

자료구조2 실습

12주차 과제



제출일	21.11.24	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2 실습	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

| 목 차 |

1. 그래프: Floyd 최단 경로 알고리즘

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

2. 그래프: 위상 정렬

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

3. 선택 정렬 프로그램

- 3.1 문제 분석
- 3.2 소스 코드
- 3.3 소스 코드 분석
- 3.4 실행 창
- 3.5 느낀 점

4. 과제를 하며 느낀 점

1. 그래프

Floyd 최단 경로 알고리즘

- 428 페이지에 있는 프로그램 11.11의 Floyd의 최단 경로 프로그램을 참고하여 data.txt 파일에 입력되어 있는 정점과 간선의 정보를 가지고 초기 상태 및 426 페이지의 모든 결과를 순서에 맞게 출력하라.

1.1 문제 분석

조건 1 동적할당을 통하여 더 많은 정점이 입력될 경우 삽입될 수 있게 함

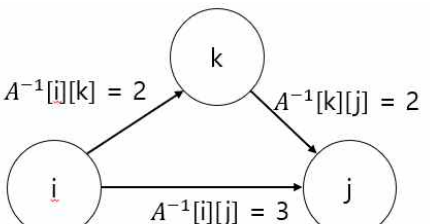
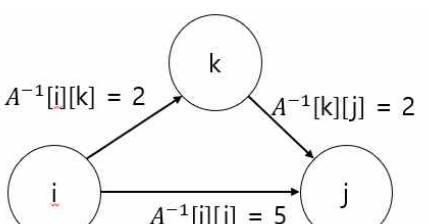
조건 2 업데이트 된 배열의 부분에 *로 표시할 것

조건 3 data.txt파일에는 간선과 가중치가 있고 직접 생성하여 입력받음

- 이 문제는 입력받은 그래프를 Floyd 알고리즘을 사용하여 최단 거리를 찾는 문제이다. Floyd알고리즘은 그래프에 존재하는 모든 정점 사이의 최단 경로를 한 번에 모두 찾아주는 알고리즘이다.

Floyd알고리즘은 정점 I에서 j로 가는 경로를 두 가지로 나누어 생각할 수 있다. 정점 k를 거쳐서 가지 않는 경우와 정점 k를 통과하는 경우로 정점 I에서 j로 가는 길이 더 빠른지 k를 통과하는 경우가 더 빠른지 판단하여 그 가중치 값을 최단 경로 배열에 삽입한다.

한 번에 모든 정점 간의 최단 경로를 구하는 Floyd알고리즘은 3중 반복문을 이용하여 경로 값을 찾는다. 그러므로 시간 복잡도는 $O(n^3)$ 이 될 수 있다. Floyd알고리즘은 간결한 반복 구문을 사용하므로 빠르게 모든 정점 간의 최단 경로를 찾을 수 있다.

k를 거쳐 가는 길이 더 길 때	k를 거쳐 가는 길이 더 빠를 때
 <p>$A^{-1}[i][k] = 2$ $A^{-1}[k][j] = 2$ $A^{-1}[i][j] = 3$</p> <p>$A^{-1}[i][j] < A^{-1}[i][k] + A^{-1}[k][j]$</p> <p>$A^{-1}[i][j] = A^{-1}[i][j]$</p>	 <p>$A^{-1}[i][k] = 2$ $A^{-1}[k][j] = 2$ $A^{-1}[i][j] = 5$</p> <p>$A^{-1}[i][j] > A^{-1}[i][k] + A^{-1}[k][j]$</p> <p>$A^{-1}[i][j] = A^{-1}[i][k] + A^{-1}[k][j]$</p>

1.2 소스 코드

```
/*
 * 학번 : 20184612
 * 학과 : 컴퓨터소프트웨어학과
 * 이름 : 김동민
 * 파일명: Floyd의 최단 경로 알고리즘
 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define INF 1000

typedef struct GraphType {
    int n; //정점의 개수
    int **weight; //가중치 그래프
}GraphType;

typedef struct Floyd {
    int A; //가중치를 담은 변수
    int check; //간선이 새로 생기면 TRUE
}Floyd;
Floyd *data; //Floyd형 이중포인터를 최단경로 배열로 사용

void init(GraphType *g, int n); //그래프 초기화 함수
void insert_vertex(GraphType *g, int num); //정점을 삽입하는 함수
void insert_edge(GraphType *g, int start, int end, int weight); //무방향 그래프로 간선을 삽입하는 함수
void printA(GraphType *g); //data배열을 형식에 맞게 출력하는 함수
void floyd(GraphType *g); //Floyd 알고리즘 함수
void Delete(GraphType *g); //그래프 삭제함수

int main() {
    GraphType *g; //그래프 변수
    FILE *fp; //파일포인터
    int max = -1; //가장 큰 정점을 담은 max변수
    int temp1, temp2, weight, i; //파일로부터 값을 입력받을 temp와 weight
    char c; //어떤 동작을 할지 파일로부터 입력받은 문자 c

    fp = fopen("data.txt", "r"); //data파일을 읽기 형식으로 open
    if (!fp) {
        printf("file not open");
        return 0;
    }
    g = (GraphType*)malloc(sizeof(GraphType)); //그래프 동적할당 생성
    while (!feof(fp)) {
        fscanf(fp, "%c", &c); //파일 끝까지 값을 입력
        if (c == 'v') {
            fscanf(fp, "%d", &temp1); //입력받은 문자가 v하면 정점 삽입 동작
            if (max < temp1) max = temp1; //정점중에 가장 큰 값을 찾을
        }
    }
    max++; //가장 큰값에서 1증가
    init(g, max); //max값으로 그래프 초기화
    rewind(fp); //파일포인터를 다시 앞으로 옮김
    for (i = 0; i < max; i++) {
        insert_vertex(g, max); //정점 삽입
    }
    while (!feof(fp)) {
        fscanf(fp, "%c", &c);
        if (c == 'e') {
            fscanf(fp, "%d%d", &temp1, &temp2, &weight); //시작과 끝값, 가중치 값을 입력
            insert_edge(g, temp1, temp2, weight); //간선 삽입
        }
    }
    floyd(g); //floyd함수 호출
    Delete(g); //그래프 삭제
    fclose(fp); //파일포인터 닫음
    return 0;
}
```

```

void Init(GraphType *g, int n) {
    int i, j;
    g->n = 0; //정점의 개수 초기화
    g->weight = (int **)malloc(sizeof(int*)*n); //가중치 그래프 n[정점]로 할당할
    for (i = 0; i < n; i++) {
        g->weight[i] = (int*)malloc(sizeof(int)*n);
        for (j = 0; j < n; j++) {
            g->weight[i][j] = INF; //가중치 그래프를 INF로 초기화
        }
    }

    g->data = (float **)malloc(sizeof(float)*n); //data[]용도인자를 할당할
    for (i = 0; i < n; i++) {
        g->data[i] = (float*)malloc(sizeof(float)*n);
    }
}

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, int weight) { //무방향 그래프로 간선을 삽입하는 함수
    if (start > g->n || end > g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }

    g->weight[start][end] = weight; //무방향 그래프이기 때문에 start와 end를
    g->weight[end][start] = weight; //end와 start를 둘 다 저장
    g->weight[start][start] = g->weight[end][end] = 0; //자기 자신은 0으로 초기화
    g->data[start][end].check = g->data[end][start].check = TRUE; //서로가 생긴 간선의 check값을 TRUE
    g->data[start][start].check = g->data[end][end].check = TRUE; //서로가 생긴 간선의 check값을 TRUE
}

void PrintK(GraphType *g) { //data[]를 출력하여 현재 출력하는 함수
    int i, j;
    printf(" ");
    for (i = 0; i < g->n; i++) printf(" %3d ", i); //정점을 나타내는 정점 출력
    printf("\n");
    printf("-----><-----\n");
    for (i = 0; i < g->n; i++) {
        printf(" %3d ", i); //정점을 나타내는 정점 출력
        for (j = 0; j < g->n; j++) {
            if (g->data[i][j].A == INF) { //방향 간선이 존재하지 않는다면
                printf(" %3d ", ' '); //공백 출력
            }
            else { //간선이 존재한다면
                if (g->data[i][j].check) { //서로가 생긴 간선이려면 +출력
                    printf(" %3d ", g->data[i][j].A);
                    g->data[i][j].check = FALSE; //check값을 다시 FALSE로 바꿔줌
                }
                else printf(" %3d ", g->data[i][j].A); //현재 현재 간선이려면 그냥 출력
            }
        }
        printf("\n");
    }
    printf("-----><-----\n");
}

void Floyd(GraphType *g) { //Floyd 알고리즘 함수
    int i, j, k;
    for (i = 0; i < g->n; i++) { //data[]의 초기값을 가중치 할당된 weight값을 가짐
        for (j = 0; j < g->n; j++) {
            g->data[i][j].A = g->weight[i][j];
        }
    }

    printf("< 초기 상태 >\n");
    PrintK(g);

    for (k = 0; k < g->n; k++) {
        for (i = 0; i < g->n; i++) {
            for (j = 0; j < g->n; j++) {
                //방향 (에서 1로 가는 길보다 k를 거쳐가는 길의 더 빠르다면
                if (g->data[i][k].A + g->data[k][j].A < g->data[i][j].A) {
                    g->data[i][j].A = g->data[i][k].A + g->data[k][j].A; //여기서 k, i에서 1로 가는 길의 최종 값을 갱신
                    g->data[i][j].check = TRUE; //서로가 생긴 간선으로 check값을 TRUE
                }
            }
        }
    }

    printf("< 최종 정점-정점 >\n", g); //최종 정점들 지나온지 출력
    PrintK(g); //최종 그래프 출력
}

void Delete(GraphType *g) { //그래프 삭제함수
    int i;
    for (i = 0; i < g->n; i++) {
        free(g->data[i]); //floyd 그래프 data[] 삭제
        free(g->weight[i]); //가중치 그래프 weight[]를 삭제
    }
    free(g->data); //data[]와 weight[] 삭제
    free(g->weight);
    free(g); //그래프 삭제
}

```

1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: Floyd의 최단 경로 알고리즘
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define INF 1000

typedef struct GraphType {
    int n; //정점의 개수
    int **weight; //가중치 그래프
}GraphType;
typedef struct Floyd {
    int A; //가중치를 담을 변수
    int check; //간선이 새로 생기면 TRUE
}Floyd;
Floyd **data; //Floyd형 이중포인터를 최단경로 배열로 사용
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 참을 나타내는 TRUE와 거짓을 나타내는 FALSE를 정의한다. 추가로 가중치 값을 초기화할 때 무한대를 나타내는 INF를 선언한다.

2. 그래프를 나타내는 GraphType구조체를 선언한다.

구조체 안의 변수로는 정점의 개수를 나타내는 n과 가중치 그래프를 저장할 weight이중포인터가 있다. weight는 정점의 개수에 맞게 동적할당하여 사용한다.

3. 최단 경로를 저장할 Floyd구조체를 선언한다.

구조체 안에는 가중치를 담을 변수 A와 업데이트 된 부분을 표시하기 위한 check값을 선언한다. check값은 길이 새로 생긴다면 TRUE가 된다.

4. Floyd형으로 이중포인터를 전역변수로 선언하고 최단경로를 저장한다.

```
void Init(GraphType *g, int n); //그래프 초기화 함수
void insert_vertex(GraphType*g, int num); //정점을 삽입하는 함수
void insert_edge(GraphType* g, int start, int end, int weight); //무방향 그래프로 간선을 삽입하는 함수
void printA(GraphType *g); //data배열을 형식에 맞게 출력하는 함수
void floyd(GraphType*g); //Floyd 알고리즘 함수
void Delete(GraphType*g); //그래프 삭제함수
```

5 필요한 함수를 선언한다.

