KNU KYUNGPOOK NATIONAL UNIVERSITY

Lecture #20: Graph

School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

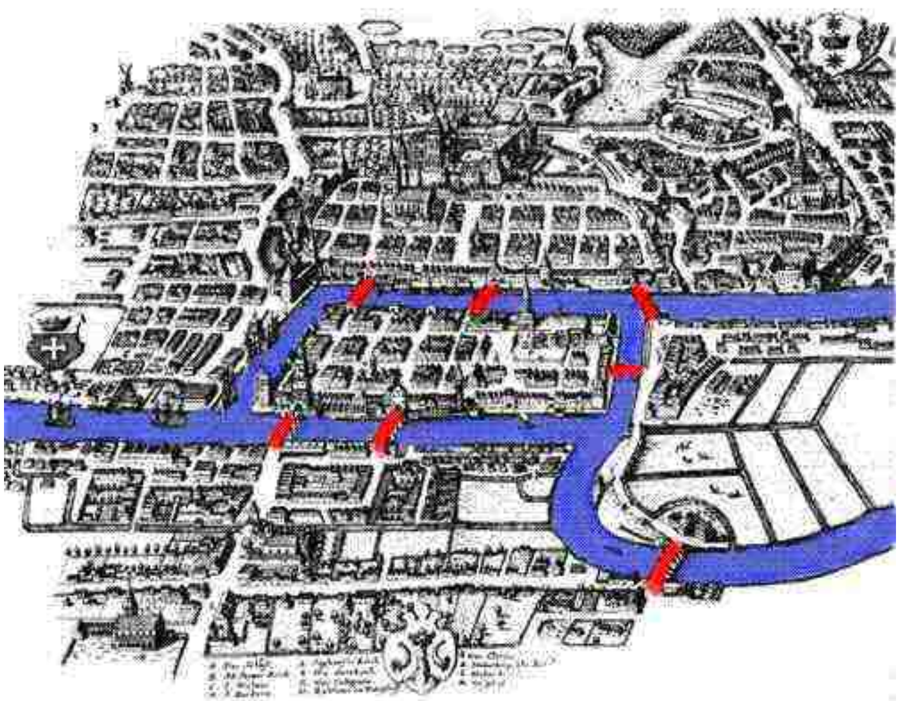
Agenda

- **Graph elementary**
  - ➢ Definitions & Terminologies
  - ➢ Properties
  - ➢ Connected components

- **Graph algorithms**
  - ➢ Breadth First Search (BFS)
  - ➢ Depth First Search (DFS)
  - ➢ Minimum Spanning Trees (MST)
    - Prims & Kruskals
  - ➢ Shortest Paths and Transitive Closure
    - Dijkstra's algorithm

Introduction to Graph

- **The use of graph dates back to 1736 when Euler used to solve a bridge problem (‘konigsberg’s bridege problem)**
- 철학자 칸트의 산책
  - 어떤 경로를 선택해도 모든다리를 한번씩만 지나면서 건널수가 없었다
  - 마을사람들은 늙은 칸트가 걱정되서 오일러에게 한번에 건널수 있는 다리 제안



쾨니히스베르크 시와 7개의 다리

Leonardo Euler
(1707-1783)

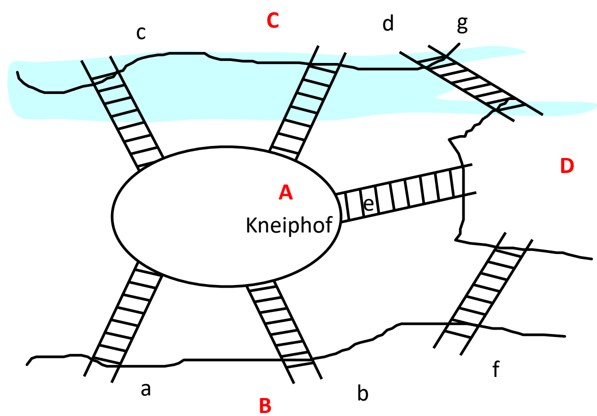
Introduction to Graph

- 오일러는 수학적으로 해답이 존재하지 않는다고 증명
- 그래프 문제로 변환하여 풀었다
  - ➢ 각 점들 사이를 잇는 선분을 모두 한번씩만 지나서 처음 자리로 돌아오는 경로를 '오일러 경로'라고 한다

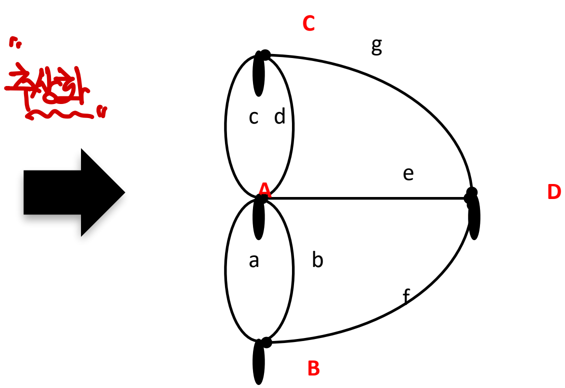
- **Given situation:**
  - ➢ **In a city, a river flows around an island and then divides into two**
  - ➢ **There are 4 land areas with river as boundary. These 4 lands are connected by 7 bridges**

- **Problem:**
  - ➢ **Starting at one land area, is it possible to walk across all bridges exactly once and rerun back to starting land area?**

- **Abstraction of the problem**
  - ➢ **He transformed land areas into vertices and the bridges into edges.**
  - ➢ **He defined the degree of a vertex as the number of edges incident on it.**
  - ➢ **He proved that this is possible if the degree of each vertex is even**
  - ➢ **Euler's Walk satisfy this.**

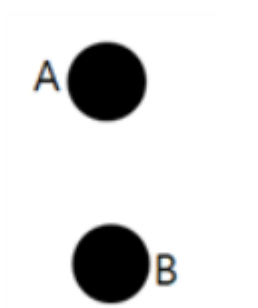


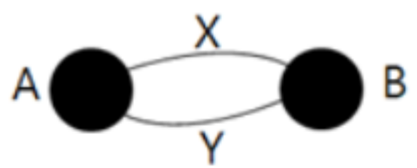
"추상화"

