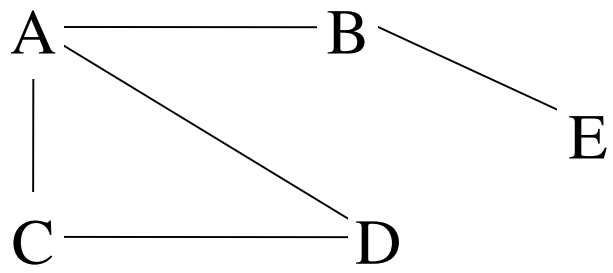
Graphs – Adjacency Matrix

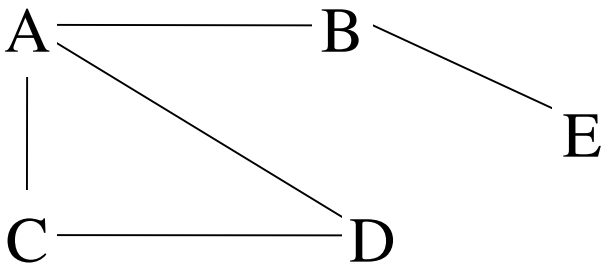
- Graph with N vertices: NxN adjacency matrix
- A[i][j] = 1 if vertex i is adjacent to vertex j; 0 otherwise

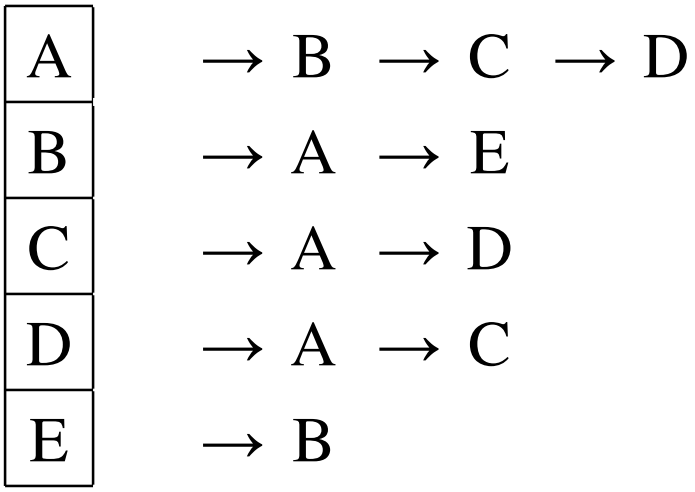


	A	**B**	**C**	**D**	**E**
A	0	1	1	1	0
B	1	0	0	0	1
C	1	0	0	1	0
D	1	0	1	0	0
E	0	1	0	0	0

Graphs – Adjacency List

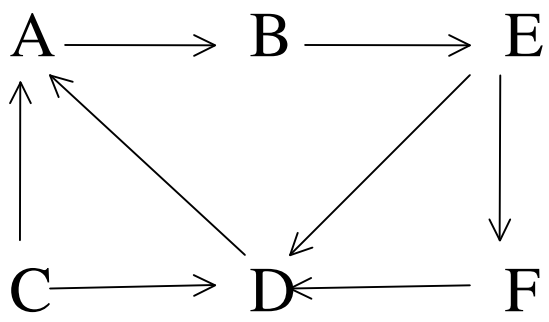
- Array of linked lists
- If vertex i is adjacent to vertex j then:
  - j is in the adjacency list for i and
  - i is in the adjacency list for j





Directed Graphs

- Adjacency matrix: A[i][j] = 1 iff there is an edge from i to j
- Adjacency list: j is in list i iff there is an edge from i to j



	A	**B**	**C**	**D**	**E**	**F**
A	0	1	0	0	0	0
B	0	0	0	0	1	0
C	1	0	0	1	0	0
D	1	0	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	1	0	0

A	→	B		
B	→	E		
C	→	A	→	D
D	→	A		
E	→	D	→	F
F	→	D		

Depth-first Search

Suppose that our starting vertex is v.

```
DFS(v)
{
  v.wasVisited = true;
  while there is an unvisited vertex u
     adjacent to v
  {
    DFS(u);
  }
}
```

Exercise: Think about how to do this with an iterative method

Breadth-first Search

- We need a queue Q that stores vertices.
- Suppose that our starting vertex is v.

```
v.wasVisited = true;
Q.enqueue(v);
while !Q.isEmpty()          // loop1
{
  v = Q.dequeue();
  while there is an unvisited vertex u
   adjacent to v            // loop2
  {
    u.wasVisited = true;
    Q.enqueue(u);
  }
}
```