- init함수는 그래프와 data포인터를 초기화하는 함수이다.
- insert_vertex함수는 그래프에 정점을 삽입하는 함수이다.
- insert_edge함수는 무방향 그래프로 간선을 삽입하는 함수이다.
- PrintA함수는 최단 경로 배열을 형식에 맞게 출력하는 함수이다.
- floyd함수는 floyd알고리즘을 사용하여 최단 거리를 찾고 저장하는 함수이다.
- Delete함수는 그래프와 data배열을 동적할당해제하는 함수이다.

```
int main() {
    GraphType *g;        //그래프 변수
    FILE *fp;            //파일포인터
    int max = -1;         //가장 큰 정점을 담을 max변수
    int temp1, temp2, weight, i; //파일로부터 값을 입력받을 temp와 weight
    char c;               //어떤 동작을 할지 파일로부터 입력받을 문자 c

    fp = fopen("data.txt", "r"); //data파일을 읽기 형식으로 open
    if (!fp) {
        printf("file not open");
        return 0;
    }
    g = (GraphType*)malloc(sizeof(GraphType)); //그래프 동적할당 생성
    while (!feof(fp)) { //파일 끝까지 값을 입력
        fscanf(fp, "%c", &c);
        if (c == 'v') { //입력받은 문자가 v라면 정점 삽입 동작
            fscanf(fp, "%d", &temp1);
            if (max < temp1) max = temp1; //정점중에 가장 큰 값을 찾음
        }
        max++; //가장 큰값에서 1증가
        Init(g, max); //max값으로 그래프 초기화
        rewind(fp); //파일포인터를 다시 앞으로 옮김
        for (i = 0; i < max; i++) {
            insert_vertex(g, max); //정점 삽입
        }
        while (!feof(fp)) {
            fscanf(fp, "%c", &c);
            if (c == 'e') { //입력받은 문자가 e라면 간선 삽입 동작
                fscanf(fp, "%d%d%d", &temp1, &temp2, &weight); //시작과 끝값, 가중치 값을 입력
                insert_edge(g, temp1, temp2, weight); //간선 삽입
            }
        }
    }
    floyd(g); //floyd함수 호출
    Delete(g); //그래프 삭제
    fclose(fp); //파일포인터 닫음
    return 0;
}
```

6. 필요한 변수들을 선언한다.

- g는 그래프변수, fp는 파일포인터, max는 정점 중 가장 큰 값을 찾는다.
- temp1, temp2, weight는 모두 파일로부터 값을 입력받을 임시변수이다.
- c는 v또는 e를 입력받아 어떤 동작을 수행할지 판별한다.

7. data.txt파일을 읽기형식으로 open한다. open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 정점을 삽입하는 동작인 v가 입력 되면 정수 하나를 입력받고 정점 중 가장 큰 값을 찾는다. max에 가장 큰 값이 저장되면 최댓값의 1을 더한 값으로 동적할당을 해야 하므로 max를 1 증가시킨다.

8. init함수로 max값을 전달하여 그래프를 초기화하고 rewind함수로 파일 포인터를 앞으로 다시 옮긴다.

그리고 for문을 이용하여 max값만큼 정점을 삽입한다. 정점의 최대값은 max를 넘길 수 없으므로 max를 매개변수로 전달한다.

9. 다시 파일 끝까지 데이터를 입력받는다. 만약 c의 값이 e라면 간선을 삽입한다. 간선을 삽입할 때 시작 정점, 도착 정점, 가중치의 순으로 파일에 저장되어 있기 때문에 이 순서대로 입력을 받고 insert_edge로 그래프에 삽입한다.

10. floyd함수를 호출하여 모든 정점 사이의 최단 경로를 찾는다.

floyd함수에서는 반복문을 돌며 그래프가 변화하는 과정을 확인할 수 있다.

11. 모든 동작이 끝나면 Delete함수를 호출하여 그래프와 최단 경로 배열을 삭제하고 fclose함수로 파일을 닫은 후 프로그램을 종료한다.

```
void Init(GraphType *g, int n) {
    int i, j;
    g->n = 0; //정점의 개수 초기화
    g->weight = (int **)malloc(sizeof(int*)*n); //가중치 그래프 n값으로 동적할당
    for (i = 0; i < n; i++) {
        g->weight[i] = (int*)malloc(sizeof(int)*n);
        for (j = 0; j < n; j++) {
            g->weight[i][j] = INF; //가중치 그래프를 INF로 초기화
        }
    }
    data = (Floyd **)malloc(sizeof(Floyd*)*n); //data이중포인터를 동적할당
    for (i = 0; i < n; i++) {
        data[i] = (Floyd*)malloc(sizeof(Floyd)*n);
    }
}
```

12. init함수는 그래프와 최단 경로 배열을 초기화하는 함수이다.

정점의 개수를 나타내는 n을 0으로 초기화하고, weight이중포인터를 입력받은 n값으로 동적할당한다. 동적할당하며 weight의 가중치를 INF로 초기화한다.

13. 최단 경로를 나타내는 이중포인터 data변수 또한 n값으로 동적할당한다. 배열로 선언하지 않고 포인터를 동적할당하여 사용했기 때문에 큰 수의 정점이 들어와도 저장할 수 있고, 메모리의 낭비를 줄일 수 있다.


```

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, int weight) { //무방향 그래프로 간선을 삽입하는 함수
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->weight[start][end] = weight; //무방향 그래프이기 때문에 start행 end열
    g->weight[end][start] = weight; //end행 start열 둘 다 저장
    g->weight[start][start] = g->weight[end][end] = 0; //자기 자신은 0으로 초기화
    data[start][end].check = data[end][start].check = TRUE; //새롭게 생긴 간선의 check값을 TRUE
    data[start][start].check = data[end][end].check = TRUE; //새롭게 생긴 간선의 check값을 TRUE
}

```

14. insert_vertex함수는 그래프의 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.

15. insert_edge함수는 무방향그래프로 간선을 삽입하는 함수이다.

매개변수로는 그래프 g와 시작 인덱스인 start와 끝 인덱스인 end와 가중치 weight를 전달받는다.

무방향그래프이기 때문에 start행 end열, end행 start열에 모두 가중치를 삽입하고 자기 자신은 0으로 초기화한다.

또한 새롭게 생긴 간선의 check값을 TRUE로 선언한다.

```

void printA(GraphType *g) { //data배열을 형식에 맞게 출력하는 함수
    int i, j;
    printf(" ");
    for (i = 0; i < g->n; i++) printf(" %3d ", i); //열을 나타내는 정점 출력
    printf("\n");
    printf("-----\n");
    for (i = 0; i < g->n; i++) {
        printf(" %d |", i); //행을 나타내는 정점 출력
        for (j = 0; j < g->n; j++) {
            if (data[i][j].A == INF) { //만약 간선이 존재하지 않는다면
                printf("%3c |", 'x'); //x출력
            }
            else { //간선이 존재한다면
                if (data[i][j].check) { //새롭게 생긴 간선이라면 *출력
                    printf("%2d |", data[i][j].A);
                    data[i][j].check = FALSE; //check값을 다시 FALSE로 바꿔줌
                }
                else printf("%3d |", data[i][j].A); //원래 있던 간선이라면 그냥 출력
            }
        }
        printf("\n");
    }
    printf("-----\n\n");
}

```

16. PrintA함수는 최단 경로 배열 data를 형식에 맞게 출력하는 함수이다. 먼저 열과 행을 나타내는 정점을 출력하고 그래프를 출력한다.

17. 만약 간선이 존재하지 않는 INF라면 x를 출력하여 길이 없음을 표시한다. 만약 간선이 존재한다면 이 간선이 새롭게 생긴건지 원래 있던 길인지 확인한다.

18. 만약 새롭게 업데이트 된 길이라면 data의 check값이 TRUE이므로 *과 가중치를 함께 출력한다. 출력 후에는 check값을 다시 FALSE로 하여 다음에 업데이트 되는 간선과 헷갈리지 않도록 한다.

반대로 원래 있던 길이면 check값이 FALSE이므로 *표시 없이 그냥 가중치만 출력한다.

```
void floyd(GraphType*g) {           //Floyd 알고리즘 함수
    int i, j, k;
    for (i = 0; i < g->n; i++) {      //data의 초기값은 가중치 행렬인 weight값을 가짐
        for (j = 0; j < g->n; j++) {
            data[i][j].A = g->weight[i][j];
        }
    }
    printf("< 초기 상태 >\n");        //초기상태 출력
    printA(g);
    for (k = 0; k < g->n; k++) {
        for (i = 0; i < g->n; i++) {
            for (j = 0; j < g->n; j++) {
                //만약 i에서 j로 바로 가는 길보다 k를 거쳐가는 길이 더 빠르다면
                if (data[i][k].A + data[k][j].A < data[i][j].A) {
                    data[i][j].A = data[i][k].A + data[k][j].A; //i에서 k, k에서 j로 가는 길의 합을 대신 삽입
                    data[i][j].check = TRUE; //새로운 길이 생겼으므로 check값을 TRUE
                }
                //k를 거쳐가지 않는 경우는 그대로
            }
        }
    }
    printf("< %d번 정점 열람 >\n", k); //몇번 정점을 지나는지 출력
    printA(g);                        //floyd그래프 출력
}
```

19. floyd함수는 floyd알고리즘을 이용하여 최단 거리를 찾고 저장한다. 먼저 최단 거리 배열인 data의 A의 초기값을 가중치 행렬인 weight를 삽입한다. 그리고 PrintA함수를 호출하여 초기 상태를 출력한다.

20. Floyd알고리즘은 2차원 배열 data를 3중 반복하여 최단 경로를 찾는다. I에서 j로 가는 길을 찾기 위해 2가지로 나누어 정점 k를 거쳐서 가지 않는 경우와 정점 k를 통과하는 경우 두 가지를 생각한다.

21. 만약 I에서 j로 가는 거리가 I에서 k로 가는 거리 + k에서 j로 가는 거리보다 크다면, 이것이 더 좋은 경로이므로 I에서 k로 가는 거리 + k에서 j로 가는 거리를 I에서 j로 가는 거리에 삽입한다.

만약 거리가 바뀌게 된다면 새롭게 업데이트가 되었다는 의미이므로 data의 check값을 TRUE로 바꾸어준다.

22. 반대로 I에서 j로 가는 거리가 더 빠른 거리라면 값을 그대로 유지한다는 의미이므로 data배열을 변경하지 않고 그대로 둔다.

23. 반복을 할 때마다 정점이 변화하는 과정을 PrintA를 통해 출력한다.

```
void Delete(GraphType*g) {           //그래프 삭제 함수
    int i;
    for (i = 0; i < g->n; i++) {
        free(data[i]);               //floyd 그래프 data 열과
        free(g->weight[i]);          //가중치 그래프 weight열을 삭제
    }
    free(data);                      //data와 weight 삭제
    free(g->weight);
    free(g);                         //그래프 삭제
}
```

24. Delete함수는 그래프와 data를 삭제하는 함수이다.

for문을 정점의 개수만큼 반복하며 data의 열과 그래프의 weight열을 삭제한 후 data, weight 모두 삭제한다.

마지막으로 그래프변수인 g도 삭제한다.

