Week 15
Graph

Team Teaching for Data Structures and Algorithm

Learning Objective

Students are able to understand the definition of Graph and its terminologies

Students are able to model the real world problems using Graph

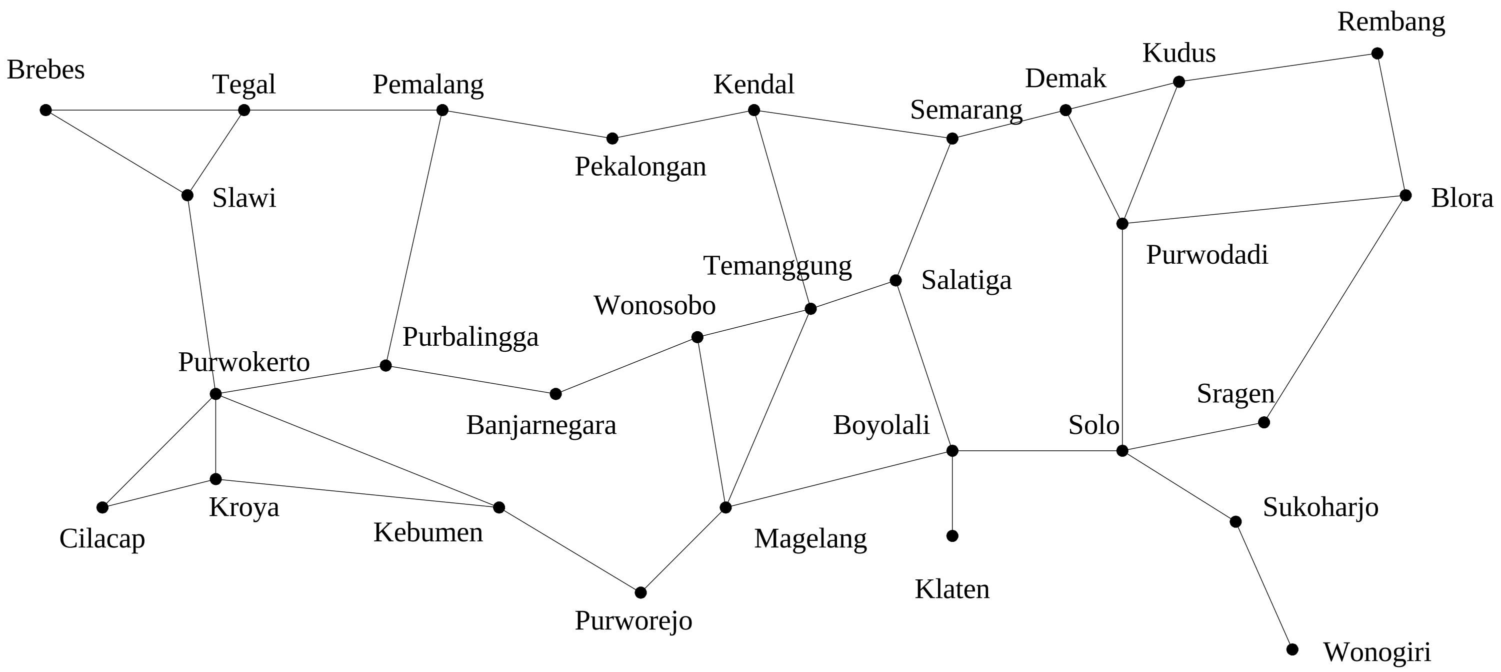
Students are able to map and explain Graph Data Structure

Outlines

- Graph and its history
- Terms used in Graph
- Example of Graph implementation
- Graph Representation

Graph

- Graph is used to represent discrete objects and its relation
- Following image represents the roads and distance among the cities in Central Java

