



순천향대학교
SOON CHUN HYANG
UNIVERSITY

자료구조2 실습 HW 12

자료구조2 실습

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 11월 23일

| 목 차 |

1. Floyd의 최단 경로 알고리즘

- 1.1 문제분석
- 1.2 소스 코드
- 1.3 소스 코드 분석
- 1.4 실행 결과
- 1.5 느낀점

2. 위상 정렬 알고리즘

- 2.1 문제분석
- 2.2 소스 코드
- 2.3 소스 코드 분석
- 2.4 실행 결과
- 2.5 느낀점

3. 발걸음 데이터 선택 정렬

- 3.1 문제분석
- 3.2 소스 코드
- 3.3 소스 코드 분석
- 3.4 실행 결과
- 3.5 느낀점

1.1 문제 분석

▣ Floyd의 최단 경로 알고리즘

- 428 페이지에 있는 프로그램 11.11의 Floyd의 최단 경로 프로그램을 참고하여 data.txt 파일에 입력되어있는 정점과 간선의 정보를 가지고 초기 상태 및 426 페이지의 모든 결과를 순서에 맞게 출력하라.
- 동적 할당을 통하여 더 많은 정점이 입력될 경우 삽입될 수 있게 할 것
- 업데이트 된 배열의 부분에 *로 표시할 것



이 문제는 Floyd의 알고리즘을 이용해 데이터를 읽어서 해결해야 한다. 가장 중요하게 생각해야 할 부분은 Dijkstra와 다르게 간단하고 각 정점간 최단의 경로를 구할 때 유용하다. 계속해서 비교하고 교체하는 형식으로 진행된다. 또한 조건 중 업데이트 된 배열 부분을 계속 체크하면서 표시하는 부분도 중요하고 정점의 크기에 따라 그래프 동적할당을 해주는 형식으로 진행해 줘야 하는 부분을 계속해서 생각하고 만들어야 한다

우선 그래프 구조체를 선언하는데 구조체 안에는 정점의 개수 변수와 2차원 그래프를 이중 포인터로 선언해서 추후 동적할당을 진행하게 한다

2개의 배열을 만들어서 방문, 상태 저장 2차원 배열을 만들어서 Floyd 알고리즘에서 사용한다

그래프 초기화 함수를 만들어 그래프 배열을 초기화 하고 모든 행과 열의 방문을 없애 주는 함수를 만들어 준다

그래프 정점 삽입 함수에서 정점의 개수를 확인하고 정점을 추가한다.

간선 삽입 함수를 만들어서 간선을 확인하고 완료되면 받은 가중치를 2차원 배열 무방향 그래프로 만들어서 넣어준다.

Floyd 알고리즘 함수에서는 우선 이중 for문을 통해 A 상태 배열에 행과 열에 그래프의 가중치의 값들을 넣어준다. 초기 그래프를 출력하고 3중 for 문을 통해 값의 크기를 비교한 후 더 작은 값이 될 경우 변경해주고 방문 표시를 변경해준다.

그래프 출력 함수에서는 정점의 크기만큼 반복하면서 값을 보면서 값이 있을 경우 출력을 한다. 만약 조건을 통해 업데이트가 된 값이면 *를 남기고 아닐 경우는 그냥 출력을 한다.

메인 함수에서는 필요한 변수를 선언하고 파일을 오픈한다. 파일을 읽은 후 정점의 개수 만큼 동적할당을 진행한다. 구조체에서 이중 포인터로 선언했기 때문에 그래프의 행과 열을 정점의 개수 만큼 동적할당을 진행해준다. 그 후 간선 삽입을 한 후 floyd 알고리즘 호출, 동적할당을 해제하고 종료한다