1.4 실행 창

최단 경로 그래프의 변화 과정

< 초기 상태 >

	0	1	2	3	4	5	6						
0	*	0	*	7	x	x	*	3	*10	x			
1	*	7	*	0	*	4	*10	*	2	*	6	x	
2		x	*	4	*	0	*	2	x	x	x		
3		x	*	10	*	2	*	0	*11	*	9	*	4
4	*	3	*	2	x	*11	*	0	x	*	5		
5	*10	*	6	x	*	9	x	*	0	x			
6	x	x	x	*	4	*	5	x	*	0			

< 0번 정점 열림 >

	0	1	2	3	4	5	6	
0		0	7	x	x	3	10	x
1		7	0	4	10	2	6	x
2		x	4	0	2	x	x	x
3		x	10	2	0	11	9	4
4		3	2	x	11	0	*13	5
5		10	6	x	9	*13	0	x
6		x	x	x	4	5	x	0

< 1번 정점 열림 >

	0	1	2	3	4	5	6	
0		0	7	*11	*17	3	10	x
1		7	0	4	10	2	6	x
2		*11	4	0	2	*6	*10	x
3		*17	10	2	0	11	9	4
4		3	2	*6	11	0	*8	5
5		10	6	*10	9	*8	0	x
6		x	x	x	4	5	x	0

< 2번 정점 열림 >

	0	1	2	3	4	5	6	
0		0	7	11	*13	3	10	x
1		7	0	4	*6	2	6	x
2		11	4	0	2	6	10	x
3		*13	*6	2	0	*8	9	4
4		3	2	6	*8	0	8	5
5		10	6	10	9	8	0	x
6		x	x	x	4	5	x	0

데이터 파일

data - Windows 메모

파일(F) 편집(E) 서식(C)

56

*10x

*6x

xx

*9*4

xx*5

*0x

x*0

56

10x

6x

xx

*94

*135

0x

x0

56

10x

6x

*10x

는 길이 더

.A) {

.A: //i에서

길이 생겼으

v0

v1

v2

v3

v4

v5

v6

e017

e043

e0510

e124

e1310

e142

e156

e232

e3411

e359

e364

e465

< 3번 정점 열람 >								
	0	1	2	3	4	5	6	
0	0	7	11	13	3	10	*17	
1	7	0	4	6	2	6	*10	
2	11	4	0	2	6	10	* 6	
3	13	6	2	0	8	9	4	
4	3	2	6	8	0	8	5	
5	10	6	10	9	8	0	*13	
6	*17	*10	* 6	4	5	*13	0	
< 4번 정점 열람 >								
	0	1	2	3	4	5	6	
0	0	* 5	* 9	*11	3	10	* 8	
1	* 5	0	4	6	2	6	* 7	
2	* 9	4	0	2	6	10	6	
3	*11	6	2	0	8	9	4	
4	3	2	6	8	0	8	5	
5	10	6	10	9	8	0	13	
6	* 8	* 7	6	4	5	13	0	
< 5번 정점 열람 >								
	0	1	2	3	4	5	6	
0	0	5	9	11	3	10	8	
1	5	0	4	6	2	6	7	
2	9	4	0	2	6	10	6	
3	11	6	2	0	8	9	4	
4	3	2	6	8	0	8	5	
5	10	6	10	9	8	0	13	
6	8	7	6	4	5	13	0	
< 6번 정점 열람 >								
	0	1	2	3	4	5	6	
0	0	5	9	11	3	10	8	
1	5	0	4	6	2	6	7	
2	9	4	0	2	6	10	6	
3	11	6	2	0	8	9	4	
4	3	2	6	8	0	8	5	
5	10	6	10	9	8	0	13	
6	8	7	6	4	5	13	0	

1.5 느낀 점

이번 과제는 최단 경로 알고리즘 중 하나인 Floyd알고리즘을 사용하여 최단 경로를 구하는 문제였습니다. 처음에 Floyd알고리즘에 대해 설명을 들을 때는 머리속에서는 이해가 되었어도 직접 해보려고 하니 잘 되지 않던 부분들이 있었습니다. 하지만 이번 과제로 알고리즘에 대해 완벽히 이해할 수 있는 계기가 되었습니다.

처음 최단 경로 배열을 구현했을 때는 책에 나와 있는 것과 마찬가지로 int형 2차원 배열 하나만 이용하여 최단 경로를 구했었습니다. 하지만 그렇게 하게 되면 문제에서 요구하는 새롭게 업데이트 된 부분을 표시할 수가 없었고 저는 이를 해결하기 위해 check라는 정수형 변수 하나를 더 추가하여 구조체로 만들었습니다. 이를 통해 새롭게 업데이트 된 부분을 표시할 수 있었습니다.

이번 문제는 책에 있는 코드를 이용하여 프로그래밍을 했었기에 생각보다 쉽게 할 수 있었던 것 같습니다. 나아가 Floyd알고리즘에 대해 공부할 수 있었고 알고리즘의 진행 과정에 대해 생각해 볼 수 있는 계기가 되었습니다. 관련된 공부를 더 해서 앞으로 Floyd알고리즘이 어디에 사용될 수 있을지 생각해 보고 필요할 때 쉽게 사용이 가능하도록 노력하겠습니다.

2. 그래프

위상 정렬

- 433 페이지의 프로그램 11.13을 참고하여 data.txt에 저장된 정점과 인접 리스트의 데이터에 위상 정렬 알고리즘을 사용하여 데이터에 대한 위상 순서를 출력하라

2.1 문제 분석

조건 1 위상 정렬 알고리즘을 사용하여 위상 순서 출력을 함

조건 2 그래프에 대해서 data.txt파일들을 직접 생성하여 입력받음

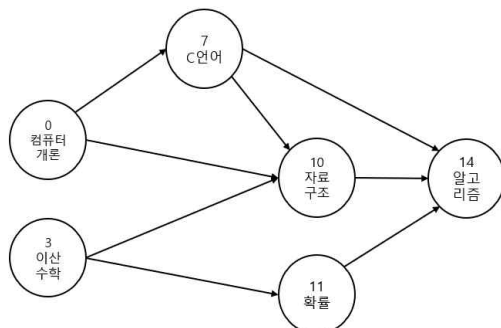
조건 3 정점에는 정점 정수와 과목명이 저장되어 있음

- 이 문제는 인접 리스트로 저장된 그래프를 위상 정렬을 이용하여 순서대로 출력하는 프로그램이다. 이번 data파일에는 정점 번호가 순서대로 있지 않기 때문에 그래프에 정점을 저장할 수 있는 구조체 배열을 하나 더 생성하여 정점 데이터를 저장하고, 인덱스 번호를 통해 접근이 가능하도록 한다.

그래프의 정점 배열에서 인덱스 값을 쉽게 찾기 위해 찾고 싶은 정점의 인덱스 값을 반환하는 함수를 하나 만들고 필요할 때마다 사용하도록 한다.

위상정렬을 구현하기 위해선 스택을 이용한다. 진입 차수가 0인 정점을 스택에 담고 하나씩 꺼내어 그 꺼낸 정점에서 갈 수 있는 정점의 진입 차수를 1씩 줄이면서 알고리즘을 수행한다. 위상정렬은 진입 차수가 0인 정점부터 차례대로 삭제하며 모든 정점이 선택, 삭제 될 때까지 반복을 수행한다. 만약 모든 정점의 진입차수가 0이 된다면 알고리즘이 제대로 수행된 것이라고 할 수 있다.

파일에 있는 데이터의 그래프



정점의 진입 차수

In degree 배열 (진입 차수)					
0	0	1	3	1	3
컴퓨터개론	이산수학	C언어	자료구조	확률	알고리즘

2.2 소스 코드

[illegible]

```

println("배열에 들어갑니다.");
for (i = 0; i < count; i++) {
    printf("%d - %d - %d\n", array[i].num, array[i].str); //배열을 순회하며 출력
}
printf("-----\n");
free_array(a); //배열을 할당 해제
Delete(a); //그래프 삭제
fclose(stdin); //입력 종료 후 닫음
return 0;
}

//최소 비용 경로 찾기
void find_index(int list[], int first, int count) {
    int i;
    for (i = 0; i < count; i++) {
        if (list[i].num == first) return i; //이전 숫자가 같은 경우 앞의 경우와 같은 번호의 인덱스 반환
    }
    return -1; //이전 같은 인덱스가 없다는 의미임
}

//노드 삽입
void insert_node(breadthmap *b, int v) { //그래프 초기화 할때
    int i;
    array = 0; //배열을 초기화
    array_list = (breadthmap *) malloc(sizeof(breadthmap)); //배열의 노드 초기화
    for (i = 0; i < v; i++) {
        array_list[i] = 0; //배열에 0을 삽입
    }
    array = (vertex *) malloc(sizeof(vertex) * v); //배열에 vertex 초기화
}

//노드 삽입
void insert_vertex(breadthmap *b, int v, int index, char *str) { //노드 삽입할때
    if ((array == 0) || v > v) {
        fprintf(stderr, "그래프 : 노드와 개수 초과\n");
        return;
    }
    array[array].num = index; //배열에 vertex의 인덱스 삽입
    array[array].str = str;
    array++; //배열에 개수 증가
}

//노드 삽입
void insert_edge(breadthmap *b, int start, int end) { //노드 삽입할때, 인접 리스트를 삽입
    breadthmap *node;
    int u = 0, v = 0;
    u = find_index(array, start, array); //배열에서 find_index 함수를 통해 인덱스값을 출력
    v = find_index(array, end, array);
    //printf("%d, %d\n", u, v);
    if (u == v) { //자기 자신에 연결
        fprintf(stderr, "그래프 : 자기 자신에 연결\n");
        return;
    }
    node = (breadthmap *) malloc(sizeof(breadthmap)); //노드 할당
    node->num = array[u].num; //노드의 vertex값에 인덱스 삽입
    node->next = array[v].str;
    node->link = array_list[u]; //insert_first 함수
    array_list[u] = node; //인접 리스트에 인접한 vertex의 인덱스값을 기록하고
    //노드들의 개수 증가
}

//노드 삭제
void delete_node(breadthmap *b) {
    int i;
    breadthmap *node;
    int count = 0, n = 0;
    int in_degree = 0; //in_degree(array == 0)
    //배열에 인접한 노드 개수를 계산
    for (i = 0; i < array; i++) {
        in_degree[i] = 0;
    }
    for (i = 0; i < array; i++) {
        breadthmap *node = array_list[i]; //배열에서 노드의 인덱스값
        while (node != NULL) {
            n = find_index(array, node->num, array); //배열에서 인접한 노드의 인덱스값을 출력
            in_degree[n]++; //인접한 노드의 인접한 in_degree 증가
            node = node->link;
        }
    }
    for (i = 0; i < array; i++) {
        printf("%d - %d - %d\n", i, in_degree[i]);
    }
}

//노드 삭제
void delete_node(breadthmap *b) {
    int i;
    for (i = 0; i < array; i++) {
        if (in_degree[i] == 0) {
            printf("배열에 들어갑니다.");
            printf("-----\n");
            while (array_list[i]) {
                //배열에 들어간 노드의 인덱스값을 출력
                int n;
                n = find_index(array, array_list[i].num, array); //배열에서 인접한 노드의 인덱스값을 출력
                node = array_list[i]; //배열에서 인접한 노드의 인덱스값을 출력
                while (node != NULL) {
                    n = find_index(array, node->num, array); //배열에서 인접한 노드의 인덱스값을 출력
                    in_degree[n]--; //인접한 노드의 인접한 in_degree 감소
                    if (in_degree[n] == 0) {
                        //배열에서 인접한 노드의 인덱스값을 출력
                        node = node->link;
                    }
                }
            }
            printf("-----\n");
            free(in_degree); //in_degree 배열 삭제
            free(array); //배열 삭제
            printf("배열에 들어갑니다.");
            return (i == array);
        }
    }
}

//노드 삭제
void delete(breadthmap *b) {
    int i;
    for (i = 0; i < array; i++) {
        free(array_list[i].str); //배열에서 인접한 노드의 인덱스값을 출력
        free(array_list[i]); //배열에서 인접한 노드의 인덱스값을 출력
    }
    //배열에서 인접한 노드의 인덱스값을 출력
    free(array); //배열에서 인접한 노드의 인덱스값을 출력
    free(array_list); //배열에서 인접한 노드의 인덱스값을 출력
    free(a); //배열에서 인접한 노드의 인덱스값을 출력
}

```

2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 위상정렬 알고리즘 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS

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

#define TRUE 1
#define FALSE 0

typedef struct GraphNode { //그래프 노드 구조체 선언
    char *subject;        //정점 문자열 변수
    int vertex;            //정점 정수 변수
    struct GraphNode *link; //다음 노드를 가리키는 링크
} GraphNode;

typedef struct Vertex { //정점의 정보를 저장할 구조체
    int num;             //정점 정수 변수
    char *str;           //정점 문자열 변수
} Vertex;

typedef struct GraphType { //그래프 타입 구조체 선언
    int n;                //정점의 개수 n선언
    Vertex *ary;           //정점의 정보를 저장할 Vertex자료형 포인터 선언
    GraphNode **adj_list; //링크리스트를 저장할 그래프 노드 포인터 선언
} GraphType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다. 참을 나타내는 TRUE와 거짓을 나타내는 FALSE를 정의한다. 추가로 가중치 값을 초기화할 때 무한대를 나타내는 INF를 선언한다.