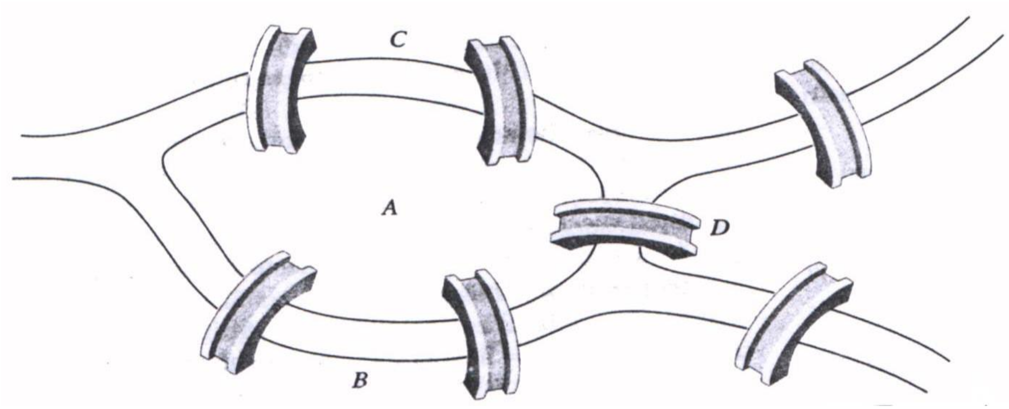


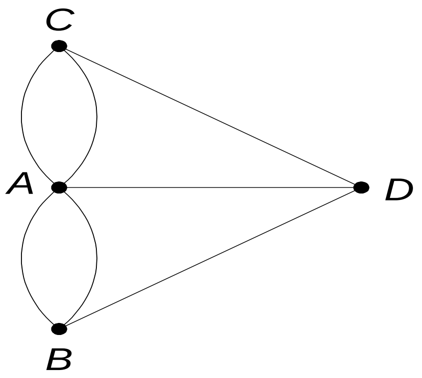
History of Graph

- Königsberg bridge in 1973 M
- Graph that represents Königsberg bridge:

  vertex (points) → represents lands

  edge (edge/ lines) → represents bridges

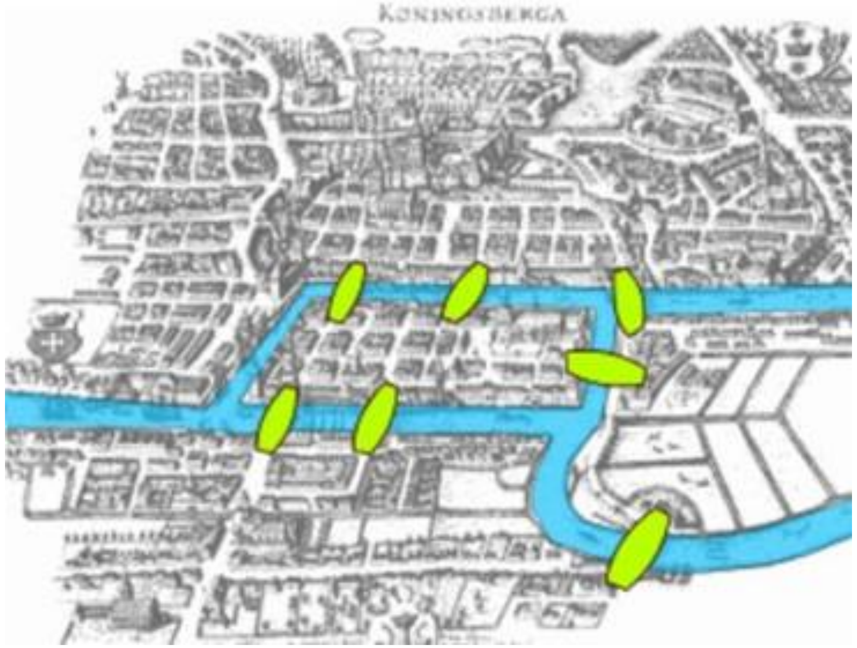




Can we go through each bridge only once and still go back to initial point ?

Graph HISTORY

- 7 bridges of Königsberg (1736)
- Königsberg was a city in Germany, and the city was built around a river called the Pregel River



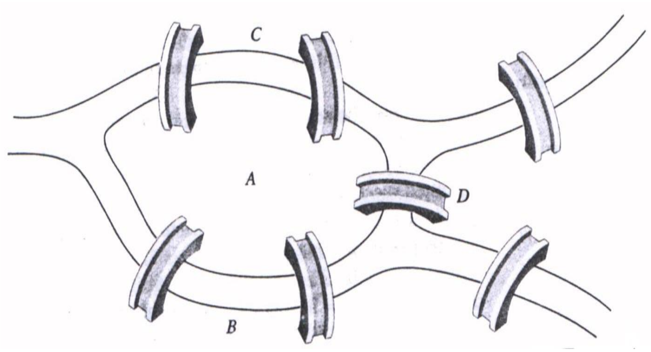
Graph HISTORY

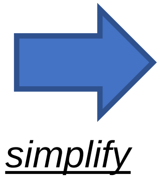
- The citizens of Königsberg spent their Sundays walking around town, enjoying their beautiful city
- **Challenge** → walk across all of the seven bridges crossing the islands only once, without ever repeating a single bridge in the course of one's walk.

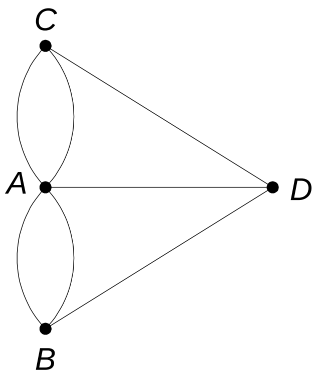
Graph HISTORY

- Finally, Leonhard Euler can solve the problem, by using a rule later called as **Euler Path**
- ***Euler path*** *is a path wherein we only visit each edge in the graph once, while* ***Euler Circuit*** *is a Eulerian path that is a cycle — we only visit every edge once, and we end on the exact same node that we started off on.*
- *Euler Path basic rule:*
  - The number of **vertices of odd degree** must be either **zero** or **two**.
  - And if there are two vertices with odd degree, then they are the starting and ending vertices.