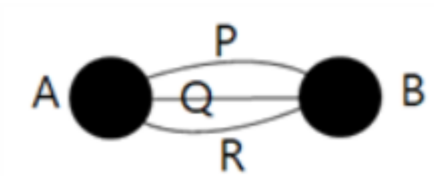


Euler's Walk (special walk in a graph)

- **Example**
  - A, B 두지점이 있다고 가정
  - A지점에서 B지점 갔다가 A로 돌아오는 경우
  - 이때 놓인 다리는 모두 한번씩만 지나야 한다고 가정
  - 다리의 개수가 짝수개이면 나가는 다리와 들어오는 다리의 개수가 같아져 조건을 만족
  - 다리가 홀수개이면 어느 하나가 부족한 상황이 필연적으로 발생
  - 따라서 자연수 n에 대해 n개의 지점을 연결하는 모든 다리를 1번씩만 지나 출발점으로 들어오기 위한 필요충분 조건은 다리의 개수가 모두 각각 짝수개여야 함



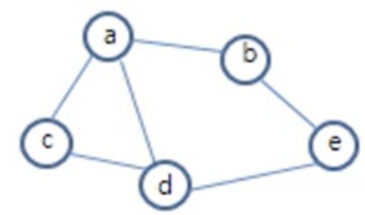




Other Applications of Graph

- Dealing with problems which have a fairly natural graph/network structure, for example:
  - Road networks (subway networks)
    - nodes = towns/road junctions
    - arcs = roads
  - Communication networks
    - Designing telephone systems
    - Computer network planning
    - Routing tree building
  - Computer systems
    - Circuit equation
  - Foreign exchange/multinational tax planning (network of fiscal flows)
  - Social graph

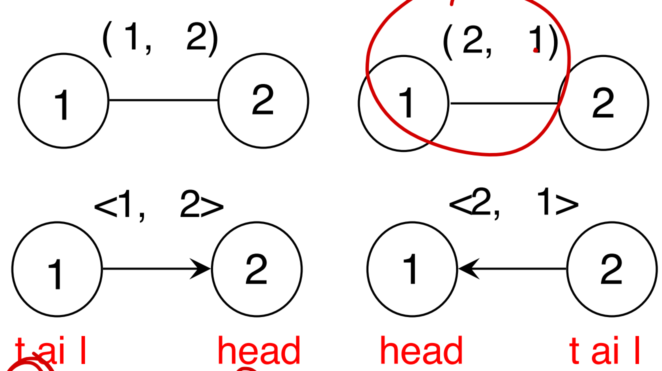
Definitions



- A graph G consists of to sets(V and E): $G = (V, E)$, where
  - ➢ $V(G)$: a finite and nonempty set of vertices (정점, 꼭지점)
    - {a, b, c, d, e}
  - ➢ $E(G)$: is a set of pairs of vertices called edges (Vertex 간의 관계)
    - {(a,b), (a, c), (a, d), (d, e)}
  - ➢ $E(G)$ is finite and possibly empty
  - ➢ Restrictions
    - A graph may not have an edge from a vertex i back to itself
    - A graph may not have multiple occurrence of the same edge

G=(V,E)
E(G)={(a,b), (a,c), (a,d), (d,e)

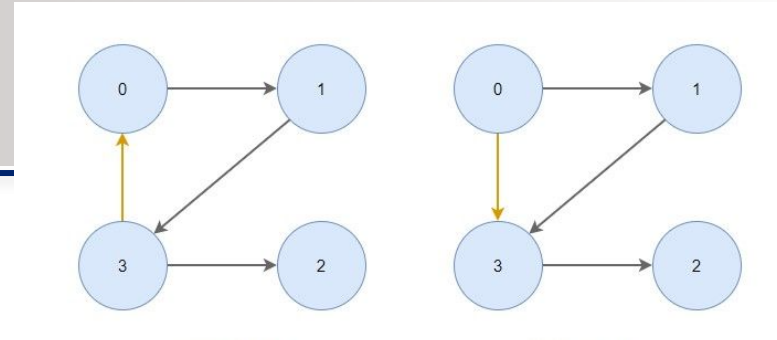
- Edge의 표기
  - ➢ Undirected graph
    - 방향성이 없음: $(u, v) = (v, u)$
  - ➢ Directed graph (digraph)
    - 방향성이 존재: $< u, v > \neq < v, u >$



T F

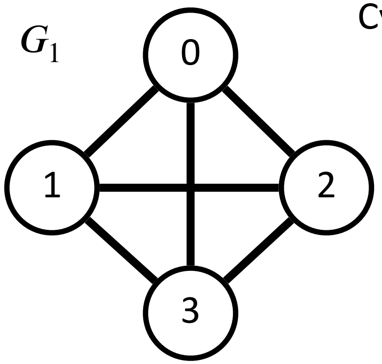
<1,2>
(1,2)

Example 1



- **Cycle:** 시작과 끝이 동일한 **vertex**인 **path**
- **Acyclic:** 방향 비순환 그래프, 다시 돌아갈 수 없다