2. 그래프의 노드를 저장할 GraphNode구조체를 선언한다. 구조체 안에는 정점의 정보를 담은 subject문자열과 vertex정수 변수가 있고 다음노드를 가리킬 link변수가 있다.

3. 정점의 정보를 저장할 Vertex구조체를 선언한다. 정점의 정보를 담기 위해 정점 문자열과 정수를 담은 변수를 선언한다.

4. 그래프를 나타내는 GraphType구조체를 선언한다.

구조체 안의 변수로는 정점의 개수를 나타내는 n과 정점의 정보를 저장할 Vertex자료형 ary포인터, 링크리스트를 저장할 adj_list를 선언한다. 정점의 개수로 동적할당 하기 때문에 ary변수를 통해 인덱스 값을 찾는다.

```

int Find_index(Vertex list[], int find, int count); //정점 배열에서 인덱스 값을 찾는 함수
void graph_init(GraphType *g, int n); //그래프 초기화 함수
void insert_vertex(GraphType*g, int v, int index, char *str); //정점 삽입함수
void insert_edge(GraphType*g, int start, int end); //간선 삽입함수, 인접 리스트에 삽입
int topo_sort(GraphType *g); //위상정렬 함수
void Delete(GraphType *g); //그래프 삭제 함수

```

5. 필요한 함수들을 선언한다.

- Find_index함수는 정점 배열에서 입력받은 정점 값의 인덱스 값을 반환한다.
- Graph_init함수는 그래프에 대한 정보를 초기화하는 함수이다.
- insert_vertex함수는 그래프에 정점을 삽입하는 함수이다.
- insert_edge함수는 인접 리스트 그래프에 간선을 삽입하는 함수이다.
- topo_sort함수는 위상정렬 알고리즘을 사용하여 데이터를 출력하는 함수이다.
- Delete함수는 그래프에 대한 정보를 삭제하는 함수이다.

```

typedef int element;
typedef struct { //스택 타입 구조체 선언
    element *stack; //element형 스택 포인터
    int Stack_size; //스택 사이즈 변수
    int top; //현재 스택의 위치 변수
} StackType;

void init(StackType *s, int n) { //스택 초기화 함수
    s->top = -1; //top값을 -1로 초기화
    s->stack = (element*)malloc(sizeof(element)*n); //n값으로 스택 배열 동적할당
    s->Stack_size = n; //스택의 크기를 n값으로 저장
}

int is_empty(StackType *s) { //스택이 공백상태인지 확인
    return (s->top == -1); //top값이 -1이면 TRUE리턴
}

int is_full(StackType*s) { //스택이 포화상태인지 확인
    return(s->top == (s->Stack_size - 1)); //top값이 size-1이면 TRUE리턴
}

```

6. 위상정렬을 할 때 필요한 스택을 구현한다.

element형으로 스택포인터를 개수에 맞게 동적할당하여 사용하고 개수를 저장할 Stack_size를 선언하고 스택의 가장 위에 있는 값을 가리키는 top을 선언한다.

7. init함수를 사용하여 스택에 있는 변수를 초기화한다.

top값을 -1로 초기화하고 stack 포인터를 입력받은 n값으로 동적할당한다. 그리고 그 n값을 Stack_size에 저장하여 스택에서 사용할 수 있도록 한다.

8. is_empty함수는 스택이 공백상태인지 확인하는 함수이다.

만약 top값이 -1이라면 스택이 비었음을 의미하므로 TRUE를 리턴한다.

9. is_full함수는 스택이 포화상태인지 확인하는 함수이다. top값이

Stack_size-1과 같다면 스택이 꽉 찬 상태이므로 TRUE를 리턴한다.


```

void push(StackType*s, element item) {    //스택에 값을 삽입하는 함수
    if (is_full(s)) {
        fprintf(stderr, "스택 포화 에러\n");
        return;
    }
    s->stack[++(s->top)] = item;    //top을 증가시키고 item변수를 스택에 삽입
}

element pop(StackType*s) {    //스택에서 pop하는 함수
    if (is_empty(s)) {
        fprintf(stderr, "스택 공백 에러");
        exit(1);
    }
    return s->stack[s->top--];    //top값 인덱스를 리턴하고 감소
}

```

10. push함수는 스택에 값을 삽입하는 함수이다.

먼저 스택이 포화상태인지 확인하는 is_full함수를 호출하고 만약 is_full함수가 TRUE를 리턴했다면 스택이 포화되었음을 알리고 함수를 종료한다. 스택에 자리가 있다면 top값을 증가시킨 후 item을 스택에 삽입한다.

11. pop함수는 스택에 가장 마지막으로 들어온 값을 반환하는 함수이다.

먼저 스택에 공백상태인지 확인하는 is_empty함수를 호출하고 스택이 공백상태가 아니라면 top인덱스에 있는 스택 값을 리턴하고 top값을 감소시킨다.

```

int main() {
    FILE *fp;    //파일포인터
    GraphType *g;    //그래프 변수
    char str[20], *stemp;    //파일로부터 문자열을 입력받을 변수
    int temp1, temp2;    //파일로부터 정수를 입력받을 변수
    char c;    //어떤 동작을 할지 입력
    int count = 0, i;

    fp = fopen("data.txt", "r");    //파일을 읽기형식으로 open
    if (!fp) {
        printf("file not opne");
        return 0;
    }
    while (!feof(fp)) {    //파일끝까지 반복
        fscanf(fp, "%c", &c);
        if (c == 'v') {    //v는 정점삽입을 나타냄
            fscanf(fp, "%d%s", &temp1, str);
            count++;    //정점의 개수 증가
        }
    }

    g = (GraphType*)malloc(sizeof(GraphType));    //그래프 동적할당 생성
    rewind(fp);    //파일포인터를 처음으로 옮김
    graph_init(g, count);    //count개로 그래프 초기화
    while (!feof(fp)) {
        fscanf(fp, "%c", &c);    //파일 데이터 입력
        if (c == 'v') {    //v면 정점 삽입 동작 수행
            fscanf(fp, "%d%s", &temp1, str);
            stemp = (char*)malloc(sizeof(char)*(strlen(str)+1));    //stemp를 str의 길이로 동적할당
            strcpy(stemp, str);    //str값을 stemp에 복사
            insert_vertex(g, count, temp1, stemp);    //정점 삽입
        }
    }
}

```


12. 필요한 변수들을 선언한다.

- fp는 파일포인터, g는 인접 리스트 그래프 변수이다.
- str과 temp1, temp2는 파일로부터 데이터를 입력받을 때 사용하고 stemp는 동적할당하여 데이터를 삽입할 때 사용한다.

13. data.txt파일을 읽기형식으로 open한다. open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는다. 먼저 문자형 변수 c에 e또는 v를 입력받는다. v이면 정점 삽입 동작을 수행하고 e면 간선을 삽입한다. v가 입력되었다면 count의 개수를 증가시키며 정점의 개수를 파악한다.

14. 그래프 변수 g를 동적할당하여 생성하고, rewind함수로 파일포인터를 앞으로 옮긴 후 graph_init함수를 이용하여 그래프를 초기화한다. 초기화할 때는 정점의 개수 count를 전달하여 개수에 맞게 동적할당한다.

15. 다시 파일의 처음부터 끝까지 반복하며 데이터를 입력받는다.

만약 v가 입력되었다면 데이터를 입력받았다면 stemp를 str의 길이+1로 동적할당한 후 strcpy함수로 문자열을 복사한다.

그리고 insert_vertex함수를 이용하여 정점 정수와 문자열을 전달하고 정점을 삽입하도록 한다.

```
rewind(fp); //파일포인터를 처음으로 옮김
while (!feof(fp)) {
    fscanf(fp, "%c", &c);
    if (c == 'e') { //e면 간선 삽입 동작 수행
        fscanf(fp, "%d%d", &temp1, &temp2);
        insert_edge(g, temp1, temp2); //간선 삽입 insert_edge호출
    }
}

printf("< 데이터 >\n");
printf("-----\n");
for (i = 0; i < count; i++) {
    printf("%-2d - %s\n", g->ary[i].num, g->ary[i].str); //위상정렬 전 정점 정보 출력
}
printf("-----\n");
topo_sort(g); //위상정렬 함수 호출
Delete(g); //그래프 삭제
fclose(fp); //파일 포인터 닫음
return 0;
```

16. 정점 삽입이 모두 완료되었다면 rewind함수를 이용하여 다시 파일포인터를 처음으로 옮긴 후 feof함수로 파일의 끝까지 다시 입력받는다.

만약 e가 입력되었다면 간선 삽입 동작을 수행한다. 각 정점 정수 값을 파일로부터 입력받고, 이 값을 insert_edge함수로 전달하여 간선이 삽입될 수 있도록 한다.

17. 정점의 개수만큼 for문을 이용하여 반복하며 위상정렬을 하기 전 정점을 출력한다. g의 ary배열은 정점에 대한 정보를 담고 있기 때문에 이를 이용하여 위상정렬 전 어떤 정점이 존재하는지 출력한다.

그리고 topo_sort함수를 호출하여 위상정렬을 수행하도록 한다.

18. Delete함수는 그래프에서 동적할당된 부분들을 삭제할 수 있는 함수이다. 동적할당된 부분을 메모리 해제하고 fclose함수로 파일포인터를 닫은 후 프로그램을 종료한다.

```
int Find_index(Vertex list[], int Find, int count) {    //정점 배열에서 인덱스 값을 찾는 함수
    int i;
    for (i = 0; i < count; i++) {
        if (list[i].num == Find) return i; //정점 정수가 같은 값을 찾으면 현재 인덱스 리턴
    }
    return -1;    //만약 같은 인덱스가 없다면 -1리턴
}

void graph_init(GraphType *g, int n) { //그래프 초기화 함수
    int v;
    g->n = 0;    //정점의 개수 초기화
    g->adj_list = (GraphNode**)malloc(sizeof(GraphNode*)*n);    //링크리스트 초기화
    for (v = 0; v < n; v++) {
        g->adj_list[v] = NULL; //초기값에 NULL삽입
    }
    g->ary = (Vertex*)malloc(sizeof(Vertex)*n);    //정점 ary배열 초기화
}
```

19. Find_index함수는 정점 배열에서 입력받은 정수가 존재하는 지 확인하고 만약 존재한다면 그 인덱스 값을 반환하는 함수이다.

20. 리스트의 처음부터 정점의 개수 count까지 반복하며 찾아야 하는 정수 값인 find가 존재하는지 확인한다. 만약 find가 존재한다면 그 인덱스 값 I를 반환하고 만약 배열에 값이 존재하지 않는다면 -1을 반환한다.

