Chap 6. Graph (1)

Contents

1. **The Graph Abstract Data Type**
2. Elementary Graph Operations
3. Minimum Cost Spanning Trees
4. Shortest Path
5. ACTIVITY NETWORKS

6.1 The Graph Abstract Data Type

6.1.1 Introduction

❖ Königsberg bridge problem

Pregel

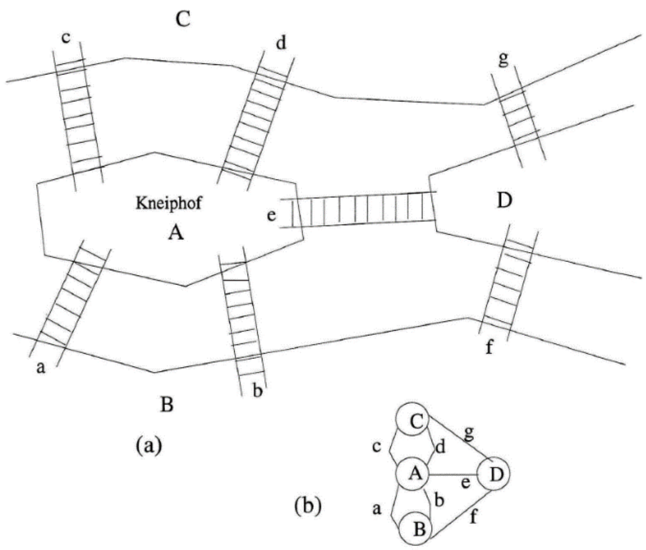


Figure 6.1: (a) Section of the river Pregel in Konigsberg; (b) Euler's graph

- 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*

6.1.2 Definition

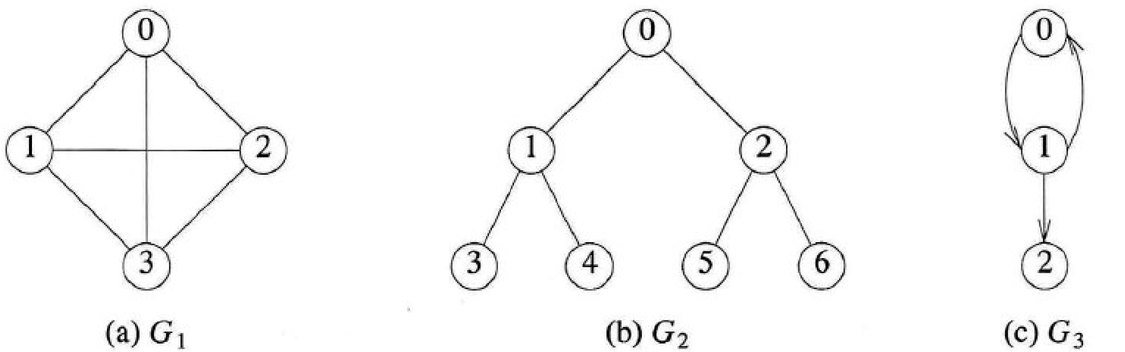


Figure 6.2: Three sample graphs

- **V(G 1) = {0,1,2,3};**
  E(G 1) = {(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)}
- ***V(G2)* = {0,1,2,3,4,5,6};**
  ***E(G2)* = {(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)}**
- ***V(G3)* = {0, 1,2};**
  ***E(G3)* = { <0,1>,<1,0>,<1,2> }.**

6.1.2 Definition

❖ **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.

