

Data Structure

자료구조



그래프



한국기술교육대학교
온라인평생교육원

학습내용

- 그래프의 개념
- 그래프의 표현 방법

학습목표

- 그래프의 개념을 설명할 수 있다.
- 인접 행렬과 인접 리스트를 이용해 그래프를 표현할 수 있다.



그래프의 개념

그래프 Graph 그래프의 개념

1 그레프란?

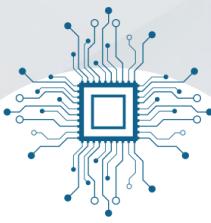
그래프

- 복잡하게 연결되어 있는 **객체**들 사이의 **연결 관계**를 표현할 수 있는 자료구조
- 가장 일반적인 형태의 자료구조

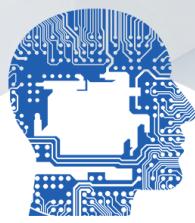
▶ 일상에서 볼 수 있는 그래프의 예



지하철 노선도
역이나 도시의 연결 관계



전기회로
소자들의 연결 관계



딥러닝 분야의 그래프
학습과 인식을 위한 데이터 흐름

그래프 Graph 그래프의 개념

1 그레프란?

● 그래프의 예

▶ 지하철 노선도



역들의 연결 관계를 쉽게 알 수 있음



그래프의 개념

그래프 Graph 그래프의 개념

1 그레프란?

◉ 그래프의 예

▶ 항공 노선도



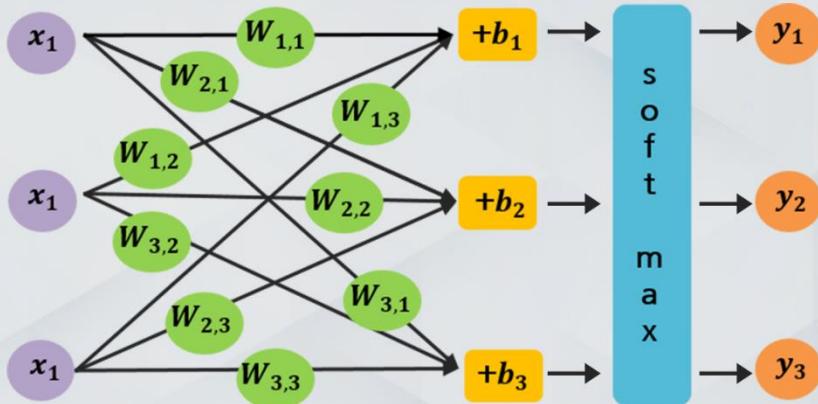
도시들 사이의 항공편 정보를 쉽게 알 수 있음

그래프 Graph 그래프의 개념

1 그레프란?

◉ 그래프의 예

▶ 텐서플로 그래프



신경망의 학습과 인식을 위한 데이터의 흐름을 보여줌

그래프의 개념



그래프 Graph 그래프의 개념

1 그레프란?

● 그래프의 역사

그래프 이론(Graph Theory)

- 그래프와 관련된 다양한 문제를 연구하는 학문
- 그래프는 수학자 오일러(Euler)에 의해 처음 창안

오일러 경로 문제(1800년대)

- 다리를 한 번만 건너서 처음 출발했던 장소로 돌아오는 문제
- 위치: 정점(Node) / 다리: 간선(Edge)

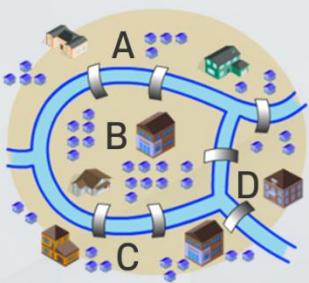
그래프 Graph 그래프의 개념

1 그레프란?

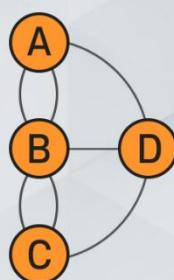
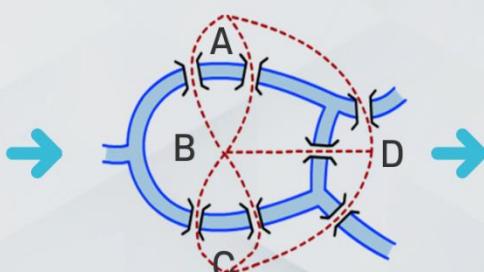
● 그래프의 역사

오일러 경로

모든 다리를 한 번씩만 건너서 처음 출발했던 장소로 돌아오는 경로가 있는가에 대한 문제



원래의 문제



그래프 문제

그래프의 개념



그래프 Graph 그래프의 개념

1 그레프란?

