파이썬 프로그래밍

17. 알고리즘(2)

00

Contents

❖ 수업 목표

- 그래프 개요를 이해할 수 있다.
- 깊이 우선 탐색의 개념을 알고 구현할 수 있다.
- 너비 우선 탐색의 개념을 알고 구현할 수 있다.

❖ 세부 목표

1. 그래프(Graph)

❖ 그래프

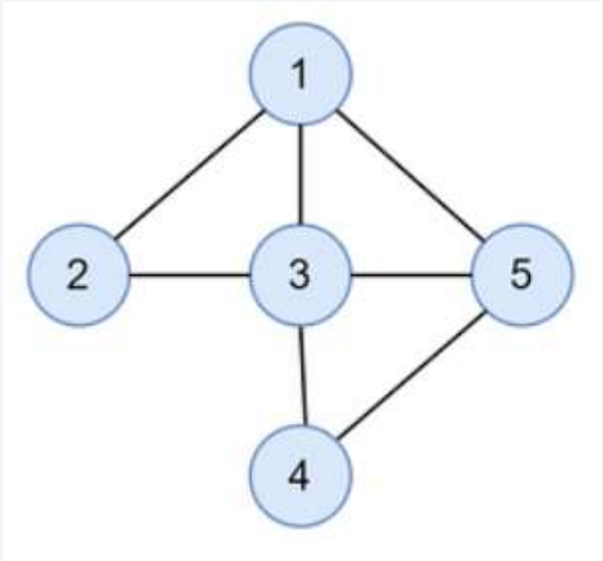
- 관계를 표현하는 구조
  - 예) 도시와 도시를 연결하는 도로, 사람 인맥 관계, 컴퓨터 네트워크 등
- 노드(Node)와 엣지(Edge)로 구성된 자료 구조
  - 노드(=정점, Vertex): 하나의 데이터 단위를 나타내는 객체
  - 엣지(간선): 두 노드간의 연결 관계를 나타내는 데이터 (연결선)
  - 두 노드 사이에 엣지가 있으면, [두 노드는 인접해 있다] 라고 표현함
- 그래프 관련 용어
  - 차수(degree) : 하나의 노드에 연결된 엣지의 수
  - 경로(path) : 한 노드에서 다른 노드까지 가는 길(노드들의 순서)

1. 그래프(Graph)

