Graph Algorithms

Graph

Graph G

$G.V$ = Set of vertices of graph G.

$G.E$ = Set of edges of graph G.

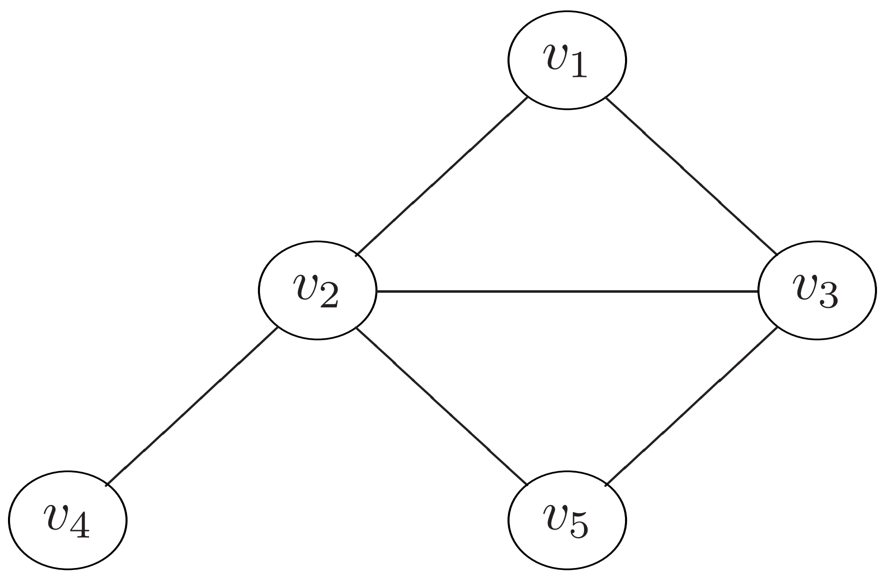
(v_i, v_j) = edge with endpoints v_i and v_j.

Vertices $v_i \in G.V$ and $v_j \in G.V$ are *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 $\deg(v_i)$

m = number of graph edges.

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

Proof 1:

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

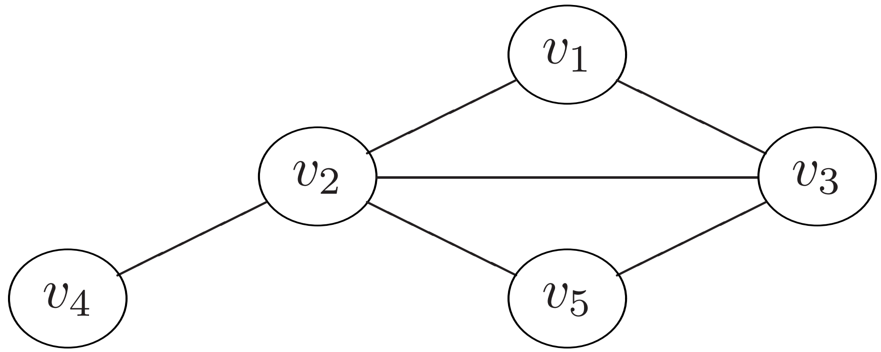
□

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

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

□

Graph Representation



Adjacency matrix:

	v_1	v_2	v_3	v_4	v_5
v_1 :	0	1	1	0	0
v_2 :	1	0	1	1	1
v_3 :	1	1	0	0	1
v_4 :	0	1	0	0	0
v_5 :	0	1	1	0	0

$A[i, j] = 1$ 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)

$V[i]$.degree = size of adj list.
$V[i]$.AdjList$[k]$ =
k'th element of the adjacency list.

Sample Graph Algorithm

Input : Graph G represented by adjacency lists.