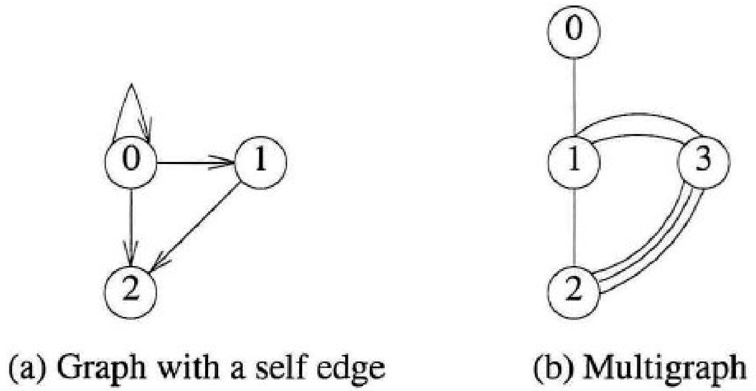
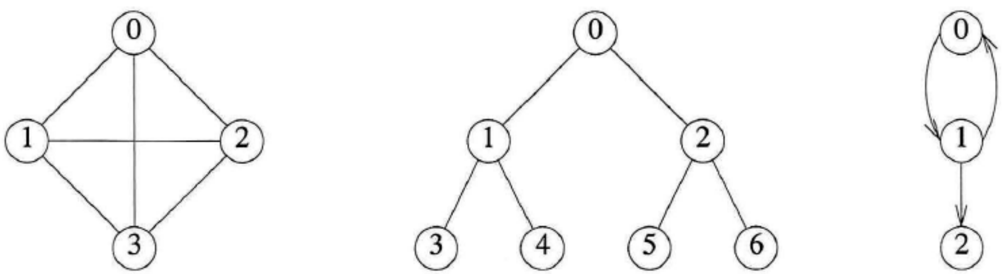


Figure 6.3: Examples of graphlike structures

6.1.2 Definition

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

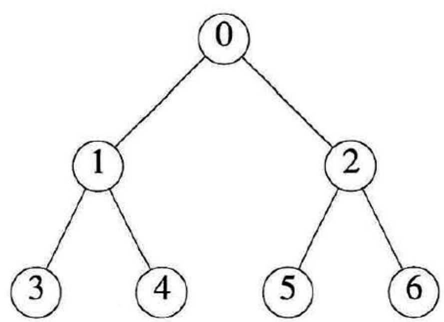


(a) G_1 C.G. (b) G_2 Not C.G. (c) G_3 Not C.G.

- **In the case of directed graph on *n* vertices,**
  - the maximum number of edges is $n(n-1)$

6.1.2 Definition

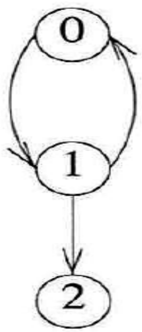
- **If (*u*,*v*) is an edge in E(G),**
  - vertices *u* and *v* are *adjacent.*
  - the edge (*u*, *v*) is *incident* on vertices *u* and *v.*
- **G2**
  - The vertices adjacent to vertex 1 are 3, 4, and 0.
  - The edges incident on vertex 2 are (0,2), (2,5), and (2,6).



(b) G_2

6.1.2 Definition

- **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

6.1.2 Definition

- **G' : *Subgraph* of G**
  - graph G' such that V(G')⊆V(G) and E(G')⊆E(G)

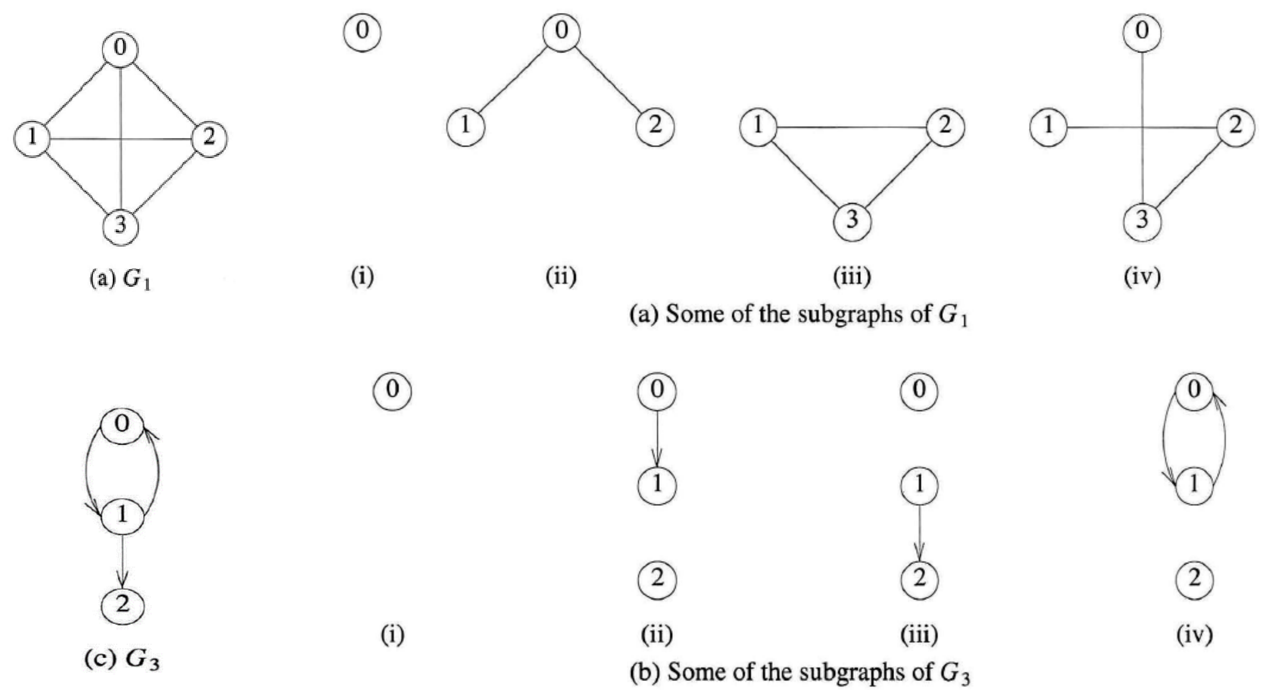
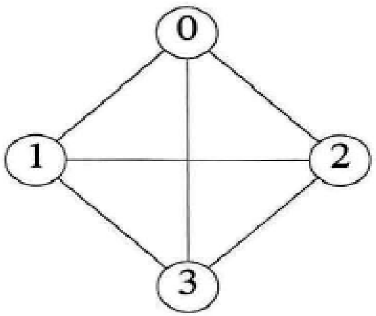


