# 그래프
# (Graph)

- 그래프 정의
- 용어 설명
- 그래프 표현

# 1. 그래프 정의

- 그래프(graph) 혹은 무향그래프(undirected graph) G는 다음의 두 집합 V, E로 정의된다: G = (V, E)

  V: 정점(vertex)들의 집합 (정점 대신에 노드를 사용하기도 함)

  E: 에지(edge, 간선)들의 집합

  E의 원소(에지 혹은 간선)는 정점들의 쌍 (순서를 고려하지 않음)

  => E의 원소(에지 혹은 간선)는 정점 쌍 (v,w) 로 나타냄

- 방향그래프(유향그래프, directed graph 혹은 digraph) G는 두 집합 V, E로 정의된다: G = (V, E)

  V: 정점들의 집합

  E: 에지들의 집합

  E의 원소(에지(유향에지) 혹은 간선)는 정점들의 순서 쌍

  => E의 원소(에지(유향에지) 혹은 간선)는 정점들의 순서쌍 (v,w)으로 나타냄 (순서고려)

# 그래프 정의

- 그래프의 graphical representation

  vertex u는 u○로

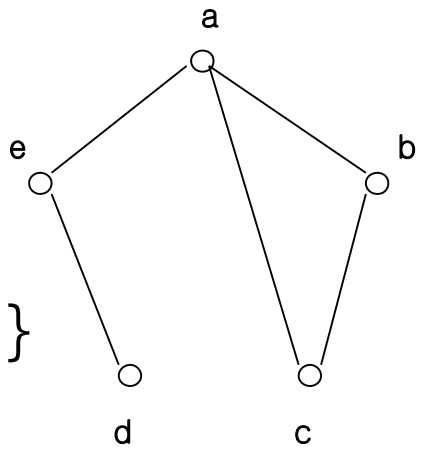
  (무향)그래프의 에지 (u,v)는 u ○——○ v로

  방향그래프의 에지 (u,v)는 u ○——→○ v로

- 예: 그래프 G = (V, E)

  V = {a, b, c, d, e}

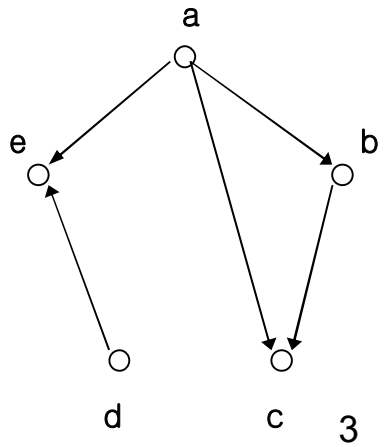
  E = {(a,b), (b,c), (d,e), (a,e), (a,c)}




- 예: 방향그래프 G = (V, E)

  V = {a, b, c, d, e}

  E = {(a,b), (b,c), (d,e), (a,e), (a,c)}



# 그래프 정의

- 쾨니히스베르크( Königsberg)의 다리문제

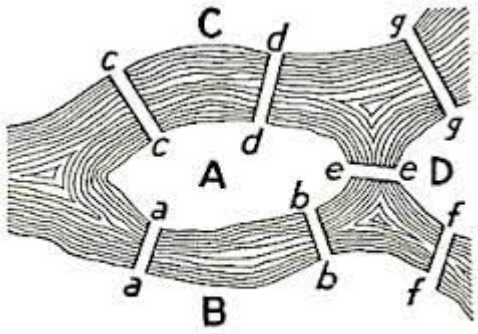
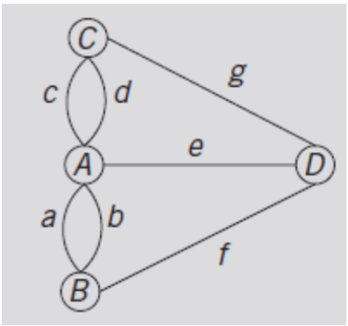



FIGURE 98. *Geographic Map: The Königsberg Bridges.*




- 프로이센의 쾨니히스베르크에는 프레겔 강이 흐르는데, 이 강에는 안쪽에 두 개의 큰 섬과 각 섬을 연결하는 총 7개의 다리가 있었다. 이때 7개의 다리들을 한 번씩만 건너면서 처음 시작한 위치로 돌아오는 길이 존재하는가?
- 오일러(Euler)가 그러한 길이 존재하지 않음을 증명하였다.
- 이 문제는 평면그래프에서의 한붓그리기 문제로 그래프이론의 시초

오일러는 모든 정점의 짝수개의 차수를 갖는다면
모든 다리를 한번씩만 건너서
도달하는 것이 성립한다고 말함

# 그래프 정의

Euler Path : 꼭 시작 지점으로 돌아오지 않아도 됨.
Euler Cycle : 꼭 시작 지점으로 돌아와야 함.