21. graph_init함수는 그래프에 있는 변수들을 초기화하는 함수이다.

먼저 정점의 개수를 나타내는 g->n을 0으로 초기화하고 링크리스트를 나타내는 adj_list를 정점의 개수 n의 값으로 동적할당한 후 그 값에 NULL값을 삽입한다.

22. 그래프의 ary 포인터 또한 정점의 정보를 저장해야 하기 때문에 정점의 개수 n값을 이용하여 동적할당한다.

```

void insert_vertex(GraphType*g, int v, int index, char *str) { //정점 삽입함수
    if (((g->n) + 1) > v) {
        fprintf(stderr, "그래프 : 정점의 개수 초과");
        return;
    }
    g->ary[g->n].num = index; //ary정점 배열에 정점 값 삽입
    g->ary[g->n].str = str;
    g->n++; //정점의 개수 증가
}

void insert_edge(GraphType*g, int start, int end) { //간선 삽입함수, 인접 리스트에 삽입
    GraphNode *node;
    int u = 0, v = 0; //배열의 시작 값과 끝값 인덱스 저장 변수

    u = Find_index(g->ary, start, g->n); //ary에서 Find_index함수를 통해 인덱스값을 찾음
    v = Find_index(g->ary, end, g->n);
    // printf("%d, %d\n", u, v);
    if (u >= g->n || v >= g->n) { //만약 인덱스 값이 정점 개수보다 크다면 종료
        fprintf(stderr, "그래프 : 정점 번호 오류");
        return;
    }
    node = (GraphNode*)malloc(sizeof(GraphNode)); //노드를 새로 생성

    node->vertex = g->ary[v].num; //노드의 vertex값에 ary값 삽입
    node->subject = g->ary[v].str;

    node->link = g->adj_list[u]; //insert_first연산 수행
    g->adj_list[u] = node; //첫 노드의 링크가 원래 리스트를 가리키고
    //리스트의 처음이 node가 됨
}

```

23. insert_vertex함수는 그래프에 정점을 삽입하는 함수이다.

매개변수로 정점의 정수값(index)과 문자열(str)을 입력받는다. v는 정점의 개수를 나타낸다.

24. 만약 삽입한 후 정점의 개수인 g->n+1이 메인에서의 정점 개수 v값보다 크면 함수를 종료한다. v값보다 작다면 그래프의 정점의 정보를 저장하는 ary배열에 정점 데이터를 삽입한다. num과 str에 차례대로 정점 데이터를 삽입하고 삽입이 끝나면 정점의 개수인 g->n을 증가시킨다.

25. insert_edge함수는 인접 리스트 그래프에 간선을 삽입하는 함수이다.

매개변수로 시작 정점과 끝 정점을 입력받는다.

Find_index함수를 통해 정점 배열에서 start 값과 end 값을 가지고 있는 정점의 인덱스를 반환받고 이 값을 u와 v에 저장한다.

26. 노드를 새롭게 동적할당하여 생성하고 end값을 이용하여 Find_index함수로 찾은 인덱스에 맞는 위치의 정점을 node의 데이터에 삽입한다.

그리고 start값을 이용하여 찾은 u값의 리스트에 insert_first연산을 수행한다. 노드의 링크가 리스트의 처음을 가리키도록 하고 리스트의 처음이 노드가 될 수 있도록 한다.

```

int topo_sort(GraphType *g) { //위상정렬 함수
    int i;
    StackType s; //스택 변수
    GraphNode *node; //임시 노드 변수
    int count = 0, n = 0;
    int *in_degree = (int *)malloc(g->n * sizeof(int));
    //모든 정점의 진입 차수를 계산
    for (i = 0; i < g->n; i++) {
        in_degree[i] = 0; //in_degree를 0으로 초기화
    }
    for (i = 0; i < g->n; i++) {
        GraphNode *node = g->adj_list[i]; //정점 i에서 나오는 간선들
        while (node != NULL) {
            n = Find_index(g->ary, node->vertex, g->n); //정점배열에서 인덱스 값을 찾음
            in_degree[n]++; //그 인덱스에 맞는 in_degree를 증가
            node = node->link;
        }
    }
    /* for (i = 0; i < g->n; i++) {
        printf("%d ", in_degree[i]);
    } */
    init(&s, g->n); //스택 초기화
    for (i = 0; i < g->n; i++) { //진입 차수가 0인 정점을 스택에 삽입
        if (in_degree[i] == 0) push(&s, i);
    }
    printf("< 위상 순서 출력 >#\n");
    printf("-----#\n");
    while (!is_empty(&s)) { //스택이 공백상태가 될 때까지 반복
        int w;
        w = pop(&s); //가장 마지막으로 삽입된 인덱스 값을 pop
        printf("%d, %-2d - %s#\n", ++count, g->ary[w].num, g->ary[w].str); //pop된 인덱스 값에 맞는 정점 출력
        node = g->adj_list[w]; //각 정점의 진입 차수를 변경
        while (node != NULL) {
            n = Find_index(g->ary, node->vertex, g->n); //정점배열에서 인덱스 값을 찾음
            in_degree[n]--; //찾은 인덱스 값의 진입 차수를 감소
            if (in_degree[n] == 0) push(&s, n); //진입 차수가 0인 정점인덱스를 삽입
            node = node->link;
        }
    }
    printf("-----#\n");
    free(in_degree); //in_degree배열 삭제
    free(s.stack); //스택 삭제
    printf("#\n");
    return (i == g->n);
}

```

27. topo_sort는 위상정렬 알고리즘을 통해 정점을 출력하는 함수이다.

정렬을 위해 스택 변수 s와 임시 노드 변수 node를 선언한다.

그리고 진입 차수를 저장할 in_degree포인터를 정점의 개수로 동적할당한 후 반복문을 통해 in_degree를 0으로 초기화한다.

28. 모든 정점의 간선 리스트를 돌며 진입차수 in_degree를 계산한다. 한 정점에서 갈 수 있는 정점의 인덱스를 Find_index로 n에 저장한 후 그 인덱스에 맞는 in_degree를 증가시킨다.

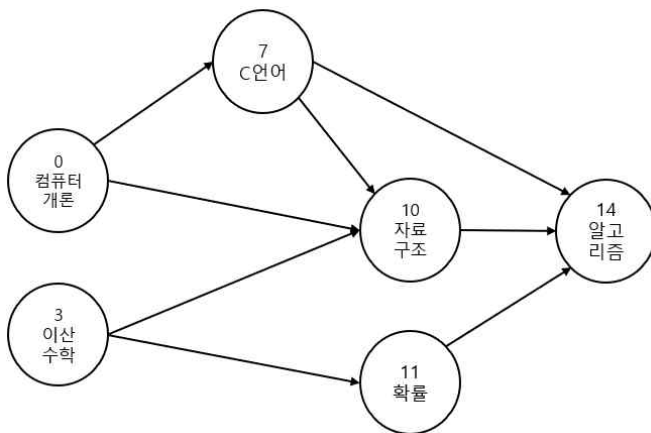
29. 스택을 정점의 개수 g->n값을 init함수로 전달하여 스택을 초기화한다.

그리고 반복문을 돌며 진입차수가 0인 정점을 스택에 삽입한다.

스택배열은 int형으로 선언되었기 때문에 정점의 인덱스 값을 저장할 수 있다.

30. while문을 통해 스택이 공백상태가 될 때까지 반복을 수행한다. 먼저 스택에 가장 마지막으로 삽입된 인덱스를 pop하여 w에 저장하고 그 w인덱스에 맞는 정점 값을 출력한다.

31. 그리고 그 w인덱스 정점에서 갈 수 있는 정점을 찾기 위해 w인덱스 리스트를 node에 저장하고 끝까지 반복하며 w인덱스 정점에서 갈 수 있는 정점의 인덱스를 찾고 그 인덱스에 맞는 진입차수 in_degree값을 감소시킨다. 그리고 진입차수가 0이 되었다면 다시 스택에 삽입한다.



그래프의 구조

```

정점 컴퓨터개론의 인접 리스트 -> 10 자료구조 -> 7 C언어
정점 이산수학의 인접 리스트 -> 11 확률 -> 10 자료구조
정점 C언어의 인접 리스트 -> 14 알고리즘 -> 10 자료구조
정점 자료구조의 인접 리스트 -> 14 알고리즘
정점 확률의 인접 리스트 -> 14 알고리즘
정점 알고리즘의 인접 리스트
0 0 1 3 1 3
  
```

컴퓨터개론에서 갈 수 있는 정점은 자료구조, C언어
 이산수학에서 갈 수 있는 정점은 확률, 자료구조
 C언어에서 갈 수 있는 정점은 알고리즘, 자료구조
 확률에서 갈 수 있는 정점은 알고리즘
 알고리즘에서 갈 수 있는 정점은 존재하지 않는다.

▶이 갈 수 있는 정점들을 이용해 정점의 인덱스를 찾고 진입 차수를 계산한다. 컴퓨터 개론과 이산수학의 진입 차수는 0, C언어는 1, 자료구조는 3, 확률은 1, 알고리즘은 3개의 진입 차수를 가지기 때문에 in_degree는 0 0 1 3 1 3의 값을 가지게 된다.

```

void Delete(GraphType *g) {           //그래프 삭제 함수
    int i;
    GraphNode *p, *q;
    for (i = 0; i < g->n; i++) {
        free(g->ary[i].str);          //ary배열의 str문자열 삭제
        p = g->adj_list[i];
        q = p;
        if (p != NULL) {
            while (p != NULL) {
                q = q->link;
                free(p);              //링크리스트 삭제
                p = q;
            }
        }
    }
    //리스트의 노드들은 str과 같은 공간을 가지므로 삭제 x
    free(g->ary);                    //ary포인터 삭제
    free(g->adj_list);              //adj_list삭제
    free(g);                        //그래프 삭제
}