G_1 Cyclic

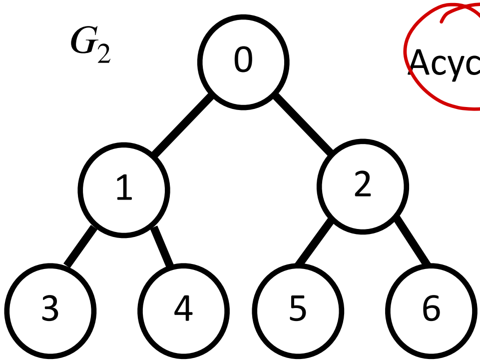


$$V(G_1) = \{0,\ 1,\ 2,\ 3\}$$
$$E(G_1) = \{(0,1),\ (0,2),\ (0,3),\ (1,2),\ (1,3),\ (2,3)\}$$

0
1

Cycle: 시작과 끝이 동일한 vertex인 path

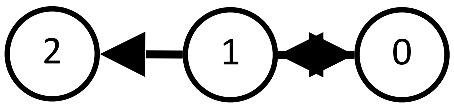
G_2 Acyclic



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

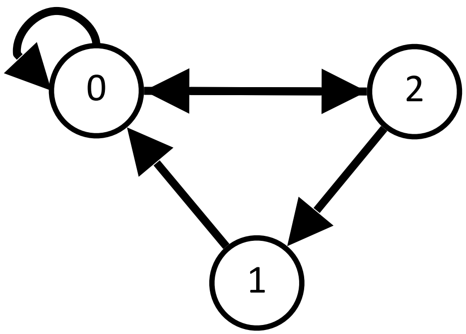
Example 2

G_3



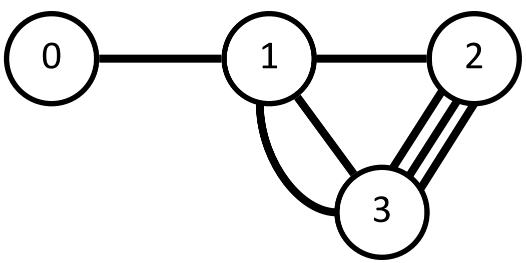
$$V(G_3) = \{0, 1, 2\}$$
$$E(G_3) = \{ <0,1>, <1,0>, <1,2> \}$$

G_4



$$V(G_4) = \{0, 1, 2\}$$
$$E(G_4) = \{ <0,0>, <0,2>, <1,0>, <2,1>, <2,0> \}$$

G_5



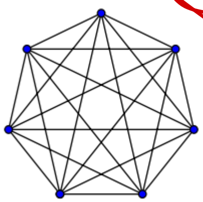
$$V(G_5) = \{0, 1, 2,3\}$$
$$E(G_5) = \{(0,1), (1,2), (1,3), (3,1), (3,2), (2,3), (3,2)\}$$

Terminology 1

- **A Complete graph (or a max clique)**
- 서로 다른 두 개의 꼭짓점이 반드시 하나의 변으로 연결된 그래프이다.

➢ **A graph that has the maximum number of distinct edges**

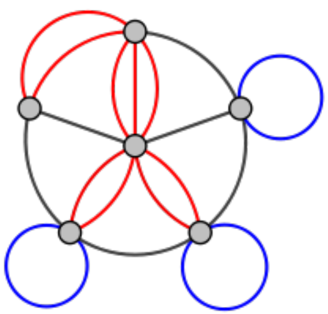
- **Undirected graph with n vertices, maximum number of distinct edges** $= n(n-1)/2$
- **Directed graph with n vertices, maximum number of distinct edges** $= n(n-1)$



- **Multi-graph**

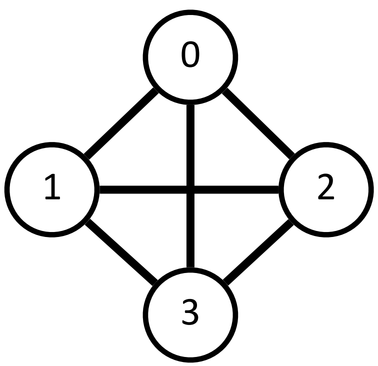
➢ 두 꼭짓점 사이에 여러 변이 허용되는, 그래프의 일반화

➢ **A graph whose edges are unordered pairs of vertexes, and the same pair of vertexes can be connected by multiple edges**



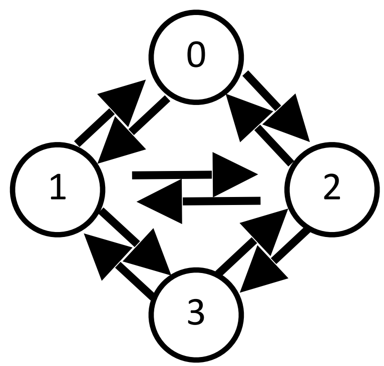
Terminology 1 - example

G_1



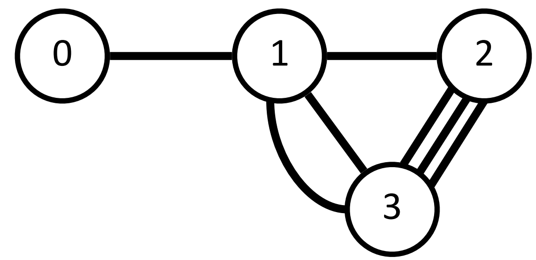
6 distinct edges

G_2



10 distinct edges

G_5



7 edges

Terminology 2

- 인접 (adjacent)
  - ➢ 무방향성 그래프에서 정점 u, v에 대하여 간선(u, v)이 있으면 정점 u는 정점 v에 인접 (adjacent) 하다고 한다. (역도 성립)
  - ➢ 방향성 그래프에서 정점 u, v에 대하여 간선(u,v)이 있으면 정점 v는 정점 u에 인접한다고 한다. (역은 성립하지 않음.)
- 부속 (incident)
  - ➢ 무방향 그래프에서 정점 u,v에 대하여 간선(u, v)이 있으면 간선(u, v)은 정점 u와 v에 부속 (incident)한다고 합니다.

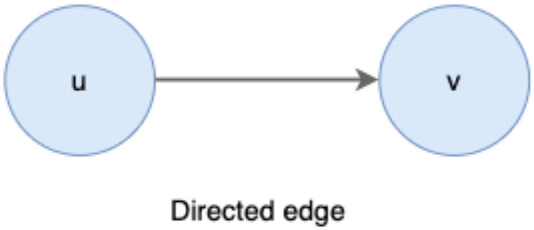
TF

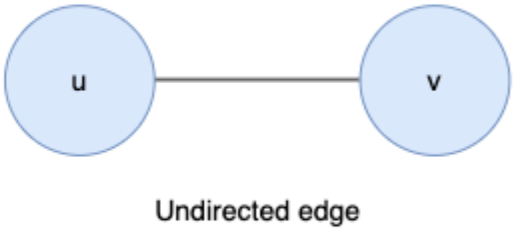
v는 v에 인접한다.

