CHAPTER 6

GRAPHS

GRAPHS

6.1 The Graph Abstract Data Type

6.2 Elementary Graph Operations

6.3 Minimum Cost Spanning Trees

6.1.1 Introduction

- Königsberg bridge problem

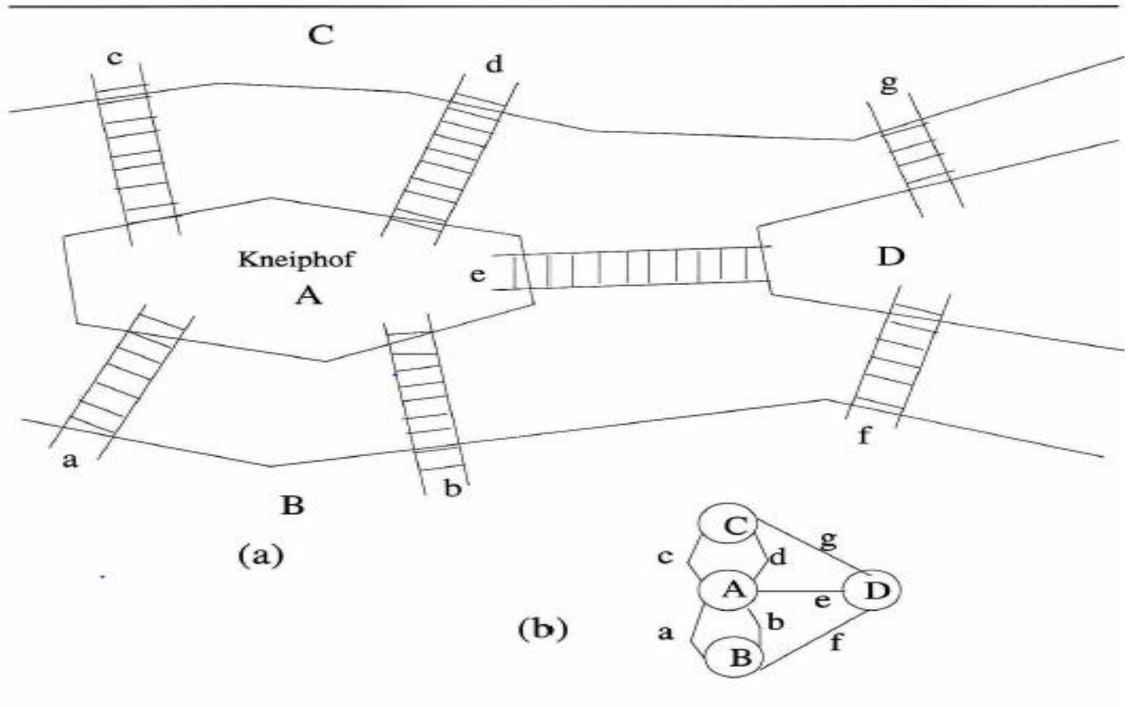
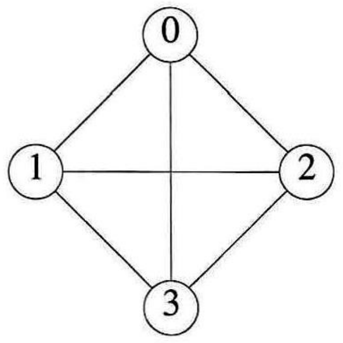


Figure 6.1: (a) Section of the river Pregel in Königsberg; (b) Euler's graph

6.1.2 Definitions

- Graph G=(V, E)
  - V is a finite, nonempty set of *vertices*
  - E is a set of *edges*
  - An *edge* is a pair of vertices
  - V(G) is the set of vertices of G
  - E(G) is the set of edges of G

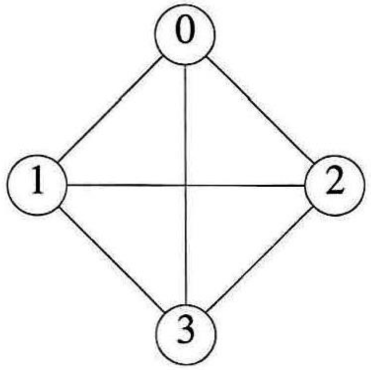


- Undirected/Directed Graph

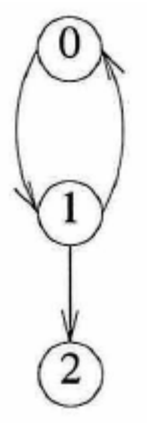
- Undirected graph
  - The pair of vertices in an edge is unorderec
    - (u,v) and (v,u) : the same edge
  - Ex)

    $V(G_1)=\{0,1,2,3\}$

    $E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$



- Directed graph (**digraph**)
  - Each edge is represented by $<u,v>$
    - u: tail , v: head $\qquad$ $\textit{tail} \longrightarrow \textit{head}$
  - $< v, u >$ and $< u, v >$ represent two different edges
  - Ex)
    $V(G_3) = \{0,1,2\}$
    $E(G_3) = \{<0,1>, <1,0>, <1,2>\}$


