알고리즘과 자료구조 워크북

교과목명 : 알고리즘과 자료구조
차시명: 15차시 그래프

◈ 담당교수: 신일훈 **(서울과학기술대학교)**

◉ 세부목차
- 그래프의 개념
- 그래프의 저장과 표현
- 그래프의 탐색 방법과 탐색의 구현

학습에 앞서

■ 학습개요

그래프는 객체를 나타내는 정점과 각 정점을 연결하는 간선으로 구성된 자료구조이다. 그래프는 네트워크 연결도, SNS에서 사람들의 관계, 각 도시를 연결하는 도로망 등의 데이터를 표현할 수 있다. 본 강에서는 그래프의 개념을 이해하고 그래프를 프로그램에서 표현하기 위한 방법과 그래프의 각 정점을 탐색하는 방법을 배우고 이를 파이썬으로 구현한다.

■ 학습목표

1	그래프의 개념을 이해한다.
2	그래프의 표현 및 저장 방법을 이해한다.
3	그래프의 탐색 방법을 이해하고 이를 구현할 수 있다.

■ 주요용어

용어	해설
그래프	정점과 정점을 연결하는 간선으로 구성된 자료구조이다. 트리와의 차이점은 트리는 사이클을 포함하지 않는 데 반해, 그래프는 사이클을 포함할 수 있다는 것이다.
신장트리	그래프의 모든 정점을 사이클 없이 연결하는 부분 그래프
깊이우선탐색	그래프의 정점들을 탐색하는 방법 중 하나로서, 임의의 정점을 출발점으로 방문을 시작하고, 현재 출발점의 이웃 정점들 중 아직 방문하지 않은 정점을 방문한다. 그리고 방문한 이 정점을 새로운 출발점으로 하여 동일한 작업을 반복한다. 만약 연결된 정점이 없다면 직전에 방문한 정점으로 되돌아가서 직전 정점을 새로운 출발점으로 하여 동일한 작업을 반복한다. 모든 연결된 정점을 방문하면 완료한다.
너비우선탐색	그래프의 정점들을 탐색하는 방법 중 하나로서, 임의의 정점을 출발점

	으로 방문을 시작하고, 현재 출발점의 이웃 정점들 중, 아직 방문하지 않은 모든 정점들을 방문한다. 모든 이웃 정점 방문 후, 이전에 방문했으나 아직 출발점이 되지 못한 정점들 중에서 가장 먼저 방문한 정점을 새로운 출발점으로 하여 동일한 작업을 반복한다. 모든 연결된 정점을 방문하면 완료한다.

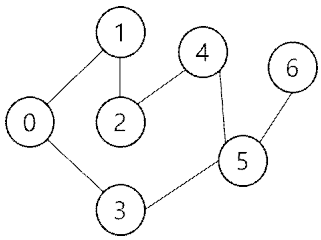
	학습하기

	학습하기

1. 그래프 개념

가. 개념

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



- 유사한 비선형 자료구조인 트리와의 차이점은 사이클이 존재할 수 있다는 점이다. 트리는 그래프의 일종이다.
- 그래프를 활용하는 예제
  - SNS에서 사람들의 관계
  - 교통망 (도시를 연결하는 도로), 지하철 노선도
  - 신경망
  - 교과목 이수체계
  - ...

나. 종류

- 무방향(undirected) 그래프
  - 간선의 방향성이 없음
- 방향(directed) 그래프
  - 간선의 방향성이 있음
- 가중치(weighted) 그래프
  - 간선에 비용 또는 가중치가 할당된 그래프
- 가중치가 없는 그래프
  - 간선의 비용이 동일한 그래프

다. 용어

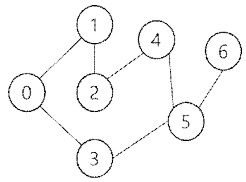
- 차수(Degree): 타겟 정점과 간선으로 연결된 정점의 수
- 진입(In-degree) 차수: 방향 그래프에서 정점으로 들어오는 간선의 수
- 진출(Out-degree) 차수: 방향 그래프에서 정점에서 나가는 간선의 수

- 경로(Path): 시작 정점부터 도착 정점까지의 정점들을 나열하여 표현
- 단순 경로: 경로 상의 정점들이 모두 다른 경로
- 사이클(Cycle): 시작 정점과 도착 정점이 동일한 경로
- 트리(Tree): 사이클이 없는 그래프
- 신장(spanning) 트리: 그래프의 모든 정점을 사이클 없이 연결하는 부분 그래프

2. 그래프의 저장과 표현

가. 그래프의 표현

- 그래프는 정점과 이를 연결하는 간선으로 구성되므로 정점의 집합과 간선의 집합으로 표현됨.



