CHAPTER.17

그래프의 기본 및 간단 구현

학습 내용

[1] 그래프의 기본

[2] 그래프의 구현

학습 목표

- 그래프의 개념을 설명할 수 있다.
- 그래프를 구성하는 파이썬 코드를 작성할 수 있다.

01 그래프의 기본

1) 그래프 구조란?
2) 그래프의 개념
3) 그래프의 종류
4) 깊이 우선 탐색의 작동
5) 그래프의 인접 행렬 표현

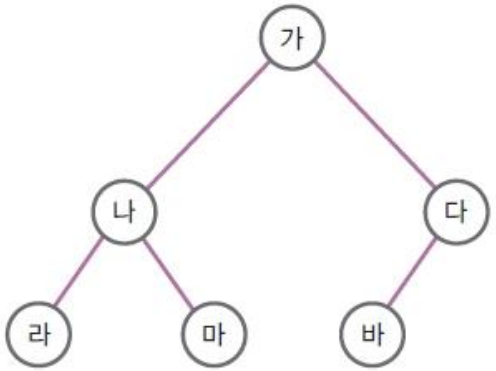
1] 그래프 구조란?

버스 정류장과 여러 노선이 함께 포함된 형태

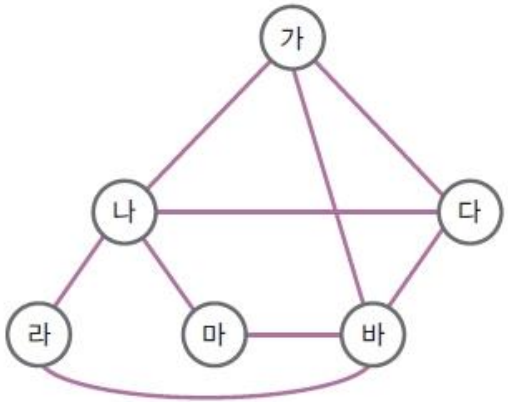
링크드인(Linked in)과 같은 사회 관계망 서비스 연결 등의 형태

여러 노드가 서로 연결된 자료구조

- 트리와 그래프의 차이



트리 예



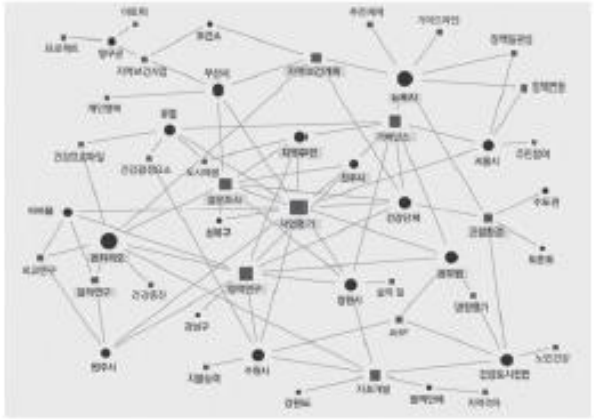
그래프 예

그래프를 활용한 다양한 예

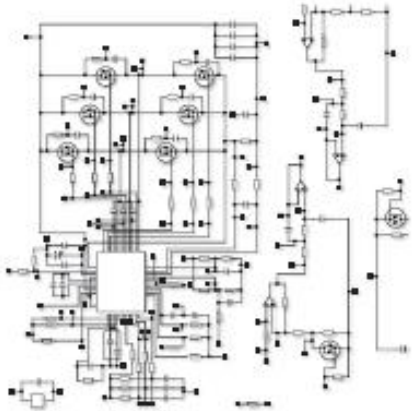
지하철 노선도(부산)

KTX 노선도

그래프를 활용한 다양한 예



도시 도로망



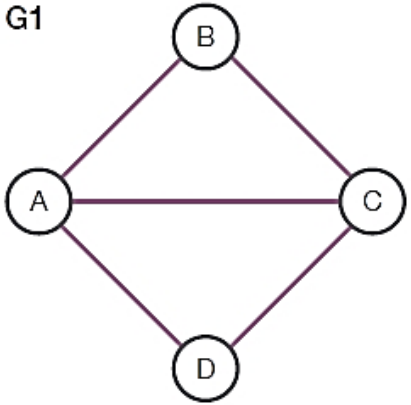
전기 회로도

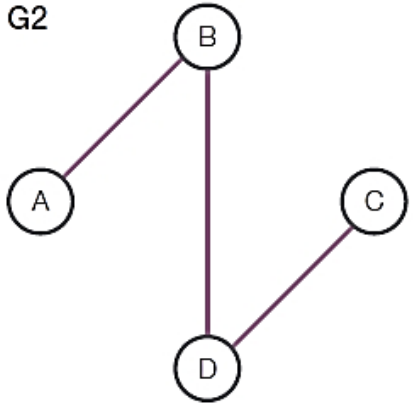
그래프를 활용한 다양한 예

인맥 관계도

1 무방향 그래프

- 간선에 방향성이 없는 그래프
- 무방향 그래프의 형태





1 무방향 그래프

- G1, G2의 정점 집합 표현

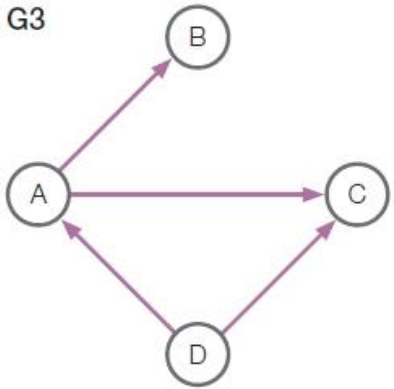
```
V(G1) = { A, B, C, D }
V(G2) = { A, B, C, D }
```

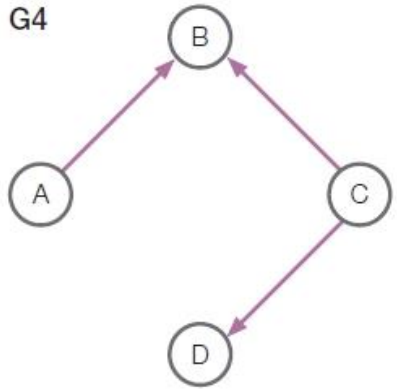
- G1, G2의 간선 집합 표현

```
E(G1) = { (A, B), (A, C), (A, D), (B, C), (C, D) }
E(G2) = { (A, B), (B, D), (D, C) }
```

2 방향 그래프

- 화살표로 간선 방향을 표기하고, 그래프의 정점 집합이 무방향 그래프와 같음
- 방향 그래프의 형태





2 방향 그래프

- G3, G4의 정점 집합 표현(무방향 그래프와 같음)

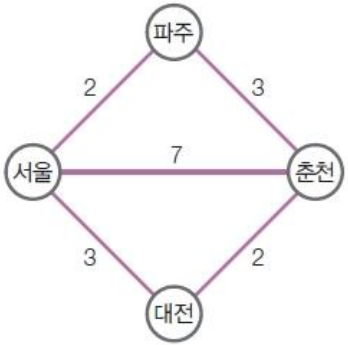
V(G3) = { A, B, C, D }
V(G4) = { A, B, C, D }

- G3, G4의 간선 집합 표현

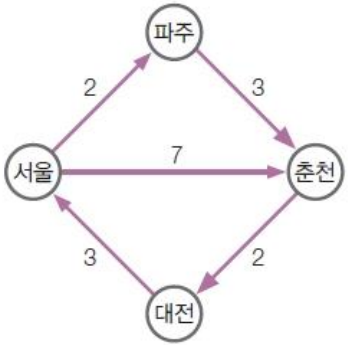
E(G3) = { <A, B>, <A, C>, <D, A>, <D, C> }
E(G4) = { <A, B>, <C, B>, <C, D> }

