

## 자료구조2 실습 레포트



과목명		자료구조2 실습
담당교수		홍 민 교수님
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20204059
이름		이예빈

# 목 차

---

## 1. 인접 행렬로 구현하는 지하철 노선 그래프

### 1.1 문제 분석

### 1.2 소스 코드

### 1.3 소스 코드 분석

### 1.4 실행창

### 1.5 느낀점

## 2. 연결 리스트로 구현하는 지하철 노선 그래프

### 2.1 문제 분석

### 2.2 소스 코드

### 2.3 소스 코드 분석

### 2.4 실행창

### 2.5 느낀점

## 3. 느낀점

# 1. 인접행렬로 구현하는 지하철 노선 그래프

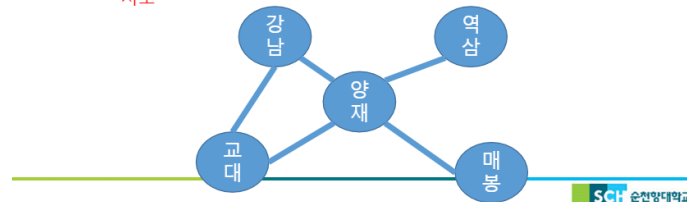
## 인접 행렬

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

- data.txt 파일의 데이터 형식은 에지 정보(정점 정점)로 아래와 같이 구성되어 있음

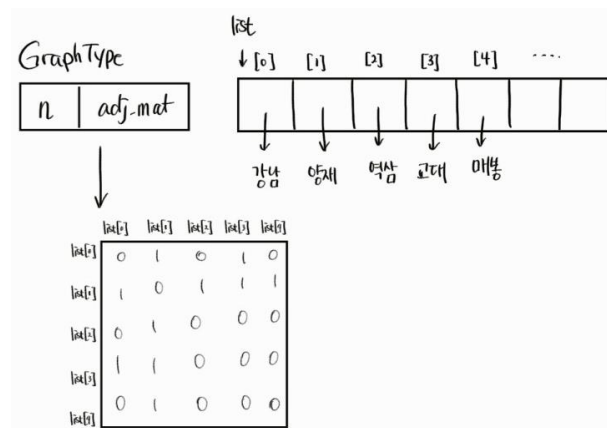
```
강남 양재
양재 강남
역삼 양재
역삼 교대
강남 교대
교대 양재
```

- 아래의 그래프에 대해서 data.txt 파일들을 직접 생성하여 입력받으시오



## 1.1 문제 분석

이 문제는 인접 행렬을 생성하여 지하철 노선 그래프를 표현하는 프로그램이다. 위 그림에 나타나 있는 것과 같이, 양재, 강남, 교대, 역삼, 매봉, 총 5 개의 정점을 가진 그래프는 방향성이 존재하지 않는, 5 개의 간선을 가지는 무방향 그래프이다. 무방향 그래프이지만 예시의 data.txt 파일을 자세히 살펴보면 두 정점 사이의 방향성이 존재한다는 것을 확인할 수 있다. 예를 들어 강남과 양재와 같은 경우, 파일의 첫번째 줄에는 “강남 양재”, 그리고 두번째 줄에는 “양재 강남”이 입력되어 있다. 따라서 모든 정점 사이에 방향이 존재하지는 않지만, 간선을 통해서 두 정점의 양방향성을 갈 수 있도록 구현해서는 안 된다는 것이다. 따라서 두 정점 사이의 입출을 구분할 수 있도록 해야 한다. 이번 과제의 그래프는 5 개의 정점을 가졌으므로, (5 X 5) 크기의 2 차원 배열로 표현을 해야 한다.



2021/10/17

그래프의 구조체 GraphType 는 요소의 개수와 인접 행렬을 가리키는 이중 포인터로 구성된다. 인접 행렬의 각 인덱스 번호는 정점에 대한 문자열을 참조하는데 사용되는 값을 의미한다고 할 수 있다.

## 1.2 소스 코드

```

1  /*
2  →  작성자: .이예빈 (20204059)
3  →  작성일: .2021.10.16
4  →  프로그램명: .인접·행렬로·표현하는·지하철·노선·그래프
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 typedef struct GraphType {
12     int n; // 정점의 개수
13     int **adj_mat;
14 } GraphType;
15
16 // 그래프 초기화. (인접행렬·동적·할당)
17 void init(GraphType *g, int size) {
18     int v, w;
19     g->n = size;
20     g->adj_mat = (int **) malloc(sizeof(int *) * size);
21     for (v = 0; v < size; v++) {
22         g->adj_mat[v] = (int *) malloc(sizeof(int) * size);
23         for (w = 0; w < size; w++)
24             g->adj_mat[v][w] = 0; // 배열의 모든 요소 값을 0으로 초기화
25     }
26 }
27
28 // 간선 삽입·연산 (방향 그래프)
29 void insert_edge(GraphType *g, int start, int end) {
30     if (start >= g->n || end >= g->n) {
31         fprintf(stderr, "그래프: 정점 번호 오류");
32         return;
33     }
34     g->adj_mat[start][end] = 1;
35 }
36
37 void print_adj_mat(GraphType *g) {
38     int r, c;
39     for (r = 0; r < g->n; r++) {
40         for (c = 0; c < g->n; c++)
41             printf("%2d", g->adj_mat[r][c]);
42         printf("\n");
43     }
44 }
45
46 int searchName(char **list, int size, char *key) {
47     int i;
48     for (i = 0; i < size; i++)
49         if (strcmp(list[i], key) == 0) return 1;
50     return 0;
51 }
52
53 int findIndex(char **list, int size, char *key) {
54     int i;
55     for (i = 0; i < size; i++)
56         if (strcmp(list[i], key) == 0) return i;
57     return 0;
58 }
59
60
61
62