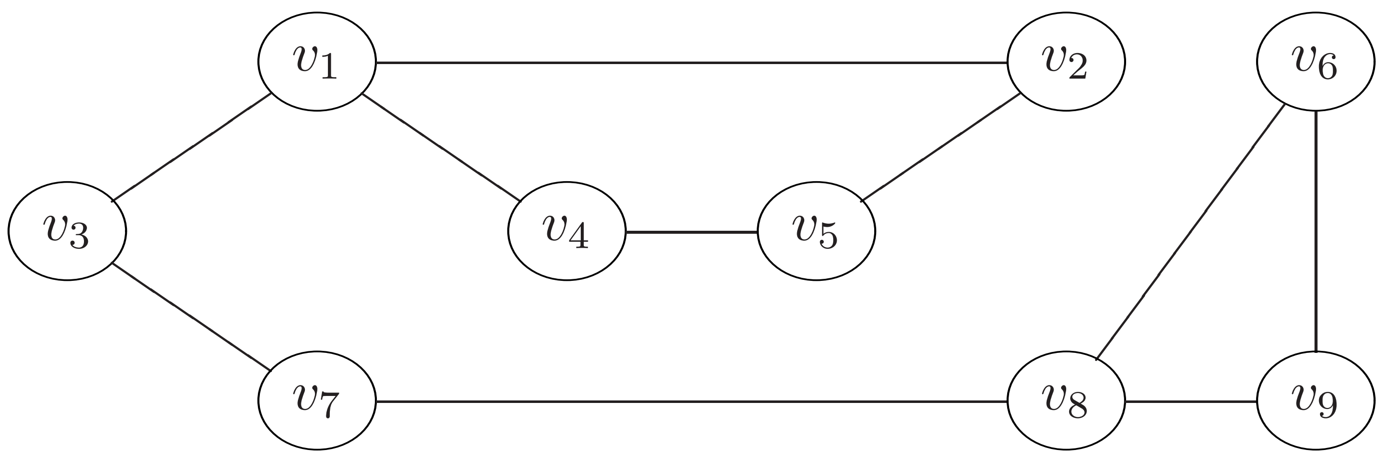
```
Func(G)
k ← 0;
foreach vertex v_i in G.V do        /* G.V = vertices of G */
    foreach edge (v_i, v_j) incident on v_i do
        k ← k + 1;
    end
end
return (k);
```

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.

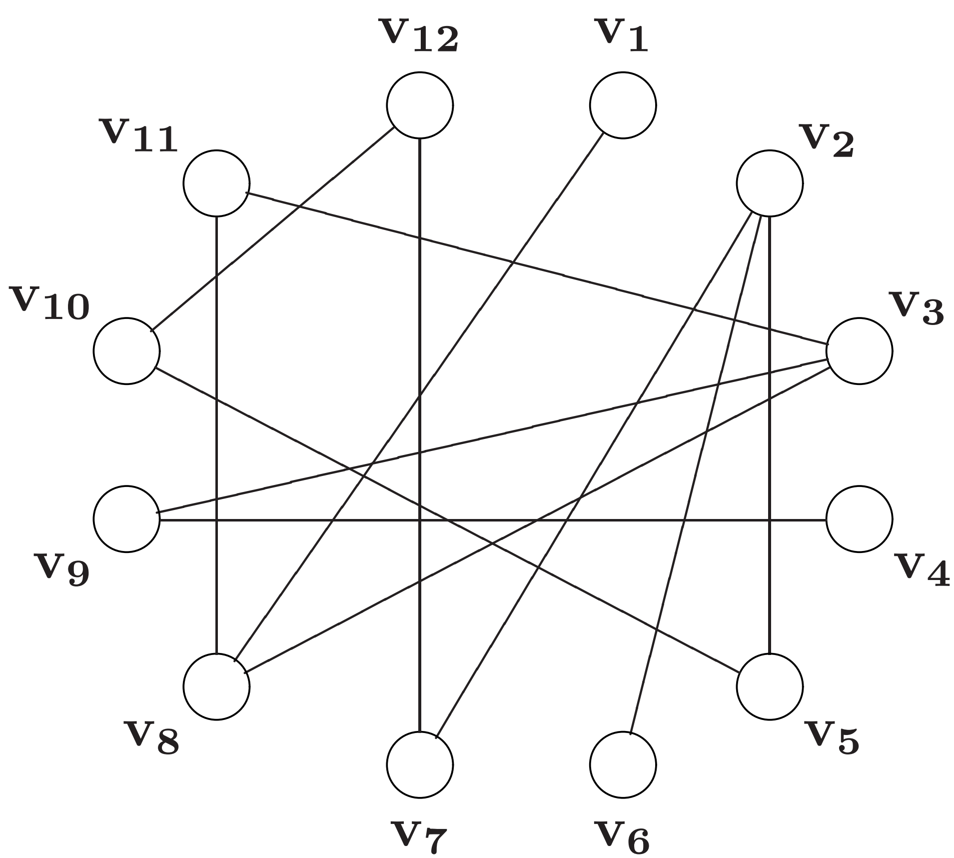
Definition. A graph is **connected** if for every two vertices $v, w \in G.V$, the graph has some path from v to w.

Is the following graph connected?



