CPE 112 Programming with Data Structures

Graph

Dr. Piyanit Ua-areemitr
Dr. Taweechai Nuntawisuttiwong

Contents

Graphs

Graphs

A graph is an abstract data structure that is used to implement the mathematical concept of graphs.

Why are Graphs Useful?

Graphs are widely used to model any situation where entities or things are related to each other in pairs.

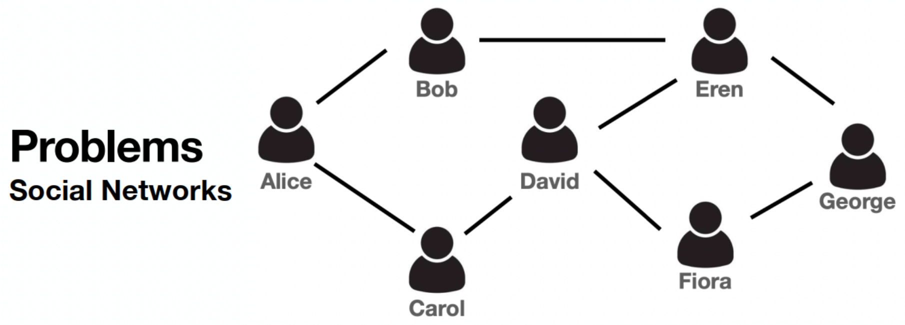
Examples

Problems

Transportation

From CPE to Library

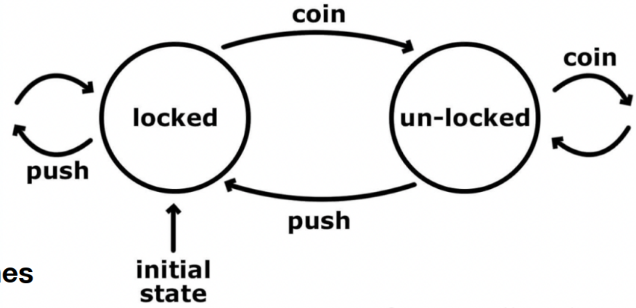
Examples


Problems
Social Networks
Bob
Eren
Alice
David
George
Carol
Fiora

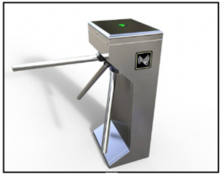
Examples

Problems

Finite State Machines (FSM)



Source: www.smashingmagazine.com



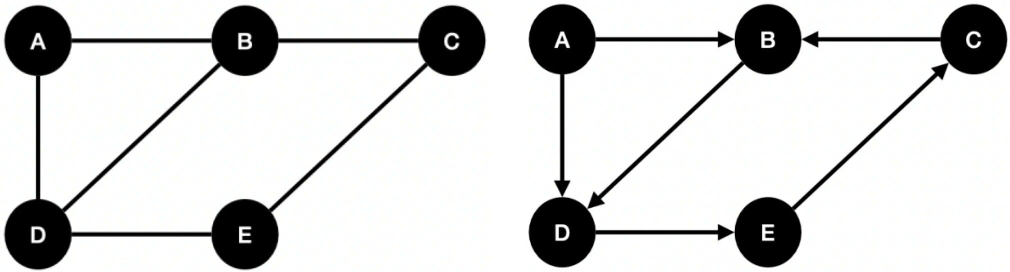
Source: www.alibaba.com

Graph

Definition

A graph G is defined as an ordered set (V, E), where

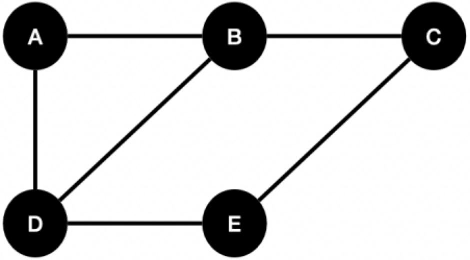
- $V(G)$ represents the set of vertices,
- $E(G)$ represents the edges that connect these vertices.



Graph Terminology

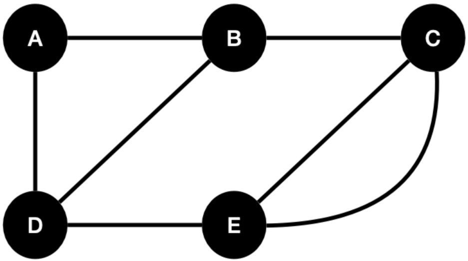
Node

- Adjacent nodes or neighbors:
  $e = (u, v)$
- Degree of a node: $deg(A) = 2$
  - Isolate node: $deg(u) = 0$
  - A k-regular graph if every node has the same degree.



