

# 자료구조 실습 보고서

## 실습 11. 그래프

2016 년 6 월 3 일

학번: 201404051

이름: 정 용 석

## 1. 실습 문제 소개

### 1) 그래프 생성

Graph.txt 파일을 읽고 파일 내의 정보를 바탕으로 그래프를 생성하고 정점과 간선을 모두 출력하는 프로그램을 만들어야 한다. 실습 내용에 있는 베이스 코드들을 바탕으로 함수들을 만들고 그 중에서도 정점과 간선을 추가하는 함수의 일부분은 개개인이 구현할 수 있어야 한다.

### 2) 그래프 탐색

그래프 생성 프로그램을 바탕으로 깊이 우선 탐색과 너비 우선 탐색 내용을 출력하는 프로그램을 작성한다. 실행 결과 역시 전 실습 결과에 추가하여 나타내야 하며, 실습 내용에 명시된 깊이 우선 탐색과 너비 우선 탐색 알고리즘을 참고하도록 한다. 그 중에서도 깊이 우선 탐색은 재귀 함수를 이용하여 구현하도록 한다.

## 2. 소스 코드

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_VERTICES 26 //정점의 최대 갯수

//탐색을 위한 큐 자료구조

int visited[MAX_VERTICES];

#define INIT_QUEUE() { front = rear = 1;}
#define ENQUEUE(v) {queue[++front] = v;}
#define DEQUEUE() ((front >= rear) ? queue[++rear] : -1)
#define ISEMPY() (front < rear)

int queue[MAX_VERTICES];
int front, rear;

typedef struct _Graph {
    int nVertices;           //정점의 개수
    int nEdges;              //간선의 개수
    int vertex[MAX_VERTICES];
    //그래프에 존재하는 vertex를 구분
    int edge[MAX_VERTICES][MAX_VERTICES];
    //그래프에 존재하는 edge를 구분
} Graph;
```

```

// Graph g를 초기화한다.
void initGraph(Graph *g)
{
    int i, j;
    g->nVertices = 0;
    g->nEdges = 0;

    for (i = 0; i < MAX_VERTICES; i++) {
        g->vertex[i] = 0;
        for (j = 0; j < MAX_VERTICES; j++)
            g->edge[i][j] = 0;
    }
}

// Graph g의 vertex 리스트와 edge 리스트를 출력한다.
void printGraph(Graph *g)
{
    int i, j, ne = 0;

    //정점 집합 출력
    printf("V(G) = { ");
    for (i = 0; i < g->nVertices; i++) {
        printf("%c", 'A' + i);
        if (i < g->nVertices - 1)
            printf(", ");
    }
    printf(" }\n");

    // 간선 집합 출력
    printf("E(G) = { ");
    for (i = 0; i < MAX_VERTICES; i++)
        for (j = i + 1; j < MAX_VERTICES; j++) {
            if (g->edge[i][j] > 0) {
                printf("(%c, %c)", 'A' + i, 'A' + j);
                if (++ne < g->nEdges)
                    printf(", ");
            }
        }
    printf(" }\n");
}

// Graph g에 vertex v를 추가한다.
int insertVertex(Graph *g, int v)
{
    if (!(0 <= v && v < MAX_VERTICES))
        return 0; // 정점 범위 오류
    if (g->vertex[v] == 1)
        return 0; // 이미 존재하는 정점 오류

    // 정점 v의 존재 입력 및 정점 갯수 증가
    g->vertex[v] = 1;
    g->nVertices++;
}

```

```

        return -1;
    }

    // Graph g에 v1-v2를 연결하는 edge를 추가한다.
    int insertEdge(Graph *g, int v1, int v2)
    {
        if (!(0 <= v1 && v1 < MAX_VERTICES))
            return 0; // 정점 범위 오류
        if (g->vertex[v1] == 0)
            return 0; // 없는 정점 오류
        if (!(0 <= v2 && v2 < MAX_VERTICES))
            return 0; // 정점 범위 오류
        if (g->vertex[v2] == 0)
            return 0; // 없는 정점 오류

        //간선 존재 유무 추가 및 간선의 갯수 증가
        g->edge[v1][v2] = 1;
        g->edge[v2][v1] = 1;
        g->nEdges++;

        return -1; // 성공
    }

    // 그래프 파일을 읽어 Graph g에 저장한다.
    int readGraphFile(Graph *g, char* fname) {
        FILE *fp = fopen(fname, "r");
        int nv, ne, i;
        int v1, v2;
        char ch1, ch2;

        if (fp == NULL) return 0; // 파일 오류

        initGraph(g);