❖ 그래프

- 수학적 표현
  - G = (V, E)
    - V: 노드의 집합
    - E: 두 노드를 연결하는 간선의 집합

- 예시
  - 다음 그래프에서 V와 E의 집합은 다음과 같음

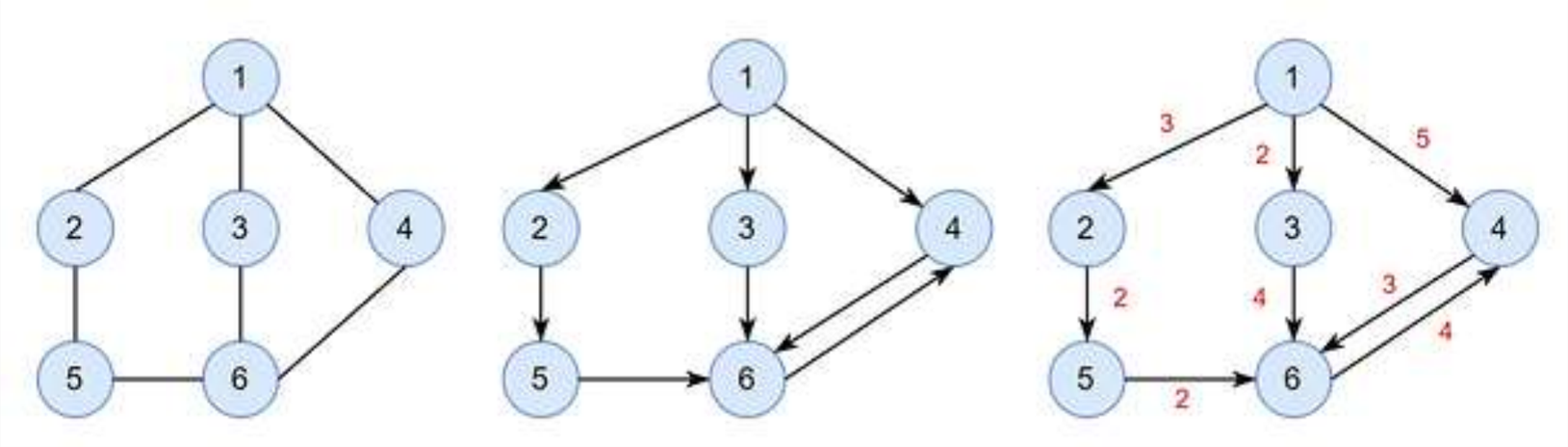


  - V = {1, 2, 3, 4, 5}
  - E = {(1, 2), (1, 3), (1, 5), (2, 3), (3, 4), (3, 5), (4, 5)}

1. 그래프(Graph)

❖ 그래프의 종류

- 무방향 그래프(undirected graph)
  - 두 노드를 연결하는 엣지에 방향이 없는 그래프
- 방향 그래프(directed graph)
  - 엣지에 방향이 있는 그래프
- 가중 그래프(weighted graph)
  - 노드를 연결하는 엣지에 가중치(weight)를 부여한 그래프
    - 가중치는 한 지점에서 다른 지점으로 이동하는 데 필요한 비용이나 시간 등

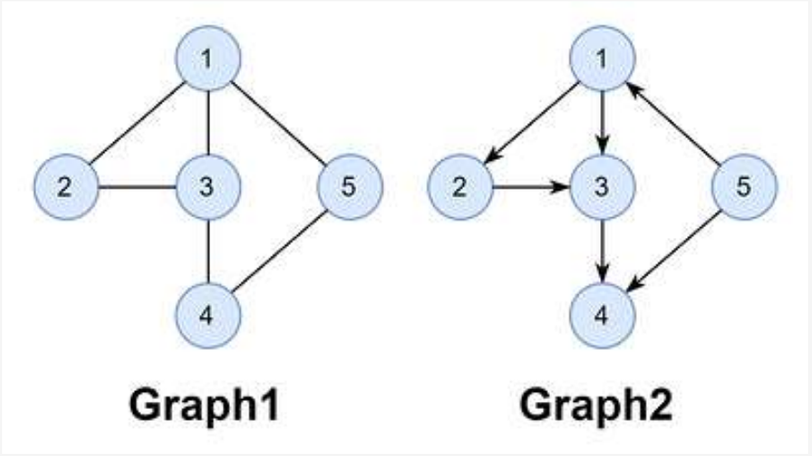


1. 그래프(Graph)