Figure 6.4: Some subgraphs

6.1.2 Definition

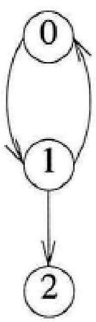
- ***Path*** **from *u* to *v* in G**
  - a sequence of vertices $u, i_1, i_2, ..., i_k, v$ such that $(u, i_1), (i_1, i_2), ...,$ (i_k, v) are edges in E(G)
  - The *length* of path is the number of edges on it.
  - A *simple path* is a path in which all vertices except possibly the first and last are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

6.1.2 Definition



(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

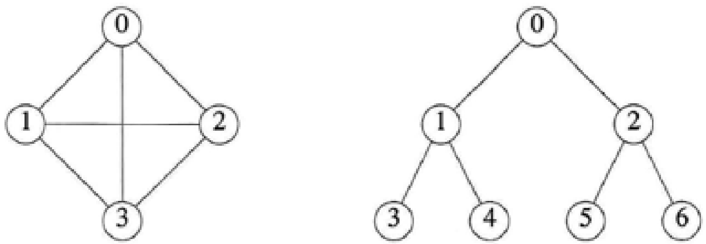
0, 1, 2 - simple *directed* path

0, 1, 2, 1 - not a path

6.1.2 Definition

- **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)



(a) G_1 (b) G_2

6.1.2 Definition

- ***Connected component***
  - *maximal* connected subgraph

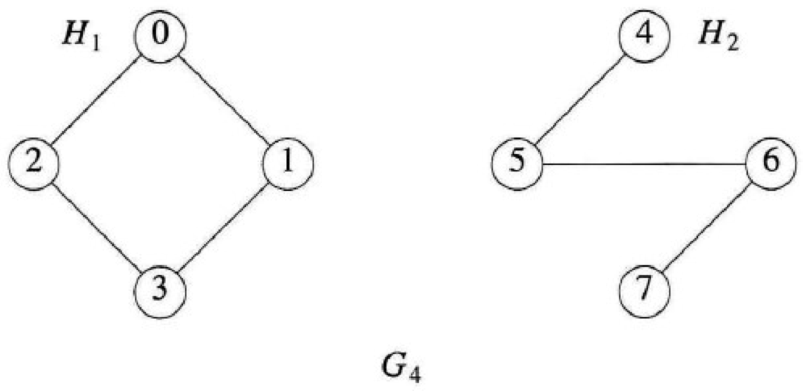
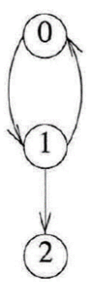


Figure 6.5: A graph with two connected components

6.1.2 Definition

- **A *tree* is a connected acyclic graph.**
- **For a directed graph G,**
  - *strongly connected graph*
  - *strongly connected component*
  - G3 is not strongly connected
  - G3 has two strongly connected components.



