15 강

알고리즘과 자료구조

그래프

서울과학기술대학교 신일훈 교수

한국방송통신대학교 프리임칼리지

학습목표

1. 그래프의 개념을 이해한다.
2. 그래프의 표현 및 저장 방법을 이해한다.
3. 그래프의 탐색 방법을 이해하고 이를 구현한다.

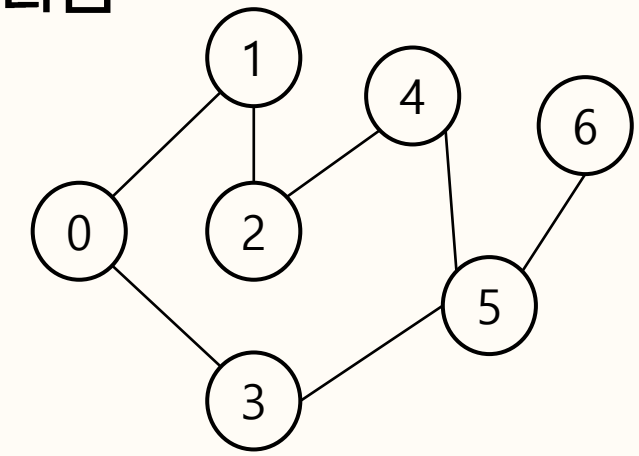
01

그래프 개념

1. 그래프 개념

그래프란?

- 정점(vertex, 노드)과 정점을 연결하는 간선(edge, 링크)으로 구성된 자료구조
  - 정점은 보통 객체, 아이템 등을 나타냄
  - 간선은 정점 간의 관계를 표현
  - G = (V, E)



1. 그래프 개념

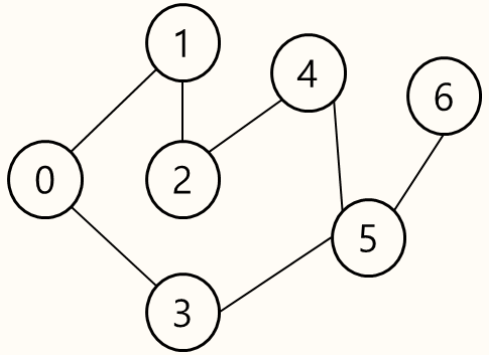
그래프 예제

- SNS에서 사람들의 관계
- 교통망 (도시를 연결하는 도로), 지하철 노선도
- 신경망
- 교과목 이수체계

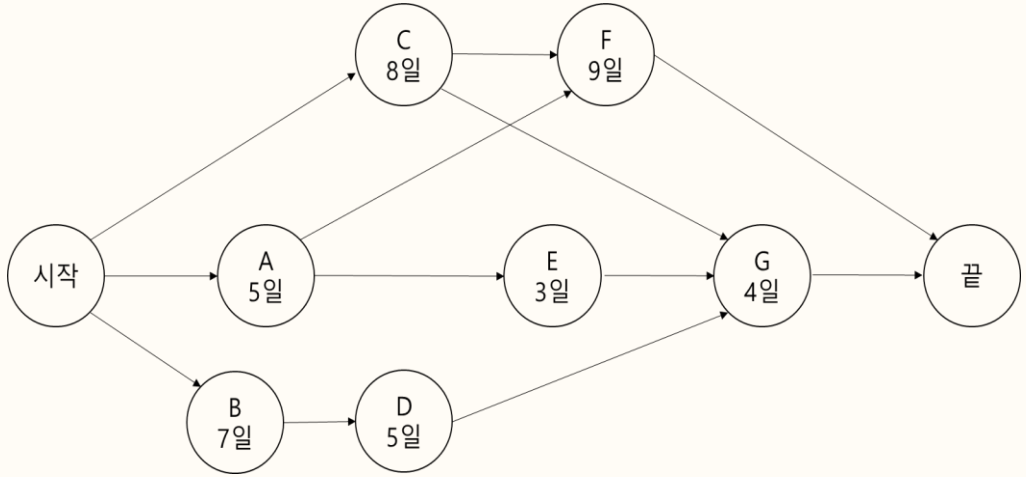
1. 그래프 개념

그래프 종류

- 무방향(undirected) 그래프
  - 간선의 방향성이 없음



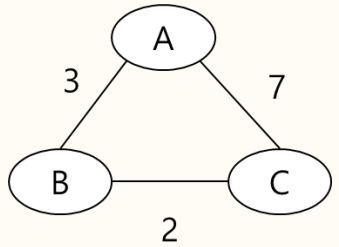
- 방향(directed) 그래프
  - 간선의 방향성이 있음
  - 교과목이수체계, 버스노선도, …