        // 정점 갯수를 읽어 정점 추가
        fscanf(fp, "%d", &nv);
        for (i = 0; i < nv; i++)
            insertVertex(g, i);

        // 간선 갯수와 간선 정보를 읽어 간선 추가
        fscanf(fp, "%d", &ne);
        for (i = 0; i < ne; i++) {
            // 첫번째 정점 읽기
            do {
                fscanf(fp, "%c", &ch1);
            } while (ch1 < 'A' || 'Z' < ch1);
            v1 = ch1 - 'A';
            // 두번째 정점 읽기
            do {
                fscanf(fp, "%c", &ch2);
            } while (ch2 < 'A' || 'Z' < ch2);

```

```

        v2 = ch2 - 'A';
        // 간선 (v1, v2) 추가
        insertEdge(g, v1, v2);
    }
    fclose(fp);

    return -1;
}

void dfsCore(Graph *g, int v) {
    int i;
    visited[v] = 1;
    printf("%c ", 'A' + v); // printf v 값
    // 모든 정점 i에서 v와 인접한 정점들 중 방문하지 않은 정점에 관해서 재귀
    for (i = 0; i < MAX_VERTICES; i++) {
        if (g->edge[v][i] == 1)
            if (visited[i] == 0)
                dfsCore(g, i);
    }
}

void depthFirstSearch(Graph *g, int v) {
    int i;

    //visited 플래그 초기화
    for (i = 0; i < MAX_VERTICES; i++)
        visited[i] = 0;

    //정점 v부터 그래프 g의 깊이 우선 탐색 시작
    printf("Depth First Search :");
    dfsCore(g, v);
    printf("\n");
}

void breathFirstSearch(Graph *g, int v) {
    int w, i;

    //큐 초기화
    INIT_QUEUE();

    //visited 플래그 초기화
    for (i = 0; i < MAX_VERTICES; i++)
        visited[i] = 0;

    printf("Breath First Search :");

    visited[v] = 1;
    printf("%c ", 'A' + v); // print v 값
    ENQUEUE(v);
    while (!IS_EMPTY()) {
        v = DEQUEUE();
        for (i = 0; i < MAX_VERTICES; i++) {

```

```

//v의 인접한 정점 i에 관해서 방문하지 않았을 때
if (g->edge[i][v] == 1)
    if (visited[i] == 0) {
        visited[i] = 1;
        printf("%c ", 'A' + i); //print i값
        ENQUEUE(i);
    }
}
}
printf("\n");
}

void main()
{
    Graph G;

    readGraphFile(&G, "graph.txt");
    printGraph(&G);

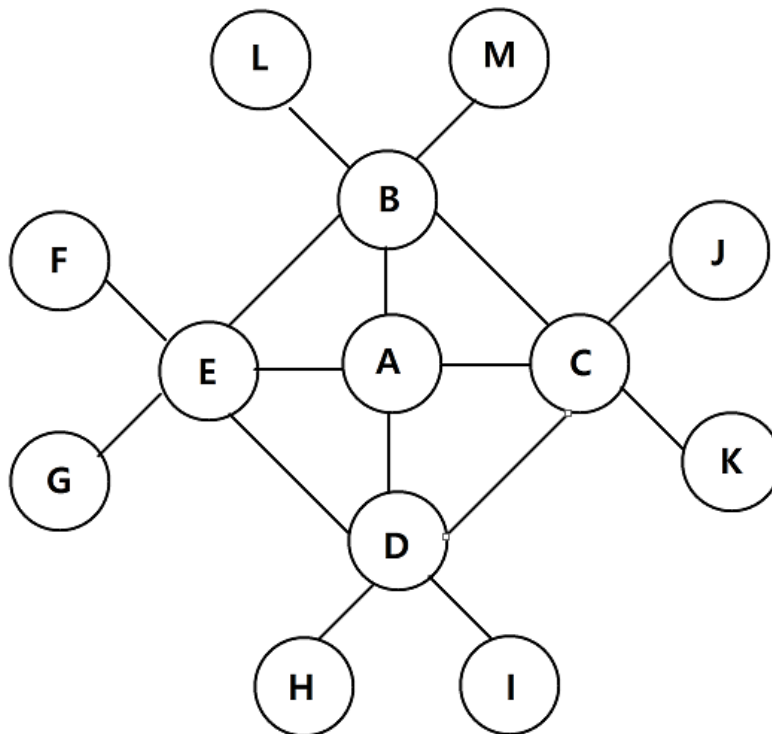
    depthFirstSearch(&G, 0);