● 그래프의 역사

오일러 정리

- 모든 정점에 연결된 간선의 수가 짝수이면 오일러 경로 존재
- 위의 그래프에는 오일러 경로가 존재하지 않음



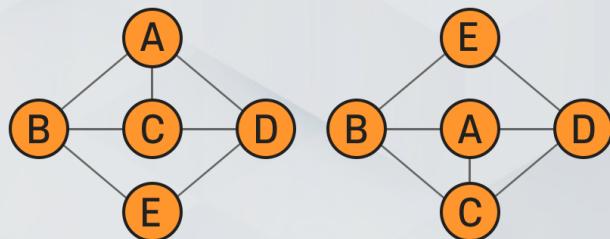
그래프 Graph 그래프의 개념

1 그레프란?

● 그래프의 정의

▶ 그래프 G는 (V, E) 로 표시

V
정점(Vertexes) 또는 노드(Node)



E
간선(Edge) 또는 링크(Link), 정점들 간의 관계 의미

두 그래프가 시각적으로 달라 보임
모든 정점 사이의 관계 동일 → 그렇다면 같은 그래프



그래프의 개념

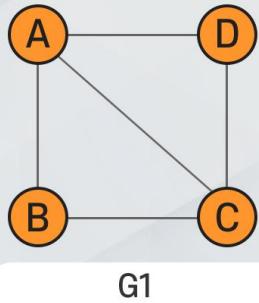
그래프 Graph 그래프의 개념

2 그레프의 종류

● 무방향 그래프(Undirected Graph)

▶ 간선에 방향성이 없는 그래프

▶ $(A, B) = (B, A)$



$$\rightarrow V(G1) = \{ A, B, C, D \}$$

$$\rightarrow E(G1) = \{ (A, B), (A, C), (A, D), (B, C), (C, D) \}$$

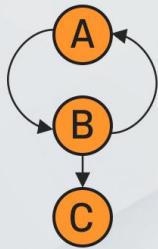
그래프 Graph 그래프의 개념

2 그레프의 종류

● 방향 그래프(Directed Graph)

▶ 간선에 방향성이 있는 그래프

▶ $\langle A, B \rangle \neq \langle B, A \rangle$



$$\rightarrow V(G3) = \{ A, B, C \}$$

$$\rightarrow E(G3) = \{ \langle A, B \rangle, \langle B, A \rangle, \langle B, C \rangle \}$$



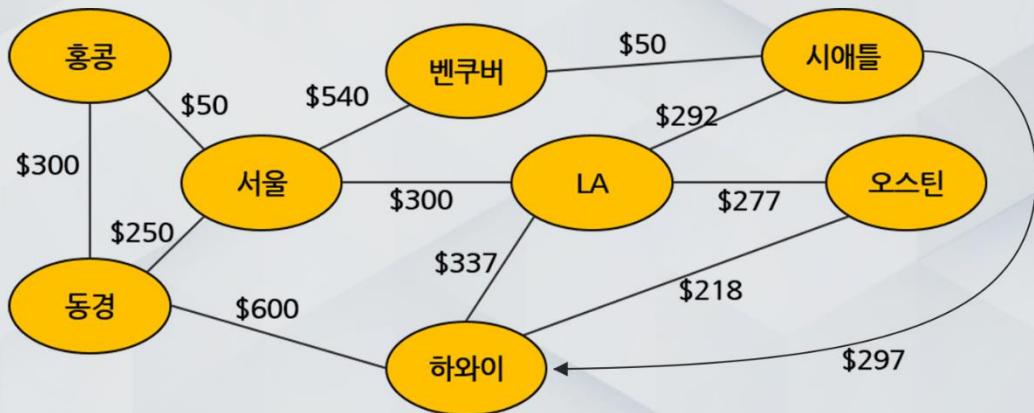
그래프의 개념

그래프 Graph 그래프의 개념

2 그레프의 종류

- 가중치 그래프(Weighted Graph)