Graph HISTORY







Vertex	Degree
A	5
B	3
C	3
D	3

Since there are 4 vertices with odd degree, then it is not Euler Path

Definition of Graph

Graph G = (V, E) is a system that has unlimited set of non-empty V(G) and set of E(G) (potentially empty) that each of its element is a pair of unordered set from different elements of V(G)

Graph G = (V, E), in which in this case is:

V = non-empty set from *vertices*

= {a, b, … , vn }

E = set of (*edges*) that connects the vertices

= $\{e_1 , e_2, \ldots , e_n\}$ atau {(a,b) (a,c) (n, n)}

Terms used in a Graph

- ***Vertex / points***

  Dots in *graph* also known as *vertex*. Usually symbolized with circle.

- ***Edge* (*Lines or sides or corners*)**

  Line connectors that unites all the vertex inside the graph is called (*edge*)

- ***Adjacency***

  2 Vertexes is *adjacent* if it is connected with one line / (*edge*).

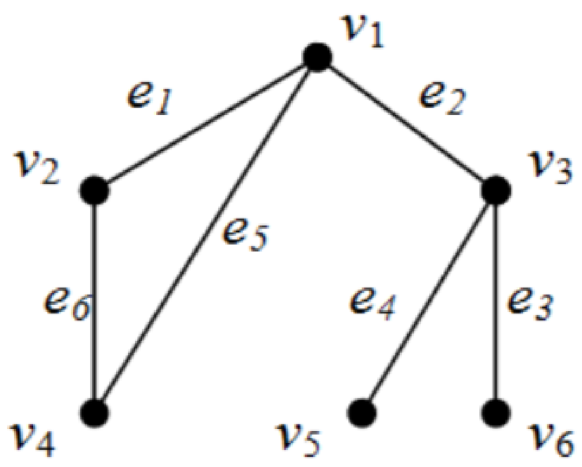
- ***Path***

  Path represents of a way from one point to another

Example

Graph G
v1, v2, v3, v4, v5, v6 are *vertices*
e_1, e_2, e_3, e_4, e_5, e_6 are *lines* or *edges*
v1 is adjacent with *v2*, *v3* and *v4*
v2 is not adjacent with *v3*, *v5* and *v6*
Path from v4 to v6 are v4 → v2 → v1 → v3 → v6
Another path from v4 to v6 are v4 → v1 → v3 → v6

The shortest path indicates how close are the points to each other that is connected with a line

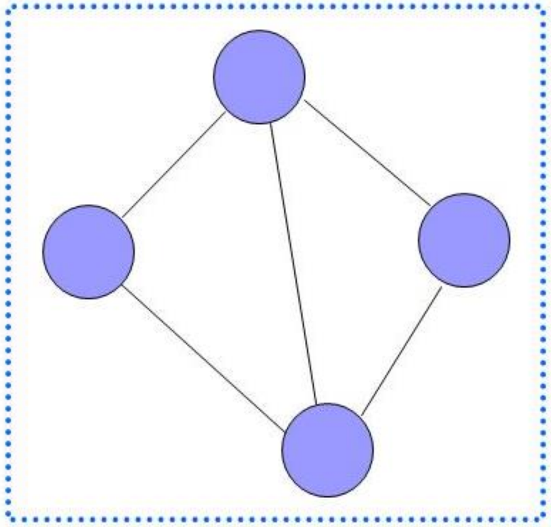


Graf G

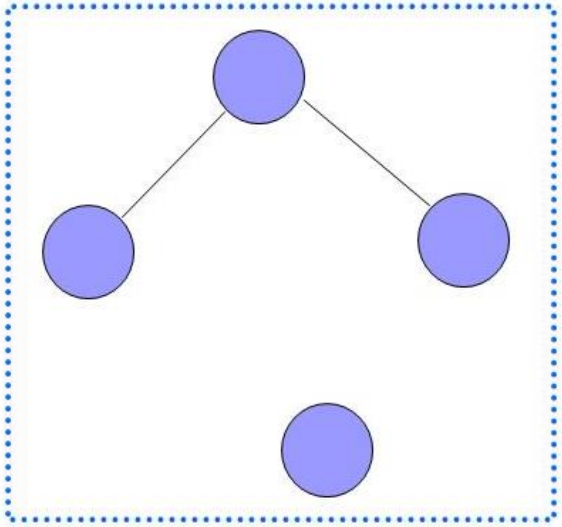
Terms used in a Graph

- ***Connected***

  A *connected* graph will require at least one edge that connects one vertex to each other. This image is an example of *connected graph*. While for the unconnected graph, the vertexes are not connected as a whole



