Graph Algorithms
Graph $G$

$G.V = \text{Set of vertices of graph } G$.

$G.E = \text{Set of edges of graph } G$.

$(v_i, v_j) = \text{edge with endpoints } v_i \text{ and } v_j$.

Vertices $v_i \in G.V$ and $v_j \in G.V$ are \textit{incident} on edge $(e_i, e_j) \in G.E$. 
Vertex Degree

deg(v_i) = degree of v_i = \# edges incident on v_i.

What is the degree of each vertex?
Sum of $\text{deg}(v_i)$

$m =$ number of graph edges.

Proposition: $\sum_{v_i \in V} \text{deg}(v_i) = 2m$.

Proof 1:

$\sum_{v_i \in V} \text{deg}(v_i) =$ sum of $\# $ edges incident on vertices
   $=$ sum of $\# $ vertices incident on edges
   $= 2 \times (\# $ edges $) = 2m$.

Proof 2: Let $\delta_{ij} = 1$ if $e_j$ is incident on $v_i$ and 0 otherwise.

$$\sum_{v_i \in V} \text{deg}(v_i) = \sum_{v_i \in V} \sum_{e_j \in E} \delta_{ij} = \sum_{e_j \in E} \sum_{v_i \in V} \delta_{ij} = \sum_{e_j \in E} 2 = 2m.$$
Graph Representation

Adjacency matrix:

<table>
<thead>
<tr>
<th></th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$ :</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_2$ :</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$v_3$ :</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$v_4$ :</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$v_5$ :</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$A[i, j] = 1$ if $(v_i, v_j)$ is an edge.

Adjacency lists:

- $v_1 : (v_2, v_3)$
- $v_2 : (v_1, v_3, v_4, v_5)$
- $v_3 : (v_1, v_2, v_5)$
- $v_4 : (v_2)$
- $v_5 : (v_2, v_3)$

$V[i].\text{degree} = \text{size of adj list.}$
$V[i].\text{AdjList}[k] =$
$k^{\text{th}} \text{ element of the adjacency list.}$
Sample Graph Algorithm

Input : Graph $G$ represented by adjacency lists.

Func($G$)

1. $k \leftarrow 0$

2. $\textbf{foreach}$ vertex $v_i$ in $G.V$ $\textbf{do}$ /* $G.V =$ vertices of $G$ */

3. $\quad \textbf{foreach}$ edge $(v_i, v_j)$ incident on $v_i$ $\textbf{do}$

4. $\quad \quad k \leftarrow k + 1$

5. $\quad \textbf{end}$

6. $\textbf{end}$

7. $\textbf{return}$ ($k$);
Connected Graphs

**Definition.** A path in a graph is a sequence of vertices \((w_1, w_2, \ldots, w_k)\) such that there is an edge \((w_i, w_{i+1})\) between every two adjacent vertices in the sequence.

**Definition.** A graph is **connected** if for every two vertices \(v, w \in G.V\), the graph has some path from \(v\) to \(w\).

Is the following graph connected?
Connected Graphs

Is the following graph connected?
Connected Graphs

Is the following graph connected?
Query Connected: Depth First Search

procedure QueryConnected(G)
1 foreach vertex \( v_i \) in \( G.V \) do /* \( G.V = \) vertices of \( G \) */
2 \( v_i \).mark ← NotVisited;
3 end
4 DFS(G,1);
5 foreach vertex \( v_i \) in \( G.V \) do
6 if \( (v_i \).mark = \) NotVisited) then return false;
7 end
8 return true;
Depth First Search

procedure DFS(G, i)

/* Depth first search from vertex vi */

1  G.V[i].mark ← Visited;

2  foreach edge (i, j) incident on vertex i do

3    if (G.V[j].mark = NotVisited()) then

4      DFS (G,j);

5    end

6  end
procedure DFStree(G, i)

/* Depth first search from vertex vi */

1   G.V[i].mark ← Visited;

2   foreach edge (i, j) incident on vertex i do
3       if (G.V[j].mark = NotVisited()) then
4           G.V[j].parent ← i;
5           DFStree (G, j);
6       end

7   end
Example
Directed Graphs
Directed Graph Representation

Adjacency matrix:

\[
\begin{bmatrix}
  v_1 & v_2 & v_3 & v_4 & v_5 \\
  v_1 : & 0 & 1 & 1 & 0 & 0 \\
  v_2 : & 0 & 0 & 0 & 1 & 1 \\
  v_3 : & 0 & 1 & 0 & 0 & 1 \\
  v_4 : & 0 & 0 & 0 & 0 & 0 \\
  v_5 : & 0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