```

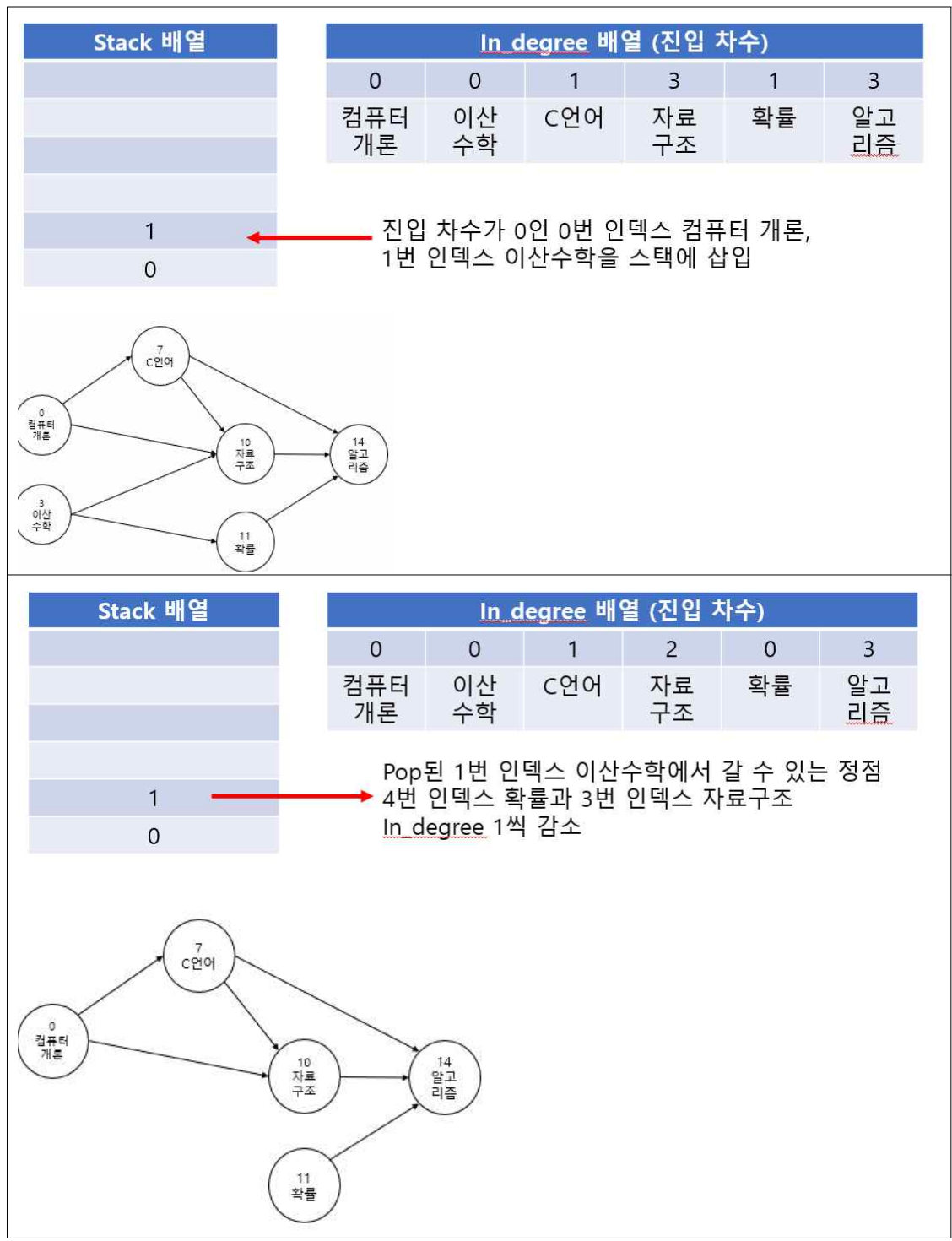
32. Delete함수는 그래프를 삭제하는 역할을 하는 함수이다.
리스트의 노드를 모두 삭제하기 위해 p와 q를 선언한다.

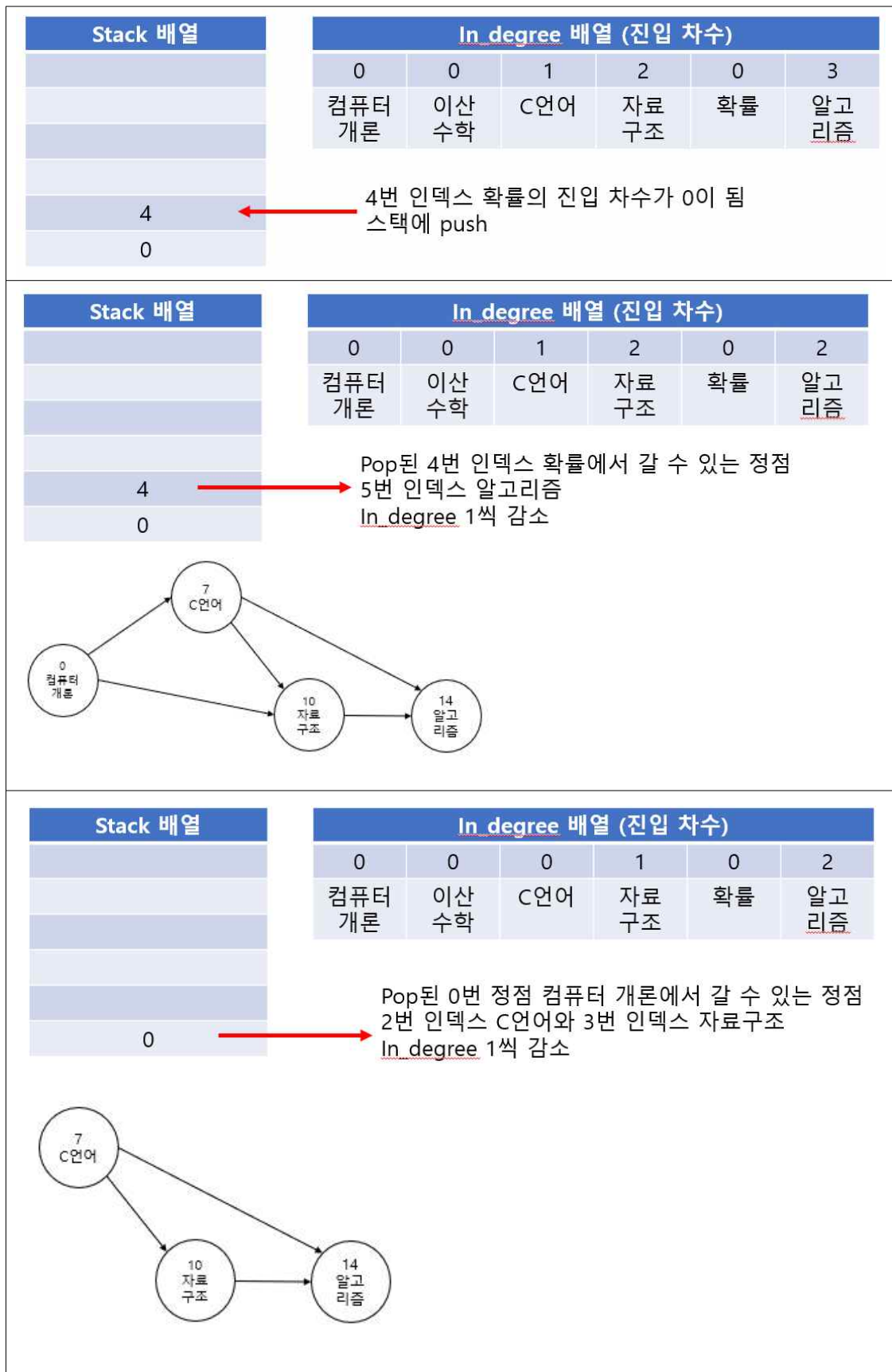
33. 반복문을 통해 먼저 ary배열의 정점 문자열을 삭제한다. 그리고 리스트의 값을 p에 담고 p가 NULL이 될 때까지 반복하며 리스트의 노드를 삭제한다.

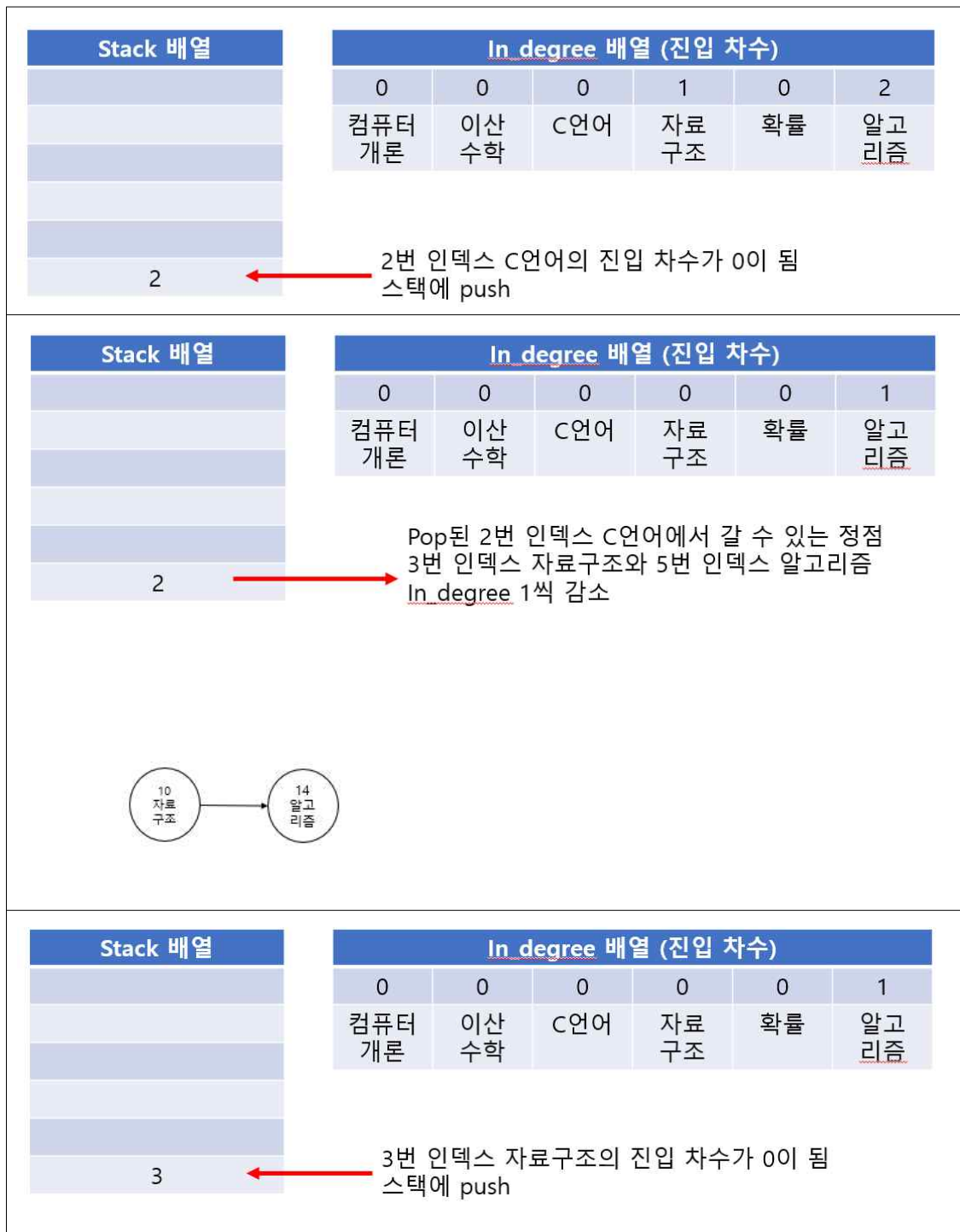
여기서 노드의 문자열을 삭제하지 않는 이유는 노드 문자열은 ary정점 배열의 문자열과 같은 주소를 가지기 때문이다. 정점 문자열은 위에서 이미 삭제 했기 때문에 다시 삭제하면 오류가 발생한다.

