201201871 서현택

1.목표

최단경로 알고리즘 구현(BFS, DFS)

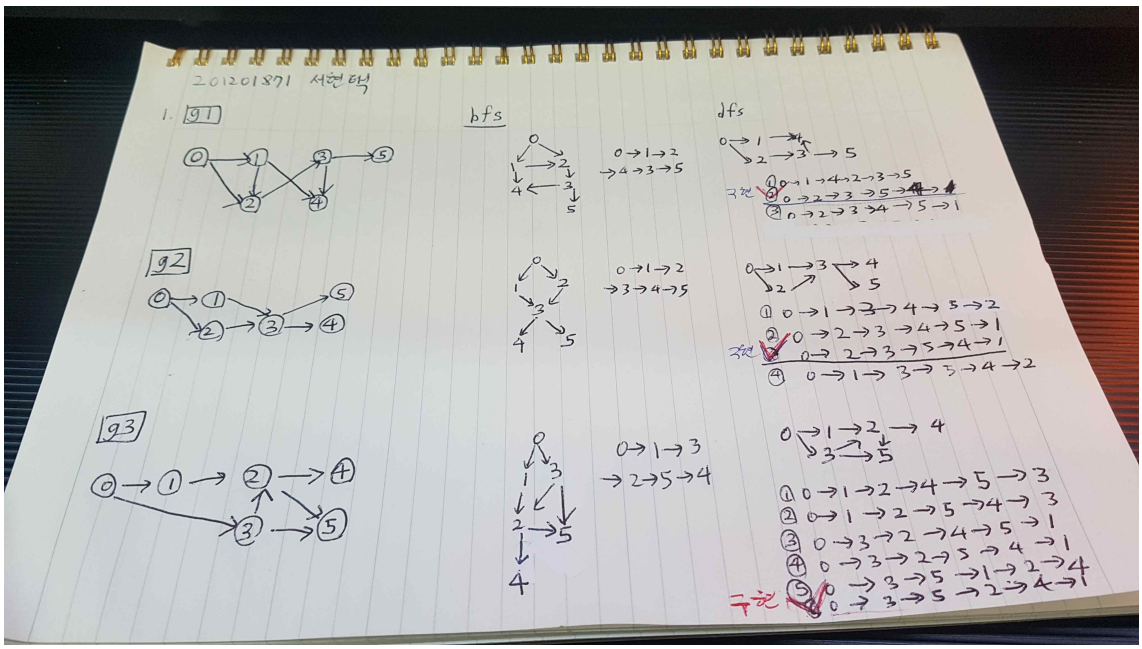
2.과제를 해결하는 방법

(1) 실습 내용 기반으로 BFS, DFS 알고리즘 구현

(2) 그림을 그려보면서 알고리즘 이해

3.과제를 해결한 방법

(1) 그림을 통한 이해



(2) BFS

list형 변수 queue와 set형 변수 visited를 생성한 뒤, start_node를 추가한다. 그 후, start_node를 출력한다. queue의 길이가 0보다 클 때 동안, visiting에 queue의 첫 번째 자료를 pop시킨 것을 저장한다. neighbor가 graph의 리스트에 있는지 확인하고, vistited에 neighbor가 없다면 visted에 neighbor를 추가하고, neighbor를 출력한 뒤, queue에 neighbor를 저장한다.

(3) DFS

list형 변수 stack과 set형 변수 visited를 생성한 뒤, stack에 start_node를 삽입하한다. stack의 길이가 0보다 클 때 동안, visiting에 stack을 pop한 값을 저장하고, visited에 visiting이 없다면 visited에 visiting을 추가한다. 그 후 visiting을 출력한 뒤, graph의 vert_list에 포함되어있는 값을 stack에 저장한다.

(4) Graph 클래스

1) init

Graph 클래스를 초기화한다.

2) add_vertex

Graph 클래스에 key를 통해 vertex를 추가한다.

3) get_vertex

Graph 클래스에 저장된 vertex의 list에서 해당되는 vertex를 불러온다.