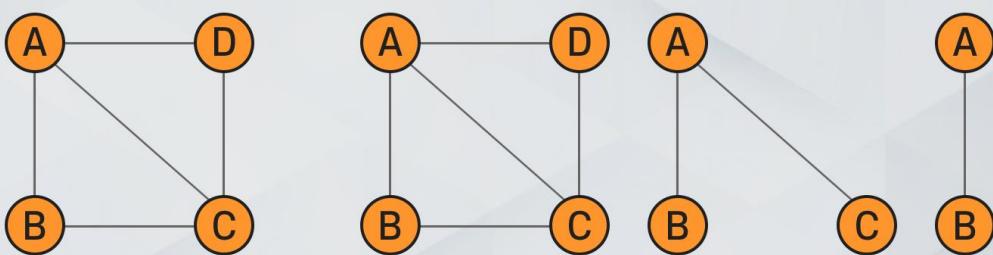
- ▶ 네트워크(Network)라고도 함
- ▶ 간선에 비용이나 가중치가 할당된 그래프



그래프 Graph 그래프의 개념

2 그레프의 종류

- 부분 그래프





그래프의 개념

그래프 Graph 그래프의 개념

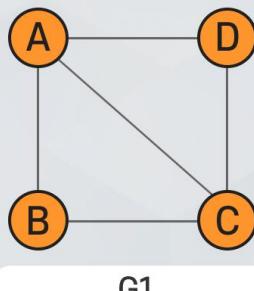
3 그레프의 용어

정점(Vertices), 노드(Node)

간선(Edge), 링크(Link)

인접 정점

- 간선에 의해 직접 연결된 정점



G1

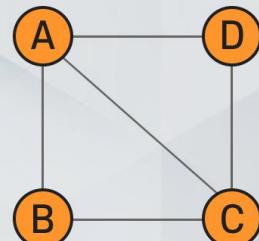
그래프 Graph 그래프의 개념

3 그레프의 용어

정점의 차수(Degree)

- 정점에 연결된 간선의 수
- 무방향 그래프

- 모든 정점의 차수의 합은?
→ 간선 수의 2배



G1



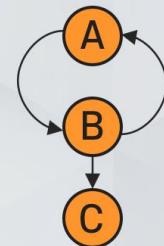
그래프의 개념

그래프 Graph 그래프의 개념

3 그래프의 용어

방향 그래프

- 진입 차수(in-degree)
- 진출 차수(Out-degree)
- 모든 진입(진출) 차수의 합은 간선의 수



G3

그래프 Graph 그래프의 개념

3 그래프의 용어

● 그래프의 경로(Path)

무방향 그래프의 정점 s로부터 정점 e까지의 경로

- 정점의 나열 $s, v_1, v_2, \dots, v_k, e$
- 반드시 간선 $(s, v_1), (v_1, v_2), \dots, (v_k, e)$ 존재해야 함

방향 그래프의 정점 s로부터 정점 e까지의 경로

- 정점의 나열 $s, v_1, v_2, \dots, v_k, e$
- 반드시 간선 $\langle s, v_1 \rangle, \langle v_1, v_2 \rangle, \dots, \langle v_k, e \rangle$ 존재해야 함



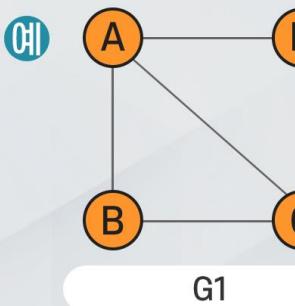
그래프의 개념

그래프 Graph 그래프의 개념

3 그레프의 용어

● 경로의 길이(Length)

경로를 구성하는데 사용된 간선의 수



→ 경로 $A, C, D \rightarrow$ 길이 2

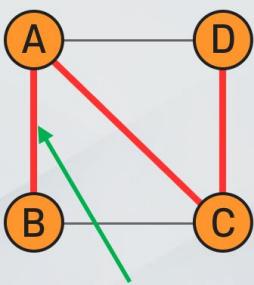
→ 경로 $A, D, C, B \rightarrow$ 길이 3

그래프 Graph 그래프의 개념

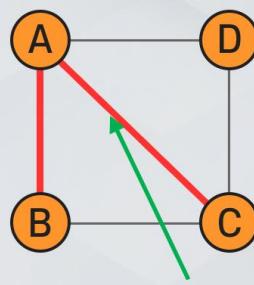
3 그레프의 용어

● 단순 경로(Simple Path)

