[18. 11. 20. 추가]
그동안 바빠서 업로드하지 못했던 과제 설명 pdf 업로드 하였습니다.

알고리즘 실습

181115 - BFS/DFS

오늘의 목표

- 최단 경로(Shortest Path) 알고리즘
  - 다익스트라(Dijkstra's algorithm)

오늘의 목표

- ~~최단 경로(Shortest Path) 알고리즘~~
  - ~~다익스트라(Dijkstra's algorithm)~~
- 그래프(Graph) 기초
  - Breadth First Search
  - Depth First Search

Feedback

소스코드 검사 결과

	A	B	C
1	week9_merged/201600827.py	week9_merged/201601534.py:	21%
2	week9_merged/201201874.py	week9_merged/201502273.py:	19%
3	week9_merged/201201871.py	week9_merged/201600827.py:	18%
4	week9_merged/201301782.py	week9_merged/201301949.py:	18%
5	week9_merged/201201871.py	week9_merged/201601534.py:	17%
6	week9_merged/201301782.py	week9_merged/201400588.py:	17%
7	week9_merged/201201871.py	week9_merged/201400114.py:	16%
8	week9_merged/201201874.py	week9_merged/201601278.py:	16%
9	week9_merged/201400978.py	week9_merged/201401577.py:	16%
10	week9_merged/201401577.py	week9_merged/201600827.py:	15%
11	week9_merged/201200077.py	week9_merged/201201874.py:	14%
12	week9_merged/201200077.py	week9_merged/201400978.py:	14%
13	week9_merged/201301939.py	week9_merged/201502273.py:	14%
14	week9_merged/201400592.py	week9_merged/201401062.py:	14%
15	week9_merged/201401577.py	week9_merged/201601534.py:	14%
16	week9_merged/201200077.py	week9_merged/201400592.py:	13%

지난 시간: 이진 탐색 트리

- 제출률: 75.61%(31/41)
- 질문: 이메일 50건

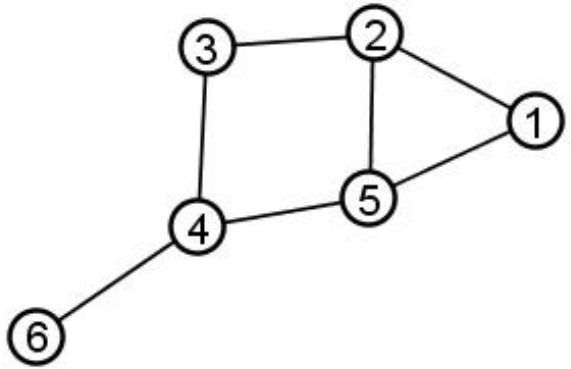
지난 시간: 이진 탐색 트리

- 제출률: 75.61%(31/41)
  - 완벽 구현: 15명
  - 사소한 오류: 6명
  - 삽입만 구현 or 우선 제출 10명
  - 따라서… 51%
- 질문: 이메일 50건

Graph

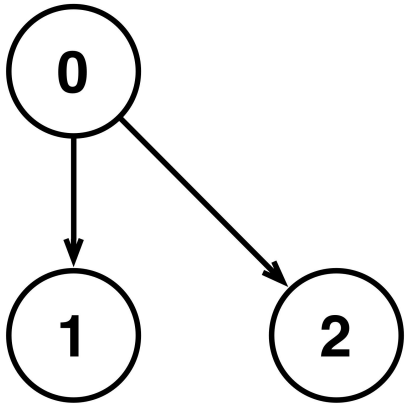
Graph

- Node, Edge로 구성
  - Ex) G = (V, E)
    V = {1,2,3,4,5,6}
    E = {{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}



Dictionary type Graph

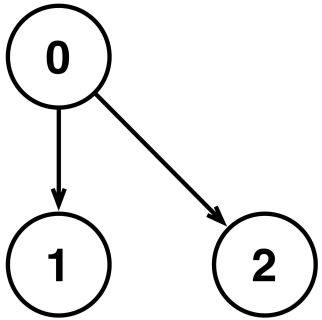
- Node는 Dictionary의 key로 표현
- Edge는 Dictionary의 value로 표현
  - value는 Set type으로 표현



Dictionary type Graph

- Node는 Dictionary의 key로 표현
- Edge는 Dictionary의 value로 표현
  - value는 Set type으로 표현
- 주의! 이론 교재와 다른 표현법!
  - 빠른 실습을 위해서 사용