(c) G_3

tree 모양의 그래프

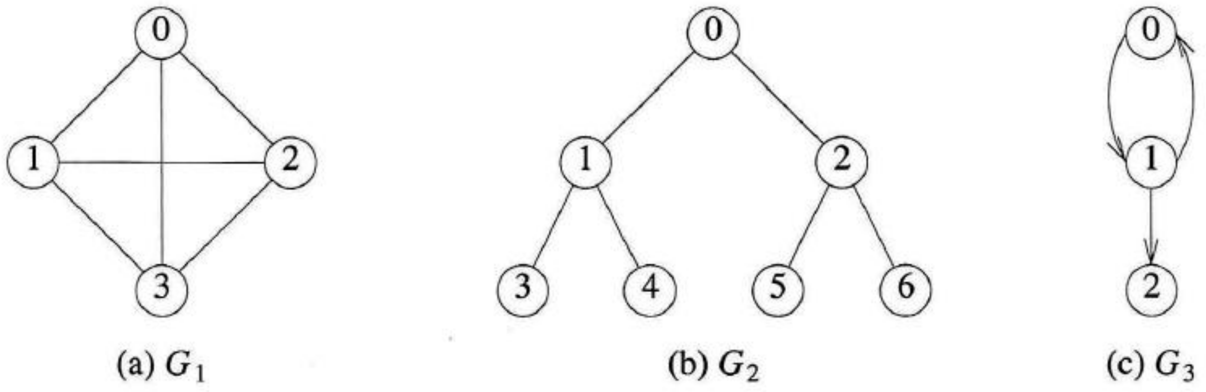


Figure 6.2: Three sample graphs

- $V(G_1) = \{0,1,2,3\}$
  $E(G_1) = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$
- $V(G_2) = \{0,1,2,3,4,5,6\}$
  $E(G_2) = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$
- $V(G_3) = \{0,1,2\}$
  $E(G_3) = \{ <0,1>,<1,0>,<1,2> \}$

- Restrictions on Graphs 이런 것들은 고려하지 않는다
  - NOT *self edges* or *self loops*
    - (v, v) and <v, v> : Not legal
  - NOT multiple occurrences of the same edge

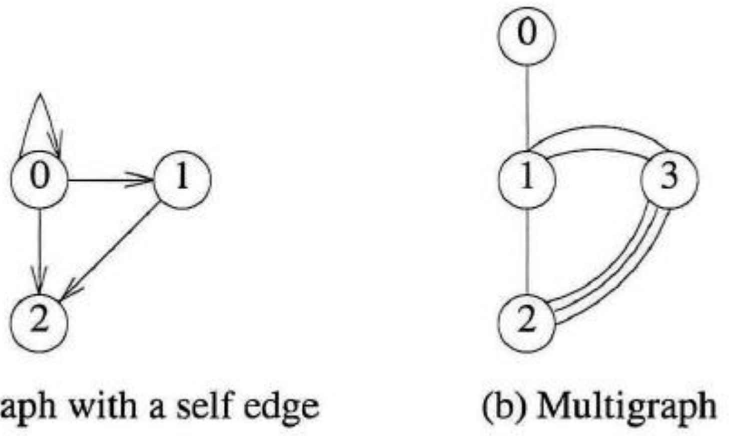
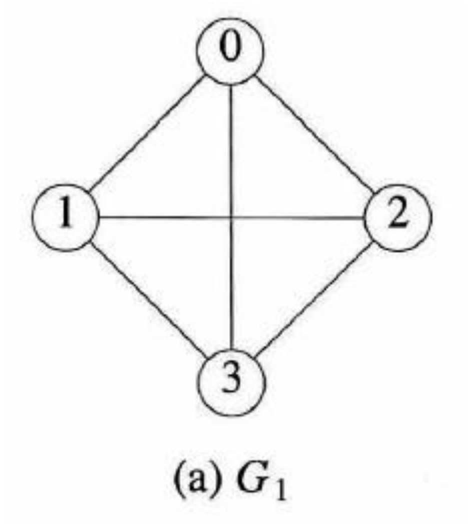
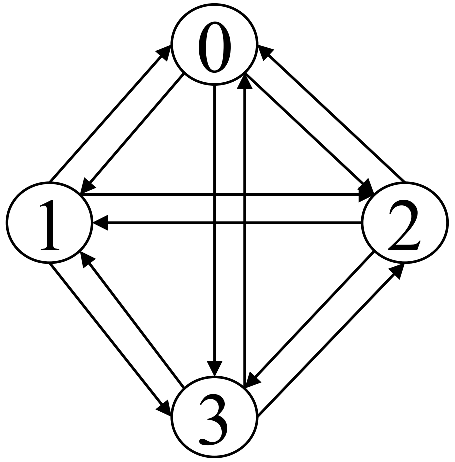


Figure 6.3: Examples of graphlike structures

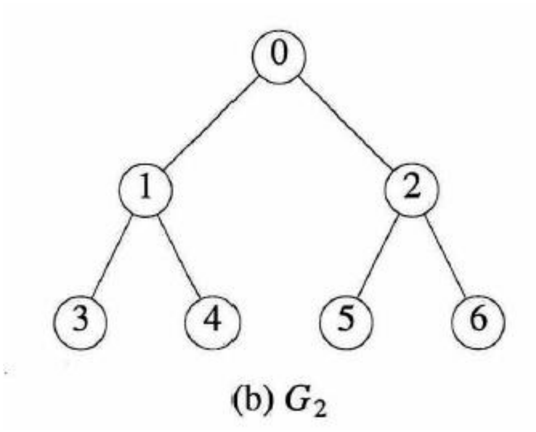
- Complete graph
  - Has the maximum number of edges;
  - An n-vertex, undirected graph with exactly $n(n-1)/2$ edges is said to be *complete*
- Complete directed graph
  - The max # of edges is $n(n-1)$



(a) G_1

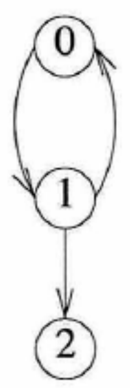


- If (u,v) is an edge in E(G)
  - Vertices u and v are *adjacent*
  - Edge (u, v) is *incident* on vertices u and v
  - Ex)
    - Edges incident in vertex 2 in G_2: …