❖ 그래프의 종류

- 파이썬에서 어떤 관계를 표현할 때 적합한 문법 => 딕셔너리

- 예) 다음 그림의 그래프를 딕셔너리로 표현하기
  - 키(key): 노드
  - 값(value): 키와 연결된 노드 리스트

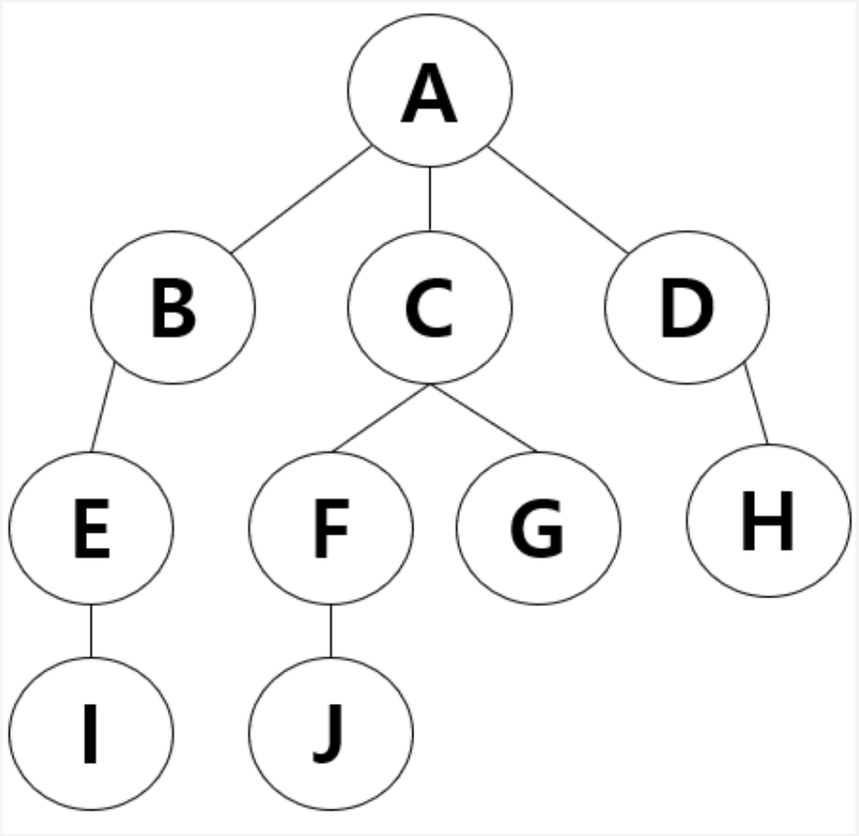


  - 무방향 그래프
    - graph1 = {1: [2, 3, 5], 2: [1, 3], 3: [1, 2, 4], 4: [3, 5], 5: [1, 4]}
  - 방향 그래프
    - graph2 = {1: [2, 3], 2: [3], 3: [4], 4: [], 5: [1, 4]}

1. 그래프(Graph)

❖ 그래프

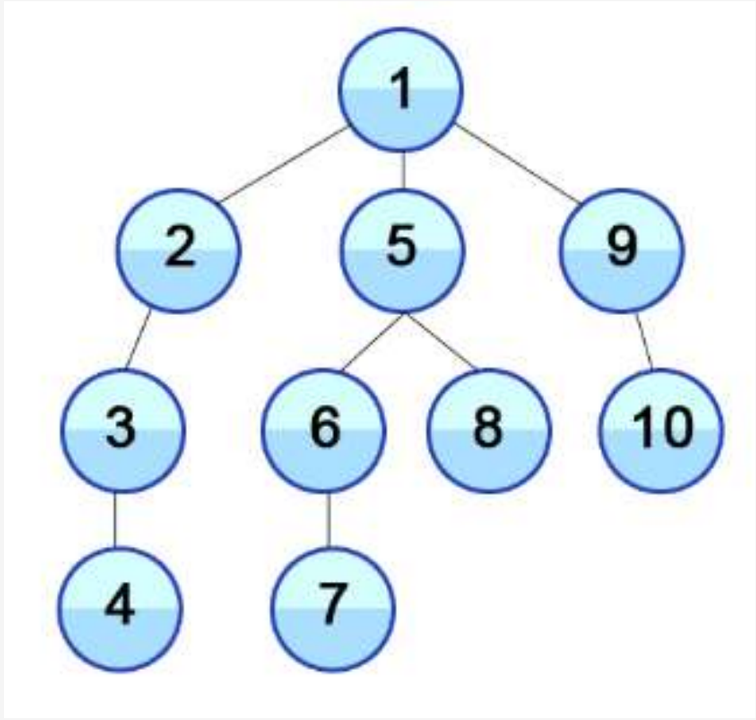
- 다음 그래프를 파이썬 딕셔너리로 표현



2. DFS(Depth First Search) 개요

❖ DFS(Depth First Search)

- 깊이 우선 탐색
  - 노드에 연결된 모든 엣지를 방문하지 않고 특정 1개의 엣지를 계속 따라 노드들을 방문하다가 더 이상 다른 노드를 방문할 수 없는 노드에 이르면, 다시 가장 가까운 분기점으로 돌아가 방문하지 않은 노드 방향으로 탐색을 이어가는 방법
  - 스택 이용 구현
  - 주로 모든 노드를 방문하고자 하는 경우에 사용



3. DFS(Depth First Search) 생성

❖ DFS(Depth First Search)

▪ 동작 방식

- 1. 방문(탐색)할 노드를 담을 스택(Stack) 자료형 생성
- 2. 각 노드가 방문한 노드인지를 구분할 수 있는 리스트 생성
- 3. 탐색 시작 노드(첫 번째 노드)를 스택에 삽입
- 4. 방문할 노드를 스택에서 하나씩 꺼내기
- 5. 스택에서 꺼낸 노드가 방문한 노드가 아니면,
  - 그 인접 노드를 스택에 삽입하고 방문 처리
- 6. 스택에 방문할 노드가 남지 않을 때까지 4~5의 과정을 반복

