Chap 6. Graph (1)

Contents

6.1 The Graph Abstract Data Type

6.1.1 Introduction

- Königsberg bridge problem

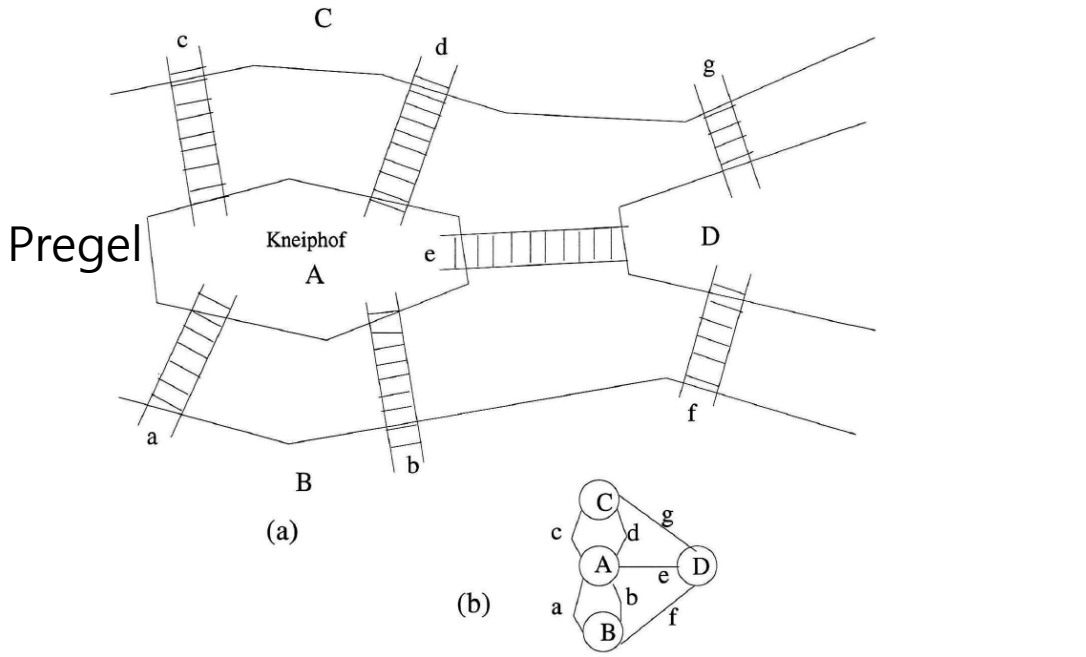


Figure 6.1: (a) Section of the river Pregel in Königsberg; (b) Euler's graph

- Eulerian walk
  - *degree* of each vertex is even

6.1.2 Definition

- *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 graph*
  - the pair of vertices representing an edge is unordered
    - (*u*,*v*) and (*v*,*u*) : the same edge

- *Directed graph*
  - the pair of vertices representing an edge is ordered
    - <*u*,*v*> and <*v*,*u*> : two different edges
    - <*u*,*v*> : *u* is the *tail* and *v* is the *head*

