2018년도 가을학기 SCSC 알고리즘
실습 11주차

문현수, munhyunsu@cs-cnu.org

Thursday November 22, 2018

1 개요

컴퓨터공학에서 노드(Node)간의 연결을 표현할 때 자주 사용되는 그래프(Graph)는 컴퓨터 네트워크뿐 아니라 OS, 보안 등 다양한 분야에서 활용되고 있다. 특히 그래프는 에지(Edge)의 무게(Weight)를 고려하여 표현할 수 있기 때문에 최소 비용을 계산하거나 최소 거리 경로를 찾아낼 때 활용된다. 이번 시간에는 그래프를 활용하여 최단 거리를 찾고 최소 비용으로 모든 노드를 연결하는 문제를 해결해 본다.

2 그래프(Graph)

그래프는 꼭지점(Vertex or Node)과 에지(Edge)로 구성된다. 지난 시간에 그래프 탐색을 위하여 `dict` 타입으로 표현된 그래프와 `class` 타입으로 표현된 그래프를 만들어보았다. 이번 시간에도 `dict` 타입 그래프를 통해 알고리즘을 연습하고 `class` 타입으로 표현된 그래프로 과제를 진행한다.

2.1 가중치있는 방향 그래프

그래프의 노드를 잇는 에지는 가중치(무게)뿐 아니라 방향을 가질 수 있다. Figure 1처럼 가중치와 방향을 모두 표현하는 방법을 익혀야한다. 방향은 출발지에서 도착지로 나타내며 반대로는 이동할 수 없음을 나타낸다. 일방통행 도로라고 생각할 수 있다.

Listing 1: 가중치있는 방향 그래프 표현

```
1 WDGRAPH02 = {
2     'a': {'b': 8, 'd': 9, 'f': 11},
3     'b': {'c': 10},
4     'c': {'e': 2},
```

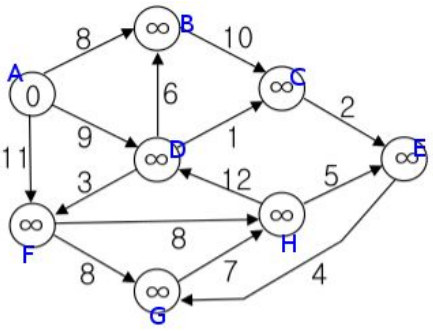

Figure 1: 가중치있는 방향 그래프

```
5       'd': {'c': 1, 'f': 3},
6       'e': {'g': 4},
7       'f': {'g': 8, 'h': 8},
8       'g': {'h': 7},
9       'h': {'d': 12, 'e': 5}
10      }
```

위의 소스코드는 `dict` 타입 변수를 활용하여 가중치 있는 방향 그래프를 표현한 것이다. 읽어보면 **a**에서 **b**로 가는 길은 **8**의 가중치로 이어져있으나 반대로 돌아오는 길은 없다. 이런 그래프를 가중치있는 방향 그래프라고 한다.

2.2 가중치있는 무방향 그래프

보통 길이 '이어져있다.'라고 표현하면 양방향에서 오갈 수 있는 상태를 생각한다. 그래프에서도 모든 에지가 양방향으로 표현되어있을 때 '무방향 그래프'라고 이야기한다. 가중치가 표현된 상태로 방향이 없는 그래프는 컴퓨터간 '네트워크' 연결 상태를 표현할 때 주로 사용한다. Figure 2처럼 서로 연결된 에지에 방향이 없을 때 '가중치있는 무방향 그래프'라고 말한다.

Listing 2: 가중치있는 무방향 그래프 표현

```
1  WUGRAPH01 = {
2       'a': {'b': 7, 'd': 5},
3       'b': {'a': 7, 'c': 8, 'd': 9, 'e': 7},
4       'c': {'b': 8, 'e': 5},
5       'd': {'a': 5, 'b': 9, 'e': 15, 'f': 6},
6       'e': {'b': 7, 'c': 5, 'd': 15, 'f': 8, 'g': 9},
7       'f': {'d': 6, 'e': 8, 'g': 11},
8       'g': {'e': 9, 'f': 11}
9  }
```

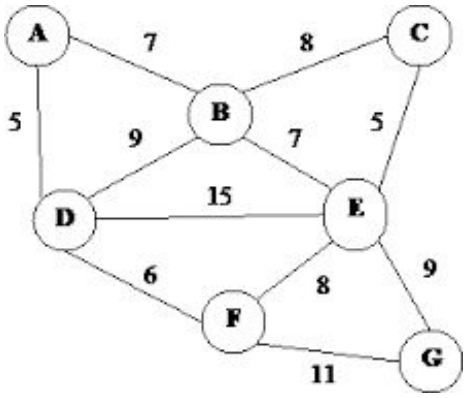

Figure 2: 가중치있는 무방향 그래프

위의 소스코드는 가중치있는 무방향 그래프를 표현한 것이다. 방향 그래프와는 다르게 **a**에서 **b**로 **7**의 가중치를 가진 에지가 있으면 그 반대의 에지도 표현되어 있다. 이러한 그래프를 가중치있는 무방향 그래프라고 부른다.