3 가중치 그래프

- 간선마다 가중치가 다르게 부여된 그래프
- 무방향 그래프와 방향 그래프에 각각 가중치를 부여한 경우의 예



무방향 가중치 그래프



방향 가중치 그래프

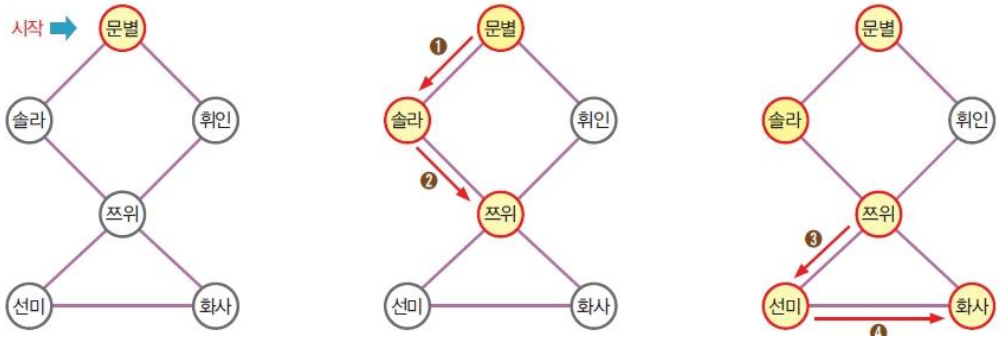
4] 깊이 우선 탐색의 작동

그래프 순회(Graph Traversal)

그래프의 모든 정점을 한 번씩 방문하는 것

그래프 순회 방식은 깊이 우선 탐색, 너비 우선 탐색이 대표적

- 깊이 우선 탐색 그래프의 예 : '문별'부터 시작



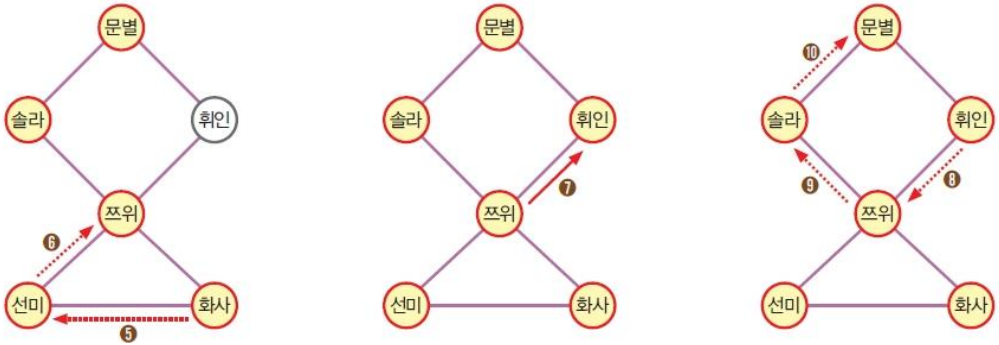
4] 깊이 우선 탐색의 작동

그래프 순회(Graph Traversal)

