Algorithm HW #3

컴파일 환경: g++ (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0

.out 파일 실행 권한이 없는 경우: chmod +x [파일이름] (ex. chmod +x 3-1.out)

3-1. DFS(Depth First Search) with edge classification

※ 파일 이름: 3-1.cpp, 3-1.out

1) 헤더 파일 목록

- <iostream>, <fstream>, <queue>

2) Functions

function	설명
Graph Build_Graph(int l, ifstream &infile)	크기 l을 인자로 받아 ifstream으로 특정 파일로부터 l x l인 인접행렬을 읽고 이를 바탕으로 Graph G를 만들어 return한다.
void DFS(Graph G, int n)	Graph G와 이 그래프의 노드 수 n을 인자로 받아 DFS를 실행한다. 처음에는 모든 노드의 상태를 초기화하고 DFS 규칙에 맞게 각 노드를 방문하는 DFS_visit 함수를 실행한다.
void DFS_visit(Graph G, int u)	index가 u인 노드의 상태를 gray로 바꾸고 queue에 넣는다. 아직 방문하지 않은 인접 노드 edge가 있으면 DFS_visit을 재호출한다. 노드의 탐색이 끝나면(연결된 모든 노드를 이미 다 방문한 경우) 상태를 black으로 바꾼다. 이 과정에서 각 edge의 종류를 출력한다. 방문한 노드가 white: tree edge 방문한 노드가 gray: back edge 방문한 노드가 black인 경우 1) (u,v): u의 발견시점이 v보다 빠른 경우: forward edge 2) 그 외: cross edge 또한 discovery time과 finished time을 저장한다.

- Queue는 방문한 노드 index를 기록하는 데 쓴다. DFS가 끝난 뒤 queue에서 차례대로 원소를 빼내면서 DFS 결과를 출력한다.

3) 실행 화면 캡처

- 주어진 예시

```
skiller09@skiller09-VirtualBox:~/algorithms/HW3$ g++ -o 3-1.out 3-1.cpp
skiller09@skiller09-VirtualBox:~/algorithms/HW3$ ./3-1.out
(1, 2) tree edge
(2, 5) tree edge
(5, 4) tree edge
(4, 2) back edge
(1, 4) forward edge
(3, 5) cross edge
(3, 6) tree edge
(6, 6) back edge
1 2 5 4 3 6
```

3-2. DFS with connected component identification

※ 파일 이름: 3-2.cpp, 3-2.out

1) 헤더 파일 목록

- <iostream>, <fstream>, <queue>

2) Functions

function	설명
Graph Build_Graph(int l, ifstream &infile)	(3-1과 동일) 크기 l을 인자로 받아 ifstream으로 특정 파일로부터 l x l인 인접행렬을 읽고 이를 바탕으로 Graph G를 만들어 return한다.
void DFS(Graph G, int n)	Graph G와 이 그래프의 노드 수 n을 인자로 받아 DFS를 실행한다. 처음에는 모든 노드의 상태를 초기화하고 groupnum++을 한 뒤 DFS 규칙에 맞게 각 노드를 방문하는 DFS_visit 함수를 실행한다.
void DFS_visit(Graph G, int u)	index가 u인 노드의 상태를 gray로 바꾸고, queue에 노드의 index를 넣고, group[index]에 groupnum을 저장한다. 아직 방문하지 않은 인접 노드 edge가 있으면 DFS_visit을 재호출한다. 노드의 탐색이 끝나면(연결된 모든 노드를 이미 다 방문한 경우) 상태를 black으로 바꾼다. Time을 저장한다. group은 각 노드 별 component number를 저장한 배열이다.

- Queue는 방문한 노드 index를 기록하는 데 쓴다. DFS가 끝난 뒤 queue에서 차례대로 원소를 빼내면서 DFS 결과를 출력한다.

3) 실행 화면 캡처

- 주어진 예시

```
skiller09@skiller09-VirtualBox:~/algorithms/HW3$ g++ -o 3-2.out 3-2.cpp
skiller09@skiller09-VirtualBox:~/algorithms/HW3$ ./3-2.out
1 2 5 3 6 4
1: 1
2: 1
3: 2
4: 3
5: 1
6: 2
```

3-3. Topological sort

※ 파일 이름: 3-3.cpp, 3-3.out

1) 헤더 파일 목록

- <iostream>, <fstream>, <stack>

2) Functions

function	설명
Graph Build_Graph(int l, ifstream &infile)	(3-1과 동일) 크기 l을 인자로 받아 ifstream으로 특정 파일로부터 l x l인 인접행렬을 읽고 이를 바탕으로 Graph G를 만들어 return한다.
void DFS(Graph G, int n)	(3-1과 동일) Graph G와 이 그래프의 노드 수 n을 인자로 받아 DFS를 실행한다. 처음에는 모든 노드의 상태를 초기화하고 DFS 규칙에 맞게 각 노드를 방문하는 DFS_visit 함수를 실행한다.
void DFS_visit(Graph G, int u)	index가 u인 노드의 상태를 gray로 바꾸고 queue에 넣는다. 아직 방문하지 않은 white 상태의 인접 edge가 있으면 DFS_visit을 재호출한다. 노드의 탐색이 끝나면(연결된 모든 노드를 이미 다 방문한 경우) 상태를 black으로 바꾸고 stack에 노드의 index를 집어넣는다. 만약 인접 노드가 gray이면 cycle이 있으므로 0을 출력하고 프로그램을 종료한다.

- Stack는 방문이 완료된 노드 index를 기록하는 데 쓴다. DFS가 끝난 뒤 Stack이 빌 때까지 top()을 출력과 pop()을 반복하면 topological sort를 출력한 결과가 된다.

3) 실행 화면 캡처

- 주어진 예시