```

```

63 //그래프의 모든 메모리 해제
64 void destroy_graph(GraphType* g) {
65     int v;
66     for (v = 0; v < g->n; v++) {
67         free(g->adj_mat[v]);
68     }
69 }
70
71 int main() {
72     FILE* fp;
73     GraphType* g;
74     char* list[100];
75     int size = 0;
76     char first[20], second[20];
77     int start, end;
78
79     g = (GraphType*) malloc(sizeof(GraphType));
80
81     //파일 읽기 list 문자열 포인터 배열에 저장
82     fp = fopen("data.txt", "rt");
83     while (!feof(fp)) {
84         fscanf(fp, "%s%s", first, second);
85         if (searchName(list, size, first) == 0) {
86             list[size] = (char*) malloc(strlen(first) + 1);
87             strcpy(list[size++], first);
88         }
89         if (searchName(list, size, second) == 0) {
90             list[size] = (char*) malloc(strlen(second) + 1);
91             strcpy(list[size++], second);
92         }
93     }
94
95     // (size x size) 크기의 인접 행렬 동적 할당
96     init(g, size);
97
98     rewind(fp);
99     // 다시 파일 읽기 인접 행렬에 저장 (인덱스 구하여 간선 삽입)
100    while (!feof(fp)) {
101        fscanf(fp, "%s%s", first, second);
102        // first, second 문자열이 저장돼있는 list의 인덱스 start, end 구하기
103        start = findIndex(list, size, first);
104        end = findIndex(list, size, second);
105        insert_edge(g, start, end); // 간선 (start, end) 삽입
106    }
107
108    print_adj_mat(g); // 그래프 인접 행렬 출력
109    destroy_graph(g); // 그래프 인접 행렬 메모리 해제
110    free(g); // 그래프 메모리 해제
111    fclose(fp); // 파일 닫기
112    return 0;
113 }

```

### 1.3 소스 코드 분석

```

/*
→  작성자::이예빈 (20204059)
→  작성일::2021.10.16
→  프로그램명::인접·행렬로·표현하는·지하철·노선·그래프
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef struct GraphType{
→  int n; → // 정점의 개수
→  int **adj_mat;
} GraphType;

```

1. 주석에 작성자, 작성일, 프로그램명을 작성한다. 프로그램에 필요한 헤더 파일들을 추가하고, 인접 행렬을 나타내는 GraphType 구조체를 선언한다. GraphType 구조체는 삽입된 정점의 개수를 나타내는 n, 그리고 2 차원 배열의 인접 행렬의 시작 주소를 가리킬 이중 포인터 adj\_mat 을 멤버로 가진다.

main 함수를 살펴보기 전에 프로그램에서 사용되는 각 함수들을 살펴보도록 하자.

```

// 그래프 초기화 (인접행렬 동적 할당)
void init(GraphType *g, int size){
→  int v, w;
→  g->n = size;
→  g->adj_mat = (int**) malloc(sizeof(int*) * size);
→  for(v = 0; v < size; v++){
→      g->adj_mat[v] = (int*) malloc(sizeof(int) * size);
→      for(w = 0; w < size; w++){
→          g->adj_mat[v][w] = 0; // 배열의 모든 요소 값을 0으로 초기화
→      }
→  }
}

```

2. 인접 행렬을 동적 할당하고 그래프에 대해 초기화를 하는 init 함수이다. 그래프를 가리키는 구조체 포인터 g 와 size 변수를 매개 변수로 전달 받아, g 가 가리키는 n 멤버에 size 의 값을 대입하고, adj\_mat 이중 포인터에는 size 크기만큼의 int 정수형 포인터를 할당한다. 그리고 size 번 반복하여 각 포인터에 정수형 크기의 메모리를 size 만큼 할당하고, 각 요소의 값을 0 으로 초기화한다.

```

// 간선 삽입 연산 (방향 그래프)
void insert_edge(GraphType* g, int start, int end) {
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;
}