(c) G_3

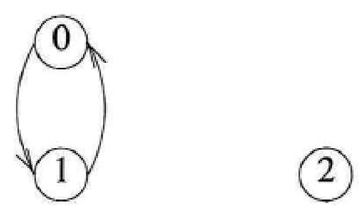


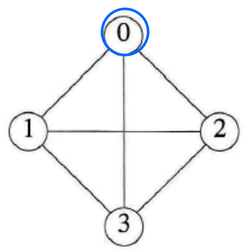
Figure 6.6: Strongly connected components of G_3

6.1.2 Definition

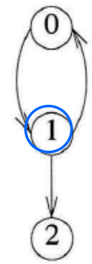
- ***degree* of vertex**
  - The number of edges incident to that vertex
  - For directed graph, *in-degree* and *out-degree*

- **If d_i is the degree of vertex i in undirected graph G with n vertices and e edges , *the number of edges* is**

degree : 3



(a) G_1



(c) G_3

in-degree : 1
out-degree : 2

$$e = \left(\sum_{i=0}^{n-1} d_i\right)/2$$

6.1.2 Definition

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 $graph \in Graph$, v, v_1, and $v_2 \in Vertices$

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*

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

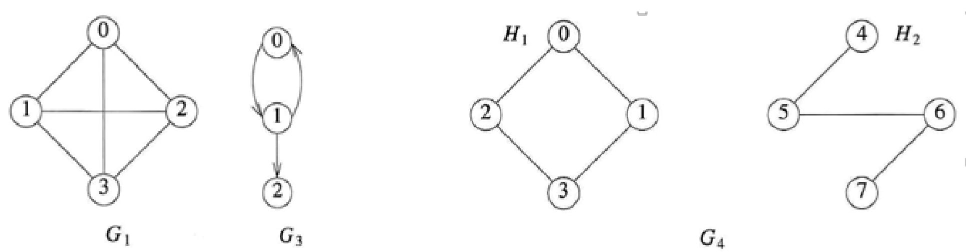
6.1.3 Graph Representation

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)

6.1.3 Graph Representation

6.1.3.1 Adjacency 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

head

tail

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

(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

6.1.3 Graph Representation

- **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

6.1.3 Graph Representation

❖ 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*

6.1.3.2 Adjacency Lists

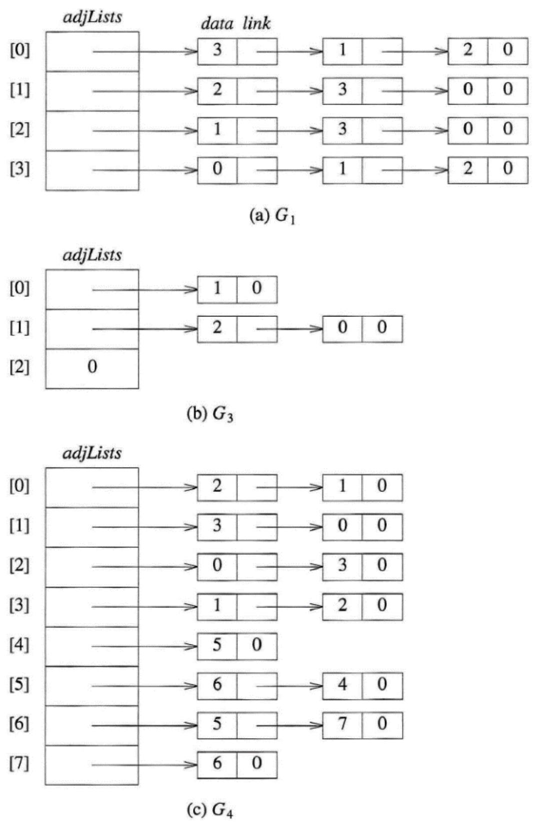
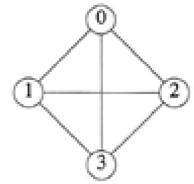
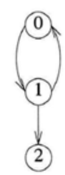