Connected Graph

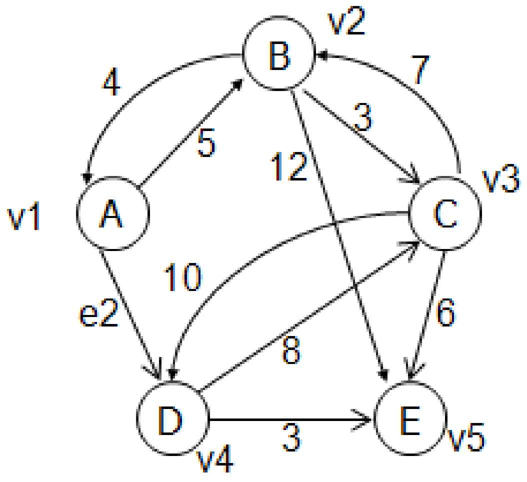


Unconnected Graph

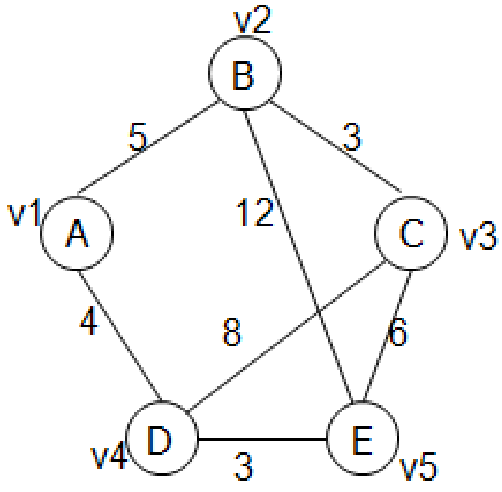
Terms used in a Graph

- ***Directed Graph* and *Weighted Graph***

  Directed and *weighted Graph* is a *graph* with a line that connects the vertexes that has direction and weight



Directed graph



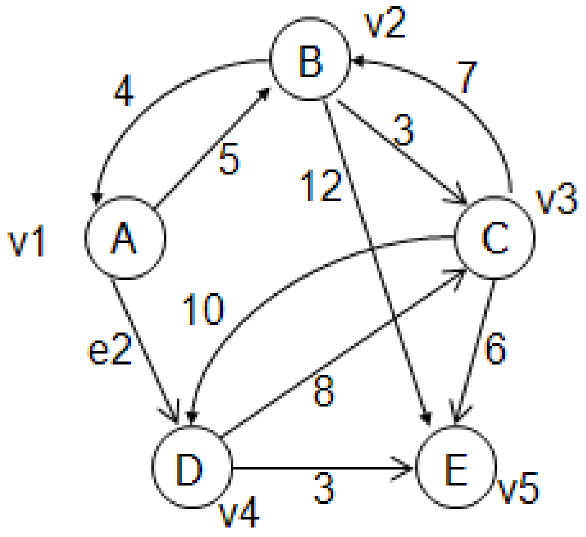
Undirected graph

Terms used in a Graph

- ***Degree, in-degree*** **and** ***out-degree***

  Degree of a vertex is the number of lines are connected to that vertex

  - ***In-degree*** is a vertex in a directed graph that has some path refers **to** the vertex itself
  - ***Out-degree*** is a vertex in a directed graph that has some path refers **from** that vertex
  - Notated as d(v)



Directed graph

D_{in} (A) = 1
D_{out}(A) = 2

Graph Representation

- **Adjacency list**

  Adjacency list uses an array in linked list. This array will be used to store the vertex amount. The value of the linked list wil be used to store graph’s weight.

- **Adjacency matrix**

  Adjacency matrix is an 2D array with size V x V. Which V are the vertex’s amount of a graph. If adj[i][j] = 1, then it means that there is a line / edge from point i to point j.

Adjacency list undirected graph

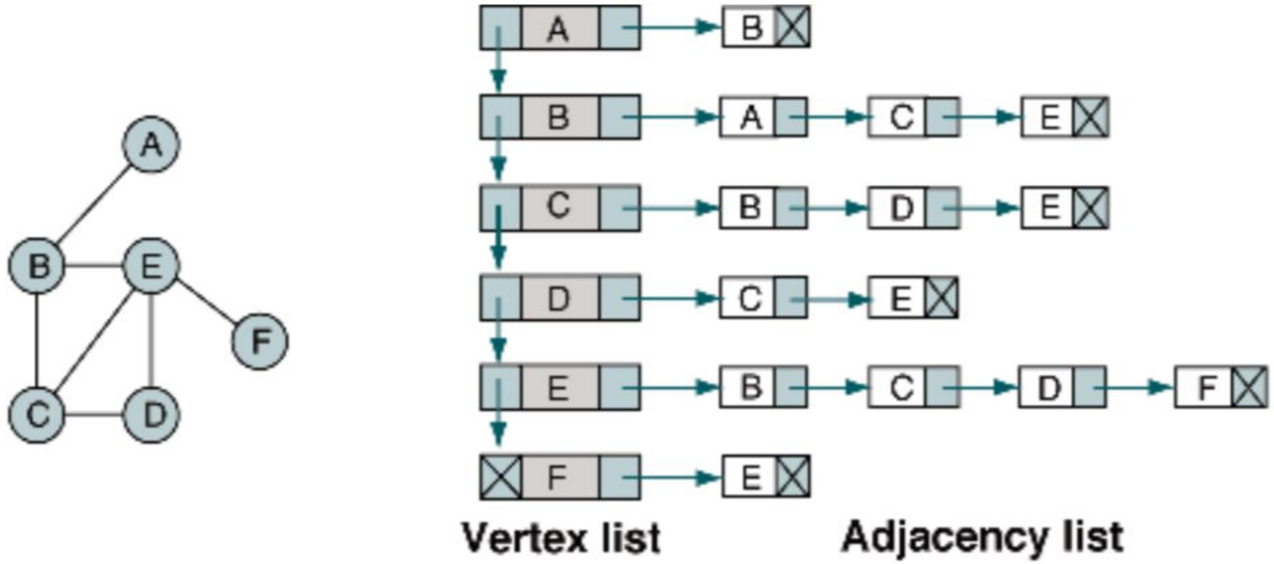
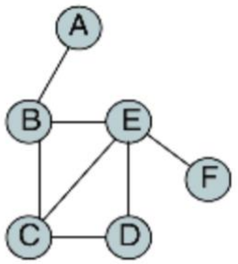


