

# 자료구조2 실습

## 9주차 과제



제출일	21.11.02	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2 실습	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

# | 목 차 |

## 1. 그래프: 깊이 우선 탐색

- 1.1 문제 분석
- 1.2 소스 코드
- 1.3 소스 코드 분석
- 1.4 실행 창
- 1.5 느낀 점

## 2. 그래프: 너비 우선 탐색

- 2.1 문제 분석
- 2.2 소스 코드
- 2.3 소스 코드 분석
- 2.4 실행 창
- 2.5 느낀 점

## 3. 그래프를 하며 느낀 점

## 1. 그래프

### 깊이 우선 탐색 프로그램

---

- 382 페이지에 있는 프로그램 10.3의 깊이 우선 탐색 프로그램을 참고하여 파일에 입력되어있는 정점과 간선의 정보를 이용하여 그래프를 구성하고 이 그래프를 깊이 우선 탐색을 통해 출력하는 코드를 작성하시오.

#### 1.1 문제 분석

조건 1 그래프가 그려지는 과정과 함께 깊이를 탐색하는 순서를 그림

조건 2 행렬을 이용하여 그래프 정보 저장

조건 3 data.txt파일들을 직접 생성하여 입력받음

- 이 문제는 깊이 우선 탐색 알고리즘을 이용하여 그래프의 정점을 탐색하는 프로그램이다. 깊이를 이용한 우선 탐색은 스택 또는 순환을 이용하여 프로그래밍하는데 이번 경우는 순환을 이용하여 프로그래밍 해보았다. 그래프는  $n \times n$  행렬을 이용하여 인접행렬 그래프로 만들고, 행렬의 크기는 메모리의 크기를 아낄 수 있도록 포인터를 동적할당하여 사용한다. 깊이 우선 탐색은 그래프의 시작 정점에서 출발하여 시작정점을 방문하였다고 표시한 후 인접한 정점들 중에서 아직 방문이 되지 않은 정점을 찾고 그 정점을 시작으로 하여 다시 깊이 우선 탐색을 사용한다. 만약 인접 정점이 모두 방문 되었다면 프로그램을 종료한다. 깊이 우선 탐색은 방문 여부를 기록하기 위해 배열을 사용하며, 모든 정점의 배열 값은 FALSE로 초기화 되고 정점이 방문될 때마다 정점 방문 배열값은 TRUE로 변경된다. 이번 문제의 경우 메모리를 아끼기 위해 방문 변수를 포인터로 선언하여 크기에 맞게 동적할당하여 사용한다.

## 1.2 소스 코드

```
//
//  학번 : 20154012
//  학과 : 컴퓨터소프트웨어공학부
//  이름 : 김준민
//  담당 과목 : 그래프 알고리즘 수업
//

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

typedef struct {
    int n; //그래프의 노드 개수
    int **adj_list; //인접리스트를 이용한 그래프
}GraphType;

int VtxCnt; //탐색하면서 확인하는 변수

void Init(GraphType* g, int n); //그래프 초기화 함수
void Insert_Vertex(GraphType* g, int n); //꼭짓점 삽입 함수
void Insert_Edge(GraphType* g, int start, int end); //간선 삽입 함수
void dfs_visit(GraphType* g, int v); //깊이 우선 탐색 함수
void Delete_Edge(GraphType* g); //그래프 삭제 함수

int main() {
    FILE* fptr;
    int max = 10000; //max에 int의 가장 작은 값 삽입
    char* str; //문자를 저장하는 변수
    int i, j, k, l; //문자를 저장하는 변수
    GraphType* g;
    fptr = fopen("data.txt", "r");
    if (!fptr) {
        printf("File not open");
        return 0;
    }
    while (!feof(fptr)) {
        fscanf(fptr, "%d", &n); //노드 개수 입력
        g = (GraphType*)malloc(sizeof(GraphType));
        Init(g, n);
        while (!feof(fptr)) {
            fscanf(fptr, "%d %d", &i, &j);
            Insert_Edge(g, i, j);
        }
        dfs_visit(g, 0);
        Delete_Edge(g);
        fclose(fptr);
        return 0;
    }
}

void Init(GraphType* g, int n) {
    g->n = n;
    g->adj_list = (int**)malloc(sizeof(int*) * n);
    for (i = 0; i < n; i++) {
        g->adj_list[i] = (int*)malloc(sizeof(int) * n);
    }
}

void Insert_Vertex(GraphType* g, int n) {
    g->n++;
    g->adj_list = (int**)realloc(g->adj_list, sizeof(int*) * g->n);
    g->adj_list[g->n-1] = (int*)malloc(sizeof(int) * g->n);
}

void Insert_Edge(GraphType* g, int start, int end) {
    if (start < 0 || end < 0 || start > g->n || end > g->n) {
        printf("Invalid edge\n");
        return;
    }
    g->adj_list[start][end] = 1;
}

void dfs_visit(GraphType* g, int v) {
    int i;
    if (v < 0 || v > g->n) {
        printf("Invalid vertex\n");
        return;
    }
    VtxCnt++;
    printf("Vertex: %d", v);
    Delete_Edge(g, v);
    for (i = 0; i < g->n; i++) {
        if (g->adj_list[v][i] == 1) {
            dfs_visit(g, i);
        }
    }
}

void Delete_Edge(GraphType* g) {
    int i, j;
    for (i = 0; i < g->n; i++) {
        for (j = 0; j < g->n; j++) {
            g->adj_list[i][j] = 0;
        }
    }
    g->n--;
    g->adj_list = (int**)realloc(g->adj_list, sizeof(int*) * g->n);
    g->adj_list[g->n-1] = (int*)malloc(sizeof(int) * g->n);
}

void Delete_Edge(GraphType* g, int v) {
    int i;
    for (i = 0; i < g->n; i++) {
        g->adj_list[i][v] = 0;
    }
    g->n--;
    g->adj_list = (int**)realloc(g->adj_list, sizeof(int*) * g->n);
    g->adj_list[g->n-1] = (int*)malloc(sizeof(int) * g->n);
}
```

## 1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 그래프 깊이 우선 탐색 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

typedef struct {
    int n;
    int **adj_mat;
}GraphType;

int *visited;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 참을 나타내는 TRUE와 거짓을 나타내는 FALSE를 정의한다.

2. 그래프의 정보를 저장하는 GraphType 구조체를 선언한다.

구조체 안에는 정점의 개수를 나타내는 n과 인접 행렬의 정보가 저장되어 있는 정수형 이중포인터 adj\_mat변수가 있다.

3. 정점이 방문되었는지 확인하는 변수인 visited를 전역변수로 선언한다.

visited변수는 배열로 선언하지 않고 메모리를 아끼기 위해 그래프 초기화 함수에서 동적할당하여 사용한다.

```
void init(GraphType*g, int num); //그래프 초기화 함수
void insert_vertex(GraphType*g, int num); //정점 삽입 함수
void insert_edge(GraphType* g, int start, int end); //간선 삽입 함수
void dfs_mat(GraphType*g, int v); //깊이 우선 탐색 함수
void Delete_Graph(GraphType*g); //그래프 삭제 함수
```

4. 필요한 함수들을 선언한다.