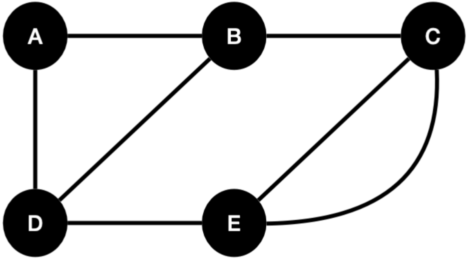
Edge

- Multiple edges: $e_1 = (u, v)$ & $e_2 = (u, v)$
- Loop: $e = (u, u)$



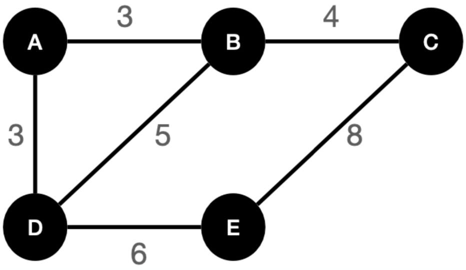
Path

- Path $P = (v_0, v_1, \ldots, v_n)$ - a sequence of vertices
- Path length: number of edges
- Closed path: $v_0 = v_n$
- Simple path: all nodes are distinct except the closed simple path



Weighted Graph (Labelled Graph)

- Size of a graph: number of edges
- Connected graph: for any two vertices, there is a path from u to v
- Completed graph: all nodes are adjacent nodes



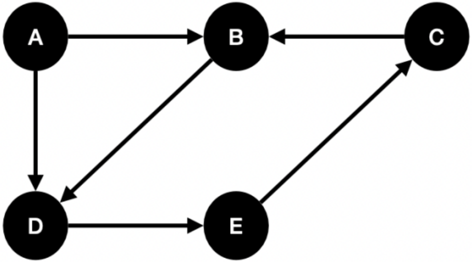
Directed Graphs

Directed Graphs

A directed graph is a graph in which every edge has a direction assigned to it.

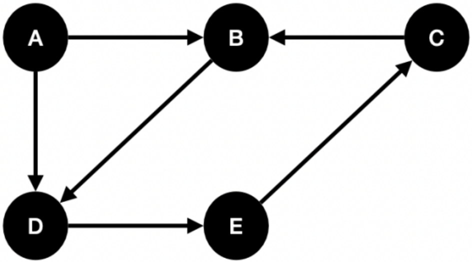
For an edge $e(u, v)$,

- The edge begins at u and terminates at v.
- u is knows as the origin or initial point of e.
- v is known as the destination or terminal point of e.
- Nodes u and v are adjacent to each other.



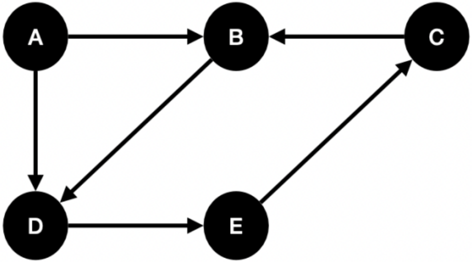
Terminology of a Directed Graph

- Out-degree: $outdeg(B) = 1$
- In-Degree: $indeg(B) = 2$
- Degree:
  $def(B) = indeg(B) + outDeg(B) = 3$



Terminology of a Directed Graph

- Source: Positive out-degree but zero in-degree
- Sink: Positive in-degree but zero out-dgree
- Strongly connected: There exists a path between every pair of nodes in graph.
- Weakly connected: A digraph is connected by ignoring the direction of edges.

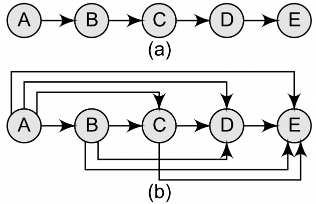


Transitive Closure of a Directed Graph

A transitive clousure of a graph is contructed to answer reachability questions.

Definition

For a directed graph $G(V, E)$, the transitive closure of G is a graph $G^* = (V, E^*)$. In G^*, for every vertex pair v, w in V where is an edge (v, w) in E^* if and only if there is a valid path from v to w in G.



Where and Why is Transitive Closure Needed?

- Transitive closure is used to find the reachability analysis of transition networks representing distributed and parallel systems.
- It is used in the construction of parsing automata in compiler construction.
- Transitive closure computation is being used to evaluate recursive database queries.