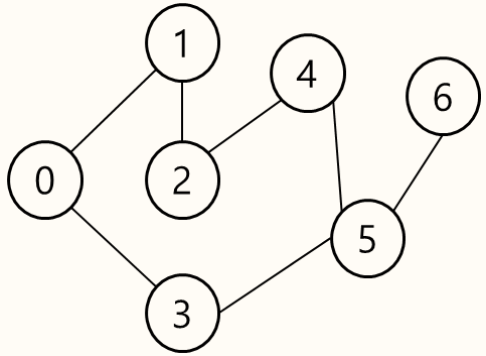
1. 그래프 개념

그래프 종류

- 가중치(weighted) 그래프
  - 간선에 비용 또는 가중치가 할당된 그래프
  - 교통망, …



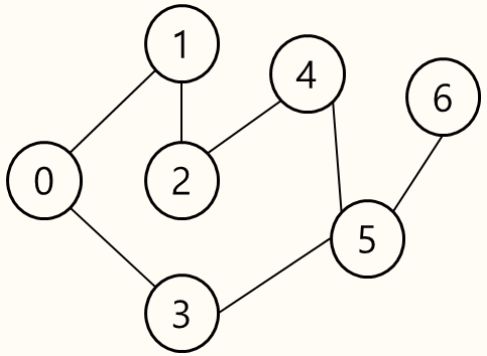
- 가중치가 없는 그래프
  - 간선의 비용이 동일한 그래프
  - 교과목 이수 체계도, …



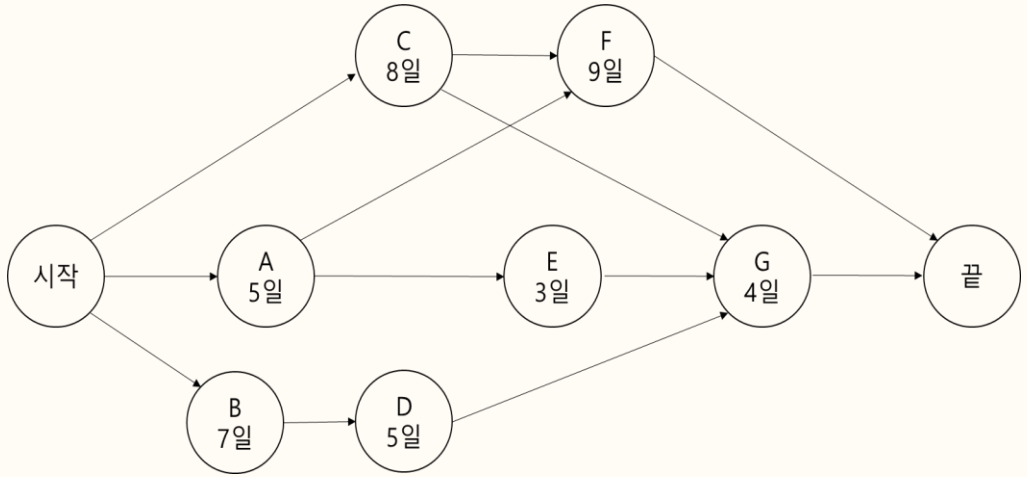
1. 그래프 개념

용어

- 차수(Degree)
  - 타겟 정점과 간선으로 연결된 정점의 수
  - 점점 0의 차수는?



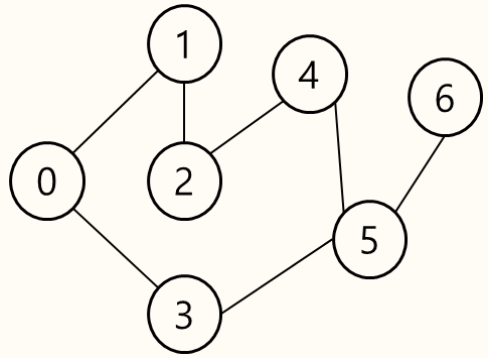
- 진입(In-degree) 차수
  - 방향 그래프에서 정점으로 들어오는 간선의 수
- 진출(Out-degree) 차수
  - 방향 그래프에서 정점에서 나가는 간선의 수