- init함수는 그래프의 변수들을 동적할당하고, 초기화하는 변수이다.
- insert\_vertex함수는 인접 행렬의 정점을 삽입하는 함수이다.
- insert\_edge함수는 그래프에 간선을 삽입하는 함수이다.
- dfs\_mat함수는 그래프를 깊이 우선 탐색하여 출력하는 함수이다.
- Delete\_Graph함수는 그래프의 정보를 해제하는 함수이다.

```

int main() {
    FILE *fp;
    int max = INT_MIN; //max에 int의 가장 작은 값 삽입
    char ctemp; //문자를 입력받는 임시변수
    int itemp1, itemp2, i; //정수를 입력받는 임시변수 2개
    GraphType *g; //그래프 변수
    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) { //파일끝까지 반복하며 최대값을 찾음
        fscanf(fp, "%c", &ctemp);
        if (ctemp == 'v') { //v를 입력받았을 때 정수 하나 더 입력받음
            fscanf(fp, "%d", &itemp1);
            if (max < itemp1) max = itemp1; //입력받은 정수가 max보다 클때 max에 값을 삽입
        }
    }

    g = (GraphType*)malloc(sizeof(GraphType));
    max++; //max를 1 증가시키고 초기화해야함
    init(g, max); //그래프 초기화

    for (i = 0; i < max; i++) { //max값만큼 정점삽입
        insert_vertex(g, max); //간선 삽입 함수호출
    }
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%c", &ctemp);
        if (ctemp == 'e') { //e를 입력받았을 때 간선 삽입
            fscanf(fp, "%d%d", &itemp1, &itemp2); //간선 두 정수를 입력받고
            insert_edge(g, itemp1, itemp2); //방향 그래프로 삽입
        }
    }
    printf("-깊이 우선 그래프 탐색 결과-\n");

    dfs_mat(g, 0); //깊이 우선탐색함수 호출
    printf(">\n");
    Delete_Graph(g); //그래프 삭제
    fclose(fp); //파일 닫음
    return 0;
}

```

5. 필요한 변수들을 선언한다.

- fp는 파일포인터, max는 데이터 파일에서 가장 큰 정점을 저장한다.
- ctemp는 문자 임시변수, itemp1, itemp2는 정수를 입력받는 임시변수이다.
- g는 그래프를 나타내는 변수이다.

6. data파일을 읽기 형식으로 open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 임시변수 ctemp로 입력받은 데이터가 v인지, e인지 확인하고 v이면 정점임을 나타내므로 정점을 다시 입력받고 그 중에 가장 큰 값을 찾아 max에 저장한다.

7. 파일을 모두 입력받았다면 그래프 변수 g를 동적할당하여 생성하고 max값을 1증가시킨다. max를 증가시키는 이유는 만약 최대값이 7이라면 그래프를 초기화할 때 8로 동적할당해야 값을 모두 넣을 수 있기 때문이다.

8. 값을 1 증가시킨 max값으로 init함수를 호출하여 그래프를 초기화 한다.  
그리고 max값만큼 for문을 이용해 반복하며 정점을 삽입한다. 여기서 정점개수는 최대 개수인 max를 넘을 수 없으므로 이를 매개변수로 전달한다.

9. rewind함수를 이용하여 파일포인터를 다시 처음으로 되돌리고, 간선을 삽입한다. ctemp에 데이터를 입력받는다. 만약 e라면 간선을 삽입하는 동작이므로 시작 정점과 끝 정점을 itemp에 입력받은 후 값을 insert\_edge함수로 간선을 삽입한다.

10. 모든 간선이 입력되었다면 깊이 우선 탐색으로 정점을 호출한다.  
dfs\_mat함수를 호출하여 정점 0부터 깊이 우선 탐색을 수행한다.

11. 모든 출력이 완료 되었다면 동적할당을 모두 해제할 수 있도록 한다.  
Delete\_Graph함수를 이용하여 동적할당한 그래프의 정보를 모두 해제하도록 한다. 그리고 fclose로 파일포인터를 닫고 프로그램을 종료한다.

```
void init(GraphType*g, int num) { //그래프 초기화 함수
    int r;
    g->n = 0; //정점의 개수를 0으로 초기화
    g->adj_mat = (int**)calloc(sizeof(int*), num); //calloc으로 행렬의 행 동적할당
    for (r = 0; r < num; r++) {
        g->adj_mat[r] = (int*)calloc(sizeof(int), num); //calloc으로 행렬의 열 동적할당
    }
    visited = (int*)calloc(sizeof(int), num); //visited변수를 num값에 맞게 동적할당
}
```

12. init함수는 그래프를 초기화하는 함수이다. 메인에서 구한 정점 개수인 max값을 매개변수로 전달받는다.

13. 정점 개수 n을 0으로 초기화 하고, num값에 맞게 행렬 adj\_mat을 동적할당하는데, 여기서 adj\_mat은 calloc을 이용하여 동적할당한다.  
calloc으로 동적할당하게 되면 무조건 0값으로 초기화가 되기 때문에 malloc으로 할당한 후 다시 0으로 초기화하는 동작을 줄일 수 있다.  
calloc은 malloc에서 \*로 표현하는 부분을 ,으로 표현하여 동적할당한다.

14. 방문되었는지 확인하는 변수인 visited함수 또한 포인터로 선언되었기 때문에 초기화 함수에서 num값에 맞게 동적할당 해준다.



```

void insert_vertex(GraphType* g, int num) {    //정점 삽입함수
    if ((g->n) + 1 > num) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;    //정점의 개수 증가
}

void insert_edge(GraphType* g, int start, int end) {    //방향그래프로 삽입하는 함수
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;    //입력받은 start행 end열을 1을 삽입
}

```

15. insert\_vertex함수는 그래프에 정점을 삽입하는 함수이다.

만약 삽입한 후 정점의 개수인 g->n+1이 메인에서의 정점 개수 num값보다 크면 함수를 종료한다. 여기서 num값은 메인에서 구한 max값을 나타낸다. 삽입이 끝나면 정점의 개수인 g->n을 증가시킨다.

16. insert\_edge함수는 방향그래프로 간선을 삽입하는 함수이다. 매개변수로는 그래프 g와 시작 인덱스인 start와 끝 인덱스인 end를 전달받는다.

인덱스를 먼저 정점의 개수 g->n과 비교한다. 만약 정점의 개수보다 인덱스의 값이 작다면 행렬에 값을 삽입한다.

이 함수는 방향 그래프로 삽입하기 때문에 한쪽 방향만 삽입하고 반대쪽은 삽입하지 않는다. 그렇기 때문에 start행 end열의 값에만 1을 삽입한다.

```

void dfs_mat(GraphType* g, int v) {    //깊이 우선탐색 함수
    int w;

    visited[v] = TRUE;    //현재 정점을 방문했다고 표시
    printf("%d ", v);
    for (w = 0; w < g->n; w++) {    //정점의 개수만큼 반복
        if (g->adj_mat[v][w] && !visited[w])    //adj_mat[v][w]가 1이고 visited[w]가 방문되지 않았다면
            dfs_mat(g, w);    //순환을 통해 정점 w에서 새로 시작
    }
}

```

17. dfs\_mat함수는 깊이 우선 탐색을 이용하여 그래프의 정점을 탐색하는 함수이다. 매개변수로는 그래프 변수와 시작 정점을 전달받는다.

먼저 visited변수에 현재 정점을 방문했다는 의미로 TRUE를 삽입하고 그 정점을 출력한다.