3 networkx

Python 3에는 Graph를 표현하고 시각화를 도와주는 도구가 있다. `networkx`라는 도구는 지난 시간에 정렬 성능 비교를 위하여 사용했었던 `matplotlib`를 이용하여 Graph를 시각화한다. 예를 들어 fig. 3는 fig. 4처럼 시각화할 수 있다.

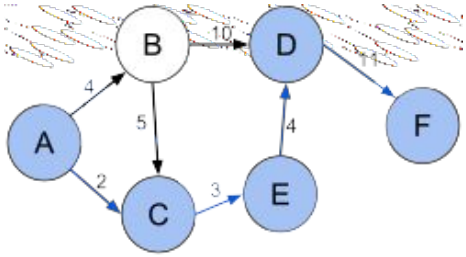

Figure 3: 최단거리 그래프

Figure 4처럼 시각화하기 위하여 `networkx` 라이브러리를 설치해야한다. `pip`를 이용하여 설치한 후 아래 소스코드를 입력하면 같은 그래프가 출력될 것이다. 매 실행시마다 노드의 위치가 바뀌기 때문에 구조는 다를 수 있다. 만약 '무방향 그래프'를 표현해야한다면 `nx.Graph()`로 생성하면 된다.

Listing 3: networkx를 이용한 그래프 시각화 예제

```
1 def main():
2     graph = nx.DiGraph()
```

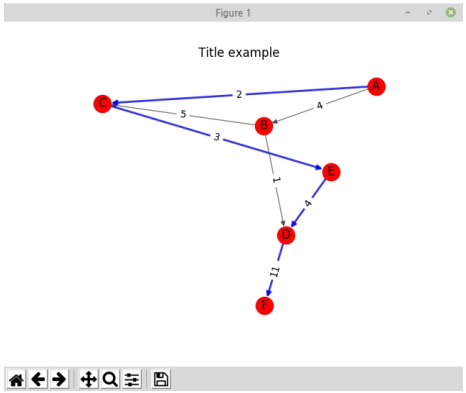

Figure 4: 최단거리 그래프의 networkx를 활용한 시각화

```
3
4       data = {('A', 'B'): 4,
5               ('A', 'C'): 2,
6               ('B', 'C'): 5,
7               ('B', 'D'): 1,
8               ('C', 'E'): 3,
9               ('E', 'D'): 4,
10              ('D', 'F'): 11}
11
12      for key in data.keys():
13          u, v = key
14          weight = data[key]
15          graph.add_edge(u, v, weight=weight)
16
17      path = [('A', 'C'),
18              ('C', 'E'),
19              ('E', 'D'),
20              ('D', 'F')]
21      non_path = list(set(graph.edges) - set(path))
22
23      position = nx.spring_layout(graph)
24      # position = nx.circular_layout(graph)
25      nx.draw_networkx_nodes(graph, position)
26      nx.draw_networkx_labels(graph, position)
27      nx.draw_networkx_edges(graph, position, edgelist=
            non_path, alpha=0.5)
28      nx.draw_networkx_edges(graph, position, edgelist=path
            , alpha=0.8, edge_color='blue', width=2)
```

```
29      nx.draw_networkx_edge_labels(graph, position,
            edge_labels=data)
30
31      plt.title('Title example')
32      plt.axis('off')
33      plt.show()
```

4 최단 거리 알고리즘: Dijkstra

다익스트라(Dijkstra) 알고리즘은 그래프 노드 사이의 최단 거리를 찾는 알고리즘 중 가장 유명한 알고리즘이다. 또한 '알고리즘'이라는 교과목을 이야기할 때 가장 대표되는 알고리즘이기도 하다. 다익스트라 알고리즘은 아래 수도코드로 구현이 된다.

Listing 4: 다익스트라 알고리즘 수도코드

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:                 //
           Initialization
6          dist[v] <- INFINITY                      // Unknown
               distance from source to v
7          prev[v] <- UNDEFINED                     // Previous
               node in optimal path from source
8          add v to Q                               // All nodes
               initially in Q (unvisited nodes)
9
10     dist[source] <- 0                            // Distance
           from source to source
11
12     while Q is not empty:
13         u <- vertex in Q with min dist[u]        // Node with
               the least distance will be selected first
14         remove u from Q
15
16         for each neighbor v of u:                // where v is
               still in Q.
17             alt <- dist[u] + length(u, v)
```

```
18              if alt < dist[v]:                // A shorter
                    path to v has been found
19                  dist[v] <- alt
20                  prev[v] <- u
21
22      return dist[], prev[]
```

위의 알고리즘을 dict타입 그래프 대상으로 구현하면 아래와 같이 구현된다.

Listing 5: dict타입 그래프를 위한 다익스트라 알고리즘 스켈레톤 코드