```

3. 간선을 삽입하는 insert\_edge 함수이다. 그래프를 가리키는 포인터 g, 간선의 시작점에 대한 값 start 와 끝점에 대한 값 end 을 매개 변수로 전달 받아 두 정점에 대해 방향을 갖는 간선을 삽입하도록 한다. 우선적으로 start 와 end 의 값이 g 의 n 멤버보다 큰 경우에는 정점 번호 오류가 났다는 출력문과 함께 함수를 종료하도록 한다. 그렇지 않은 경우, 0 으로 초기화되어 있었던 2 차원 배열의 start 번 행에 대한 end 번 열에 1 을 대입한다.

```

void print_adj_mat(GraphType* g) {
    int r, c;
    for (r = 0; r < g->n; r++) {
        for (c = 0; c < g->n; c++)
            printf("%2d", g->adj_mat[r][c]);
        printf("\n");
    }
}

```

4. 그래프 행렬의 전체 요소를 출력하는 print\_adj\_mat 함수이다. 한 정점에서 정점으로까지의 방향성이 있는 간선은 1 이, 간선이 없는 경우에는 0 이 출력된다.

```

int searchName(char** list, int size, char* key) {
    int i;
    for (i = 0; i < size; i++)
        if (strcmp(list[i], key) == 0) return 1;
    return 0;
}

```

5. size 만큼의 문자형 이중 포인터 배열 list 에서 key 를 통해 전달 받은 문자열과 동일한 값이 존재하는지를 탐색하는 searchName 함수이다. size 번만큼 반복하여 list 포인터 배열의 각 요소와 key 값을 비교하여 strcmp 함수를 통해 얻은 값이 0 인 경우에는 1 을 반환하여 탐색에 성공했음을 알린다. 반복문이 종료된 후에는 0 을 반환하여 key 값과 동일한 문자열을 갖는 요소가 list 포인터 배열에 없음을 전달한다.



```

int·findIndex(char**·list,·int·size,·char*·key)·{
    int·i;
    for·(i·=·0; i·<·size; i++)·{
        if·(strcmp(list[i],·key)·==·0)·return·i;
    }
    return·0;
}

```

6. searchName 함수와 구현이 동일하지만, key 값과 동일한 문자열이 발견된 경우 해당 인덱스를 반환하는 findIndex 함수이다.

```

//·그래프의·모든·메모리·해제
void·destroy_graph(GraphType*·g)·{
    int·v;
    for·(v·=·0; v·<·g->n; v++)·{
        free(g->adj_mat[v]);
    }
}

```

7. 그래프의 모든 메모리를 해제하는 destroy\_graph 함수이다. 인접 행렬의 각 행에 대해 할당된 정수형 메모리들을 모두 해제하도록 한다.

이제 main 함수의 동작을 살펴보자.

```

int·main()·{
    FILE**fp;
    GraphType*·g;
    char*·list[100];
    int·size·=·0;
    char·first[20],·second[20];
    int·start,·end;

    g·=·(GraphType*)malloc(sizeof(GraphType));
}

```

8. 파일 포인터 fp, 그래프 구조체 포인터 g, 100 개의 값을 저장할 수 있는 문자열 포인터 배열 list, 0 으로 초기화된 size 정수형 변수, 두 문자열을 임시로 저장할 first, second 문자형 배열, 그리고 출발하는 정점인 start 와 끝 정점인 end 를 선언한다.

9. 포인터 g 에 GraphType 구조체의 크기만큼 메모리를 동적 할당한다.

```

→ //파일.읽어.list.문자열.포인터.배열에.저장
→ fp=fopen("data.txt","rt");
→ while(!feof(fp)){
→     fscanf(fp,"%s%s",first,second);
→     if(searchName(list,size,first)==0){
→         list[size]=(char*)malloc(strlen(first)+1);
→         strcpy(list[size++],first);
→     }
→     if(searchName(list,size,second)==0){
→         list[size]=(char*)malloc(strlen(second)+1);
→         strcpy(list[size++],second);
→     }
→ }

```

10. data.txt 파일을 읽기 모드로 연다. 파일의 끝을 가리킬 때까지 읽으며, 각 줄에 저장되어 있는 두 문자열을 first 와 second 배열로 읽는다.

11. 문자열을 읽은 후에는, list 포인터 배열, list 에 현재 저장된 문자열의 개수를 의미하는 size, 그리고 list 에 새로 추가할 문자열(first 와 second)을 인수로 전달하여 searchName 함수를 호출한다. 해당 list 에 새로 추가하고자 하는 문자열과 동일한 문자열을 가진 요소가 없어 0 을 반환하게 되면, list 에 새로 추가하도록 한다.

12. list 의 size 번 인덱스에 strlen 함수를 이용해 얻은 문자열의 길이와 마지막 NULL 값을 저장하기 위해 1 을 더한 크기만큼 동적 할당을 한 후, 문자열을 해당 메모리에 복사하도록 한다. 이후, 다음 인덱스에 새로운 문자열을 추가하기 위해 size 는 1 증가시키도록 한다.

```

→ //(size*x*size)크기의.인접.행렬.동적.할당
→ init(g,size);
→ rewind(fp);

```

13. 파일을 모두 읽어 list 에 저장한 후, 그래프를 초기화한다. 그래프에 대한 포인터 g 와 앞서 얻었던 list 의 크기인 size 를 인수로 전달해 init 함수를 호출하고 그래프를 초기화한다. 해당 함수는 위에 설명한 바와 같이, 인접 행렬을 동적 할당하는 역할을 한다.