18. 그리고 정점의 개수만큼 for문을 이용해 반복하고 만약 현재 정점에서 가는 길이 존재하고(g->adj\_mat[v][w]==1) 그 정점이 방문되지 않은 정점이라면(!visited[w]) 그 w정점에서 dfs\_mat함수를 다시 호출하여 순환을 이용하여 다시 함수를 수행한다.



```

void Delete_Graph(GraphType*g) {           //그래프를 삭제하는 함수
    int i = 0;
    for (i = 0; i < g->n; i++) {           //행렬의 열 메모리 해제
        free(g->adj_mat[i]);
    }
    free(g->adj_mat);                       //행렬의 행 메모리해제
    free(g);                               //그래프 메모리 해제
    free(visited);                         //visited변수 메모리 해제
}

```

19. Delete함수는 그래프의 포인터를 모두 메모리 해제하는 함수이다.  
 먼저 그래프 정점만큼 반복하며 열값 메모리를 해제한 후 행도 해제해준다.  
 그래프 g를 메모리 해제하고 방문되었는지 확인하는 변수인 visited를 메모리 해제하면 Delete가 끝난다.

## 1.4 실행 창

### - 정점 0부터 깊이우선 탐색 결과

```
-깊이 우선 그래프 탐색 결과-
<0 1 4 3 5 2 7 >

C:\Users\wdmk46\OneDrive\바탕 화면\2학기 수업자료\자료구조2\실습\week9_1\Deb
인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

data - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

v 0 v 1 v 2 v 3 v 4 v 5 v 7
e 0 1 e 0 2 e 0 3 e 1 0 e 1 4 e 1 5
e 2 0 e 2 5 e 2 7 e 3 0 e 3 4 e 4 1
e 4 3 e 5 1 e 5 2 e 5 7 e 7 2 e 7 5
```

### - 정점 7부터 깊이우선 탐색 결과

```
-깊이 우선 그래프 탐색 결과-
<7 2 0 1 4 3 5 >

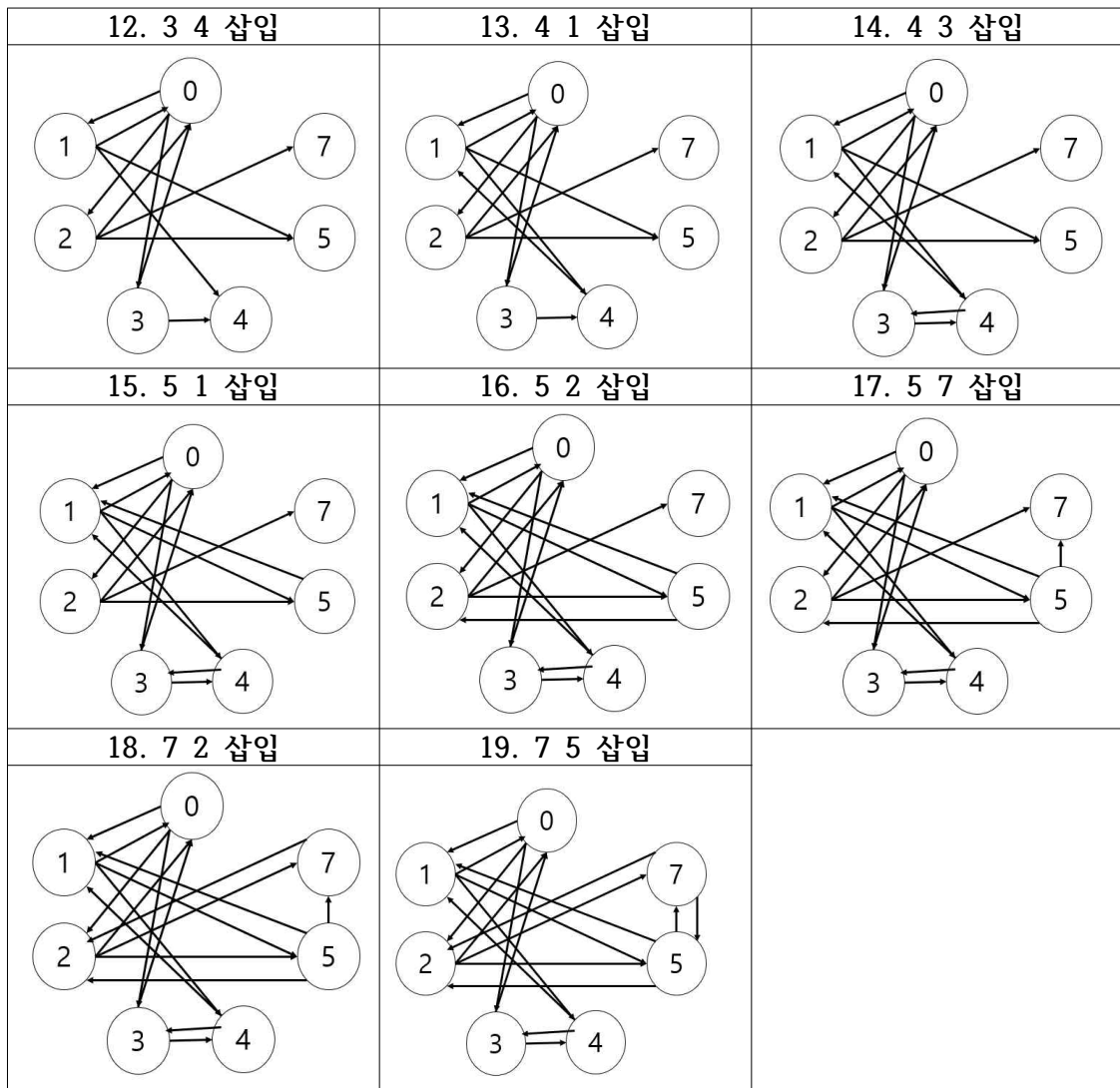
C:\Users\wdmk46\OneDrive\바탕 화면\2학기 수업자료\자료구조2\실습\week9_
인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

data - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

v 0 v 1 v 2 v 3 v 4 v 5 v 7
e 0 1 e 0 2 e 0 3 e 1 0 e 1 4 e 1 5
e 2 0 e 2 5 e 2 7 e 3 0 e 3 4 e 4 1
e 4 3 e 5 1 e 5 2 e 5 7 e 7 2 e 7 5
```

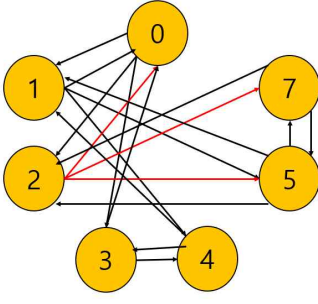
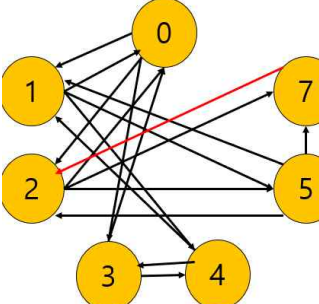
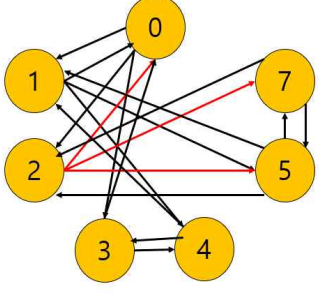
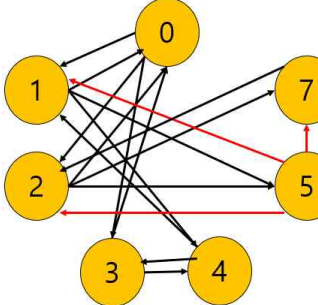
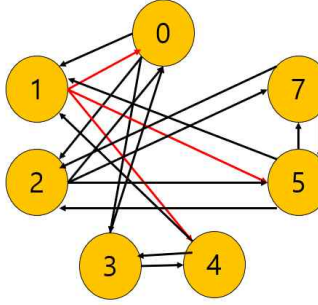
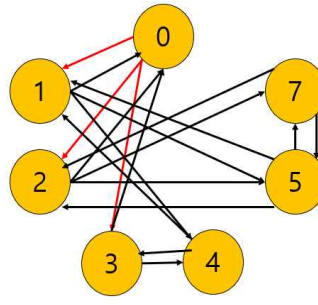
그래프가 그려지는 과정