4) contains

vertex에 해당 값이 저장되어 있나 확인한다.

5) add_edge

vert_list[f]에서 vert_list[t]로의 edge를 생성한다.

6) get_vertices

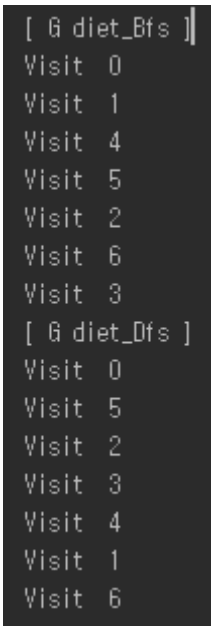
vert_list의 keys를 불러온다.

7) iter

vert_list.values의 iter 함수 값을 반환한다.

4.결과화면

(1) 제공된 코드



(2) G1

```
[ G1 diet_Bfs ]
Visit  0
Visit  1
Visit  2
Visit  4
Visit  3
Visit  5
[ G1 diet_Dfs ]
Visit  0
Visit  2
Visit  3
Visit  5
Visit  4
Visit  1
```

(3) G2

```
[ G2 diet_Bfs ]
Visit  0
Visit  1
Visit  2
Visit  3
Visit  4
Visit  5
[ G2 diet_Dfs ]
Visit  0
Visit  2
Visit  3
Visit  5
Visit  4
Visit  1
```

(4) G3

```
[ G3 diet_Bfs ]
Visit  0
Visit  1
Visit  3
Visit  2
Visit  5
Visit  4
[ G3 diet_Dfs ]
Visit  0
Visit  3
Visit  5
Visit  2
Visit  4
Visit  1
```

1 개요

그래프(Graph)는 컴퓨터공학에서 노드(Node)간의 연결을 표현할 때 자주 사용되는 자료형이다. 컴퓨터 네트워크뿐 아니라 OS, 보안 등 다양한 분야에서 활용되기 때문에 그래프 이론이라는 과정이 있을 정도로 중요하게 다루어지고 있다. 따라서 문제 해결을 위한 다양한 알고리즘을 배우는 이 과목에서 그래프에 대한 기초와 응용을 배우는 것은 꼭 거쳐야할 내용이다. 이번 시간에는 지난 트리를 배웠을 때 처럼 노드 하나하나를 탐색하는 방법에 대하여 배운다. 노드 탐색을 구현하며 그래프에 대해 익숙해지고 활용하기 위한 기초를 다진다.

2 그래프(Graph)

그래프는 꼭지점(Vertex or Node)과 에지(Edge)로 구성된다. 이 때 에지는 각 노드로 가는 길이 되며 방향(출발, 도착)과 거리(무게)가 있다.

2.1 dict를 활용한 그래프

본래 그래프는 노드와 에지 클래스를 이용하여 구성해야하지만 `dict` 타입 변수를 이용하면 `Class`를 구현하지 않고 표현할 수 있다. Key-Value 형식을 이용하여 그래프의 '노드'와 '이웃'을 표현한다. 예를 들어 Figure 1과 같은 그래프는 `dict`와 `set`을 이용하여 표현할 수 있다.

Listing 1: 3개 노드 그래프(Graph1) 표현 방법

```
graph1 = {0: set([1, 2]),
          1: set([]),
          2: set([])}
```