```
1  import sys
2
3  # FILL_HERE: contents of WDGRAPH02
4  WDGRAPH02 = {
5      }
6
7
8  def dijkstra(wdgraph, start_node):
9      qnode = set()
10     dist = dict()
11     prev = dict()
12
13     # FILL_HERE: initialize variables(multi lines)
14
15     # FILL_HERE: set distance 0 from start node to start
           node
16
17     while len(qnode) != 0:
18         udist = sys.maxsize
19         unode = None
20         for node in qnode:
21             if dist[node] < udist:
22                 udist = dist[node]
23                 unode = node
24         # FILL_HERE: remove node from queue
25
26         # (Most important!) FILL_HERE: compute new
               distance(multi lines)
27
28     return dist, prev
29
```

```
30
31 def main():
32     print('Start a')
33     print(dijkstra(WDGRAPH02, 'a'))
34
35
36 if __name__ == '__main__':
37     main()
```

5 최소 비용 신장 트리 알고리즘: Prim

모든 노드를 최소한의 비용(에지)로 연결하려면 어떻게 해야할까? 최소 비용 신장 트리 알고리즘(MST)어떤 그래프에 포함된 노드를 모두 연결하기 위한 최소 비용 에지를 찾아내는 알고리즘이다. Prim's algorithm은 MST의 대표 알고리즘으로 컴퓨터 네트워크 노드 연결에서 굉장히 자주 사용된다.

Listing 6: 프림 알고리즘 수도코드

```
1 Initialize a tree with a single vertex, chosen
      arbitrarily from the graph.
2 Grow the tree by one edge: of the edges that connect the
      tree to vertices not yet in the tree, find the minimum
      -weight edge, and transfer it to the tree.
3 Repeat step 2 (until all vertices are in the tree).
```

프림 알고리즘은 굉장히 많은 방법으로 구현될 수 있기 때문에 수도 코드도 문장 형태로 전파되고 있다. 위의 수도 코드를 `dict` 타입 그래프 대상으로 구현하면 아래의 코드가 된다.

Listing 7: dict타입 그래프를 위한 프림 알고리즘 스켈레톤 코드

```
1 # FILL_HERE: contents of WUGRAPH01
2 WUGRAPH01 = {
3 }
4
5
6 def search_min(wugraph, an, rn):
7     src_node = None
8     dst_node = None
9     dist = float('inf')
10    for src in an:
```

```
11          for dst in wugraph[src]:
12              if dst in rn:
13                  if (wugraph[src])[dst] < dist:
14                      dist = (wugraph[src])[dst]
15                      src_node = src
16                      dst_node = dst
17
18      return src_node, dst_node, dist
19
20
21  def prim(wugraph):
22      mst = dict()
23      added_node = set()
24      remain_node = set()
25
26      for node in wugraph.keys():
27          mst[node] = dict()
28
29      # FILL_HERE: initialize remain_node
30
31      import random
32      start_node = random.choice(list(wugraph.keys()))
33
34      # FILL_HERE: handle start_node - add to added_node,
            remove from remain_node
35
36      # (Most important!) FILL_HERE: compute new distance,
            and handle sets - HINT: while loop
37
38      return mst
39
40
41  def main():
42      print(prim(WUGRAPH01))
43
44
45  if __name__ == '__main__':
46      main()
```

모든 노드를 통해서 갈 수 있는 모든 에지중에 최소 비용 에지를 선택하는 것을 반복한다.

6 (과제) networkx: Dijkstra, Prim

이번 실습의 과제는 networkx(class기반 그래프)를 활용하여 Dijkstra와 Prim 알고리즘 구현하고 시각화하는 것이다. 최단 거리와 최소 비용 에지를 표현할 때 에지의 '색'과 '두께'가 다르다면 시각적으로 훨씬 편할 것이다. networkx예제를 이용하여 fig. 1를 대상으로 Dijkstra, fig. 2를 대상으로 Prim을 구현하고 시각화 해보자. 그리고 각각의 알고리즘에 대하여 임의의 그래프를 작성하고 시각화하자. 완료하였다면 fig. 5, fig 6과 같은 그림이 나온다.

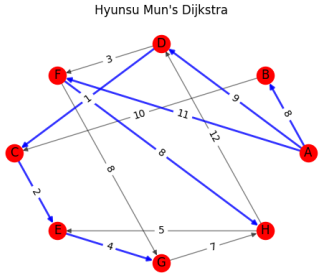

Figure 5: Dijkstra 알고리즘 결과

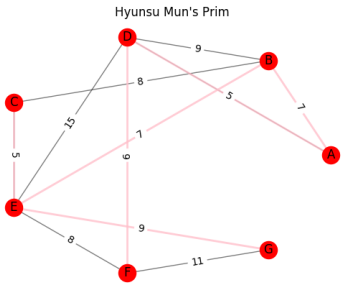

Figure 6: Prim 알고리즘 결과

6.1 과제 목표

- networkx기반 Dijkstra 알고리즘 구현 및 시각화
- networkx기반 Prim 알고리즘 구현 및 시각화
- 제공된 그래프 1개 + 임의의 그래프를 작성하여 총 2개 이상의 그래프 실험

6.2 제출 관련

- 마감 날짜: 2018. 11. 28. 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!)

6.3 조교 연락처

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