FIGURE 11-14 Adjacency List

Graph and *matrix adjacency directed graph*



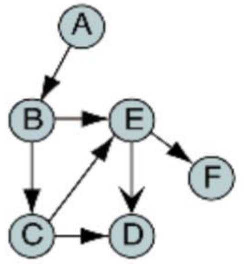
A
B
C
D
E
F

Vertex vector

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

Adjacency matrix

(a) Adjacency matrix for nondirected graph



A
B
C
D
E
F

Vertex vector

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

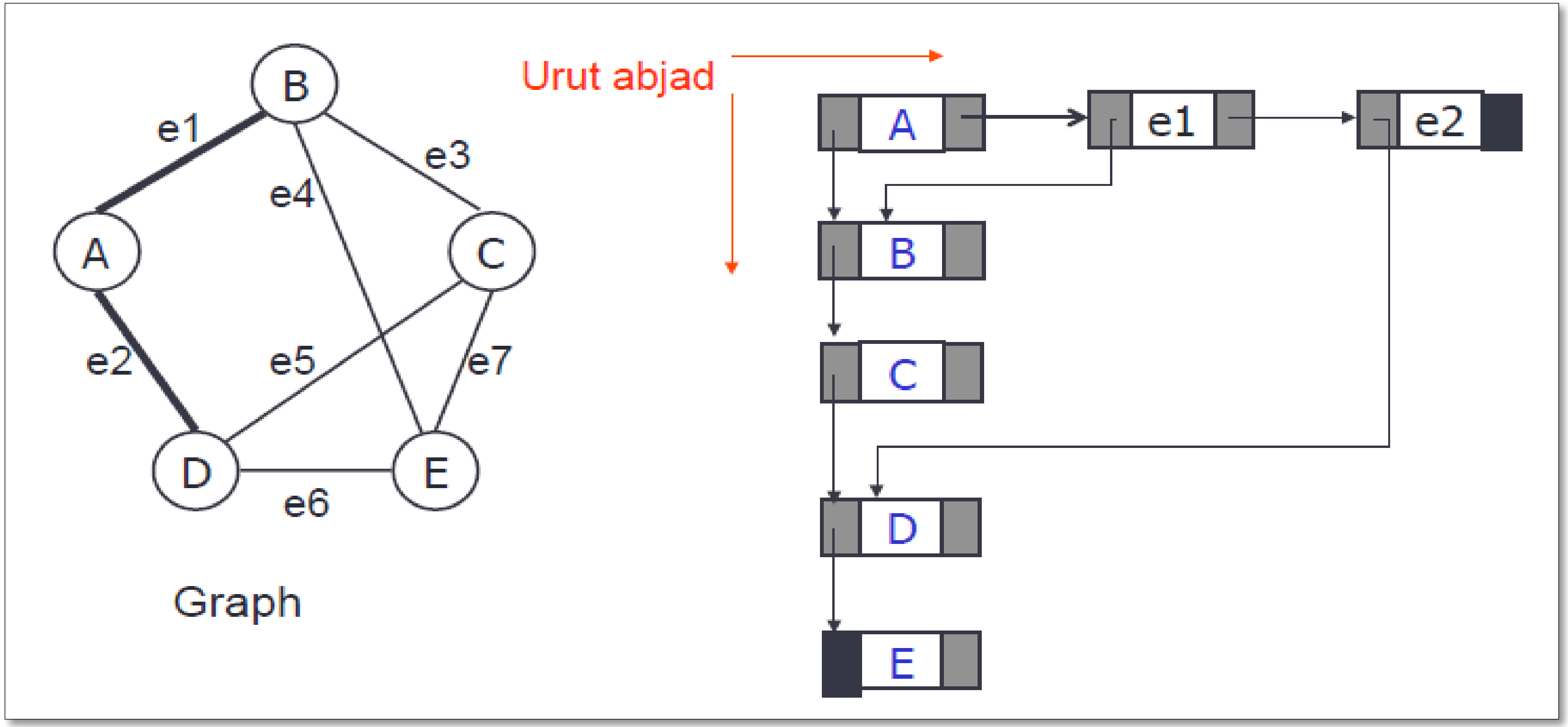
Adjacency matrix

(b) Adjacency matrix for directed graph

FIGURE 11-13 Adjacency Matrix

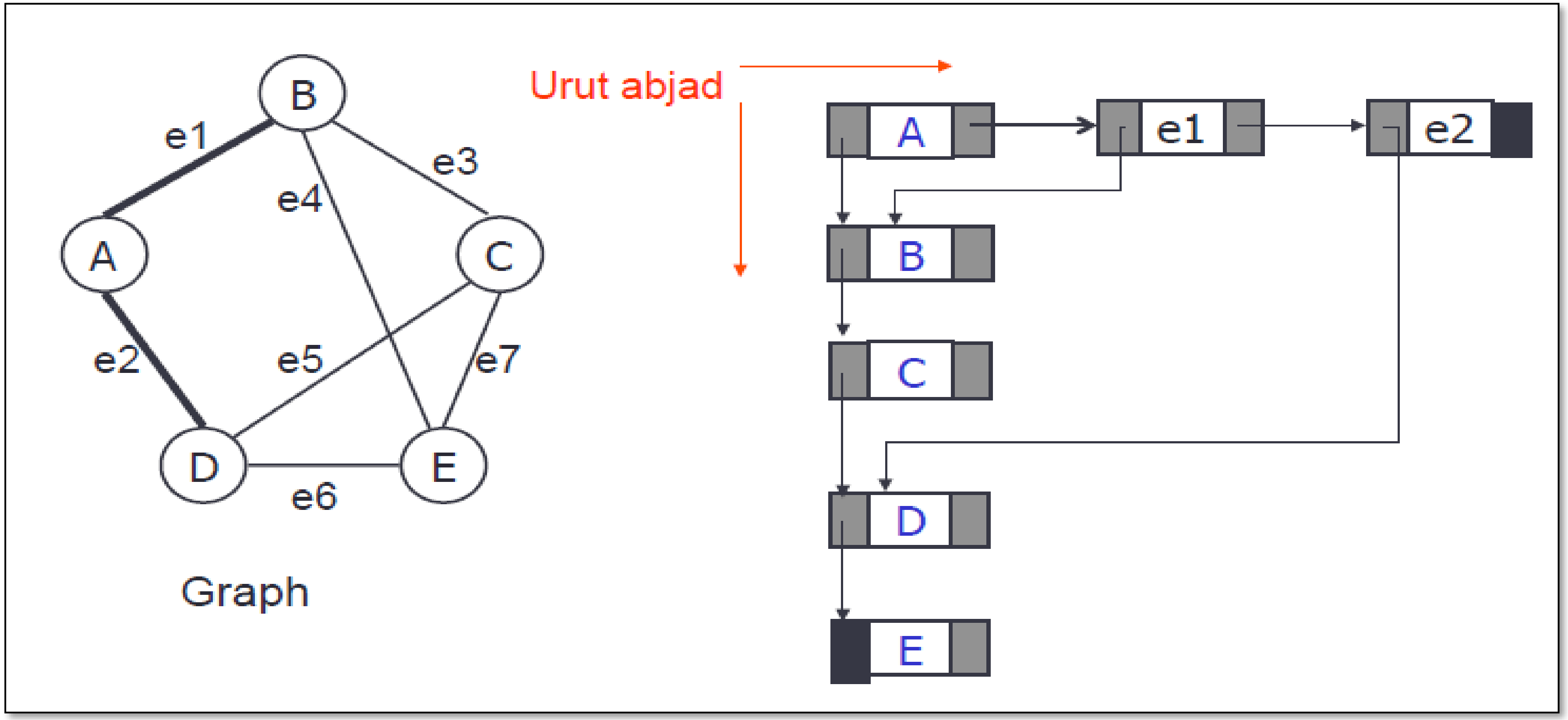
Graph Representation in Linked List

- Adjency List graph undirected/ directed
- Illustrated below is a vertex that has 2 pointers (vertex pointer and line pointer)