1.2 소스 코드

```
1 //=====
2 // 제작일자: 21년 11월 17일 ~ 11월 22일
3 // 제작자: 20204005 김필중
4 // 프로그램명: Floyd의 최단 경로 알고리즘
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 오류 방지 구문을 선택한다
13 #pragma warning(disable : 4996)
14
15 // 필요한 변수들을 정의한다
16 #define TRUE 1
17 #define FALSE 0
18 #define MAX_VERTICES 100
19 #define INF 1000000 /* 무한대 (연결이 없는 경우) */
20
21 // 그래프 구조체를 선언한다
22 typedef struct GraphType {
23     int n; // 정점의 개수
24     int **weight; // 2차원 그래프 가중치 포인터 선언
25 } GraphType;
26
27 int A[MAX_VERTICES][MAX_VERTICES]; // 상태를 저장하는 2차원 배열
28 int visited[MAX_VERTICES][MAX_VERTICES]; // 방문을 체크하는 2차원 배열
29
30 // 그래프 초기화 함수
31 void init(GraphType *g, int cnt)
32 {
33     int r, c; // 변수 선언
34     g->n = 0; // 정점의 개수 0개로 초기화
35     // 2차원 배열 초기화
36     for (r = 0; r < cnt; r++)
37     {
38         for (c = 0; c < cnt; c++)
39         {
40             g->weight[r][c] = INF; // r행 c열의 값을 INF로 초기화
41             visited[r][c] = FALSE; // r행 c열을 방문 false 체크
42         }
43     }
44     //r행 r열 의 가중치를 0으로 초기화
45     for (r = 0; r < cnt; r++)
46         g->weight[r][r] = 0;
47 }
48
49
```

```

50 // 그래프 정점 삽입 함수
51 void insert_vertex(GraphType *g, int n, int cnt)
52 {
53     // 정점의 개수 확인
54     if ((g->n) + 1 > cnt) // 정점의 개수가 최대를 넘을 경우 오류 발생
55     {
56         fprintf(stderr, "그래프: 정점 번호 오류");
57         return;
58     }
59     // 아닐 경우는 정점 1개 추가
60     g->n++;
61 }
62
63 // 간선 삽입 함수
64 void insert_edge(GraphType *g, int start, int end, int w)
65 {
66     // 받은 간선이 정점의 개수보다 많으면 오류 발생
67     if (start >= g->n || end >= g->n)
68     {
69         fprintf(stderr, "그래프: 정점 번호 오류");
70         return;
71     }
72     // 아닐 경우는 2차원 배열의 가중치 넣기
73     g->weight[start][end] = w;
74     g->weight[end][start] = w;
75 }
76

```

```

127 // 그래프 출력 함수
128 void printA(GraphType *g)
129 {
130     // 필요한 변수 선언
131     int i, j;
132     for (int i = 0; i < g->n; i++)
133     {
134         printf("  %d ", i);
135     }
136     printf("\n |=====|\n");
137     // i가 정점의 크기만큼 반복
138     for (i = 0; i < g->n; i++)
139     {
140         printf("%d| ", i);
141         // j가 정점의 크기만큼 반복
142         for (j = 0; j < g->n; j++)
143         {
144             // 만약 A의 i행 j열의 값이 INF일 경우
145             if (A[i][j] == INF)
146                 printf(" x | "); // x표시를 출력
147             // 만약 i의 j열의 방문이 TRUE가 아닐 경우
148             else if (visited[i][j] != TRUE)
149             {
150                 if (A[i][j] < 10) // 표시 깔끔하게 하기 위한 숫자 구분
151                 {
152                     printf(" * %d| ", A[i][j]); // * 값을 출력하고
153                     visited[i][j] = TRUE; // TRUE로 변경
154                 }
155                 else
156                 {
157                     printf(" *%d| ", A[i][j]); // * 값을 출력하고
158                     visited[i][j] = TRUE; // TRUE로 변경
159                 }
160             }
161             // 이마저도 아닐경우
162             else
163             {
164                 if (A[i][j] < 10) // 표시 깔끔하게 하기 위한 숫자 구분
165                 {
166                     printf("   %d| ", A[i][j]); // 그냥 동일하기 때문에 출력
167                 }
168                 else
169                 {
170                     printf("  %d| ", A[i][j]); // 그냥 동일하기 때문에 출력
171                 }
172             }
173         }
174         printf("\n");
175     }
176     printf(" |=====|\n\n");
177 }