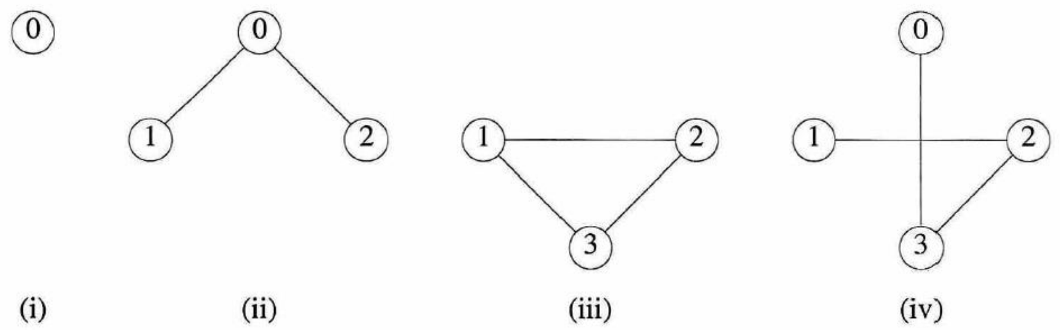
(b) G_2

- If $<u, v>$ is a directed edge
  - Vertex u is *adjacent* to v, and v is *adjacent* from u
  - Edge $<u,v>$ is incident on u and v.
  - Ex)
    - Edges incident to vertex 1 in G_3: …

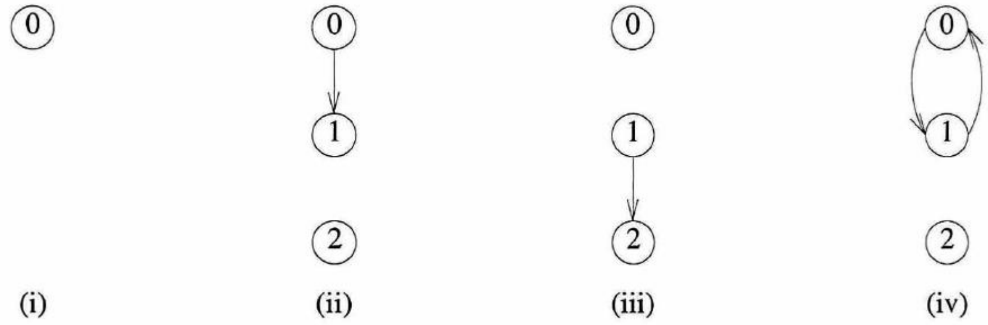


(c) G_3

- Subgraph of G
  - A graph G’ such that V(G’) $\subseteq$ V(G) and E(G’) $\subseteq$ E(G)

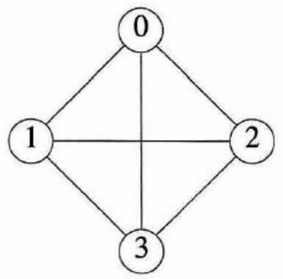


(a) Some of the subgraphs of G_1

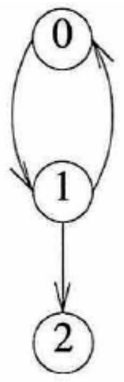


(b) Some of the subgraphs of G_3

Figure 6.4: Some subgraphs

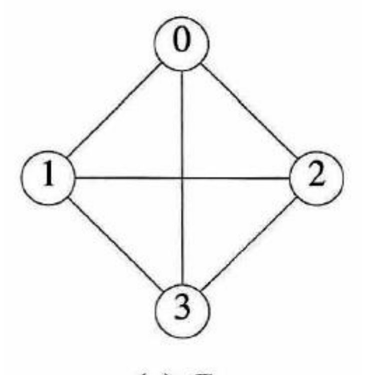


(a) G_1



(c) G_3

- A *path* from vertex *u* to vertex *v* in graph G
  - A sequence of vertices ***u*, $\boldsymbol{i_1}$, $\boldsymbol{i_2}$, ..., $\boldsymbol{i_k}$, *v*** such that (u,i_1), (i_1,i_2), ..., (i_k,v) are edges in E(G)
  - path (**0**, 2), (2, 1), (1, **3**) is also written as 0, 2, 1, 3



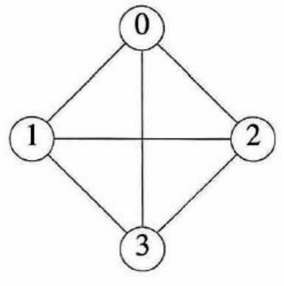
(a) G_1

- The *length* of a path
  - The number of edges on it

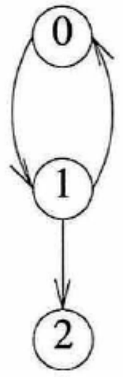
- Simple path

겹치는거 X

  – A path in which all vertices, except possibly the first and the last, are distinct
  – Ex) G_1
    - path 0, 1, 2, 0: simple path
    - path 0, 1, 3, 2: simple path
    - path 0, 1, 3, 1: NOT simple path
  – Ex) G_3
    - path 0, 1, 2: simple directed path

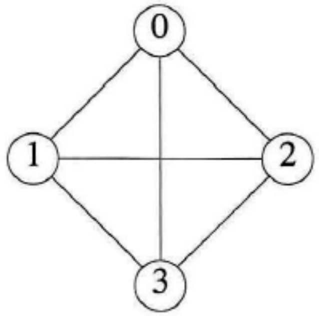


(a) G_1



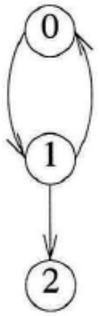
(c) G_3

- Cycle
  – A simple path in which the first and the last vertices are the same
  – Ex)
    - path 0, 1 ,2, 0 (G_1), path 0, 1, 0 (G_3)