그래프의 모든 정점을 한 번씩 방문하는 것

그래프 순회 방식은 깊이 우선 탐색, 너비 우선 탐색이 대표적

- 깊이 우선 탐색 그래프의 예 : '문별'부터 시작



5] 그래프의 인접 행렬 표현

그래프를 코드로 구현할 때는 인접 행렬을 사용

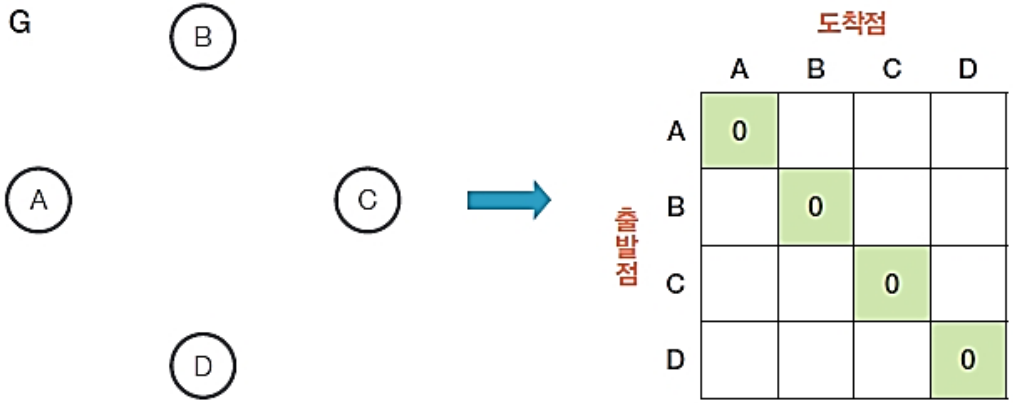
인접 행렬

정방형으로 구성된 행렬

5] 그래프의 인접 행렬 표현

정점이 4개인 그래프는 4×4로 표현

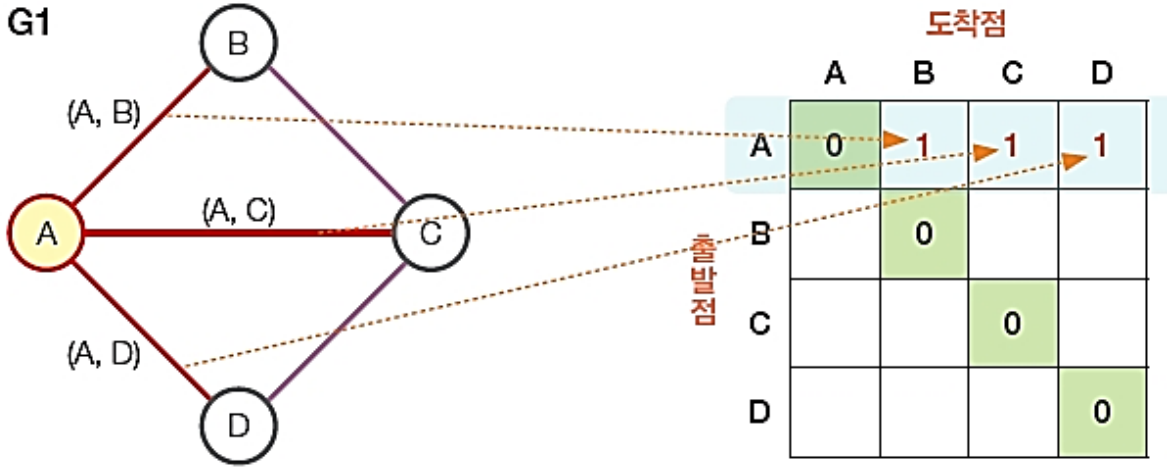
- 정점 4개로 된 그래프의 인접 행렬 초기 상태



5] 그래프의 인접 행렬 표현

무방향 그래프의 인접 행렬

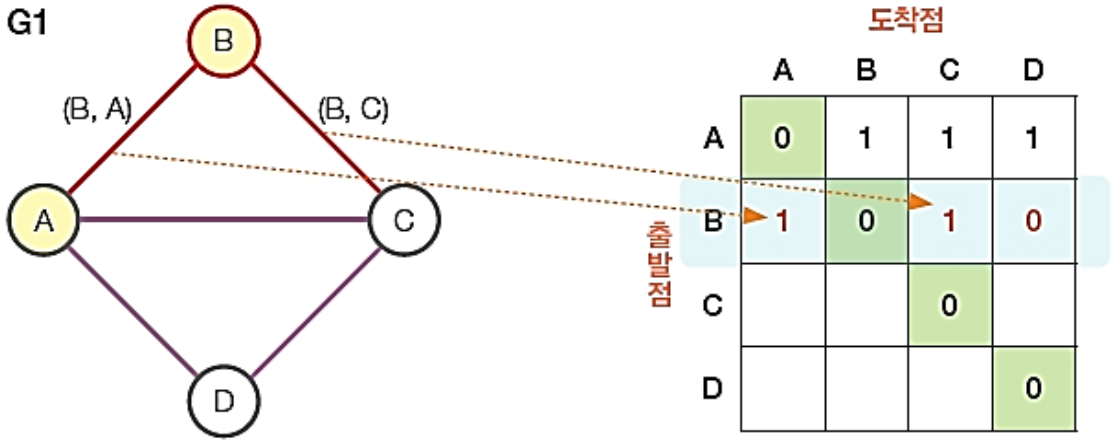
1 출발점 A와 연결된 도착점 B, C, D의 칸을 1로 설정



5] 그래프의 인접 행렬 표현

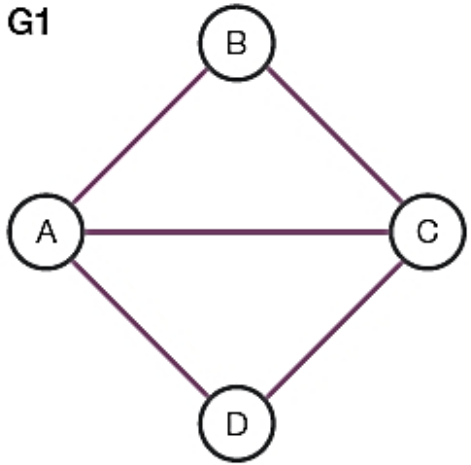
무방향 그래프의 인접 행렬

2 출발점 B와 연결된 도착점 A와 C의 칸을 1로 설정하고, 연결되지 않은 도착점 D는 0으로 설정



무방향 그래프의 인접 행렬

3 같은 방식으로 출발점 C와 D를 인접 행렬로 추가



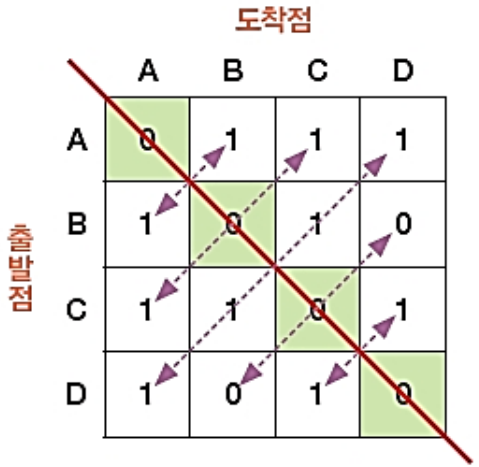
출발점 \ 도착점	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