데이터 파일	1. 정점 삽입	2. 0 1 삽입
<div>v0v1v2v3v4v5v7 e01 e02e03e10e14e15 e20e25e27e30e34e41 e43e51e52e57e72e75</div>		
3. 0 2 삽입	4. 0 3 삽입	5. 1 0 삽입
6. 1 4 삽입	7. 1 5 삽입	8. 2 0 삽입
9. 2 5 삽입	10. 2 7 삽입	11. 3 0 삽입



## 깊이를 탐색하는 순서

그래프 원본	1. 정점 0 시작	2. 0에서 1 방문
3. 1에서 4 방문	4. 4에서 갈 수 있는 1이 이미 방문된 상태 -> 3 방문	5. 3에서 갈 수 있는 0, 4가 이미 방문된 상태 -> 4로 돌아옴
6. 4에서 갈 수 있는 1과 3이 이미 방문된 상태 -> 1로 돌아옴	7. 1에서 0, 4 차례로 검사-> 이미 방문되어 있음 5 방문	8. 1은 이미 방문되어있음 -> 2 방문

<p>9. 2에서 갈 수 있는 0과 5가 이미 방문되어있음 -&gt; 7 방문</p>	<p>10. 7에서 갈 수 있는 2, 5가 이미 방문되어 있음 -&gt; 2로 돌아옴</p>	<p>11. 2에서 갈 수 있는 0, 7, 5가 이미 방문됨 -&gt; 5로 돌아옴</p>
		
<p>12. 5에서 갈 수 있는 1, 2, 7이 이미 방문됨 -&gt; 1로 돌아옴</p>	<p>13. 1에서 갈 수 있는 0, 4, 5가 이미 방문됨 -&gt; 0으로 돌아옴</p>	<p>14. 0에서 갈 수 있는 1, 2, 3이 이미 방문됨 -&gt; 더 이상 방문할 정점이 없으므로 함수 종료</p>
		

## 1.5 느낀 점

이번 문제는 저번 주에 했던 그래프 출력에서 더 나아가 깊이 우선 탐색 알고리즘을 사용하여 그래프를 탐색하는 문제였습니다. 처음 깊이 우선 탐색을 배울 때는 잘 이해가 되지 않았는데 이번에 프로그래밍을 직접 해보며 어떤 방식으로 진행되는지 잘 이해할 수 있었던 것 같습니다.

프로그래밍을 하며 코드의 진행 과정을 직접 그려보며 하나하나 확인해보았습니다. 그 결과 깊이 우선 탐색을 더 쉽게 이해할 수 있었고 진행과정도 파악할 수 있었습니다.

이번 프로그램의 경우는 순환을 이용하여 코드를 작성했지만 스택을 이용하여 코드를 작성하는 것이 더 빠르다고 교수님께 배웠기 때문에 다음에 기회가 생긴다면 스택을 이용하여 프로그래밍을 해보고 싶다고 생각했습니다.

또한 이번 코드의 경우는 그래프를 인접 행렬을 이용하여 프로그래밍을 했지만 앞으로는 그래프를 리스트로도 만들어서 테스트 해보고 싶다고 생각했습니다.

시험 때도 깊이 우선 탐색에 대해 공부했지만 그때 잘 이해가 되지 않았던 것 같습니다. 하지만 이번 기회로 확실히 이해할 수 있었고 앞으로 깊이 우선 탐색이 필요할 때 바로 사용할 수 있을 것입니다.



## 2. 그래프

### 너비 우선 탐색 프로그램

---

- data.txt에서 데이터를 읽어와 프로그램을 작성하시오.  
이때 v는 정점, e는 간선을 의미한다.

#### 2.1 문제 분석

조건 1 그래프가 그려지는 과정과 함께 너비를 탐색하는 순서를 그림

조건 2 그래프에 대해서 data.txt파일들을 직접 생성하여 입력받음

조건 3 그래프 간선 정보를 행렬에 저장

- 이 문제는 너비 우선 탐색 알고리즘을 이용하여 그래프의 정점을 탐색하는 프로그램이다. 너비를 이용한 우선 탐색은 큐를 이용하여 프로그래밍한다. 그래프는  $n \times n$  행렬을 이용하여 인접행렬 그래프로 만들고, 행렬의 크기는 메모리의 크기를 아낄 수 있도록 포인터를 동적할당하여 사용한다. 너비 우선 탐색은 그래프의 시작 정점에서 출발하여 시작정점을 방문하였다고 표시한 후 큐에 그 정점을 삽입한다. 그리고 큐의 가장 처음 값을 dequeue한 후에 그 정점에서 갈 수 있는 길을 찾고 그 정점이 방문되어 있지 않다면 출력한 후 큐에 삽입한다. 만약 인접 정점이 모두 방문 되었다면 프로그램을 종료한다. 너비 우선 탐색에서 방문 여부를 기록하기 위해 배열을 사용하며, 모든 정점의 배열 값은 FALSE로 초기화 되고 정점이 방문될 때마다 정점 방문 배열값은 TRUE로 변경된다. 이번 문제의 경우 메모리를 아끼기 위해 방문 변수를 포인터로 선언하여 크기에 맞게 동적할당하여 사용하고 큐 배열 또한 포인터로 선언하여 초기화 함수에서 동적할당한다.

