2019-2 임을규 교수님 알고리즘및문제해결기법

Programming Assignment #3

과제 제출: 세 개의 소스코드(.c, .cpp), 세 개의 실행파일(.exe, .out 등), 보고서(.docx, .hwp, .pdf 등)를 압축하여 블랙보드에 업로드 (ex. 3-1.cpp, 3-2.cpp, 3-3.cpp, 3-1.out, 3-2.out, 3-3.out, report3.pdf)

보고서: 1페이지 이상의 자유 양식. 소스코드가 주어졌을 때 이를 컴파일하고 실행하는 과정에서 필요한 정보(컴파일러 버전 또는 필요에 의해 추가로 작성한 명령어 등)에 대해 작성. 즉 과제 명세의 기본적인 형식은 지키되 필요에 의해 추가한 코드에 대해서 사용법 등을 명시한 경우 채점에 반영. 또 실행 과정에서 입력해야 하는 명령어 등이 있는 경우 스크린샷 등을 활용하여 명세.

파일명: HW3_학번.zip

제출 기한: **11월 27일 화요일 15:59**까지

제출 기한에서 한 시간 단위(16:00-16:59, 17:00-17:59, ...)로 10%씩 감점, 28일 02:00 제출부터 0점 처리

인터넷을 참고한 알고리즘 공부는 권장하나, 코드 작성은 본인이 직접 할 것. 소스코드 및 바이너리 파일에 대한 유사도를 검사하여, copy한 과제는 0점 처리

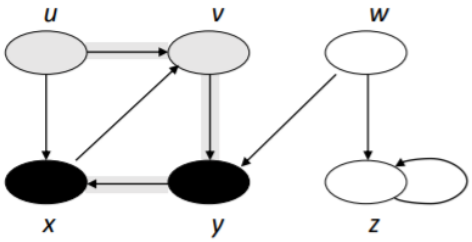
문의 사항: 장준영 조교, lartist@hanyang.ac.kr (제출 관련 문의 등)

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

directed graph 에 대한 인접 행렬이 주어졌을 때, DFS 방식으로 탐색할 때의 모든 edge 의 종류를 출력하고, 마지막으로 DFS 의 결과를 출력하는 코드를 작성하라. (hint: DFS-visit 함수 내에서 발견하는 인접 node 의 종류(색깔)에 따라 출력)

<input 예시 (input3-1.txt)>

```
6                    // node의 개수
0 1 0 1 0 0
0 0 0 0 1 0
0 0 0 0 1 1
0 1 0 0 0 0
0 0 0 1 0 0
0 0 0 0 0 1
```



<output 예시 (stdout)>

```
(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              // DFS 결과
```

- ✓ node는 1개 이상 100개 미만
- ✓ node는 1번부터 1씩 증가하는 번호를 가짐
- ✓ node 방문 순서는 번호 오름차순

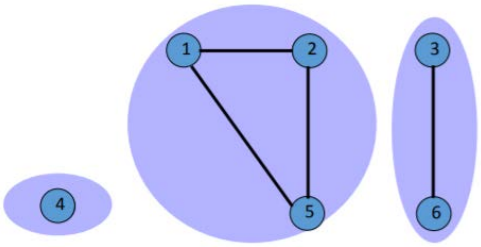
- ✓ 파일입출력 활용 (input3-1.txt)
- ✓ input3-1.txt는 실행 코드와 같은 위치에 있는 것으로 간주
- ✓ DFS 관련 라이브러리 사용 금지. 그 외 제약은 없음 (자료구조 자유 선택)
- ✓ 그 외 기타 사항은 입력, 출력 예시에 맞출 것

3-2. DFS with connected component identification

undirected graph 에 대한 인접 행렬이 주어졌을 때, DFS 의 결과를 출력하고, 각 node 의 component 번호를 출력하는 코드를 작성하라. (hint: DFS-visit 함수 내에서 발견하는 인접 node 는 직전 node 와 같은 component 번호를 가진다.)

<input 예시 (input3-2.txt)>

```
6                    // node의 개수
0 1 0 0 1 0
1 0 0 0 1 0
0 0 0 0 0 1
0 0 0 0 0 0
1 1 0 0 0 0
0 0 1 0 0 0
```



<output 예시 (stdout)>

```
1 2 5 3 6 4              // DFS 결과
1: 1                     // node 번호: component 번호
2: 1
3: 2
4: 3
5: 1
6: 2
```

- ✓ node는 1개 이상 100개 미만
- ✓ node는 1번부터 1씩 증가하는 번호를 가짐
- ✓ node 방문 순서는 번호 오름차순
- ✓ component는 1번부터 1씩 증가하는 번호를 가짐

- ✓ 파일입출력 활용 (input3-2.txt)
- ✓ input3-2.txt는 실행 코드와 같은 위치에 있는 것으로 간주
- ✓ DFS 관련 라이브러리 사용 금지. 그 외 제약은 없음 (자료구조 자유 선택)
- ✓ 그 외 기타 사항은 입력, 출력 예시에 맞출 것

3-3. Topological sort

인접행렬이 주어진 그래프가 DAG(Directed Acyclic Graph)인지 판별하고, DAG 라면 DFS 를 이용한 topological sort 를 수행하여 결과를 출력하라. (hint1: DAG 의 판별은 DFS 의 back edge 를 활용할 수 있다.) (hint2: DFS 의 과정에서 BLACK 이 된 node 의 역순이 topological sort 의 결과이다.)

<input 예시 (input3-3.txt)>

```
7                   // node의 개수
0 0 0 1 1 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 0
0 0 0 0 1 1 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 1 0
```

<output 예시 (stdout)>

```
1                   // DAG가 맞으면 1, 아니면 0
7 3 2 1 4 6 5
```

- ✓ node는 1개 이상 100개 미만
- ✓ node는 1번부터 1씩 증가하는 번호를 가짐
- ✓ node 방문 순서는 번호 오름차순

- ✓ 파일입출력 활용 (input3-3.txt)
- ✓ input3-3.txt는 실행 코드와 같은 위치에 있는 것으로 간주
- ✓ DFS, DAG, topological sort 관련 라이브러리 사용 금지. 그 외 제약은 없음 (자료구조 자유 선택)
- ✓ 그 외 기타 사항은 입력, 출력 예시에 맞출 것