- 그래프의 응용분야

(1) 도로망:

도시 – vertex

두 도시를 직접 연결하는 도로가 있으면 두 도시사이의 에지 edge

(2) 컴퓨터 망:

컴퓨터 – vertex

두 컴퓨터 사이에 통신 link가 있으면 에지

(3) 웹그래프 (Web Graph):

web page – vertex

web page A에서 web page B로의 하이퍼링크가 있으면 방향에지 edge

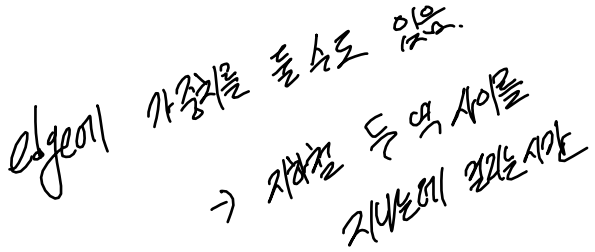
(4) Social Networks

개인 – vertex

개인들 사이의 관계 – 에지

# 그래프 정의

- 그래프와 관련한 문제들
  - 도로망에서 도시 A로부터 갈수 있는 모든 도시들 찾기
  - 도로망에서 도시 A에서 도시 B까지 가장 빨리 가는 경로 찾기
  - …

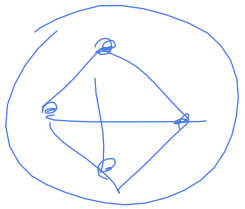


# 2. 그래프 용어

- 그래프 G = (V,E)의 부그래프(subgraph)

  V' ⊆V and E'⊆ E인 그래프 G' = (V', E')를 G의 부그래프라고 함

- 완전그래프(A complete graph): 모든 두 정점 사이에 에지가 있는 그래프



$K_n$ n은정점수

$K_4$

- v 와 w가 인접하다 (v is adjacent to w) if (v,w) ∈E
- 에지 (v,w)는 정점 v, w에 부속 (incident)되어있다.
- 정점 v로부터 정점 w까지의 경로(path)

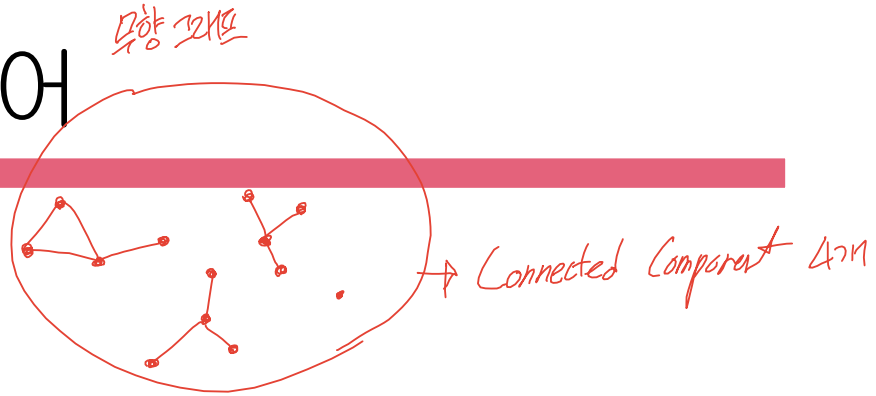
v = $v_0$이고 $v_k$ = w인 일련의 정점들 ($v_0$, $v_1$, …, $v_{k-1}$,$v_k$)로서 다음 조건을 만족한다: 0 ≤i≤k−1에 대하여, ($v_i$,$v_{i+1}$) ∈ E이다.

이 경로의 길이는 k이다.

$v_0$, $v_1$,…, $v_k$들이 모두 다르면 이 경로를 단순경로(simple path)라 함.

- 정점 v로부터 정점 w까지의 경로가 존재하면 w는 v로부터 도달가능(reachable)하다고 함

# 그래프 용어



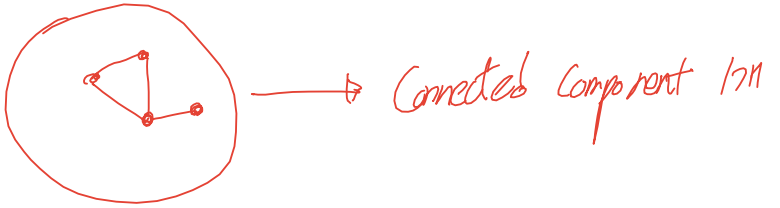

■ 사이클

1) (무향)그래프에서의 사이클 (양방향)

시작 정점과 마지막 정점이 같은 경로 (A → B, B → A 모두 가능)