경로 중에서 반복되는 간선이 없는 경로



경로 B, A, C, D 는 단순 경로



경로 B, A, C, A 는 단순 경로가 아님



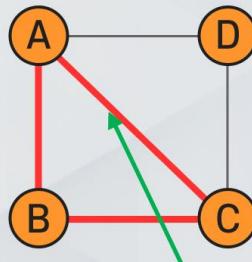
그래프의 개념

그래프 Graph 그래프의 개념

3 그레프의 용어

◎ 사이클(Cycle)

시작 정점과 종료 정점이 동일한 경로



경로 B, A, C, B는 사이클

그래프 Graph 그래프의 개념

3 그레프의 용어

**연결 그래프
(Connected Graph)**

모든 정점들 사이에 경로가 존재하는 그래프

트리(Tree)

사이클을 가지지 않는 연결 그래프



그래프의 개념



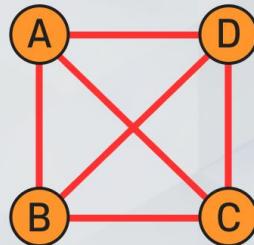
그래프 Graph 그래프의 개념

3 그레프의 용어

● 완전 그래프(Complete Graph)

모든 정점들 사이에 간선이 존재하는 그래프

- N개의 정점을 가진 무방향 완전 그래프의 간선의 수
- $n \times (n-1)/2$ 개
- 예 N=4, 간선의 수 = $(4 \times 3)/2 = 6$



모든 정점 간에 간선이 존재하므로
완전 그래프가 맞음

그래프 Graph 그래프의 개념

4 그레프의 추상 자료형

● Graph ADT

데이터	연산
정점과 간선의 집합	<ul style="list-style-type: none"> ▪ isEmpty(): 그래프가 공백 상태인지 확인 ▪ countVertex(): 정점의 수 반환 ▪ countEdge(): 간선의 수 반환 ▪ getEdge(u, v): 정점 u에서 정점 v로 연결된 간선 반환 ▪ degree(v): 정점 v의 차수 반환 ▪ adjacent(v): 정점 v에 인접한 모든 정점의 집합 반환 ▪ insertVertex(v): 그래프에 정점 v 삽입 ▪ insertEdge(u, v): 그래프에 간선 (u, v) 삽입 ▪ deleteVertex(v): 그래프의 정점 v 삭제 ▪ deleteEdge(u, v): 그래프의 간선 (u, v) 삭제