1. 그래프 개념

용어

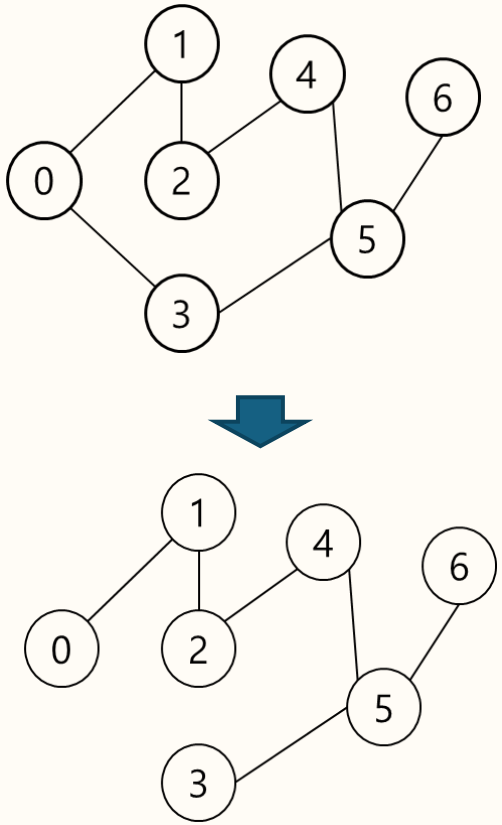
- 경로(Path)
  - 시작 정점부터 도착 정점까지의 정점들을 나열하여 표현
  - [0, 3, 5]: 정점 0으로부터 정점 5까지의 경로들 중 하나
- 단순 경로
  - 경로 상의 정점들이 모두 다른 경로
- 사이클(Cycle)
  - 시작 정점과 도착 정점이 동일한 경로
  - [0, 1, 2, 4, 5, 3, 0]



1. 그래프 개념

용어

- 트리(Tree): 사이클이 없는 그래프
- 신장(spanning) 트리
  - 그래프의 모든 정점을 사이클 없이 연결하는 부분 그래프



신장트리

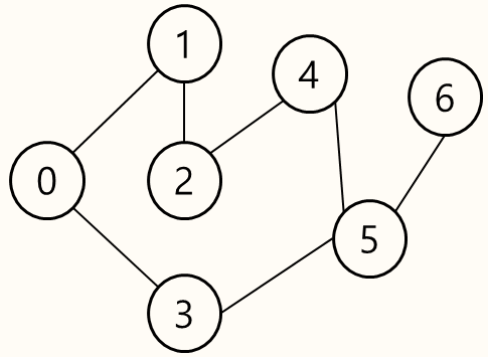
02

그래프의 표현과 저장

2. 그래프의 표현과 저장

그래프의 표현

- V(G) = {0, 1, 2, 3, 4, 5, 6}
- E(G) = {(0, 1), (0, 3), (1, 2), (2, 4), (3, 5), (4, 5), (5, 6)}



2. 그래프의 표현과 저장

그래프의 저장

- 인접 행렬 (adjacency matrix)
- 인접 리스트 (adjacency list)

2. 그래프의 표현과 저장

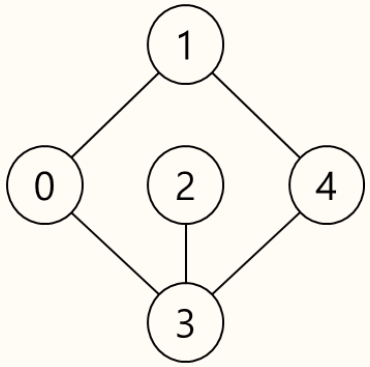
❑ 인접 행렬

- 2차원 배열 활용
- 행 정점과 열 정점을 연결하는 간선이 있으면 해당 배열 원소의 값이 1 또는 가중치 값, 아니면 0

2. 그래프의 표현과 저장

인접 행렬

- 예시
  - vertex = [0, 1, 2, 3, 4]
  - adjMatx = [[0, 1, 0, 1, 0],
    [1, 0, 0, 0, 1],
    [0, 0, 0, 1, 0],
    [1, 0, 1, 0, 1],
    [0, 1, 0, 1, 0]]