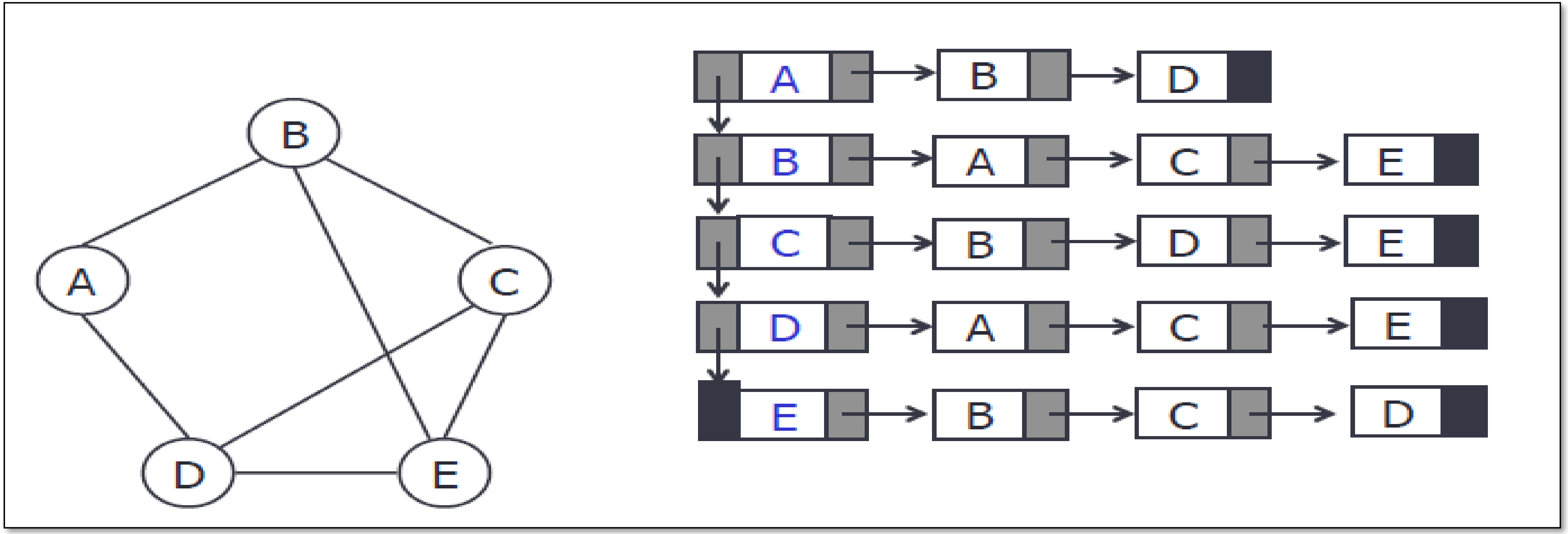
Example(1)-Adjacency Undirected Graph

- For vertex A, it has 2 line that connect e_1 and e_2

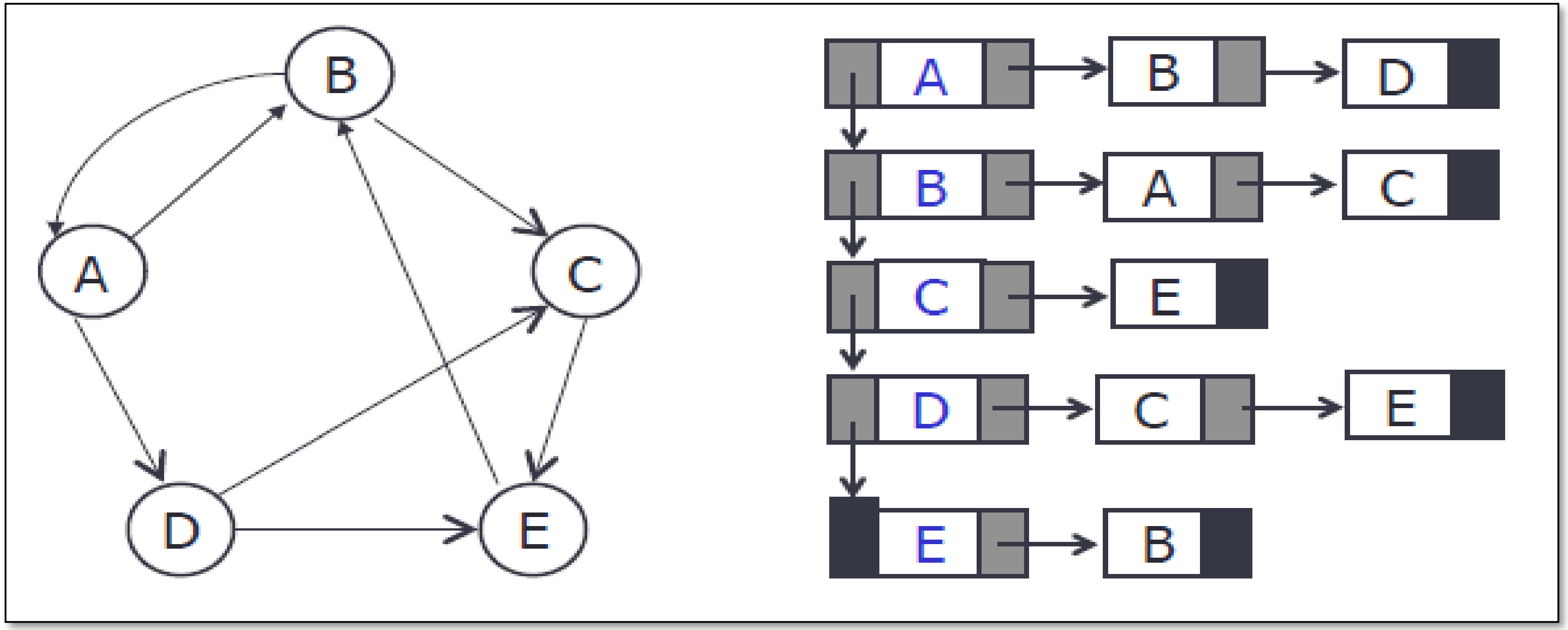


Example(1)