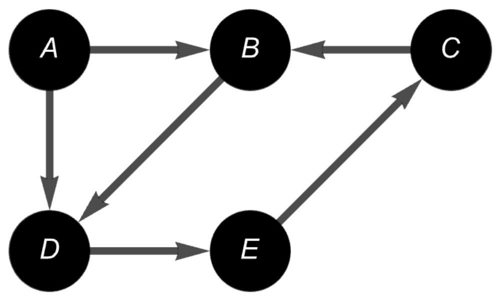
Representation of Graphs

Representation of Graphs

- Sequential representation: adjacency matrix
- Linked representation: adjacency list
- Adjacency multi-list: extension of linked representation

Adjacency Matrix

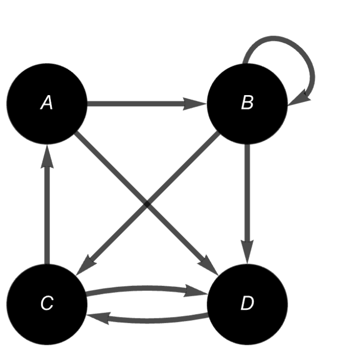
Directed graph



$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

Adjacency Matrix

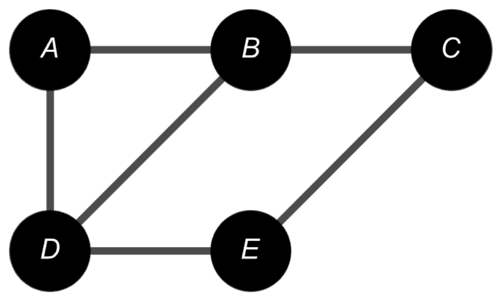
Directed graph with loop



$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Adjacency Matrix

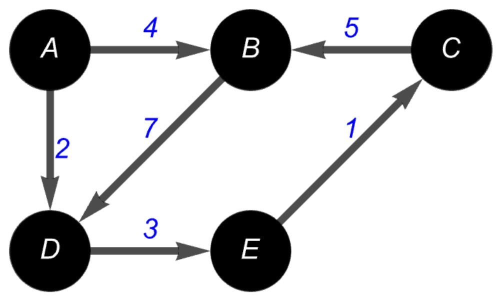
Undirected graph



$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{bmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

Adjacency Matrix

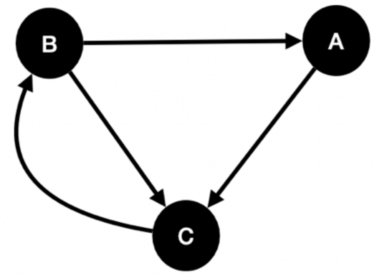
Weighted graph



$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{bmatrix}
0 & 4 & 0 & 2 & 0 \\
0 & 0 & 0 & 7 & 0 \\
0 & 5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & 0
\end{bmatrix}
\end{array}
$$

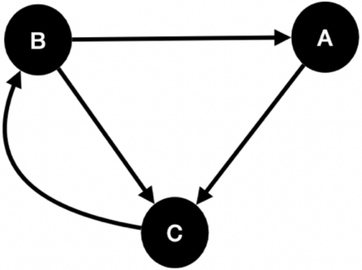
Path Matrix

For a matrix M^k, each element $m_{ij} = n$ where n is number of the paths (length = k) from v_i to v_j.



$$M = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Path Matrix



$$M = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Path Matrix

Transitive Closure

$$M_L = M^1 + M^2 + \ldots + M^k \text{ or } M_P = M^1 \wedge M^2 \wedge \ldots \wedge M^k$$

$$ML = M^1 + M^2 + M^3 = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 3 \\ 1 & 2 & 2 \end{bmatrix}$$

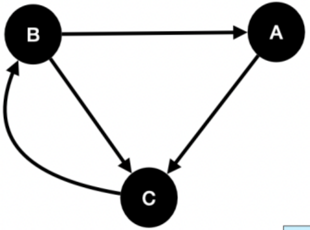
$$ML = M^1 \wedge M^2 \wedge M^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Adjacency Matrix

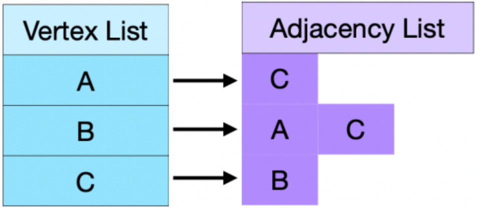
Summary

- Easy to understand.
- Handle a fixed number of vertices.
- Could take more memory than other methods because there are cells in the matrix for every possible combination of vertices.