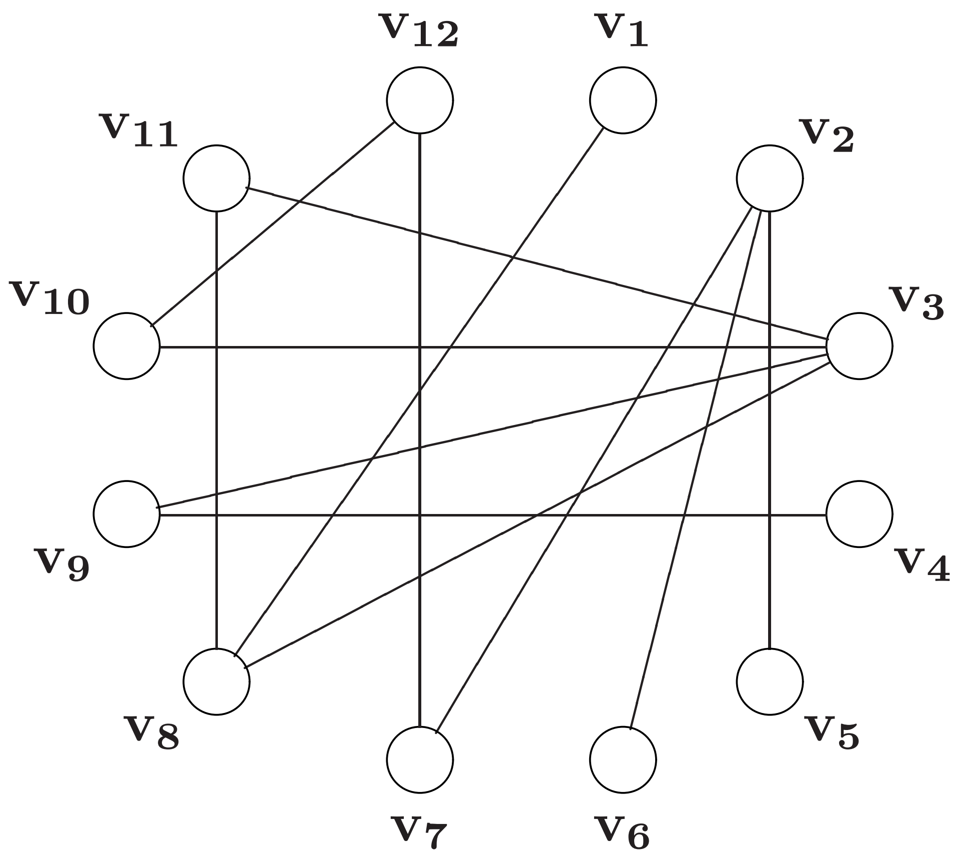
Connected Graphs

Is the following graph connected?



Connected Graphs

Is the following graph connected?



Query Connected: Depth First Search

```
procedure QueryConnected(G)
foreach vertex v_i in G.V do          /* G.V = vertices of G */
    v_i.mark ← NotVisited;
end
DFS(G,1);
foreach vertex v_i in G.V do
    if (v_i.mark = NotVisited) then return false;
end
return true;
```

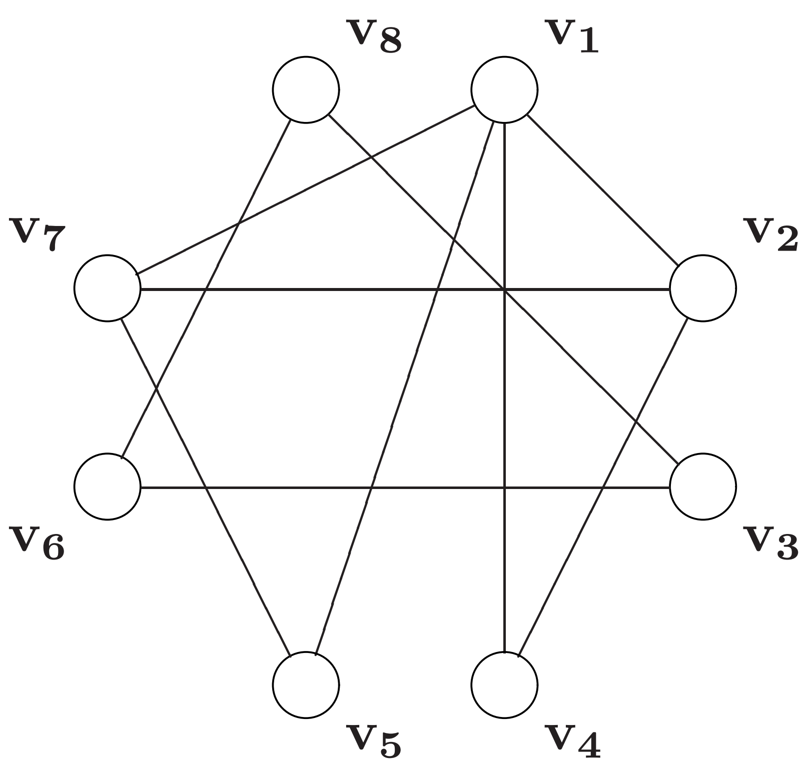
Depth First Search

```
procedure DFS(G, i)
/* Depth first search from vertex v_i */
G.V[i].mark ← Visited;
foreach edge (i, j) incident on vertex i do
    if (G.V[j].mark = NotVisited()) then
        DFS (G,j);
    end
end
```