그래프의 표현 방법

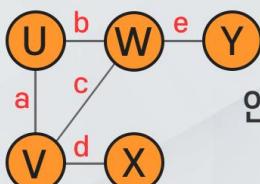


그래프 그래프의 표현 방법

1. 그래프의 표현

배열 구조 표현

- 인접 행렬(Adjacent Matrix)을 이용한 표현
- 행렬을 이용해 정점들의 연결 관계(간선 정보)를 표시



인접 행렬 표현 →

	0	1	2	3	4
U → 0		a	b		
V → 1	a		c	d	
W → 2	b	c			e
X → 3		d			
Y → 4			e		

가중치 그래프가 아니면
a, b, … e는 모두 1

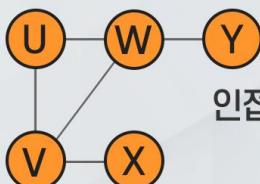
빈 곳은 0 또는 None

그래프 그래프의 표현 방법

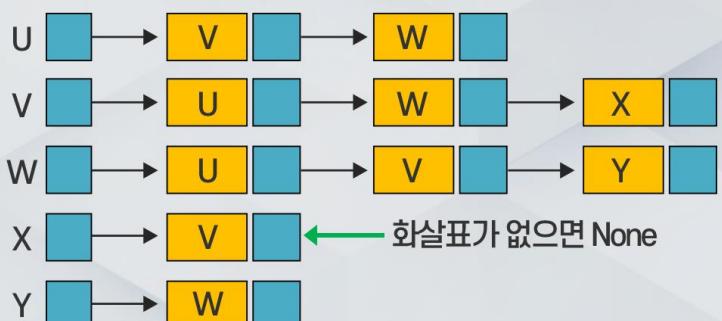
1. 그래프의 표현

연결된 구조 표현

- 인접 리스트(Adjacent List)를 이용한 표현
- 각 정점과 연결된 인접 정점 리스트를 이용해 연결 관계 표시



인접 리스트 표현 →



화살표가 없으면 None

그래프의 표현 방법



그래프 그래프의 표현 방법

1. 그래프의 표현

● 인접 행렬과 인접 리스트의 복잡도 비교

▶ 공간 복잡도(메모리 사용량)

인접 행렬	인접 리스트
<ul style="list-style-type: none"> 간선의 수에 무관하게 항상 n^2개의 메모리 공간 필요 정점에 비해 간선의 수가 매우 많은 조밀 그래프 (Dense Graph)에서 효과적 	<ul style="list-style-type: none"> n개의 연결 리스트, $2e$개의 노드 필요 → $n + 2e$개의 메모리 공간 필요 정점에 비해 간선의 개수가 매우 적은 희소 그래프 (Sparse Graph)에서 효과적

그래프 그래프의 표현 방법

1. 그래프의 표현

● 인접 행렬과 인접 리스트의 복잡도 비교

▶ `getEdge(u, v)`: 간선(u, v)에 접근

인접 행렬	인접 리스트
<ul style="list-style-type: none"> u와 v를 연결하는 간선의 유무는 $M[u][v]$를 조사하면 바로 알 수 있음 시간 복잡도: $O(1)$ 	<ul style="list-style-type: none"> <code>getEdge(u, v)</code> 연산은 정점 u의 연결 리스트 전체를 조사해야 함 정점 u의 차수를 d_u라고 한다면 시간 복잡도: $O(d_u)$



그래프의 표현 방법

그래프 그래프의 표현 방법

1 그래프의 표현

● 인접 행렬과 인접 리스트의 복잡도 비교

▶ degree(v): 정점 v의 차수 계산

인접 행렬	인접 리스트
<ul style="list-style-type: none"> degree(v) 연산은 정점 v에 해당하는 행을 조사하면 되므로 O(n) 정점 v의 차수는 다음과 같이 계산 $\text{degree}(v) = \sum_{k=0}^{n-1} M[v][k]$	<ul style="list-style-type: none"> 정점 v의 차수 degree(v)는 v의 연결 리스트의 길이를 반환 시간 복잡도: O(d_v)

그래프 그래프의 표현 방법

1 그래프의 표현

● 인접 행렬과 인접 리스트의 복잡도 비교

▶ adjacent(v): 정점 v의 모든 인접 정점

인접 행렬	인접 리스트
<ul style="list-style-type: none"> adjacent(v) 연산은 해당 행의 모든 요소를 검사 시간 복잡도: O(n) 	<ul style="list-style-type: none"> 정점 v에 간선으로 직접 연결된 모든 정점을 구하는 adjacent(v) 연산도 해당 연결 리스트의 모든 요소를 방문 시간 복잡도: O(d_v)



그래프의 표현 방법

그래프 그래프의 표현 방법

1 그레프의 표현

● 인접 행렬과 인접 리스트의 복잡도 비교

▶ 그래프의 모든 간선의 수 계산

인접 행렬	인접 리스트
<ul style="list-style-type: none"> 그래프에 존재하는 모든 간선의 수를 알아내려면 인접 행렬 전체를 조사해야 하므로 n^2번의 조사 필요 시간 복잡도: $O(n^2)$ 	<ul style="list-style-type: none"> 전체 간선의 수를 알아내려면 헤더 노드를 포함하여 모든 인접 리스트를 조사 시간 복잡도: $O(n + e)$

그래프 그래프의 표현 방법

1 그레프의 표현

● 파이썬을 이용한 그래프의 표현

▶ 다양한 방법의 표현이 가능

인접 행렬	인접 리스트
<ul style="list-style-type: none"> 리스트의 리스트(2차원 배열) 	<ul style="list-style-type: none"> 연결 리스트 파이썬 리스트 파이썬 집합 (정점, 인접 리스트) 형태의 엔트리를 딕셔너리에 저장할 수도 있음