5] 그래프의 인접 행렬 표현

세종사이버대학교

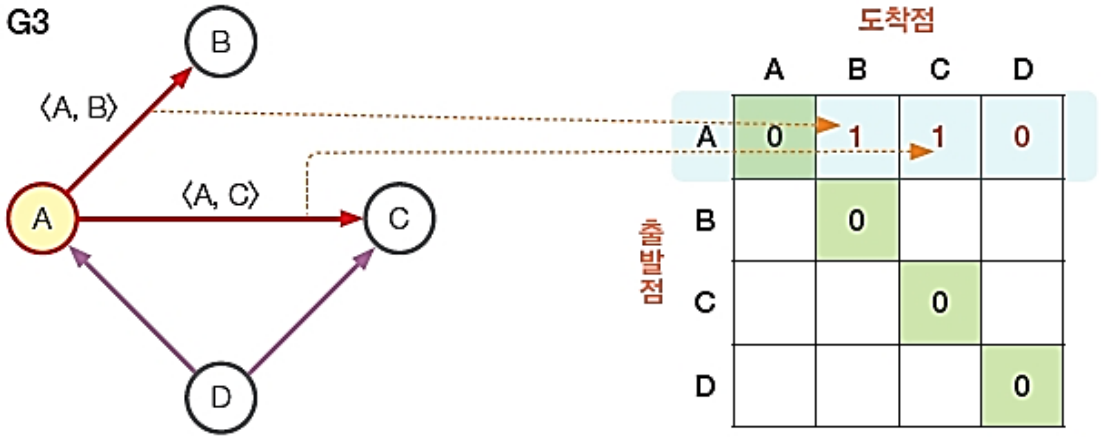
무방향 그래프의 인접 행렬은 대각선을 기준으로 서로 대칭

- **무방향 그래프의 대칭 특성**



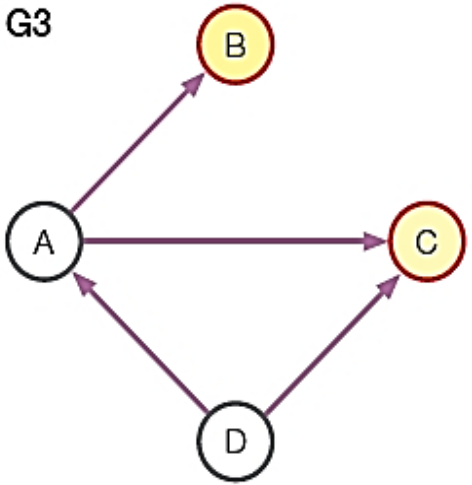
방향 그래프의 인접 행렬

1 출발점 A에서 나가 도착점이 B, C의 칸만 1로 설정하고 나머지는 0으로 채움



방향 그래프의 인접 행렬

2 출발점 B와 C는 나가는 곳이 없으므로 모두 0으로 채움



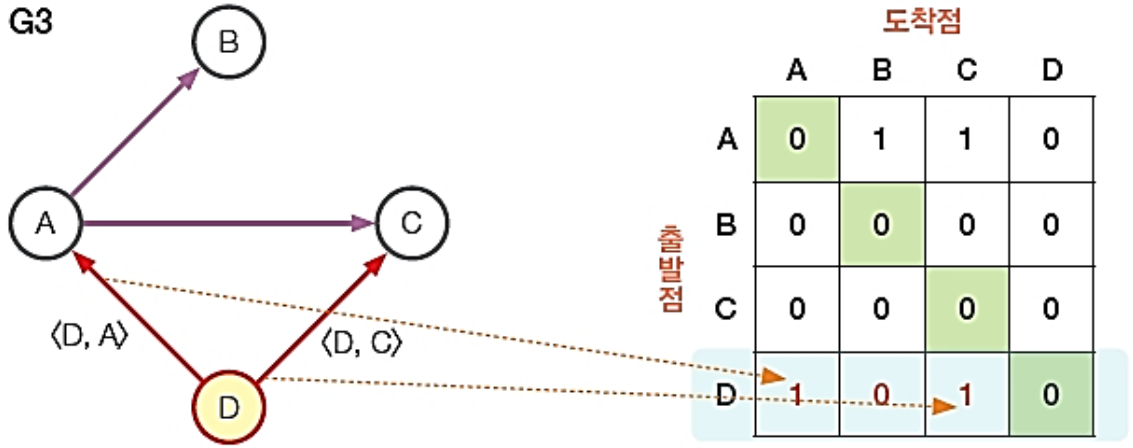
도착점

출발점	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	0
D				0

5] 그래프의 인접 행렬 표현

방향 그래프의 인접 행렬

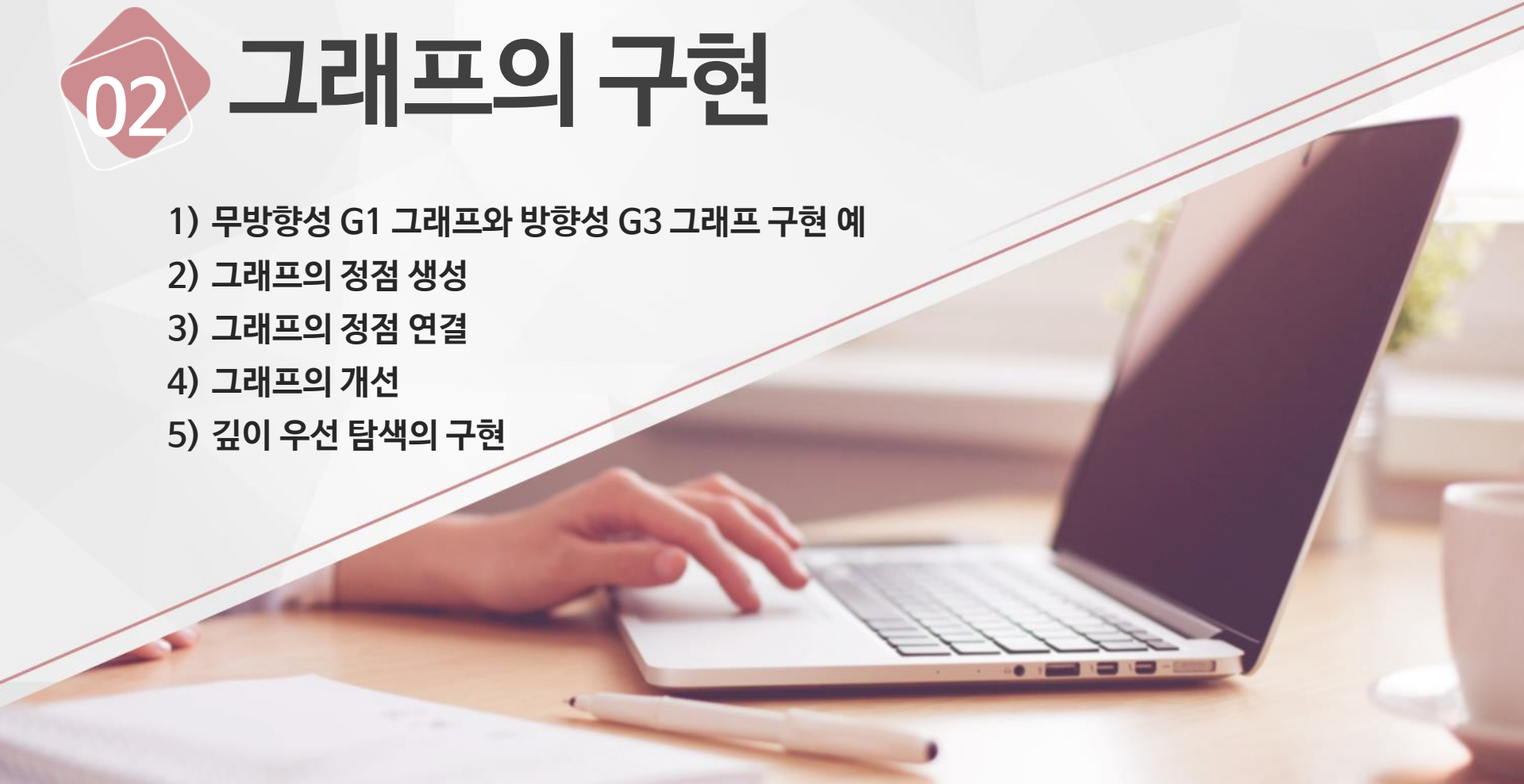
3 출발점 D는 도착점 A와 C만 1로 설정하고
나머지는 0으로 채움



출발점 \ 도착점	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	0	0	0
D	1	0	1	0

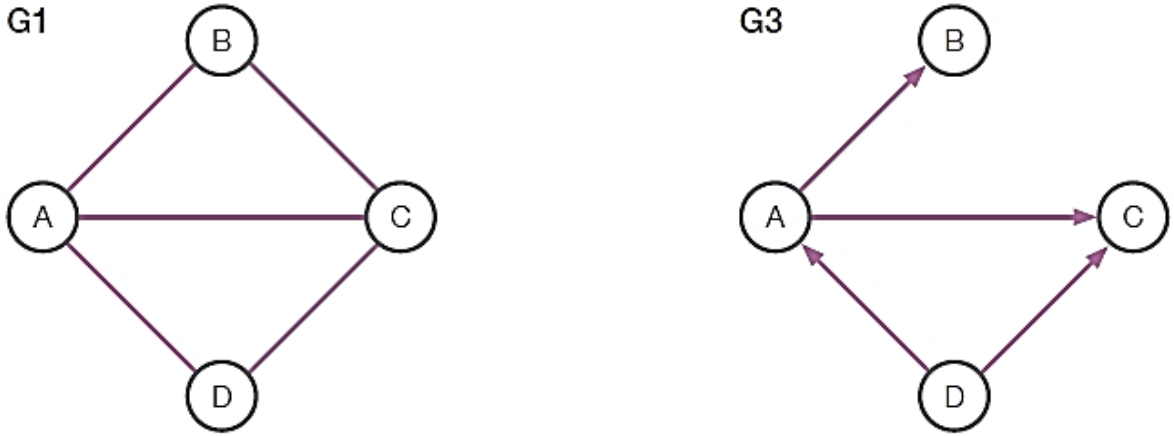
02 그래프의 구현

1) 무방향성 G1 그래프와 방향성 G3 그래프 구현 예
2) 그래프의 정점 생성
3) 그래프의 정점 연결
4) 그래프의 개선
5) 깊이 우선 탐색의 구현



1] 무방향성 G1 그래프와 방향성 G3 그래프 구현 예

구현할 무방향 그래프 G1과 방향 그래프 G3



2] 그래프의 정점 생성

행과 열이 같은 2차원 배열을 생성하는 클래스로 작성

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

G1 = Graph(4)
```

2] 그래프의 정점 생성

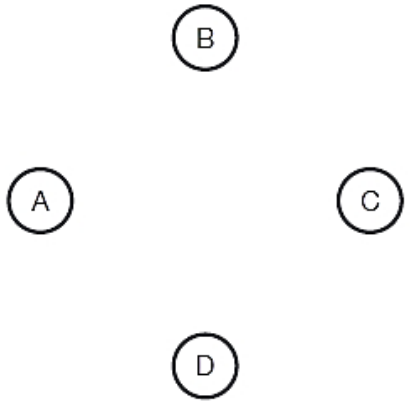
행과 열이 같은 2차원 배열을 생성하는 클래스로 작성

- 4×4 크기의 초기화된 그래프를 생성

→ 정점 4개를 가진 그래프의 초기 상태(인접 행렬)와 그래프

출발점 \ 도착점	A=0	B=1	C=2	D=3
A=0	0	0	0	0
B=1	0	0	0	0
C=2	0	0	0	0
D=3	0	0	0	0

G1(size=4)

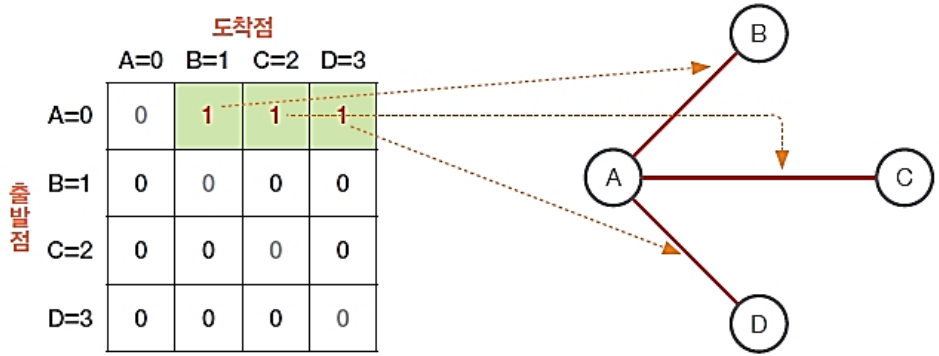


G1(size=4)

정점 A와 B에 연결된 간선 구현