14. rewind 함수를 사용해 파일 포인터를 파일의 시작 위치로 이동시킨다.

```

→ //다시.파일.읽어.인접.행렬에.저장.(인덱스.구하여.간선.삽입)
→ while(!feof(fp)){
→     fscanf(fp,"%s%s",first,second);
→     //first,second.문자열이.저장돼있는.list의.인덱스.start,end.구하기
→     start=findIndex(list,size,first);
→     end=findIndex(list,size,second);
→     insert_edge(g,start,end); //간선.(start,end).삽입
→ }

```

15. 다시 파일을 읽으며, 인접 행렬의 각 정점에 대해 간선을 삽입하도록 한다. 파일에서 각 줄에 저장된 두 지하철 역 문자열을 읽으며, list 에서 위치한 해당 문자열의 인덱스 번호를 findIndex 함수를 통해 얻어 각각 start 와 end 에 저장한다.


16. 그리고 insert\_edge 함수를 호출하여 두 지하철 역 사이의 간선(방향성이 존재하는)을 삽입하도록 한다.

```
→ print_adj_mat(g); → → //그래프.인접.행렬.출력
→ destroy_graph(g); → → //그래프.인접.행렬.메모리.해제
→ free(g); → → → //그래프.메모리.해제
→ for(i:=0;i<.size;i++)//문자열.포인터.배열.메모리.해제
→ → free(list[i]);
→ fclose(fp); → → → //파일.닫기
→ return 0;
}0
```

17. 마지막으로, 그래프의 인접 행렬을 출력한다. destroy\_graph 함수를 호출하여 동적 할당된 인접 행렬의 메모리들을 해제하고, 이후 포인터 g 에 할당된 그래프에 대한 메모리도 해제하도록 한다. 문자열 포인터 배열 list 에도 할당된 메모리들을 모두 해제하고, 파일을 닫고 프로그램 실행을 종료한다.

## 1.4 실행창

data.txt 파일은 다음과 같이 두 정점을 의미하는 지하철 역 문자열이 각 줄에 두 개씩 저장되어 있다. 전체적으로는, 방향이 존재하지 않는 무방향 그래프이지만 한 정점에 대한 입출이 명확히 구분되어야 한다. 따라서 a에서 b로 가는 간선이 있다면, b에서 a로 가는 간선도 함께 저장되어 있어야 한다. 강남, 양재, 역삼, 교대, 매봉에 대한 간선의 관계가 아래와 같이 data.txt 파일에 나타나 있다.


 data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

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

실행 결과는 다음과 같다.

(list 에 차례대로 저장되어 있는 지하철 역은 0 번 인덱스인 강남부터, 1 번 양재, 2 번 역삼, 3 번 교대, 4 번 매봉 순이다.)

 Microsoft Visual Studio 디버그 콘솔

```
0 1 0 1 0
1 0 1 1 1
0 1 0 0 0
1 1 0 0 0
0 1 0 0 0
```

C:\Users\이예빈\Desktop\CSE 2021-2\자료구조2\

### 1.5 느낀점

이번 과제를 하면서 가장 고심해야 했던 부분은 인접 행렬에 어떠한 방법으로 각 정점을 참조할것이나 였다. 인접 행렬은 기본적으로 정수인 인덱스 값을 사용하여 한 정점으로부터 다른 정점으로의 간선 여부를 표시한다. 따라서 정수 인덱스를 적절히 활용하는 방법을 사용하여 문자열인 지하철의 정점을 나타내는 것이 적합하다고 판단하였다. 결국 list 문자열 배열에 중복되지 않도록 각 문자열들을 저장하고, 인덱스를 통해 이 문자열들을 참조하는 방법으로 정점을 표현하였다. 이러한 방법은 인덱스인 키 값들을 이용하는 '해싱'이라는 기법을 사용한 것인데, 나중에 자료구조 시간에 배우게 될 해시 함수를 자세히 배우면서 지금 배우고 있는 것들을 더 자유롭게 사용해 보고 싶다.