** 단순사이클: 시작 정점과 마지막 정점이 같은 것을 제외하고

어떤 정점도 중복되지 않는 사이클



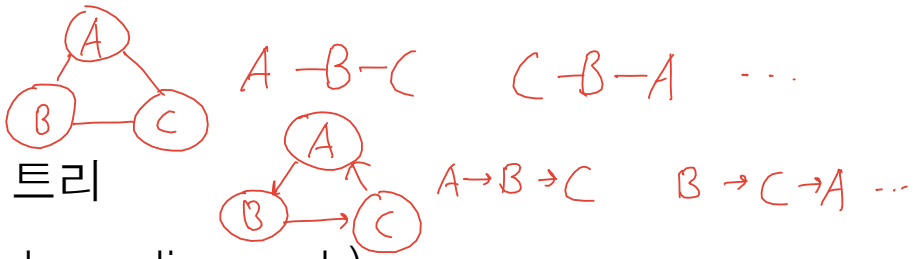

2) 방향그래프에서의 사이클

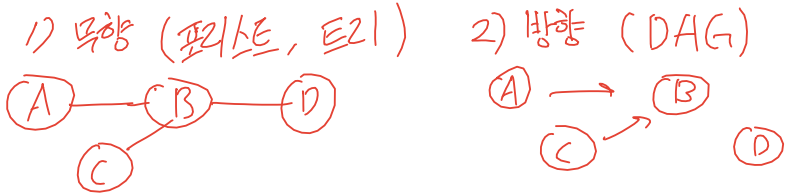
시작 정점과 마지막 정점이 같은 경로

** 단순사이클: 첫째 정점과 마지막 정점이 같은 것을 제외하고

어떤 정점도 중복되지 않는 사이클

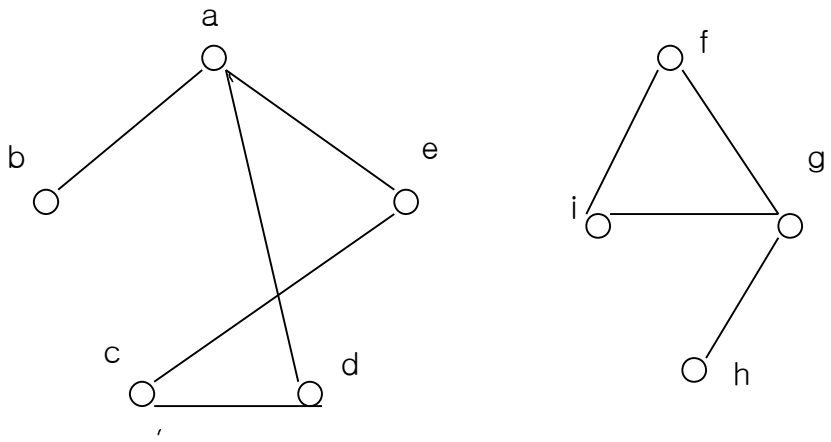
- 사이클이 없는 (무향)그래프: 포리스트
- 사이클이 없으면서 연결된 (무향)그래프: 트리
- 사이클이 없는 방향그래프: DAG (directed acyclic graph)



# 그래프 용어

- 연결그래프(connected graph): 임의의 두 정점 u,v에 대하여, u에서 v까지의 경로가 존재하는 그래프
- 강연결 (유향)그래프 (strongly connected graph): 임의의 두 정점 u,v에 대하여, u에서 v까지의 경로가 존재하는 방향그래프
- 그래프 G의 (연결)성분(connected component): 그래프 G의 최대 연결 부그래프(maximal connected subgraph of G)
- 방향그래프 G의 강연결성분(strongly connected component): 방향 그래프 G의 최대 강연결 부그래프 (maximal strongly connected subgraph of G)




경로의 예: a, e, c, d

사이클의 예: a, d, c, e, a

연결성분:
{a, b, c, d, e}, {f, g, h, i},

# 그래프 용어

- 가중치그래프(weighted graph): 에지에 가중치(weight)가 있는 그래프.
  - (V, E, W)로 정의됨

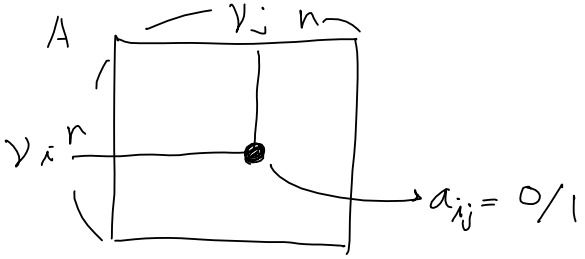
    W는 E에서 실수집합 R로 매핑되는 함수

    에지의 가중치:

    비용, 길이, 용량 등 에지의 성질을 나타냄

# 3. 그래프 표현

- n: |V| (그래프의 정점 수), m: |E| (그래프의 에지 수)
- 그래프의 정점 집합 $V = \{v_0, v_1, \cdots, v_{n-1}\}$라 하자.
- 그래프 G의 n개의 정점들을 0부터 n-1의 정수로 대응시킨다: 정점 $v_i$를 정수 i에 대응시킨다고 가정




**1) 인접행렬 (Adjacency Matrix) 표현 => 2차원 배열로 나타냄**

(1) 그래프 G의 인접행렬 $A = (a_{ij})$ 표현

$a_{ij}$ = 1 , $(v_i, v_j) \in E$인 경우

0 , 그렇지 않은 경우

❖ 무향그래프에 대한 인접행렬은 대칭적이다.