```
G1.graph[0][1] = 1    # (A, B) 간선
G1.graph[0][2] = 1    # (A, C) 간선
G1.graph[0][3] = 1    # (A, D) 간선
```

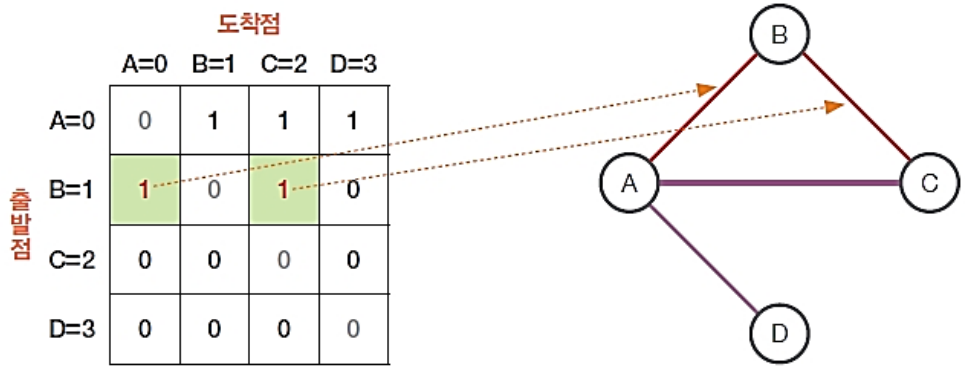
- 정점 A와 연결된 간선 구현 코드와 결과



정점 A와 B에 연결된 간선 구현

```
G1.graph[1][0] = 1    # (B, A) 간선
G1.graph[1][2] = 1    # (B, C) 간선
```

- 정점 B와 연결된 간선 구현 코드와 결과



3] 그래프의 정점 연결

같은 방식으로 출발점 C와 D를 다음과 같이 연결

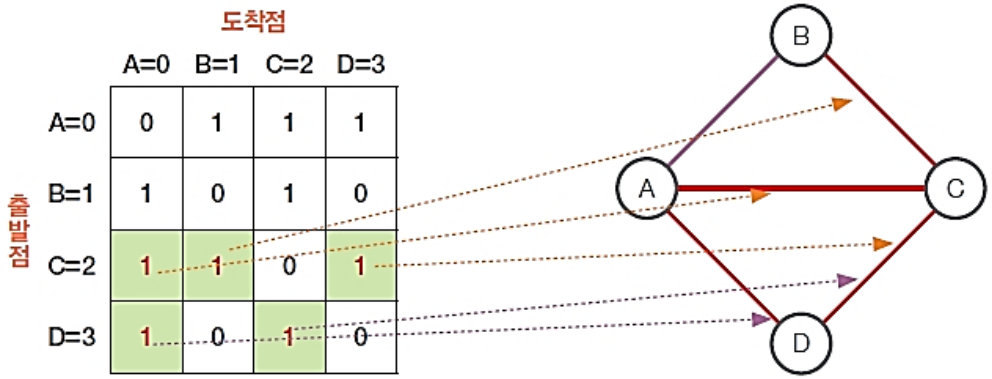
```
G1.graph[2][0] = 1   # (C, A) 간선
G1.graph[2][1] = 1   # (C, B) 간선
G1.graph[2][3] = 1   # (C, D) 간선

G1.graph[3][0] = 1   # (D, A) 간선
G1.graph[3][2] = 1   # (D, C) 간선
```

3] 그래프의 정점 연결

같은 방식으로 출발점 C와 D를 다음과 같이 연결

- 정점 C 및 D와 연결된 간선 구현 코드와 결과



무방향 그래프 G1과 방향 그래프 G3의 구현

```
## 함수 선언 부분 ##
class Graph() :
    def __init__(self, size) :
        self.SIZE = size
        self.graph = [[0 for _ in range(size)] for _ in range(size)]

## 전역 변수 선언 부분 ##
G1, G3 = None, None

## 메인 코드 부분 ##
G1 = Graph(4)
G1.graph[0][1] = 1; G1.graph[0][2] = 1; G1.graph[0][3] = 1
G1.graph[1][0] = 1; G1.graph[1][2] = 1
```

무방향 그래프 G1과 방향 그래프 G3의 구현

```
14 G1.graph[2][0] = 1; G1.graph[2][1] = 1; G1.graph[2][3] = 1
15 G1.graph[3][0] = 1; G1.graph[3][2] = 1
16
17 print('## G1 무방향 그래프 ##')
18 for row in range(4) :
19     for col in range(4) :
20         print(G1.graph[row][col], end = ' ')
21     print()
22
23 G3 = Graph(4)
24 G3.graph[0][1] = 1; G3.graph[0][2] = 1
25 G3.graph[3][0] = 1; G3.graph[3][2] = 1
26
```

무방향 그래프 G1과 방향 그래프 G3의 구현

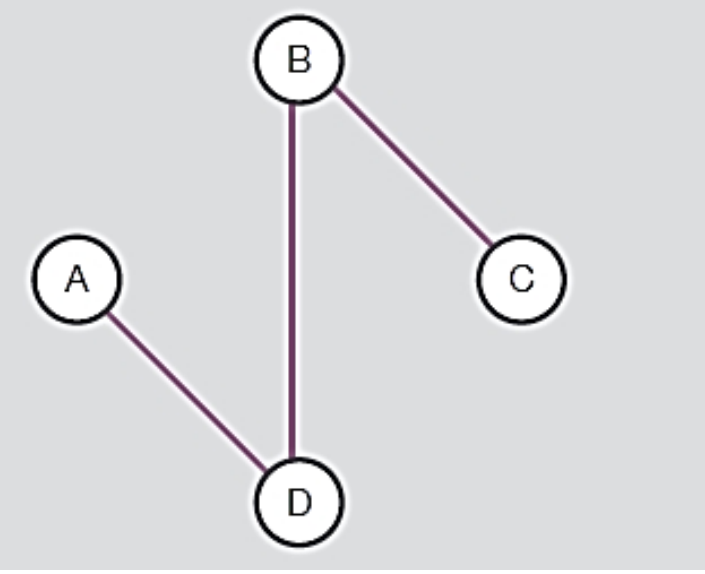
```
23 G3 = Graph(4)
24 G3.graph[0][1] = 1; G3.graph[0][2] = 1
25 G3.graph[3][0] = 1; G3.graph[3][2] = 1
26
27 print('## G3 방향 그래프 ##')
28 for row in range(4) :
29     for col in range(4) :
30         print(G3.graph[row][col], end = ' ')
31     print()
```

실행 결과

```
## G1 무방향 그래프 ##
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
## G3 방향 그래프 ##
0 1 1 0
0 0 0 0
0 0 0 0
1 0 1 0
```

그래프의 정점 연결 실습

> 앞쪽의 소스를 수정해서 다음 그림과 같은 무방향 그래프가 출력되도록 하자



그래프의 정점 연결 실습

앞쪽의 소스를 수정해서 다음 그림과 같은 무방향 그래프가 출력되도록 하자

실행 결과

