자료구조설계

제 출 일	2017.09.23.
과 제 번 호	03
분 반	01
학 과	컴퓨터공학과
학 번	201602038
이 름	이 미 진

1. 구현 내용 설명

```
        int size;
        String[] vertices;          // 정점
        boolean[][] a;    // 인접한 행렬의 배열
        boolean[] visit;  // 방문했는지 안했는지 체크
        Stack stack = new Stack();        // stack 생성

public Graph(String[] args) {
                this.size = args.length;
                vertices = new String[size];
                System.arraycopy(args, 0, vertices, 0, size);
                a = new boolean[size][size];       // size만큼 인접한 행렬의 배열을
생성
                visit = new boolean[size];         // size만큼 정점 방문을 체크하는
배열을 생성
        }
```

방문한 정점을 체크하는 boolean 타입의 visit 배열을 생성하고, 방문하면 true, 방문하지 않았으면 false이다.

1) 인접 행렬을 이용하는 순환 깊이우선탐색

```
public void recu_dfs(int v) {
                visit[v] = true;  // 방문했으므로 true

                System.out.print(vertices[v]+" "); // 정점 출력
                for( int i=0; i<size; i++ ) {
                        if( a[v][i]  && visit[i] == false) { // 인접하고 아직 방문되지
않은 정점들이 존재한다면
                                recu_dfs(i);       // recu_dfs 메소드를 호출
                        }
                }
        }
```

재귀를 이용하는 reucursive dfs의 구현이다. recu_dfs() 메소드가 실행되면 정점을 방문한 것이므로 visit[] 배열에 true값을 저장한다.
정점을 출력하고 0부터 size 까지 인접하되 아직 방문되지 않은 정점들이 존재하면 recu_dfs() 메소드를 호출하여 재귀적으로 깊이 우선 탐색을 한다.

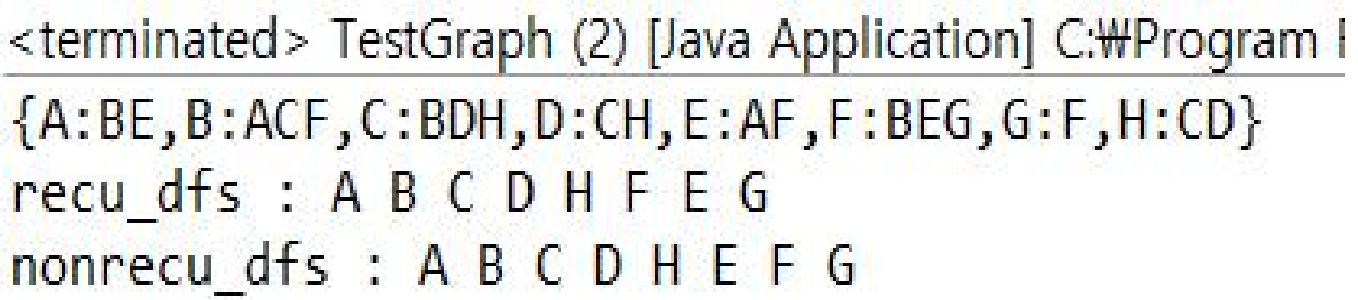
2) 인접 리스트 스택을 이용하는 깊이우선탐색

```
        public void nonrecu_dfs(int v){
                int i;
                int u = 0;        // w에 인접하면서 방문하지 않은 정점 u

                for( i=0; i < size; i++) { // 방문 체크 초기화
                        visit[i] = false;
                }

                visit[v] = true;  // 방문 체크
```

```
                stack.push(v);   // 시작 정점 v를 stack에 삽입

                int w = v;       // w = v

                System.out.print(vertices[v]+" "); // 시작 정점 출력

                for( u = 0; u < size; u++ ){
                        if( a[w][u] && visit[u] == false){ // w에 인접하면서
방문되지 않은 u가 있다면
                                System.out.print(vertices[u]+" "); // 인접 정점 출력

                                visit[u] = true;  // 방문 체크
                                stack.push(u);   // stack에 u 삽입
                                w = u;
                                u = 0;

                        }
                        else if( u == size-1 && visit[u] == true && stack.empty()
== false ){       // w에 인접하면서 방문되지 않은 정점이 없고 스택에 원소가 있다면
                                        w = (int)stack.pop(); // 스택에서 원소
삭제

                                        u = 0;
                                        w = u;
                        }

                        if( stack.empty()) {       // w에 인접하면서 방문되지 않은
정점이 없고 스택이 비어있다면
                                break;  // 종료
                        }
                }
        }
```

stack을 생성하고 w에 인접하면서 방문하지 않은 정점 u을 선언해준다.
방문 했던 결과를 저장한 visit[] 배열을 false로 모두 초기화 시킨다.
recu_dfs() 메소드와 마찬가지로, nonrecu_dfs()메소드가 호출되면 정점을 방문한 것으로 하여 visit[] 배열에 true를 저장한다.
시작 정점 v를 스택에 push하고 w=v 라고 한 후, 시작 정점을 출력한다.
u가 size가 될 때까지, 조건에 따라 수행한다.
1) w에 인접하면서(a[w][u]) 방문되지 않은 u(visit[u]==false) 가 있다면 인접 정점을 출력하고 방문했다고 체크한다. 그리고 stack에 u를 push하고 w=u라고 한다.
2) w에 인접하면서 방문되지 않은 정점이 없고(visit[u] == true) 스택에 원소가 있다면 (stack.empty() == false) 스택에서 원소 u를 삭제해서 w=u 라고 한다.
3) w에 인접하면서 방문되지 않은 정점이 업고 스택이 비어있다면, 종료한다.

2. 실행 결과 화면


<terminated> TestGraph (2) [Java Application] C:₩Program
{A:BE,B:ACF,C:BDH,D:CH,E:AF,F:BEG,G:F,H:CD}
recu_dfs : A B C D H F E G
nonrecu_dfs : A B C D H E F G