```



```

131 //floyd 알고리즘 함수
132 void floyd(GraphType* g)
133 {
134     int i, j, k; // 필요 변수 선언
135     // 그래프의 입력된 값들을 A 2차원 배열에 저장
136     for (i = 0; i < g->n; i++) // i가 정점의 크기만큼 반복
137         for (j = 0; j < g->n; j++) // j가 정점의 크기만큼 반복
138             A[i][j] = g->weight[i][j]; // A의 i행 j열에 값 대입
139
140     // 초기 그래프 출력
141     printf("< 초기 상태 >\n");
142     printA(g);
143
144     // k가 정점의 크기만큼 반복
145     for (k = 0; k < g->n; k++)
146     {
147         // i가 정점의 크기만큼 반복
148         for (i = 0; i < g->n; i++)
149             // j가 정점의 크기만큼 반복
150             for (j = 0; j < g->n; j++)
151                 // A[i][k] + A[k][j]의 값이 A[i][j]보다 크다면
152                 if (A[i][k] + A[k][j] < A[i][j])
153                 {
154                     A[i][j] = A[i][k] + A[k][j]; // A[i][j]값에 A[i][k] + A[k][j]의 값을 저장
155                     visited[i][j] = FALSE; // 방문을 안했음을 저장
156                 }
157     }
158     // 정점 표시
159     printf("< %d번 정점 열림 >\n", k);
160     printA(g);
161 }

```

```

162
163 // 메인 함수
164 int main(void)
165 {
166     FILE *fp; // 파일 포인터 선언
167     GraphType *g; // 그래프 포인터 선언
168     char tmp[100]; // 임시 저장 문자열 변수 선언
169     int check; // CHECK 변수 선언
170     // 필요한 임시 저장 변수를 선언
171     int cnt = 0;
172     int tmp1, tmp2, tmp3;
173
174     // 파일을 열고 실패하면 종료한다
175     fp = fopen("data01.txt", "r");
176
177     if (fp == NULL)
178     {
179         printf("파일 오픈 실패\n");
180         return;
181     }
182
183     // 파일이 끝날때 까지 반복을한다
184     while (!feof(fp))
185     {
186         // 맨앞 문자를 읽고 v일 경우
187         fscanf(fp, "%s", tmp);
188         if (strcmp(tmp, "v") == 0)
189         {
190             fscanf(fp, "%d", &check); // check에 저장하고
191             cnt++; // cnt 값을 더한다
192         }
193         // 아닐경우는 그냥 한줄을 읽는다
194         else
195         {
196             fgets(tmp, 100, fp);
197         }
198     }
199     // 파일 포인터를 맨앞으로 돌린다
200     rewind(fp);
201
202     // 그래프 g를 동적할당을 해준다
203     g = (GraphType *)malloc(sizeof(GraphType));
204
205     // cnt의 크기만큼 그래프를 동적할당 해준다
206     g->weight = (int **)malloc(sizeof(int*) * cnt);
207
208     // 열을 동적할당 해준다
209     for (int i = 0; i < cnt; i++)
210     {
211         g->weight[i] = (int *)malloc(sizeof(int)*cnt);
212     }
213

```

```

213
214 // 그래프를 초기화한다
215 init(g, cnt);
216 // 위에서 체크한 cnt 만큼 정점을 넣어준다
217 for (int i = 0; i < cnt; i++)
218     insert_vertex(g, i, cnt);
219
220 // 파일이 끝날때 까지 반복한다
221 while (!feof(fp))
222 {
223     // 맨앞 문자를 읽고 e일 경우
224     fscanf(fp, "%s", tmp);
225     if (strcmp(tmp, "e") == 0)
226     {
227         // 값 3개를 읽고 간선 삽입 함수를 호출한다
228         fscanf(fp, "%d %d %d", &tmp1, &tmp2, &tmp3);
229         insert_edge(g, tmp1, tmp2, tmp3);
230     }
231     // 아닐경우는 그냥 한줄을 읽는다
232     else
233     {
234         fgets(tmp, 100, fp);
235     }
236 }
237
238 // floyd 알고리즘 함수를 호출한다
239 floyd(g);
240
241 // 그래프 열의 동적할당을 해제한다
242 for (int i = 0; i < cnt; i++)
243 {
244     free(g->weight[i]);
245 }
246 // 그래프의 행의 동적할당을 해제한다
247 free(g->weight);
248 // 그래프를 해제한다
249 free(g);
250 //파일을 닫고 종료한다
251 fclose(fp);
252 return 0;
253 }