```
## 무방향 그래프 ##
0 0 0 1
0 0 1 1
0 1 0 0
1 1 0 0
```

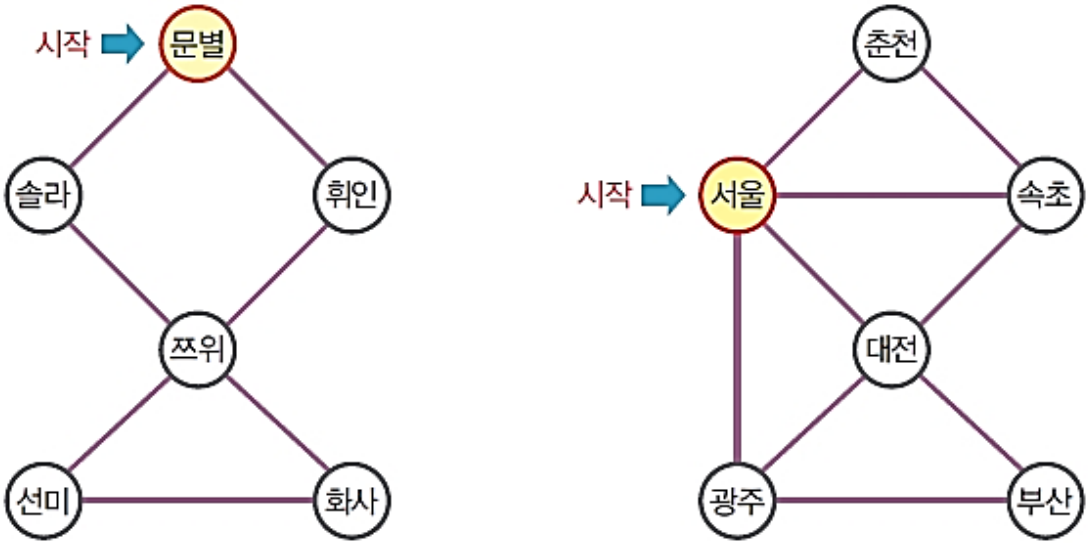
3] 그래프의 정점 연결

세종사이버대학교

그래프의 정점 연결 실습

무방향 그래프를 인접 행렬로 구성할 때
사람 이름, 도시 이름으로 구성한 예

- **그래프의 실제 형태**



무방향 그래프를 인접 행렬로 구성할 때 사람 이름, 도시 이름으로 구성한 예

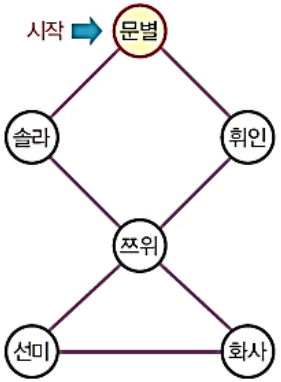
```
G1.graph[0][1] = 1; G1.graph[0][2] = 1
G1.graph[1][0] = 1; G1.graph[1][3] = 1
```

```
문별, 솔라, 휘인, 쯔위 = 0, 1, 2, 3
G1.graph[문별][솔라] = 1; G1.graph[문별][휘인] = 1
G1.graph[솔라][문별] = 1; G1.graph[솔라][쯔위] = 1
```

➡ 변수 이름을 정점 번호로 지정하면 더 직관적임

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- 행과 열의 주석이 없는 인접 행렬



(a) 구현할 그래프

```
0  1  1  0  0  0
1  0  0  1  0  0
1  0  0  1  0  0
0  1  1  0  1  1
0  0  0  1  0  1
0  0  0  1  1  0
```

(b) 행과 열의 주석이 없는 인접 행렬

	문별	솔라	휘인	쯔위	선미	화사
문별	0	1	1	0	0	0
솔라	1	0	0	1	0	0
휘인	1	0	0	1	0	0
쯔위	0	1	1	0	1	1
선미	0	0	0	1	0	1
화사	0	0	0	1	1	0

(c) 행과 열의 주석이 있는 인접 행렬

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- (c)와 같이 출력하기 위한 코드

```
nameAry = ['문별', '솔라', '휘인', '쯔위', '선미', '화사']
print(' ', end = ' ')
for v in range(G1.SIZE) :
    print(nameAry[v], end = ' ')
print()
for row in range(G1.SIZE) :
    print(nameAry[row], end = ' ')
    for col in range(G1.SIZE) :
        print(G1.graph[row][col], end = ' ')
    print()
print()
```

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- 무방향 그래프만 (c)와 같이 행과 열의 주석이 있는 인접 행렬 형태로 구현

```
## 클래스와 함수 선언 부분 ##
class Graph() :
    def __init__ (self, size) :
        self.SIZE = size
        self.graph = [[0 for _ in range(size)] for _ in range(size)]

def printGraph(g) :
    print(' ', end = ' ')
    for v in range(g.SIZE) :
        print(nameAry[v], end = ' ')
    print()
```

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- 무방향 그래프만 (c)와 같이 행과 열의 주석이 있는 인접 행렬 형태로 구현

```
12      for row in range(g.SIZE) :
13          print(nameAry[row], end = ' ')
14          for col in range(g.SIZE) :
15              print(g.graph[row][col], end = ' ')
16          print()
17      print()
18
19
20  ## 전역 변수 선언 부분 ##
21  G1 = None
22  nameAry = ['문별', '솔라', '휘인', '쯔위', '선미', '화사']
23  문별, 솔라, 휘인, 쯔위, 선미, 화사 = 0, 1, 2, 3, 4, 5
```

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- 무방향 그래프만 (c)와 같이 행과 열의 주석이 있는 인접 행렬 형태로 구현

```
24
25
26 ## 메인 코드 부분 ##
27 gSize = 6
28 G1 = Graph(gSize)
29 G1.graph[문별][솔라] = 1; G1.graph[문별][휘인] = 1
30 G1.graph[솔라][문별] = 1; G1.graph[솔라][쯔위] = 1
31 G1.graph[휘인][문별] = 1; G1.graph[휘인][쯔위] = 1
32 G1.graph[쯔위][솔라] = 1; G1.graph[쯔위][휘인] = 1; G1.graph[쯔위][선미] = 1; G1.graph[쯔위][화사] = 1
33 G1.graph[선미][쯔위] = 1; G1.graph[선미][화사] = 1
```

인접 행렬 출력 시 주석 없이 출력하는 예와 주석을 추가하여 출력하는 예

- 무방향 그래프만 (c)와 같이 행과 열의 주석이 있는 인접 행렬 형태로 구현

```
34 G1.graph[화사][쯔위] = 1; G1.graph[화사][선미] = 1
35
36 print('## G1 무방향 그래프 ##')
37 printGraph(G1)
```

실행 결과

```
## G1 무방향 그래프 ##
      문별  솔라  휘인  쯔위  선미  화사
문별    0     1     1     0     0     0
솔라    1     0     0     1     0     0
휘인    1     0     0     1     0     0
쯔위    0     1     1     0     1     1
선미    0     0     0     1     0     1
화사    0     0     0     1     1     0
```

5] 깊이 우선 탐색의 구현

깊이 우선 탐색 구현을 위한 준비

1. 깊이 우선 탐색을 구현하려면 스택을 사용해야 함

2. 코드를 좀 더 간략히 하고자 별도의 top을 사용하지 않고 append( )로 푸시를, pop( )으로 팝하는 스택을 사용

```
stack = []
stack.append(값1)    # push(값1) 효과
data = stack.pop()   # data = pop() 효과

if len(stack) == 0 :
    print('스택이 비었음')
```

5] 깊이 우선 탐색의 구현

깊이 우선 탐색 구현을 위한 준비

3 visitedAry 배열에 방문 정점을 저장해서 visitedAry 배열에 해당 정점이 있다면 방문한 적이 있는 것으로 처리