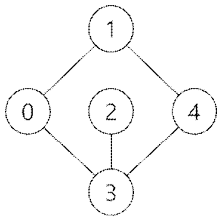
- 위 그래프는 7개의 정점과 이를 연결하는 간선으로 구성됨. 따라서 다음과 같이 표현될 수 있음.
  - V(G) = {0, 1, 2, 3, 4, 5, 6}
  - E(G) = {(0, 1), (0, 3), (1, 2), (2, 4), (3, 5), (4, 5), (5, 6)}

나. 그래프의 저장

- 그래프 데이터를 프로그램에서 표현하고 저장하는 방식에는 크게 인접행렬을 활용한 방식과 인접리스트를 활용한 방식이 있음.

다. 인접행렬

- 2차원 배열 활용
- 행 정점과 열 정점을 연결하는 간선이 있으면 해당 배열 원소의 값을 1 또는 간선의 가중치 값으도 함.
- 행 정점과 열 정점을 연결하는 간선이 없으면 해당 배열 원소의 값을 0으로 함.
- 예시



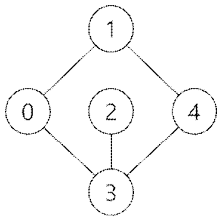
  - vertex = [0, 1, 2, 3, 4]
  - adjMatx = [[0, 1, 0, 1, 0],
    [1, 0, 0, 0, 1],
    [0, 0, 0, 1, 0],
    [1, 0, 1, 0, 1],
    [0, 1, 0, 1, 0]]

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

라. 인접리스트

- 여러 연결 리스트를 활용하여 그래프를 표현
- 각 정점과 연결된 정점 정보를 하나의 리스트로 저장

- 예시



- vertex = [0, 1, 2, 3, 4]
- adjList = [[1, 3],
  [0, 4],
  [3],
  [0, 2, 4],
  [1, 3]]
- adjList의 첫 번째 행은 정점 0과 연결되어 있는 정점을 나열함. 즉 위의 그래프에서 1, 3이 연결되어 있으므로 해당 정점들을 나열함.
- 동일하게, adjList의 두 번째 행은 정점 1과 연결되어 있는 정점을 나열함. 즉 위의 그래프에서 0, 4와 연결되어 있으므로 해당 정점들을 나열함.

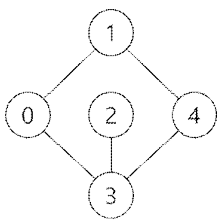
- 정점의 개수가 V라면 V개의 연결 리스트가 필요하며, 전체 간선의 수가 E라면 2*E개의 연결 리스트 노드가 필요함.
- 따라서 메모리 사용량 측면에서 보면, 간선의 수가 적은 그래프를 표현할 때 상대적으로 유리함

3. 그래프의 탐색
- 깊이우선탐색(DFS)과 너비우선탐색(BFS)의 두 가지 방식이 있다.

가. 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.

- 예시1

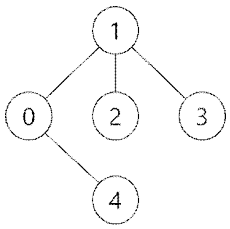


- 0에서 시작
- 방문 가능한 정점들이 여러 개인 경우 낮은 숫자의 정점을 먼저 방문한다고 가정
- 결과: 0 -> 1 -> 4 -> 3 -> 2

- 0을 가장 먼저 방문하고, 0에서 연결된 방문하지 않은 정점은 1, 3인데 1이 숫자가 낮으므로 1을 방문한다. 1은 새로운 출발점이 된다.