Depth First Search Tree

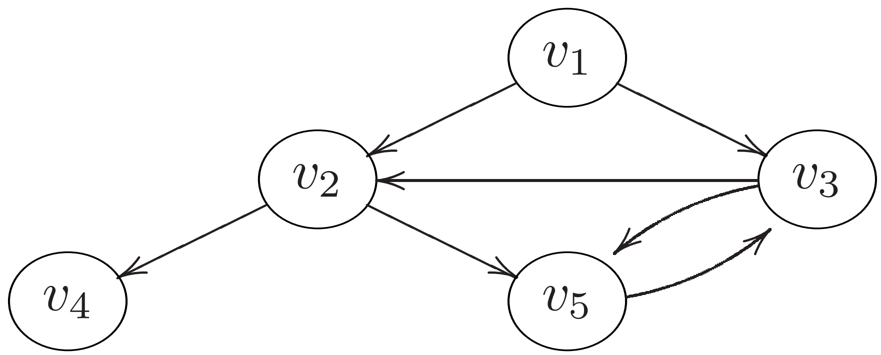
```
procedure DFStree(G, i)
/* Depth first search from vertex v_i */
G.V[i].mark ← Visited;
foreach edge (i, j) incident on vertex i do
    if (G.V[j].mark = NotVisited()) then
        G.V[j].parent ← i;
        DFStree (G,j);
    end
end
```

Example



Directed Graphs

Directed Graph Representation



Adjacency matrix:

$$
\begin{array}{r} \\ v_1 : \\ v_2 : \\ v_3 : \\ v_4 : \\ v_5 : \end{array}
\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}
$$

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

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)

Sum of $\text{outdeg}(v_i)$ and sum of $\text{indeg}(v_i)$

m = number of graph edges.