(a) G_1

path : 0, 1, 3, 2	0, 1, 3, 1	0, 1, 2, 0
length : 3	3	3
simple path : O	X	O
cycle: X	X	O

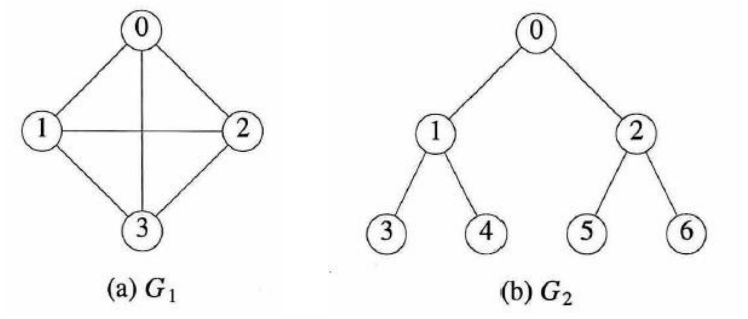


(c) G_3

0, 1, 0 - cycle
0, 1, 2 - simple *directed* path
0, 1, 2, 1 - not a path
2→1이 없음

- An undirected graph is said to be ***connected***
  - iff for every pair of distinct vertices *u* and *v* in V(G) there is a path from *u* to *v* in G At ∀u,v, u→v로 가는 Path가 있어야함
  - A *tree* is a connected acyclic (no cycles) graph

Cycle이 없는 그래프



- (Connected) *component* of an undirected graph
  - A *maximal* connected subgraph

Connected를 만드는 가장 큰 subgraph

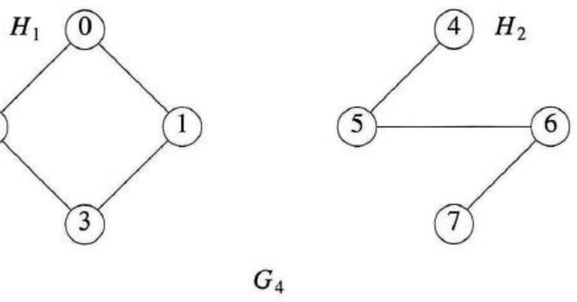
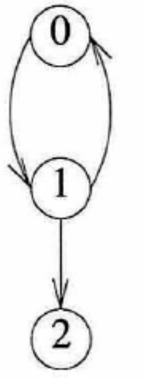


Figure 6.5: A graph with two connected components

- A directed graph is *strongly connected*
  - iff for every pair of distinct vertices *u* and *v* in V(G), there is a directed path from *u* to *v* and also from *v* to *u*
  - Ex) G_3 : … not strongly connected



(c) G_3

- Strongly connected component
  - A maximal subgraph that is strongly connected

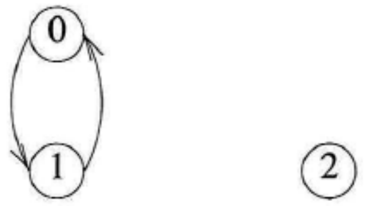
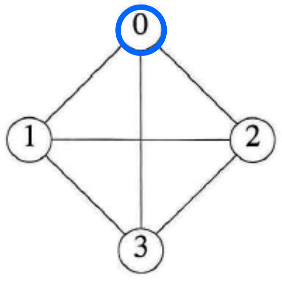


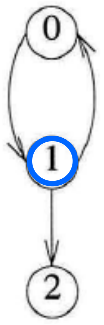
Figure 6.6: Strongly connected components of G_3

- *degree* of vertex
  - The # of edges incident to that vertex
  - For directed graph, *in-degree* and *out-degree*

degree: …



(a) G_1



(c) G_3

in-degree : … 1
out-degree : … 2

- The # of edges in G with n vertices : undirected $e = (\sum_{i=0}^{n-1} d_i)/2$
  - d_i : degree of vertex i

ADT *Graph* is

objects: a nonempty set of vertices and a set of undirected edges, where each edge is a pair of vertices.

functions:

for all *graph* ∈ *Graph*, v, v_1, and v_2 ∈ *Vertices*

Graph Create() → Java에서 생성자	::=	**return** an empty graph.
Graph InsertVertex(*graph*, v)	::=	**return** a graph with v inserted. v has no incident edges.
Graph InsertEdge(*graph*, v_1, v_2)	::=	**return** a graph with a new edge between v_1 and v_2.
Graph DeleteVertex(*graph*, v)	::=	**return** a graph in which v and all edges incident to it are removed.
Graph DeleteEdge(*graph*, v_1, v_2)	::=	**return** a graph in which the edge (v_1, v_2) is removed. Leave the incident nodes in the graph.
Boolean IsEmpty(*graph*)	::=	**if** (*graph* == empty graph) **return** *TRUE* **else return** *FALSE*.
List Adjacent(*graph*, v)	::=	**return** a list of all vertices that are adjacent to v.