- 1에서 연결된 방문하지 않은 정점은 4이므로 4를 방문한다. 4는 새로운 출발점이 된다.
- 4에서 연결된 방문하지 않은 정점은 3이므로 3을 방문한다. 3은 새로운 출발점이 된다.
- 3에서 연결된 방문하지 않은 정점은 2이므로 2를 방문한다. 2는 새로운 출발점이 된다.
- 2에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 3으로 돌아온다.
- 3에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 4로 돌아온다.
- 4에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 1로 돌아온다.
- 1에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 0으로 돌아온다.
- 0에서 연결된 방문하지 않은 정점은 없고 직전 방문 정점도 없으므로 탐색을 종료한다.

- 예시2



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

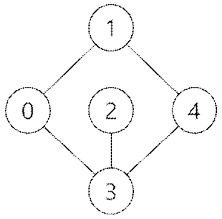
- 0을 가장 먼저 방문하고, 0에서 연결된 방문하지 않은 정점은 1, 4인데 1이 숫자가 낮으므로 1을 방문한다. 1은 새로운 출발점이 된다.
- 1에서 연결된 방문하지 않은 정점은 2, 3인데 2가 숫자가 낮으므로 2를 방문한다. 2는 새로운 출발점이 된다.
- 2에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 1로 돌아온다.
- 1에서 연결된 방문하지 않은 정점은 3이므로 3을 방문한다. 3은 새로운 출발점이 된다.
- 3에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 1로 돌아온다.
- 1에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 0으로 돌아온다.
- 0에서 연결된 방문하지 않은 정점은 4이므로 4를 방문한다. 4는 새로운 출발점이 된다.
- 4에서 연결된 방문하지 않은 정점은 없으므로 직전 방문 정점인 0으로 돌아온다.
- 0에서 연결된 방문하지 않은 정점은 없고 직전 방문 정점도 없으므로 탐색을 종료한다.

나. 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.

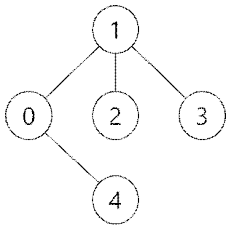
- 예시1



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

- 0을 가장 먼저 방문하고, 0에서 연결된 방문하지 않은 정점은 1, 3이므로 이들을 차례로 방문한다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 1, 3인데 1을 먼저 방문했으므로 1이 새로운 출발점이 된다.
- 1에서 연결된 방문하지 않은 정점은 4이므로 4를 방문한다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 3, 4인데 3을 먼저 방문했으므로 3이 새로운 출발점이 된다.
- 3에서 연결된 방문하지 않은 정점은 2이므로 2를 방문한다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 4, 2인데 4를 먼저 방문했으므로 4가 새로운 출발점이 된다.
- 4에서 연결된 방문하지 않은 정점은 없다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 2으므로 2가 새로운 출발점이 된다.
- 2에서 연결된 방문하지 않은 정점은 없다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점이 존재하지 않으므로 탐색을 종료한다.

- 예시2



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

- 0을 가장 먼저 방문하고, 0에서 연결된 방문하지 않은 정점은 1, 4이므로 이들을 차례로 방문한다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 1, 4인데 1을 먼저 방문했으므로 1이 새로운 출발점이 된다.
- 1에서 연결된 방문하지 않은 정점은 2, 3이므로 이들을 차례로 방문한다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 4, 2, 3인데 4를 먼저 방문했으므로 4가 새로운 출발점이 된다.
- 4에서 연결된 방문하지 않은 정점은 없다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 2, 3인데 2를 먼저 방문했으므로 2가 새로운 출발점이 된다.
- 2에서 연결된 방문하지 않은 정점은 없다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점은 3이므로 3이 새로운 출발점이 된다.
- 3에서 연결된 방문하지 않은 정점은 없다.
- 방문한 정점 중 아직 출발점이 되지 못한 정점이 존재하지 않으므로 탐색을 종료한다.