그래프의 표현 방법

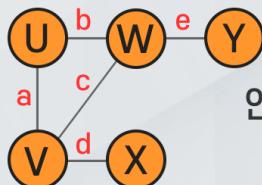
그래프 그래프의 표현 방법

2 인접 행렬을 이용한 표현

인접 행렬 M을 이용

- 간선 (i, j) 가 있으면: $M[i][j] = 1$ 또는 true
- 간선 (i, j) 가 없으면: $M[i][j] = 0$ 또는 false 또는 None

무방향 그래프



인접 행렬 표현 →

인접 행렬이 대칭

	0	1	2	3	4
U → 0		a	b		
V → 1	a		c	d	
W → 2	b	c			e
X → 3		d			
Y → 4			e		

가중치 그래프가 아니면
a, b, … e는 모두 1

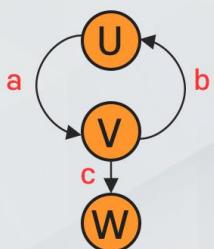
빈 곳은 0 또는 None

그래프 그래프의 표현 방법

2 인접 행렬을 이용한 표현

방향 그래프

인접 행렬이 대칭이 아님



인접 행렬 표현 →

	0	1	2
U → 0		a	
V → 1	b		c
W → 2			

방향 그래프에서는
인접 행렬이 대칭이 아님

그래프의 표현 방법



그래프 그래프의 표현 방법

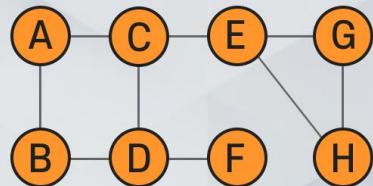
2 인접 행렬을 이용한 표현

● 무방향 그래프 표현 예

```

vertex = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
adjMat = [
    [0, 1, 1, 0, 0, 0, 0, 0],
    [1, 0, 0, 1, 0, 0, 0, 0],
    [1, 0, 0, 1, 1, 0, 0, 0],
    [0, 1, 1, 0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 0, 1, 1],
    [0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 1],
    [0, 0, 0, 0, 1, 0, 1, 0]
]
graph0 = (vertex, adjMat)

```



그래프 그래프의 표현 방법

2 인접 행렬을 이용한 표현

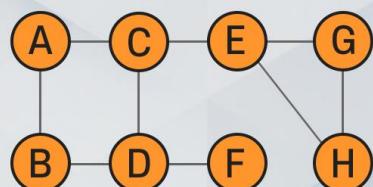
● 무방향 그래프의 간단한 연산

▶ 모든 정점의 인접 정점 개수 출력

```

def printDegree_AM(graph):
    vlist = graph[0]          # 정점 리스트
    M = graph[1]              # 인접 행렬
    for i in range(len(vlist)): # 모든 정점에 대해
        row = M[i]             # 하나의 행
        count = 0                # 인접 정점의 수 초기화
        for e in M[i]:           # 행의 모든 항목에 대해
            if e > 0: count += 1
        print(vlist[i], ":", count, end=' ')
    print()
print('AM : ', end='')
printDegree_AM(graph0)

```



```

C:\WINDOWS\system32\cmd.exe
AM : A : 2  B : 2  C : 3  D : 3  E : 3  F : 1  G : 2  H : 2

```

그래프의 표현 방법

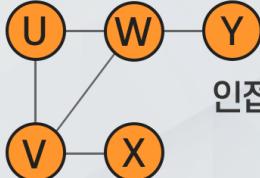


그래프 Graph 그래프의 표현 방법

3 인접 리스트를 이용한 표현

각 정점과 연결된 인접 정점 리스트 이용

- 인접 정점 리스트를 연결 리스트로 구현한 경우의 예



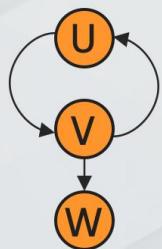
인접 리스트 표현 →



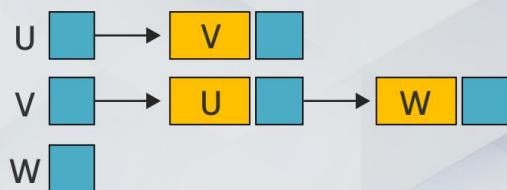
그래프 Graph 그래프의 표현 방법

3 인접 리스트를 이용한 표현

- 방향 그래프의 표현



인접 리스트 표현 →



그래프의 표현 방법



그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

- “인접 정점 리스트”를 위해 무엇을 사용할 것인가?