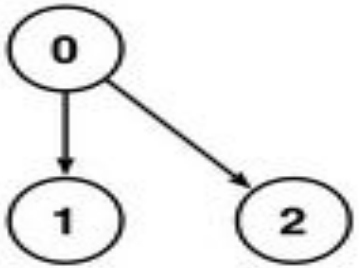
```
EX_GRAPH0 = {0: set([1, 2]),
             1: set([]),
             2: set([])}
```



Graph draw 01

- graph_draw_01.py

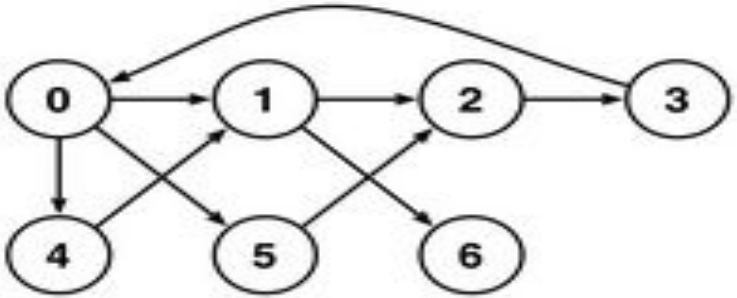
```
# 그래프 생성
graph = dict()

# 노드 생성
graph[0] = set()
graph[1] = set()
graph[2] = set()

# 에지 생성
#graph[0].add((1, 2))
graph[0].add(1)
graph[0].add(2)
```



Graph draw 02

- graph_draw_02.py

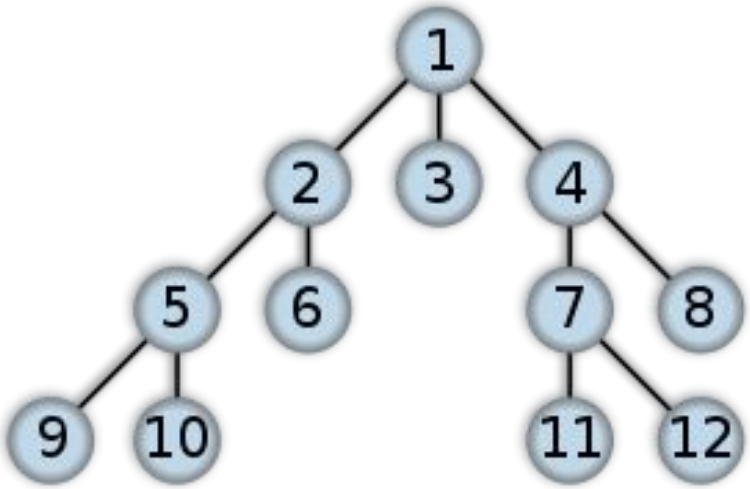
```python
# 그래프 생성
graph = dict()

# 노드 생성
graph[0] = set()
graph[1] = set()
graph[2] = set()
graph[3] = set()
graph[4] = set()
graph[5] = set()
graph[6] = set()

# 에지 생성
graph[0].add(1)
graph[0].add(4)
graph[0].add(5)
graph[1].add(2)
graph[1].add(6)
graph[2].add(3)
graph[3].add(0)
graph[4].add(1)
graph[5].add(2)
```

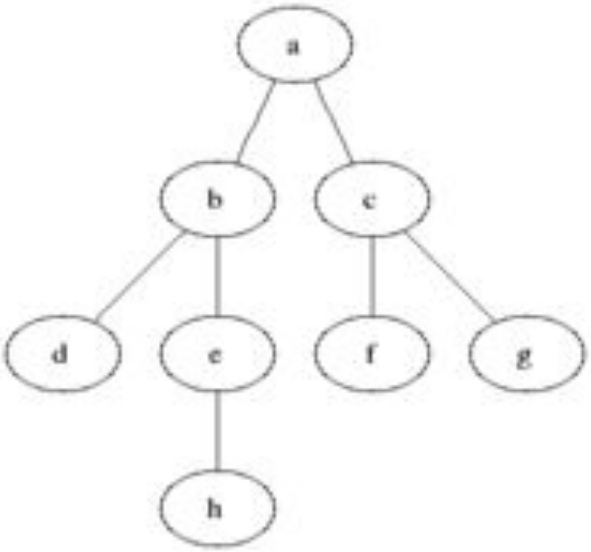


Breadth First Search

너비 우선 탐색

- 노드->노드의 모든 이웃->... 순서





너비 우선 탐색: 알고리즘1

- 알고리즘1

```
Breadth-First-Search(Graph, root):

    for each node n in Graph:
        n.distance = INFINITY
        n.parent = NIL

    create empty queue Q

    root.distance = 0
    Q.enqueue(root)

    while Q is not empty:
        current = Q.dequeue()
        for each node n that is adjacent to current:
            if n.distance == INFINITY:
                n.distance = current.distance + 1
                n.parent = current
                Q.enqueue(n)
```