```

1.3 소스 코드 분석

```
1 //=====
2 // 제작일자: 21년 11월 17일 ~ 11월 22일
3 // 제작자: 20204005 김필중
4 // 프로그래밍: Floyd의 최단 경로 알고리즘
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 오류 방지 구문을 선택한다
13 #pragma warning(disable : 4996)
14
15 // 필요한 변수들을 정의한다
16 #define TRUE 1
17 #define FALSE 0
18 #define MAX_VERTICES 100
19 #define INF 1000000 /* 무한대 (연결이 없는 경우) */
20
```

1. 필요한 헤더파일을 선언한다
2. 오류 방지 구문을 선언한다
3. 필요한 변수들을 정의한다

```
// 그래프 구조체를 선언한다
typedef struct GraphType {
    int n; // 정점의 개수
    int **weight; // 2차원 그래프 가중치 포인터 선언
} GraphType;

int A[MAX_VERTICES][MAX_VERTICES]; // 상태를 저장하는 2차원 배열
int visited[MAX_VERTICES][MAX_VERTICES]; // 방문을 체크하는 2차원 배열
```

4. 그래프 구조체를 정의한다
5. 정점의 개수를 선언하고 2차원 그래프 가중치 이중 포인터를 선언해서 정점의 개수만큼 동적할당을 할 수 있게 준비한다
6. 상태를 저장하는 2차원 배열을 선언한다
7. 방문을 저장하는 2차원 배열을 선언한다

```

// 그래프 초기화 함수
void init(GraphType *g, int cnt)
{
    int r, c; // 변수 선언
    g->n = 0; // 정점의 개수 0개로 초기화
    // 2차원 배열 초기화
    for (r = 0; r < cnt; r++)
    {
        for (c = 0; c < cnt; c++)
        {
            g->weight[r][c] = INF; // r행 c열의 값을 INF로 초기화
            visited[r][c] = FALSE; // r행 c열을 방문 false 체크
        }
    }
    // r행 r열의 가중치를 0으로 초기화
    for (r = 0; r < cnt; r++)
        g->weight[r][r] = 0;
}

```

8. 그래프 초기화 함수를 선언한다

9. 정점의 개수를 0개로 초기화 하고 행과 열을 cnt크기만큼 반복하면서 inf로 초기화 하고 해당 행과 열의 방문을 false로 체크한다

10. 그 후 r행 r열의 값을 0으로 바꿔준다

```

// 그래프 정점 삽입 함수
void insert_vertex(GraphType *g, int n, int cnt)
{
    // 정점의 개수 확인
    if ((g->n) + 1 > cnt) // 정점의 개수가 최대를 넘을 경우 오류 발생
    {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    // 아닐경우는 정점 1개 추가
    g->n++;
}

```

11. 정점 삽입 함수를 만든다

12. 정점의 개수를 확인하고 오류가 없을 경우 정점 한 개를 추가한다

```

// 간선 삽입 함수
void insert_edge(GraphType *g, int start, int end, int w)
{
    // 받은 간선이 정점의 개수보다 많으면 오류 발생
    if (start >= g->n || end >= g->n)
    {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    // 아닐 경우는 2차원 배열의 가중치 넣기
    g->weight[start][end] = w;
    g->weight[end][start] = w;
}