연결 리스트	파이썬의 리스트	정점과 간선을 한꺼번에 표현: 딕셔너리 사용
<ul style="list-style-type: none"> ▪ C나 C++ 등에서 사용 	<ul style="list-style-type: none"> ▪ 인덱스를 저장 → 방법1 ▪ 정점 키를 저장 → 방법2 	<ul style="list-style-type: none"> ▪ 인접 리스트로 파이썬의 리스트(방법2) 사용 → 방법3 ▪ 인접 리스트로 파이썬의 집합 사용 → 방법4

그래프 그래프의 표현 방법

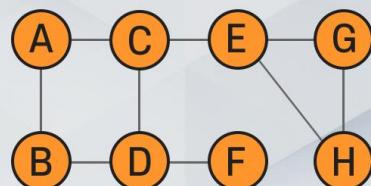
3 인접 리스트를 이용한 표현

- 방법1: 인접 정점 인덱스의 리스트

```

vertex = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H' ]
adjList1 = [[ 1, 2 ],      # 'A'의 인접정점 인덱스 리스트
            [ 0, 3 ],      # 'B'의 인접정점 인덱스 리스트
            [ 0, 3, 4 ],    # 'C'의 인접정점 인덱스 리스트
            [ 1, 2, 5 ],    # 'D'의 인접정점 인덱스 리스트
            [ 2, 6, 7 ],    # 'E'의 인접정점 인덱스 리스트
            [ 3 ],          # 'F'의 인접정점 인덱스 리스트
            [ 4, 7 ],        # 'G'의 인접정점 인덱스 리스트
            [ 4, 6 ] ]       # 'H'의 인접정점 인덱스 리스트
graph1 = (vertex, adjMat)

```



그래프의 표현 방법



그래프 그래프의 표현 방법

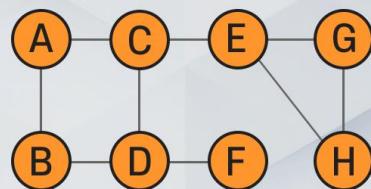
3 인접 리스트를 이용한 표현

● 방법2: 인접 정점 키의 리스트

```

vertex  = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H' ]
adjList1 = [[ 'B', 'C' ],      # 'A'의 인접정점 키 리스트
            [ 'A', 'D' ],      # 'B'의 인접정점 키 리스트
            [ 'A', 'D', 'E' ], # 'C'의 인접정점 키 리스트
            [ 'B', 'C', 'F' ],# 'D'의 인접정점 키 리스트
            [ 'C', 'G', 'H' ],# 'E'의 인접정점 키 리스트
            [ 'D' ],          # 'F'의 인접정점 키 리스트
            [ 'E', 'H' ],      # 'G'의 인접정점 키 리스트
            [ 'E', 'G' ] ]    # 'H'의 인접정점 키 리스트
graph1 = (vertex, adjList1)

```



그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

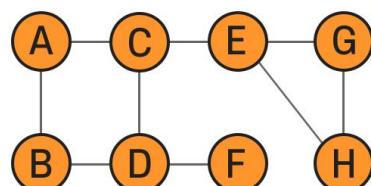
● 모든 정점의 인접 정점 개수 출력

```

def printDegree_AL1(graph):
    vlist = graph[0]          # 정점 리스트
    alist = graph[1]          # 인접 정점 리스트
    for i in range(len(vlist)): # 모든 정점에 대해
        count = len(alist[i]) # 인접 정점의 수
        print( vlist[i], ":", count, end=' ')
    print()

print('AL1: ', end='')
printDegree_AL1(graph2)

```



```
C:\#WINDOWS\system32#cmd.exe
AL1: A : 2  B : 2  C : 3  D : 3  E : 3  F : 1  G : 2  H : 2
```