## 2. 연결 리스트로 구현하는 그래프

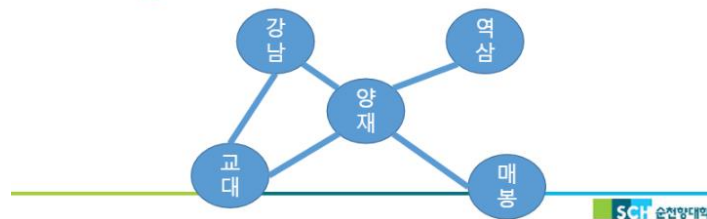
### 인접 리스트

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

- data.txt 파일의 데이터 형식은 에지 정보(정점 정점)로 아래와 같이 구성되어 있음

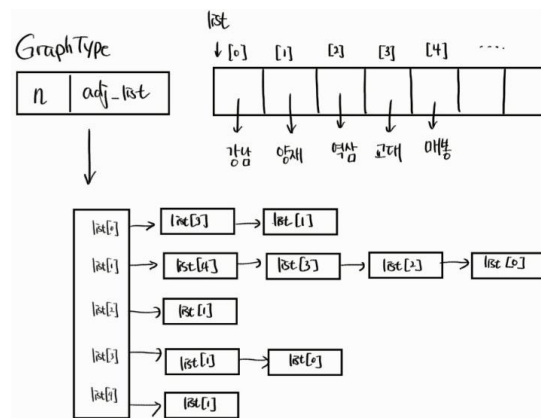
```
강남 양재
양재 강남
양재 역삼
역삼 양재
강남 교대
교대 양재
```

- 아래의 그래프에 대해서 data.txt 파일들을 직접 생성하여 입력받으시오



### 2.1 문제 분석

1번 과제와 동일한 문제이지만, 그래프의 표현을 인접 행렬이 아닌 인접 리스트로 나타내는 것이 차이점이다. 구현의 방법이 다른 것은 한 정점에서 다른 정점으로의 간선을 2차원 배열이 아닌 노드들로 구성되어 있는 리스트로 각각 표현한 것이다. 이때에도, 1번문제와 같이 배열의 인덱스 정수 값을 활용한 해싱 기법을 토대로 정점을 표현하게 된다.



그래프 구조체 GraphType은 정점의 개수와 리스트 포인터 배열을 가리키는 이중 포인터로 구성된다. 리스트 포인터 배열에도 역시나 인덱스 번호가 부여되므로, 각 번호는 list 문자열 배열의 인덱스 번호를 이용하여 정점을 표현하게 되고, 리스트의 각 노드들은 해당 정점이 향하는 정점

2021/10/17

들을 의미한다. 각 노드들 또한 해싱 기법을 통해 정점 문자열을 참조하게 된다.

## 2.2 소스 코드

```

1  /*
2   *   작성자: ·이예빈 (20204059)
3   *   작성일: ·2021.10.16
4   *   프로그램명: ·인접·리스트로·표현하는·지하철·노선·그래프
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 typedef struct GraphNode { // 인접·리스트의·노드
12     int vertex;
13     struct GraphNode *link;
14 } GraphNode;
15
16 typedef struct GraphType { // 인접·리스트
17     int n;
18     GraphNode **adj_list; // 인접·리스트·배열
19 } GraphType;
20
21 // 그래프·초기화·(인접리스트·동적·할당)
22 void init(GraphType *g, int size) {
23     int i;
24     g->n = size;
25     // size·크기의·포인터·배열·할당·(GraphNode를·가리키는·포인터)
26     g->adj_list = (GraphNode**) malloc(sizeof(GraphNode*) * size);
27     for (i = 0; i < size; i++) // NULL값으로·초기화
28         g->adj_list[i] = NULL;
29 }
30
31 // 간선(u,v)·삽입·연산
32 void insert_edge(GraphType *g, int u, int v) {
33     GraphNode *node;
34     if (u >= g->n || v >= g->n) {
35         fprintf(stderr, "그래프: 정점·번호·오류");
36         return;
37     }
38     node = (GraphNode*) malloc(sizeof(GraphNode));
39     node->vertex = v;
40     // 리스트의·맨·처음에·삽입
41     node->link = g->adj_list[u]; // 노드의·링크·필드에·u번째·리스트·주소·삽입
42     g->adj_list[u] = node; // u번째·리스트의·헤더에·node·주소·삽입
43 }
44
45 int searchName(char **list, int size, char *key) {
46     int i;
47     for (i = 0; i < size; i++)
48         if (strcmp(list[i], key) == 0) return 1;
49     return 0;
50 }
51
52 int findIndex(char **list, int size, char *key) {
53     int i;
54     for (i = 0; i < size; i++) {
55         if (strcmp(list[i], key) == 0) return i;
56     }
57     return 0;
58 }
59