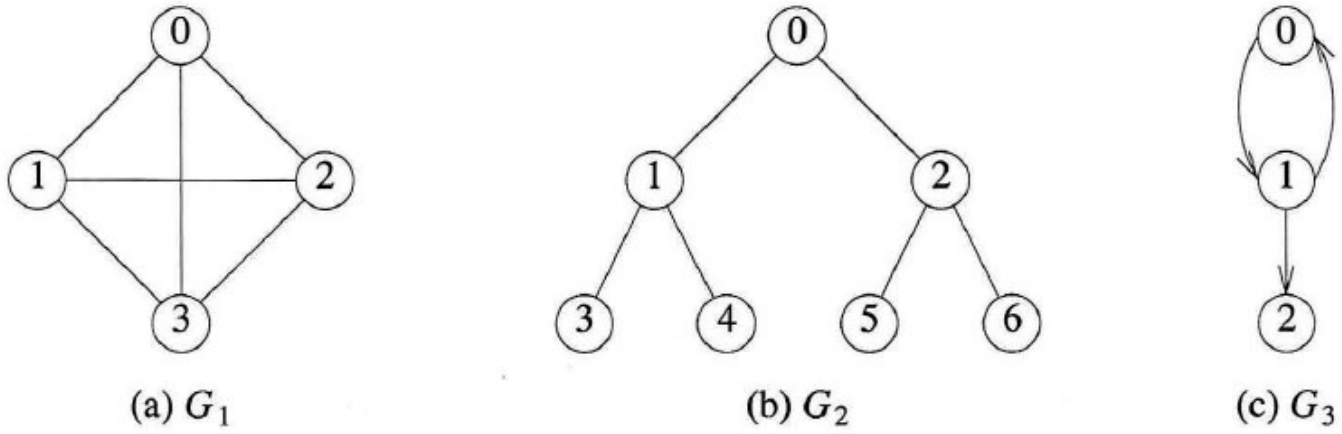


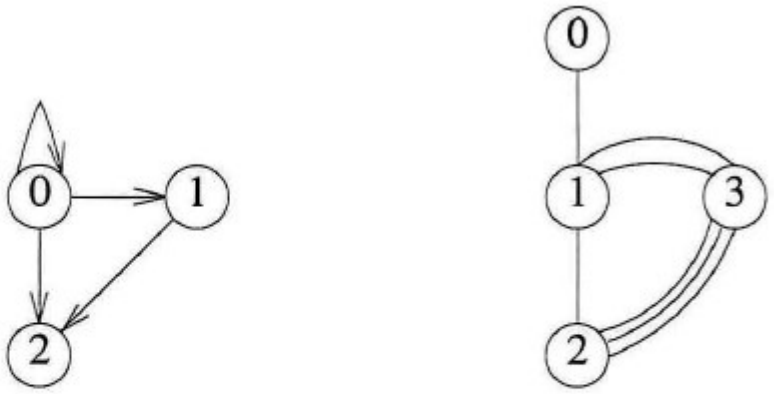
Figure 6.2: Three sample graphs

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

$V(G_{2)} = \{0,1,2,3,4,5,6\}$; $E(G_2) = \{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$

$V(G_3) = \{0,1,2\}$; $E(G_3) = \{<0,1>,<1,0>,<1,2>\}$.

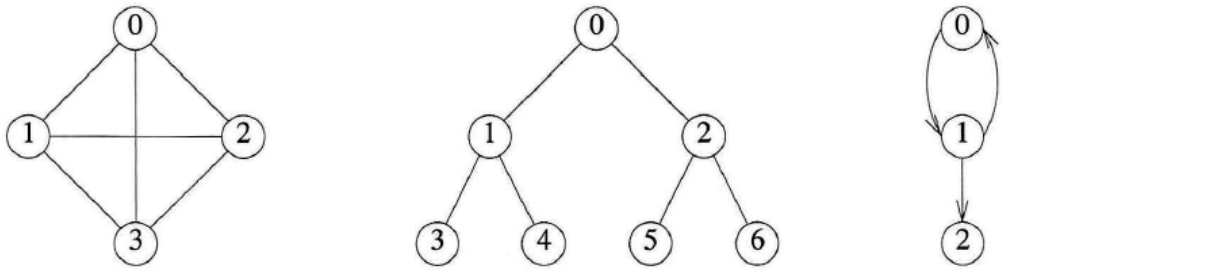
- Restrictions on Graphs
  1) A graph may not have an edge from a vertex back to itself, that is , *self edges* or *self loops*.
  2) A graph may not have multiple occurrences of the same edge.



(a) Graph with a self edge (b) Multigraph

Figure 6.3: Examples of graphlike structures

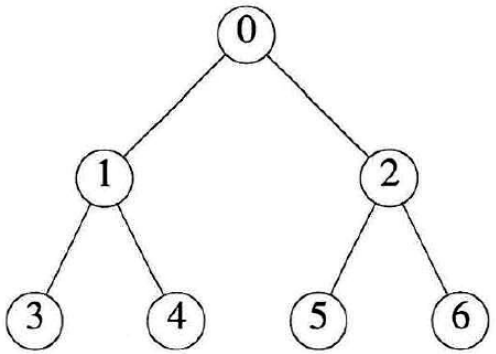
- *Complete graph*
  - n-vertex, undirected graph with $n(n\text{-}1)/2$ edges



- In the case of directed graph on n vertices,
  - the maximum number of edges is $n(n\text{-}1)$

- If (u,v) is an edge in E(G),
  - vertices u and v are *adjacent*.
  - the edge (u, v) is *incident* on vertices u and v.