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

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

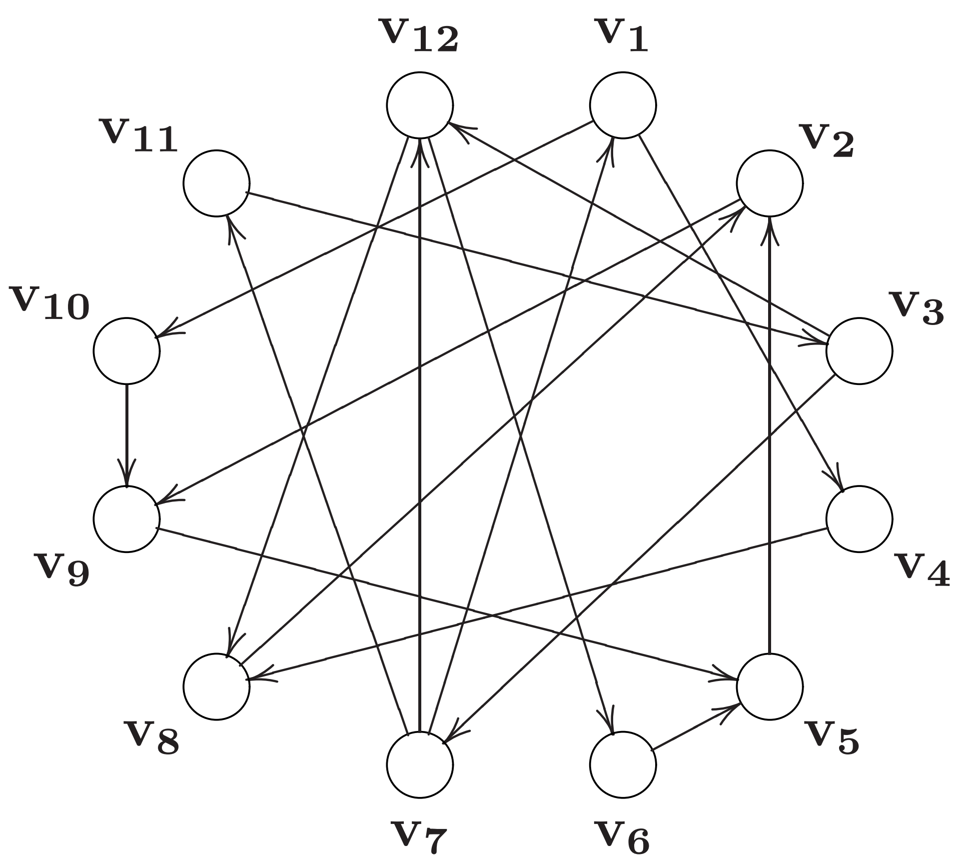
Sample Graph Algorithm

Input : Directed graph G represented by adjacency lists.

```
Func(G)
k ← 0;
foreach vertex v_i in G.V do        /* G.V = vertices of G */
    foreach directed edge (v_i, v_j) incident on v_i do
        k ← k + 1;
    end
end
return (k);
```

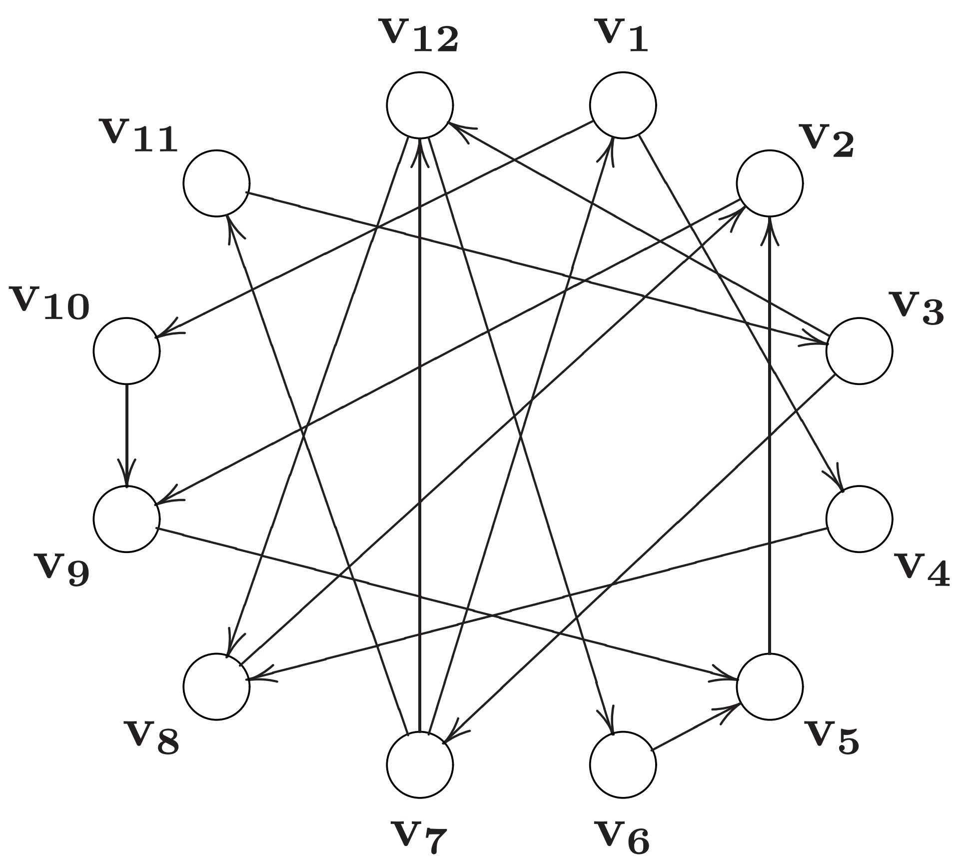
Directed Graph

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}?