## 2.2 소스 코드

```

/*
 * 학번 : 20154812
 * 학과 : 컴퓨터소프트웨어공학과
 * 이름 : 김동민
 * 과업명: 그래프의 N개 주석 삽입 프로그램
 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

typedef struct {
    int Qsize;           //큐의 크기
    int Qvalue;          //큐 배열
    int front, rear;     //큐의 처음과 끝
} QueueType;

void error(char*message) { //에러 출력 함수
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void queue_init(QueueType *q, int num) { //큐 초기화 함수
    q->Qsize = num;           //큐의 크기에 num값을
    q->Qvalue = (int*)calloc(sizeof(int)*num); //큐를 num값으로 할당할함
    q->front = q->rear = 0;    //front와 rear를 0으로 설정
}

int is_empty(QueueType *q) { //큐가 비었는지 확인하는 함수
    return (q->front == q->rear); //앞표면 1, 뒤표면 0 반환
}

int is_full(QueueType *q) { //큐가 모두 차있는지 확인
    return ((q->rear + 1) % q->Qsize == q->front); //rear+1을 큐의 크기로 modulo연산
}

void enqueue(QueueType *q, int item) { //큐에 삽입하는 함수
    if (is_full(q)) {
        error("큐가 포화상태입니다.");
    }
    q->rear = (q->rear + 1) % q->Qsize; //rear+1을 modulo연산하여 삽입
    q->Qvalue[q->rear] = item;
}

int dequeue(QueueType *q) { //큐의 가장 앞을 반환하는 함수
    if (is_empty(q)) {
        error("큐가 공백상태입니다.");
    }
    q->front = (q->front + 1) % q->Qsize; //front+1을 modulo연산하여 반환
    return q->Qvalue[q->front];
}

typedef struct {
    int n;           //그래프의 노드개수
    int head_node;   //이동노드의 할당
} GraphType;

int is_visited; //방문이 완료되었는지 확인하는 변수

void init(GraphType *g, int num) { //그래프를 초기화하는 함수
    void insert_vertex(GraphType *g, int num); //정점 삽입함수
    void insert_edge(GraphType *g, int start, int end); //간선삽입함수
    void bfs_visit(GraphType *g, int v, int); //DFS 주석 삽입함수
    void Delete_Graph(GraphType *g); //그래프 삭제 함수
}

int main() {
    FILE *fo;
    int max = INT_MIN; //rear에 int의 가장 작은 값 삽입
    char chano;        //문자를 입력받은 일시변수
    int item1, item2;   //정수를 입력받은 일시변수2개
    GraphType *g;      //그래프 변수
    fo = fopen("data.txt", "r");
    if (!fo) {
        printf("File not open");
        return 0;
    }
    while (!feof(fo)) { //파일 끝까지 반복하여 최대값을 찾음
        fscanf(fo, "%c", &chano);
        if (chano == '\n') { //\n을 입력받았을 때 정수 하나 더 입력받음
            fscanf(fo, "%d", &item1);
            if (max < item1) max = item1; //입력받은 정수가 max보다 클지 max에 값을 삽입
        }
    }
    g = (GraphType*)calloc(sizeof(GraphType));
    max++;
    init(g, max); //max를 1 증가시키고 그래프 초기화
    for (i=0; i<max; i++){
        insert_vertex(g, max); //그래프로 간선을 삽입
    }
    rewind(fo);
    while (!feof(fo)) {
        fscanf(fo, "%c", &chano);
        if (chano == '\n') { //\n을 입력받았을 때 간선 삽입
            fscanf(fo, "%d", &item1, &item2); //정수 2개를 입력받고
            insert_edge(g, item1, item2); //두정점의 그래프로 삽입
        }
    }
    printf("\nN개 주석 삽입 : ");

    bfs_visit(g, 0, max); //DFS 주석 삽입 함수 호출
    Delete_Graph(g); //그래프 삭제 함수 호출
    fclose(fo); //파일포인터의 닫힘
    return 0;
}

```

```

//그래프를 초기화 하는 함수
void init(GraphType* g, int num) {
    int v;
    vnum = 0;
    adj_mat = (int**)calloc(sizeof(int*), num); //calloc을 이용하여 행 초기화
    for (v = 0; v < num; v++) {
        adj_mat[v] = (int*)calloc(sizeof(int), num); //calloc을 이용하여 열 초기화
    }
    visited = (int*)calloc(sizeof(int), num); //visited변수를 num값에 맞게 초기화
}

//그래프에 정점을 삽입하는 함수
void insert_vertex(GraphType* g, int num) { //정점을 삽입하는 함수
    if ((vnum) + 1 > num) {
        fprintf(stderr, "그래프 정점의 개수 초과");
        return;
    }
    vnum++; //정점의 개수 증가
}

//두 정점 간에 간선을 삽입하는 함수
void insert_edge(GraphType* g, int start, int end) { //두방향 그래프로 간선을 삽입하는 함수
    if (start > vnum || end > vnum) {
        fprintf(stderr, "그래프 정점 번호 오류");
        return;
    }
    adj_mat[start][end] = 1; //두방향 그래프이기 때문에 start와 end를
    adj_mat[end][start] = 1; //end와 start를 둘 다 기록
}

//bfs를 수행하는 함수
void bfs(GraphType* g, int s, int num) { //bfs를 수행하는 함수
    int u;
    QueueType q; //큐 선언
    queue_init(&q, num); //큐 초기화
    visited[s] = TRUE;
    printf("%d ", s);
    enqueue(&q, s); //우에 현재 정점 삽입
    while (!is_empty(&q)) { //큐가 빌 때까지 반복
        u = dequeue(&q); //큐 가장 앞에 있는 값을 dequeue
        for (v = 0; v < num; v++) {
            if (adj_mat[u][v] && !visited[v]) { //정점으로 가는 길이 있고 방문하지 않은 정점이라면
                visited[v] = TRUE; //현재 정점을 방문하였다고 표시
                printf("%d ", v); //정점을 출력
                enqueue(&q, v); //정점을 큐에 삽입
            }
        }
    }
    printf("\n");
}

//그래프를 삭제하는 함수
void Delete_Graph(GraphType* g) {
    int i = 0;
    for (i = 0; i < vnum; i++) { //그래프의 열 메모리 해제
        free(adj_mat[i]);
    }
    free(adj_mat); //그래프의 행 메모리 해제
    free(q); //그래프 메모리 해제
    free(visited); //visited변수 메모리 해제
}

```

## 2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 그래프 너비 우선 탐색 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

typedef struct {
    int Qsize;           //큐의 크기
    int *queue;          //큐 배열
    int front, rear;     //큐의 처음과 끝
}QueueType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 참을 나타내는 TRUE와 거짓을 나타내는 FALSE를 정의한다.

2. 큐를 나타내는 QueueType 구조체를 선언한다.

구조체 안에는 큐의 크기인 Qsize와 동적할당으로 나타낼 queue포인터 큐의 처음과 끝 인덱스를 나타내는 front, rear변수가 있다.

```
void error(char*message) { //에러 출력 함수
    fprintf(stderr, "%s\n", message);
    exit(1);
}

void queue_init(QueueType *q, int num) { //큐 초기화 함수
    q->Qsize = num; //큐의 크기에 num삽입
    q->queue = (int*)malloc(sizeof(int)*num); //큐를 num값으로 동적할당
    q->front = q->rear = 0; //front와 rear를 0으로 설정
}

int is_empty(QueueType *q) { //큐가 비었는지 확인하는 함수
    return (q->front == q->rear); //같으면 1, 아니면 0 반환
}

int is_full(QueueType *q) { //큐가 모두 차있는지 확인
    return ((q->rear + 1) % q->Qsize == q->front); //rear+1을 큐의 크기로 modulo연산
```

3. error함수는 에러를 출력하는 함수이다. 에러로 출력될 문자열을 전달받고, fprintf를 이용하여 에러 내용을 출력한 후 exit함수로 프로그램을 종료한다.

4. queue\_init함수는 큐 변수들을 초기화하는 함수이다.

큐의 크기를 num값으로 저장하고 queue포인터를 num값에 맞게 동적할당한 후 front와 rear값을 0으로 설정한다. 큐에 최대 들어올 수 있는 크기는 정점의 최대 크기를 넘을 수 없기 때문에 num값으로 동적할당한다.

5. is\_empty함수는 큐가 비었는지 확인하는 함수이다. front==rear를 리턴하는데, 만약 front와 rear가 같다면 1을 반환하고 다르다면 0을 반환하게 될 것이다.

6. is\_full함수는 큐가 포화상태인지 확인하는 함수이다.