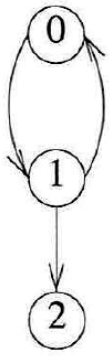
  - G2
    - The vertices adjacent to vertex 1 are 3, 4, and 0.
    - The edges incident on vertex 2 are (0,2), (2,5), and (2,6).



(b) G_2

- If <*u*,*v*> is a directed edge,
  - vertex *u* is *adjacent to* *v*, and *v* is *adjacent from* *u*.
  - the edge <*u*,*v*> is *incident* to *u* and *v*.

  - G3
    - The edges incident to vertex 1 are <0,1>, <1,0>, and <1,2>.



(c) G_3

- G' : *Subgraph* of G
  - graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$

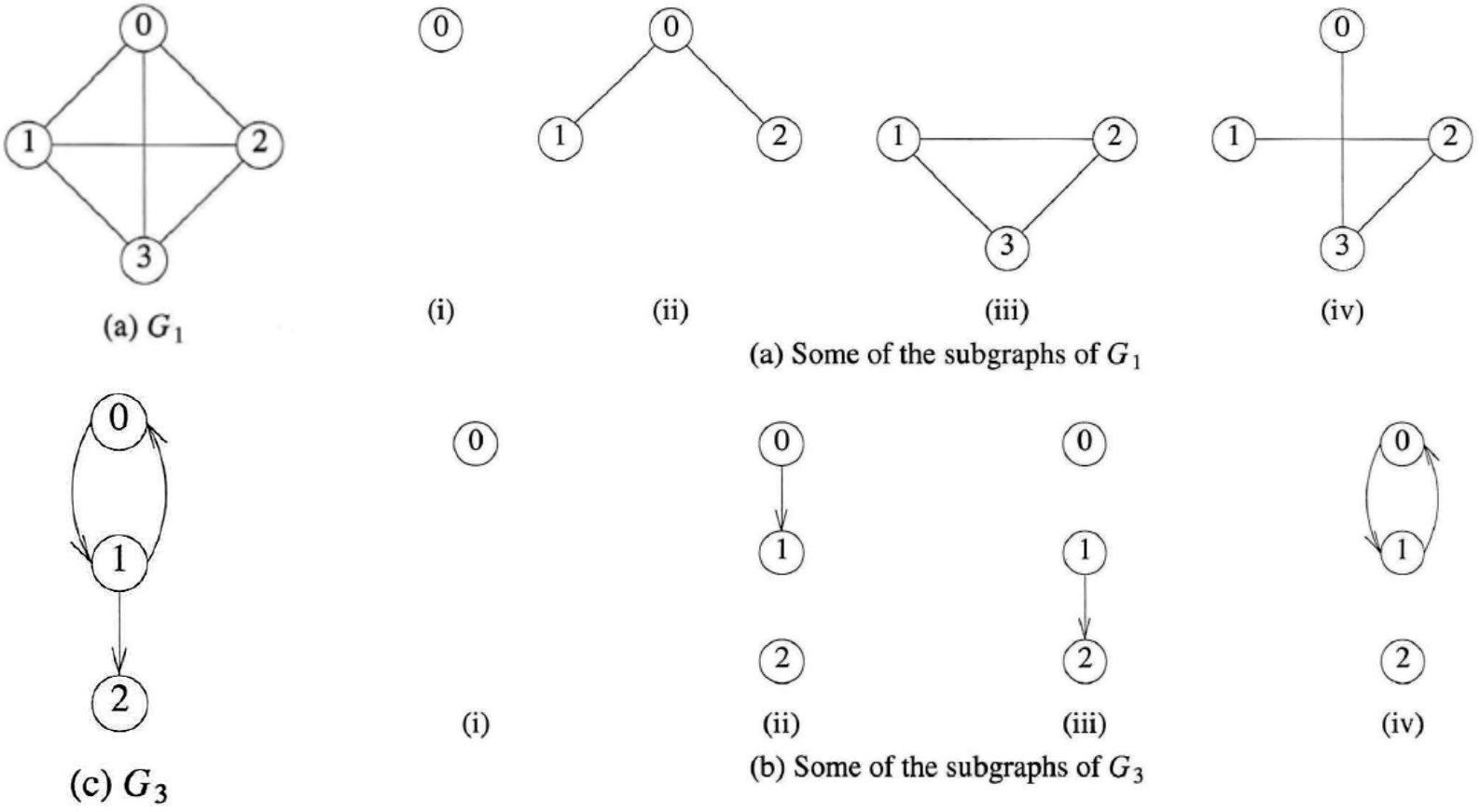
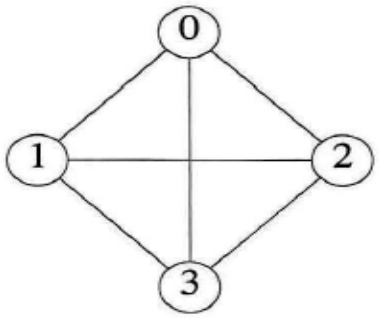