4. 깊이우선탐색 구현

가. 클래스 Graph 정의

- 그래프를 나타냄
- 멤버 변수
  - adjlist: 인접 리스트로 표현한 그래프
  - vertex_count: 정점의 개수
  - visited: 정점 별로 방문 여부를 나타내는 리스트
- 메서드
  - 생성자
  - dfs()
  - ...

나. 클래스 Graph 생성자 코드

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

- 인자로 전달된 graph는 그래프 데이터를 인접 리스트로 표현한다고 가정한다.
- 따라서 그래프의 원소의 개수를 구하면 정점의 개수를 알 수 있다. (len())
- 모든 정점을 아직 방문하지 않았으므로 visited는 모두 False로 초기화된다.

다. 클래스 Graph의 dfs() 메서드 코드

- 깊이우선탐색을 수행하는 메서드이다.
- 스택을 활용하여 구현할 수도 있고 재귀함수를 활용할 수도 있다.
- 여기서는 재귀함수를 활용하여 구현한다.

```
1.      def dfs(self, start) :
2.              for vertex in range(self.vertex_count) :
3.                      self.visited[vertex] = False
4.
5.              self.dfs_recursive(start)
6.              print()
```

- 2-3번 라인에서, 모든 정점을 아직 방문하지 않은 상태로 초기화한다. 그래프 객체 생성 후 dfs()를 여러 번 호출할 수 있으므로, dfs()가 호출될 때마다 초기화를 수행했다.
- 5번 라인에서 인자로 전달된 start 정점을 출발점으로 하여 dfs_recursive()를 호출함으로써 깊이우선탐색을 수행한다.
- dfs_recursive()가 종료되면, 모든 정점의 방문이 끝났으므로 한 행을 띄운다.

- dfs_recursive()의 구현은 다음과 같다.

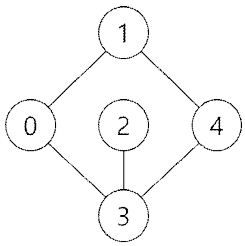
```
1.      def dfs_recursive(self, vertex):
2.              self.visited[vertex] = True
3.              print(vertex, ' ', end='')
4.              for neighbor in self.graph[vertex] :
5.                      if not self.visited[neighbor] :
6.                              self.dfs_recursive(neighbor)
```

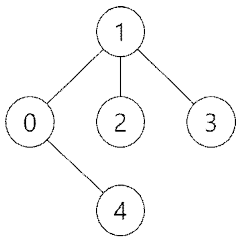
- 인자로 전달된 vertex가 출발점이므로 이를 출력하고 visited를 True로 한다.
- vertex에서 연결된 정점 중 아직 방문하지 않은 정점이 새로운 출발점이 되어야 하므로 4번 라인의 for

문을 통해 vertex와 연결된 이웃 정점들을 알아내고 이들 중 visited가 False인 정점을 새로운 출발점으로 하여 재귀함수를 호출한다.

- 만약 방문하지 않은 이웃 정점이 없다면 현재 호출된 dfs_recursive()는 종료된다.

라. 깊이우선탐색 테스트 코드 및 실행 결과






```
1. if __name__ == '__main__':
2.         graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]
3.         graph = Graph(graph_adjlist)
4.         graph.dfs(0)
5.
6.         graph_adjlist = [[1, 4], [0, 2, 3], [1], [1], [0]]
7.         graph = Graph(graph_adjlist)
8.         graph.dfs(0)
```

- 2번 라인에서 왼쪽 그래프를 인접리스트로 표현한다.
- 3번 라인에서 그래프 객체를 생성한다.
- 4번 라인에서 출발점을 0으로 하여 깊이우선탐색을 수행한다.
- 0 -> 1 -> 4 -> 3 -> 2 순서로 방문을 수행해야 한다.

- 6번 라인에서 오른쪽 그래프를 인접리스트로 표현한다.
- 7번 라인에서 그래프 객체를 생성한다.
- 8번 라인에서 출발점을 0으로 하여 깊이우선탐색을 수행한다.
- 0 -> 1 -> 2 -> 3 -> 4 순서로 방문을 수행해야 한다.

- 다음은 이에 대한 실행 결과이다.