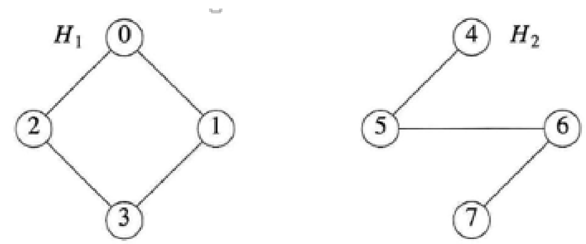
Figure 6.8: Adjacency lists



(a) G_1



(c) G_3



G_4

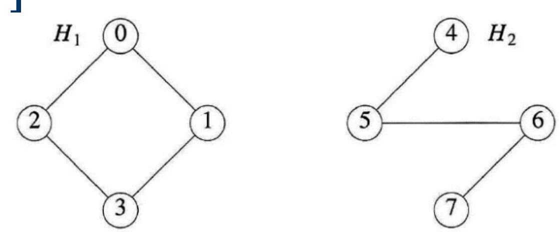
6.1.3.2 Adjacency Lists

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

Sequential representation of graph

- **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]



G_4

int *node* $[n + 2*e + 1]$;

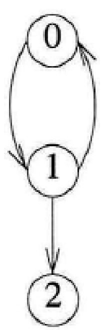
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
9	11	13	15	17	18	20	22	23	2	1	3	0	0	3	1	2	5	6	4	5	7	6

to know the end of *node* array

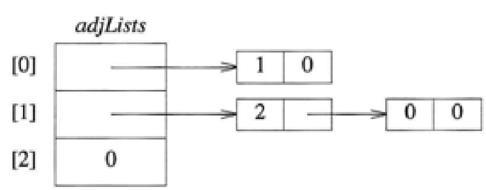
Figure 6.9: Sequential representation of graph G_4

6.1.3.2 Adjacency Lists

- **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*



(c) G_3



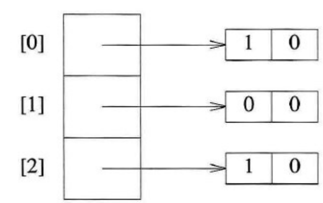


Figure 6.10: Inverse adjacency lists for G_3 (Figure 6.2(c))

6.1.3.3 Adjacency Multilists

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

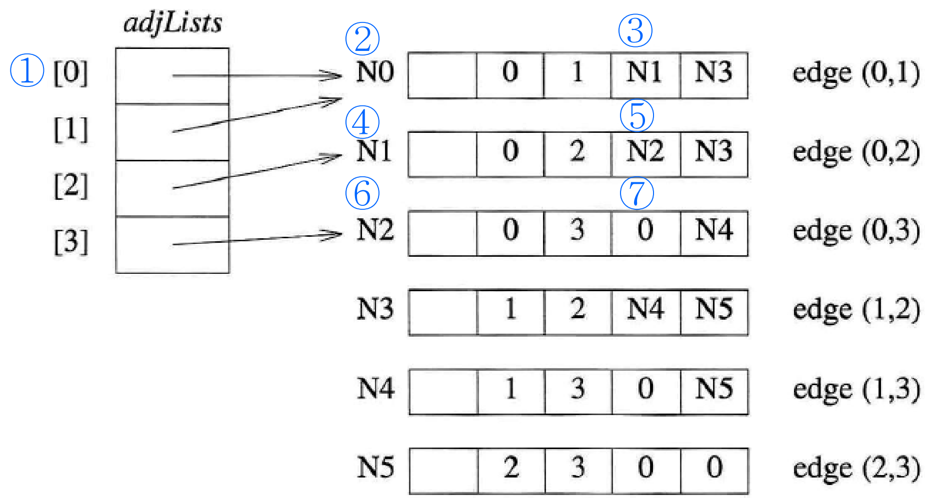
- **Multilist**
  - Lists in which nodes may be shared among several lists.

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

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

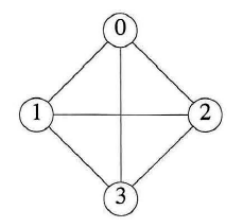
- **Node structure**
  - *m* : whether or not the edge has been examined
  - *link*1 : the next edge of *vertex*1
  - *link*2: the next edge of *vertex*2

6.1.3.3 Adjacency Multilists



The lists are

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



(a) G_1

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