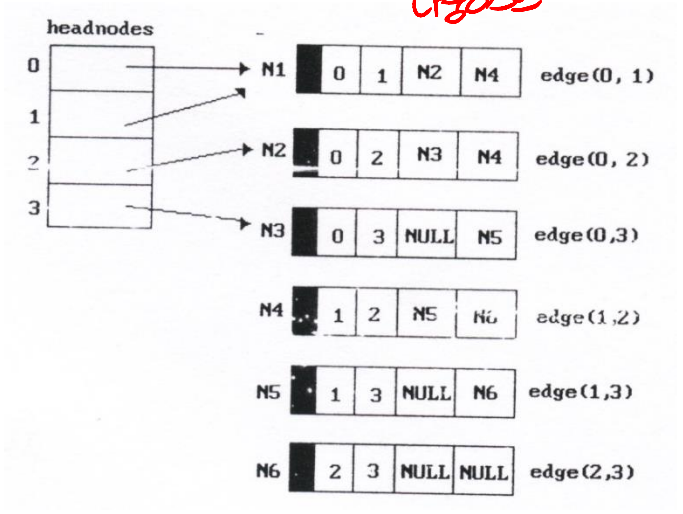
ADT 6.1: Abstract data type *Graph*

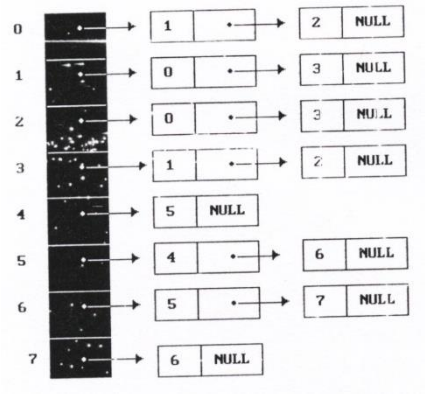
6.1.3 Graph Representation

- Adjacency Matrix
- Adjacency Lists
- Adjacency Multilists

$$
\begin{array}{c|cccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

다중리스트



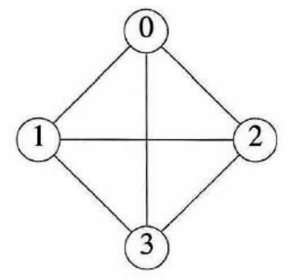


6.1.3.1 Adjacency Matrix

- Adjacency matrix *a* of G
  - two dimensional $n \times n$ array ↱ 정점의 개수
  - $a[i][j]=1$ iff edge(i, j) is in E(G)
  - $a[i][j]=0$ iff there is no edge(i, j) in E(G)

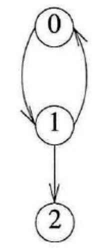
Space for adjacency matrix: …

↙ undirected graph는 symmetry matrix



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 1 & 1 & 1 & 0 \end{array}$$

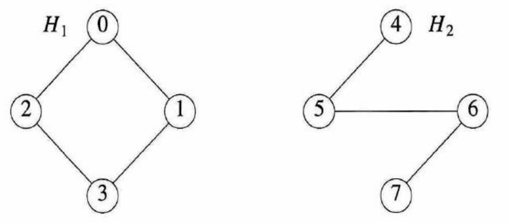
(a) G_1



$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 0 \end{array}$$

(b) G_3

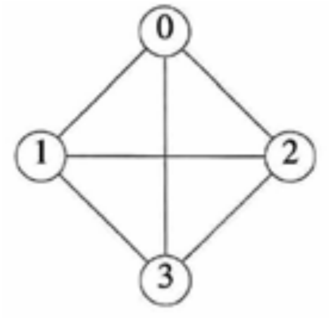
$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$



(c) G_4

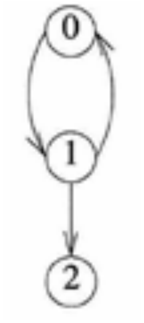
Figure 6.7: Adjacency matrices

- Properties of Adjacency Matrix
  - For a *graph*, degree of vertex *i* is its *row sum*: $\sum_{j=0}^{n-1} a[i][j]$
  - For a *digraph*, the *row sum* is the out-degree, and the *column sum* is the in-degree
  - Time complexity: O(...)
    - How many edges are there in G?
    - Is G connected?



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 1 & 1 & 1 & 0 \end{array}$$

(a) G_1



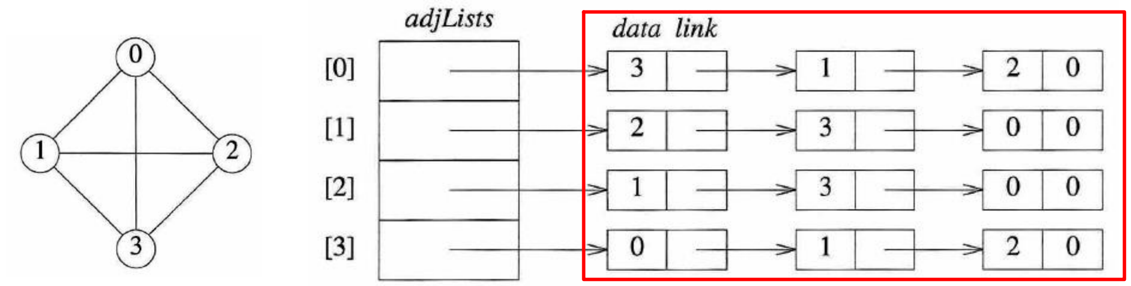
$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 0 & 0 \end{array}$$

6.1.3.2 Adjacency Lists

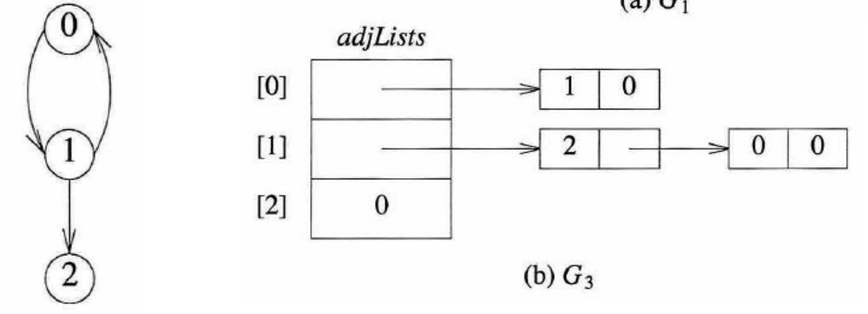
- Chain Representation
  - The n rows of the adjacency matrix are represented as n chains
  - Graph with n vertices and e edges
    - Requires n head nodes and $2e$ list nodes
  - For a digraph: e nodes