Num Reachable: Depth First Search

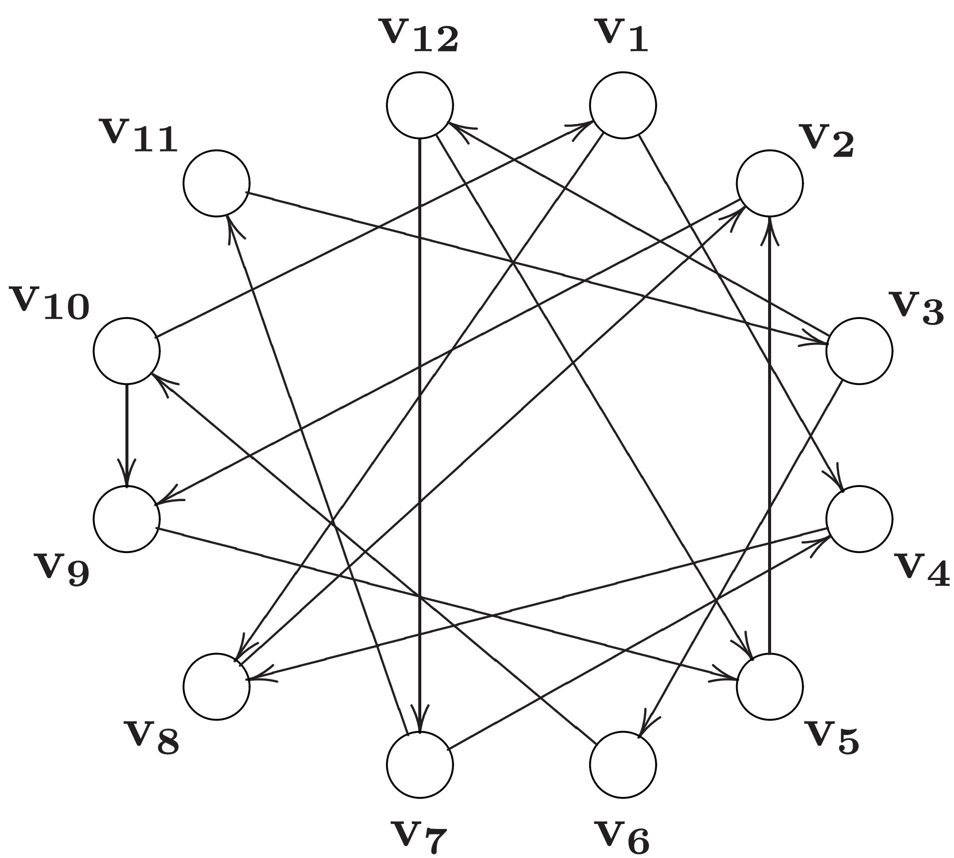
```
procedure NumReachable(G, k)
foreach vertex v_i in G.V do          /* G.V = vertices of G */
    v_i.mark ← NotVisited;
end
DirectedDFS(G,k);
count ← 0;
foreach vertex v_i in G.V do
    if (v_i.mark = Visited) then count ← count + 1;
end
return count;
```

Directed Depth First Search

```
procedure DirectedDFS(G, i)
/* Depth first search from vertex v_i                    */
G.V[i].mark ← Visited;
foreach directed edge (i, j) incident on vertex i do
    if (G.V[j].mark = NotVisited()) then
        DFS (G,j);
    end
end
```

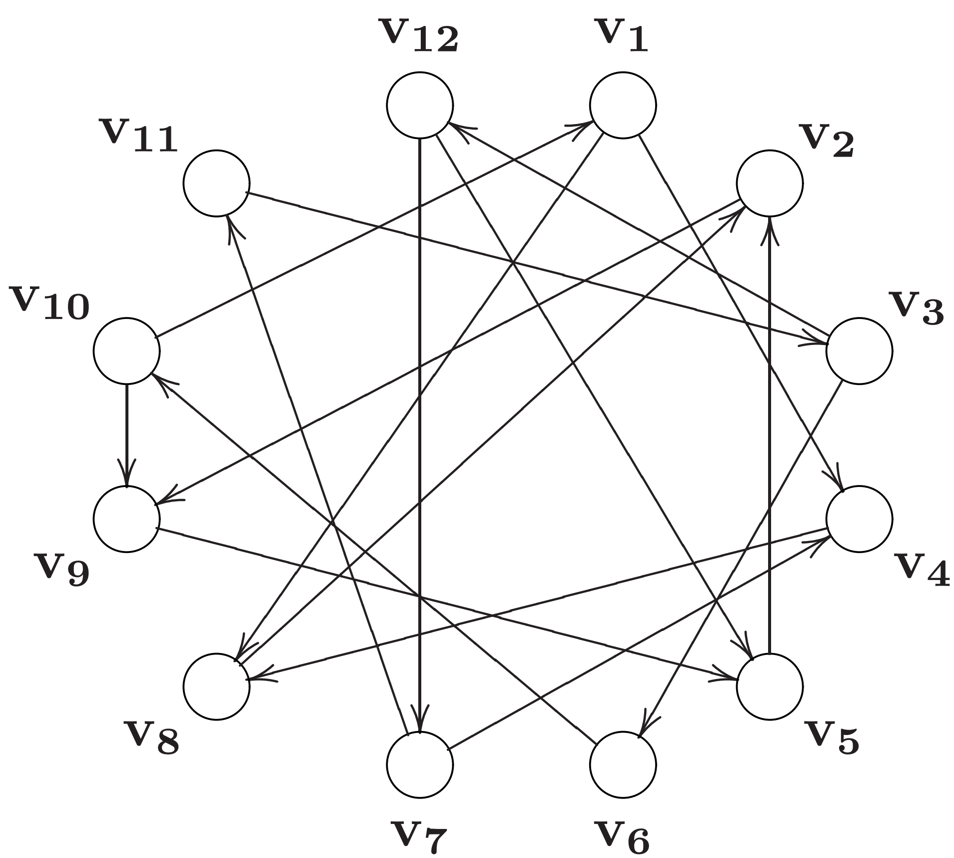
Is Reachable?