```



```

60 void print_adj_list(GraphType* g, char** list) {
61     GraphNode* v;
62     int i = 0;
63     for (i = 0; i < g->n; i++) {
64         // 정점 list[i] 에 대하여
65         printf("[%s] -> ", list[i]);
66         v = g->adj_list[i];
67         while (v != NULL) {
68             printf("[%s] -> ", list[v->vertex]); // vertex에 해당하는 문자열 출력
69             v = v->link;
70         }
71         printf("\b\b\b...\n");
72     }
73 }
74
75 // 그래프의 모든 메모리 해제
76 void destroy_graph(GraphType* g) {
77     GraphNode* p, *next;
78     int v = 0;
79
80     for (v = 0; v < g->n; v++) {
81         p = g->adj_list[v];
82         while (p != NULL) { // 각 노드 해제
83             next = p->link;
84             free(p);
85             p = next;
86         }
87     }
88 }
89
90 int main() {
91     FILE* fp;
92     GraphType* g;
93     char* list[100];
94     int size = 0, i;
95     char first[20], second[20];
96     int start, end;
97
98     g = (GraphType*) malloc(sizeof(GraphType));
99
100     // 파일 읽어 list 문자열 포인터에 저장
101     fp = fopen("data.txt", "rt");
102     while (!feof(fp)) {
103         fscanf(fp, "%s %s", first, second);
104         if (searchName(list, size, first) == 0) {
105             list[size] = (char*) malloc(strlen(first) + 1);
106             strcpy(list[size++], first);
107         }
108         if (searchName(list, size, second) == 0) {
109             list[size] = (char*) malloc(strlen(second) + 1);
110             strcpy(list[size++], second);
111         }
112     }
113 }

```

```

114 → //·(size·X·size)·크기의·인접·행렬·동적·할당
115 → init(g,·size);
116
117 → rewind(fp);
118 → //·다시·파일·읽어·인접·행렬에·저장·(인덱스·구하여·간선·삽입)
119 → while·(!feof(fp))·{
120 →     → fscanf(fp,·"%s·%s",·first,·second);
121 →     → //·first,·second·문자열이·저장돼·있는·list의·인덱스·start,·end·구하기
122 →     → start·=·findIndex(list,·size,·first);
123 →     → end·=·findIndex(list,·size,·second);
124 →     → insert_edge(g,·start,·end);
125 → }
126
127 → print_adj_mat(g,·list);→//·인접·리스트·출력
128 → destroy_graph(g);→→→//·그래프의·인접·리스트에·대한·모든·메모리·해제
129 → for·(i·=·0;i·<·size;i++)
130 →     → free(list[i]);→→→//·list·문자열·배열·메모리·해제
131 → free(g);→→→→//·그래프·메모리·해제
132 → fclose(fp);→→→→//·파일·닫기
133 → return·0;
134 }

```

## 2.3 소스 코드 분석

```

/*
 * 작성자: ··이예빈 (20204059)
 * 작성일: ··2021.10.16
 * 프로그램명: ··인접·리스트로·표현하는·지하철·노선·그래프
 */

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

```

1. 주석에 작성자, 작성일, 프로그램명을 작성한다. 이후 프로그램에 필요한 헤더 파일들을 추가한다.

```

typedef struct GraphNode { //·인접·리스트의·노드
    int vertex;
    struct GraphNode* link;
} GraphNode;

typedef struct GraphType { //·인접·리스트
    int n;
    GraphNode** adj_list; //·인접·리스트·배열
} GraphType;

```

2. 인접 리스트로 표현하는 그래프를 구현하기 위한 구조체를 선언한다. 인접 리스트의 각 노드를 표현하는 GraphNode 구조체는 정점 번호와 다음 노드를 가리키는 link 포인터를 멤버로 갖는다. GraphType 구조체는 리스트의 개수와 각 리스트의 첫 헤더 노드를 가리키는 이중 포인터 adj\_list를 멤버로 갖는다.

```

//·그래프·초기화·(인접리스트·동적·할당)
void init(GraphType* g, int size) {
    int i;
    g->n = size;
    //·size·크기의·포인터·배열·할당·(GraphNode를·가리키는·포인터)
    g->adj_list = (GraphNode**) malloc(sizeof(GraphNode*) * size);
    for (i = 0; i < size; i++) //·NULL·값으로·초기화
        g->adj_list[i] = NULL;
}

```

3. 그래프를 초기화하며 인접 리스트를 동적으로 할당하는 init 함수이다. 그래프 포인터 g와 리스트의 개수를 의미하는 size를 매개 변수로 전달 받는다. 포인터 g가 가리키는 그래프의 n 멤버에는 size의 값을 대입하고, adj\_list 이중 포인터에는 size개수 만큼의 GraphNode 포인터를 할당한다. 이후, 다시 size의 값만큼 반복문을 돌리며 adj\_list 포인터들을 각각 NULL값으로 초기화한다.

```

// 간선 (u, v) 삽입. 연산
void insert_edge(GraphType* g, int u, int v) {
    GraphNode* node;
    if (u >= g->n || v >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    node = (GraphNode*) malloc(sizeof(GraphNode));
    node->vertex = v;
    // 리스트의 맨 처음에 삽입
    node->link = g->adj_list[u]; // 노드의 링크 필드에 u번째 리스트 주소 삽입
    g->adj_list[u] = node; // u번째 리스트의 헤더에 node 주소 삽입
}

```

4. 정점  $u$  에서 정점  $v$  로 향하는 간선을 삽입하는 `insert_edge` 함수이다. 그래프 포인터  $g$  와 두 개의 정점  $u$  와  $v$  를 매개 변수로 전달받는다. `GraphNode` 형 포인터 `node` 를 선언하고,  $u$  와  $v$  중  $g$  의  $n$  멤버 값보다 같거나 큰 경우에는 정점 번호에 오류가 있다는 출력문을 띄우고 해당 함수를 종료한다.

