that* \(G\) *is stored in the adjacency list format.*

**Theorem**

*A non-trivial Graph* \(G\) *is bipartite iff it does not have an odd cycle. Moreover, we can decide if* \(G\) *is bipartite, or not, using BFS in\( O(|V| + |E|)\) *time.*

### Additional Applications of BFS

Finding a spanning tree, finding shortest paths from a source to all other vertices, finding connected components, determining if \(G\) is acyclic, determining if \(G\) is bipartite, can be done in \(O(|V| + |E|)\) time.
Graph Traversals

**Depth First Search (DFS)**

**DFS(s)**

1. \(visit[s] \leftarrow 1\)
2. For all \(y \in G[s]\) do
3. If \(visit[y] = 0\) then \(DFS(y)\)

**Theorem**

*DFS can be implemented in \(O(|V| + |E|)\) time and requires \(O(|V| + |E|)\) storage, provided that \(G\) is stored in the adjacency list format. Moreover using DFS, finding a spanning tree, finding connected components, finding biconnected components, determining if \(G\) is acyclic, and determining if \(G\) is bipartite, can be done in \(O(|V| + |E|)\) time.*
Topological Ordering

Directed Acyclic Graphs: DAG

Let $G = (V, E)$ be a directed graph, $G$ is a DAG, if it does not contain any cycles. A topological ordering of a DAG is an ordering $v_1, v_2, \ldots, v_n$ of vertices so that for any $v_i v_j \in E$, we have $i < j$. The following simple algorithm that uses a Queue finds a topological ordering in a DAG.

TOPO(G)

1. For each $x \in V$ do if $\text{indeg}(x) = 0$ then $\text{InQ}(x)$
2. $i \leftarrow 0$
3. While $Q \neq \emptyset$ do
4. $v \leftarrow \text{DeQ}(); i \leftarrow i + 1; \text{Num}[i] \leftarrow v$
5. For all vertices $y$ adjacent from $v$ do
   $\text{indeg}(y) = \text{indeg}(y) - 1;$ If $\text{indeg}(y) = 0$ then $\text{InQ}(y)$ Endfor
6. EndWhile
Topological Ordering

Theorem

Let $G = (V, E)$ be a DAG, then the previous algorithm finds a topological ordering of $G$ in $O(|V| + |E|)$ time.