너비 우선 탐색: 알고리즘2

- 알고리즘2

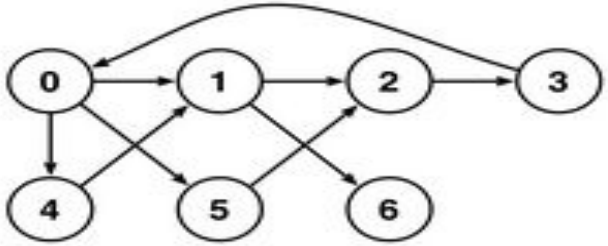
Algorithm 3: BFS-Visited

Input: Undirected graph $g = (V, E)$; source node i.
Output: $Visited$: the set of all nodes visited by the algorithm.

```
1  Initialize Q to an empty queue;
2  Visited ← {i};
3  enqueue(Q, i);
4  while Q is not empty do
5      j ← dequeue(Q);
6      foreach neighbor h of j do
7          if h ∉ Visited then
8              Visited ← Visited ∪ {h};
9              enqueue(Q, h);
10 return Visited;
```

BFS 출력 결과

- 방문: 출력
- def dict_bfs(graph, start_node):



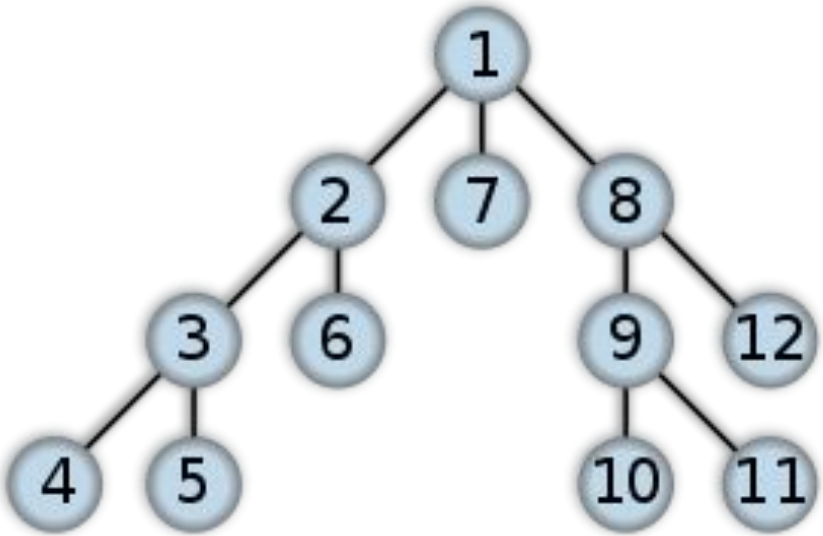

```
BFS
Visit 0
Visit 1
Visit 4
Visit 5
Visit 2
Visit 6
Visit 3
```

```
4   def main():
5       ex_graph02 = dict()
6       ex_graph02[0] = set()
7       ex_graph02[1] = set()
8       ex_graph02[2] = set()
9       ex_graph02[3] = set()
10      ex_graph02[4] = set()
11      ex_graph02[5] = set()
12      ex_graph02[6] = set()
13      ex_graph02[0].add(1)
14      ex_graph02[0].add(4)
15      ex_graph02[0].add(5)
16      ex_graph02[1].add(2)
17      ex_graph02[1].add(6)
18      ex_graph02[2].add(3)
19      ex_graph02[3].add(0)
20      ex_graph02[4].add(1)
21      ex_graph02[5].add(2)
22
23      print('BFS')
24      dict_bfs(ex_graph02, 0)
```

Depth First Search

깊이 우선 탐색

- 노드->노드의 한 이웃->... 순서



깊이 우선 탐색: 알고리즘

- 알고리즘1, 2

Pseudocode [edit]

Input: A graph G and a vertex *v* of G

Output: All vertices reachable from *v* labeled as discovered

A recursive implementation of DFS:[5]