5. 정상적인 정점 번호들을 전달 받은 경우라면, `node` 포인터에 `GraphNode` 노드 크기의 메모리를 동적 할당하고, `vertex` 멤버에는  $v$  의 값을 대입한다.

6. 이후 해당 `node` 의 `link` 필드에는  $u$  번째 리스트의 주소를 삽입하고, 다시  $u$  번째 리스트의 헤더에 `node` 주소를 삽입함으로써 리스트의 맨 처음에 삽입하도록 구현을 한다.

```

int searchName(char** list, int size, char* key) {
    int i;
    for (i = 0; i < size; i++)
        if (strcmp(list[i], key) == 0) return 1;
    return 0;
}

int findIndex(char** list, int size, char* key) {
    int i;
    for (i = 0; i < size; i++) {
        if (strcmp(list[i], key) == 0) return i;
    }
    return 0;
}

```

7. 1 번 문제와 동일하게 구현된 `searchName` 과 `findIndex` 함수이다. `list` 문자열 배열에서 `key` 에 해당하는 문자열 값이 존재하는 경우, `searchName` 은 1 을 반환하게 되고, `findIndex` 는 탐색에 성공했을 때의 `list` 의 인덱스 번호를 반환하도록 한다.

```

void print_adj_list(GraphType* g, char** list) {
    GraphNode* v;
    int i = 0;
    for (i = 0; i < g->n; i++) {
        // 정점 list[i]에 대하여
        printf("[%s]-> ", list[i]);
    }
}

```

8. 그래프의 인접 리스트들을 모두 출력하는 print\_adj\_list 함수이다. 각 리스트를 순회하며 정점들을 출력하는데 사용할 GraphNode 형 포인터 v, 그리고 반복문의 제어 변수 i를 선언한다. 변수 i는 for 문 내에서 0 부터 g->n 이전까지 증가하며, 해당 i 번째 list 문자열 배열에 해당하는 문자열(지하철 역)을 출력하도록 한다.

```

    v = g->adj_list[i];
    while (v != NULL) {
        printf("%s-> ", list[v->vertex]); // vertex에 해당하는 문자열 출력
        v = v->link;
    }
    printf("\b\b\b...\n");
}

```

9. v는 각 포인터 배열에 접근한다. 첫번째 주소 값을 먼저 대입하고, NULL에 도달할 때까지 각 노드의 vertex에 저장된 인덱스에 해당하는 list 문자열의 정점 값을 출력하도록 한다.

10. 마지막에 출력되는 "->"를 제거하기 위해 backspace 문자를 사용한다.

```

// 그래프의 모든 메모리 해제
void destroy_graph(GraphType* g) {
    GraphNode* p, *next;
    int v = 0;

    for (v = 0; v < g->n; v++) {
        p = g->adj_list[v];
        while (p != NULL) { // 각 노드 해제
            next = p->link;
            free(p);
            p = next;
        }
    }
}

```

11. 인접 리스트의 모든 메모리를 해제하는 destroy\_graph 함수이다. 각 포인터 배열에 접근하여, 리스트의 모든 메모리들을 해제한다.

다음으로 main 함수의 동작을 살펴보자. Main 함수는 1번문제와 모든 과정이 동일하므로, 자세한 설명을 생략한다.

```

int main() {
    FILE* fp;
    GraphType* g;
    char* list[100];
    int size = 0, i;
    char first[20], second[20];
    int start, end;

    g = (GraphType*) malloc(sizeof(GraphType));

    // 파일 읽어 list 문자열 포인터에 저장
    fp = fopen("data.txt", "rt");
    while (!feof(fp)) {
        fscanf(fp, "%s%s", first, second);
        if (searchName(list, size, first) == 0) {
            list[size] = (char*) malloc(strlen(first) + 1);
            strcpy(list[size++], first);
        }
        if (searchName(list, size, second) == 0) {
            list[size] = (char*) malloc(strlen(second) + 1);
            strcpy(list[size++], second);
        }
    }
}

```

12. 변수의 선언부터, 파일 입출력 그리고 list 문자열 배열에 각 정점을 저장하는 것까지, 모두 1번 문제와 동일하다.

```

// size * size 의 인접 리스트 동적 할당 (포인터 배열 공간 할당)
init(g, size);

rewind(fp);
// 다시 파일 읽어 인접 리스트에 저장 (인덱스 구하여 간선 삽입)
while (!feof(fp)) {
    fscanf(fp, "%s%s", first, second);
    // first, second 문자열이 저장돼있는 list의 인덱스 start, end 구하기
    start = findIndex(list, size, first);
    end = findIndex(list, size, second);
    insert_edge(g, start, end); // 간선 (start, end) 삽입
}