Directed edge Undirected edge

Terminology 2

- 방향성(유향) 간선 **(Directed edge)**
  - 방향을 가진 정점의 쌍 **(u, v)**으로 화살표로 표현하고 단방향을 가리킵니다.
  - 첫 번째 정점 **u**는 출발점을 의미하고 두 번째 정점 **v**는 도착점을 의미합니다.
  - 방향성 간선을 가진 그래프를 방향성 그래프**(Directed Graph)**라고 합니다.
- 무방향성(무향) 간선 **(Undirected edge)**
  - 방향이 없는 정점의 쌍 **(u, v)**으로 직선으로 표현한다.
  - 무방향성 간선 **(u, v)**와 **(v, u)**는 같다. (양방향을 가리킴)
  - 무방향성 간선을 가진 그래프를 무방향성 그래프**(Undirected Graph)**라고 합니다.





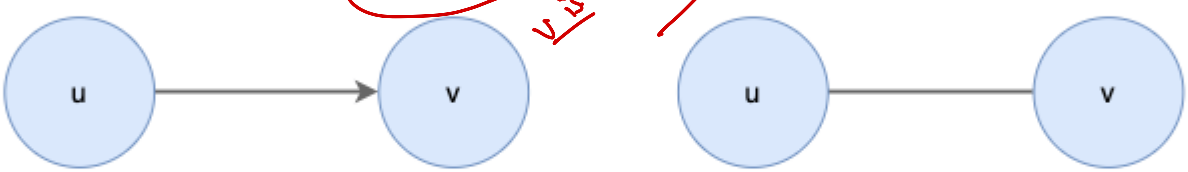
Terminology 2

- If $(u\ ,\ v)$ is an edge of an undirected graph,
  - ➢ The vertices u and v are adjacent
  - ➢ The edge $(u\ ,\ v)$ is incident on u and v

- If $< u\ ,\ v >$ is a directed edge,
  - ➢ The vertex u is adjacent to v
  - ➢ The vertex v is adjacent from u
  - ➢ The edge $< u\ ,\ v >$ is incident on u and v

v is adjacent from v.

v is adjacent from v. ...



Directed edge

Undirected edge

Terminology 3

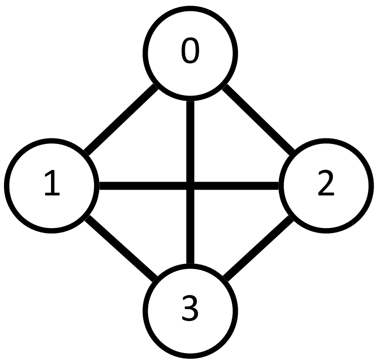
- A subgraph of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$

G

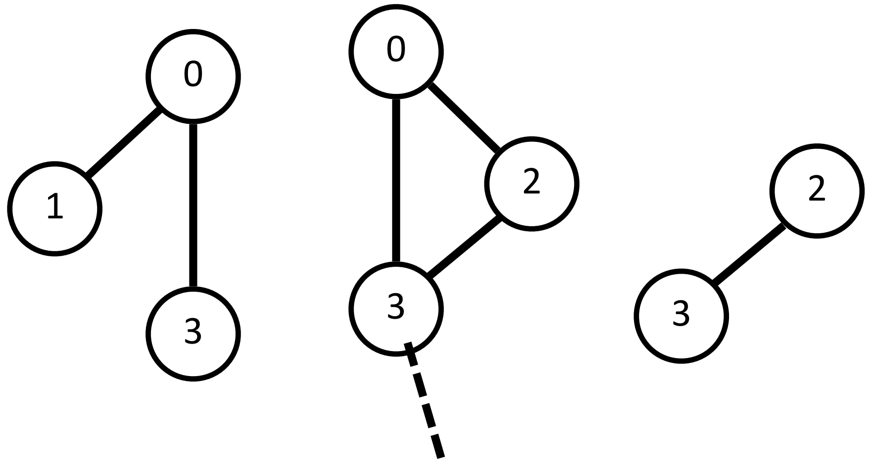
$V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$

Some subgraphs of $G1$

G_1



$\supseteq$



clique

Terminology 4

- Path: 두 vertex간에 Edge로서 연결되는 Vertex sequence
- A path from vertex u to v in graph G is a sequence of vertices, $u, i_1, i_2, \ldots, i_k, v$ such that $(u, i_1), (i_1, i_2), \ldots, (i_k, v)$ are edges in an undirected graph
- If G is a directed graph, the path consists of $\langle u, i_1 \rangle, \langle i_1, i_2 \rangle, \ldots, \langle i_k, v \rangle$ edges
- The length of a path is the number of the edges on it

→ 같은 정점은 여러번 x.

- A simple path is a path in which all vertices, except possibly the first and the last, are distinct (does not have a cycle and multiple edges)
- A cycle is a simple path in which the first and the last vertices are the same

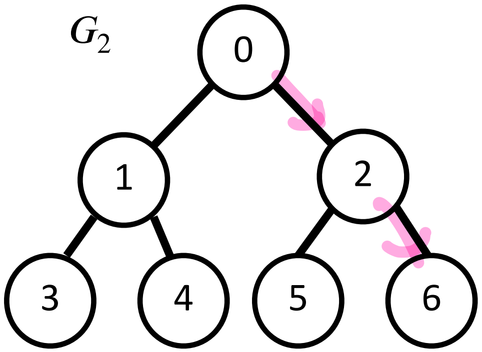
simplepath는 path인데 first와 last의 정점 말고는 다름. cycle이 없는 path를 말한다

↓

cycle은 simplepath인데 first와 last의 정점이 같은.