원형 큐 알고리즘을 사용하여 프로그래밍했기 때문에 modulo연산으로 위치를 확인한다. rear의 다음 배열 값을 크기로 나눈 나머지의 값이 front와 같다면 포화상태이므로 1을 반환하고 다르다면 0을 반환한다.

```
void enqueue(QueueType *q, int item) { //큐에 삽입하는 함수
    if (is_full(q)) {
        error("큐가 포화상태입니다.");
    }
    q->rear = (q->rear + 1) % q->Qsize; //rear+1을 modulo연산하여 삽입
    q->queue[q->rear] = item;
}

int dequeue(QueueType*q) { //큐의 가장 앞을 반환하는 함수
    if (is_empty(q))
        error("큐가 공백상태입니다.");
    q->front = (q->front + 1) % q->Qsize; //front+1을 modulo연산하여 반환
    return q->queue[q->front];
}
```

7. enqueue함수는 큐에 데이터를 삽입하는 함수이다.

먼저 is\_full함수를 통해 큐가 포화상태인지 확인한 후, 큐에 자리가 있다면 삽입을 한다. 먼저 rear값을 modulo연산을 통해 배열의 다음 위치로 옮겨가도록 하고 그 자리에 데이터를 삽입한다.

8. dequeue함수는 front위치의 하나 앞을 반환하는 함수이다.

먼저 큐가 공백상태인지 확인하는 함수인 is\_empty로 확인한 후 큐가 공백상태가 아니라면 front에 1을 더하고 modulo연산을 통해 front의 위치를 지정한 후 그 자리에 있는 큐 값을 반환한다.

```
typedef struct { //그래프 구조체 선언
    int n; //정점의 개수
    int **adj_mat; //이중포인터 행렬
}GraphType;

int *visited; //정점이 방문되었는지 확인하는 변수

void init(GraphType*g, int num); //그래프를 초기화하는 함수
void insert_vertex(GraphType*g, int num); //정점 삽입함수
void insert_edge(GraphType* g, int start, int end); //간선삽입함수
void bfs_mat(GraphType*g, int v, int); //너비 우선 탐색함수
void Delete_Graph(GraphType*g); //그래프 삭제 함수
```



9. 큐에 관련된 함수 구현이 끝났으므로 그래프 구조체를 선언한다.

그래프의 정보를 저장하는 GraphType 구조체를 선언한다.

구조체 안에는 정점의 개수를 나타내는 n과 인접 행렬의 정보가 저장되어 있는 정수형 이중포인터 adj\_mat변수가 있다.

10. 정점이 방문되었는지 확인하는 변수인 visited를 전역변수로 선언한다.

visited변수는 배열로 선언하지 않고 메모리를 아끼기 위해 동적할당하여 사용한다.

11. 필요한 함수들을 선언한다.

- init함수는 그래프의 변수들을 동적할당하고, 초기화하는 변수이다.
- insert\_vertex함수는 인접 행렬의 정점을 삽입하는 함수이다.
- insert\_edge함수는 그래프에 간선을 삽입하는 함수이다.
- bfs\_mat함수는 그래프를 너비 우선 탐색하여 출력하는 함수이다.
- Delete\_Graph함수는 그래프의 정보를 해제하는 함수이다.

```
int main() {
    FILE *fp;
    int max = INT_MIN;           //max에 int의 가장 작은 값 삽입
    char ctemp;                 //문자를 입력받는 임시변수
    int itemp1, itemp2, i;       //정수를 입력받는 임시변수 2개
    GraphType *g;               //그래프 변수
    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {         //파일끝까지 반복하며 최대값을 찾음
        fscanf(fp, "%c", &ctemp);
        if (ctemp == 'v') {     //v를 입력받았을 때 정수 하나 더 입력받음
            fscanf(fp, "%d", &itemp1);
            if (max < itemp1) max = itemp1; //입력받은 정수가 max보다 클 때 max에 값을 삽입
        }
    }
    g = (GraphType*)malloc(sizeof(GraphType));
    max++;
    init(g, max);               //max를 1 증가시키고 그래프 초기화
    for(i=0; i<max; i++){
        insert_vertex(g, max);  //그래프에 간선을 삽입
    }
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%c", &ctemp);
        if (ctemp == 'e') {     //e를 입력받았을 때 간선 삽입
            fscanf(fp, "%d%d", &itemp1, &itemp2); //정수 2개를 입력받고
            insert_edge(g, itemp1, itemp2); //무방향 그래프에 삽입
        }
    }
    printf("너비 우선 탐색 : ");

    bfs_mat(g, 0, max);         //너비 우선 탐색 함수 호출
    Delete_Graph(g);            //그래프 삭제 함수
    fclose(fp);                 //파일 포인터 닫음
    return 0;
}
```

12. 필요한 변수들을 선언한다.

- fp는 파일포인터, max는 데이터 파일에서 가장 큰 정점을 저장한다.
- ctemp는 문자 임시변수, itemp1, itemp2는 정수를 입력받는 임시변수이다.
- g는 그래프를 나타내는 변수이다.

max는 가장 큰 정점을 찾아야 하므로 초기값은 정수의 가장 작은 값을 삽입한다.

13. data파일을 읽기 형식으로 open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 임시변수 ctemp로 입력받은 데이터가 v인지, e인지 확인하고 v이면 정점임을 나타내므로 정점을 다시 입력받고 그 중에 가장 큰 값을 찾아 max에 저장한다.

14. 파일을 모두 입력받았다면 그래프 변수 g를 동적할당하여 생성하고 max값을 1증가시킨다. max를 증가시키는 이유는 만약 최대값이 7이라면 그래프를 초기화할 때 8로 동적할당해야 값을 모두 넣을 수 있기 때문이다.

15. 값을 증가시킨 max값으로 init함수를 호출하여 그래프를 초기화 한다. 그리고 max값만큼 for문을 이용해 반복하며 정점을 삽입한다. 여기서 정점 개수는 최대 개수인 max를 넘을 수 없으므로 이를 매개변수로 전달한다.

16. rewind함수를 이용하여 파일포인터를 다시 처음으로 되돌리고, 간선을 삽입한다. ctemp에 데이터를 입력받는다. 만약 e라면 간선을 삽입하는 동작이므로 시작 정점과 끝 정점을 itemp에 입력받은 후 값을 insert\_edge함수로 간선을 삽입한다.

17. 모든 간선이 입력되었다면 너비 우선 탐색으로 정점을 호출한다.

bfs\_mat함수를 호출하여 정점 0부터 너비 우선 탐색을 수행한다.

여기서 큐를 초기화할 때 max값이 필요하므로 이를 매개변수로 전달한다.

18. 모든 출력이 완료 되었다면 동적할당을 모두 해제할 수 있도록 한다. Delete\_Graph함수를 이용하여 동적할당한 그래프의 정보를 모두 해제하도록 한다. 그리고 fclose로 파일포인터를 닫고 프로그램을 종료한다.



```

void init(GraphType* g, int num) { //그래프를 초기화 하는 함수
    int r;
    g->n = 0;
    g->adj_mat = (int**)calloc(sizeof(int*), num); //calloc을 이용하여 행 초기화
    for (r = 0; r < num; r++) {
        g->adj_mat[r] = (int*)calloc(sizeof(int), num); //calloc을 이용하여 열 초기화
    }
    visited = (int*)calloc(sizeof(int), num); //visited변수를 num값에 맞게 초기화
}

```

19. init함수는 그래프를 초기화하는 함수이다. 메인에서 구한 정점 개수인 max값을 매개변수로 전달받는다.