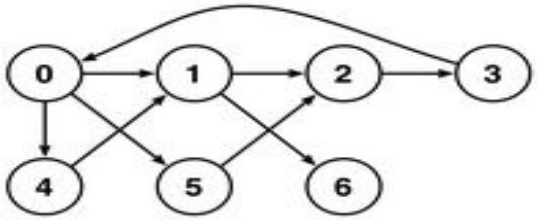
```
1  procedure DFS(G,v):
2      label v as discovered
3      for all edges w in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```

A non-recursive implementation of DFS:[6]

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5            v = S.pop()
6            if v is not labeled as discovered:
7                label v as discovered
8                for all edges from v to w in G.adjacentEdges(v) do
9                    S.push(w)
```

DFS 출력 결과

- 방문: 출력
- def dict_dfs(graph, start_node):




```
DFS
Visit 0
Visit 5
Visit 2
Visit 3
Visit 4
Visit 1
Visit 6
```

```
4   def main():
5       ex_graph02 = dict()
6       ex_graph02[0] = set()
7       ex_graph02[1] = set()
8       ex_graph02[2] = set()
9       ex_graph02[3] = set()
10      ex_graph02[4] = set()
11      ex_graph02[5] = set()
12      ex_graph02[6] = set()
13      ex_graph02[0].add(1)
14      ex_graph02[0].add(4)
15      ex_graph02[0].add(5)
16      ex_graph02[1].add(2)
17      ex_graph02[1].add(6)
18      ex_graph02[2].add(3)
19      ex_graph02[3].add(0)
20      ex_graph02[4].add(1)
21      ex_graph02[5].add(2)
22
23      print('DFS')
24      dict_dfs(ex_graph02, 0)
```

실습! Going Home

- ~~집으로 돌아가는 길을 찾자~~
- Dict 기반 BFS/DFS 구현하기!
  - 완료 후 조교/TA에게 확인받고 귀가
  - 이번주 과제 점수 중 2점

더 나은 형태로 연습(과제)

Graph

- 대부분의 언어에서(알고리즘 교재 혹은 코딩 문제) 클래스를 활용하여 구현
- Graph는 Vertex와 Edge로 구성
- Class Vertex
- Class Edge
    - 이번 과제에서는 Graph 멤버 변수로 표현
    - [18. 11. 20. 추가] 수업시간에 이야기했던 대로 정석은 Vertex와 Edge 클래스로 구현합니다. 하지만 이번 과제에서는 Python 3의 특징을 이용하여 Edge클래스는 Graph 멤버 변수(dict 타입)로 표현합니다. 자꾸 질문이 들어와서 적어둡니다.

교재에 있는 Vertex, Edge

- http://interactivepython.org/courselib/static/pythonds/Graphs/Implementation.html
- 2개의 클래스를 이용하여 6개의 꼭지점, 9개의 Edge로 이루어짐
- 예제 실행해볼 것!
  - 클래스, 객체, 멤버 함수, 멤버 변수!!
- 더 나은 코딩
  - Google Python Style로 변경해보자.

```
>>> g = Graph()
>>> for i in range(6):
...     g.addVertex(i)
>>> g.vertList
{0: <adjGraph.Vertex instance at 0x41e18>,
 1: <adjGraph.Vertex instance at 0x7f2b0>,
 2: <adjGraph.Vertex instance at 0x7f288>,
 3: <adjGraph.Vertex instance at 0x7f350>,
 4: <adjGraph.Vertex instance at 0x7f328>,
 5: <adjGraph.Vertex instance at 0x7f300>}
>>> g.addEdge(0,1,5)
>>> g.addEdge(0,5,2)
>>> g.addEdge(1,2,4)
>>> g.addEdge(2,3,9)
>>> g.addEdge(3,4,7)
>>> g.addEdge(3,5,3)
>>> g.addEdge(4,0,1)
>>> g.addEdge(5,4,8)
>>> g.addEdge(5,2,1)
>>> for v in g:
...     for w in v.getConnections():
...         print("( %s , %s )" % (v.getId(), w.getId()))
...
( 0 , 5 )
( 0 , 1 )
( 1 , 2 )
( 2 , 3 )
( 3 , 4 )
( 3 , 5 )
( 4 , 0 )
( 5 , 4 )
( 5 , 2 )
```