```
visitedAry = []
visitedAry.append(0)   # 정점 A(번호 0)를 방문했을 때
visitedAry.append(1)   # 정점 B(번호 1)를 방문했을 때

if 1 in visitedAry :
    print('A는 이미 방문함')
```

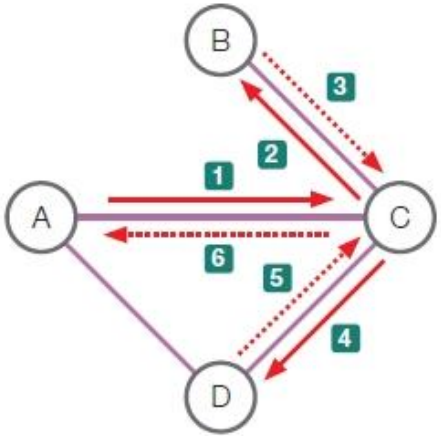
다음장 그림에서는 정점 이름이 A, B, C, D로 표현

```
for i in visitedAry :
    print(chr(ord('A')+i), end = ' ')
```

방문 기록 visitedAry 배열을 출력할 때는 알파벳으로 출력하도록 변경

깊이 우선 탐색의 단계별 구현

간단한 그래프를 깊이 우선 탐색하는 과정 구현 예



출발점 \ 도착점	A=0	B=1	C=2	D=3
A=0	0	0	1	1
B=1	0	0	1	0
C=2	1	1	0	1
D=3	1	0	1	0

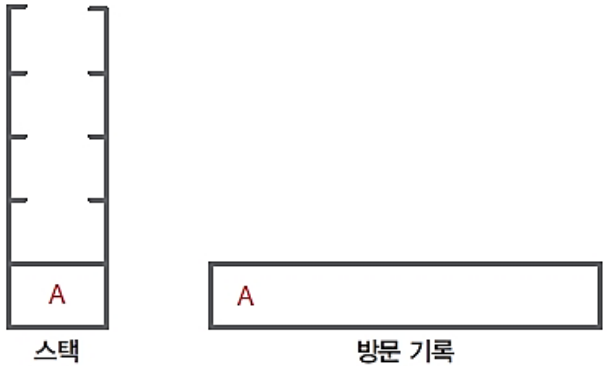
깊이 우선 탐색으로 탐샐할 그래프

깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

0 첫 번째 정점 방문

```
❶ current = 0    # 시작 정점
❷ stack.append(current)
❸ visitedAry.append(current)
```



5] 깊이 우선 탐색의 구현

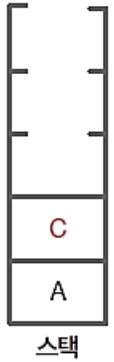
깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

1 과정(정점 C 방문)

```
❶ next = None
  for vertex in range(4) :
      if G1.graph[current][vertex] == 1 :
❷         next = vertex
          break

❸ current = next
❹ stack.append(current)
  visitedAry.append(current)
```



스택

방문 기록

깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

2 과정(정점 B 방문)

```
    next = None
❶   for vertex in range(4) :
        if G1.graph[current][vertex] == 1 :
❷           if vertex in visitedAry :   # 방문한 적이 있는 정점이면 탈락
                pass
❸           else :                      # 방문한 적이 없으면 다음 정점으로 지정
                next = vertex
                break

❹   current = next
❺   stack.append(current)
    visitedAry.append(current)
```

깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

2 과정(정점 D 방문)

깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

3 과정(되돌아오는 이동)

```
    next = None
❶   for vertex in range(4) :
        if G1.graph[current][vertex] == 1 :
❷           if vertex in visitedAry :    # 방문한 적이 있는 정점이면 탈락
                pass
❸           else :                       # 방문한 적이 없으면 다음 정점으로 지정
                next = vertex
                break
    if next != None :                    # 다음에 방문할 정점이 있는 경우
        current = next
        stack.append(current)
        visitedAry.append(current)
❹   else :                               # 다음에 방문할 정점이 없는 경우
❺       current = stack.pop()
```

깊이 우선 탐색의 단계별 구현

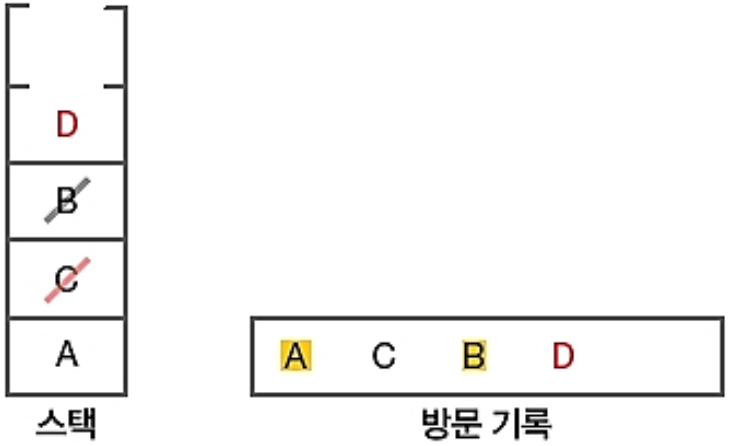
첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

3 과정 (되돌아오는 이동)

깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

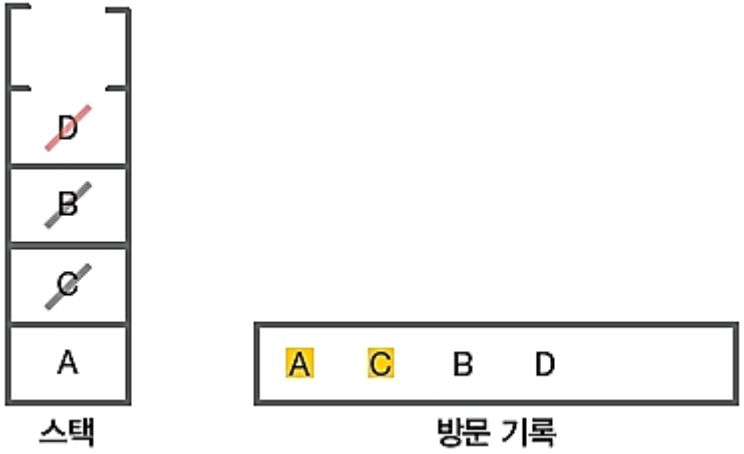
4 과정(정점 D 방문)



깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

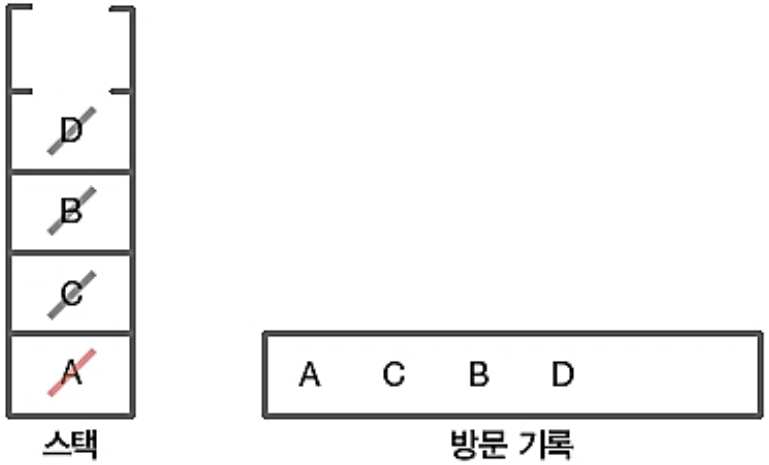
5 과정 (되돌아오는 이동)



깊이 우선 탐색의 단계별 구현

첫 번째 정점을 방문하는 것부터 시작하여 ❶ ~ ❻ 이동을 단계별로 구현

❻ 과정 (되돌아오는 이동)



깊이 우선 탐색의 구현

```
## 클래스와 함수 선언 부분 ##
class Graph() :
    def __init__ (self, size) :
        self.SIZE = size
        self.graph = [[0 for _ in range(size)] for _ in range(size)]