20. 정점 개수 n을 0으로 초기화 하고, num값에 맞게 행렬 adj\_mat을 동적할당하는데, 여기서 adj\_mat은 calloc을 이용하여 동적할당한다.

calloc으로 동적할당하게 되면 무조건 0값으로 초기화가 되기 때문에 malloc으로 할당한 후 다시 0으로 초기화하는 동작을 줄일 수 있다.

calloc은 malloc에서 \*로 표현하는 부분을 ,으로 표현하여 동적할당한다.

21. 방문되었는지 확인하는 변수인 visited함수 또한 포인터로 선언되었기 때문에 초기화 함수에서 num값에 맞게 동적할당 해준다.

```

void insert_vertex(GraphType* g, int num) { //정점을 삽입하는 함수
    if ((g->n) + 1 > num) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++; //정점의 개수 증가
}

void insert_edge(GraphType* g, int start, int end) { //무방향 그래프로 간선을 삽입하는 함수
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1; //무방향 그래프이기 때문에 start행 end열
    g->adj_mat[end][start] = 1; //end행 start열 둘 다 저장
}

```

22. insert\_vertex함수는 그래프에 정점을 삽입하는 함수이다.

만약 삽입한 후 정점의 개수인 g->n+1이 메인에서의 정점 개수 num값보다 크면 함수를 종료한다. 여기서 num값은 메인에서 구한 max값을 나타낸다. 삽입이 끝나면 정점의 개수인 g->n을 증가시킨다.

23. insert\_edge함수는 방향그래프로 간선을 삽입하는 함수이다. 매개변수로는 그래프 g와 시작 인덱스인 start와 끝 인덱스인 end를 전달받는다.

인덱스를 먼저 정점의 개수 g->n과 비교한다. 만약 정점의 개수보다 인덱스의 값이 작다면 행렬에 값을 삽입한다.

이 함수는 무방향 그래프로 삽입하기 때문에 두 방향 모두 삽입해야 한다. 그렇기 때문에 start행 end열의 값과 end행 start열 둘다 1을 삽입한다.

```

void bfs_mat(GraphType*g, int v, int num) { //너비 우선 탐색 함수
    int w;
    QueueType q; //큐 선언

    queue_init(&q, num); //큐 초기화
    visited[v] = TRUE;
    printf("%d ", v);
    enqueue(&q, v); //큐에 현재 정점 삽입
    while (!is_empty(&q)) { //큐가 빌때까지 반복
        v = dequeue(&q); //큐 가장 앞에 있는 값을 dequeue
        for (w = 0; w < g->n; w++) {
            if (g->adj_mat[v][w] && !visited[w]) { //정점으로 가는 길이 있고 방문되어 있지 않다면
                visited[w] = TRUE; //현재 정점을 방문되었다고 표시
                printf("%d ", w); //정점을 출력
                enqueue(&q, w); //정점을 큐에 삽입
            }
        }
    }
    printf("\n");
}

```

24. bfs\_mat함수는 너비우선탐색을 이용하여 그래프를 탐색하는 함수이다.  
너비 우선 탐색을 위해선 큐 알고리즘이 필요하므로 큐를 선언한다.

25. 먼저 queue\_init큐 초기화 함수를 통해 큐를 초기화 한다. 이때 메인에서 전달받은 max값을 전달하여 큐 포인터가 동적할당될 수 있도록 한다.  
현재 정점이 방문되었다고 표시한 후 출력하고 그 정점을 enqueue로 큐에 삽입한다.

26. while반복문을 이용하여 큐가 공백상태가 될 때까지 반복한다.  
dequeue를 이용하여 큐로부터 가장 앞에 있는 데이터를 반환받고 정점의 개수만큼 for문으로 반복한다.  
만약 dequeue로 반환받은 v에서 w로 가는 길이 있고 그 정점이 방문되지 않았다면 정점을 방문되었다고 표시한 후 출력하고 큐에 정점을 삽입한다.

```

void Delete_Graph(GraphType*g) { //그래프를 삭제하는 함수
    int i = 0;
    for (i = 0; i < g->n; i++) { //그래프의 열 메모리 해제
        free(g->adj_mat[i]);
    }
    free(g->adj_mat); //그래프의 행 메모리 해제
    free(g); //그래프 메모리 해제
    free(visited); //visited변수 메모리 해제
}

```

27. Delete함수는 그래프의 포인터를 모두 메모리 해제하는 함수이다.  
먼저 그래프 정점만큼 반복하며 열값 메모리를 해제한 후 행도 해제해준다.  
그래프 g를 메모리 해제하고 방문되었는지 확인하는 변수인 visited를 메모리 해제하면 Delete가 끝난다.

## 2.4 실행 창

### -0번 정점부터 너비 우선 탐색

너비 우선 탐색 : 0 1 2 4 3

C:\Users\wdmk46\OneDrive\바탕 화면\2학기 수업자료\자료구  
인해 종료되었습니다.  
이 창을 닫으려면 아무 키나 누르세요.

data - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
v 0 v 1 v 2 v 3 v 4
e 3 4 e 2 4 e 2 3
e 1 2 e 0 4 e 0 2
e 0 1|
```

### -2번 정점부터 너비 우선 탐색

너비 우선 탐색 : 2 0 1 3 4

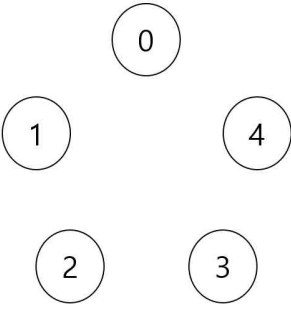
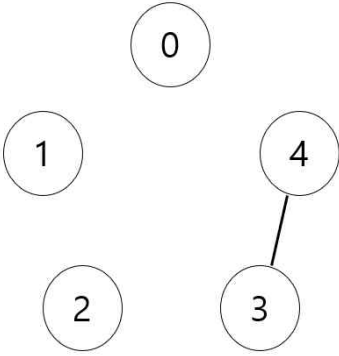
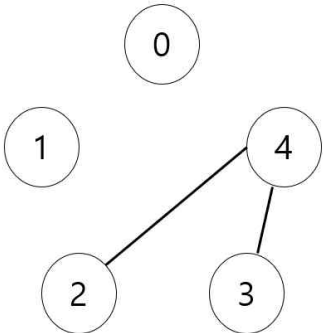
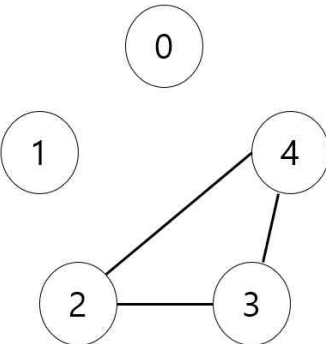
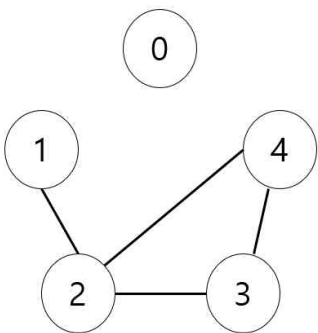
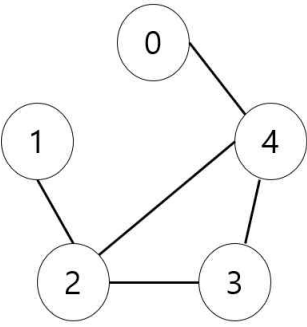
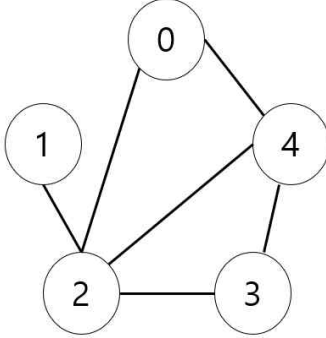
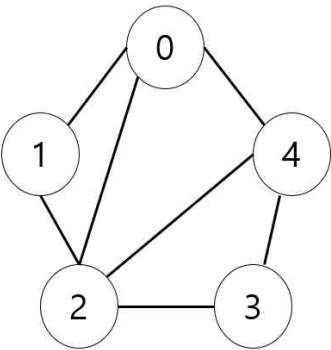
C:\Users\wdmk46\OneDrive\바탕 화면\2학기 수업자료\자료구조2\실습\week9\_2\De  
인해 종료되었습니다.  
이 창을 닫으려면 아무 키나 누르세요.

data - Windows 메모장

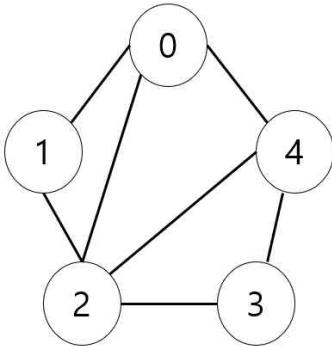
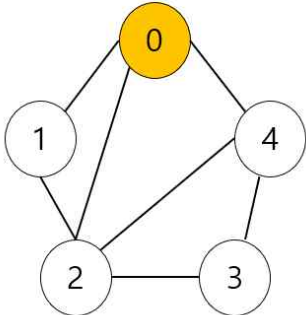

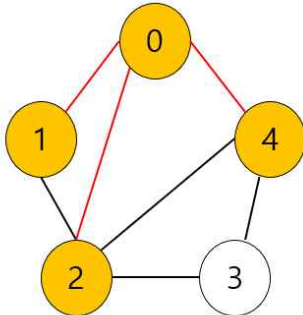

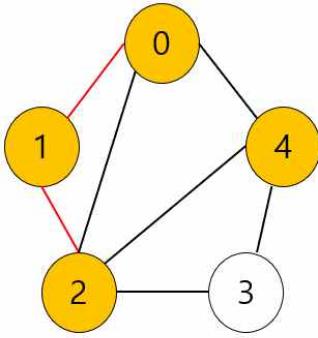

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

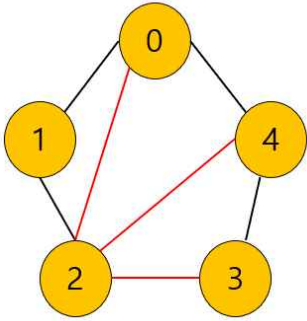

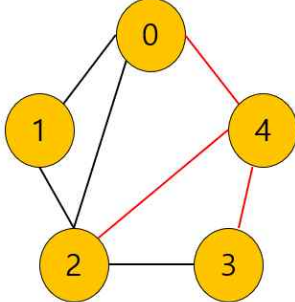

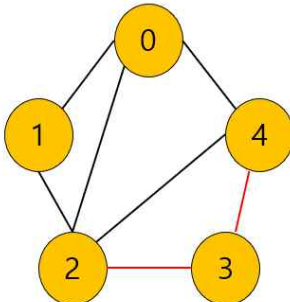

```
v 0 v 1 v 2 v 3 v 4
e 3 4 e 2 4 e 2 3
e 1 2 e 0 4 e 0 2
e 0 1|
```

그래프 생성 과정

데이터 파일 (무방향 그래프)	1. 정점 삽입	2. 3 4 삽입
<pre>v 0 v 1 v 2 v 3 v 4 e 3 4 e 2 4 e 2 3 e 1 2 e 0 4 e 0 2 e 0 1</pre>		
3. 2 4 삽입	4. 2 3 삽입	5. 1 2 삽입
		
6. 0 4 삽입	7. 0 2 삽입	8. 0 1 삽입
		

너비 우선 탐색 과정

기본 그래프 데이터(무방향 그래프)		
		
1. 0부터 너비 우선 탐색하기 때문에 0출력한 후 0을 원형 큐에 삽입한다.		
2. 0을 큐에서 dequeue한 후 0과 인접한 1, 2, 4를 차례로 큐에 삽입한다.		
3. 1을 큐에서 dequeue한 후 1에 인접한 정점을 확인한다. 0과 2가 이미 방문되었다고 표시가 되어 있으므로 큐에 삽입하지 않는다.		

<p>4. 2를 큐에서 dequeue한 후 2와 인접한 정점을 확인한다. 0과 4는 방문되었지만 3은 방문되어있지 않으므로 큐에 삽입한다.</p>		
<p>5. 4를 큐에서 dequeue한 후 4와 인접한 정점을 확인한다. 0, 2, 3 모두 방문 표시가 되어있으므로 큐에 삽입하지 않는다.</p>		
<p>6. 3을 큐에서 dequeue한 후 3과 인접한 정점을 확인한다. 2, 4 모두 방문 되었으므로 큐에 삽입하지 않는다. 큐가 empty로 공백상태가 되었으므로 반복을 종료하고 함수를 종료한다.</p>		

## 2.5 느낀 점

이번 문제는 저번 주에 했던 그래프 출력에서 더 나아가 너비 우선 탐색 알고리즘을 사용하여 그래프를 탐색하는 문제였습니다. 처음 너비 우선 탐색을 배울 때는 잘 이해가 되지 않았는데 이번에 프로그래밍을 직접 해보며 어떤 방식으로 진행되는지 잘 이해할 수 있었던 것 같습니다.

너비 우선 탐색은 큐를 이용하여 프로그래밍을 해야 했기 때문에 눈으로만 봐서는 이해가 잘 되지 않았습니다. 그래서 직접 큐의 변화 과정을 그림으로 그려서 확인해보니 확실히 그냥 프로그램으로 확인하는 것보다는 이해가 더 잘 되었던 것 같습니다.

또한 이번 코드의 경우는 그래프를 인접 행렬을 이용하여 프로그래밍을 했지만 나중에 시간이 된다면 그래프를 리스트로도 만들어서 테스트 해보고 싶다고 생각했습니다.

이번 과제를 통해 너비 우선 탐색에 대해 확실히 이해할 수 있는 기회가 되었고 앞으로 이와 관련된 문제가 나온다면 자신있게 풀 수 있을 것 같습니다.



### 3. 그래프를 하며 느낀 점

이번 문제는 저번 주에 했던 그래프에서 더 나아가 깊이 우선 탐색과 너비 우선 탐색 알고리즘에 대해 자세하게 공부할 수 있는 기회가 되었습니다. 이론 시간에 눈으로만 배웠을 때는 이해가 잘 되지 않던 부분들이 많았는데 이번 실습시간을 통해 코드를 프로그래밍해보고 직접 알고리즘의 진행을 그림으로 그려보니 이해가 되지 않던 부분도 쉽게 이해할 수 있었던 기회가 되었습니다.

특히 1학기 때 배웠던 스택과 큐를 다시 공부하니 옛날에 배운 것에서 더 나아가 공부할 수 있었고 이해도 쉽게 할 수 있었던 것 같습니다.

아직은 많이 부족하다고 생각되는 부분이 많지만 자료구조에 대해 더 열심히 노력해서 부족한 부분을 채울 수 있도록 하겠습니다.