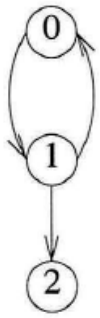
Figure 6.4: Some subgraphs

- *Path* from u to v in G
  - a sequence of vertices $u, i_1, i_2, ..., i_k, v$ such that $(u, i_1), (i_1, i_2), ..., (i_k, v)$ are edges in E(G)
  - The *length* of path is the number of edges on it.
  - A *simple path* is a path in which all vertices except possibly the first and last are distinct.
  - A *cycle* is a simple path in which the first and last vertices are the same.



(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

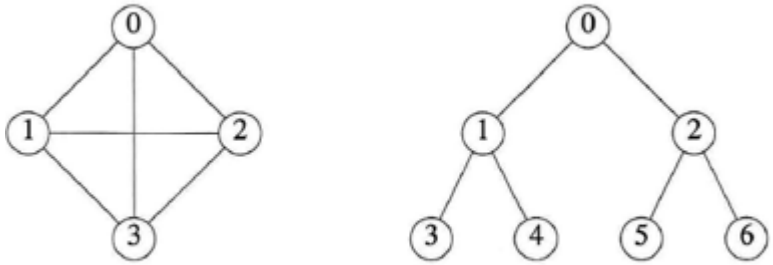


(c) G_3

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

- Vertices *u* and *v* are *connected* in (undirected) graph G *iff* there is a path in G from *u* to *v*

- *Connected graph*
  - for every pair of distinct vertices *u* and *v* in V(G), there is a path from *u* and *v* (ex: G_1, G_2 in Figure 6.2)



- *Connected component*
  - *maximal* connected subgraph

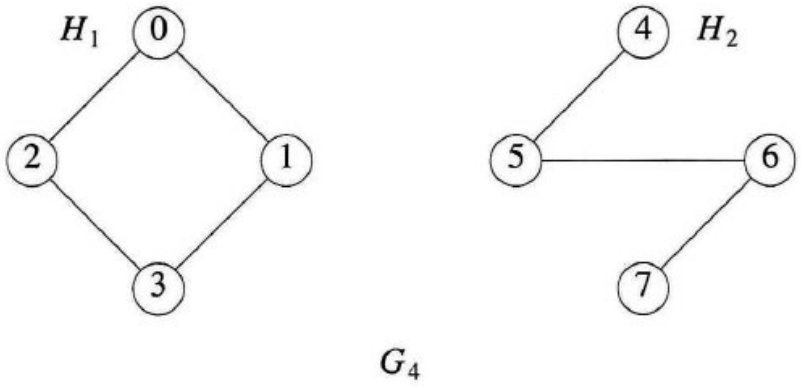
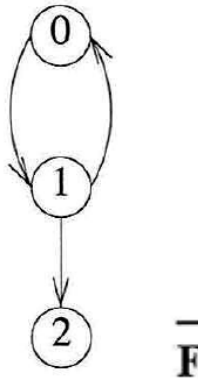


Figure 6.5: A graph with two connected components

- A *tree* is a connected acyclic graph.

- For a directed graph G,
  - *strongly connected graph*
  - *strongly connected component*

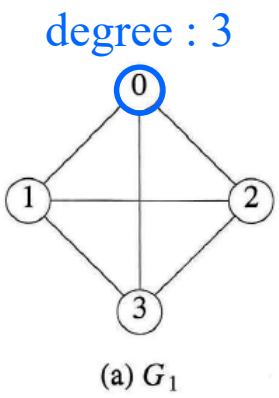


(c) G_3

Figure 6.6: Strongly connected components of G_3

G3 is not strongly connected
G3 has two strongly connected components.