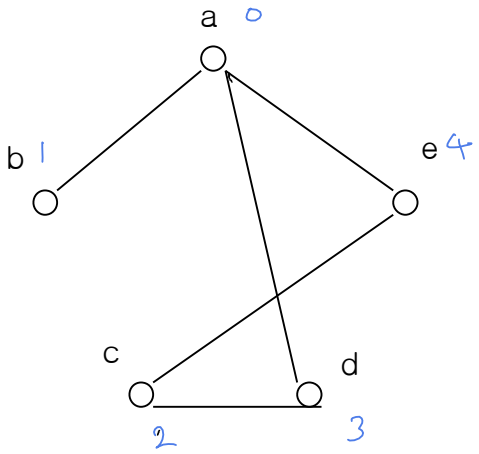
(2) 가중치 그래프 $G = (V, E, W)$의 인접행렬 $A = (a_{ij})$ 표현

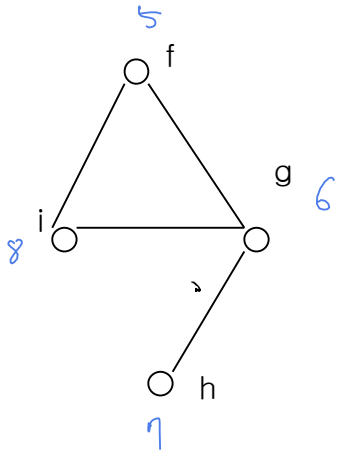
$a_{ij}$ = weight(i, j) , $(v_i, v_j) \in E$ 인 경우 [weight(i, j)는 에지 $(v_i, v_j)$의 가중치임]

c , 그렇지 않은 경우

c는 문제에 따라 0 혹은 ∞

# 그래프 인접행렬 표현 예




n = 9

단, edge가 없는 것(인접하지 않은 것)도 0 이라는 해서의 값으로 포함하고 있다는 점에 유의

인접행렬 표현 예 adjMatrix

| | 0(a) | 1(b) | 2(c) | 3(d) | 4(e) | 5(f) | 6(g) | 7(h) | 8(i) |
|---|---|---|---|---|---|---|---|---|---|
| 0(a) | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1(b) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2(c) | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3(d) | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4(e) | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5(f) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6(g) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 7(h) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8(i) | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

무향 그래프는 대칭적임

# 그래프 표현

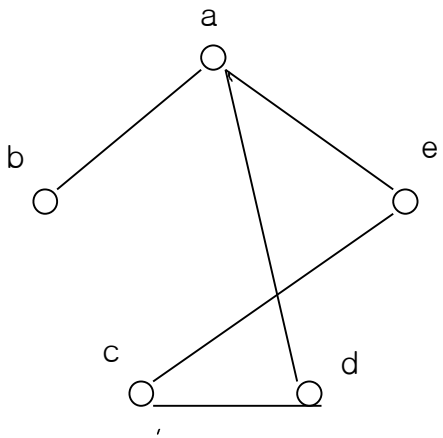
2) 인접리스트 (Adjacency List) 표현

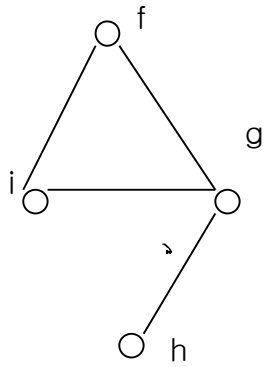
– 각 정점에 대하여, 이 정점과 인접한 모든 정점들을 연결리스트로 만듦

- 리스트의 i번째 원소:

  vi와 인접한 정점들의 연결리스트를 참조함

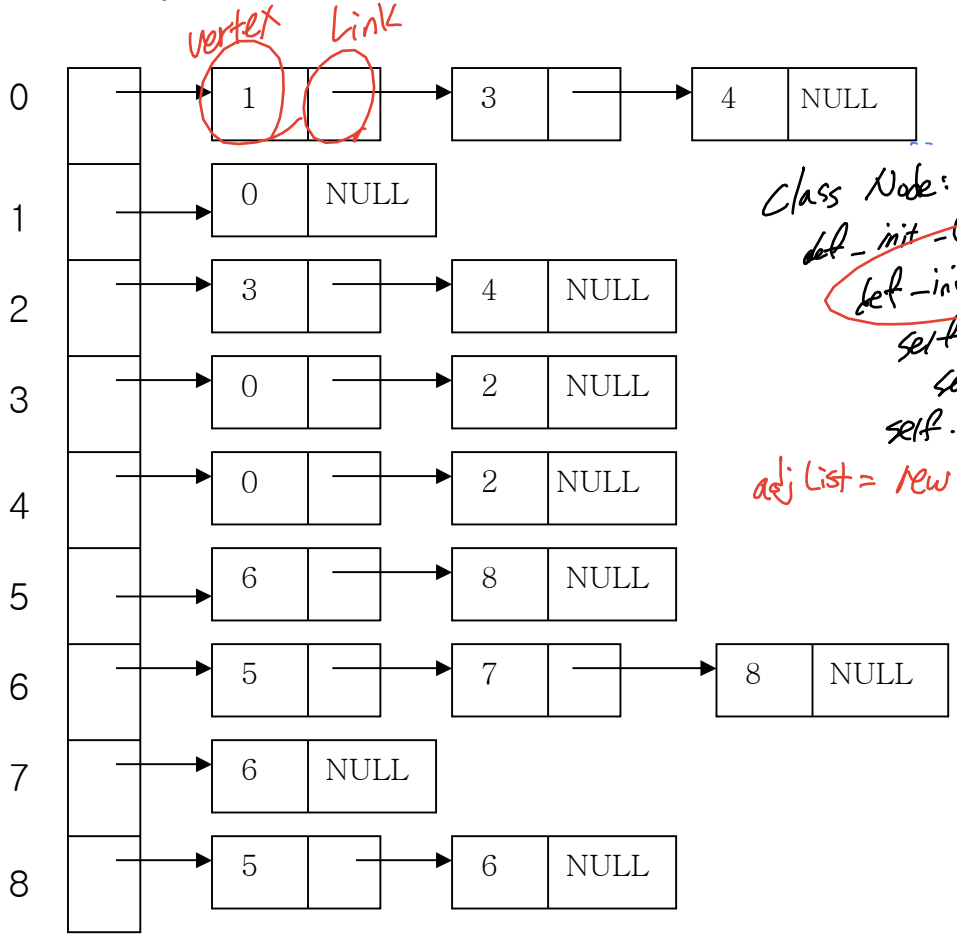
# 인접리스트 표현 예



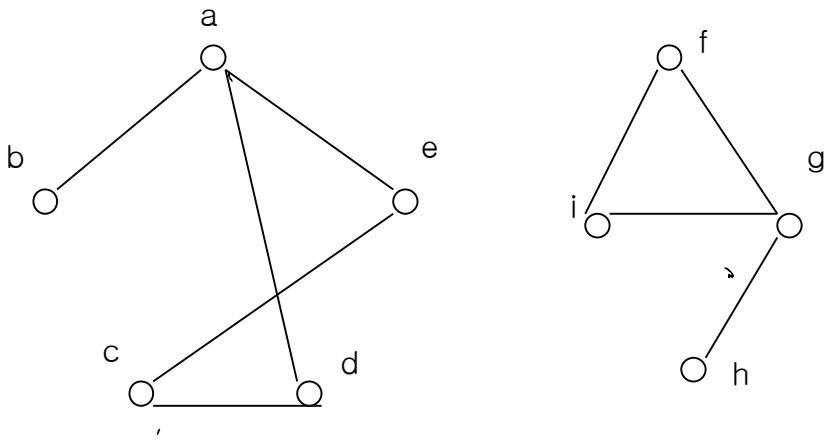

최악의 경우 $O(n^2)$

adjList (연결리스트)




```
Class Node:
  def __init__(self,v):
  def __init__(self,v,wt)
    self.vertex=v
    self.weight=wt
    self.link=None
adjList = new Node*[n];
```

# 인접리스트 표현 예




adjList(C++: vector of vectors)

| | | | |
|---|---|---|---|
| 0 | 1 | 3 | 4 |
| 1 | 0 | | |
| 2 | 3 | 4 | |
| 3 | 0 | 2 | |
| 4 | 0 | 2 | |
| 5 | 6 | 8 | |
| 6 | 5 | 7 | 8 |
| 7 | 6 | | |
| 8 | 5 | 6 | |

adjList(**파이썬**: list of lists)

[[1,3,4], [0], [3,4], [0,2], [0,2], [6,8], [5,7,8], [6], [5,6]]

# 그래프 입력

- 가중치 (에지에 가중치가 있는) 그래프의 입력

입력: n m // 정점 수, 에지 수

$i_1$ $j_1$ // (추가적 입력 $wt_1$) : 에지 ($i_1$, $j_1$)와 가중치

$i_2$ $j_2$ // (추가적 입력 $wt_2$) : 에지 ($i_2$, $j_2$)와 가중치

……

$i_m$ $j_m$ // (추가적 입력 $wt_m$) : 에지 ($i_m$, $j_m$)와 가중치

# 그래프 표현 (인접행렬 1: 파이썬)

- 가중치가 없는 그래프의 인접행렬 표현 # **그래프를 클래스로 정의**

```
class Graph:
    def __init__(self, size):
        self.adjMatrix = [[0 for _ in range(size)] for _ in range(size)]
        self.size = size

    def insertEdge(self, v1, v2):
        self.adjMatrix[v1][v2] = 1
# in case of undirected graph
        self.adjMatrix[v2][v1] = 1

    def printGraph(self):
        for i in range(self.size):
            for j in self.adjMatrix[i]:
                print(j, end = ' ')
            print()

def main():
    n, m =  input().split()
    n, m = int(n), int(m)
    g = Graph(n)
    for i in range(m):
        v1, v2 = input(). split()
        v1, v2 = int(v1), int(v2)
        g.insertEdge(v1,v2)
    g.printGraph()
```

```
if __name__ == '__main__':
    main()
```