```

13. 간선 삽입 함수를 선언한다

14. 간선을 체크하고 가능하면 받은 가중치 값을 넣어준다. 단 무방향 그래프여서
행과 열을 바꿔서도 넣는다

```

// 그래프 출력 함수
void printA(GraphType *g)
{
    // 필요한 변수 선언
    int i, j;
    for (int i = 0; i < g->n; i++)
    {
        printf("  %d ", i);
    }
    printf("\n |=====| \n");
    // i가 정점의 크기만큼 반복
    for (i = 0; i < g->n; i++)
    {
        printf("%d | ", i);
        // j가 정점의 크기만큼 반복
        for (j = 0; j < g->n; j++)
        {
            // 만약 A의 i행 j열의 값이 INF일 경우
            if (A[i][j] == INF)
                printf(" x | "); // x표시를 출력
            // 만약 i의 j열의 방문이 TRUE가 아닐 경우
            else if (visited[i][j] != TRUE)
            {
                if (A[i][j] < 10) // 표시 깔끔하게 하기 위한 숫자 구분
                {
                    printf("* %d | ", A[i][j]); // * 값을 출력하고
                    visited[i][j] = TRUE; // TRUE로 변경
                }
                else
                {
                    printf("%d | ", A[i][j]); // * 값을 출력하고
                    visited[i][j] = TRUE; // TRUE로 변경
                }
            }
        }
    }
}

```

15. 그래프 출력 함수를 만든다

16. 필요한 변수를 선언하고 값을 우선 출력한다

17. 정점의 크기만큼 i를 반복하고 j도 마찬가지로 반복한다.

18. 만약 A[i][j]의 크기가 INF면 X를 표시한다

19. 아니면서 i행 j열의 방문이 FALSE일 경우, 값이 10 이상일 경우와 아닐 경우를 구분해서 깔끔하게 *이 붙은 값으로 출력한다. 즉 업그레이드가 되었다는 것이다. 추후 방문을 변경한다

```
// 이마저도 아닐경우
else
{
    if (A[i][j] < 10) // 표시 깔끔하게 하기 위한 숫자 구분
    {
        printf(" %dI ", A[i][j]); // 그냥 동일하기 때문에 출력
    }
    else
    {
        printf(" %dI ", A[i][j]); // 그냥 동일하기 때문에 출력
    }
}
```

```
printf("\n");
```

20. 이 마저도 아니라면 10이상일 경우 아닐 경우를 구분해서 형식에 맞게 출력을 한다

```
//floyd 알고리즘 함수
void floyd(GraphType* g)
{
    int i, j, k; // 필요 변수 선언
    // 그래프의 입력된 값들을 A 2차원 배열에 저장
    for (i = 0; i < g->n; i++) // i가 정점의 크기만큼 반복
        for (j = 0; j < g->n; j++) // j가 정점의 크기만큼 반복
            A[i][j] = g->weight[i][j]; // A의 i행 j열에 값 대입

    // 초기 그래프 출력
    printf("< 초기 상태 >\n");
    printA(g);
}
```

21. Floyd 알고리즘 함수를 선언한다

22. 필요한 변수를 선언한다

23. 이중 for문을 통해 i와 j를 정점의 크기만큼 반복하고 A[i][j]에 그래프의 i행 j열 값들을 넣어준다

```
// k가 정점의 크기만큼 반복
for (k = 0; k < g->n; k++)
{
    // i가 정점의 크기만큼 반복
    for (i = 0; i < g->n; i++)
        // j가 정점의 크기만큼 반복
        for (j = 0; j < g->n; j++)
            // A[i][k] + A[k][j]의 값이 A[i][j]보다 크다면
            if (A[i][k] + A[k][j] < A[i][j])
            {
                A[i][j] = A[i][k] + A[k][j]; // A[i][j]값에 A[i][k] + A[k][j]의 값을 저장
                visited[i][j] = FALSE; // 방문을 안했음을 저장
            }
    // 정점 표시
    printf("< %d번 정점 열림 >\n", k);
    printA(g);
}
```

24. 3중 for문을 만들어서 정점의 크기만큼 반복해준다.