- 정점의 수가 V라면 V*V 크기의 메모리 공간이 필요함
- 메모리 사용량 측면에서, 간선의 수가 많은 그래프를 표현할 때 상대적으로 유리

2. 그래프의 표현과 저장

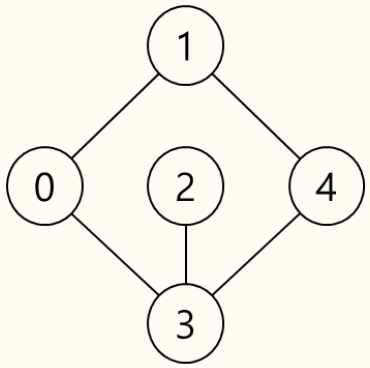
인접 리스트

- 여러 연결 리스트를 활용
- 각 정점과 연결된 정점 정보를 리스트로 저장

2. 그래프의 표현과 저장

❑ 인접 리스트

- 예시
  - vertex = [0, 1, 2, 3, 4]
  - adjMatx = [[1, 3],
    [0, 4],
    [3],
    [0, 2, 4],
    [1, 3]]



- 정점의 개수가 V라면 V개의 연결 리스트가 필요하며,
  전체 간선의 수가 E라면 2*E개의 연결리스트 노드가 필요함.
- 메모리 사용량 측면에서 간선의 수가 적은 그래프를 표현할 때 상대적으로 유리

03

그래프 탐색

3. 그래프 탐색

종류

- 깊이우선탐색 (DFS)
- 너비우선탐색 (BFS)

3. 그래프 탐색

❑ 깊이우선탐색

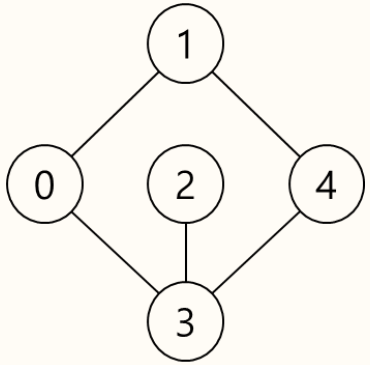
- 알고리즘

1. 임의의 정점을 출발점으로 하여 방문 시작
2. 현재 출발점의 이웃 정점 중, 아직 방문하지 않은 정점 중 하나를 방문
3. 이 정점을 새로운 출발점으로 하여 2번을 반복.
4. 이 때 방문하지 않은 연결된 정점이 없으면 직전 정점으로
   되돌아가서 직전 정점을 새로운 출발점으로 하여 2번을 반복함
5. 모든 연결된 정점을 방문하면 완료.

3. 그래프 탐색

깊이우선탐색

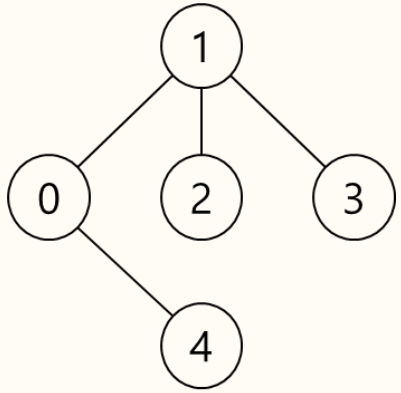
- 예시
  - 0에서 시작
  - 방문 가능한 정점들이 여러 개인 경우 낮은 숫자의 정점을 먼저 방문한다고 가정
  - 0 -> 1 -> 4 -> 3 -> 2



3. 그래프 탐색

깊이우선탐색

- 예시
  - 0에서 시작
  - 방문 가능한 정점들이 여러 개인 경우 낮은 숫자의 정점을 먼저 방문한다고 가정
  - 0 -> 1 -> 2 -> 3 -> 4



3. 그래프 탐색

너비우선탐색

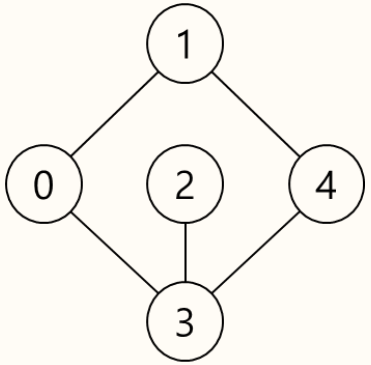
- 알고리즘

1. 임의의 정점을 출발점으로 하여 방문 시작
2. 출발점의 이웃 정점 중, 아직 방문하지 않은 모든 이웃 정점들을 방문
3. 모든 이웃 정점 방문 후, 이전에 방문했으나
   아직 출발점이 되지 못한 정점들 중에서 가장 먼저 방문한 정점을
   새로운 출발점으로 하여 2번을 반복.
4. 모든 연결된 정점을 방문하면 완료.