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



(a) G_1

in-degree : 1
out-degree : 2

(c) G_3

- If d_i is the degree of vertex i in undirected graph G with n vertices and e edges , *the number of edges* is $e = (\sum_{i=0}^{n-1} d_i)/2$

※ In the remainder of this chapter,
graph : undirected graph, **digraph** : directed graph

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()	::=	**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

6.1.3.1 Adjacency Matrix

- Definition
  - G=(V, E) is a graph with n vertices, $n \geq 1$
  - *adjacency matrix* a of G
    - two dimensional $n \times n$ array
    - $a[i][j]=1$ *iff* edge(i, j) is in E(G)
    - $a[i][j]=0$ *iff* there is no edge(i, j) in E(G)

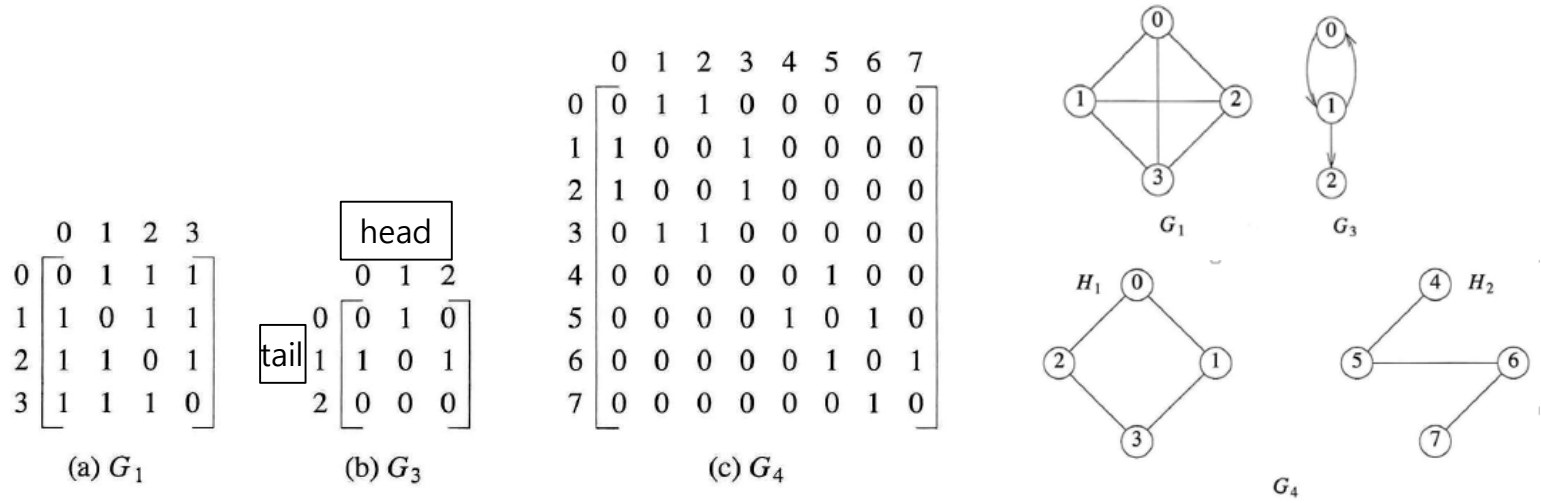


Figure 6.7: Adjacency matrices

- Properties
  - The adjacency matrix of G is a two dimensional n x n array, say a
    - a is *symmetric* for undirected G
    - edge(i, j) is in E(G) *iff* edge(j, i) is also in E(G)

- For an undirected graph,
  - degree of vertex i is its *row sum*: $\sum_{j=0}^{n-1} a[i][j]$