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

5. 너비우선탐색 구현

가. 클래스 Graph 정의

- 그래프를 나타냄
- 멤버 변수
  - adjlist: 인접 리스트로 표현한 그래프
  - vertex_count: 정점의 개수
  - visited: 정점 별로 방문 여부를 나타내는 리스트
  - bfsQ: 너비우선탐색에 사용할 큐

- 메서드
  - 생성자
  - bfs()
  - …

나. 클래스 Graph 생성자 코드

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

- 너비우선탐색은 큐를 활용하여 구현하므로 큐가 구현된 Queue.py를 import한다.
- 인자로 전달된 graph는 그래프 데이터를 인접 리스트로 표현한다고 가정한다.
- 따라서 그래프의 원소의 개수를 구하면 정점의 개수를 알 수 있다. (len())
- 모든 정점을 아직 방문하지 않았으므로 visited는 모두 False로 초기화된다.
- 사용할 빈 큐를 생성한다.

다. 클래스 Graph의 bfs() 메서드 코드

- 너비우선탐색을 수행하는 메서드이다.
- 큐를 활용하여 구현한다.

```
1.          def bfs(self, start) :
2.                  for vertex in range(self.vertex_count) :
3.                          self.visited[vertex] = False
4.
5.                  self.bfsQ.enqueue(start)
6.                  self.visited[start] = True
7.                  self.do_bfs()
8.                  print()
```

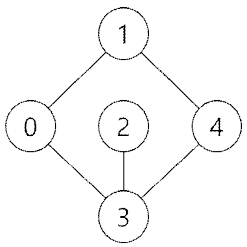
- 2-3번 라인에서, 모든 정점을 아직 방문하지 않은 상태로 초기화한다. 그래프 객체 생성 후 bfs()를 여러 번 호출할 수 있으므로, bfs()가 호출될 때마다 초기화를 수행했다.
- 5번 라인에서 인자로 전달된 start 정점부터 방문을 시작할 수 있도록 이를 큐에 추가한다.
- 큐는 방문이 확정되었으나 아직 출발점이 되지 못한 정점들을 순서대로 관리하는 역할을 한다.
- start가 큐에 추가되었으므로 visited를 True로 한다. 즉, bfs()에서 visited의 역할은 사실 큐에 추가되었는지의 여부를 나타내는 것이다. 큐에 추가된 적이 있는 정점은 다시 추가하면 안 되기 때문에 이를 기억하는 용도로 사용한다.
- 7번 라인에서 do_bfs()를 호출하여 너비우선탐색을 수행하고, 종료되면 모든 정점의 방문이 끝났으므로 한 행을 띄운다.

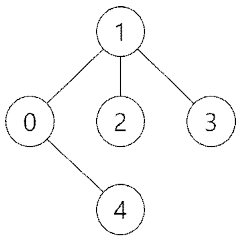
- do_bfs()의 구현은 다음과 같다.

```
1.          def do_bfs(self):
2.                  while self.bfsQ.get_size() > 0 :
3.                          vertex = self.bfsQ.dequeue()
4.                          print(vertex, ' ', end= ' ')
5.
6.                          for neighbor in self.graph[vertex] :
7.                                  if not self.visited[neighbor] :
8.                                          self.bfsQ.enqueue(neighbor)
9.                                          self.visited[neighbor] = True
```

- 2번 라인에서 큐에 출발점이 되지 못한 정점이 남아 있는 한 반복한다.
- 3번 라인에서 큐에서 정점을 하나 꺼낸다. 해당 정점은 방문하기로 예정되었고 아직 출발점이 되지 못한 정점들 중에서 가장 먼저 큐에 추가된 정점이다. 즉 가장 먼저 방문할 정점이다.
- 4번 라인에서 꺼낸 정점을 방문한다.
- 꺼낸 정점은 새로운 출발점이 되며, 해당 정점과 연결된 이웃 정점들을 향후 모두 방문해야 하므로 이들을 큐에 추가한다. (8번 라인). 단, 이 때 해당 정점이 기존에 큐에 추가된 적이 있다면 해당 정점은 다시 큐에 추가하지 않는다. (7번 라인).
- 큐에 추가한 정점은 visited를 True로 바꾼다. (8번 라인)
- 따라서 더 이상 큐에 존재하는 출발점이 되지 못한 정점이 없으면 do_bfs()는 종료된다.

