자료구조2 실습

7주차 과제

제출일	21.10.26	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2 실습	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

| 목 차 |

1. 그래프

인접 행렬

- 372페이지의 프로그램 10.1을 이용하여 파일 data.txt에서 정점과 에지 정보를 입력받아 인접 행렬을 생성하여 그래프를 생성하는 프로그램을 작성하시오.

1.1 문제 분석

조건 1 data.txt파일의 데이터 형식은 에지정보(정점 정점)로 구성되어 있음
조건 2 행렬을 이용하여 그래프 정보 저장
조건 3 data.txt파일들을 직접 생성하여 입력받음

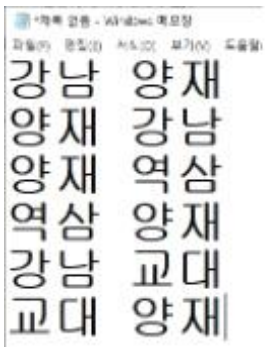
- 이 문제는 2차원 배열을 이용한 행렬로 간선을 나타내는 문제이다. 여기서 정점에 대한 정보는 정수로 표현하지 않고 문자열로 표현한다.

메모리를 아끼기 위해 정점에 대한 정보를 포인터로 선언하여 동적할당하여 저장한다. 정점의 정보를 알맞게 동적할당하고 입력받기 위해서는 파일을 입력받는 과정에서 정점의 개수를 정확히 파악해야 메모리 낭비 없이 동적할당을 할 수 있을 것이다. 이를 위해 중복 없이 파일의 개수의 *2로 정점을 저장하는 임시 문자열 배열을 만들고, 이 문자열 배열이 들어있는 값만큼 그래프에서 정점 배열을 만들어 정점을 삽입하면 될 것이다.

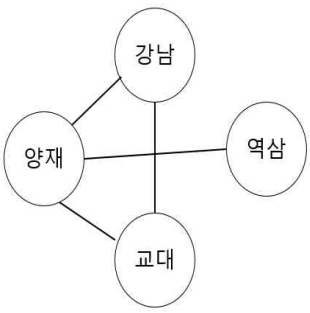
인접행렬로 나타낼 수 있는 그래프는 무방향 그래프와 방향 그래프가 있다. 이 두 개를 모두 나타내기 위해 그래프를 2개 생성하고 값을 삽입하여 각각 무방향 그래프와 방향 그래프로 만들어 본다. 그리고 그래프의 출력을 확인해보며 둘의 차이점을 확인해본다.

값을 출력할 때는 최대 3글자까지 입력받아도 행렬 형식이 깨지지 않도록 고려하여 출력한다.

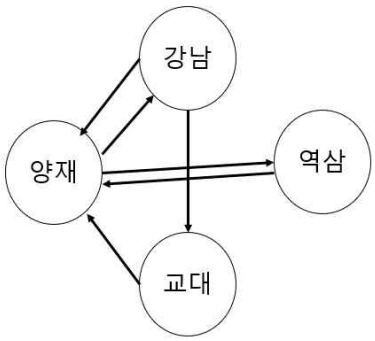
데이터 파일



무방향 그래프



방향 그래프



1.2 소스 코드

```
/*
    학번 : 20154812
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김홍빈
    파일 명 : 인접 행렬을 이용한 그래프 추상 데이터 타입의 구현
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
typedef struct Element {
    char start[10];                     //삽입 연산을 할 때 값을 전달
    char end[10];
}Element;
typedef struct GraphType {
    int n;                              //정점의 개수
    char **aru;                         //정점의 정보
    int **adj_mat;                      //그래프 배열
}GraphType;

void init(GraphType*g, int num, char**);            //그래프 초기화 함수
void insert_vertex(GraphType*g, int count);    //정점 삽입 연산 함수
void insert_undirected_edge(GraphType*g, Element data); //무방향그래프 삽입함수
void insert_directed_edge(GraphType*g, Element data);   //방향그래프 삽입함수
void print_adj_mat(GraphType*g);                    //행렬에 맞게 그래프를 출력하는 함수
int Check_Same_String(char** aru, int num, char *s);            //배열에 같은 정점이 있는지 확인하는 함수
void Delete(GraphType*g);                           //그래프의 포인터를 해제하는 함수

int main() {
    FILE *fp;                       //파일포인터
    Element data;                   //파일로부터 입력받은 변수
    Element *insert_data;           //파일에 있는 데이터를 저장
    int count = 0, num = 0;         //count는 파일에 있는 데이터의 개수, num은 정점의 개수
    char  **temp;                 //temp은 정점의 이름을 일시로 저장하는 문자열 포인터 변수
    GraphType * g, *g2;             //g는 무방향그래프, g2는 방향 그래프
    int i=0;
    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return;
    }
    while (!feof(fp)) {                         //파일을 끝까지 읽으며 데이터의 개수 증가
        fscanf(fp, "%s%s", data.start, data.end);
        count++;
    }
    g = (GraphType*)malloc(sizeof(GraphType));      //무방향 그래프 생성
    g2 = (GraphType*)malloc(sizeof(GraphType));     //방향 그래프 생성
    temp = (char**)malloc(sizeof(char*)*count);     //파일의 개수에 맞게 temp문자열 배열 할당
    insert_data = (Element*)malloc(sizeof(Element)*count);

    rewind(fp);                                 //파일포인터를 처음으로 되돌림
    while (!feof(fp)) {
        fscanf(fp, "%s%s", data.start, data.end);
        insert_data[i++] = data;                //파일 개수를 동적할당한 배열에 파일 데이터의 값 삽입
                                                //(파일포인터를 계속 되돌리고 입력받는 동작을 함)
        if (Check_Same_String(temp, num, data.start)) {     //temp에 같은 정점이 있는지 판단 같은 정점이 없다면 TRUE리턴
            temp[num] = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));   //start의 개수로 temp의 문자열을 동적할당하여 temp에 저장
            strcpy(temp[num++], data.start);
        }
        if (Check_Same_String(temp, num, data.end)) {                           //temp에 end와 같은 정점이 있는지 판단
            temp[num] = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));     //같은 정점이 없다면 temp를 동적할당 후 저장
            strcpy(temp[num++], data.end);                  //num이은 중복을 제외한 정점의 개수가 됩니다를
        }
    }
    //      */
////////////////////////////////////////////////////////////////
    init(g, num, temp);                                     //무방향 그래프 g와 g2를 초기화
    init(g2, num, temp);
    for (i = 0; i < num; i++) {
        insert_vertex(g, num);                          //그래프의 정점을 삽입(정점의 최대 개수는 num개이므로 num를 전달)
        insert_vertex(g2, num);
    }
    for(i=0; i<count; i++){
        insert_undirected_edge(g, insert_data[i]);      //그래프에 간선 삽입
        insert_directed_edge(g2, insert_data[i]);       //방향 그래프에 간선삽입
    }
    for (i = 0; i < count; i++) printf("(%s, %s), ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Undirected Graph-₩n");
    print_adj_mat(g);                               //무방향 그래프 출력
    for (i = 0; i < count; i++) printf("<%s, %s>, ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Directed Graph-₩n");
    print_adj_mat(g2);                              //방향 그래프 출력

    for (i = 0; i < num; i++)   free(temp[i]);          //temp의 문자열은 그래프 정점 문자열과 같은 주소를 가지므로 여기에서 한번만 해제할
    free(temp);                                         //함수 Delete함수에서 aru를 해제하면 무방향, 방향에서 두번 해제하는 것이 되버림
    free(insert_data);                                  //파일의 데이터 배열을 받은 insert_data 해제
    Delete(g);                                          //Delete함수로 그래프 동적할당 해제
    Delete(g2);
    fclose(fp);                                         //파일포인터 닫음
    return 0;
```

```c
void Init(GraphType*g, int num, char**temp) {                    //그래프를 초기화하는 함수
    int r;
    g->n = 0;                                                    //정점의 개수 초기화
    g->adj_mat = (int**)calloc(num, sizeof(int*));               //이중포인터 adj_mat를 calloc을 이용해 초기화(0으로 초기화됨)
    g->arv = (char**)malloc(sizeof(char*)*num);                  //arv를 num크기만큼 동적할당(num은 정점의 개수와 같음)
    for (r = 0; r < num; r++) {
        g->adj_mat[r] = (int*)calloc(num, sizeof(int));
        g->arv[r] = temp[r];                                     //arv를 num만큼 동적할당하고 temp에 있는 값을 전달받음
    }
}
void Insert_vertex(GraphType*g, int count) {                     //정점을 삽입하는 함수 count는 메인에서의 num값(정점의 개수)
    if (((g->n) + 1) > count) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;                                                      //정점의 개수 증가
}
void Insert_undirected_edge(GraphType*g, Element data) {         //무방향그래프 삽입함수
    int i, start, end;

    for (i = 0; i < g->n; i++) {
        if (strcmp(g->arv[i], data.start) == 0) start = i;       //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->arv[i], data.end) == 0)end = i;
    }
    if (start >= g->n || end >= g->n) {                          //만약 인덱스 값이 그래프 값보다 크다면 오류
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;                                  //무방향그래프이므로 서로 반대 인덱스에 저장
    g->adj_mat[end][start] = 1;
}
void Insert_directed_edge(GraphType*g, Element data) {  //방향그래프 삽입함수
    int i, start, end;
    for (i = 0; i < g->n; i++) {                                 //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->arv[i], data.start) == 0) start = i;
        if (strcmp(g->arv[i], data.end) == 0) end = i;
    }
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;                          //방향그래프이므로 한쪽 인덱스에만 저장
}
void print_adj_mat(GraphType*g) {          //그래프 출력함수
    int i, j;

    printf("      | ");
    for (i = 0; i < g->n; i++) {
        printf("%6s", g->arv[i]);                //arv배열의 정점 출력
    }
    printf("\n");
    for (i = 0; i < g->n; i++) printf("--------");  //정점의 수에 맞게 브길이가 달라짐

    for (i = 0; i < g->n; i++) {
        printf("\n%6s |", g->arv[i]);             //행 값에 정점 출력
        for (j = 0; j < g->n; j++) {
            printf("%6d ", g->adj_mat[i][j]);     //길이 있으면 1, 없으면 0 출력
        }
    }
    printf("\n\n");
}
int Check_Same_String(char** arv, int num, char *s) {   //정점을 찾을 때 같은 문자열이 있는지 확인하는 함수
    int i;
    for (i = 0; i < num; i++) {                  //문자열 배열을 처음부터 끝까지 돌며 입력받은 문자열과 같은지 확인
        if (strcmp(arv[i], s) == 0)return FALSE;  //만약 같은 문자열이 존재한다면 FALSE리턴
    }
    return TRUE;
}
void Delete(GraphType*g) {                       //그래프 정보를 삭제하는 함수
    int i;
    for (i = 0; i < g->n; i++) {                 //정점의 개수만큼 반복하여 2차원 배열 동적할당을 해제
        free(g->adj_mat[i]);                     //arv문자열 배열은 메인함수에서 temp로 해제됨
    }
    free(g->arv);                                //arv포인터 해제(같록 해제 방지로 같은 메인에서 해제함)
    free(g->adj_mat);                            //2차원 배열 adj_mat해제
    free(g);                                     //그래프 해제
}
```