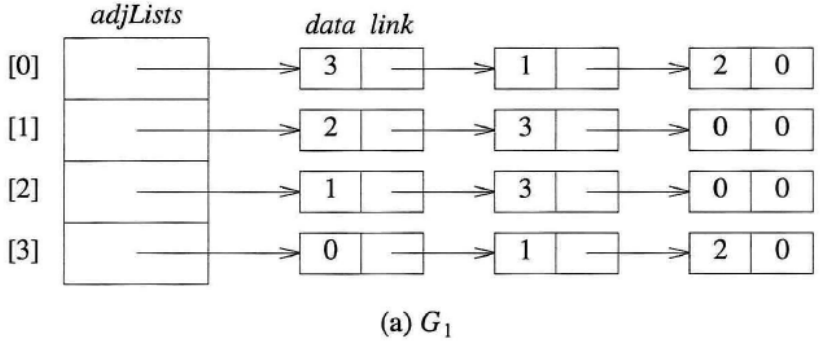
- For a directed graph,
  - the *row sum* is the out-degree
- the *column sum* is the in-degree

- How many edges are there in G?
  - Complexity of operations
    - n^2 - n entries of the matrix have to be examined
    - $O(n^2)$

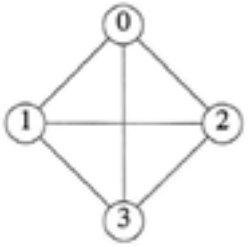
6.1.3.2 Adjacency Lists

- Representation
  - one list for each vertex in G
    - nodes in list *i* represent vertices that are adjacent from vertex *i*
    - each list has a head node

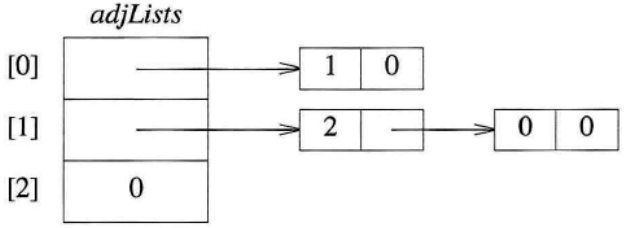
- Vertices in a list are not ordered
  - fields of node
    - *data* : index of vertex adjacent to vertex *i*
    - *link*



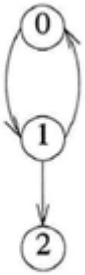
(a) G_1



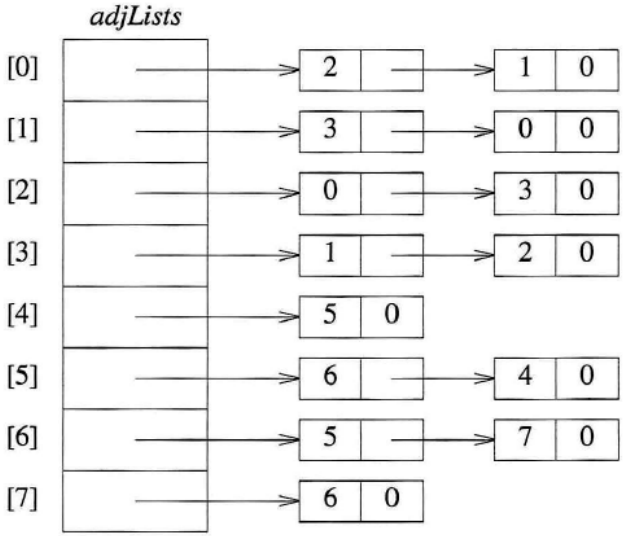
(a) G_1



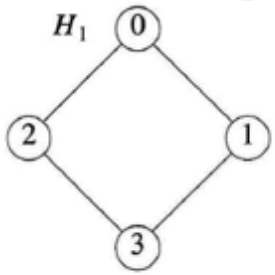
(b) G_3

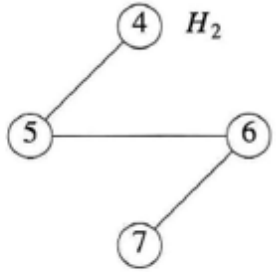


(c) G_3



(c) G_4





G_4

Figure 6.8: Adjacency lists

- An undirected graph with n vertices and e edges
  - Adjacency Lists
    - requires n head nodes and $2e$ list nodes
    - the number of edges in G : the number of list nodes / 2

- Packing nodes
  - eliminate pointers
  - starting point of list for vertex i : *node*[i]
  - vertices adjacent from node i :

    node[*node*[i]], ..., *node*[*node*[i+1]-1]