```

13. 위 과정도 1번 문제와 동일하다. 그래프를 초기화하여 인접 리스트에 동적 할당을 하고, 다시 파일을 읽으며, 문자열에 대한 인덱스 번호를 구하여 간선을 삽입하도록 한다.

```

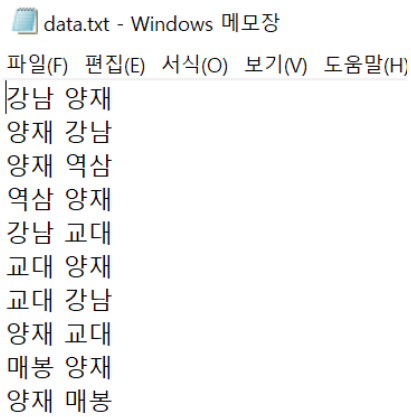
print_adj_mat(g, list); // 인접 리스트 출력
destroy_graph(g); // 그래프의 인접 리스트에 대한 모든 메모리 해제
for (i = 0; i < size; i++)
    free(list[i]); // list 문자열 배열 메모리 해제
free(g); // 그래프 메모리 해제
fclose(fp); // 파일 닫기
return 0;
}

```

14. 인접 리스트를 출력하고, 프로그램에서 사용된 모든 메모리를 해제하고, 파일을 닫은 뒤 프로그램을 종료한다.

## 2.4 실행창

data.txt 파일에 저장되어 있는 데이터는 1번문제와 동일하다.

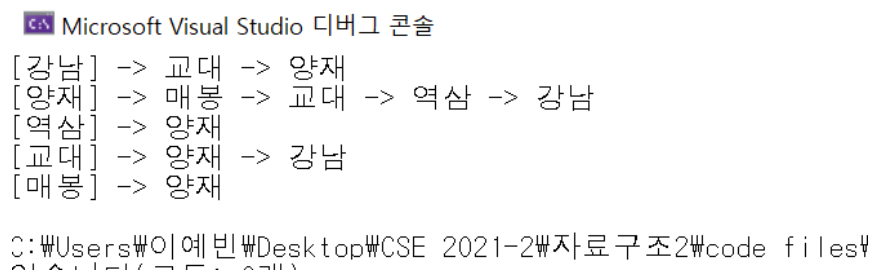


data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

강남 양재  
양재 강남  
양재 역삼  
역삼 양재  
강남 교대  
교대 양재  
교대 강남  
양재 교대  
매봉 양재  
양재 매봉

인접 리스트가 출력된 실행 결과는 다음과 같다.



Microsoft Visual Studio 디버그 콘솔

[강남] -> 교대 -> 양재  
[양재] -> 매봉 -> 교대 -> 역삼 -> 강남  
[역삼] -> 양재  
[교대] -> 양재 -> 강남  
[매봉] -> 양재

C:\Users\이예빈\Desktop\CSE 2021-2\자료구조2\code files\

## 2.5 느낀점

인접 리스트는 그래프를 표현하는 방법이 인접 행렬과는 매우 다르다는 것을 이번 과제를 통해 깨닫게 되었다. 인접 행렬을 통해서는 0과 1이라는 두 정수를 통해 간선에 의한 정점 사이의 연결 여부와 방향을 확인할 수 있다. 그에 반해 인접 리스트는 해당 정점에서 시작하여 다른 정점으로 연결되는 간선을 표현할 때, 직접 해싱을 이용하여 문자열을 출력하기에 용이하고 시각적으로 나타내는 데 있어 더욱 직관적으로 표현할 수 있다. 물론 정점들 사이에 연결된 간선이 많이 존재하는 밀집한 그래프(dense graph)의 경우, 리스트에 삽입되는 노드들의 개수도 많아지기 때문에 이러한 경우에는 인접 행렬로 나타내기에 더욱 적합하다. 그러나 이번 과제와 같이, 정점 사이의 연결성이 도드라지게 많지 않기 때문에 인접 리스트로 표현하는 것이 훨씬 좋은 방법인 것 같다.



### 3. 느낀점

---

선형 구조인 리스트부터 시작하여, 선형 구조가 아닌 계층 구조의 이진 트리, 그리고 노드들 간의 관계를 나타내는 그래프까지, 자료 구조들을 하나하나씩 배워 나가는 것이 매우 즐겁게 느껴진다. 수많은 데이터들을 수집하고 저장하고 이를 효율적으로 관리하는데 있어서 다양한 자료 구조들이 쓰인다는 것을 알게 되며, 앞으로 배울 것들이 더욱 기대가 된다. 또 앞으로 더욱 많은 종류의 그래프들을 배우게 될텐데, 인접 행렬과 인접 리스트에 대한 깊은 이해를 토대로 현재까지 배운 깊이 우선 탐색과 너비 우선 탐색 이외에 그래프를 탐색하며 정렬하는 여러 알고리즘들을 배워나가고 싶다.