1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 인접 행렬을 이용한 그래프 추상 데이터 타입의 구현
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
typedef struct Element {
    char start[10];                         //삽입 연산을 할 때 값을 전달
    char end[10];
}Element;
typedef struct GraphType {
    int n;                                  //정점의 개수
    char **ary;                             //정점의 정보
    int **adj_mat;                          //그래프 배열
}GraphType;
```

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

2. 파일에 있는 데이터를 저장할 Element 구조체를 선언한다.
Element는 파일에 있는 시작과 도착 문자열을 받아와서, 그래프에 전달하거나 동적할당하여 파일의 정보를 저장하는 역할을 한다.

3. GraphType은 그래프의 정보를 저장하는 구조체이다.
정점의 개수를 나타내는 n과 정점 문자열의 정보를 담고 있는 ary, 인접 행렬의 정보가 저장되어 있는 정수형 adj_mat변수가 있다.

```
void init(GraphType*g, int num);              //그래프 초기화 함수
void insert_vertex(GraphType*g, int count, char* temp);     //정점 삽입 연산 함수
void insert_undirected_edge(GraphType*g, Element data); //무방향그래프 삽입함수
void insert_directed_edge(GraphType*g, Element data);   //방향그래프 삽입함수
void print_adj_mat(GraphType*g);                        //형식에 맞게 그래프를 출력하는 함수
int Check_Same_String(char** ary, int num, char *s);            //배열에 같은 정점이 있는지 확인하는 함수
void Delete(GraphType*g);                               //그래프의 포인터를 해제하는 함수
```

4. 필요한 함수를 선언한다.

- init함수는 그래프의 정보를 초기화하는 함수이다.
- insert_vertex함수는 인접 행렬의 정점을 삽입하는 함수이다.
- insert_undirected_edge함수는 무방향 그래프로 간선을 삽입하는 함수이다.
- insert_directed_edge함수는 방향그래프로 간선을 삽입하는 함수이다.

- print_adj_mat함수는 그래프를 형식에 맞게 출력하는 함수이다.
- Check_Same_String함수는 배열에 같은 정점이 있는지 확인하는 함수이다.
- Delete함수는 그래프의 정보를 모두 동적할당 해제하는 함수이다.

```
int main() {
    FILE *fp;                           //파일포인터
    Element data;                       //파일로부터 입력받는 변수
    Element *insert_data;               //파일에 있는 데이터를 저장
    int count = 0, num = 0;             //count는 파일에 있는 데이터의 개수, num은 정점의 개수
    char  **temp;                     //temp는 정점의 이름을 임시로 저장하는 문자열 포인터 변수
    GraphType * g, *g2;                 //g는 무방향그래프, g2는 방향 그래프
    int i=0;
    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return;
    }
    while (!feof(fp)) {                             //파일을 끝까지 읽으며 데이터의 개수 증가
        fscanf(fp, "%s%s", data.start, data.end);
        count++;
    }
    g = (GraphType*)malloc(sizeof(GraphType));      //무방향 그래프 생성
    g2 = (GraphType*)malloc(sizeof(GraphType));     //방향 그래프 생성
    temp = (char**)malloc(sizeof(char*)*(count*2));     //파일의 개수에 맞게 temp문자열 배열 할당
    insert_data = (Element*)malloc(sizeof(Element)*count);
```

5. 필요한 변수를 선언한다.

- fp는 파일포인터, data는 파일로부터 데이터를 입력받는 변수, insert_data는 파일 정보만큼 동적할당하여 데이터를 저장하는 역할을 한다.
- count는 파일에 존재하는 데이터의 개수, num은 정점의 개수를 저장한다.
- temp는 정점 문자열을 임시로 저장하는 변수이다.
- g는 무방향 그래프를 나타내고 g2는 방향 그래프를 나타낸다.

6. data파일을 읽기 형식으로 open하고 만약 파일이 존재한다면 파일 끝까지 임시변수 data에 데이터를 입력받으며 파일의 개수를 나타내는 count를 증가시킨다.

7. 무방향 그래프를 나타내는 g와 방향 그래프를 나타내는 g2를 동적할당하여 생성한다.
temp는 정점의 정보를 임시로 저장할 변수이다. 정점이 (정1 정2)의 형태로 있다고 할 때 정1, 정2가 다 다르다면 최대 count의 2배까지 정점이 입력될 수도 있기 때문에 count에 *2하여 동적할당한다.

8. insert_data는 파일데이터를 저장할 Element형 구조체 변수이다.
만약 이 변수가 없다면 파일의 정보가 필요할 때마다 파일포인터를 다시 앞으로 돌리고 다시 입력받는 동작을 해야 하므로 파일 데이터를 변수에 저장하여 불필요한 동작을 줄일 수 있도록 한다.

```
rewind(fp);                                  //파일포인터를 처음으로 되돌림
while (!feof(fp)) {
    fscanf(fp, "%s%s", data.start, data.end);
    insert_data[i++] = data;                 //파일 개수로 동적할당한 배열에 파일 데이터의 값 삽입
                                             //(파일포인터를 계속 되돌리고 입력받는 동작을 줄임)
    if (Check_Same_String(temp, num, data.start)) {     //temp에 같은 정점이 있는지 파악 같은 정점이 없다면 TRUE리턴
        temp[num] = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));   //start의 개수로 temp의 문자열을 동적할당하며 temp에 저장
        strcpy(temp[num++], data.start);
    }
    if (Check_Same_String(temp, num, data.end)) {                       //temp에 end와 같은 정점이 있는지 파악
        temp[num] = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));     //같은 정점이 없다면 temp를 동적할당 후 저장
        strcpy(temp[num++], data.end);                  //num에는 중복을 제외한 정점의 개수가 담겨있음
    }
}
```

9. rewind함수로 파일을 다시 처음으로 돌린 후 방금 동적할당한 변수들에 값을 삽입할 수 있도록 한다.
먼저 파일을 입력받고 인덱스를 증가시키며 insert_data에 입력받은 정보를 저장한다.

10. temp에는 데이터 파일에 있는 정보를 가지고 어떤 정점이 존재하는지 파악하는데, 같은 정점이 있으면 안 된다.
그렇기 때문에 check_Same_String 함수를 호출하여 문자열 배열에 입력받은 start또는 end가 존재하는지 확인한다.

11. 만약 이 함수가 FALSE를 리턴하면 배열에 같은 문자열이 존재한다는 의미이므로 temp에 삽입하지 않고, TRUE가 리턴되면 temp의 가장 처음 포인터부터 동적할당하며 strcpy함수를 이용하여 복사한다. 복사를 한 후에는 num값을 증가시킨다.
이 동작을 모두 마치면 num값은 정점의 개수와 같아질 것이다.

```
강남, 양재
양재, 강남
양재, 역삼
역삼, 잠실
강남, 교대
교대, 양재
종로, 잠실
count = 7, num = 6
강남  양재  역삼  잠실  교대  종로
```

모든 반복을 마친 후 테스트 해본 결과, 총 7개의 데이터를 입력받아 count는 7값을 가지고 있는 것을 알 수 있고, 같은 정점을 제거한 num은 정점의 개수와 같은 값인 6개임을 알 수 있다.

```
    init(g, num);                                   //무방향 그래프 g와 g2를 초기화
    init(g2, num);
    for (i = 0; i < num; i++) {
        insert_vertex(g, num, temp[i]);     //그래프에 정점을 삽입(정점의 최대 개수는 num개이므로 num을 전달)
        insert_vertex(g2, num, temp[i]);
    }
    for(i=0; i<count; i++){
        insert_undirected_edge(g, insert_data[i]);      //그래프에 간선 삽입
        insert_directed_edge(g2, insert_data[i]);       //방향 그래프에 간선삽입
    }
    for (i = 0; i < count; i++) printf("(%s, %s), ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Undirected Graph-₩n");
    print_adj_mat(g);                               //무방향 그래프 출력
    for (i = 0; i < count; i++) printf("<%s, %s>, ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Directed Graph-₩n");
    print_adj_mat(g2);                              //방향 그래프 출력

    for (i = 0; i < num; i++)   free(temp[i]);  //temp의 문자열은 그래프 정점 문자열과 같은 주소를 가지므로 메인에서 한번만 해제함
    free(temp);                 //만약 Delete함수에서 ary를 해제하면 무방향, 방향에서 두번 해제하는 것이 되버림
    free(insert_data);              //파일의 데이터 배열를 담은 insert_data 해제
    Delete(g);                          //Delete함수로 그래프 동적할당 해제
    Delete(g2);
    fclose(fp);                     //파일포인터 닫음
    return 0;
}
```

12. 위에서 구한 정점의 개수 num 값으로 그래프를 초기화하는 init함수를 호출한다. init함수에서는 이 num값을 가지고 행렬과 정점 문자열 배열 ary를 동적할당 할 것이다.

13. I를 num값만큼 반복하며 insert_vertex함수로 그래프에 정점을 삽입한다. 매개변수로는 그래프 변수와 정점의 개수 num값, temp문자열(정점의 정보)를 전달한다. 여기서 temp값을 전달한 이유는 임시로 저장한 temp의 정점 정보가 그래프의 정점에 삽입될 수 있도록 하기 위해서이다.

14. 정점의 삽입이 끝나면 insert_undirected_edge와 insert_direct_edge 함수를 호출하여 간선을 삽입하도록 한다.
여기서는 데이터 파일의 정보를 전달하여 간선을 삽입해야 하므로 I를 count까지 반복하고, 데이터 정보를 담고 있는 insert_data를 매개변수로 전달한다.

15. 모든 간선까지 삽입이 완료 되었다면, 그래프의 정보를 출력한다.
무방향그래프는 (정점 정점)으로 나타내고 무방향그래프는<정점 정점>으로 나타내므로 그에 맞게 출력을 한 후 print_adj_mat함수를 호출하여 그래프를 출력할 수 있도록 한다.

16. 모든 출력이 완료 되었다면 동적할당을 모두 해제할 수 있도록 한다.
먼저 temp에 담겨있는 정점 문자열을 모두 해제해준다.

이를 Delete함수에서 수행하지 않은 이유는 temp 정점 문자열의 주소, 무방향 그래프 정점 문자열 주소, 방향 그래프 정점 문자열 주소가 모두 같기 때문에 Delete를 두 번 수행하면 두 번 메모리 해제가 되는 문제가 생겼다. 그렇기 때문에 메인함수에서 temp의 문자열만 해제할 수 있도록 한다.

17. 정점 문자열을 모두 해제했다면 temp와 insert_data변수도 모두 해제할 수 있도록 하고, 그래프 정보를 삭제하는 Delete함수로 그래프도 삭제한다. 모든 동작을 마치면 fclose로 파일을 닫고 프로그램을 종료한다.

