Outline

Graphs

Graphs can represent airlines, electrical circuits, or computer networks.

A Graph G will consist of:

- A set V of **vertices** (nodes, points).
  (*Cities, circuit connections, computers*).
  We will use V to mean the **number** of vertices as well (mathematicians use *cardinality* notation, $|V|$).
- A set E of **edges** (lines connecting vertices).
  (*Air lanes, elements in a circuit, computer connections in a network*).
  We will use E to mean the **number** of edges as well.

Graphs

- A path is a series of edges connecting two vertices.
- In an undirected graph edges are “two-way streets”
- A connected graph is one in which every pair of vertices is connected by a path.
- A complete graph is one in which every pair of vertices is connected by an edge.
- Two vertices are adjacent if there is an edge connecting them.
- A cycle in a directed graph is a loop formed by adjacent vertices.

GRAPHS

In directed graphs:

- edges are "one-way streets" beginning at one vertex and ending at another.
- in-degree of a vertex = # of edges ending at that vertex.
- out-degree of a vertex = # of edges beginning at that vertex.
- A directed acyclic graph (DAG) is a directed graph containing no cycles.
- Vertex B is **adjacent** to vertex A if there is an edge from A to B.
- *Example*: A tree is a special type of DAG.

Graphs

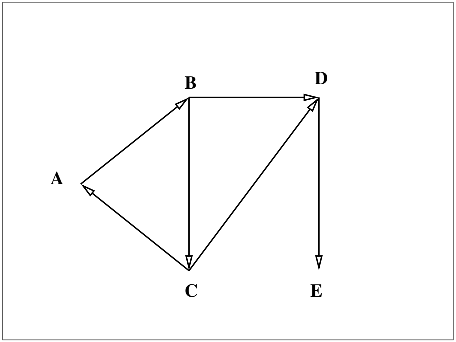
A directed graph example

in-degree of A =

out-degree of C =

in-degree of D =

Is there a cycle?



Graphs

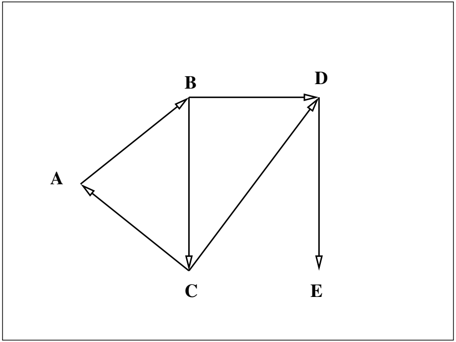
A directed graph example

in-degree of A = 1

out-degree of C = 2

in-degree of D = 2

Is there a cycle? Yes, $A \to B \to C \to A$ is a cycle



Graphs

- A graph is dense if it has many edges connecting vertices.
- A graph is sparse if it has much less than the maximum possible number of edges.
- The best implementation of a graph depends on how sparse it is.
- Two commonly used data structures to represent graphs are *adjacency matrix* and *adjacency list*.

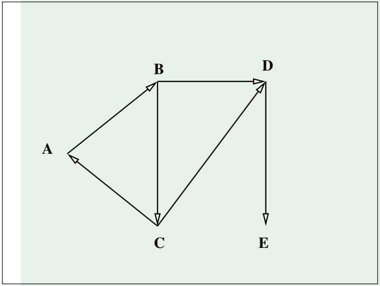
Representing Graphs

Adjacency Matrices

- An adjacency matrix has rows and columns of zeros and ones. 1 in column `i`, row `j` means that an edge connects vertex `i` with vertex `j` (i.e. vertices `i` and `j` are adjacent).
- An adjacency matrix is used to implement a dense graph.
- It requires $\Theta(V^2)$ time to find all the edges (by checking every entry in the matrix).

Representing Graphs

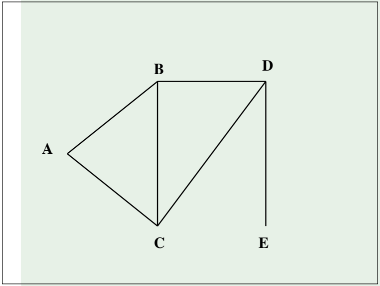
Adjacency Matrices (Directed Graph)



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

Representing Graphs

Adjacency Matrices(Undirected Graph)

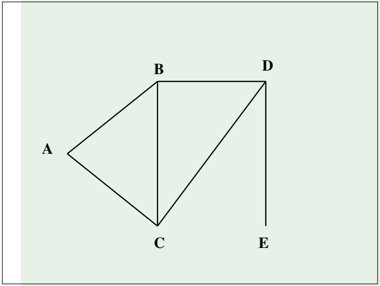


	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	1
E	0	0	0	1	0

The matrix is symmetric (entry at row i, column j is the same as at row j, column i), hence we need only half of the matrix to represent a graph (using diagonal to split it).