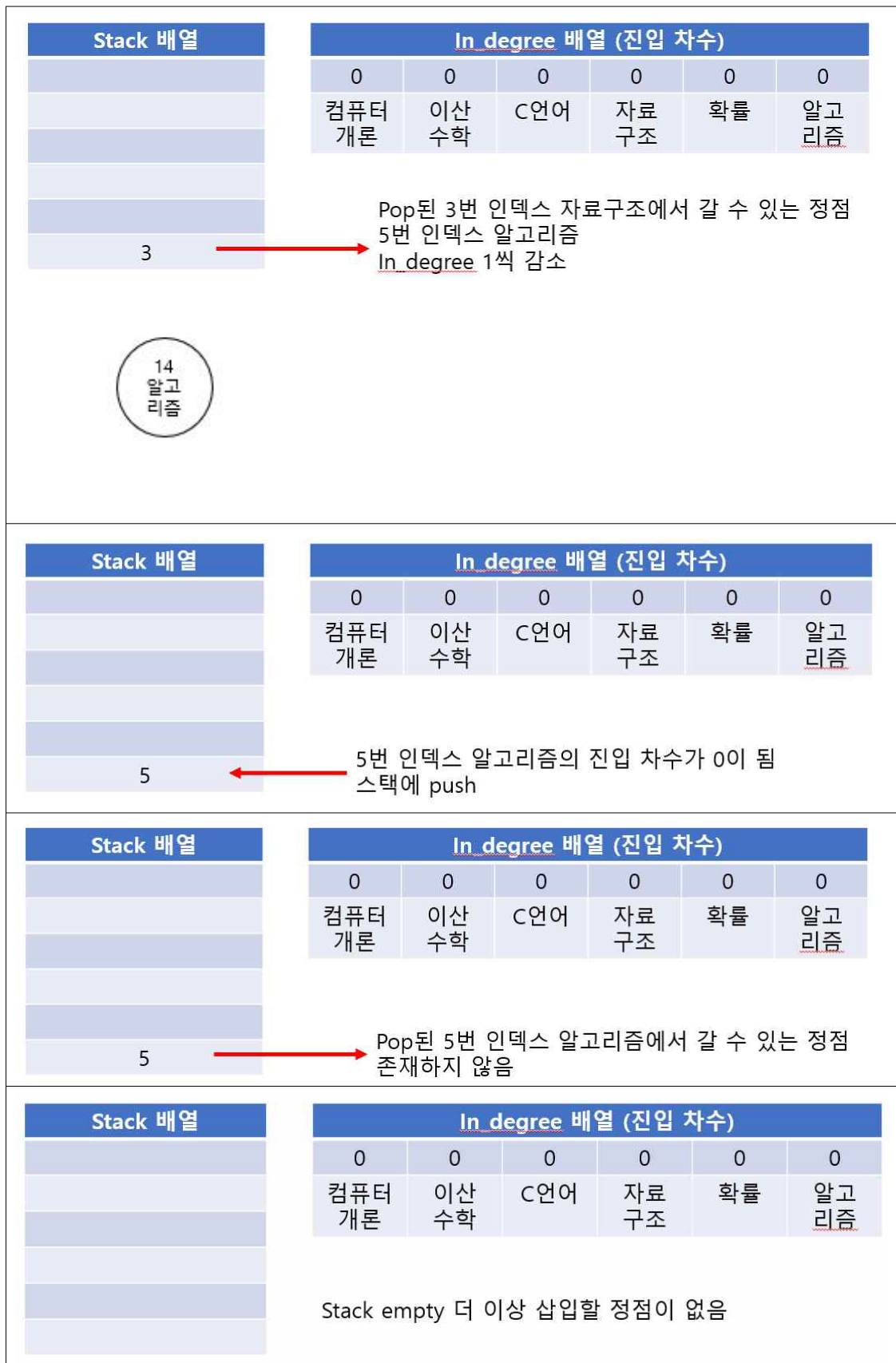
34. 모든 리스트와 문자열을 삭제했다면 ary배열도 삭제해주고 리스트도 삭제해주도록 한다. 마지막으로 그래프 변수 g도 삭제한다.

위상정렬의 스택 변화 과정









2.4 실행 창

< 데이터 >

0 - 컴퓨터개론
3 - 이산수학
7 - C언어
10 - 자료구조
11 - 확률
14 - 알고리즘

< 위상 순서 출력 >

1. 3 - 이산수학
2. 11 - 확률
3. 0 - 컴퓨터개론
4. 7 - C언어
5. 10 - 자료구조
6. 14 - 알고리즘

C:\Users\dmk46\OneDrive\바탕
t.exe(6384 프로세스)이(가) 0
이 창을 닫으려면 아무 키나 누

data - Windows 메모장

파일(F) 편집(E) 서식(O) 보

v 0 컴퓨터개론
v 3 이산수학
v 7 C언어
v 10 자료구조
v 11 확률
v 14 알고리즘
e 0 7
e 0 10
e 3 10
e 3 11
e 7 10
e 7 14
e 10 14
e 11 14

2.5 느낀 점

이 과제는 위상정렬 알고리즘을 사용하여 위상 순서에 맞게 정점을 출력하는 문제였습니다. 처음 보는 알고리즘인데다가 스택을 이용한 프로그램이었기 때문에 처음에는 어렵게 느꼈지만 이번 과제를 하고나니 위상정렬에 대해 이해를 할 수 있었습니다.

처음에 막혔었던 부분은 정점을 나타내는 정수가 무작위로 존재하는 부분이었습니다. 처음에는 전에 했던 것처럼 최댓값을 이용하여 해결해보려 했지만 실패했었고, 이를 해결하기 위해 정점을 저장할 구조체를 하나 더 만들고 정점에 대한 데이터를 따로 그래프에 저장했습니다. 그리고 필요한 값을 찾기 위해선 정수 값이 일치하는 정점의 인덱스가 필요하므로 이를 구하는 함수를 하나 만들어서 인덱스 값을 반환할 수 있도록 코드를 짰습니다.

위상 정렬의 알고리즘의 진행을 확인하기 위해 `in_degree` 진입차수 배열과 스택의 변화과정에 대해 생각해보았습니다. 코드를 짜서 직접 확인해보고 그림도 그려가며 `in_degree` 배열과 스택이 어떻게 변화하는지와 그래프는 어떻게 변화하는지 확인해보았습니다. 이를 통해 위상정렬의 알고리즘이 어떤 방식을 진행하는지 깨달을 수 있었습니다.

이번 문제는 책에 있는 코드에서 살짝 더 코드를 추가하여 프로그래밍할 수 있었습니다. 이번기회를 통해 위상정렬의 알고리즘에 대해 알 수 있었고 이를 어떤 이용할 방법에 대해 생각해보는 계기가 되었습니다.

3. 정렬

선택 정렬 프로그램

- results_stpes.txt에는 우리 대학교 SW중심대학 사업단에서 운영하고 있는 사업인 웰라이프 생활-실습형 BLEP의 데이터로, Fitbit을 이용하여 활동 데이터로 걸음 수 정보가 저장되어 있다. 많이 걸음을 걸은 날부터 출력하도록 선택정렬을 이용하여 정렬하시오.

2.1 문제 분석

조건 1 동적할당을 이용하여 데이터를 저장

조건 2 선택 정렬을 이용하여 데이터를 정렬

조건 3 가장 많이 걸음을 걸은 날부터 내림차순으로 정렬

- 이 문제는 Fitbit을 이용한 활동 데이터 걸음 수 정보를 선택정렬을 이용하여 가장 많이 걸은 날부터 내림차순으로 정렬하고, 이를 출력하는 문제이다. 이 문제의 경우 가장 많이 걸은 날이 배열의 첫 번째 요소로 와야 하기 때문에 현재 배열에서 가장 걸음 수가 많은 날의 인덱스를 저장하고, 이를 첫 번째 요소와 변경한다. 그 다음으로 첫 번째 요소를 제외한 나머지 요소들 중에서 가장 큰 값을 선택하고 이를 두 번째 요소와 교환한다. 이를 배열의 개수-1만큼 되풀이하면 추가적인 배열 없이 선택정렬을 수행할 수 있다.

선택정렬의 장점은 자료 이동 횟수가 미리 결정된다는 것이지만 이동 횟수는 $3(n-1)$ 로 큰편에 속한다. 또한 자료가 정렬된 경우에는 불필요하게 자기 자신과 이동을 하게 될 수도 있다. 또한 전체 비교 횟수는 외부 루프는 $n-1$ 번 실행되고 내부루프는 $(n-1)-i$ 번 이므로 시간 복잡도는 $O(n^2)$ 가 된다.

3.2 소스 코드

```
/*
 * 학번 : 20154812
 * 학과 : 컴퓨터소프트웨어공학부
 * 이름 : 김동진
 * 파일명 : 선택정렬 프로그램
 */
#define _CRT_SECURE_NO_WARNINGS

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

#define SWAP(x,y,t) ((t)=(x), (x)=(y), (y)=(t)) //두 변수의 값을 변경하는 SWAP 정의

typedef struct Data { //자료 구조를 데이터 구조체
    char *date; //년도를 저장할 date 문자열 포인터
    int walking; //걸음 횟수를 저장할 walking 변수
}Data;

void selection_sort(Data list[], int n); //선택정렬 함수
void Print(Data arr[], int n); //배열 출력 함수
void Delete(Data *arr, int count); //메모리 삭제 함수

int main() {
    FILE *fo; //파일 포인터
    Data *arr; //Data 구조체 포인터, 메모리의 개수로 출력할 것
    int count = 0; //메모리의 개수를 셀 count 변수
    char str[20]; //파일로부터 문자열을 읽어서 입력받은 문자열 변수
    int date, i = 0; //걸음 수를 입력받은 date

    fo = fopen("results_steps.txt", "r"); //파일을 읽기 목적으로 open
    if (!fo) {
        printf("file not open");
        return 0;
    }
    while (!feof(fo)) { //파일 끝까지 입력
        fscanf(fo, "%s%d", str, &date);
        count++; //메모리의 개수 증가
    }
    arr = (Data*)malloc(sizeof(Data)*count); //메모리의 개수로 출력할 것
    rewind(fo);
    while (!feof(fo)) {
        fscanf(fo, "%s%d", str, &date); //파일 문자열, 걸음 수 입력
        //현재 위치의 date 문자열을 str의 길이에 출력할 것
        arr[i].date = (char*)malloc(sizeof(char)*(strlen(str) + 1));
        strcpy(arr[i].date, str); //str 문자열을 date에 복사
        arr[i].walking = date; //걸음 수 삽입

        i++; //증가
    }
    printf("++ Selection Sort 정렬 전 출력 ++\n");
    printf("===== \n");
    Print(arr, count); //선택정렬 전 출력

    selection_sort(arr, count); //선택정렬 함수 출력
    printf("===== Selection Sort 정렬 후 출력 ===== \n");
    Print(arr, count); //선택정렬 후 출력

    Delete(arr, count); //arr 메모리 삭제
    fclose(fo); //파일 포인터 닫음
    return 0;
}

void selection_sort(Data list[], int n) { //선택정렬 함수
    int i, j, min; //i는 반복문을 돌 변수, min은 가장 큰 값의 인덱스
    Data temp;
    for (i = 0; i < n - 1; i++) {
        min = i; //현재 값의 인덱스를 min에 할당
        for (j = i + 1; j < n; j++) {
            if (list[j].walking > list[min].walking) min = j; //min에 가장 큰 값의 인덱스 저장
        }
        SWAP(list[i], list[min], temp); //현재 인덱스와 min인덱스의 값을 변경
    }
}

void Print(Data arr[], int n) { //배열 출력 함수
    int i;
    for (i = 0; i < n; i++) {
        printf("%s %d\n", arr[i].date, arr[i].walking); //파일, 걸음 수 순서로 출력
    }
}

void Delete(Data *arr, int count) { //메모리 삭제 함수
    int i;
    for (i = 0; i < count; i++) {
        free(arr[i].date); //파일 끝까지 출력 date 문자열 삭제
    }
    free(arr); //arr 배열 삭제
}
```

3.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: 선택정렬 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS

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

#define SWAP(x,y,t) ((t)=(x), (x)=(y), (y)=(t)) //두 변수의 값을 변경하는 SWAP 정의

typedef struct Data { //기본 자료형 데이터 구조체
    char *date;        //년도를 저장할 date 문자열 포인터
    int walking;        //걸음 횟수를 저장할 walking변수
}Data;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성하고, 필요한 헤더를 포함한다.

2. 값을 서로 변경하는 SWAP매크로를 정의한다. SWAP은 입력받은 x값을 t에 담고 y값을 x에 담고, y에 t값을 담으면서 입력받은 x와 y값을 서로 변경하는 역할을 한다.

3. 파일 데이터를 저장할 Data구조체를 선언한다.

파일에 저장되어있는 날짜를 저장할 문자열 변수 date와 걸음 수 정보를 저장할 int형 walking변수를 선언한다.

```
void selection_sort(Data list[], int n); //선택정렬함수
void Print(Data ary[], int n); //배열 출력 함수
void Delete(Data *ary, int count); //데이터 삭제함수
```

4. 필요한 함수들을 선언한다.