25. 만약 A[i][k]의 값과 A[k][j]의 값을 더한 것이 A[i][j]의 값보다 작다면 더 작은 값이 있다는 것이기 때문에 값을 변경해준다

26. 추후 방문을 변경해주고 그래프를 출력해준다

```
FILE *fp; // 파일 포인터 선언
GraphType *g; // 그래프 포인터 선언
char tmp[100]; // 임시 저장 문자열 변수 선언
int check; // CHECK 변수 선언
// 필요한 임시 저장 변수를 선언
int cnt = 0;
int tmp1, tmp2, tmp3;

// 파일을 열고 실패하면 종료한다
fp = fopen("data01.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return;
}
```

27. 파일 포인터를 선언한다

28. 그래프 포인터를 선언한다

29. 필요한 임시 변수들을 선언한다

30. 파일을 열고 만약 오픈이 실패하면 종료하는 조건문을 만든다

```
// 파일이 끝날때 까지 반복을한다
while (!feof(fp))
{
    // 맨앞 문자를 읽고 v일 경우
    fscanf(fp, "%s", tmp);
    if (strcmp(tmp, "v") == 0)
    {
        fscanf(fp, "%d", &check); // check에 저장하고
        cnt++; // cnt 값을 더한다
    }
    // 아닐경우는 그냥 한줄을 읽는다
    else
    {
        fgets(tmp, 100, fp);
    }
}
// 파일 포인터를 맨앞으로 돌린다
rewind(fp);
```

31. 파일이 끝날 때 까지 반복하면서 앞 문자를 읽고 V면 CNT를 올려준다

32. 아닐 경우는 그냥 한줄만 읽고 끝낸다

33. 파일 포인터를 맨 앞으로 돌린다

```
// 파일 포인터를 맨앞으로 돌린다
rewind(fp);

// 그래프 g를 동적할당을 해준다
g = (GraphType *)malloc(sizeof(GraphType));

// cnt의 크기만큼 그래프를 동적할당 해준다
g->weight = (int **)malloc(sizeof(int*) * cnt);

// 열을 동적할당 해준다
for (int i = 0; i < cnt; i++)
{
    g->weight[i] = (int *)malloc(sizeof(int)*cnt);
}
```

34. 그래프를 동적할당 해주고

35. CNT의 크기만큼 그래프의 행을 동적할당을 해준다

36. 그리고 CNT만큼 반복하면서 1행의 열을 동적할당을 해준다

이렇게 되면 개수 크기의 $CNT * CNT$ 의 2차원 행렬을 만들 수 있다

```
// 그래프를 초기화한다
init(g, cnt);
// 위에서 체크한 cnt 만큼 정점을 넣어준다
for (int i = 0; i < cnt; i++)
    insert_vertex(g, i, cnt);

// 파일이 끝날때 까지 반복한다
while (!feof(fp))
{
    // 맨앞 문자를 읽고 e일 경우
    fscanf(fp, "%s", tmp);
    if (strcmp(tmp, "e") == 0)
    {
        // 값 3개를 읽고 간선 삽입 함수를 호출한다
        fscanf(fp, "%d %d %d", &tmp1, &tmp2, &tmp3);
        insert_edge(g, tmp1, tmp2, tmp3);
    }
    // 아닐경우는 그냥 한줄을 읽는다
    else
    {
        fgets(tmp, 100, fp);
    }
}
```

37. 그래프를 초기화 하고 정점을 넣어준다

38. 파일이 끝날 때 까지 반복하고 맨 앞 문자를 읽고 e일 경우 값 3개를 읽어서 간선 삽입 함수를 호출한다

39. 아닐 경우는 그냥 한줄만 읽는다

```

// floyd 알고리즘 함수를 호출한다
floyd(g);

// 그래프 열의 동적할당을 해제한다
for (int i = 0; i < cnt; i++)
{
    free(g->weight[i]);
}
// 그래프의 행의 동적할당을 해제한다
free(g->weight);
// 그래프를 해제한다
free(g);
//파일을 닫고 종료한다
fclose(fp);
return 0;