3. 그래프 탐색

너비우선탐색

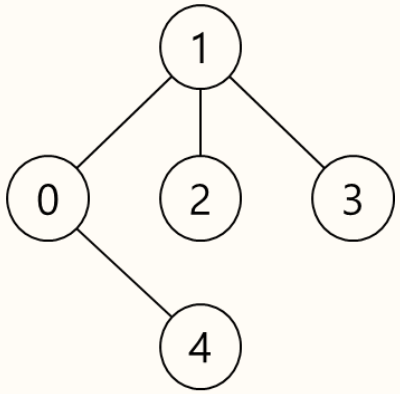
- 예시
  - 0에서 시작
  - 방문 가능한 정점들이 여러 개인 경우 낮은 숫자의 정점을 먼저 방문한다고 가정
  - 0 -> 1 -> 3 -> 4 -> 2



3. 그래프 탐색

너비우선탐색

- 예시
  - 0에서 시작
  - 방문 가능한 정점들이 여러 개인 경우 낮은 숫자의 정점을 먼저 방문한다고 가정
  - 0 -> 1 -> 4 -> 2 -> 3

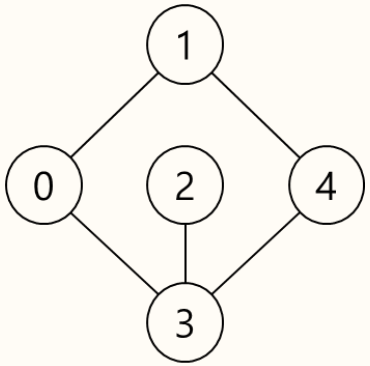


04

깊이우선탐색 구현

4. 깊이우선탐색 구현

- 그래프 표현 (인접 리스트)
  - graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]



4. 깊이우선탐색 구현

- 클래스 Graph 정의
  - 그래프를 나타냄
  - 멤버 변수
    - adjlist: 인접 리스트로 표현한 그래프
    - vertex_count: 정점의 개수
    - visited: 정점 별로 방문 여부를 나타내는 리스트

4. 깊이우선탐색 구현

- 클래스 Graph 정의
  - 그래프를 나타냄
  - 메서드
    - 생성자
    - dfs()

4. 깊이우선탐색 구현

클래스 Graph 정의 (생성자)

```
class Graph:
    def __init__(self, graph):
        self.adjlist = graph
        self.vertex_count = len(graph)
        self.visited = [False] * self.vertex_count
```

4. 깊이우선탐색 구현

클래스 Graph 정의 (dfs() 의사코드)

1. 임의의 정점을 출발점으로 하여 방문 시작
2. 현재 출발점의 이웃 정점 중, 아직 방문하지 않은 정점 중 하나를 방문
3. 이 정점을 새로운 출발점으로 하여 2번을 반복.
4. 이 때 방문하지 않은 연결된 정점이 없으면 직전 정점으로 되돌아가서 직전 정점을 새로운 출발점으로 하여 2번을 반복함
5. 모든 연결된 정점을 방문하면 완료.

➡ 재귀 또는 스택을 활용하여 구현 가능함

4. 깊이우선탐색 구현

클래스 Graph 정의 (dfs())

```
class Graph :
    def dfs(self, start) :
        for vertex in range(self.vertex_count) :
            self.visited[vertex] = False

        self.dfs_recursive(start)
        print()
```

4. 깊이우선탐색 구현

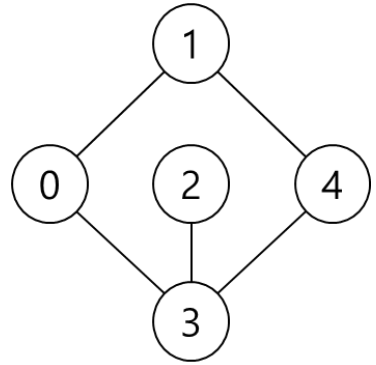
클래스 Graph 정의 (dfs())