Terminology 4 – Examples

G_2

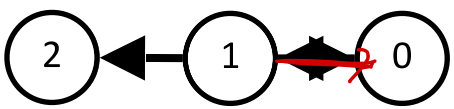


$V(G_2) = \{0, 1, 2, 3, 4, 5, 6\}$

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

Simple path: [0, 2, 6], [0, 1, 4], …

G_3



$V(G_3) = \{0, 1, 2\}$

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

Simple path: [0,1,2], [1,0], [1,2]

Cycle: [0, 1, 0]

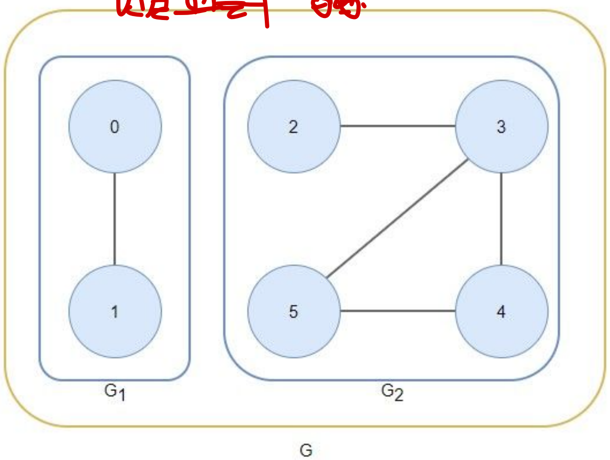
Terminology 5

connected component는 서로 중복 안되는 maximal connected subgraph 의 집합이라고 할 수 있다

서로 중복되지 않는 연결된 부분 그래프

- **Connected:** 두 vertex간에 path가 존재함을 의미
  - 연결 요소란 서로 중복되지 않는 연결된 부분 그래프 (그림예시)
  - 그래프가 연결돼있지 않다면 그래프는 연결 요소들의 집합으로 구성됩니다.
  - 연결된 그래프는 하나의 연결 요소만 가지고 있음
- **In an undirected graph G, two vertices u and v are connected if there is a path in G from u and v**
- **A connected component or simply a component of an undirected graph is a maximal connected subgraph**
  - **Let $G = (V, E)$ be a graph and $G1 = (V1, E1) \ldots, G_m = (Vm,\ Em)$ be its connected components**
  - $V_i \cap V_j = \{\emptyset\}$, **for** $\forall i,\ j$
  - **Further,** $V = V_1 \cup \ldots \cup V_m$ **and** $E = E_1 \cup \ldots \cup E_m$
- **A tree is a graph that is connected and acyclic**

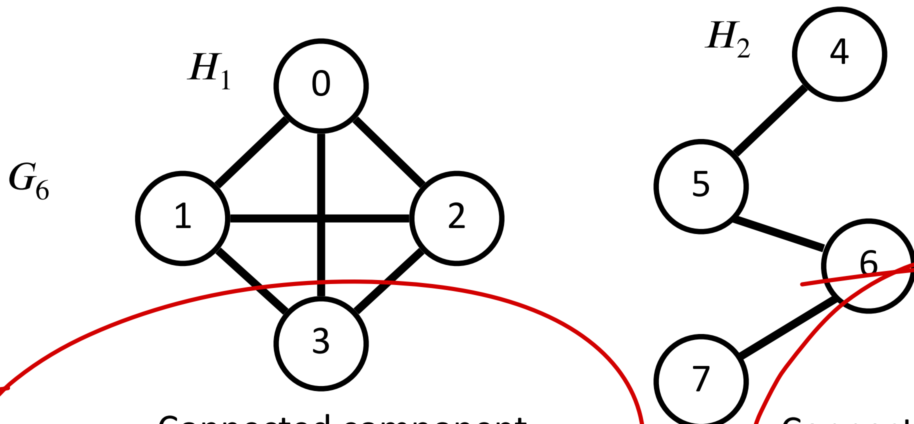
연결 요소들의 집합



$V_i \cap V_j = \{\emptyset\}$ for $\forall i, j$

Terminology 5

$$V(G_6) = \{0, 1, 2, 3, 4, 5, 6, 7\}$$
$$E(G_6) = \{(0,1), (0,2), (1,2), (1,3), (2,3), (4,5), (5,6), (6,7)\}$$



Connected component

Connected component

$$V(H_1) = \{0, 1, 2, 3\}$$
$$E(H_1) = \{(0,1), (0,2), (1,2), (1,3), (2,3)\}$$

$$V(H_2) = \{4, 5, 6, 7\}$$
$$E(H_1) = \{(4,5), (5,6), (6,7)\}$$

Terminology 6

strongly connected 란 V(G)에서 모든 정점들의 집합 u, v 끼리 오고갈 수 있어야 한다.

- **Strongly connected component:** 강하게 결합된 정점 집합
- **A directed graph is strongly connected if, for every pair of vertices, u and v in $V(G)$, there is a directed path from u to v and also from v to u**