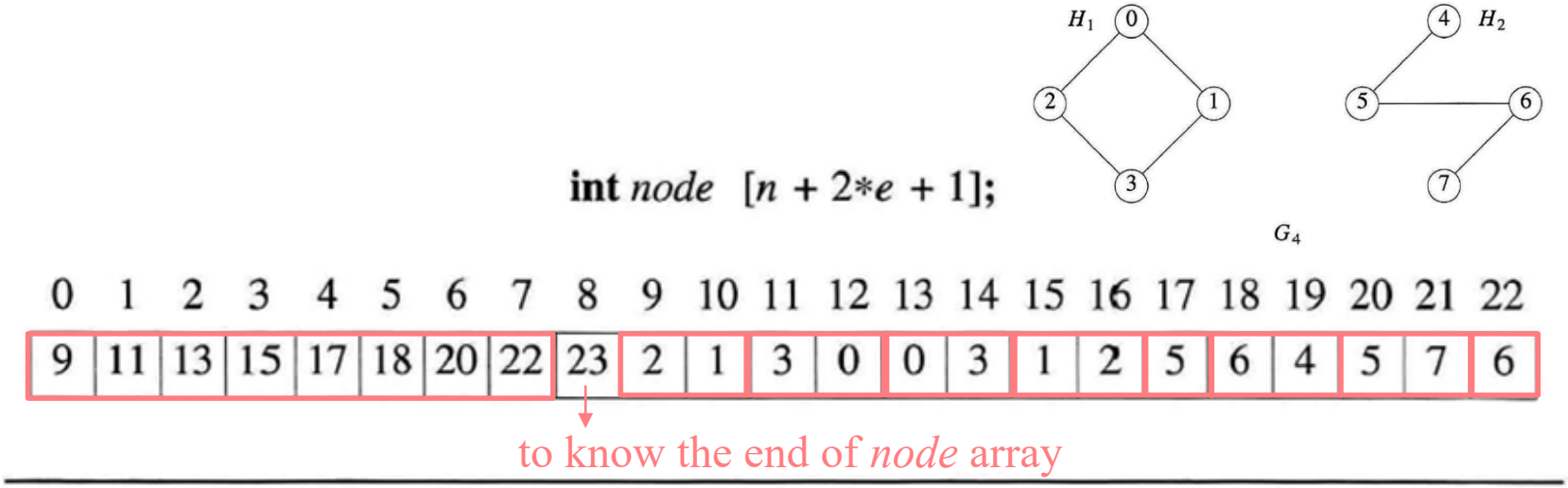
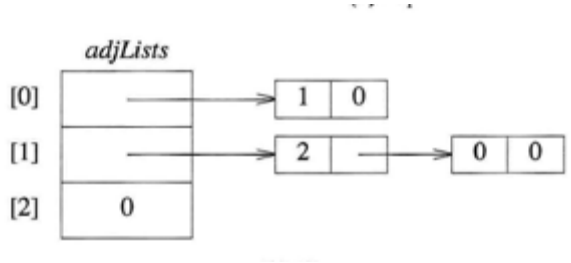


Figure 6.9: Sequential representation of graph G_4

- For a digraph,
  - the number of list nodes is only *e*
  - For any vertex
    - out-degree *:* the # of nodes on its *adjacency list*
    - in-degree : the #of nodes on its *inverse adjacency list*



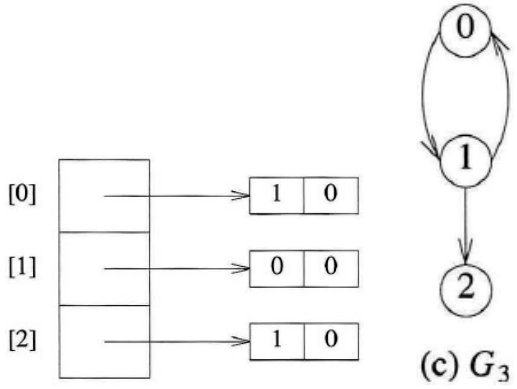


Figure 6.10: Inverse adjacency lists for G_3 (Figure 6.2(c))

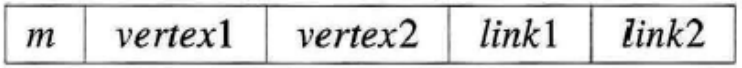
6.1.3.3 Adjacency Multilists

- Adjacency list in undirected graph
  - Each edge (u, v) is represented by two entries.