```

40. Floyd 알고리즘을 호출한다

41. 그래프의 열부터 동적할당을 해제하고 그래프의 행을 동적할당을 해제한다

42. 그래프를 해제하고 파일을 닫고 종료한다

1.4 실행 결과

선택 Microsoft Visual Studio 디버그 콘솔

< 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 열람 >

	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 열람 >

	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

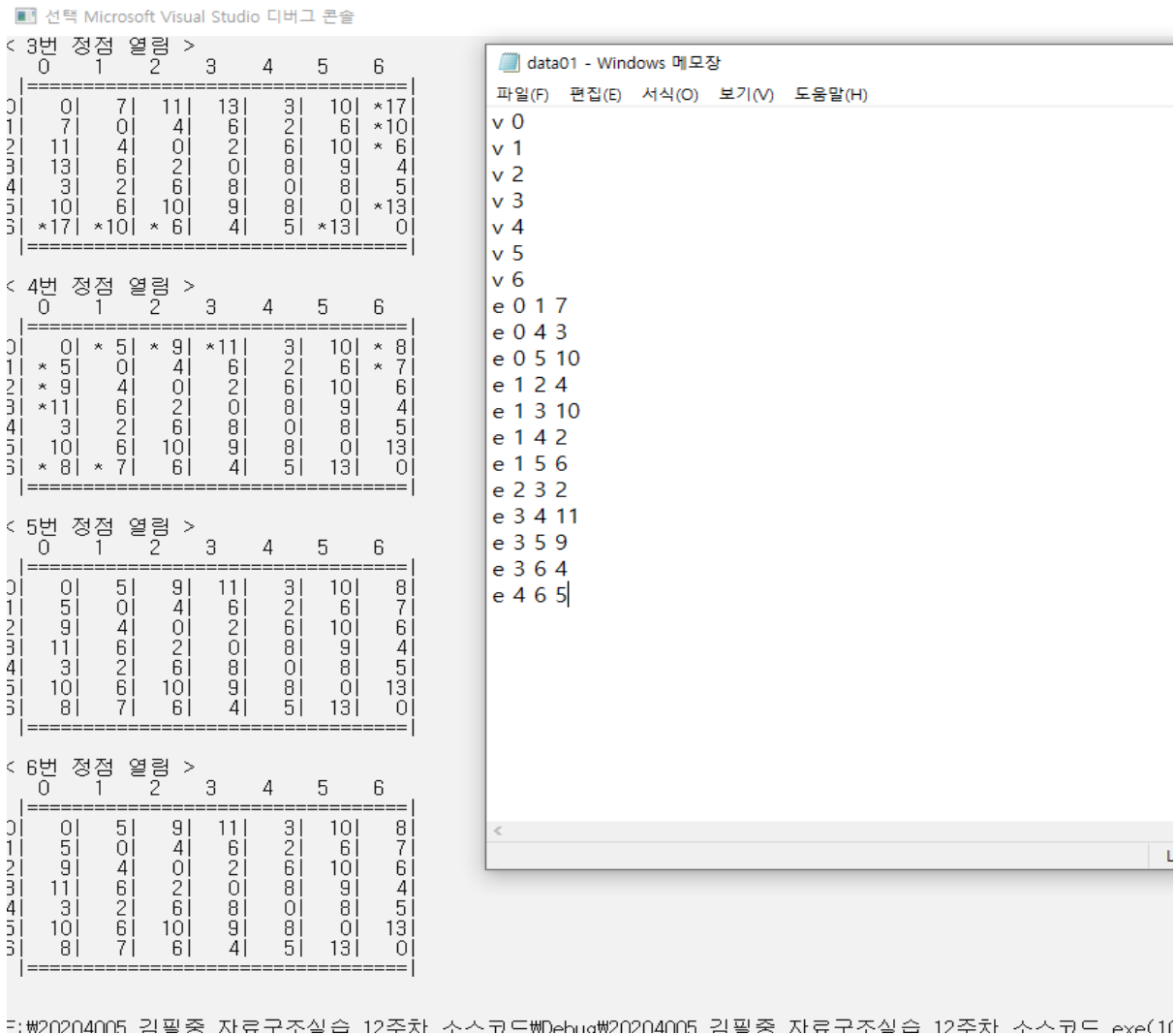
< 2번 정점 열람 >

	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

data01 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
v 0
v 1
v 2
v 3
v 4
v 5
v 6
e 0 1 7
e 0 4 3
e 0 5 10
e 1 2 4
e 1 3 10
e 1 4 2
e 1 5 6
e 2 3 2
e 3 4 11
e 3 5 9
e 3 6 4
e 4 6 5
```



1.5 느낀점

이번 과제를 진행하면서 제일 큰건 Floyd와 Dijkstra 알고리즘 구분이 확실히 가능하다는 것을 배웠다. Dijkstra는 정점에서 마무리 정점까지의 최소를 구하는 것이고 Floyd 알고리즘은 정점에서 정점까지의 연결 중 가장 짧은 것으로 변경하면서 진행하고 더 쉬운 알고리즘이었다. 이를 통해 전철 혹은 다른 이동 수단에서 많이 사용할 수 있다는 장점이 있고 추후 로드 맵을 만들어서 적용시킬 수도 있다는 생각을 했다.

2.1 문제 분석

■ 위상 정렬

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

```
C:\WINDOWS\system32\cmd.exe
< 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
-----
계속하려면 아무 키나 누르십시오 . . .
```

```
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
```