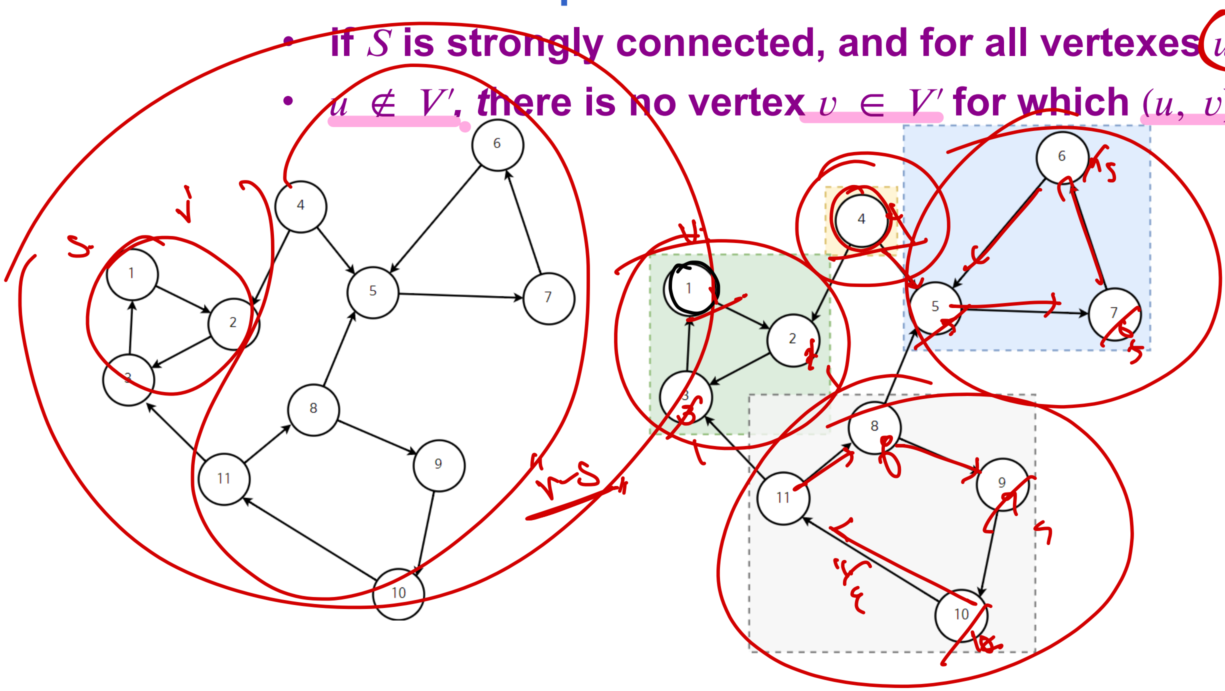
SCC는 maximal-subgraph 인데 strong connected된 것을 말한다.

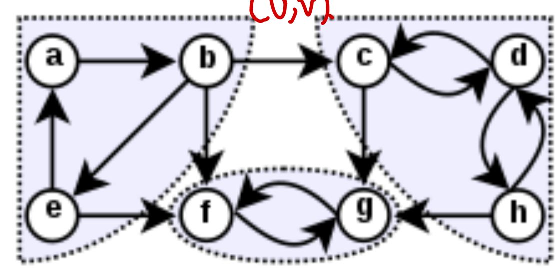
maximal subgraph

- **A strongly connected component is a maximal subgraph that is strongly connected**
  - ➢ **Given a directed graph $D = (V,\ E)$, a subgraph $S = (V',\ E')$ is a strongly connected component:**
    - **If S is strongly connected, and for all vertexes u such that $u\ \in\ V$ and**
    - **$u\ \notin\ V'$, there is no vertex $v\ \in\ V'$ for which $(u,\ v)\ \in\ E$**

$u \in V$ and $v \in V'$

(u, v)





https://en.wikipedia.org/wiki/Strongly_connected_component

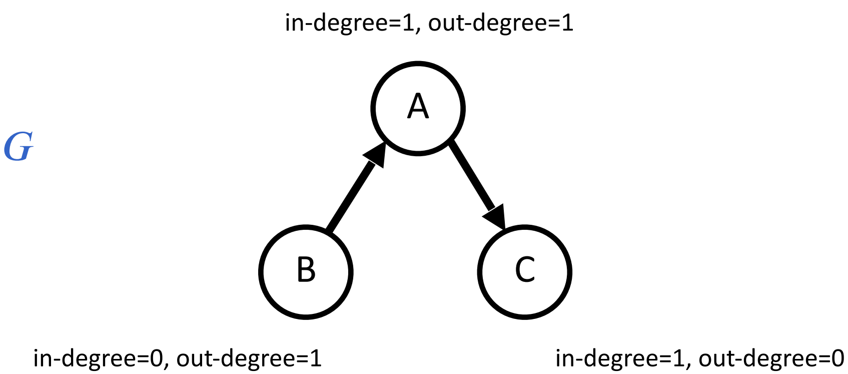
Terminology 7

- **The degree of a vertex is the number of edges incident to that vertex**
  - **Vertex**와 연결된 **edge**의 수
- **For a directed graph,**
  - **the in-degree of a vertex v is defined as the number of edges that have v as the head (vertex** 기준 들어오는 방향**)**
  - **the out-degree of a vertex v is defined as the number of edges that have v as the tail (vertex** 기준 나가는 방향**)**
- **Property**

  e **: the number of edges**

  di **: the degree of a vertex i in graph G**

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



Representing Graphs

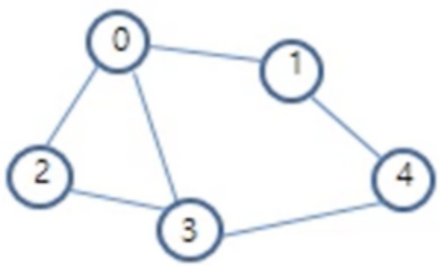
- There are three most commonly used methods to represent graph
  - ➢ 실제 컴퓨터상에서는 우리가 보는것처럼 그대로 표현하는것이 불가능하다

1. Adjacency matrix
2. Adjacency list
3. Sequential list

- The choice of a particular representation will depend on the application type, the type of dominant operations.