- Multilist
  - Lists in which nodes may be shared among several lists.

- Adjacency multilists
  - For each edge, there will be exactly one node, but
  - this node will be in two lists.

- 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

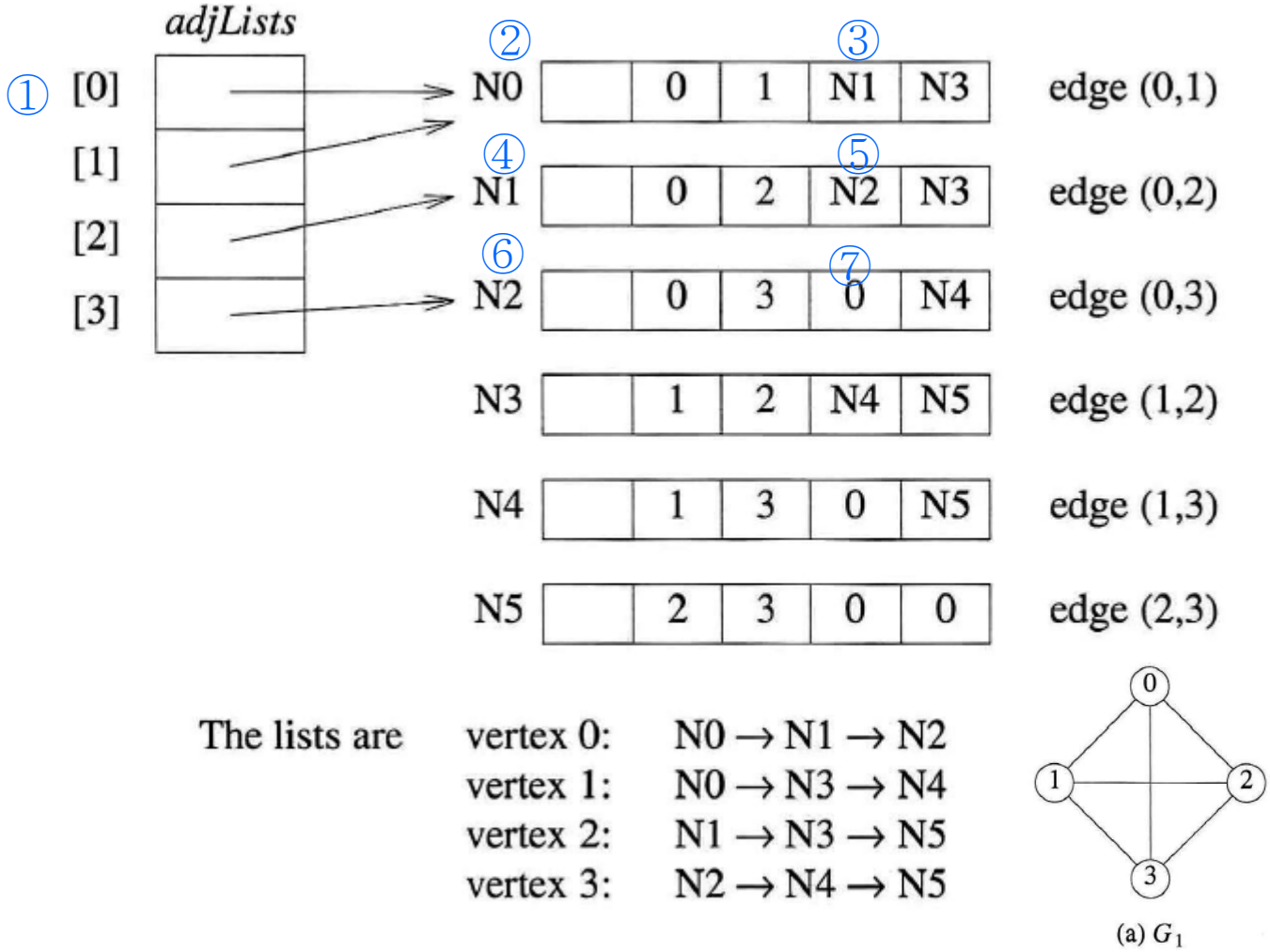


Figure 6.12: Adjacency multilists for G_1 of Figure 6.2(a)

Q. How to find all edges incident on vertex 0 ? ① ~ ⑦