# 그래프 표현 (인접행렬 2: 파이썬)

- 가중치가 없는 그래프의 인접행렬 표현: **그래프를 클래스로 정의하지 않음**

```
def main():
    n, m = input().split()
    n, m = int(n), int(m)
    adjMatrix = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(m):
        u, v = input().split()
        u, v = int(u), int(v)
    # adjacency matrix
        adjMatrix[u][v] = 1
        adjMatrix[v][u] = 1



if __name__ == '__main__':
    main()
```

# 그래프 표현 (인접행렬: C)

- 인접행렬 표현

```
int** adjMatrix;     // int adjMatrix[MAX][MAX];
int n, m, v1, v2, wt;
int i, j;
const int c = 0;
scanf("%d %d", &n, &m); // 정점 수와 에지 수 입력
adjMatrix = (int **)malloc(sizeof(int *)*n);
for (i = 0; i < n; i++)
    adjMatrix[i] = (int*)malloc(sizeof(int)*n);

for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
     adjMatrix[i][j] = c;   /* c is determined by the problem */

for(i = 0; i < m; i++)  {
     scanf("%d %d", &v1, &v2);  // 에지 (v1, v2)
//   scanf("%d %d %d", &v1, &v2, &wt);  // 에지 (v1, v2)와 그 가중치 wt를 입력
     adjMatrix[v1][v2] = 1;  // adjMatrix[v1][v2] = wt;
     adjMatrix[v2][v1] = 1;  /* in case of undirected graph  // adjMatrix[v1][v2] = wt; */
}
```

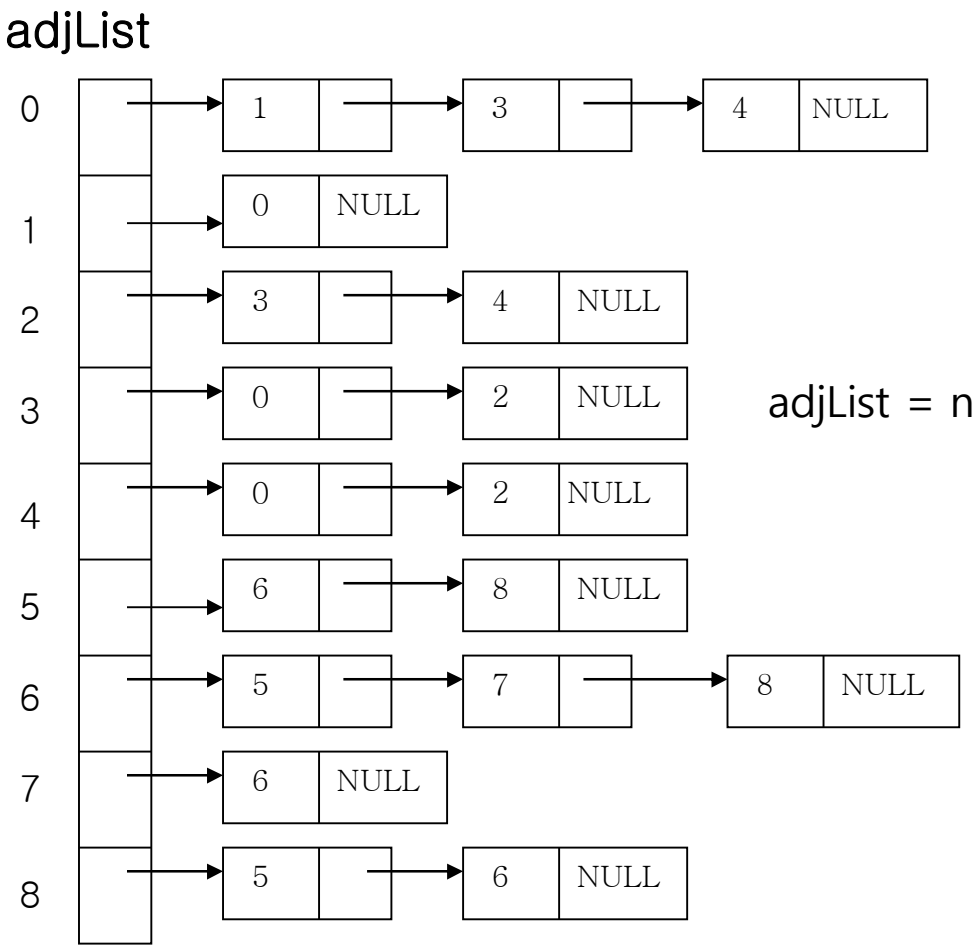
```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
       printf("%d ", adjMatrix[i][j]);
    printf("\n");
}

// 동적메모리 해제
for (i = 0; i < n; i++)
   free(adjMatrix[i]);

free(adjMatrix);
```