라. 너비우선탐색 테스트 코드 및 실행 결과






```
1. if __name__ == '__main__':
2.          graph_adjlist = [[1, 3], [0, 4], [3], [0, 2, 4], [1, 3]]
3.          graph = Graph(graph_adjlist)
4.          graph.bfs(0)
5.
6.          graph_adjlist = [[1, 4], [0, 2, 3], [1], [1], [0]]
7.          graph = Graph(graph_adjlist)
8.          graph.bfs(0)
```

- 2번 라인에서 왼쪽 그래프를 인접리스트로 표현한다.
- 3번 라인에서 그래프 객체를 생성한다.
- 4번 라인에서 출발점을 0으로 하여 너비우선탐색을 수행한다.
- 0 -> 1 -> 3 -> 4 -> 2 순서로 방문을 수행해야 한다.

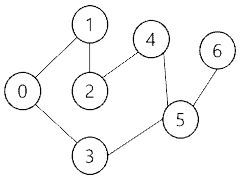
- 6번 라인에서 오른쪽 그래프를 인접리스트로 표현한다.
- 7번 라인에서 그래프 객체를 생성한다.
- 8번 라인에서 출발점을 0으로 하여 너비우선탐색을 수행한다.
- 0 -> 1 -> 4 -> 2 -> 3 순서로 방문을 수행해야 한다.

- 다음은 이에 대한 실행 결과이다.

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

연습문제

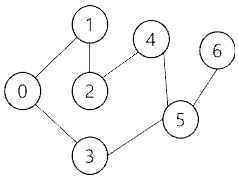
1. 다음의 그래프에서 정점 1의 차수를 구하시오..



정답 : 2

해설 : 무방향 그래프에서 차수는 정점과 연결된 간선의 수이다. 따라서 정답은 2이다.

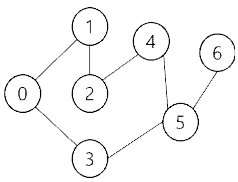
2. 다음의 그래프에서 0을 시작점으로 하여 깊이우선탐색을 수행할 때 방문하는 정점들을 순서대로 나타내시오. 단, 연결된 정점이 여러 개인 경우 정점의 숫자가 작은 것부터 방문하다고 가정한다.



정답 : 0 -> 1 -> 2 -> 4 -> 5 -> 3 -> 6

해설 : 0을 방문하고, 0에서 연결된 정점은 1, 3이므로 숫자가 작은 1을 다음으로 방문한다. 1에서 연결된 정점은 0, 2인데, 0은 방문했으므로, 2를 방문한다. 2에서 연결된 정점은 1, 4인데, 1은 방문했으므로, 4를 방문한다. 4에서 연결된 정점은 2, 5인데, 2는 방문했으므로, 5를 방문한다. 5에서 연결된 정점은 3, 4, 6이므로 3을 방문한다. 3에서 연결된 정점은 0, 5인데, 모두 방문했으므로, 직전 방문 정점인 5로 돌아간다. 마지막으로 5에서 연결된 6을 방문한다.

3. 다음의 그래프에서 0을 시작점으로 하여 너비우선탐색을 수행할 때 방문하는 정점들을 순서대로 나타내시오. 단, 연결된 정점이 여러 개인 경우 정점의 숫자가 작은 것부터 방문하다고 가정한다.



정답 : 0 -> 1 -> 3 -> 2 -> 5 -> 4 -> 6

해설 : 0을 방문하고, 0에서 연결된 정점은 1, 3이므로 1과 3을 차례로 방문한다. 다음으로는 1에서 연결된 정점인 2를 방문한다. 다음은 3에서 연결된 5를 방문한다. 다음은 2에서 연결된 4를 방문한다. 다음은 5에서 연결된 6을 방문한다.

정리하기

1. 그래프의 개념
2. 그래프를 표현하고 저장하는 방법
3. 그래프의 탐색 방법과 구현

참고자료

- 파이썬과 함께하는 자료구조의 이해, 양성봉, 2021, 생능출판