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}?



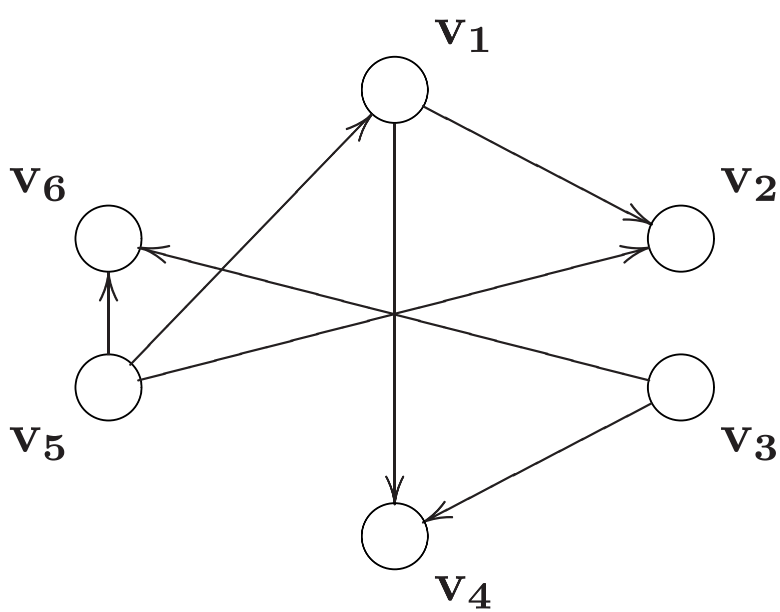
Is Reachable: Depth First Search

```
procedure IsReachable(G, k, q)
foreach vertex v_i in G.V do        /* G.V = vertices of G */
    v_i.mark ← NotVisited;
end
DirectedDFS(G,k);
if (v_q.mark = Visited) then return true;
else return false;
```

Topological Sort

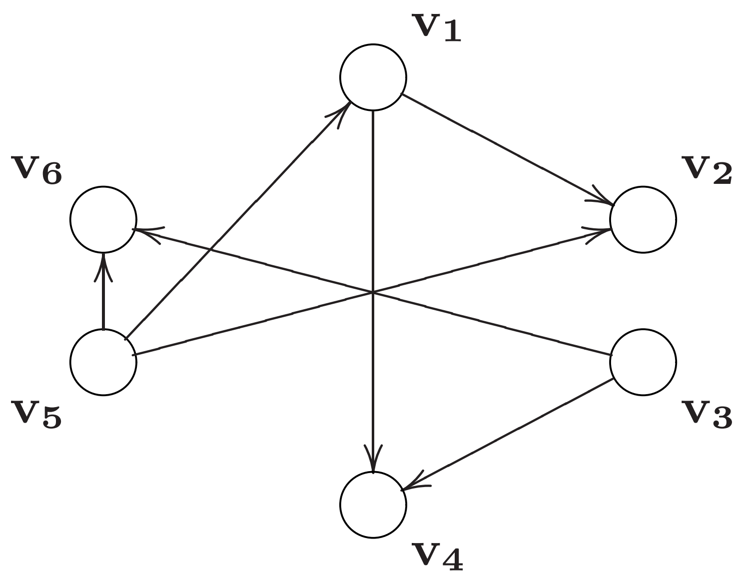
Directed Acylic Graph (DAG)