    system("pause");
}

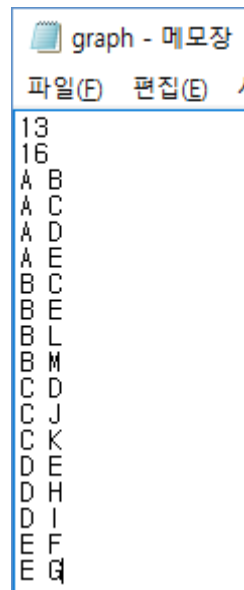
```

### 3. 테스트 결과

<나만의 그래프 G>

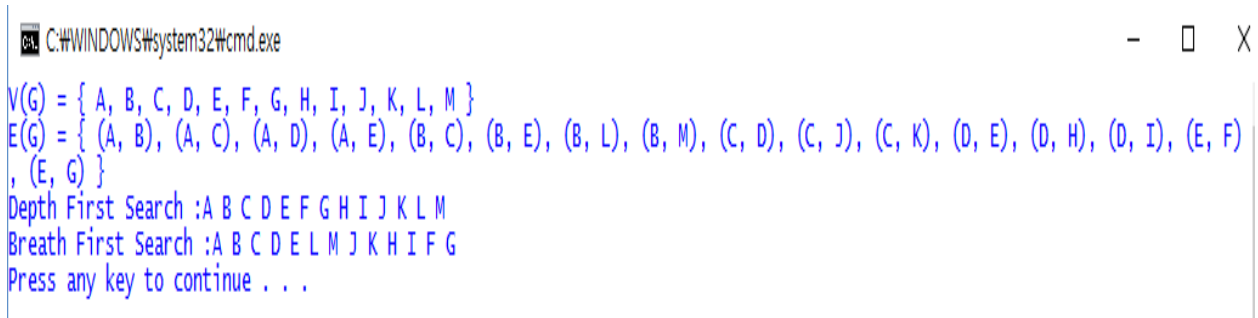


<Graph.txt>



```
graph - 메모장
파일(F) 편집(E)
13
16
A B
A C
A D
A E
B C
B E
B L
B M
C D
C J
C K
D E
D H
D I
E F
E G
```

<실행 결과>



```
C:\WINDOWS\system32\cmd.exe
V(G) = { A, B, C, D, E, F, G, H, I, J, K, L, M }
E(G) = { (A, B), (A, C), (A, D), (A, E), (B, C), (B, E), (B, L), (B, M), (C, D), (C, J), (C, K), (D, E), (D, H), (D, I), (E, F), (E, G) }
Depth First Search :A B C D E F G H I J K L M
Breath First Search :A B C D E L M J K H I F G
Press any key to continue . . .
```

#### 4. 작성자 코멘트

실습 내용 자체가 굉장히 솔직해서 내가 한 부분이 거의 느껴지지 않을 정도이다. 개인적으로 실습을 하면서 느낀 의문점은 그래프 라는 자료 구조가 쉬운 것인지 아니면 교수님이 올려 놓으신 코드 자체가 쉽게 풀이된 것인지 였다. 사실 후자에 더 가깝다고 생각한다. 굉장히 간단하고 일관적인 코드 구조라 정말 밥상에 숟가락만 얹은 느낌이 들 정도 였다.

그나마 테스트는 개인적인 창의력과 노력이 묻어나길 바라는 마음으로 만들긴 했지만 그러한지는 잘 모르겠다. 그래프를 만들기 전에 생각한 것은 깊이 우선 탐색과 너비 우선 탐색의 차이점을 어느정도 보여주고 싶었고, 따라서 위와 같은 모양의 그래프를

만들게 되었고, 결과 내용을 봐도 나의 의도가 어느정도는 들어가지 않았나 싶다. 시작 정점 A 를 기준으로 B, C, D, E 를 한바퀴 돌고 깊이 우선 탐색의 경우는 가장 마지막으로 방문했던 정점에 관하여 다시 한 번 탐색을 실시하고, 너비 우선 탐색의 경우는 시작 정점에서 가장 인접했던 B 에 관하여 탐색을 실시하게 된다. 추가적으로 너비 우선 탐색 같은 경우는 재귀적 호출을 위해 bfsCore 라는 함수를 추가하여 구현했다.