```
int Check_Same_String(char** ary, int num, char *s) {   //정점을 찾을 때 같은 문자열이 있는지 확인하는 함수
    int i;
    for (i = 0; i < num; i++) {                    //문자열 배열을 처음부터 끝까지 돌며 입력받은 문자열과 같은지 확인
        if (strcmp(ary[i], s) == 0)return FALSE;    //만약 같은 문자열이 존재한다면 FALSE리턴
    }
    return TRUE;
}
void init(GraphType*g, int num) {                          //그래프를 초기화하는 함수
    int r;
    g->n = 0;                                              //정점의 개수 초기화
    g->adj_mat = (int**)calloc(num, sizeof(int*));         //이중포인터 adj_mat를 calloc을 이용해 초기화(0으로 초기화됨)
    g->ary = (char**)malloc(sizeof(char*)*num);            //ary를 num크기만큼 동적할당(num은 정점의 개수와 같음)
    for (r = 0; r < num; r++) {
        g->adj_mat[r] = (int*)calloc(num, sizeof(int));
    }
}
void insert_vertex(GraphType*g, int count, char *temp) {//정점을 삽입하는 함수 count는 메인에서의 num값(정점의 개수)
    if (((g->n) + 1) > count) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->ary[g->n] = temp;                                   //ary정점 문자열 배열에 temp값 삽입
    g->n++;                                                //정점의 개수 증가
}
```

18. Check_Same_String함수는 문자열 배열을 반복하며 같은 문자열이 있는지 확인하는 함수이다.
문자열 배열을 입력받은 num값 동안 반복하며 strcmp로 문자열 배열과 입력받은 문자열이 같은지 확인한다. 만약 여기서 같은 값이 존재한다면 FALSE를 리턴하고 같은 값이 없다면 TRUE를 리턴한다.

19. init함수는 그래프를 초기화하는 함수이다. 메인에서 구한 정점 개수인 num값을 매개변수로 전달받는다.
정점 개수 n을 0으로 초기화 하고, num값에 맞게 행렬 adj_mat을 동적할당하는데, 여기서 adj_mat은 calloc을 이용하여 동적할당한다.
calloc으로 동적할당하게 되면 0값으로 초기화가 되기 때문에 malloc으로 할당한 후 다시 0으로 초기화하는 동작을 줄일 수 있다.
calloc은 malloc에서 *로 표현하는 부분을 ,으로 표현하여 동적할당한다.
ary또한 num값에 맞게 동적할당해준다.

20. insert_vertex함수는 정점에 값을 삽입하는 함수이다.
만약 삽입한 후 정점의 개수인 g->n+1이 메인에서의 정점 개수 count값보다 크면 함수를 종료한다.
개수가 맞다면 입력받은 temp 정점의 문자열을 그래프의 정점에 삽입한다.
삽입이 끝나면 정점의 개수인 g->n을 증가시킨다.

```
void insert_undirected_edge(GraphType*g, Element data) {    //무방향그래프 삽입함수
    int i, start, end;

    for (i = 0; i < g->n; i++) {
        if (strcmp(g->ary[i], data.start) == 0) start = i;  //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->ary[i], data.end) == 0)end = i;
    }
    if (start >= g->n || end >= g->n) {                     //만약 인덱스 값이 그래프 값보다 크다면 종료
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;                             //무방향그래프이므로 서로 반대 인덱스에 저장
    g->adj_mat[end][start] = 1;
}
```

21. insert_undirected_edge함수는 무방향그래프로 간선을 삽입하는 함수이다. 매개변수로는 그래프 g와 Element형 data를 전달받는다.

22. 그래프의 정점을 끝까지 반복하며 입력받은 start와 end값과 비교한다. strcmp함수로 정점과 문자열을 비교하고 정점의 인덱스 값을 찾는다. 이 인덱스 값을 start와 end값에 각각 저장한다.
만약 이 인덱스 값이 정점의 개수보다 크다면 오류를 나타내고 함수를 종료한다.

23. 이 함수는 무방향 그래프로 삽입하는 함수이기 때문에 위에서 구한 start행 end열의 값에 1을 삽입할 뿐만 아니라 반대 방향인 end행 start열에도 1을 삽입하여 무방향 그래프를 만족할 수 있도록 한다.

```
void insert_directed_edge(GraphType*g, Element data) {  //방향그래프 삽입함수
    int i, start, end;
    for (i = 0; i < g->n; i++) {                        //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->ary[i], data.start) == 0) start = i;
        if (strcmp(g->ary[i], data.end) == 0) end = i;
    }
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;                         //방향그래프이므로 한쪽 인덱스에만 저장
}
```

24. insert_direct_edge함수는 방향그래프로 간선을 삽입하는 함수이다.
무방향그래프에서 했던 것과 같이 방향그래프에서도 그래프의 정점을 끝까지 반복하며 입력받은 start와 end값과 비교한다.

25. 인덱스를 모두 찾았다면 먼저 정점의 개수 g->n과 비교한다. 만약 정점의 개수보다 작다면 행렬에 값을 삽입한다.
이 함수는 방향 그래프로 삽입하기 때문에 한쪽 방향만 삽입하고 반대쪽은 삽입하지 않는다. 그렇기 때문에 start행 end열의 값에만 1을 삽입한다.

```
void print_adj_mat(GraphType*g) {           //그래프 출력함수
    int i, j;

    printf(" %6c | ", ' ');
    for (i = 0; i < g->n; i++) {
        printf("%7s", g->ary[i]);                   //ary배열의 정점 출력
    }
    printf("₩n--------|-");
    for (i = 0; i < g->n; i++) printf("-------");   //정점의 수에 맞게 표길이가 달라짐

    for (i = 0; i < g->n; i++) {
        printf("₩n %-6s |", g->ary[i]);             //행 값에 정점 출력
        for (j = 0; j < g->n; j++) {
            printf("%6d ", g->adj_mat[i][j]);       //길이 있으면 1, 없으면 0 출력
        }
    }
    printf("₩n₩n");
}
```

26. print_adj_mat함수는 그래프를 형식에 맞게 출력하는 함수이다.
가로축을 나타내기 위해 정점의 개수만큼 반복하여 정점문자열을 출력한다.
여기서 표를 구분하는 ---선의 경우 정점의 수에 맞게 길이가 달라진다.

```
-Undirected Graph-
        |    강남    양재    역삼    잠실    교대
--------|-------------------------------------
 강남   |      0       1       0       0       1
```

```
-Undirected Graph-
        |    강남    양재    역삼    잠실    교대    종로  회기 청량리    건대    신창    천안
--------|-------------------------------------------------------------------------------
 강남   |      0       1       0       0       1       0       0       0       0       0       0
```

27. 이중 for문을 이용하여 간선 행렬을 출력한다. 먼저 세로축을 나타내기 위해 정점 문자열을 출력한다. 처음에는 %s로만 출력했었지만 3글자를 입력하자 형식이 깨져서 %-6s로 정렬하여 형식에 맞게 출력하였다.
세로축을 나타낸 후 한 번 더 for문을 반복하여 열값의 데이터를 출력할 수 있도록 한다.