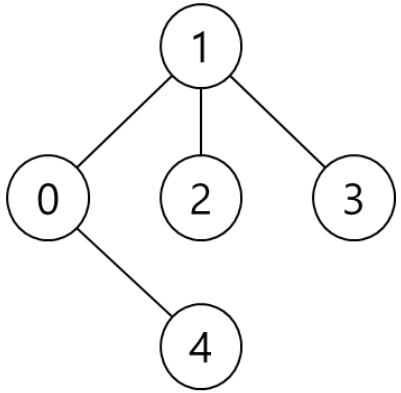
```
class Graph :
    def dfs_recursive(self, vertex):
        self.visited[vertex] = True
        print(vertex, ' ', end='')
        for neighbor in self.adjlist[vertex] :
            if not self.visited[neighbor] :
                self.dfs_recursive(neighbor)
```

4. 깊이우선탐색 구현

DFS 테스트

```
if __name__ == '__main__':
    graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]
    graph = Graph(graph_adjlist)
    graph.dfs(0)

    graph_adjlist = [[1, 4], [0, 2, 3], [1], [1], [0]]
    graph = Graph(graph_adjlist)
    graph.dfs(0)
```

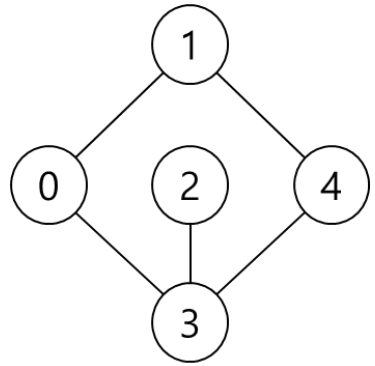


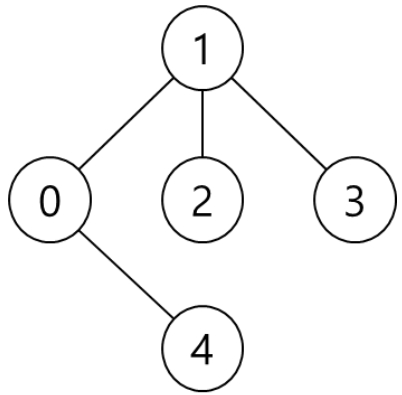


4. 깊이우선탐색 구현

DFS 테스트 실행

```
0  1  4  3  2
0  1  2  3  4
```

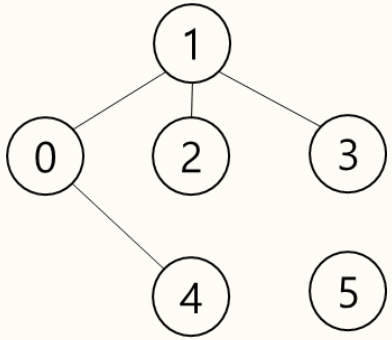




4. 깊이우선탐색 구현

DFS 구현 특성

- 그래프가 연결이 단절된 부분 그래프로 구성된 경우,
  하나의 부분 그래프만 방문하게 됨
- 연결되지 않은 정점은 방문하지 못함



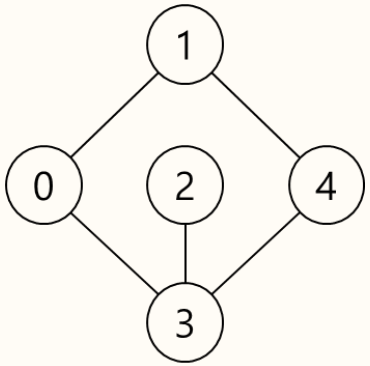
05

너비우선탐색 구현

5. 너비우선탐색 구현

그래프 표현 (인접 리스트)

- graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]



5. 너비우선탐색 구현

- 클래스 Graph 정의
  - 그래프를 나타냄
  - 멤버 변수
    - adjlist: 인접리스트로 표현한 그래프
    - vertex_count: 정점의 개수
    - visited: 정점 별로 방문 여부를 나타내는 리스트
    - bfsQ: 너비우선탐색에 사용할 큐

5. 너비우선탐색 구현

- 클래스 Graph 정의
  - 그래프를 나타냄
  - 메서드
    - 생성자
    - dfs()
    - bfs()

5. 너비우선탐색 구현

클래스 Graph 정의 (생성자)

```
import Queue
class Graph:
    def __init__(self, graph):
        self.adjlist = graph
        self.vertex_count = len(graph)
        self.visited = [False] * self.vertex_count
        self.bfsQ = Queue.Queue()
```

5. 너비우선탐색 구현

클래스 Graph 정의 (bfs() 의사코드)

1. 임의의 정점을 출발점으로 하여 방문 시작
2. 출발점의 이웃 정점 중, 아직 방문하지 않은 모든 이웃 정점들을 방문
3. 모든 이웃 정점 방문 후, 이전에 방문했으나 아직 출발점이 되지 못한 정점들 중에서 가장 먼저 방문한 정점을 새로운 출발점으로 하여 2번을 반복.
4. 모든 연결된 정점을 방문하면 완료.