그래프의 표현 방법

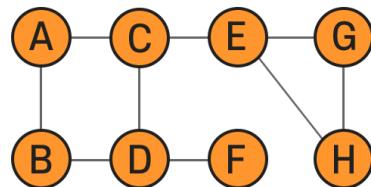


그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

● 방법3: 딕셔너리와 인접 리스트를 이용한 표현

```
graph3 = { 'A':[ 'B', 'C' ],      # 키:정점 A, 값:A의 인접 리스트
           'B':[ 'A', 'D' ],      # 키:정점 B, 값:B의 인접 리스트
           'C':[ 'A', 'D', 'E' ], # 키:정점 C, 값:C의 인접 리스트
           'D':[ 'B', 'C', 'F' ],# 키:정점 D, 값:D의 인접 리스트
           'E':[ 'C', 'G', 'H' ],# 키:정점 E, 값:E의 인접 리스트
           'F':[ 'D' ],          # 키:정점 F, 값:F의 인접 리스트
           'G':[ 'E', 'H' ],      # 키:정점 G, 값:G의 인접 리스트
           'H':[ 'E', 'G' ] }    # 키:정점 H, 값:H의 인접 리스트
```

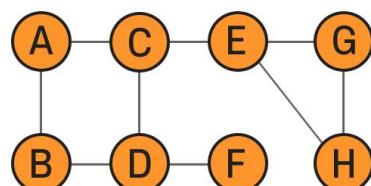


그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

● 모든 정점의 인접 정점 개수 출력

```
def printDegree_AL2(graph):      # graph: 딕셔너리 표현 (정점: 인접 리스트)
    for v in graph:              # 그래프의 모든 정점 v에 대해: 'A', ...
        count = len(graph[v])    # graph[v]: v의 인접 리스트
                                    # 예) graph['A'] → ['B', 'C']
        print( v, ":", count, end=' ')
    print( )
    print('AL2: ', end='')
    printDegree_AL2(graph3)
```



```
C:\WINDOWS\system32\cmd.exe
AL2: A : 2  B : 2  C : 3  D : 3  E : 3  F : 1  G : 2  H : 2
```

그래프의 표현 방법

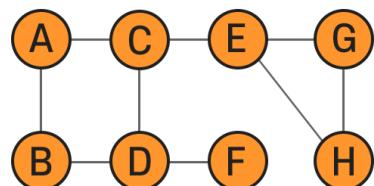


그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

● 방법4: 딕셔너리와 인접 집합을 이용한 표현

```
graph5 = {'A': {'B', 'C'},      # 키:정점 A, 값:A의 인접 집합
          'B': {'A', 'D'},      # 키:정점 B, 값:B의 인접 집합
          'C': {'A', 'D', 'E'}, # 키:정점 C, 값:C의 인접 집합
          'D': {'B', 'C', 'F'}, # 키:정점 D, 값:D의 인접 집합
          'E': {'C', 'G', 'H'}, # 키:정점 E, 값:E의 인접 집합
          'F': {'D'},           # 키:정점 F, 값:F의 인접 집합
          'G': {'E', 'H'},       # 키:정점 G, 값:G의 인접 집합
          'H': {'E', 'G'}} }     # 키:정점 H, 값:H의 인접 집합
```



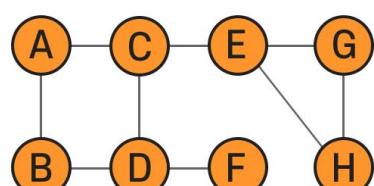
그래프 그래프의 표현 방법

3 인접 리스트를 이용한 표현

● 모든 정점의 인접 정점 개수 출력

```
def printDegree_AL3(graph):    # graph: 딕셔너리 표현 (정점: 인접 집합)
    for v in graph:            # 그래프의 모든 정점 v에 대해: 'A', ...
        count = len(graph[v])  # graph[v]: v의 인접 집합
                                # 예) graph['A'] → {'B', 'C'}
        print(v, ":", count, end=' ')
    print()

print('AL3: ', end='')
printDegree_AL3(graph3)
```



```
C:\WINDOWS\system32\cmd.exe
AL3: A : 2  B : 2  C : 3  D : 3  E : 3  F : 1  G : 2  H : 2
```