```
void Delete(GraphType*g) {          //그래프 정보를 삭제하는 함수
    int i;
    for (i = 0; i < g->n; i++) {//정점의 개수만큼 반복하며 2차원 배열 동적할당을 해제
        free(g->adj_mat[i]);    //ary문자열 배열은 메인함수에서 temp로 해제완료
    }
    free(g->ary);               //ary포인터 해제
    free(g->adj_mat);           //2차원 배열 adj_mat해제
    free(g);                    //그래프 해제
}
```

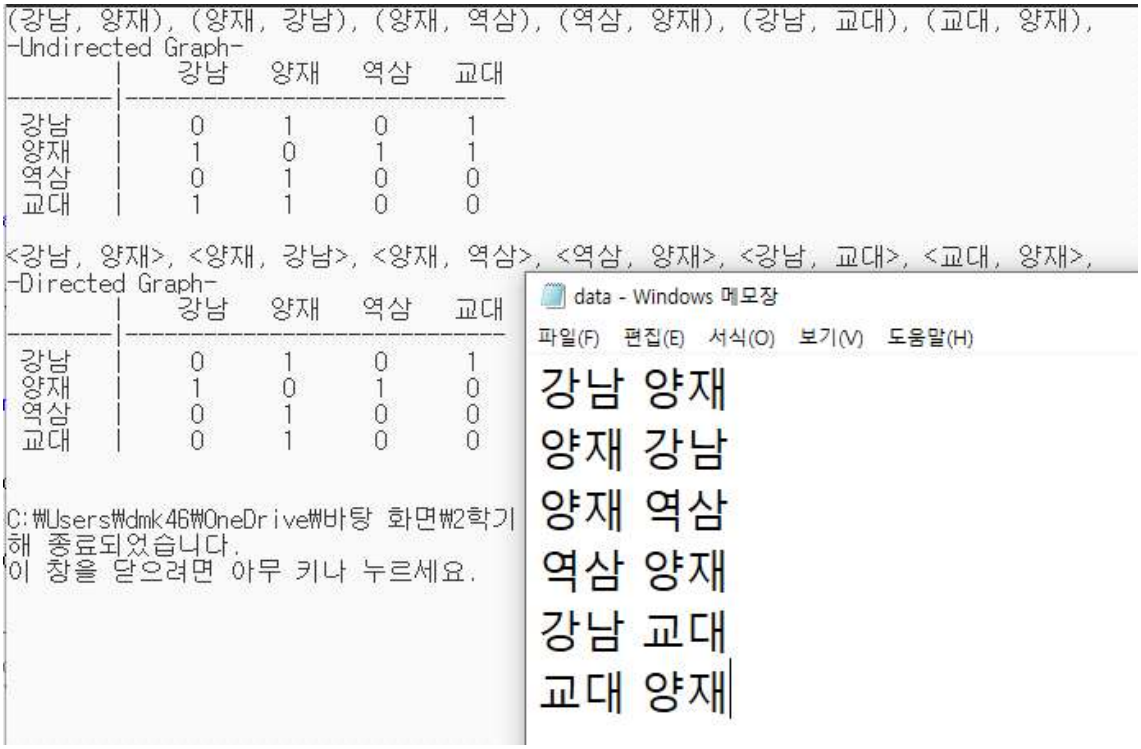
28. Delete함수는 그래프의 포인터를 모두 메모리 해제하는 함수이다.
먼저 그래프 정점만큼 반복하며 열값 메모리를 해제한 후 행도 해제해준다.

29. 여기서 ary문자열을 해제하지 않는 이유는 같은 주소를 가지고 있는 temp를 메인함수에서 이미 해제했기 때문이다.
초기화 함수에서 정점을 삽입할 때, 정점문자열을 새로 동적할당하지 않고 temp의 주소를 복사했기 때문에 여기서 ary[i]를 또 해제하게 되면 똑같은 주소를 두 번 메모리 해제하는 것이기 때문에 오류가 나게 된다. 그러므로 num값으로 동적할당한 g->ary포인터만 해제해주도록 한다.

30. 마지막으로 그래프 g를 메모리 해제하면 Delete가 끝난다.

1.4 실행 창

- 7주차 과제 기본 데이터 파일



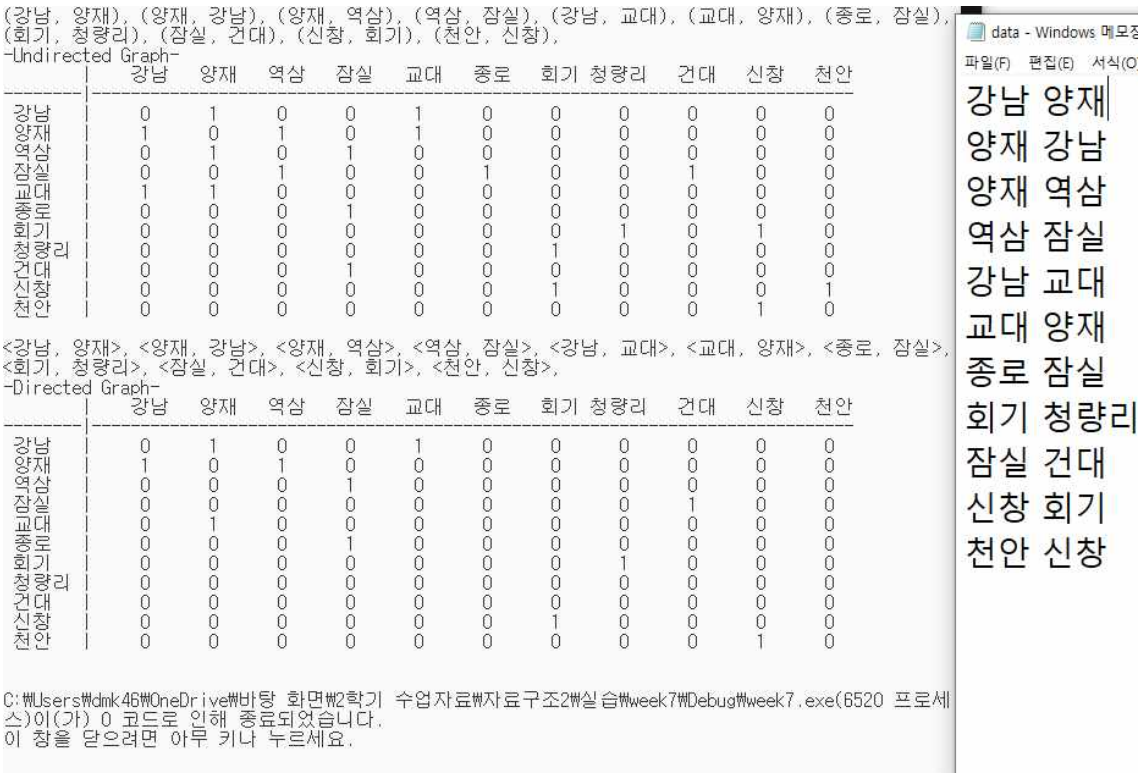
```
(강남, 양재), (양재, 강남), (양재, 역삼), (역삼, 양재), (강남, 교대), (교대, 양재),
-Undirected Graph-
        |    강남    양재    역삼    교대
--------|----------------------------
 강남   |      0       1       0       1
 양재   |      1       0       1       1
 역삼   |      0       1       0       0
 교대   |      1       1       0       0

<강남, 양재>, <양재, 강남>, <양재, 역삼>, <역삼, 양재>, <강남, 교대>, <교대, 양재>,
-Directed Graph-
        |    강남    양재    역삼    교대
--------|----------------------------
 강남   |      0       1       0       1
 양재   |      1       0       1       0
 역삼   |      0       1       0       0
 교대   |      0       1       0       0


C:₩Users₩dmk46₩OneDrive₩바탕 화면₩2학기
해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

data - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
강남 양재
양재 강남
양재 역삼
역삼 양재
강남 교대
교대 양재
```

- 데이터 파일 추가와 3글자 역 입력과 표 형식 확인(희소 행렬)



```
(강남, 양재), (양재, 강남), (양재, 역삼), (역삼, 잠실), (강남, 교대), (교대, 양재), (종로, 잠실),
(회기, 청량리), (잠실, 건대), (신창, 회기), (천안, 신창),
-Undirected Graph-
        |    강남    양재    역삼    잠실    교대    종로    회기  청량리    건대    신창    천안
--------|----------------------------------------------------------------------------
 강남   |      0       1       0       0       1       0       0       0       0       0       0
 양재   |      1       0       1       0       1       0       0       0       0       0       0
 역삼   |      0       1       0       1       0       0       0       0       0       0       0
 잠실   |      0       0       1       0       0       1       0       0       1       0       0
 교대   |      1       1       0       0       0       0       0       0       0       0       0
 종로   |      0       0       0       1       0       0       0       0       0       0       0
 회기   |      0       0       0       0       0       0       0       1       0       1       0
 청량리 |      0       0       0       0       0       0       1       0       0       0       0
 건대   |      0       0       0       1       0       0       0       0       0       0       0
 신창   |      0       0       0       0       0       0       1       0       0       0       1
 천안   |      0       0       0       0       0       0       0       0       0       1       0

<강남, 양재>, <양재, 강남>, <양재, 역삼>, <역삼, 잠실>, <강남, 교대>, <교대, 양재>, <종로, 잠실>,
<회기, 청량리>, <잠실, 건대>, <신창, 회기>, <천안, 신창>,
-Directed Graph-
        |    강남    양재    역삼    잠실    교대    종로    회기  청량리    건대    신창    천안
--------|----------------------------------------------------------------------------
 강남   |      0       1       0       0       1       0       0       0       0       0       0
 양재   |      1       0       1       0       0       0       0       0       0       0       0
 역삼   |      0       0       0       1       0       0       0       0       0       0       0
 잠실   |      0       0       0       0       0       0       0       0       1       0       0
 교대   |      0       1       0       0       0       0       0       0       0       0       0
 종로   |      0       0       0       1       0       0       0       0       0       0       0
 회기   |      0       0       0       0       0       0       0       1       0       0       0
 청량리 |      0       0       0       0       0       0       0       0       0       0       0
 건대   |      0       0       0       0       0       0       0       0       0       0       0
 신창   |      0       0       0       0       0       0       1       0       0       0       0
 천안   |      0       0       0       0       0       0       0       0       0       1       0


C:₩Users₩dmk46₩OneDrive₩바탕 화면₩2학기 수업자료₩자료구조2₩실습₩week7₩Debug₩week7.exe(6520 프로세
스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

data - Windows 메모장

파일(F) 편집(E) 서식(O)

```
강남 양재
양재 강남
양재 역삼
역삼 잠실
강남 교대
교대 양재
종로 잠실
회기 청량리
잠실 건대
신창 회기
천안 신창
```

1.5 느낀 점

이번 문제는 인접행렬을 이용하여 그래프를 표현하는 문제였습니다. 책에 있는 코드를 응용하여 코드를 짰기 때문에 처음에는 쉽게 프로그래밍할 수 있을 것이라고 생각했습니다. 하지만 중간에 갑자기 문제가 정점이 문자열로 저장하는 것으로 바뀌어서 잠깐 당황했지만 원래 짰던 코드에서 정점 문자열을 추가하고 하나하나 코드를 바꾸어서 프로그래밍을 끝마칠 수 있었습니다.

이번 프로그래밍을 할 때 처음에는 임시 문자열 배열을 데이터의 개수 count로만 동적할당을 하도록 했었습니다. 하지만 만약 정점이 모두 다를 경우에 정점 문자열의 개수가 count보다 더 많아지는 문제가 생겼었습니다. 그래서 제가 잘못 계산했다는 것을 알아내서 임시 문자열 배열을 count개의 2배로 동적할당을 했습니다. 만약 제가 데이터를 더 추가해보지 않았거나 다른 여러 데이터를 넣어보지 않았다면 찾을 수 없었을 문제였었는데, 이번을 계기로 예시로 나온 데이터 말고도 다른 여러 데이터를 테스트해봐야 완벽하게 프로그램을 짤 수 있다는 것을 깨달을 수 있었습니다.
처음 코드를 짰을 때는 데이터 파일을 담을 수 있는 배열을 만들지 않고 파일을 다시 처음으로 돌렸다가 다시 입력받는 동작을 반복했었습니다. 이게 한 번이나 두 번이면 그냥 넘어갔을 문제였었는데, 계속 파일포인터를 처음으로 돌리고 while로 다시 입력하는 동작을 하니 코드가 가독성이 떨어지고 복잡해지는 문제가 생겼었습니다. 그래서 Element형 포인터를 데이터의 개수만큼 동적할당하여 파일의 데이터를 저장하고 필요한 곳에 불러 쉽게 사용할 수 있었습니다.