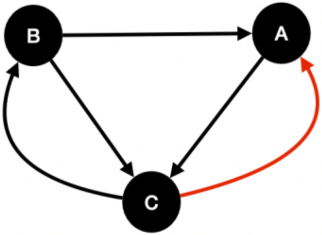
Adjacency List



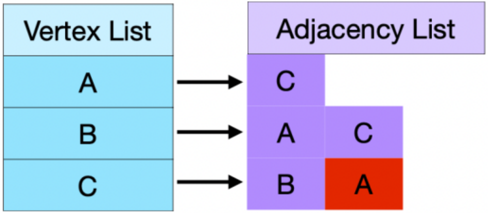
	A	B	C
A	0	0	1
B	1	0	1
C	0	1	0



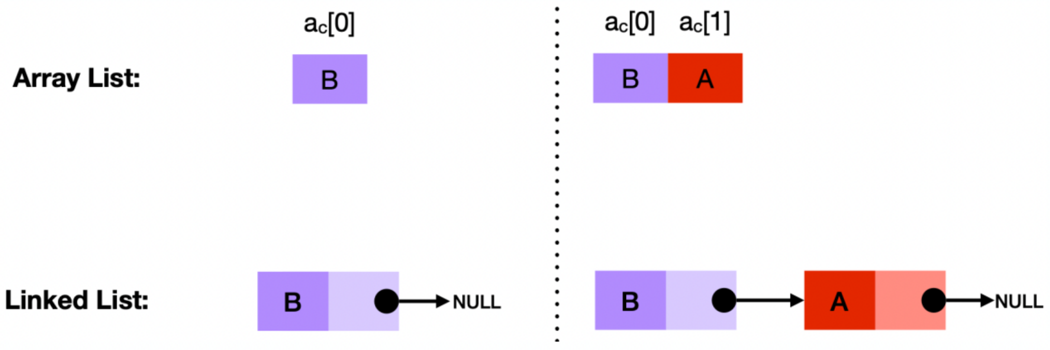
Add New Edge



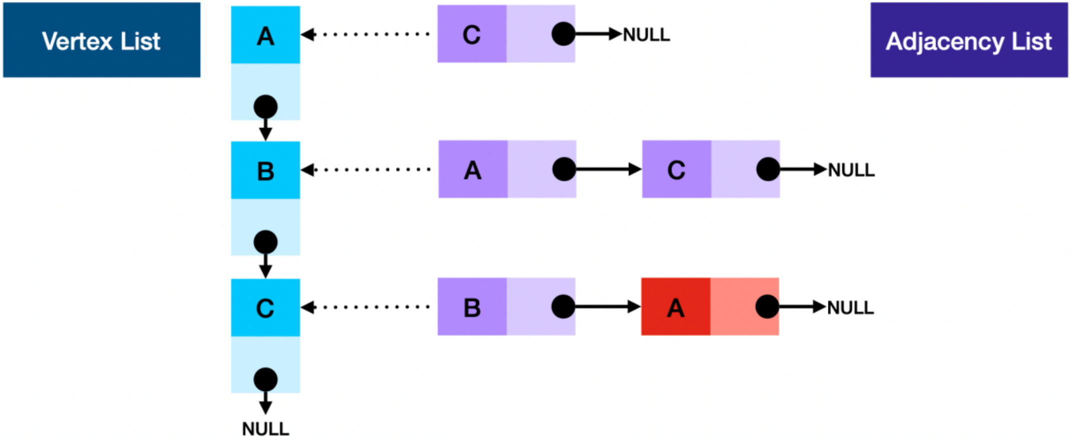
	A	B	C
A	0	0	1
B	1	0	1
C	1	1	0



Array vs Linked List



Adjacency List Using Linked List



Operations

- Add vertex
- Add edge
- Remove vertex
- Remove edge
- Get adjacent
- Print graph

Adjacency List

Advantages

- It is easy to follow and clearly shows the adjacent nodes of a particular node.
- It is often used for storing graphs that have a small-to-moderate number of edges.
- Adding new nodes in G is easy and straightforward when G is represented using and adjacency list.

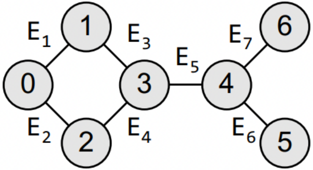
Adjacency Multi-list Representation

- Modified version of adjacency lists.
- An edge-based representation of graphs.
- Consists of two parts – node's information and edges' information.

Attributes of edge(v_i,v_j)

- M: A single bit field to indicate whether the edge has been examined or not.
- v_i: A vertex in the graph that is connected to vertex v_j by an edge.
- v_j: A vertex in the graph that is connected to vertex v_i by an edge.
- Link i for v_i: A link that points to another node that has an edge incident on v_i.
- Link j for v_j: A link that points to another node that has an edge incident on v_j.

Adjacency Multi-list Representation



VERTEX	LIST OF EDGES
0	Edge 1, Edge 2
1	Edge 1, Edge 3
2	Edge 2, Edge 4
3	Edge 3, Edge 4, Edge 5
4	Edge 5, Edge 6, Edge 7
5	Edge 6
6	Edge 7

Edge 1		0	1	Edge 2	Edge 3
Edge 2		0	2	NULL	Edge 4
Edge 3		1	3	NULL	Edge 4
Edge 4		2	3	NULL	Edge 5
Edge 5		3	4	NULL	Edge 6
Edge 6		4	5	Edge 7	NULL
Edge 7		4	6	NULL	NULL