양방향이면 2e

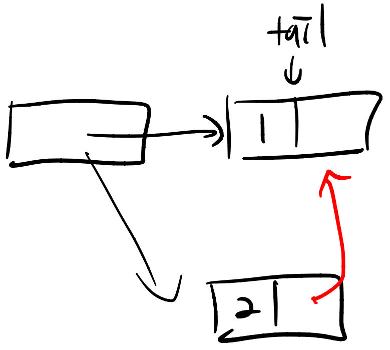
6×2

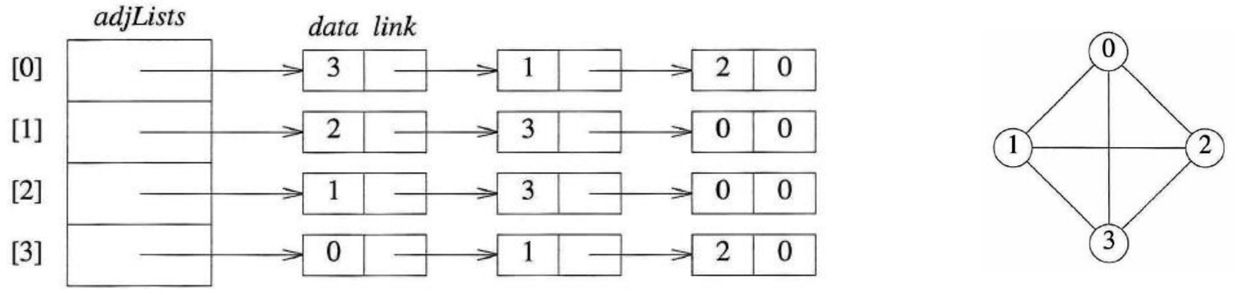


(a) G_1

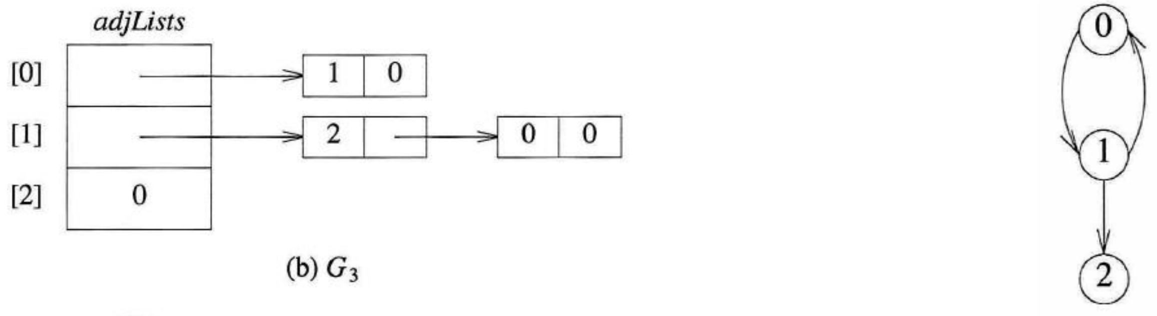


(b) G_3

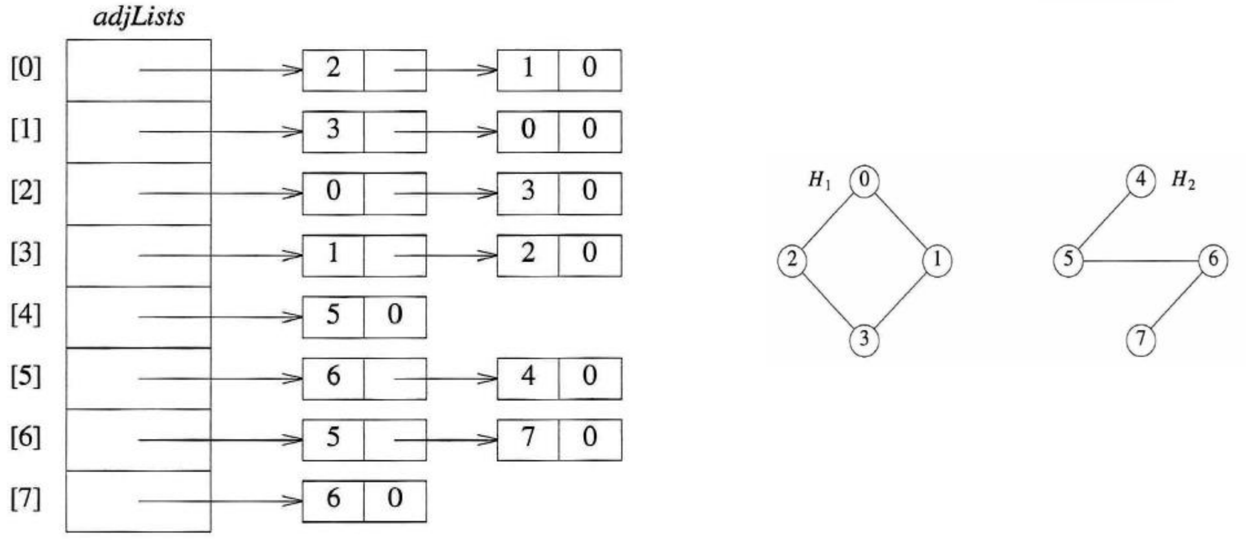




(a) G_1



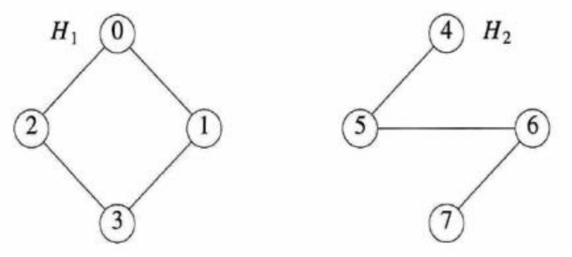
(b) G_3



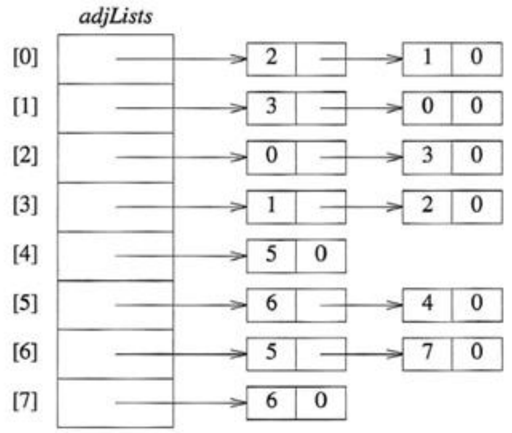
(c) G_4

Figure 6.8: Adjacency lists

1차원 배열로 adjacency list를 표현해보자



- Sequential Representation
  - The adjacency lists may be packed into an integer array *node*[*n* + *2e* + 1]
  - *node*[*i*]: the starting point of the list for vertex *i*
    - 0 <= *i* < *n*
  - *node*[*n*] = *n*+2*e*+1
  - The vertices adjacent from vertex *i*
    - Stored in *node*[*node*[*i*]], … , *node*[*node*[*i*+1]-1]



(c) G_4

0번째 vertex의 index는 9부터 시작

int *node* [*n* + 2∗*e* + 1];

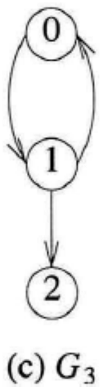
									0		1		2		3		4	5		6		7