이번 과제를 계기로 인접행렬을 이용한 그래프에 대해 이해를 할 수 있었습니다. 특히 두 번째 실행파일과 같이 행렬이 희소행렬이면 시간이 많이 소요되므로 이 경우 리스트를 이용해야겠다고 생각했습니다. 또한 책에 나와 있는 코드로만 한 것이 아니라 정점을 문자열로 바꾸어 프로그래밍하면서 더 많은 것을 배울 수 있었던 기회가 되었던 것 같습니다.

2. 그래프

인접 리스트

- 375페이지의 프로그램 10.2을 이용하여 파일 data.txt에서 정점과 에지 정보를 입력받아 인접 행렬을 생성하여 그래프를 생성하는 프로그램을 작성하시오

2.1 문제 분석

조건 1 data.txt파일의 데이터 형식은 에지정보(정점 정점)로 구성되어 있음
조건 2 그래프에 대해서 data.txt파일들을 직접 생성하여 입력받음
조건 3 그래프 간선 정보를 리스트에 저장

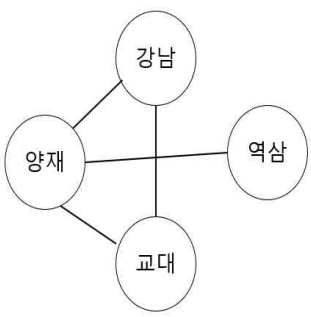
- 이 문제는 인접 리스트를 이용하여 간선을 저장하는 문제이다. 여기서 정점과 간선에 대한 정보는 정수로 표현하지 않고 문자열로 표현한다.

메모리를 아끼기 위해 정점에 대한 정보를 포인터로 선언하여 동적할당하여 저장한다. 정점의 정보를 알맞게 동적할당하고 입력받기 위해서는 파일을 입력받는 과정에서 정점의 개수를 정확히 파악해야 메모리 낭비 없이 동적할당을 할 수 있을 것이다. 이를 위해 중복 없이 파일의 개수의 *2로 정점을 저장하는 임시 문자열 배열을 만들고, 이 문자열 배열이 들어있는 값만큼 그래프에서 정점 배열을 만들어 정점을 삽입하면 될 것이다.

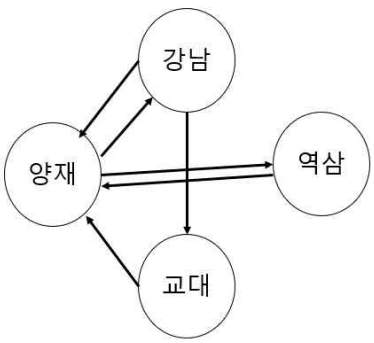
인접 리스트로 나타낼 수 있는 그래프는 무방향 그래프와 방향 그래프가 있다. 이 두 개를 모두 나타내기 위해 그래프를 2개 생성하고 값을 삽입하여 각각 무방향 그래프와 방향 그래프로 만들어 본다. 그리고 그래프의 출력을 확인해보며 둘의 차이점을 확인해본다.

데이터 파일

무방향 그래프



방향 그래프



2.2 소스 코드

```
/*
    학번 : 2015A618
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김홍민
    과제 명 : 링킹 리스트를 이용한 그래프 추상 데이터 타입의 구현
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

typedef struct Element {
    char start[10];             //삽입 연산을 할 때 값을 연결하는 구조체||데이터 파일을 저장
    char end[10];
}Element;

typedef struct GraphNode {      //간선 리스트를 저장하는 노드
    char *vertex;
    struct GraphNode *link;
}GraphNode;
typedef struct GraphType {      //간선 리스트 타입
    int n;                      //정점의 개수
    char **aru;                 //정점의 정보
    GraphNode ** adj_list;      //간선 리스트 정보
}GraphType;

void init(GraphType*, int , char**);          //그래프를 초기화하는 함수
void insert_vertex(GraphType*, int );         //정점을 삽입하는 함수
void insert_undirected_edge(GraphType*, Element);//무방향 그래프 삽입함수
void insert_directed_edge(GraphType*, Element); //방향 그래프 삽입함수
void print_adj_list(GraphType*);              //그래프와 리스트를 형식에 맞게 출력하는 함수
int Check_Node(GraphNode*, char*);            //무방향 그래프 삽입시 같은 간선이 있으를 확인하는 함수
int Check_Same_String(char** , int , char *);  //정점 삽입시 동일하은 정점이 있으를 확인하는 함수
void Delete(GraphType*);                      //그래프와 리스트를 삭제하는 함수

int main() {
    FILE *fp;
    GraphType*a, *a2;               //a은 무방향그래프, a2은 방향 그래프
    char **temo;                    //temo은 정점의 이름을 임시로 저장하는 문자열 포인터 변수
    Element data;
    Element *insert_data;           //파일에 있은 데이터를 저장
    int count = 0, num = 0;         //count은 파일에 있은 데이터의 개수, num은 정점의 개수
    int i = 0;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {                         //파일을 열기까지 있으며 데이터의 개수 증가
        fscanf(fp, "%s%s", data.start, data.end);
        count++;
    }
    a = (GraphType*)malloc(sizeof(GraphType));      //무방향그래프 a를생성함
    a2= (GraphType*)malloc(sizeof(GraphType));      //방향그래프 a2를생성함
    temo = (char**)malloc(sizeof(char*)*count);     //temo를 데이터 파일의 개수로 동적할당
    insert_data = (Element*)malloc(sizeof(Element)*count);  //insert_data를 데이터 파일의 개수로 동적할당

    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%s%s", data.start, data.end);
    //  printf("insert [%s, %s]₩n", data.start, data.end);
        insert_data[i++] = data;                    //파일을 처음부터 다시 읽은뒤으며 데이터 파일 저장

        if (Check_Same_String(temo, num, data.start)) {          //현재 temo배열에 읽은뒤은 문자열이 있은지 확인
            temo[num] = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));
            strcpy(temo[num++], data.start);                     //없으면 현재temo문자열을 동적할당하고 값을 저장, 저장후은 개수 증가
        }
        if (Check_Same_String(temo, num, data.end)) {        //현재 temo배열에 읽은뒤은 end문자열이 있은지 확인
            temo[num] = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));
            strcpy(temo[num++], data.end);
        }
    }
////////////////////////////////////////////////////////////////
    init(a, num, temo);                                     //무방향 그래프 a를 초기화 num은 정점의 개수, temo의 값 연결
    init(a2, num, temo);
    for (i = 0; i < num; i++) {                             //정점 추가 num만큼 반복하여 vertex추가
        insert_vertex(a, num);
        insert_vertex(a2, num);
    }
    for(i=0; i<count; i++){                                 //데이터 파일의 개수인 count만큼 반복
        insert_undirected_edge(a, insert_data[i]);          //무방향그래프에 현재 data파일을 연결하여 간선삽입
        insert_directed_edge(a2, insert_data[i]);           //방향그래프에 현재 data파일을 연결하여 간선삽입
    }
    for (i = 0; i < count; i++) printf("(%s, %s), ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Unlocated Graph-₩n");
    print_adj_list(a);                              //무방향 그래프 출력
    for (i = 0; i < count; i++) printf("<%s, %s>, ", insert_data[i].start, insert_data[i].end);
    printf("₩n-Directed Graph-₩n");
    print_adj_list(a2);                             //방향 그래프 출력

    for (i = 0; i < num; i++)   free(temo[i]);  //temo의 문자열은 그래프 정점 문자열과 같은 주소를 가지므로 마린에서 완전한 해제할
    free(insert_data);                          //파일의 데이터 배열을 없은 insert_data 해제
    free(temo);
    Delete(a);                          //Delete함수로 그래프 동적할당 해제
    Delete(a2);
    fclose(fp);                         //파일포인터 닫음
    return 0;
}
```

```
int Check_Node(GraphNode*list, char*s) {                //학습목록에서 같은 문자열이 있는지 체크하는 함수
    GraphNode*p = list;
    while (p != NULL) {
        if (strcmp(s, p->vertex) == 0)return FALSE;     //s를 넣으려 선택하는 같은 문자열이 있으면 FALSE리턴
        p = p->link;
    }
    return TRUE;
}
int Check_Same_String(char** arr, int num, char *s) {   //배열에서 같은 문자열이 있는지 체크하는 함수
    int i;
    for (i = 0; i < num; i++) {
        if (strcmp(arr[i], s) == 0)return FALSE;
    }
    return TRUE;                                        //아무 같은 문자열이 없으면 참이면 TRUE리턴
}
void Init(GraphType*g, int VERTICES, char**name) {          //그래프를 초기화하는 함수
    int v;
    g->n = 0;
    g->vadj_list = (GraphNode**)malloc(sizeof(GraphNode*)*VERTICES); //학습목록 인접의 개수만큼 동적할당
    g->vary = (char**)malloc(sizeof(char*)*VERTICES);               //인접도 동적할당
    for (v = 0; v < VERTICES; v++) {
        g->vadj_list[v] = NULL;
        g->vary[v] = name[v];                                       //name순서에 따라 같은 인덱스로
    }
}
void Insert_vertex(GraphType*g, int VERTICES) {             //인접을 삽입하는 함수
    if (((g->n) + 1) > VERTICES) {
        fprintf(stderr, "그래프: 인접의 개수 초과");
        return;
    }
    g->n++;                                                 //인접할때마다 인접의 개수 증가
}
void Insert_directed_edge(GraphType*g, Element data) {      //방향그래프에 삽입하는 함수
    GraphNode *node;
    int u, v, i;
    for (i = 0; i < g->n; i++) {                //시작문자열의 인덱스값과 도착 문자열의 인덱스 같은 참조
        if (strcmp(g->vary[i], data.start) == 0) u = i;
        if (strcmp(g->vary[i], data.end) == 0) v = i;
    }
    if (u == g->n || v == g->n) {
        fprintf(stderr, "그래프: 인접 번호 오류");
        return;
    }                                                       //방향그래프이므로 학습목 하여 end값만 추가하면 됨
    node = (GraphNode*)malloc(sizeof(GraphNode));           //새로운 노드 생성
    node->vertex = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));  //노드의 문자열은 end의 길이로 동적할당
    strcpy(node->vertex, data.end);                         //end문자열을 복사

    node->link = g->vadj_list[u];                           //노드의 link가 adj_list의 자료를 가져감(insert_first)
    g->vadj_list[u] = node;                                 //adj_list의 head값이 node가 됨
    // printf("%d------%s\n",u, node->vertex);
}
void Insert_undirected_edge(GraphType*g, Element data) {    //무방향 그래프에 삽입하는 함수
    GraphNode *node;
    GraphNode *node2;
    int u, v, i;
    for (i = 0; i < g->n; i++) {                //시작문자열의 인덱스값과 도착 문자열의 인덱스 같은 참조
        if (strcmp(g->vary[i], data.start) == 0) u = i;
        if (strcmp(g->vary[i], data.end) == 0) v = i;
    }
    if (u == g->n || v == g->n) {
        fprintf(stderr, "그래프: 인접 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode));           //node는 end문자열을 저장
    node2 = (GraphNode*)malloc(sizeof(GraphNode));          //node2는 start문자열을 저장
    if (Check_Node(g->vadj_list[u], data.end)) {            //현재 학습목록 end문자열이 삽입되었는지 확인
        node->vertex = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));  //삽입되지 않은것이면 node의 vertex를 동적할당하고 값을 복사
        strcpy(node->vertex, data.end);
        node->link = g->vadj_list[u];                       //노드의 link가 학습목록 자료를 가져감
        g->vadj_list[u] = node;                             //학습목록 자료이 node가 헤수 인프를 함
    }
    if (Check_Node(g->vadj_list[v], data.start)) {          //현재 학습목록에 start문자열이 삽입되었는지 확인
        node2->vertex = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));   //삽입되지 않은것이면 node2의 vertex를 동적할당하고 값을 복사
        strcpy(node2->vertex, data.start);
        node2->link = g->vadj_list[v];
        g->vadj_list[v] = node2;
    }
}
void print_adj_list(GraphType*g) {                      //그래프를 인접에 맞게 출력하는 함수
    int i;
    for (i = 0; i < g->n; i++) {
        GraphNode*p = g->vadj_list[i];                  //학습목 배열의 같은 p에 저장
        printf("인접 #'%s'의 인접 학습목 ", g->vary[i]);   //현재 인접 출력
        while (p != NULL) {                             //p학습목의 끝까지 인접목이 출력
            printf("-> %s ", p->vertex);
            p = p->link;
        }
        printf("\n");
    }
    printf("\n");
}
void Delete(GraphType*g) {              //그래프를 동적할당 해제하는 함수
    int i;
    GraphNode *p, *q;
    for (i = 0; i < g->n; i++) {
        p = g->vadj_list[i];            //p의 학습목 배열의 같은 인접번호
        q = p;
        if (p != NULL) {                //만약 p가 NULL이면 학습목이 없으므로 해제하지 않음
        // printf("NotNull, ");
            while (p != NULL) {
            // printf("%s ", p->vertex);
                p = p->link;            //p가 다음링크 주소를 가져감
                free(p->vertex); // ==free(g->vadj_list[i].vertex);  //현재 주소 p의 학습노드 해제
                free(q);    // ==free(g->vadj_list[i])             //p의 학습노드도 해제
                q = p;                                          //p의 다음 주소를 가져가는 p하감
            }
        }
    }
    free(g->vary);          //ary인접배열 해제 문자열은 이전에서 name로 해제함
    free(g->vadj_list);     //학습목포인터 해제
    free(g);
}
```