Adjacency Matrix

- 방향성이 없는 경우 (Graph)
  - ➢ **Vertex**에 대한 정방향 **2**차원 배열 생성하고, 각 **vertex**에 대한 **adjacent**한 **vertex**를 **1,** 그렇게 않은것을 **0**으로 할당
  - ➢ 대각선 기준으로 대칭하게 **matrix**가 나타난다

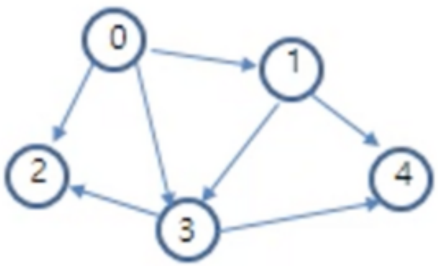


	0	**1**	**2**	**3**	**4**
0	0	1	1	1	0
1	1	0	0	0	1
2	1	0	0	1	0
3	1	0	1	0	1
4	0	1	0	1	0

Adjacency Matrix

■ 방향성이 존재하는 경우 **(DiGraph)**

➢ 한쪽을 **to,** 나머지 한쪽을 **from**으로 설정해서 방향에 대한 **adjaceny**가 존재하는 경우 **1,** 존재하지 않는 경우 **0**으로 **matri**값 할당

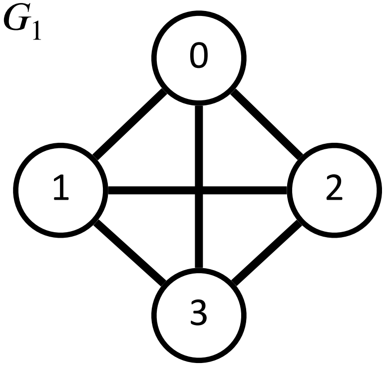


	0	**1**	**2**	**3**	**4**
0	0	1	1	1	0
1	0	0	0	1	1
2	0	0	0	0	0
3	0	0	1	0	1
4	0	0	0	0	0

Adjacency Matrix

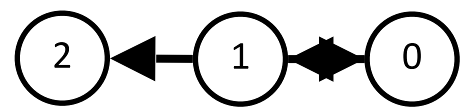
- For a graph with n number of nodes, the adjacency matrix is a $n \times n$ matrix such that
  - $a[i,\ j] = \begin{cases} 1 & \text{if } (i, j) \text{ is in } E(G) \\ 0 & \text{else} \end{cases}$

G_1



$$\begin{array}{c} \\ [0] \\ [1] \\ [2] \\ [3] \end{array} \begin{array}{c} \begin{array}{cccc} [0] & [1] & [2] & [3] \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

G_3



$$\begin{array}{c} \\ [0] \\ [1] \\ [2] \end{array} \begin{array}{c} \begin{array}{ccc} [0] & [1] & [2] \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Property of Adjacency Matrix

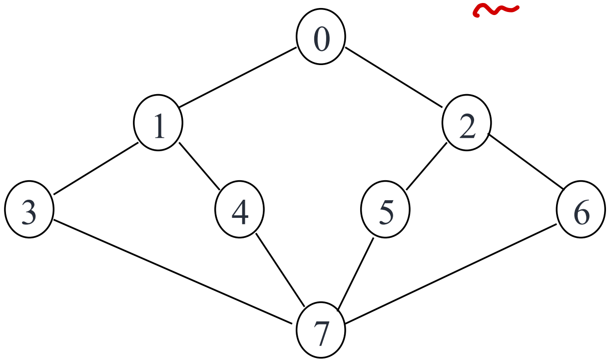
- The adjacency matrix for an undirected graph is symmetric
- The adjacency matrix for a digraph need not be symmetric
- For an undirected graph, the degree of any vertex i is its row sum:
  - $\sum_{j=0}^{n-1} a[i][j]$
- For a directed graph
  - the row(행) sum is the out-degree
  - the column(열) sum is the in-degree
- The space needed to represent a graph is n^2 bits
- For undirected graphs, the lower (or upper) triangular can be stored to save space

Inefficiency of Adjacency Matrix

- How many edges are there in graph G?
- Is G connected?
- Solution: There are $n^2 - n$ entries (excluding diagonals)
- Hence, time complexity is $\Theta(n^2)$

$n^2 - n$

G



–	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	0	1	1	0	0	0
2	1	0	0	0	0	1	1	0
3	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	0	1	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	0	1	1	1	1	0

Denseness = 20/64 = 31(%)

sparse matrix

Advantage of Adjacency Matrix

- **Many. Especially in performing machine learning on graphs.**
- **An example – the power of an adjacency matrix**
  - ➢ **How many paths of length n are there between a vertex i and j**
    - # of paths of length n between i, j $= A^n$

verts. i j

length n

A^n

$A =$

–	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	0	1	1	0	0	0
2	1	0	0	0	0	1	1	0
3	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	0	1	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	0	1	1	1	1	0

$A^2 =$

–	0	1	2	3	4	5	6	7
0	2	0	0	1	1	1	1	0
1	0	3	1	0	0	0	0	2
2	0	1	3	0	0	0	0	2
3	1	0	0	2	2	1	1	0
4	1	0	0	2	2	1	1	0
5	1	0	0	1	1	2	2	0
6	1	0	0	1	1	2	2	0
7	0	2	2	0	0	0	0	4

Advantage of Adjacency Matrix

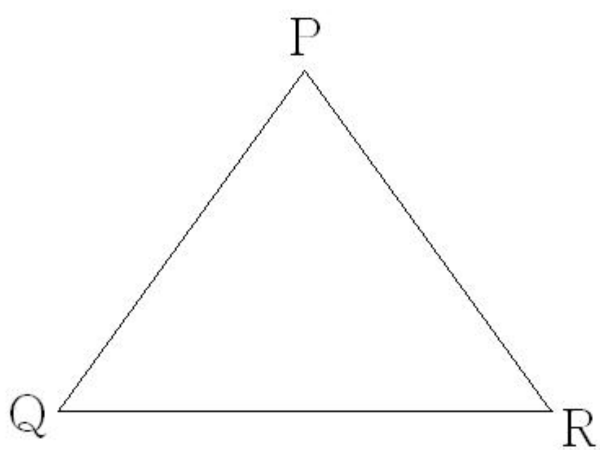
- A^2의 의미는?
  - ➢ PPPP + PQQP + PRRP

$$\begin{array}{cc} & \begin{array}{ccc} P & Q & R \end{array} \\ \begin{array}{c} P \\ Q \\ R \end{array} & \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

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

$$\begin{pmatrix} P\to P & P\to Q & P\to R \\ Q\to P & Q\to Q & Q\to R \\ R\to P & R\to Q & R\to R \end{pmatrix} = \begin{pmatrix} PP & PQ & PR \\ QP & QQ & QR \\ RP & RQ & RR \end{pmatrix}$$

$$\begin{array}{cc} & \begin{array}{ccc} P & Q & R \end{array} \\ \mathrm{A}= \begin{array}{c} P \\ Q \\ R \end{array} & \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$



$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Advantage of Adjacency Matrix

- **Another example – fast query O(1) for checking if an edge exists between vertex i and j**

$A =$

–	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	0	1	1	0	0	0
2	1	0	0	0	0	1	1	0
3	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	0	1
5	0	0	1	0	0	0	0	1
6	0	0	1	0	0	0	0	1
7	0	0	0	1	1	1	1	0

If $a[i, j] == 1$, then edge exists.
No edge otherwise.