- selection_sort함수는 배열을 선택정렬을 이용하여 정렬하는 함수이다.
- Print함수는 배열을 순서대로 출력하는 함수이다.
- Delete함수는 동적할당한 부분을 모두 삭제하는 함수이다.

```

int main() {
    FILE *fp;           //파일포인터
    Data *ary;          //Data자료형 포인터, 데이터의 개수로 동적할당
    int count = 0;      //데이터의 개수를 셀 count변수
    char str[20];       //파일로부터 문자열을 임시로 입력받을 문자열 변수
    int data, i = 0;     //걸음 수를 입력받을 data

    fp = fopen("results_stpes.txt", "r"); //파일을 읽기 형식으로 open
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) { //파일 끝까지 입력
        fscanf(fp, "%s%d", str, &data);
        count++;        //데이터의 개수 증가
    }
    ary = (Data*)malloc(sizeof(Data)*count); //데이터의 개수로 동적할당
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%s%d", str, &data); //날짜 문자열, 걸음수 입력
        //현재 위치의 date문자열을 str의 길이로 동적할당
        ary[i].date = (char*)malloc(sizeof(char)*(strlen(str) + 1));
        strcpy(ary[i].date, str); //str문자열을 date에 복사
        ary[i].walking = data; //걸음수 삽입

        i++; //i증가
    }

    printf("** Selection Sort 정렬 전 출력 **\n");
    printf("=====\n");
    Print(ary, count); //선택정렬 전 출력

    selection_sort(ary, count); //선택정렬 함수 출력
    printf("\n\n** Selection Sort 정렬 후 출력 **\n");
    printf("=====\n");
    Print(ary, count); //선택정렬 후 출력

    Delete(ary, count); //ary데이터 삭제
    fclose(fp);         //파일포인터 닫음
    return 0;
}

```

5. 필요한 변수들을 선언한다.

- fp는 파일포인터, ary는 Data자료형으로 파일의 정보를 저장하는 역할을 한다.
- count는 데이터의 개수를 세는 변수이고 str은 문자열을, data는 정수를 파일로부터 데이터를 입력받는다.

6. BLEP데이터가 저장되어 있는 result_stpes.txt파일을 읽기형식으로 open한다. open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 입력을 받으면서 데이터의 개수를 나타내는 count를 증가시킨다.

7. 위에서 구한 count의 개수로 ary포인터를 동적할당하고 rewind함수를 이용하여 파일포인터를 처음으로 옮긴다.

8. feof를 이용하여 파일을 다시 끝까지 입력받는다. 파일을 입력받은 후 ary배열의 date문자열을 str의 길이+1로 동적할당한 후 strcpy를 이용하여 문자열을 복사한다.

그리고 걸음수를 입력받은 data를 ary변수의 walking에 삽입한다.

삽입이 끝난 후엔 I를 증가시켜 다음 배열에 값을 담을 수 있도록 한다.

9. Print함수를 사용하여 선택정렬을 수행하기 전의 데이터를 출력한다. 매개변수로는 ary배열과 데이터의 개수인 count를 전달한다.

10. selection_sort함수를 호출하여 선택정렬을 수행할 수 있도록 한다. 선택정렬을 수행한 후의 데이터를 Print함수로 다시 한 번 출력한다.

11. 동적할당한 ary배열을 삭제할 Delete함수를 호출한다. Delete함수에서는 ary의 문자열과 ary배열을 삭제한다.

fclose함수를 이용하여 파일을 닫고, 프로그램을 종료한다.

```
void selection_sort(Data list[], int n) { //선택정렬함수
    int i, j, high; //i와 j는 반복문을 돌 변수, high는 가장 큰 값의 인덱스
    Data temp;
    for (i = 0; i < n - 1; i++) {
        high = i; //현재 값의 인덱스를 high에 담음
        for (j = i + 1; j < n; j++) {
            if (list[j].walking > list[high].walking) high = j; //high에 가장 큰 값의 인덱스 저장
        }
        SWAP(list[i], list[high], temp); //현재 인덱스와 high인덱스의 값을 변경
    }
}
```

12. selection_sort함수는 선택정렬을 이용하여 큰 값부터 내림차순 정렬하는 함수이다. 매개변수로는 Data형 배열과 데이터의 개수 n을 전달받는다.

13. 반복문을 돌 i, j 변수와 가장 큰 값의 인덱스를 저장할 high를 선언하고 값을 변경할 때 사용할 Data형 temp변수를 선언한다.

14. 선택 정렬은 배열에서 가장 큰 값을 선택하여 배열의 첫 번째 요소와 교환한다. 다음에는 첫 번째 요소를 제외한 나머지 요소들 중에서 가장 작은 값을 선택하고 이를 두 번째 요소와 교환한다.

이 절차를 데이터의 개수-1 만큼 반복하면 내림차순 정렬을 수행할 수 있다.

15. 선택정렬은 1 값이 0에서 n-2까지만 변화한다. 만약 list의 0번 인덱스부터 n-2번 인덱스까지 정렬이 되었으면 n-1인덱스가 가장 큰 값이기 때문에 값을 변경할 필요가 없기 때문이다.

16. 반복문을 돌며 high 변수에 가장 큰 값의 인덱스를 저장하고 SWAP매크로를 사용하여 list[i]값과 high값의 위치를 변경한다. 변경을 수행할 땐 임시로 값을 저장할 temp변수를 함께 전달한다.

```
void Print(Data ary[], int n) {    //배열 출력 함수
    int i;
    for (i = 0; i < n; i++) {
        printf("%s %d\n", ary[i].date, ary[i].walking); //날짜, 걸음 수 순으로 출력
    }
}

void Delete(Data *ary, int count) {    //데이터 삭제 함수
    int i;
    for (i = 0; i < count; i++) {
        free(ary[i].date); //파일 끝까지 돌며 data문자열 삭제
    }
    free(ary);    //ary배열 삭제
}
```

17. Print함수는 Data자료형 배열을 순서대로 출력하는 함수이다.

배열의 개수인 n만큼 for문을 이용하여 반복하고, 날짜, 걸음 수순으로 0번 인덱스부터 차례대로 출력한다.

18. Delete함수는 Data배열에 대한 정보를 삭제하는 함수이다.

배열의 개수인 count만큼 for문을 이용하여 date문자열 포인터를 메모리 해제하고 마지막으로 ary배열 또한 메모리 해제한다.

3.4 실행 창

선택정렬 하기 전 출력	선택정렬 내림차순 정렬 후 출력
<p>** Selection Sort 정렬 전 출력 **</p> <pre> ===== 2021-04-07 10174 2021-04-08 4829 2021-04-09 8262 2021-04-10 8864 2021-04-11 8467 2021-04-12 5120 2021-04-13 10557 2021-04-14 10594 2021-04-15 8313 2021-04-16 11314 2021-04-17 8810 2021-04-18 8798 2021-04-19 11358 2021-04-20 15456 2021-04-21 5149 2021-04-22 4754 2021-04-23 11104 2021-04-24 2128 2021-04-25 876 2021-04-26 5739 2021-04-27 4274 2021-04-28 10243 2021-04-29 4455 2021-04-30 4637 2021-05-01 8591 2021-05-02 1265 2021-05-03 4649 2021-05-04 9640 2021-05-05 1972 2021-05-06 0 2021-05-07 100 2021-05-08 0 2021-05-09 53 2021-05-10 2828 2021-05-11 2809 2021-05-12 5003 2021-05-13 4726 2021-05-14 2106 2021-05-15 1664 2021-05-16 0 2021-05-17 0 2021-05-18 7450 2021-05-19 2161 2021-05-20 0 2021-05-21 0 2021-05-22 956 2021-05-23 5855 2021-05-24 3074 2021-05-25 5583 2021-05-26 0 2021-05-27 2716 2021-05-28 3005 2021-05-29 624 2021-05-30 0 2021-05-31 0 </pre>	<p>** Selection Sort 정렬 후 출력 **</p> <pre> ===== 2021-11-04 19259 2021-11-05 17319 2021-04-20 15456 2021-10-18 13184 2021-11-11 12536 2021-04-19 11358 2021-04-16 11314 2021-04-23 11104 2021-11-13 10621 2021-04-14 10594 2021-04-13 10557 2021-10-17 10497 2021-09-03 10361 2021-04-28 10243 2021-04-07 10174 2021-09-12 9828 2021-11-08 9823 2021-11-03 9645 2021-05-04 9640 2021-07-01 9215 2021-11-12 8985 2021-08-27 8884 2021-04-10 8864 2021-10-08 8855 2021-04-17 8810 2021-04-18 8798 2021-05-01 8591 2021-04-11 8467 2021-04-15 8313 2021-04-09 8262 2021-09-01 8204 2021-09-02 8188 2021-11-10 7912 2021-06-30 7571 2021-09-14 7568 2021-05-18 7450 2021-09-17 7307 2021-09-15 7298 2021-08-28 7290 2021-09-10 7122 2021-10-02 7085 2021-11-02 7049 2021-10-06 7022 2021-09-04 7011 2021-09-05 6882 2021-09-13 6679 2021-08-31 6589 2021-10-01 6461 2021-10-05 6325 2021-10-03 6321 2021-09-23 6221 2021-10-04 6134 2021-08-29 6077 2021-09-06 6056 2021-09-07 6032 2021-10-07 5989 </pre>

3.5 느낀 점

이번 문제는 입력받은 데이터 파일을 내림차순 정렬하여 출력하는데, 선택정렬 알고리즘을 이용하여 정렬하는 문제였습니다. 지금까지 정렬하면 버블정렬만 해왔었는데 선택정렬을 이용하여 프로그램을 짜니 이렇게도 정렬이 가능하다는 것을 알 수 있었습니다.

이번 문제를 프로그래밍을 할 때 요구사항 중 하나는 데이터의 개수가 변화할 수 있기 때문에 동적할당을 이용하여 배열을 생성하는 것이었습니다. 그렇기 때문에 날짜 문자열과 걸음 수정수로 이루어진 구조체 배열을 데이터의 개수를 구해서 동적할당하여 선언할 수 있었습니다.

선택정렬을 할 때 1값이 $n-2$ 까지만 변화한다는 것이 처음에는 잘 이해가 되지 않았었는데, 그림을 직접 그려보며 확인해보니 배열의 $n-2$ 까지 정렬이 되었다면 배열의 $n-1$ 이 가장 큰 값이라는 것이 이해가 되었습니다. 만약 제가 처음 프로그래밍 했었다면 n 까지 반복을 했을텐데 반복 횟수를 하나라도 줄이려는 것이 중요하다는 것을 깨달을 수 있었습니다.

이번 기회를 통해 선택정렬에 대해 이해할 수 있었습니다. 앞으로 여러 가지 정렬을 배우게 될 텐데, 상황에 맞는 정렬을 사용할 수 있도록 각각 정렬의 특징을 알고 있어야 한다는 것을 깨달았습니다. 그렇기 때문에 정렬에 대한 공부를 끊임없이 해야겠다고 생각했습니다.

4. 과제를 하며 느낀 점

이번 과제를 통해 그래프에서 사용할 수 있는 알고리즘에 대해 배울 수 있었고, 정렬 중에서도 선택정렬에 대해 배울 수 있었습니다. 처음에는 많이 막히는 부분도 많았었는데, 알고리즘에 대한 공부를 통해 문제를 풀 수 있었습니다.

이번 기회로 정말 많은 알고리즘이 존재한다는 것을 알 수 있었고 이를 모두 알기 위해선 끊임없는 공부가 필요하다는 것을 느꼈습니다. 이번에 배운 알고리즘들을 이용하여 언제 어떻게 사용할 수 있을지 항상 생각하기 위해 노력할 것입니다. 그러기 위해선 알고리즘 공부를 계속 할 것이고 알고리즘을 프로 그래밍하는 실력을 쌓은 후에는 제가 직접 알고리즘을 만들어 보고 싶다는 생각도 했습니다.