# 그래프 표현

- Adjacency List(인접 리스트) 표현




```
class Node:
    def __init__(self, v):
#    def __init__(self, v, wt):
        self.vertex = v
#         self.weight = wt
        self.link = None
```

adjList = new Node*[n];

```
struct Node {
    int  vertex;
    // int weight; //
    struct Node *link;
};
```

# 그래프 표현 (인접리스트 1: 파이썬 연결리스트)

- 가중치가 없는 그래프의 인접리스트표현 : **연결리스트 (클래스로 정의)**

```
class Node:
    def __init__(self, vertex):
        self.vertex = vertex
        self.link = None  # self.next =

class Graph:
    def __init__(self, size):
        self.adjList = [None]*size
        self.size = size

    def insertEdge(self, v1, v2):
        newNode = Node(v2)
        newNode.link = self.adjList[v1]
        self.adjList[v1] = newNode
  # in case of undirected graph

        newNode = Node(v1)
        newNode.link = self.adjList[v2]
        self.adjList[v2] = newNode
```

```
def printGraph(self):
   for v in range(self.size):
       print(v, end = ': ')
       current = self.adjList[v]
       while current is not None:
           print(current.vertex, end = ' ')
           current = current.link
       print()

def main():
    n, m =  input().split()
    n, m = int(n), int(m)
    g = Graph(n)
    for i in range(m):
        v1, v2 = input(). split()
        v1, v2 = int(v1), int(v2)
        g.insertEdge(v1,v2)
    g.printGraph()


if __name__ == '__main__':
    main()
```

# 그래프 표현 (인접리스트 2: 파이썬 연결리스트)

- 가중치가 없는 그래프의 인접리스트표현 : **연결리스트 (클래스로 정의하지 않음)**

```
class Node:
    def __init__(self, vertex):
        self.vertex = vertex
        self.link = None

def main():
    n, m =  input().split()
    n, m = int(n), int(m)
    adjList = [None for _ in range(n)]

    for i in range(m):
        v1, v2 = input(). split()
        v1, v2 = int(v1), int(v2)
        newNode = Node(v2)
        newNode.link = adjList[v1]
        adjList[v1] = newNode
        newNode = Node(v1)
        newNode.link = adjList[v2]
        adjList[v2] = newNode

if __name__ == '__main__':
    main()
```

인접배열 n개만큼 None 리스트 adjList 생성.

Node(v2)

# 그래프 표현 (인접리스트 3: 파이썬 list of lists)

- 가중치가 없는 그래프의 인접리스트표현 : list of lists 이용 (클래스로 정의)

```
class Graph:
    def __init__(self, size):
        self.adjList = [[] for _ in range(size)]
        self.size = size

    def insertEdge(self, v1, v2):
        self.adjList[v1].append(v2)
  # in case of undirected graph
        self.adjList[v2].append(v1)
    def printGraph(self):
        for v in range(self.size):
            print(v, end = ': ')
            for x in self.adjList[v]:
                print(x, end = ' ')
            print()
```

```
[[1,3,4],[0],[3,4],[0,2],[0,2],[6,8],…]
```

```
n, m = input().split()
n, m = int(n), int(m)
g = Graph(n)
for i in range(m):
    v1, v2 = [int(x) for x in input().split()]
    g.insertEdge(v1,v2)
g.printGraph()
```

# 그래프 표현 (인접리스트 4: 파이썬 list of lists)

- 가중치가 없는 그래프의 인접리스트표현 : list of lists 이용 (클래스로 정의하지 않음)

```
n, m = input().split()
n, m = int(n), int(m)
adjList = [ [] for i in range(n)]

for i in range(m):
    v1, v2 = input().split()
    v1, v2 = int(v1), int(v2)
    adjList[v1].append(v2)
    adjList[v2].append(v1)
```

```
[[1,3,4],[0],[3,4],[0,2],[0,2],[6,8],…]
```

# 그래프 표현 (인접리스트: C++)

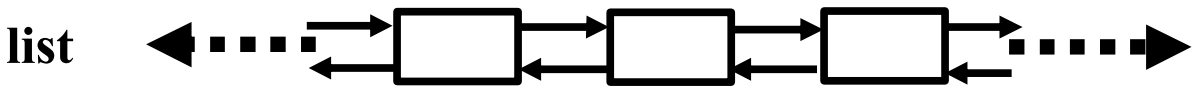
- Adjacency List(인접 리스트) 표현 – C++ STL 이용하여 만들 수 있다:
  : Vector of vectors 혹은 Vector of lists