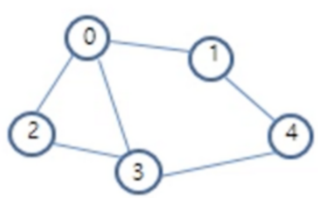
O(1) operation

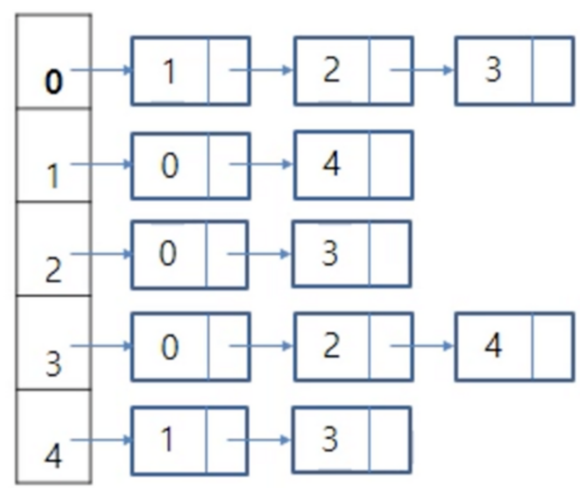
2) Adjacency Lists →인접 리스트.

- Vertex에 대해 adjacent한 다른 vertex들을 lined list형태로 표현
- When Graphs are sparse, the previous questions can be answered in $O(E + V)$ using adjacency lists
  - Only the edges that are in G are explicitly stored (only cells with 1s)
  - Replace n rows of adjacency matrix with n linked lists
  - Every vertex i in G has a list
  - The nodes in chain i represent the vertices that are adjacent from i
  - The vertices in each chain are not required to be ordered
  - The data field of a chain node stores the index of an adjacent vertex
  - An array(adlist[]) is used to access the adjacency list for an vertex in $O(1)$ time
  - Adlist[i] is a pointer to the first node in the adjacency list for vertex i

$\boxed{i}$

$\boxed{0} \to 1$

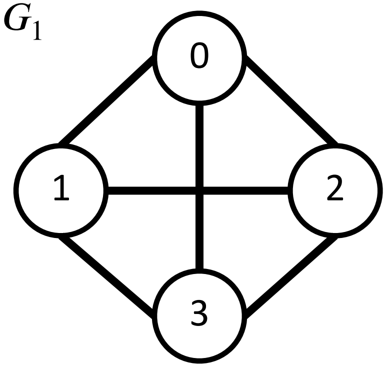




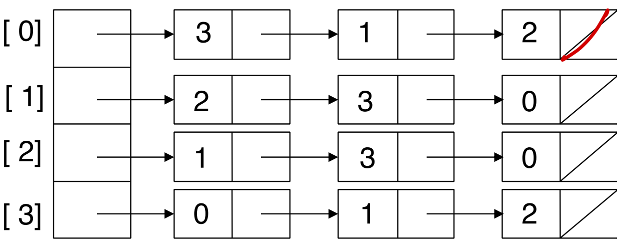
2) Adjacency Lists

Graph | Adjacency matrix | Adjacency list

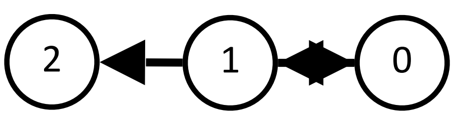
G_1



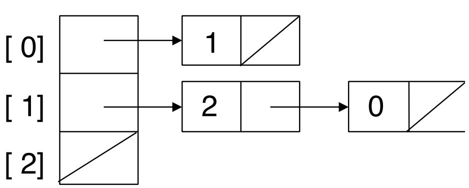
$$\begin{array}{c|cccc} & [0] & [1] & [2] & [3] \\ \hline [0] & 0 & 1 & 1 & 1 \\ [1] & 1 & 0 & 1 & 1 \\ [2] & 1 & 1 & 0 & 1 \\ [3] & 1 & 1 & 1 & 0 \end{array}$$



G_3



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



3) Sequential list Representation(1-D array)

n V+1 +2xe

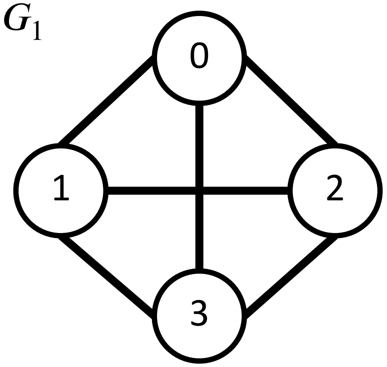
- The adjacency list can be represented as a simple 1D array L
- For a graph with n vertices and e edges, the length of the sequential list is
  - ➢ $|L| = n + 1 + 2 * e$

$|L| =$ "$n+1+2e$"

$n+1 + 2 \times n$

$2e+n+1$

G_1



$n = 4$
$e = 6$

index	pointer
0	5
1	8
2	11
3	14
4	17
5	1
6	2
7	3
8	0
9	2
10	3
11	
12	
13	
…	…

Starting position of adjacent nod list

Adjacent nodes of vertex 0

Adjacent nodes of vertex 1

* The vertices adjacent from vertex i are stored in $L[i]$ and end at $L[i+1] - 1$