Adjacency lists:

- \( v_1 \) out: \((v_2, v_3)\)  \( v_1 \) in: ()
- \( v_2 \) out: \((v_4, v_5)\)  \( v_2 \) in: \((v_1, v_3)\)
- \( v_3 \) out: \((v_2, v_5)\)  \( v_3 \) in: \((v_1, v_5)\)
- \( v_4 \) out: ()  \( v_4 \) in: \((v_2)\)
- \( v_5 \) out: \((v_3)\)  \( v_5 \) in: \((v_2, v_3)\)

\( A[i, j] = 1 \) if \((v_i, v_j)\) is an edge.
**Sum of outdeg\((v_i)\) and sum of indeg\((v_i)\)**

\[ m = \text{number of graph edges.} \]

Proposition: \[ \sum_{v_i \in G.V} \text{outdeg}(v_i) = m. \]

Proposition: \[ \sum_{v_i \in G.V} \text{indeg}(v_i) = m. \]
Sample Graph Algorithm

Input: Directed graph $G$ represented by adjacency lists.

\begin{verbatim}
Func($G$)
1 $k \leftarrow 0$;
2 foreach vertex $v_i$ in $G.V$ do /* $G.V =$ vertices of $G$ */
3     foreach directed edge $(v_i, v_j)$ incident on $v_i$ do
4         $k \leftarrow k + 1$;
5     end
6 end
7 return $(k)$;
\end{verbatim}
How many vertices are reachable by a directed path from $v_1$?
Directed Graph

How many vertices are reachable by a directed path from $v_{12}$?
procedure NumReachable(G, k)
1   foreach vertex $v_i$ in $G.V$ do  /* $G.V =$ vertices of $G$ */
2     $v_i$.mark ← NotVisited;
3   end
4   DirectedDFS(G,k);
5   count ← 0;
6   foreach vertex $v_i$ in $G.V$ do
7     if ($v_i$.mark = Visited) then count ← count + 1;
8   end
9   return count;
procedure DirectedDFS(G, i)

  /* Depth first search from vertex vi */

1. G.V[i].mark ← Visited;

2. foreach directed edge (i, j) incident on vertex i do

3.   if (G.V[j].mark = NotVisited()) then

4.     DFS (G, j);

5.   end

6. end
Is Reachable?

Is vertex $v_{12}$ reachable by a directed path from vertex $v_1$?
Is Reachable?

Is vertex $v_1$ reachable by a directed path from vertex $v_{12}$?
procedure IsReachable(G, k, q)

1 foreach vertex \( v_i \) in \( G.V \) do  /* \( G.V \) = vertices of \( G \) */
2 \( v_i \).mark ← NotVisited;
3 end
4 DirectedDFS(G,k);
5 if (\( v_q \).mark = Visited) then return true;
6 else return false;
Topological Sort
A directed graph with no cycles is called a **directed acyclic graph** or **DAG**.
Topological Sort

Sort vertices of a directed acyclic graph so that for every edge \((v_i, v_j)\), vertex \(v_i\) precedes \(v_j\).
Topological Sort: Example
**Topological Sort**

```plaintext
procedure TopologicalSort(G)
   /* Report vertices of G in topologically sorted order */
   foreach vertex v_i in G.V do
      /* S is a stack of vertices. */
      if (indeg(v_i) = 0) then S.Push(v_i);
   end
   while (S is not empty) do
      v_i ← S.Pop();
      Report v_i;
      foreach edge (v_i, v_j) incident on v_i do
         Delete (v_i, v_j) from G;
         if (indeg(v_j) = 0) then S.Push(v_j);
      end
   end
```

8.29
Topological Sort: Simplified Implementation

procedure TopologicalSort(G)
/* Report vertices of G in topologically sorted order */
1 foreach vertex \(v_i\) in G.V do
   /* S is a stack of vertices. */
   2 inCount[\(v_i\)] ← indeg(\(v_i\));
   3 if (inCount[\(v_i\)] = 0) then S.Push(\(v_i\));
end
5 while (S is not empty) do
   6 \(v_i\) ← S.Pop();
   7 Report \(v_i\);
   8 foreach edge \((v_i, v_j)\) incident on \(v_i\) do
      9 inCount[\(v_j\)] ← inCount[\(v_j\)] − 1;
      10 if (inCount[\(v_j\)] = 0) then S.Push(\(v_j\));
end
12 end