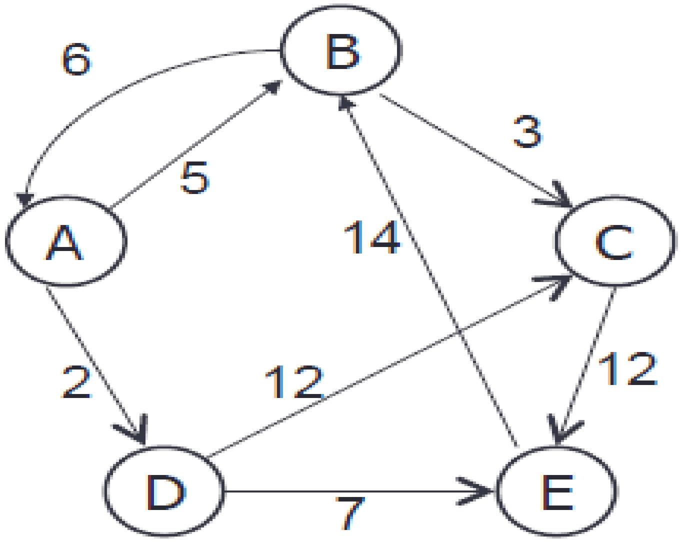
- Simpler form of previous example



Example (2)-Adjacency Directed Graph



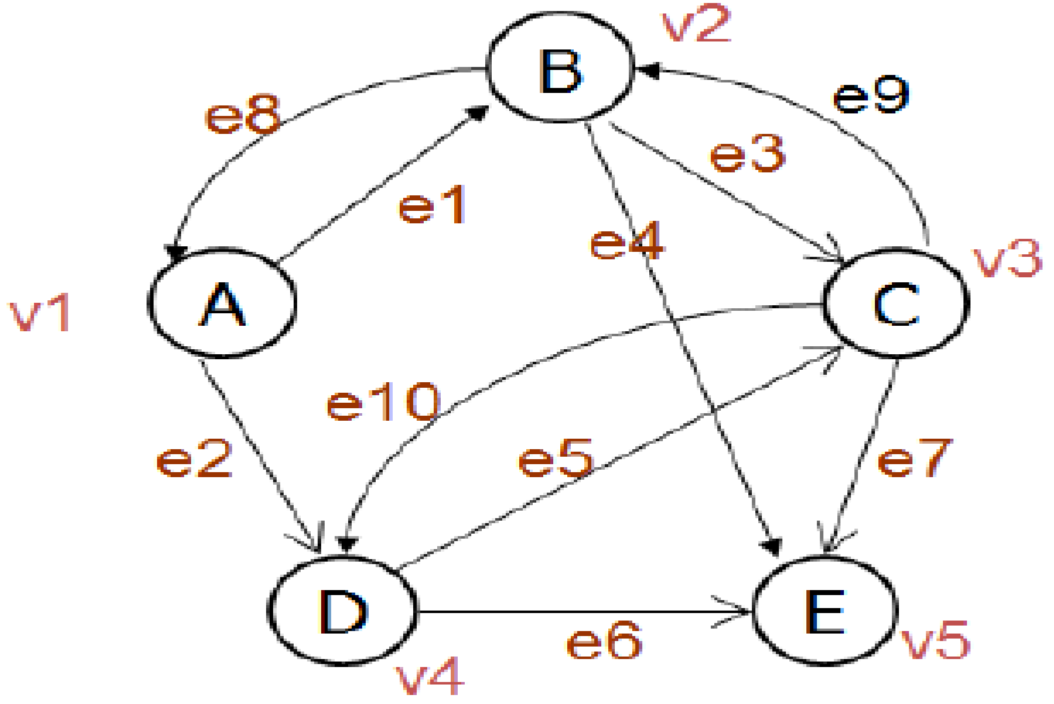
Example(3)-Directed and Weighted Graph



	A 0	B 1	C 2	D 3	E 4
A 0	0	5	0	2	0
B 1	6	0	3	0	0
C 2	0	0	0	0	12
D 3	0	0	12	0	7
E 4	0	14	0	0	0

Practicum 1

- Change this following Graph to matrix



Practicum 2

- Convert this matrix to Graph

	V1	V2	V3	V4	V5	V6
V1	0	1	0	0	0	0
V2	1	1	1	0	0	0
V3	0	1	0	1	1	1
V4	0	0	1	0	0	0
V5	0	0	1	0	0	0
V6	0	0	1	0	0	0

Practicum 3

- Convert this matrix to Graph

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
V1	1	1	0	1	1	0	0	0
V2	1	0	1	0	0	0	0	0
V3	0	1	1	0	0	1	1	0
V4	0	0	0	1	0	1	0	1
V5	0	0	0	0	0	0	0	1