## 전역 변수 선언 부분 ##
G1 = None
stack = []
visitedAry = []    # 방문한 정점

## 메인 코드 부분 ##
G1 = Graph(4)
G1.graph[0][2] = 1; G1.graph[0][3] = 1
```

깊이 우선 탐색의 구현

```
15 G1.graph[1][2] = 1
16 G1.graph[2][0] = 1; G1.graph[2][1] = 1; G1.graph[2][3] = 1
17 G1.graph[3][0] = 1; G1.graph[3][2] = 1
18
19 print('## G1 무방향 그래프 ##')
20 for row in range(4) :
21     for col in range(4) :
22         print(G1.graph[row][col], end = ' ')
23     print()
24
25 current = 0        # 시작 정점
26 stack.append(current)
27 visitedAry.append(current)
28
```

깊이 우선 탐색의 구현

```
29 while (len(stack) != 0) :
30     next = None
31     for vertex in range(4) :
32         if G1.graph[current][vertex] == 1 :
33             if vertex in visitedAry :
34                 pass
35             else :
36                 next = vertex
37                 break
38
39     if next != None :
40         current = next
41         stack.append(current)
42         visitedAry.append(current)
```

방문한 적이 있는 정점이면 탈락

방문한 적이 없으면 다음 정점으로 지정

다음에 방문할 정점이 있는 경우

깊이 우선 탐색의 구현

```
43      else :
44          current = stack.pop()
45
46
47  print('방문 순서 -->', end = ' ')
48  for i in visitedAry :
49      print(chr(ord('A')+i), end = ' ')
```

다음에 방문할 정점이 없는 경우

실행 결과

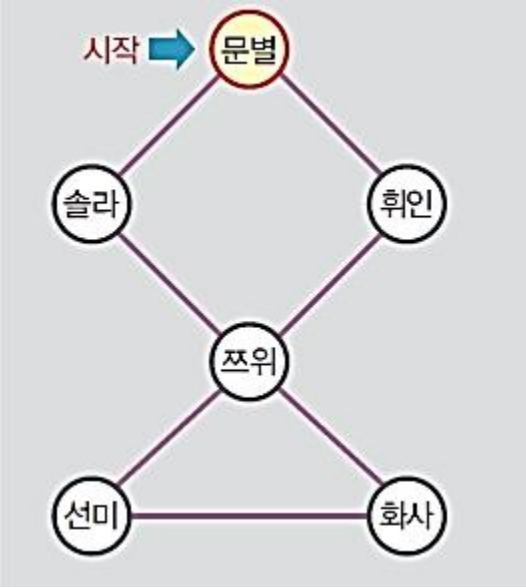
```
## G1 무방향 그래프 ##
0 0 1 1
0 0 1 0
1 1 0 1
1 0 1 0

방문 순서 -->A  C  B  D
```

5] 깊이 우선 탐색의 구현

깊이 우선 탐색의 구현 실습

앞쪽의 소스를 수정해서 다음 그림과 같은 무방향 그래프를 순회해 보자.



5] 깊이 우선 탐색의 구현

깊이 우선 탐색의 구현 실습

앞쪽의 소스를 수정해서 다음 그림과 같은 무방향 그래프를 순회해 보자.

실행 결과

```
방문 순서 -->문별    솔라    쯔위    휘인    선미    화사
```

깊이 우선 탐색의 구현 실습

학습평가

Q1 Q2 Q3

Q1 다음 중 그래프가 아닌 것은?

1. 지하철 노선도
2. 도시 도로망
3. 회사 조직도
4. 인맥 관계도

학습평가

Q1 Q2 Q3

Q1 다음 중 그래프가 아닌 것은?

1 지하철 노선도
2 도시 도로망
3 회사 조직도
4 인맥 관계도

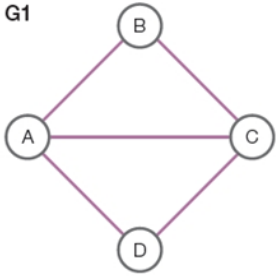
정 답 3 회사 조직도

해 설 회사 조직도는 트리 구조입니다.

학습평가

Q2

다음 무방향 그래프의 정점(Vertex) 집합을 표현한 것은?

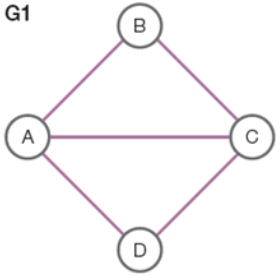


1. A, B, C, D
2. {A, B, C, D}
3. {(A,B), (B,D), (D,C)}
4. {(A,B), (A,C), (A,D), (B,C), (C,D)}

학습평가

Q1 Q2 Q3

Q2 다음 무방향 그래프의 정점(Vertex) 집합을 표현한 것은?



1. A, B, C, D
2. {A, B, C, D}
3. {(A,B), (B,D), (D,C)}
4. {(A,B), (A,C), (A,D), (B,C), (C,D)}

정 답 2 {A, B, C, D}

해 설 정점은 A,B,C,D이고, 집합으로 표현하기 위해서는 { }를 사용해야 합니다.

학습평가

Q1 Q2 Q3

Q3 그래프를 깊이 우선 탐색으로 구현하는 데 필요한 자료구조는?

1. 선형 리스트
2. 연결 리스트
3. 스택
4. 큐

학습평가

Q3 그래프를 깊이 우선 탐색으로 구현하는 데 필요한 자료구조는?

1 선형 리스트
2 연결 리스트
3 스택
4 큐

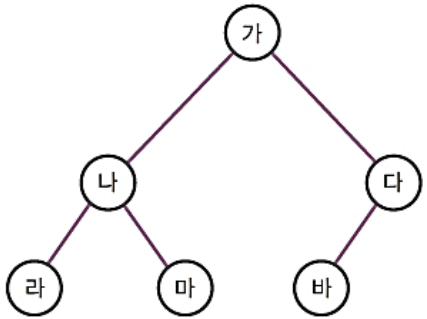
정 답 3 스택

해 설 깊이 우선 탐색을 위해서는 스택이 필요합니다.

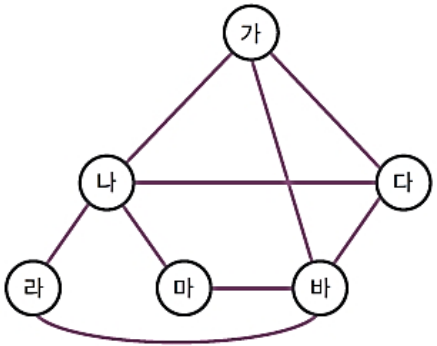
그래프의 기본

그래프의 개념

- 여러 노드가 서로 연결된 자료구조
- 트리와 그래프의 차이



(a) 트리 예



(b) 그래프 예

그래프의 기본

그래프의 종류

- 무방향 그래프 : 간선에 방향성이 없는 그래프
- 방향 그래프 : 화살표로 간선 방향을 표기하고, 그래프의 정점 집합이 무방향 그래프와 같음
- 가중치 그래프 : 간선마다 가중치가 다르게 부여된 그래프

그래프의 기본

깊이 우선 탐색의 작동

- 그래프의 모든 정점을 한 번씩 방문하는 것을 그래프 순회(Graph Traversal)라고 함

그래프의 인접 행렬 표현

- 그래프를 코드로 구현할 때는 인접 행렬을 사용
- 인접 행렬은 정방형으로 구성된 행렬로 정점이 4개인 그래프는 4×4로 표현

그래프의 구현

그래프의 정점 생성

- 행과 열이 같은 2차원 배열을 생성하는 클래스로 작성

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

G1 = Graph(4)
```