2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 인접 리스트를 이용한 그래프 추상 데이터 타입의 구현
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

typedef struct Element {
    char start[10];                 //삽입 연산을 할 때 값을 전달하는 구조체||데이터 파일을 저장
    char end[10];
}Element;

typedef struct GraphNode {          //간선 리스트를 저장하는 노드
    char *vertex;
    struct GraphNode *link;
}GraphNode;
typedef struct GraphType {          //간선 리스트 타입
    int n;                          //정점의 개수
    char **ary;                     //정점의 정보
    GraphNode ** adj_list;          //간선 리스트 정보
}GraphType;
```

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

2. 파일에 있는 데이터를 저장할 Element 구조체를 선언한다.
Element는 파일에 있는 시작과 도착 문자열을 받아와서, 그래프에 전달하거나 동적할당하여 파일의 정보를 저장하는 역할을 한다.

3. 각 항을 나타내는 노드를 정의한다. GraphNode는 간선 리스트노드를 저장하는 구조체이다. link는 GraphNode타입의 다음 노드를 가리킨다.

4. GraphType은 그래프의 정보를 저장하는 구조체이다.
정점의 개수를 나타내는 n과 정점 문자열의 정보를 담고 있는 ary, 인접 리스트의 정보가 저장되어 있는 GraphNode리스트를 담고 있다.

```
void init(GraphType*, int );            //그래프를 초기화하는 함수
void insert_vertex(GraphType*, int, char* );    //정점을 삽입하는 함수
void insert_undirected_edge(GraphType*, Element);//무방향 그래프 삽입함수
void insert_directed_edge(GraphType*, Element); //방향 그래프 삽입함수
void print_adj_list(GraphType*);                //그래프와 리스트를 형식에 맞게 출력하는 함수
int Check_Node(GraphNode*, char*);              //무방향 그래프 삽입시 같은 간선이 없도록 확인하는 함수
int Check_Same_String(char** , int , char *);   //정점 삽입시 중복되는 정점이 없도록 확인하는 함수
void Delete(GraphType*);                        //그래프와 리스트를 삭제하는 함수
```

5. 필요한 함수를 선언한다.

- init은 그래프의 정보를 초기화하는 함수이다.
- insert_vertex함수는 인접 리스트의 정점을 삽입하는 함수이다.
- insert_undirected_edge는 무방향그래프의 간선 리스트를 삽입하는 함수이다.
- insert_directed_edge함수는 방향그래프의 간선 리스트를 삽입하는 함수이다.
- print_adj_list함수는 인접 리스트 그래프를 형식에 맞게 출력하는 함수이다.
- Check_Node함수는 리스트에 같은 문자열이 있는지 확인하는 함수이다.
- Check_Same_String함수는 배열에 같은 문자열이 있는지 확인하는 함수이다.
- Delete함수는 그래프의 정보를 모두 동적할당 해제하는 함수이다.

```
int main( ) {
    FILE *fp;
    GraphType*g, *g2;                   //g는 무방향그래프, g2는 방향 그래프
    char **temp;                    //temp는 정점의 이름을 임시로 저장하는 문자열 포인터 변수
    Element data;
    Element *insert_data;           //파일에 있는 데이터를 저장
    int count = 0, num = 0;         //count는 파일에 있는 데이터의 개수, num은 정점의 개수
    int i = 0;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {                             //파일을 끝까지 읽으며 데이터의 개수 증가
        fscanf(fp, "%s%s", data.start, data.end);
        count++;
    }
    g = (GraphType*)malloc(sizeof(GraphType));      //무방향그래프 g동적할당
    g2= (GraphType*)malloc(sizeof(GraphType));      //방향그래프 g2동적할당
    temp = (char**)malloc(sizeof(char*)*(count*2));     //temp를 데이터 파일의 개수로 동적할당
    insert_data = (Element*)malloc(sizeof(Element)*count);  //insert_data를 데이터 파일의 개수로 동적할당
```

6. 필요한 변수들을 선언한다.

- fp는 파일포인터, g는 무방향그래프, g2는 방향그래프를 나타낸다.
- temp는 정점의 이름을 임시로 저장하는 이중 포인터 변수이다.
- data는 파일로부터 데이터를 입력받는 변수, insert_data는 파일 정보만큼 동적할당하여 데이터를 저장하는 역할을 한다.
- count는 파일에 존재하는 데이터의 개수, num은 정점의 개수를 저장한다.

7. data파일을 읽기 형식으로 open하고 만약 파일이 존재한다면 파일 끝까지 임시변수 data에 데이터를 입력받으며 파일의 개수를 나타내는 count를 증가시킨다.

8. 무방향 그래프를 나타내는 g와 방향 그래프를 나타내는 g2를 동적할당하여 생성한다.
temp는 정점의 정보를 임시로 저장할 변수이다. 정점이 (정1 정2)의 형태로 있다고 할 때 데이터의 정1, 정2가 다 다르다면 최대 count의 2배까지 정점이 입력될 수도 있기 때문에 count에 *2하여 동적할당한다.

9. insert_data는 파일데이터를 저장할 Element형 구조체 변수이다.
만약 이 변수가 없다면 파일의 정보가 필요할 때마다 파일포인터를 다시 앞으로 돌리고 다시 입력받는 동작을 해야 하므로 파일 데이터를 변수에 저장하여 불필요한 동작을 줄일 수 있도록 한다.

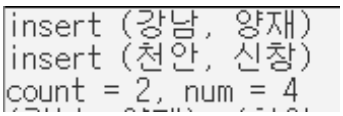
```
rewind(fp);
while (!feof(fp)) {
    fscanf(fp, "%s%s", data.start, data.end);
//  printf("insert (%s, %s)₩n", data.start, data.end);
    insert_data[i++] = data;                        //파일을 처음부터 다시 입력받으며 데이터 파일 저장

    if (Check_Same_String(temp, num, data.start)) { //현재 temp배열에 입력받은 문자열이 있는지 확인
        temp[num] = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));
        strcpy(temp[num++], data.start);    //없다면 현재temp문자열을 동적할당하고 값을 저장, 저장후엔 개수 증가
    }
    if (Check_Same_String(temp, num, data.end)) {   //현재 temp배열에 입력받은 end문자열이 있는지 확인
        temp[num] = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));
        strcpy(temp[num++], data.end);
    }
}
```

10. rewind함수로 파일을 다시 처음으로 돌린 후 방금 동적할당한 변수들에 값을 삽입할 수 있도록 한다.
먼저 파일을 입력받고 인덱스를 증가시키며 insert_data에 입력받은 정보를 저장한다.

11. temp에는 데이터 파일에 있는 정보를 가지고 어떤 정점이 존재하는지 파악하는데, 같은 정점이 있으면 안 된다.
그렇기 때문에 check_Same_String 함수를 호출하여 문자열 배열에 입력받은 start또는 end가 존재하는지 확인한다.

12. 만약 이 함수가 FALSE를 리턴하면 배열에 같은 문자열이 존재한다는 의미이므로 temp에 삽입하지 않고, TRUE가 리턴되면 temp의 가장 처음 포인터부터 동적할당하며 strcpy함수를 이용하여 복사한다. 복사를 한 후에는 num값을 증가시킨다.
이 동작을 모두 마치면 num값은 정점의 개수와 같아질 것이다.

모두 다른 정점 정보를 가지고 있는 데이터를 삽입하고 테스트 해본 결과, count가 2일 때 최대로 입력받을 수 있는 4개의 데이터까지 temp에 입력받음을 확인할 수 있었다.