- Vector of vectors

| | | | |
|---|---|---|---|
| 0 | 1 | 3 | 4 |
| 1 | 0 | | |
| 2 | 3 | 4 | |
| 3 | 0 | 2 | |
| 4 | 0 | 2 | |
| 5 | 6 | 8 | |
| 6 | 5 | 7 | 8 |
| 7 | 6 | | |
| 8 | 5 | 6 | |

- Vector of lists

  STL list: doubly linked list



# 그래프 표현 (인접리스트: C++ 연결리스트 )

- 인접리스트 표현

```
cin >> n >> m;    // n: the number of vertices, m: the number of edges
Node** adjList;
adjList = new Node*[n];
// adjList = (Node**) malloc(sizeof(Node*)*n);
for(i = 0; i < n; i++)
   adjList[i] = NULL;
for(i = 0; i < m; i++) {
   int v1, v2, wt;
   Node * ptr;
   cin  >> v1 >> v2 >> wt;
   ptr = new Node;
 //  ptr = (Node *) malloc(sizeof(Node)); //
   ptr->vertex = v2;     // ptr->weight = wt;
   ptr->link = adjList[v1];
   adjList[v1] = ptr;
 // * in case of undirected graph */
   ptr = new Node;
   ptr->vertex = v1;    // ptr->weight = wt;
   ptr->link = adjList[v2];
   adjList[v2] = ptr;
```

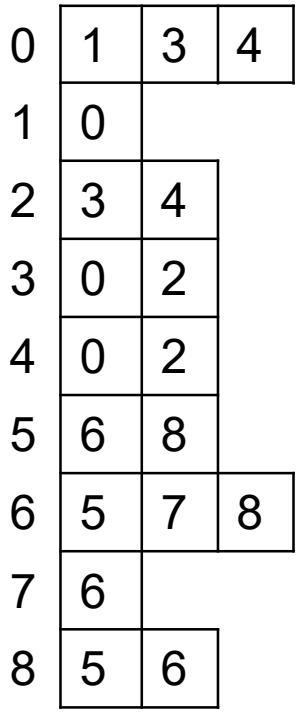
# 그래프 표현 (인접리스트: C++ 연결리스트)

```
// 인접리스트 표현 결과 확인
     for (int i = 0; i < n; i++) {
          Node *ptr = adjList[i];
          cout << "adjacencyList[" << i  << "]: ";
          while (ptr) {
               cout << ptr->vertex << "   ";
               ptr = ptr->link;
          }
          cout << endl;
     }
// 동적메모리 해제
     for (i = 0; i < n; i++){
               Node *ptr;
               while (adjList[i] != NULL){
                         ptr = adjList[i];
                         adjList[i] = ptr->link;
                         delete ptr;
               }
     }
     delete [] adjList;
```

# 그래프 표현 (인접리스트: C++ vector of vectors

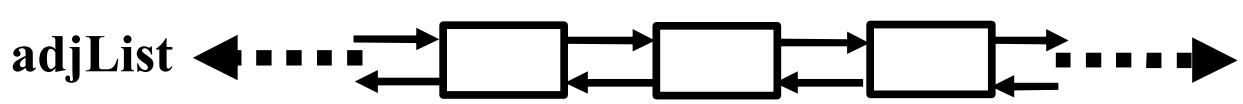
- 가중치가 없는 그래프의 인접리스트 표현 // C+ STL vector of vectors 이용

```
#include <vector>
#include <list>
cin >> n >>m;
vector<vector<int> > adjList(n);
for (int i = 0; i < m; ++i) {
     cin >> v1 >> v2;;
     adjList[v1].push_back(v2);
   //  In case of undirected graph
     adjList[v2].push_back(v1);
}
cout << "\nThe Adjacency List\n";
   // Printing Adjacency List
for (int i = 0; i < adjacencyList.size(); ++i) {
     cout <<"adjacencyList["<< i << "]: ";
     vector<int>::iterator itr =  adjList[i].begin();
     while (itr != adjList[i].end()) {
        cout << *itr  << " ";
        ++itr;
     }
     cout << endl;
}
```



# 그래프 표현 (인접리스트: C++ vector of lists)

- 가중치가 있는 그래프의 인접리스트 표현 // C+ STL vector of lists – doubly linked list이용




```
#include <vector>
#include <list>
Cin >> n >> m;
vector< list< pair<int, int> > > adjList(n);
for (int i = 0; i < m; ++i) {
     cin >> v1 >> v2 >> w;  // 에지 (v1, v2)와 그 가중치 w
     adjList[v1].push_back(make_pair(v2, weight));
   //  In case of undirected graph
     adjList[v2].push_back(make_pair(v1, weight));
}
cout <<"\nThe Adjacency List\n";
   // Printing Adjacency List
for (int i = 0; i < adjList.size(); ++i) {
     cout << "adjacencyList[" << i << "]: ";

     list< pair<int, int> >::iterator itr =  adjList[i].begin();
     while (itr != adjList[i].end()) {
          cout << (*itr).first << "(" << (*itr).second) << ")  ";
          ++itr;
     }
     cout << endl;
}
```