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
9	11	13	15	17	18	20	22	23	2	1	3	0	0	3	1	2	5	6	4	5	7	6

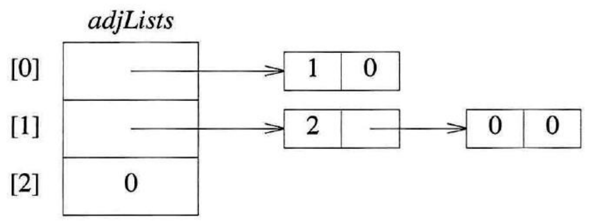
to know the end of *node* array

Figure 6.9: Sequential representation of graph G_4

⇒ 좀 복잡한 방법..

- The degree of any vertex
  - In undirected graph
    - Determined by just counting the # of nodes in its adjacency list
  - In digraph
    - Out-degree: the # of nodes on its adjacency list
    - In-degree : the # of nodes on its inverse adjacency list


(c) G_3



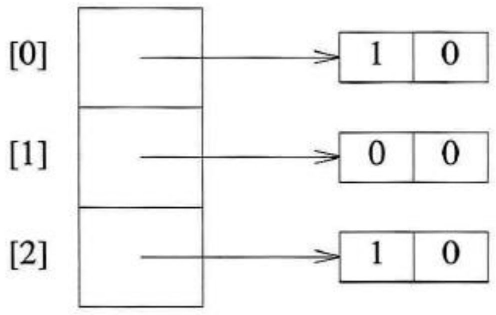


Figure 6.10: Inverse adjacency lists for G_3 (Figure 6.2(c))

- Alternate node structure of adjacency lists
  - Each node
    - edge(head/tail)
    - link for column chain, link for row chain

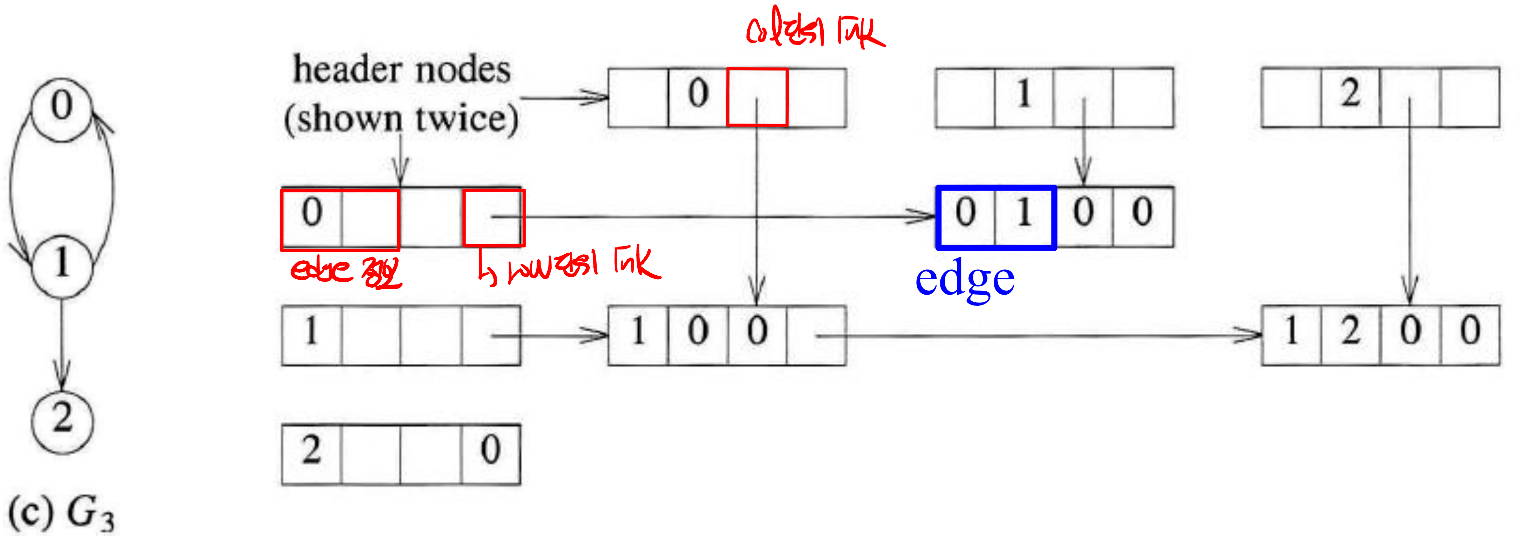


Figure 6.11: Orthogonal list representation for G_3 of Figure 6.2(c)