```
    init(g, num);                                   //무방향 그래프 g를 초기화 num은 정점의 개수
    init(g2, num);
    for (i = 0; i < num; i++) {                         //정점 추가 num만큼 반복하며 vertex추가
        insert_vertex(g, num, temp[i]);                 //i번째의 temp문자열을 전달하여 정점에 삽입
        insert_vertex(g2, num, temp[i]);
    }
    for(i=0; i<count; i++){                             //데이터 파일의 개수인 count만큼 반복
        insert_undirected_edge(g, insert_data[i]);      //무방향그래프에 현재 data파일을 전달하여 간선삽입
        insert_directed_edge(g2, insert_data[i]);       //방향그래프에 현재 data파일을 전달하여 간선삽입
    }
    for (i = 0; i < count; i++) printf("(%s, %s), ", insert_data[i].start, insert_data[i].end);
    printf("\n-Undirected Graph-\n");
    print_adj_list(g);                                  //무방향 그래프 출력
    for (i = 0; i < count; i++) printf("<%s, %s>, ", insert_data[i].start, insert_data[i].end);
    printf("\n-Directed Graph-\n");
    print_adj_list(g2);                                 //방향 그래프 출력

    for (i = 0; i < num; i++)   free(temp[i]);  //temp의 문자열은 그래프 정점 문자열과 같은 주소를 가지므로 메인에
    free(insert_data);                          //파일의 데이터 배열을 담은 insert_data 해제
    free(temp);
    Delete(g);                              //Delete함수로 그래프 동적할당 해제
    Delete(g2);
    fclose(fp);                             //파일포인터 닫음
    return 0;
```

13. 위에서 구한 정점의 개수 num 값으로 그래프를 초기화하는 init함수를 호출한다. init함수에서는 이 num값을 가지고 행렬과 정점 문자열 배열 ary를 동적할당 할 것이다.

14. I를 num값만큼 반복하며 insert_vertex함수로 그래프에 정점을 삽입한다. 매개변수로는 그래프 변수와 정점의 개수 num값, temp문자열(정점의 정보)를 전달한다. 여기서 temp값을 전달한 이유는 임시로 저장한 temp의 정점 정보가 그래프의 정점에 삽입될 수 있도록 하기 위해서이다.

15. 정점의 삽입이 끝나면 insert_undirected_edge와 insert_direct_edge 함수를 호출하여 간선을 삽입하도록 한다.
여기서는 데이터 파일의 정보를 전달하여 간선을 삽입해야 하므로 I를 count까지 반복하고, 데이터 정보를 담고 있는 insert_data를 매개변수로 전달한다.

16. 모든 간선까지 삽입이 완료 되었다면, 그래프의 정보를 출력한다. 무방향그래프는 (정점 정점)으로 나타내고 무방향그래프는<정점 정점>으로 나타내므로 그에 맞게 출력을 한 후 print_adj_mat함수를 호출하여 그래프를 출력할 수 있도록 한다.

17. 모든 출력이 완료 되었다면 동적할당을 모두 해제할 수 있도록 한다. 먼저 temp에 담겨있는 정점 문자열을 모두 해제해준다.

18. 정점 문자열을 모두 해제했다면 temp와 insert_data변수도 모두 해제할 수 있도록 하고, 그래프 정보를 삭제하는 Delete함수로 그래프도 삭제한다. 모든 동작을 마치면 fclose로 파일을 닫고 프로그램을 종료한다.

```
int Check_Node(GraphNode*list, char*s) {                //리스트에서 같은 문자열이 있는지 체크하는 함수
    GraphNode*p = list;
    while (p != NULL) {
        if (strcmp(s, p->vertex) == 0)return FALSE;     //p를 끝까지 반복하며 같은 문자열이 있다면 FALSE리턴
        p = p->link;
    }
    return TRUE;
}
int Check_Same_String(char** ary, int num, char *s) {   //배열에서 같은 문자열이 있는지 체크하는 함수
    int i;
    for (i = 0; i < num; i++) {
        if (strcmp(ary[i], s) == 0)return FALSE;
    }
    return TRUE;                                        //만약 같은 문자열이 하나도 없다면 TRUE리턴
}
```

19. Check_Node함수는 리스트에 같은 문자열이 있는지 확인하는 함수이다. 이 함수는 간선리스트를 삽입할 때 같은 문자열이 있는지 파악한다.

20. 매개변수로 전달받은 리스트를 p에 저장하고 p의 끝까지 반복하며 리스트의 문자열과 입력받은 문자열과 비교한다. 만약 중간에 같은 문자열이 존재한다면 FALSE를 리턴하고 같은 문자열이 없다면 TRUE를 리턴한다.

21. Check_Same_String함수는 배열에서 같은 문자열이 있는지 확인하는 함수이다. 이 함수는 정점을 찾을 때 같은 정점이 있는지 확인한다.

22. 문자열 배열을 I부터 입력받은 num값 동안 반복하며 strcmp로 문자열 배열과 입력받은 문자열이 같은지 확인한다. 만약 여기서 같은 값이 존재한다면 FALSE를 리턴하고 같은 값이 없다면 TRUE를 리턴한다.

```
void init(GraphType*g, int VERTICES) {              //그래프를 초기화하는 함수
    int v;
    g->n = 0;
    g->adj_list = (GraphNode**)malloc(sizeof(GraphNode*)*VERTICES); //리스트를 정점의 개수만큼 동적할당
    g->ary = (char**)malloc(sizeof(char*)*VERTICES);                //정점을 동적할당
    for (v = 0; v < VERTICES; v++) {
        g->adj_list[v] = NULL;
    }
}
void insert_vertex(GraphType*g, int VERTICES, char*temp) {      //정점을 삽입하는 함수
    if (((g->n) + 1) > VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->ary[g->n] = temp;                                        //temp로부터 정점 값을 입력받음
    g->n++;                                                     //반복할때마다 정점의 개수 증가
}
```

23. init함수는 그래프를 초기화하는 함수이다. 메인에서 구한 정점 개수인 num값을 매개변수로 전달받는다.
정점의 개수를 나타내는 n을 0으로 설정하고 정점 문자열과 리스트 배열을 num값을 이용하여 동적할당한다. 동적할당 후 정점의 개수만큼 반복하며 리스트의 초기 값을 NULL로 설정한다.

24. insert_vertex함수는 그래프의 정점을 삽입하는 함수이다.
개수가 맞다면 입력받은 temp 정점의 문자열을 그래프의 정점에 삽입한다. 삽입이 끝나면 정점의 개수인 g->n을 증가시킨다.

```
void insert_directed_edge(GraphType*g, Element data) {          //방향그래프에 삽입하는 함수
    GraphNode *node;
    int u, v, i;
    for (i = 0; i < g->n; i++) {                    //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->ary[i], data.start) == 0) u = i;
        if (strcmp(g->ary[i], data.end) == 0) v = i;
    }
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }                                                   //방향그래프이므로 리스트 뒤에 end값만 추가하면 됨
    node = (GraphNode*)malloc(sizeof(GraphNode));       //새로운 노드 생성
    node->vertex = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));  //노드의 문자열을 end의 길이로 동적할당
    strcpy(node->vertex, data.end);                     //end문자열을 복사

    node->link = g->adj_list[u];                        //노드의 link가 adj_list의 처음을 가리킴(insert_first)
    g->adj_list[u] = node;                              //adj_list의 head값이 node가 됨
    //  printf("%d-----%s₩n",u, node->vertex);
}
```

25. insert_directed_edge함수는 방향그래프로 간선을 삽입하는 함수이다.
그래프의 정점을 끝까지 반복하며 입력받은 start와 end값과 비교한다.
strcmp함수로 정점과 문자열을 비교하고 정점의 인덱스 값을 찾는다. 이 인덱스 값을 u와 v값에 각각 저장한다.
만약 이 인덱스 값이 정점의 개수보다 크다면 오류를 나타내고 함수를 종료한다.

26. 새로운 노드를 동적할당하여 생성하고 노드에 값을 삽입한다.
방향 그래프이기 때문에 end문자열에 대해서만 길이+1로 vertex간선을 동적할당하고 strcpy로 end 문자열을 복사한다.

27. 노드의 링크가 리스트의 가장 처음을 가리키도록 한 후 리스트의 처음이 그 노드가 되도록 하여 insert_first연산의 형태로 동작한다.

```
void insert_undirected_edge(GraphType*g, Element data) {        //무방향 그래프에 삽입하는 함수
    GraphNode *node;
    GraphNode *node2;
    int u, v, i;
    for (i = 0; i < g->n; i++) {                    //시작문자열의 인덱스값과 도착 문자열의 인덱스 값을 찾음
        if (strcmp(g->ary[i], data.start) == 0) u = i;
        if (strcmp(g->ary[i], data.end) == 0) v = i;
    }
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode));       //node는 end문자열을 저장
    node2 = (GraphNode*)malloc(sizeof(GraphNode));      //node2는 start문자열을 저장
    if (Check_Node(g->adj_list[u], data.end)) {         //현재 리스트에 end문자열이 존재하는지 확인
        node->vertex = (char*)malloc(sizeof(char)*(strlen(data.end) + 1));
        strcpy(node->vertex, data.end);                 //존재하지 않는다면 node의 vertex를 동적할당하고 값을 복사
        node->link = g->adj_list[u];                    //노드의 link가 리스트의 처음을 가리킴
        g->adj_list[u] = node;                          //리스트의 처음이 node가 될수 있도록 함
    }
    if (Check_Node(g->adj_list[v], data.start)) {   //현재 리스트에 start문자열이 존재하는지 확인
        node2->vertex = (char*)malloc(sizeof(char)*(strlen(data.start) + 1));
        strcpy(node2->vertex, data.start);          //존재하지 않는다면 node의 vertex를 동적할당하고 값을 복사
        node2->link = g->adj_list[v];
        g->adj_list[v] = node2;
    }
}
```

28. insert_undirected_edge함수는 무방향 그래프로 간선을 삽입하는 함수이다. 정점을 끝까지 반복하며 입력받은 start와 end값과 비교한다.
strcmp함수로 정점과 문자열을 비교하고 정점의 인덱스 값을 찾는다. 이 인덱스 값을 u와 v값에 각각 저장한다.
이 인덱스 값이 정점의 개수보다 크다면 오류를 나타내고 함수를 종료한다.

29. 무방향 그래프이기 때문에 노드를 2개 만들어서 하나는 start문자열, 다른 하나는 end문자열을 담을 수 있는 노드로 생성한다.
리스트에 삽입을 할 때는 시작 문자열과 도착 문자열을 저장해야 하는데, 같은 문자열이 리스트에 저장되는 문제가 생겼다.
이 문제를 해결하기 위해 Check_Node함수를 호출하여 리스트에 같은 문자열이 있는지 파악하고 만약 같은 문자열이 없다면 방향그래프에서 했던 것과 같이 insert_first로 리스트에 삽입할 수 있도록 하였다.

30. 무방향 그래프이기 때문에 end문자열뿐만 아니라 start문자열도 비교하여 리스트에 삽입될 수 있도록 한다.

