2018년도 가을학기 SCSC 알고리즘 실습 10주차

문현수, munhyunsu@cs-cnu.org

Thursday November 15, 2018

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

```
1 graph1 = {0: set([1, 2]),
2           1: set([]),
3           2: set([])}
```

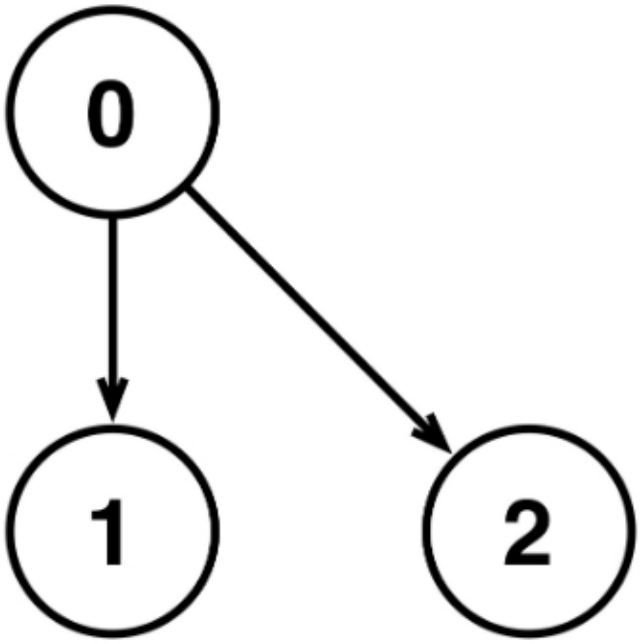

Figure 1: 3개 노드 그래프

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.

Listing 2: Class Graph

```
1  class Graph(object):
2      def __init__(self):
3          self.vert_list = {}
4          self.num_vertices = 0
5
6      def add_vertex(self, key):
```

```
7           self.num_vertices = self.num_vertices + 1
8           new_vertex = Vertex(key)
9           self.vert_list[key] = new_vertex
10          return new_vertex
11
12      def get_vertex(self, n):
13          if n in self.vert_list:
14              return self.vert_list[n]
15          else:
16              return None
17
18      def __contains__(self,n):
19          return n in self.vert_list
20
21      def add_edge(self, f, t, cost=0):
22          if f not in self.vert_list:
23              nv = self.add_vertex(f)
24          if t not in self.vert_list:
25              nv = self.add_vertex(t)
26          self.vert_list[f].add_neighbor(self.vert_list[t],
                cost)
27
28      def get_vertices(self):
29          return self.vert_list.keys()
30
31      def __iter__(self):
32          return iter(self.vert_list.values())
```

Listing 3: Class Vertex

```
1   class Vertex(object):
2       def __init__(self, key):
3           self.id = key
4           self.connected_to = {}
5
6       def add_neighbor(self, nbr, weight=0):
7           self.connected_to[nbr] = weight
8
9       def __str__(self):
10          return str(self.id) + ' connected_to: ' + str([x.
                id for x in self.connected_to])
```

```
11
12      def get_connections(self):
13          return self.connected_to.keys()
14
15      def get_id(self):
16          return self.id
17
18      def get_weight(self, nbr):
19          return self.connected_to[nbr]
```

위의 두 클래스를 이용하여 그래프를 만드는 예제를 살펴보자. 아래 코드는 Vertex가 6개, Edge가 9개인 그래프 예제다. 직접 예제를 입력, 출력해본 후 이해가 되었는지 확인하기 위하여 Edge의 '무게'를 출력해보자. 그래프를 표현하는 Class 내부의 멤버 변수, 멤버 함수를 이용하는 방법을 익혀야한다.

Listing 4: Class를 활용한 그래프 예제

```
1  def main():
2      g = Graph()
3      for i in range(6):
4          g.add_vertex(i)
5      print(g.vert_list)
6      g.add_edge(0, 1, 5)
7      g.add_edge(0, 5, 2)
8      g.add_edge(1, 2, 4)
9      g.add_edge(2, 3, 9)
10     g.add_edge(3, 4, 7)
11     g.add_edge(3, 5, 3)
12     g.add_edge(4, 0, 1)
13     g.add_edge(5, 4, 8)
14     g.add_edge(5, 2, 1)
15     for v in g:
16         for w in v.get_connections():
17             print('({0}, {1})'.format(v.get_id(), w.
                   get_id()))
```

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)
```

4 과제

실습에서 `dict` 타입 변수를 활용한 그래프 탐색을 연습해보았다. 과제에서는 Class를 이용한 그래프를 탐색하는 함수를 작성한다. 우선 Figure 2를 그래프로 만들어 보고 탐색해보자. 그 후 자신이 임의의 그래프 3개를 그려 탐색해보고 그 결과가 정확한지 설명하도록 한다.

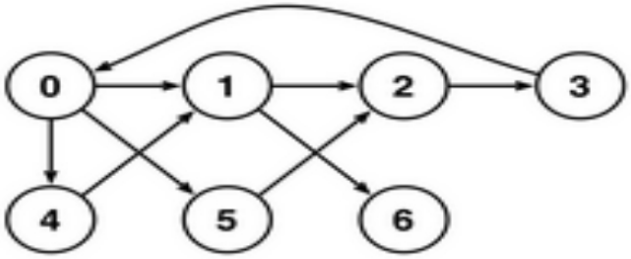

Figure 2: 7개 노드 그래프

4.1 과제 목표

- Class기반 Graph에 대한 Breadth First Search, Depth First Search 구현
- 각 방법에 대하여 임의의 그래프 3개 이상 실험(보고서에 설명)
- – 임의의 그래프 그려넣기(손으로 or 파워포인트 그림 등등)
- – 임의의 그래프에서 탐색 순서 확인(손으로 설명)
- – 임의의 그래프에서 동작된 결과 확인 및 설명

4.2 제출 관련

- 마감 날짜: 2018. 11. 21. 23:59:59
- 딜레이: 1일당 10% 감점(처음 2일까지는 -2)
- 제출 방법: 과목 사이버캠퍼스

- 제출 형식: 과제 리포트 PDF(HWP, DOC 받지 않음!), 소스코드(구현한 .py 만 추가할 것)를 압축한 .zip 파일
- 리포트에 포함해야하는 내용: 목표, 목표를 위해 알아야하는 것, 해결 방법, 결과, (선택)느낀점 or전달할 말
- 제출 파일 제목: AL_201550320_문현수_10.zip(파일명 준수!)

4.3 조교 연락처

- 문현수
- munhyunsu@cs-cnu.org
- 공학5호관 633호 데이터네트워크연구실
- 이메일, 연구실 방문