2021-2 자료구조및실습 실습과제 08

학번	2020136129	이름	최수연

1) 각 문제에 대한 분석과 및 해결 방법

1. 교재 388-389쪽의 실습문제 P10.1, P10.2와 P10.6

[문제분석 및 해결 방법]

(1) P10.1

- DFS 재귀함수의 경우, 매개변수로 받은 visited의 첫 번째 값을 True로 설정하고, 해당 인덱스의 정점을 출력한 뒤, for 문과 if 문을 사용하여 순환되도록 한다. for 문을 통해 인접행렬의 한 열의 행의 수만큼 반복하여 만약 해당 값이 1이고 visited가 False이면 다시 순환되도록 하였다. 그렇게 해서 얻은 DFS 결괏값은 'A B D C E G H F' 이다.

(2) P10.2

- BFS의 경우, 매개변수로 받은 visited의 첫 번째 값을 True로 설정하고, queue를 사용하여 큐가 공백이 아닌 동안 while 반복문을 실행한다. 먼저 큐에 있는 첫 번째 값을 꺼내 node에 저장하고, vtx 안에 node에 해당하는 인덱스값을 출력한 뒤, for 문과 if 문을 사용하여 반복하도록 한다. 만약 adj 행렬의 해당 값이 1이고 visited가 False이면 해당 인덱스값의 visited를 True로 변환한 뒤, queue에 해당 인덱스값을 넣는다. 그렇게 해서 얻은 BFS 결괏값은 'A B C D E F G H' 이다.

(3) P10.6

- find_connected_component 함수를 통해 그래프의 연결 성분 개수를 출력하고 10.2의 너비우선탐색의 bfs 함수에서 color 매개변수를 추가하여 변형한 bfs_cc 함수를 통해 정점 리스트를 반환하도록 한다. 그리고 find_bridges 함수를 통해 간선을 없앴을 때 find_connected_component 함수의 반환 값이 1보다 크면 count에 1을 더하고 해당 bridge를 출력하고 간선을 다시 붙여준다.

2) 자신이 구현한 주요 코드

```
# P10.1
def dfs_recur(adj, vtx, visited, s):
    visited[s] = True
    print(vtx[s], end=' ')
    for i in range(len(adj[s])):
        if adj[s][i] == 1 and not visited[i]:
            dfs_recur(adj, vtx, visited, i)

def dfs(adj, vtx, s):
    n = len(adj)
    visited = [False]*n
    dfs_recur(adj, vtx, visited, s)
```

```
# P10.2
def bfs(adj, vtx, s):
    n = len(adj)
```

```
    visited = [False]*n
    queue = [s]
    visited[s] = True
    while queue:
        node = queue.pop(0)
        print(vtx[node], end=' ')
        for i in range(len(adj)):
            if adj[node][i] == 1 and not visited[i]:
                visited[i] = True
                queue.append(i)
```

```
# P10.6
def bfs_cc(adj, vtx, color, s, visited):
    n = len(adj)
    visited = [False]*n
    queue = [s]
    visited[s] = True
    while queue:
        node = queue.pop(0)
        for i in range(len(adj)):
            if adj[node][i] == 1 and not visited[i]:
                visited[i] = True
                queue.append(i)
                color.append(vtx[i])
    return color

def find_connected_component(adj, vtx):
    n = len(vtx)
    visited = [False]*n
    colorList = []
    for vtx in adj:
        if vtx not in visited:
            color = bfs_cc(adj, vtx, [], 0, visited)
            colorList.append( color )
    return len(colorList)
```

3) 다양한 입력에 대한 테스트 결과

 # P10.1

```
DFS : A B D C E G H F
```

 # P10.2

```
BFS : A B C D E F G H
```

 # P10.6

```
find_bridges :
 Bridge1: (A,B)
 Bridge2: (A,D)
 Bridge3: (B,C)
 Bridge4: (B,D)
 Bridge5: (B,E)
 Bridge6: (C,F)
 Bridge7: (D,E)
```

4) 코드에 대한 설명 및 해당 문제에 대한 고찰

- (코드 설명은 **1) 각 문제에 대한 분석 및 해결 방법** 참고)

5) 이번 과제에 대한 느낀점

이번 과제는 생각보다 이해하기 어려워서 좀 오래 걸렸다. 그래도 인접행렬을 사용한 DFS, BFS에 대해 공부할 수 있어서 좋았다. 이번 과제에 대한 모범 답안을 보고 싶다.