3. DFS(Depth First Search) 생성

❖ DFS 예제

- 1. 탐색할 노드를 담을 스택(Stack) 자료형 생성
  - stack = list()
- 2. 각 노드가 방문한 노드인지를 구분할 수 있는 리스트 생성
  - visited = list()
- 3. 탐색 시작 노드(첫 번째 노드)를 스택에 삽입
  - stack.append(start_node)

3. DFS(Depth First Search) 생성

❖ DFS 예제

- 4. 방문할 노드를 스택에서 하나씩 꺼내기

  - node = stack.pop()

- 5. 스택에서 꺼낸 노드가 방문한 노드가 아니면, 그 인접 노드를 스택에 삽입하고 방문 처리
  - reversed() 사용시, 리스트 순서를 반대로 설정

  - if node not in visited:
  - visited.append(node)
  - stack.extend(reversed(graph[node]))

3. DFS(Depth First Search)(3)

❖ DFS 예제

- 6. 스택에 방문할 노드가 남지 않을 때까지 3~5의 과정을 반복
  - while 문 조건에 리스트를 넣으면 내용이 없을 때 때까지 반복
  - while stack:

3. DFS(Depth First Search)(3)

❖ DFS 예제

- DFS 프로그램 구현

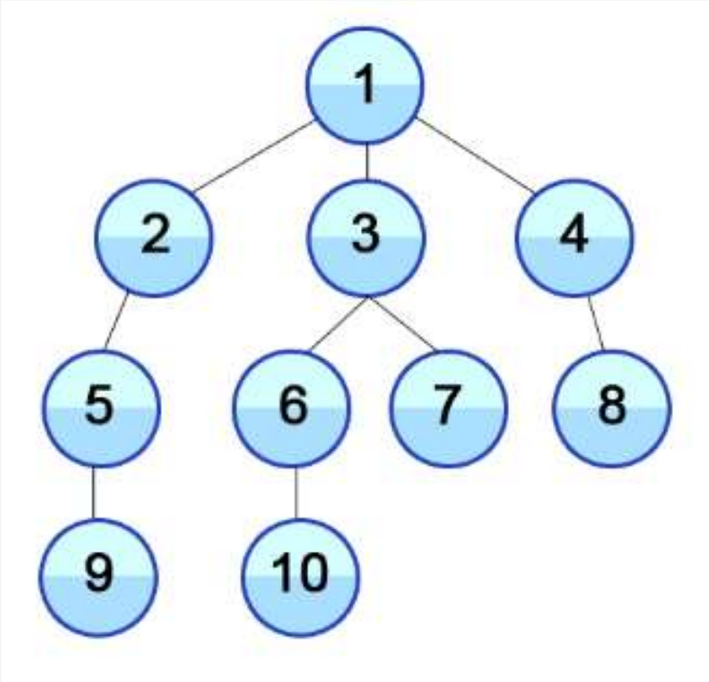
```
def my_dfs(graph, start_node):
    stack = list()
    visited = list()

    stack.append(start_node)

    while stack:
        node = stack.pop()

        if node not in visited:
            stack.extend(reversed(graph[node]))
            visited.append(node)

    print("dfs - ", visited)
    return visited

my_dfs(myGraph, 'A')
```

- 실행 결과
  - dfs - ['A', 'B', 'E', 'I', 'C', 'F', 'J', 'G', 'D', 'H']

4. BFS(Breadth First Search) 개요

❖ BFS(Breadth First Search)

- 너비 우선 탐색
  - 노드에 인접한 모든 엣지를 따라 노드에 방문하고, 다시 방문한 노드에 인접한 모든 엣지를 따라 노드에 방문하는 과정을 반복적으로 수행하며 탐색하는 방법
    - 큐 이용 구현
    - 주로 두 노드 사이의 최단 경로 혹은 임의의 경로를 찾고 싶을 때 사용



5. BFS(Breadth First Search) 생성

❖ BFS(Breadth First Search)

- 동작방식
- 1. 방문(탐색)할 노드를 담을 큐(Queue) 자료형 생성
- 2. 각 노드가 방문한 노드인지를 구분할 수 있는 리스트 생성
- 3. 탐색 시작 노드(첫 번째 노드)를 큐에 삽입
- 4. 방문할 노드를 큐에서 하나씩 꺼내기 (삽입된 순서대로)
- 5. 큐에서 꺼낸 노드가 방문한 노드가 아니면, 그 인접 노드를 큐에 삽입하고 방문 처리
- 6. 큐에 방문할 노드가 남지 않을 때까지 4~5의 과정을 계속 반복