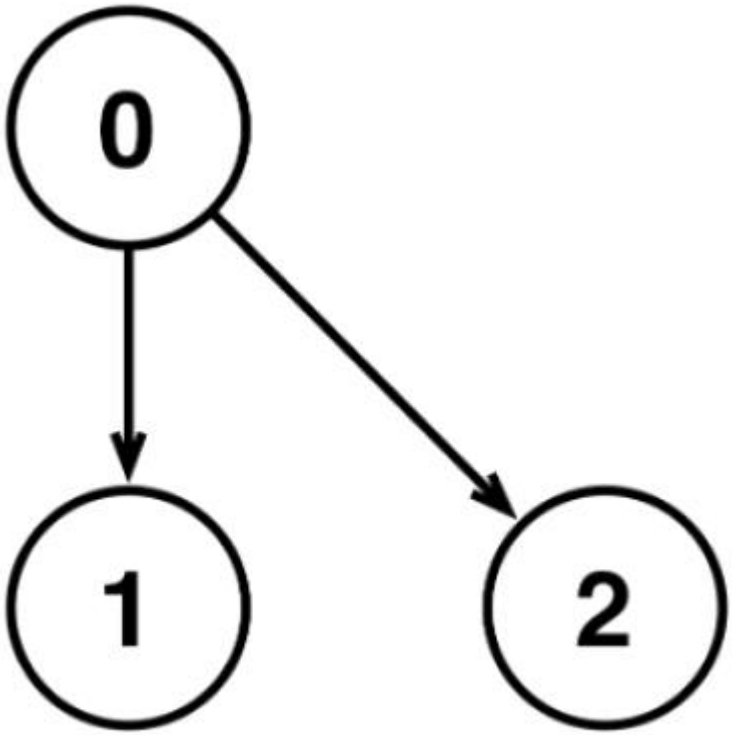


Figure 1: 3개 노드 그래프

그래프 표현 예제에서 확인할 수 있듯, 그래프에서 꼭 표현되어야 하는 정보는 '노드'와 그 '이웃'에 대한 정보다. 우리는 이번 실습에서 그래프를 표현하기 위해 `dict` 타입과 `Class`를 활용하지만 특정 상황에서는 행렬(matrix)를 사용하기도 한다. 따라서 특정 상황에서 그래프를 표현하는 방법보다는 각 변수(혹은 객체)가 어떤 정보를 담고 있어야하는지 이해해야한다.

2.2 Class를 활용한 그래프

JAVA와 같은 대표적인 언어의 그래프 예제에서는 `Class Vertex`와 `Class Edge`를 이용하여 그래프를 표현한다. Python 3와 같은 좀 더 편리한 언어에서는 `dict` 타입 변수를 쉽게 사용할 수 있으므로 '이웃'정보를 `Class Graph`에 포함하는 경우가 많다. 아래 두 코드 예제는 '그래프'와 '노드'를 표현하는 방법을 보인다.

3 그래프 탐색

3.1 너비 우선 탐색

너비 우선 탐색(BFS)은 탐색하는 노드의 이웃을 모두다 탐색한 후 다음으로 넘어가는 방법이다. 주로 Queue를 이용하여 구현하며, 주변을 모두 연산하며 진행하는 특징이 있다. `dict` 타입 변수를 이용하여 BFS를 구현하는 코드는 아래와 같다.

Listing 5: BFS

```
1  def dict_bfs(graph, start_node):
2      queue = list()
3      queue.append(start_node)
4      visited = set()
5      visited.add(start_node)
6      print('Visit', start_node)
7
8      while len(queue) > 0:
9          visiting = queue.pop(0)
10         for neighbor in graph[visiting]:
11             if neighbor not in visited:
12                 visited.add(neighbor)
13                 print('Visit', neighbor)
14                 queue.append(neighbor)
```

3.2 깊이 우선 탐색

깊이 우선 탐색(DFS)은 탐색하는 노드의 이웃중 하나만 탐색한 후 다음으로 넘어가는 방법이다. 주로 Stack를 이용하여 구현하며, 주변을 모두 연산하지 않고 '링크'를 살펴보는 특징이 있다. dict 타입 변수를 이용하여 DFS를 구현하는 코드는 아래와 같다.

Listing 6: DFS

```
1  def dict_dfs(graph, start_node):
2      stack = list()
3      visited = set()
4      stack.append(start_node)
5
6      while len(stack) > 0:
7          visiting = stack.pop()
8          if visiting not in visited:
9              visited.add(visiting)
10             print('Visit', visiting)
11             for neighbor in graph[visiting]:
12                 stack.append(neighbor)
```