6) 궁금한 점이나 건의사항

딱히 없습니다.

7) 자신이 구현한 전체 코드

#P10.1

```
def dfs_recur(adj, vtx, visited, s):
    visited[s] = True
    print(vtx[s], end=' ')
    for i in range(len(adj[s])):
        if adj[s][i] == 1 and not visited[i]:
            dfs_recur(adj, vtx, visited, i)

def dfs(adj, vtx, s):
    n = len(adj)
    visited = [False]*n
    dfs_recur(adj, vtx, visited, s)

vertex =  ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
adjMat = [ [ 0,  1,  1,  0,  0,  0,  0,  0 ],
           [ 1,  0,  0,  1,  0,  0,  0,  0 ],
           [ 1,  0,  0,  1,  1,  0,  0,  0 ],
           [ 0,  1,  1,  0,  0,  1,  0,  0 ],
           [ 0,  0,  1,  0,  0,  0,  1,  1 ],
           [ 0,  0,  0,  1,  0,  0,  0,  0 ],
```

```
              [ 0,    0,    0,    0,    1,    0,    0,    1 ],
              [ 0,    0,    0,    0,    1,    0,    1,    0 ] ]

print('DFS : ', end = "")
dfs(adjMat, vertex, 0)
print()
```

#P10.2

```
from queue import Queue

def bfs(adj, vtx, s):
    n = len(adj)
    visited = [False]*n
    queue = [s]
    visited[s] = True
    while queue:
        node = queue.pop(0)
        print(vtx[node], end=' ')
        for i in range(len(adj)):
            if adj[node][i] == 1 and not visited[i]:
                visited[i] = True
                queue.append(i)

vertex =    ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
adjMat = [ [ 0,    1,    1,    0,    0,    0,    0,    0 ],
             [ 1,    0,    0,    1,    0,    0,    0,    0 ],
             [ 1,    0,    0,    1,    1,    0,    0,    0 ],
             [ 0,    1,    1,    0,    0,    1,    0,    0 ],
             [ 0,    0,    1,    0,    0,    0,    1,    1 ],
             [ 0,    0,    0,    1,    0,    0,    0,    0 ],
             [ 0,    0,    0,    0,    1,    0,    0,    1 ],
             [ 0,    0,    0,    0,    1,    0,    1,    0 ] ]

print('BFS : ', end="")
bfs(adjMat, vertex, 0)
print()
```

#P10.6

```
from queue import Queue

def bfs_cc(adj, vtx, color, s, visited):
    n = len(adj)
    visited = [False]*n
    queue = [s]
    visited[s] = True
```

```
    while queue:
        node = queue.pop(0)
        for i in range(len(adj)):
            if adj[node][i] == 1 and not visited[i]:
                visited[i] = True
                queue.append(i)
                color.append(vtx[i])
    return color

def find_connected_component(adj, vtx):
    n = len(vtx)
    visited = [False]*n
    colorList = []
    for vtx in adj:
        if vtx not in visited:
            color = bfs_cc(adj, vtx, [], 0, visited)
            colorList.append( color )
    return len(colorList)

def find_bridges(adj, vtx):
    n = len(vtx)
    count = 0
    for i in range(n):       # 모든 간선에 대하여
        for j in range(i+1, n):
            if adj[i][j] != 0:    # 간선이 있으면
                adj[i][j] = adj[j][i] = 0    # 간선을 모두 없앰
                if find_connected_component(adj, vtx) > 1:
                    count += 1
                    print(" Bridge%d: (%s,%s)"%(count, vtx[i], vtx[j]))
                adj[i][j] = adj[j][i] = 1     # 다시 간선을 붙여줌
    return count


vertex =    ['A', 'B', 'C', 'D', 'E', 'F']
adjMat = [ [ 0,    1,    0,    1,    0,    0 ],
           [ 1,    0,    1,    1,    1,    0 ],
           [ 0,    1,    0,    0,    0,    1 ],
           [ 1,    1,    0,    0,    1,    0 ],
           [ 0,    1,    0,    1,    0,    0 ],
           [ 0,    0,    1,    0,    0,    0 ] ]

print('find_bridges : ')
find_bridges(adjMat, vertex)
print()
```