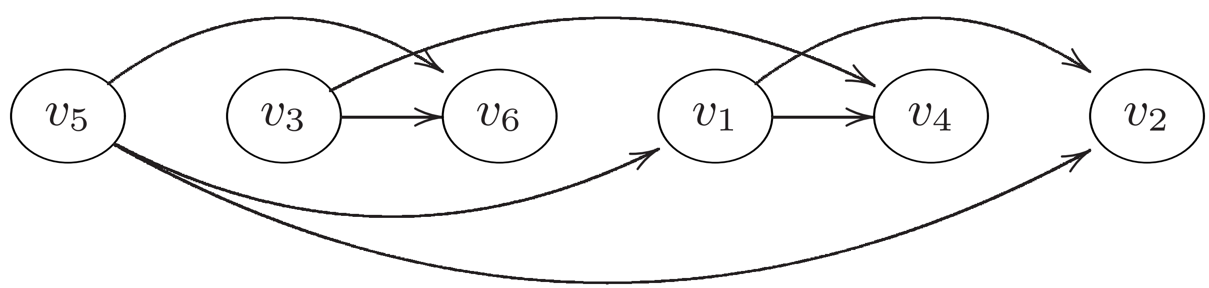
A directed graph with no cycles is called a **directed acyclic graph** or **DAG**.



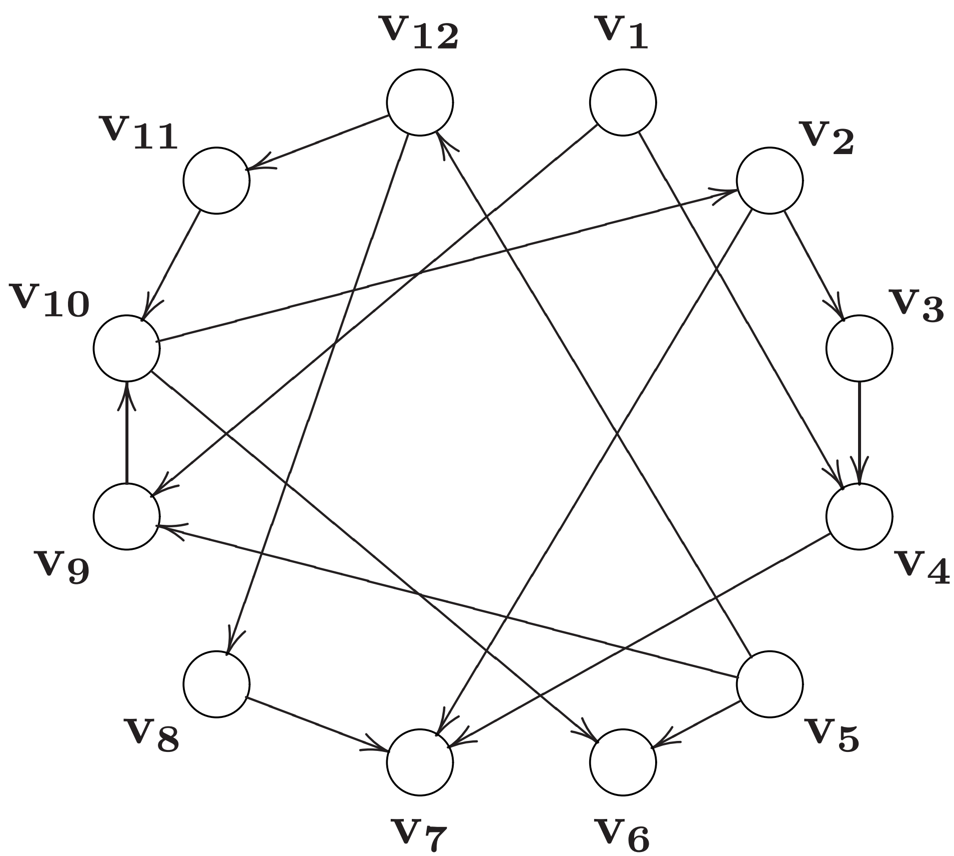
Topological Sort

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

```
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
```

Topological Sort: Simplified Implementation

```
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. */
    inCount[v_i] ← indeg(v_i);
    if (inCount[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
        inCount[v_j] ← inCount[v_j] − 1;
        if (inCount[v_j] = 0) then S.Push(v_j);
    end
end
```