```
void print_adj_list(GraphType*g) {                          //그래프를 형식에 맞게 출력하는 함수
    int i;
    for (i = 0; i < g->n; i++) {
        GraphNode*p = g->adj_list[i];                       //리스트 배열의 값을 p에 저장
        printf("정점 %6s의 인접 리스트 ", g->ary[i]);    //현재 정점 출력
        while (p != NULL) {                                 //p리스트의 끝까지 반복하며 출력
            printf("-> %s ", p->vertex);
            p = p->link;
        }
        printf("₩n");
    }
    printf("₩n");
}
void Delete(GraphType*g) {              //그래프를 동적할당 해제하는 함수
    int i;
    GraphNode *p, *q;
    for (i = 0; i < g->n; i++) {
        p = g->adj_list[i];             //p에 리스트 배열의 값을 입력받음
        q = p;
        if (p != NULL) {                //만약 p가 NULL이면 리스트가 존재하지 않으므로 해제하지 않음
        //  printf("₩n%d, ", i);
            while (p != NULL) {
            //  printf("%s ", p->vertex);
                q = q->link;            //q가 다음링크 주소를 가리킴
                free(p->vertex); //== free(g->adj_list[i].vertex);  //현재 주소 p의 리스트노드 해제
                free(p);    // ==free(g->adj_list[i])              //p의 리스트노드 해제
                p = q;                                             //p에 다음 주소를 가리키는 q저장
            }
        }
    }
    free(g->ary);           //ary정점배열 해제 문자열은 메인에서 temp로 해제함
    free(g->adj_list);      //리스트포인터 해제
    free(g);
}
```

31. print_adj_list함수는 인접리스트 그래프를 형식에 맞게 출력하는 함수이다. 먼저 정점의 개수 g->n만큼 I를 반복하여 그에 맞는 리스트를 p에 저장한다. 그리고 정점을 나타내는 g->ary[i]를 출력한 후 p를 끝까지 반복하며 간선의 정보를 출력한다.

33. Delete함수는 그래프에 대한 메모리를 해제하는 함수이다.
p와 q에 리스트 정보를 저장한 후 만약 p가 존재한다면 제거를 수행한다. 방향그래프의 경우 리스트가 아예 존재하지 않는 경우가 있기 때문이다.

34. p가 존재한다면 먼저 q를 q의 다음링크로 옮긴 후 현재 p의 간선 문자열과 리스트를 제거한다. 모두 제거했다면 p에 다음 노드를 가리키는 q를 삽입한다.
ary정점 배열과 리스트 adj_list를 free하고 마지막으로 그래프 g를 해제하면 모든 동작이 끝난다. ary의 문자열은 메인에서 temp로 제거했기 때문에 Delete함수에서 또 제거하게 되면 오류가 발생한다.

2.4 실행 창

- 7주차 과제 기본 데이터 파일

```
(강남, 양재), (양재, 강남), (양재, 역삼), (역삼, 양재), (강남, 교대), (교대, 양재)
-Undirected Graph-
정점   강남의 인접 리스트 -> 교대 -> 양재
정점   양재의 인접 리스트 -> 교대 -> 역삼 -> 강남
정점   역삼의 인접 리스트 -> 양재
정점   교대의 인접 리스트 -> 양재 -> 강남

<강남, 양재>, <양재, 강남>, <양재, 역삼>, <역삼, 양재>, <강남, 교대>, <교대, 양재>
-Directed Graph-
정점   강남의 인접 리스트 -> 교대 -> 양재
정점   양재의 인접 리스트 -> 역삼 -> 강남
정점   역삼의 인접 리스트 -> 양재
정점   교대의 인접 리스트 -> 양재


C:₩Users₩dmk46₩OneDrive₩바탕 화면₩2학기 수업자료₩자료구조2₩실습₩week7_2₩Debug₩week7
 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

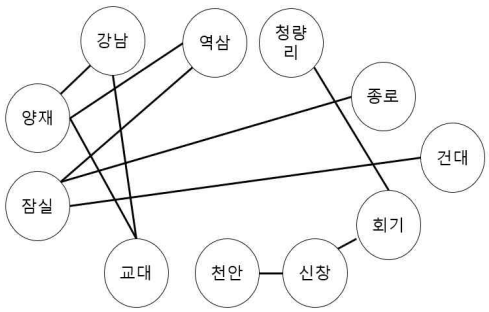
```
data - Windows 메모
파일(F)  편집(E)  서식(
강남 양재
양재 강남
양재 역삼
역삼 양재
강남 교대
교대 양재
```

- 데이터 파일 추가와 3글자 역 입력과 리스트가 비었을 때 확인

```
(강남, 양재), (양재, 강남), (양재, 역삼), (역삼, 잠실), (강남, 교대), (교대, 양재)
, (종로, 잠실), (회기, 청량리), (잠실, 건대), (신창, 회기), (천안, 신창),
-Undirected Graph-
정점   강남의 인접 리스트 -> 교대 -> 양재
정점   양재의 인접 리스트 -> 교대 -> 역삼 -> 강남
정점   역삼의 인접 리스트 -> 잠실 -> 양재
정점   잠실의 인접 리스트 -> 건대 -> 종로 -> 역삼
정점   교대의 인접 리스트 -> 양재 -> 강남
정점   종로의 인접 리스트 -> 잠실
정점   회기의 인접 리스트 -> 신창 -> 청량리
정점 청량리의 인접 리스트 -> 회기
정점   건대의 인접 리스트 -> 잠실
정점   신창의 인접 리스트 -> 천안 -> 회기
정점   천안의 인접 리스트 -> 신창

<강남, 양재>, <양재, 강남>, <양재, 역삼>, <역삼, 잠실>, <강남, 교대>, <교대, 양재>
, <종로, 잠실>, <회기, 청량리>, <잠실, 건대>, <신창, 회기>, <천안, 신창>,
-Directed Graph-
정점   강남의 인접 리스트 -> 교대 -> 양재
정점   양재의 인접 리스트 -> 역삼 -> 강남
정점   역삼의 인접 리스트 -> 잠실
정점   잠실의 인접 리스트 -> 건대
정점   교대의 인접 리스트 -> 양재
정점   종로의 인접 리스트 -> 잠실
정점   회기의 인접 리스트 -> 청량리
정점 청량리의 인접 리스트
정점   건대의 인접 리스트
정점   신창의 인접 리스트 -> 회기
정점   천안의 인접 리스트 -> 신창


C:₩Users₩dmk46₩OneDrive₩바탕 화면₩2학기 수업자료₩자료구조2₩실습₩week7_2₩Debug₩week
```

```
data - Windows 메모장
파일(F)  편집(E)  서식(O)
강남 양재
양재 강남
양재 역삼
역삼 잠실
강남 교대
교대 양재
종로 잠실
회기 청량리
잠실 건대
신창 회기
천안 신창
```

위 데이터의 무방향 그래프



2.5 느낀 점

이번 문제는 인접 리스트를 활용하여 그래프를 표현하는 문제였습니다. 이번 문제도 1번과 같이 정점이 문자열로 바뀌어서 처음엔 헷갈리는 부분이 있었지만 1번에서 했던 것을 이용하여 비슷하게 프로그램을 짤 수 있었던 것 같습니다. 또한 책에 있는 코드를 이용하여 간선 리스트를 작성할 수 있었습니다.

이번 프로그램을 짤 때 책에는 방향그래프만 나와 있었기 때문에 무방향 그래프를 새로 짜보았습니다. 그런데 간선을 삽입할 때 입력받은 간선이 이미 리스트 안에 들어있는 경우, 같은 간선을 리스트에 삽입해버리는 문제가 생겨 한 정점에 같은 간선이 들어있어서 ->강남->강남으로 출력되는 문제가 있었습니다. 이 문제를 해결하기 위해 현재 리스트에 같은 간선이 있는지 확인하는 Check_Node함수를 만들어서 리스트에 같은 문자열이 있는지 파악하고 같은 문자열이 없어야만 삽입이 가능하도록 프로그래밍을 했습니다. 이런 방법을 통해 같은 간선이 들어가는 문제를 해결할 수 있었습니다.

이번 과제를 계기로 인접 리스트를 이용한 그래프에 대해 이해를 할 수 있었습니다. 희소행렬을 리스트로 나타내니 더 쉽게 눈에 들어오고 시간 복잡도도 줄일 수 있었습니다. 이걸 계기로 희소행렬을 리스트로 출력하는 것에 대해 이해할 수 있었습니다. 그리고 책에 나와 있는 코드로만 한 것이 아니라 정점과 간선을 문자열로 바꾸고 프로그래밍하면서 더 많은 것을 배울 수 있었던 기회가 되었던 것 같습니다.

3. 그래프를 하며 느낀 점

자료구조 이론시간에 그래프에 대해 배우면서 처음 배우는 부분이기 때문에 어렵다고 생각했었는데 막상 프로그래밍을 해보니 생각보다 할 만하다고 생각했었습니다. 그래프는 인접 행렬과 인접 리스트로 나타낼 수 있고 이를 또 무방향그래프와 방향그래프로 나눌 수 있다는 것을 알 수 있었습니다. 이를 통해 앞으로 그래프를 어떤 식으로 나타내야할지 생각해볼 수 있었습니다.

특히 1번과 2번 문제에서 같은 파일로 실행 결과를 확인해보니 희소행렬일 때 왜 리스트로 출력을 하는 것이 시간 복잡도 부분에서 더 이득인지 알 수 있었습니다. 또한 크기가 큰 희소행렬을 인접 행렬 그래프로 표현하니 눈에도 잘 띄지 않아 불편하기도 했습니다.
이렇게 인접 행렬 그래프와 인접 리스트 그래프 둘의 차이점을 알고 언제 어떤 그래프를 사용해야 할지 생각해보는 계기가 되었습니다.

아직 매우 초반이라는 것을 알기 때문에 프로그래밍을 다 했다고 여기서 만족하지 않고 더 나아가 어려운 문제도 풀 수 있도록 노력해야겠다고 생각했습니다.