➡ 큐를 활용하여 구현 가능함

5. 너비우선탐색 구현

클래스 Graph 정의 (bfs())

```
class Graph :
    def bfs(self, start) :
        for vertex in range(self.vertex_count) :
            self.visited[vertex] = False

        self.bfsQ.enqueue(start)
        self.visited[start] = True
        self.do_bfs()
        print()
```

5. 너비우선탐색 구현

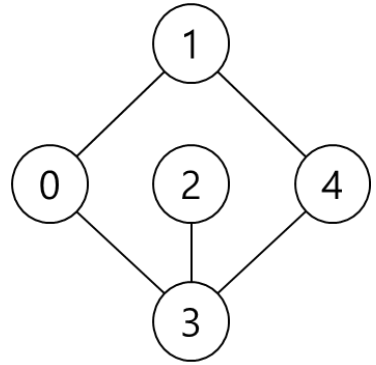
클래스 Graph 정의 (bfs())

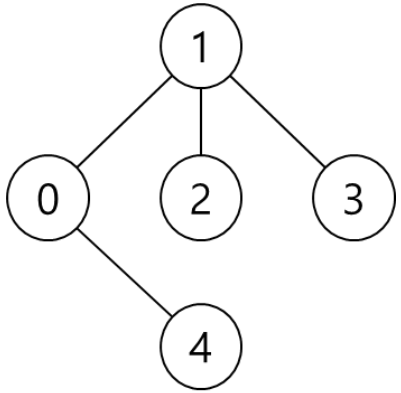
```
class Graph :
    def do_bfs(self):
        while self.bfsQ.get_size() > 0 :
            vertex = self.bfsQ.dequeue()
            print(vertex, ' ', end= ' ')

            for neighbor in self.adjlist[vertex] :
                if not self.visited[neighbor] :
                    self.bfsQ.enqueue(neighbor)
                    self.visited[neighbor] = True
```

5. 너비우선탐색 구현

BFS 테스트

```
if __name__ == '__main__':
    graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]
    graph = Graph(graph_adjlist)
    graph.bfs(0)

    graph_adjlist = [[1, 4], [0, 2, 3], [1], [1], [0]]
    graph = Graph(graph_adjlist)
    graph.bfs(0)
```

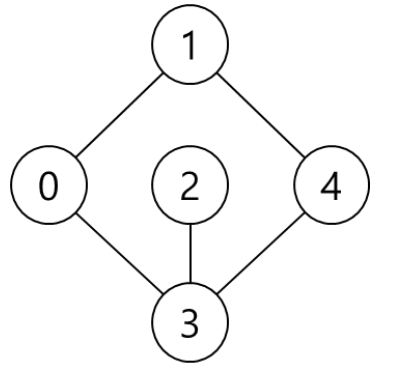


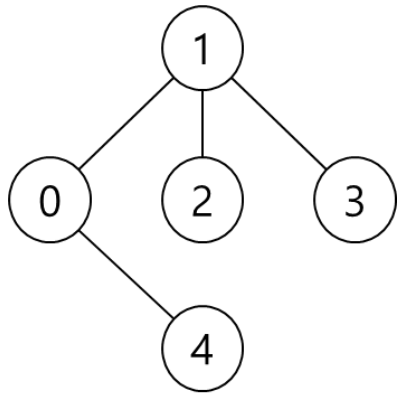


5. 너비우선탐색 구현

BFS 테스트 실행

```
Reloaded modules: CList, Queue
0   1   3   4   2
0   1   4   2   3
```





정리하기

- 그래프의 개념
- 그래프의 표현과 저장
- 그래프의 탐색(깊이우선탐색과 너비우선탐색)
- 그래프의 탐색 구현(깊이우선탐색과 너비우선탐색)

수고하셨습니다.