6.1.3.3 Adjacency Multilists

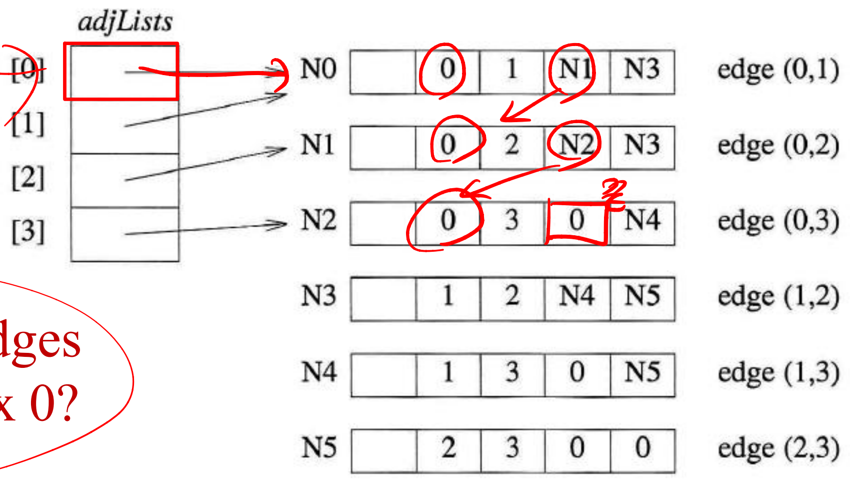
- An edge (*u*, v) in an undirected graph
  - Represented by two nodes in adjacency list representation;
  - One on the list for *u* and the other on the list for *v*

- Adjacency multilists
  - There is exactly one node for each edge
  - Lists in which nodes may be shared among several lists (an edge is shared by two different paths)

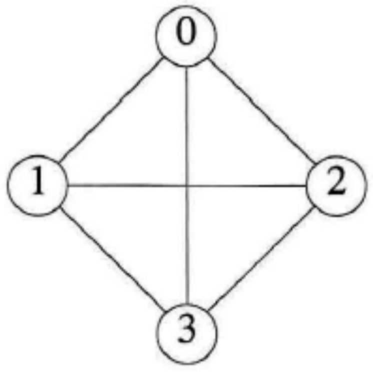
- Node structure
  - *m* : whether or not the edge has been examined
  - *link*1 : the next edge of *vertex*1
  - *link*2: the next edge of *vertex*2

m	*vertex*1	*vertex*2	*link*1	*link*2

edge


adjLists
[0]
[1]
[2]
[3]
N0 | 0 | 1 | N1 | N3 | edge (0,1)
N1 | 0 | 2 | N2 | N3 | edge (0,2)
N2 | 0 | 3 | 0 | N4 | edge (0,3)
N3 | 1 | 2 | N4 | N5 | edge (1,2)
N4 | 1 | 3 | 0 | N5 | edge (1,3)
N5 | 2 | 3 | 0 | 0 | edge (2,3)
2

How to find all edges incident for vertex 0?



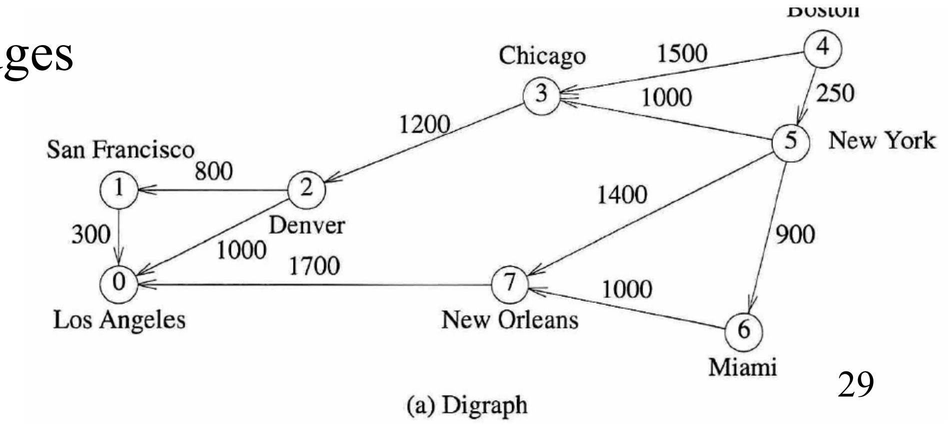
(a) G_1

The lists are		
	vertex 0:	N0 → N1 → N2
	vertex 1:	N0 → N3 → N4
	vertex 2:	N1 → N3 → N5
	vertex 3:	N2 → N4 → N5

Figure 6.12: Adjacency multilists for G_1 of Figure 6.2(a)

6.1.3.4 Weighted Edges

- The edges of a graph have weights assigned to them
- These weights may represent as
  - the distance from one vertex to another
  - cost of going from one vertex to an adjacent vertex.
- Adjacency matrix: a[i][j] would keep the weights.
- Adjacency lists
  - Add a weight field to the node structure
- Network
  - A graph with weighted edges



(a) Digraph

GRAPHS

6.1 The Graph Abstract Data Type

6.2 Elementary Graph Operations

6.3 Minimum Cost Spanning Trees