확실히 이해가 되었는가? 확인

- Edge의 거리(무게)도 함께 출력하도록 변경
- 객체, 멤버 변수, 그리고 다시 객체

```
(0, 5, 2)
(1, 2, 4)
(2, 3, 9)
(3, 4, 7)
(3, 5, 3)
(4, 0, 1)
(5, 4, 8)
(5, 2, 1)
```

```
>>> g = Graph()
>>> for i in range(6):
...     g.addVertex(i)
>>> g.vertList
{0: <adjGraph.Vertex instance at 0x41e18>,
 1: <adjGraph.Vertex instance at 0x7f2b0>,
 2: <adjGraph.Vertex instance at 0x7f288>,
 3: <adjGraph.Vertex instance at 0x7f350>,
 4: <adjGraph.Vertex instance at 0x7f328>,
 5: <adjGraph.Vertex instance at 0x7f300>}
>>> g.addEdge(0,1,5)
>>> g.addEdge(0,5,2)
>>> g.addEdge(1,2,4)
>>> g.addEdge(2,3,9)
>>> g.addEdge(3,4,7)
>>> g.addEdge(3,5,3)
>>> g.addEdge(4,0,1)
>>> g.addEdge(5,4,8)
>>> g.addEdge(5,2,1)
>>> for v in g:
...     for w in v.getConnections():
...         print("( %s , %s )" % (v.getId(), w.getId()))
...
( 0 , 5 )
( 0 , 1 )
( 1 , 2 )
( 2 , 3 )
( 3 , 4 )
( 3 , 5 )
( 4 , 0 )
( 5 , 4 )
( 5 , 2 )
```

과제: Class기반 BFS/DFS 구현

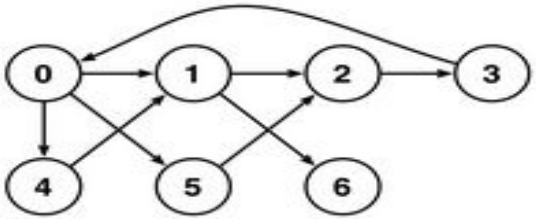
- 과제1: 아래 그래프를 그려서 BFS/DFS 수행
- 과제2: 노드 5개 이상, 에지 5개 이상 그래프 3개 그리고 탐색
  - 그래프 모양, 탐색 결과 설명

```
36
37
38
39
```

```
print('BFS')
bfs(g.get_vertex(0))
print('DFS')
dfs(g.get_vertex(0))
```

```
BFS
Visit 0
Visit 1
Visit 5
Visit 4
Visit 2
Visit 6
Visit 3
```

```
DFS
Visit 0
Visit 4
Visit 1
Visit 6
Visit 2
Visit 3
Visit 5
```



기타 유용한 정보

과제 요약: Class 기반 Graph 탐색

- Class기반 Graph에 대한 Breadth First Search, Depth First Search 구현
  - 함수로 구현
  - 주어진 그래프(노드 7개짜리)에서 결과 확인 및 설명
- 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

실습 숙제 제출

- 숙제 제출 기한: 2018. 11. 21. 23:59:59
  - 실습 전 날
- 파일 제목: AL_학번_이름_10.zip
  - 파일 제목 다를 시 채점 안 합니다.
  - .egg 안 됨!

실습 숙제 제출할 것

- 2가지 파일을 제출
- AL_학번_이름_숙제번호.zip
  - 소스코드
    - Pycharm을 사용했을 경우 Project 디렉터리에 .idea, venv 같은 디렉터리는 제외
    - Jupyter + IPython을 사용했을 경우 ‘File - Download as’ 에서 .py 다운로드 가능
  - AL_학번_이름_숙제번호.pdf
    - 보고서는 무조건 .pdf
    - .hwp, .doc 등 채점 안 함

실습 보고서에 들어가야할 것

- 목표(할 일)
- 과제를 해결하는 방법
  - 알아야 할 것
- 과제를 해결한 방법
  - 주요 소스코드: 굳이 소스코드 전체를 붙일 필요는 없음
- 결과화면
  - 결과화면 설명(해석), 테스트코드 통과
- 보고서는 기본적으로 '내가 숙제를 했음'을 보이는 것
  - 지나치게 대충 작성하면 의심하게 됨

출석부 및 실습 점수가 궁금하다면?

- 출석부 및 실습 채점표
  - 수업 시작 후 30분까지 지각, 이후 결석
  - 실습 딜레이 1일당: -2점
    - 딜레이 2일까지: -2
    - 이후 -1씩 추가
- 튜터의 테스트 결과

질문이 생기면?

- 이름: 문현수
- 전공: 통신및보안
- 과정: 석박사통합과정 8학기
- 연구실: 데이터네트워크연구실(공5633)
- 메일: munhyunsu@cs-cnu.org
- 알고리즘은 함께 해결해가는 과목이므로 과감하게 연락
- 이메일로 처리가 안 되는 급한일: 문자/전화 등