5. BFS(Breadth First Search) 생성

❖ BFS 예제

- 1. 방문(탐색)할 노드를 담을 큐(Queue) 자료형 생성
  - queue = list()
- 2. 각 노드가 방문한 노드인지를 구분할 수 있는 리스트 생성
  - visited = list()
- 3. 탐색 시작 노드(첫 번째 노드)를 큐에 삽입
  - queue.append(start_node)

5. BFS(Breadth First Search) 생성

❖ BFS 예제

- 4. 방문할 노드를 큐에서 하나씩 꺼내기 (삽입된 순서대로)

  - node = queue.pop(0)

- 5. 큐에서 꺼낸 노드가 방문한 노드가 아니면, 그 인접 노드를 큐에 삽입하고 방문 처리

  - if node not in visited:
  - queue.extend(graph[node])
  - visited.append(node)

5. BFS(Breadth First Search) 생성

❖ BFS 예제

- 6. 큐에 방문할 노드가 남지 않을 때까지 4~5의 과정을 계속 반복
  - while 문 조건에 리스트를 넣으면 내용이 없을 때 때까지 반복
  - while queue:

5. BFS(Breadth First Search) 생성

❖ BFS 예제

- BFS 프로그램 구현

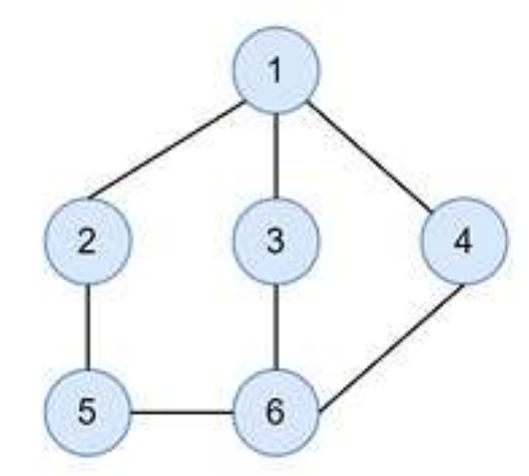
```
def my_bfs(graph, start_node):
    queue = list()
    visited = list()

    queue.append(start_node)

    while queue:
        node = queue.pop(0)

        if node not in visited:
            queue.extend(graph[node])
            visited.append(node)

    print("bfs - ", visited)
    return visited

my_bfs(myGraph, 'A')
```

- 실행 결과
  - bfs - ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']

00

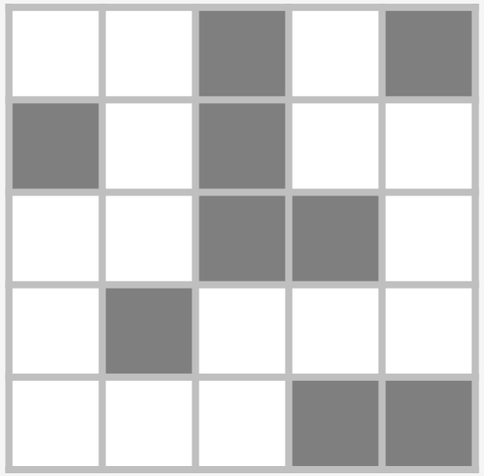
6. 연습 문제

❖ 다음 그래프를 만들고 깊이 우선 탐색(DFS)과 너비 우선 탐색(BFS)을 위한 프로그램을 각각 작성하시오.



6. 연습 문제

❖ 다음 그림과 같이 지도에 길이 표시가 되어 있다. 길은 밝은 색으로 벽은 어두운 색으로 표현되었다. 이 그림을 단순화하여 길은 0으로 벽은 1로 표현하는 이차원 배열의 리스트를 작성하시오.



Notice

❖ 과제

- 1. 임의의 그래프 코드 작성하기
- 2. 주어진 그래프의 깊이 우선 탐색 코드 구현하기
- 3. 주어진 그래프의 너비 우선 탐색 코드 구현하기

❖ 다음 수업 내용

- 그래픽스 지원 API
  - 터틀 그래픽스, 사각형 그리기, 다각형 그리기, 복잡한 도형 그리기