Representing Graphs

Adjacency Matrices(Undirected Graph)

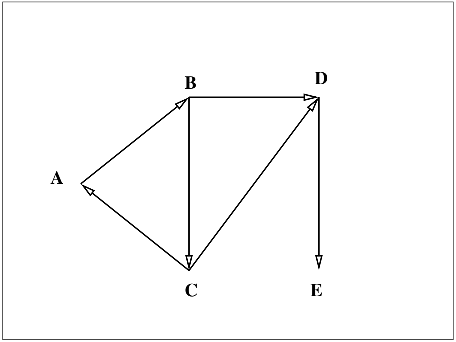


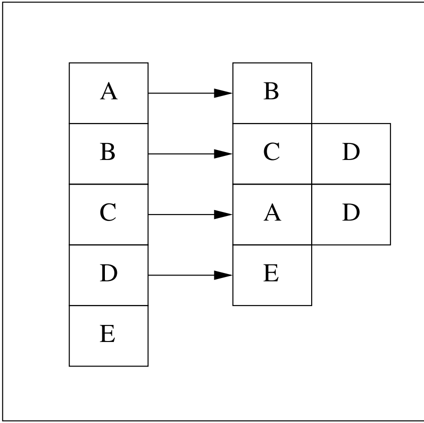
	B	C	D	E
A	1	1	0	0
B		1	1	0
C			1	0
D				1

Adjacency Lists

- An adjacency list gives each vertex an attribute which is a list of all the vertices adjacent to it.
- To represent a sparse graph, an adjacency list is more economical, since it only indicates where the edges are, not where they aren't.
- An adjacency list uses time $\Theta(\mathtt{V} * \mathtt{E})$ to find all edges.

Adjacency Lists





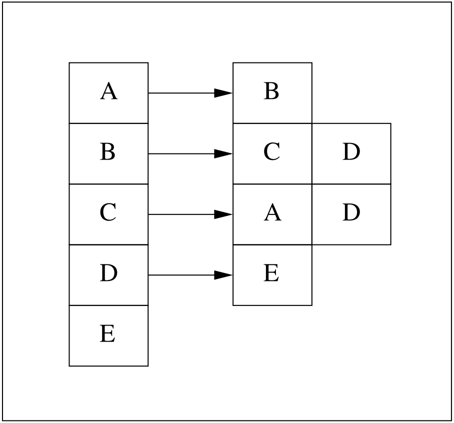
Adjacency Lists

Implementation of adjacency lists in Python:

- A list of lists.
- A dictionary.

Adjacency Lists: using Python list

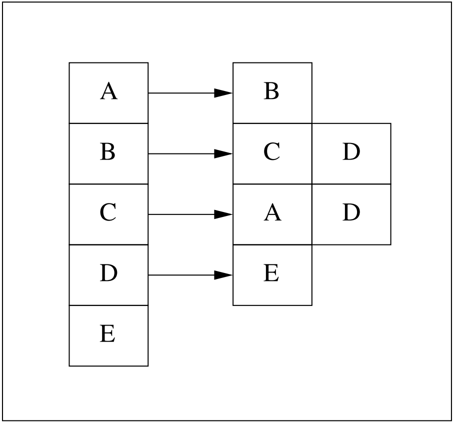
Let's assume that the graph is weighted, and the weight of each edge is 1, then using Python list we can have the following:




```
g = [
['A',[('B',1)]],
['B',[('C',1),('D',1)]],
['C',[('A',1),('D',1)]],
['D',[('E',1)]],
['E',[]]]
```

Adjacency Lists: using Python dictionary

Let's assume that the graph is weighted, and the weight of each edge is 1, then using Python dictionary:




```
g = {
'A':{'B':1},
'B':{'C':1,'D':1},
'C':{'A':1,'D':1},
'D':{'E':1},
'E':{}}
```

Adjacency Lists: C++

Implementation of adjacency lists in C++:

- For static graphs (do not change): a two-dimensional array.
- For dynamic graphs: a list of lists (linked-list implementation).

Adjacency Matrix vs Adjacency List

Adjacency Matrix vs Adjacency List

- graph is dense → the adjacency matrix representation is preferred.
- graph is sparse → the adjacency list representation is preferred.
- Most graphs in real-world applications are sparse, hence the adjacency list representation is more commonly used.
- Using matrix representation to find all edges from a vertex we will need to examine V entries,
- Using list representation to find all edges from a vertex we will need to examine only the actual edges originating from the vertex.

Adjacency Matrix vs Adjacency List

Adjacency Matrix vs Adjacency List

- graph is dense → the adjacency matrix representation is preferred.
- graph is sparse → the adjacency list representation is preferred.
- Most graphs in real-world applications are sparse, hence the adjacency list representation is more commonly used.
- Using matrix representation to find all edges from a vertex we will need to examine V entries,
- Using list representation to find all edges from a vertex we will need to examine only the actual edges originating from the vertex.

Adjacency Matrix vs Adjacency List

Adjacency Matrix vs Adjacency List

- graph is dense → the adjacency matrix representation is preferred.
- graph is sparse → the adjacency list representation is preferred.
- Most graphs in real-world applications are sparse, hence the adjacency list representation is more commonly used.
- Using matrix representation to find all edges from a vertex we will need to examine V entries,
- Using list representation to find all edges from a vertex we will need to examine only the actual edges originating from the vertex.

Adjacency Matrix vs Adjacency List

Adjacency Matrix vs Adjacency List

- graph is dense → the adjacency matrix representation is preferred.
- graph is sparse → the adjacency list representation is preferred.
- Most graphs in real-world applications are sparse, hence the adjacency list representation is more commonly used.
- Using matrix representation to find all edges from a vertex we will need to examine `V` entries,
- Using list representation to find all edges from a vertex we will need to examine only the actual edges originating from the vertex.

Adjacency Matrix vs Adjacency List

Adjacency Matrix vs Adjacency List

- graph is dense → the adjacency matrix representation is preferred.
- graph is sparse → the adjacency list representation is preferred.
- Most graphs in real-world applications are sparse, hence the adjacency list representation is more commonly used.
- Using matrix representation to find all edges from a vertex we will need to examine `V` entries,
- Using list representation to find all edges from a vertex we will need to examine only the actual edges originating from the vertex.