이 문제는 위상 정렬을 이용해 데이터를 위상 정렬하는 문제이다. 문제에서 중요한 요소는 위상 정렬 알고리즘을 이용하면서 데이터를 출력하는 부분이 가장 중요하다고 생각했다. 또한 위상 정렬 문제는 스택을 이용해야 한다는 부분도 중요하다

우선 그래프 노드 구조체를 선언해서 정점의 번호와 그래프 링크를 자기참조를 통해 선언한다

그래프 구조체를 선언하고 정점의 개수와 그래프 배열을 포인터로 선언한다

이름 저장 구조체를 만들어서 이름 저장할 변수를 선언하고 이름 저장 배열을 만들어 준다.

동시에 방문 배열도 만들어 줘서 체크를 통해 값을 출력할 때 도움을 주게 한다

그래프 초기화 함수를 만들어서 그래프 배열을 모두 초기화 한다

정점 삽입 연산을 만들어서 조건에 걸리지 않으면 정점을 한 개를 추가하는 함수를 만들어준다

간선 삽입 연산 함수를 만들어 준다. 우선 그래프 노드를 생성한다. 조건에 걸리지 않으면 노드를 동적할당을 해준다. 노드의 정점의 번호 값에 받은 값을 삽입하고 링크를 이어주는 방식으로 제작한다

스택을 위해 최대 크기 변수를 정의하고 스택 구조체를 선언한다. 스택 구조체 안에는 top 변수와 int형 스택 배열을 만들어 준다

스택 초기화 함수를 만들어서 top의 값을 -1로 초기화 시켜준다

공백 상태 확인 함수를 만들어 만약 top의 값이 -1이면 공백이므로 true를 보내 준다

포화 상태 확인 함수를 만들어 꽉 찼을 경우 true를 리턴하는 함수를 만들어 준다

삽입 함수를 만들어서 스택이 포화인지 확인하고 아닐 경우는 값을 넣어주는 함수를 만들어 준다

삭제 함수를 만들어서 스택이 공백이 아닐 경우 스택 하나를 리턴해주는 함수를

만든다

위상 정렬 함수를 만들어 준다 위상 정렬 함수에서는 스택을 선언하고 그래프 노드를 만들어 준 후 int형 변수를 만들어 정점의 개수만큼 동적할당을 해준다. 초기화를 진행하고 정점의 개수만큼 반복하면서 노드에 간선들을 저장하고 조건에 맞춰 노드를 저장하는 형식으로 만들어야 할 것이다. 스택을 초기화 하고 정점의 크기만큼 반복하면서 위에서 선언한 동적할당의 값이 0이면 값을 푸시를 통해 값을 넣어준다

스택이 공백일때까지 반복하면서 팝을 통해 숫자를 받은 후 w열에 방문을 했는지 확인하고 만약 방문했으면 정점과 과목을 출력, 진입 차수를 변경해준다. 노드가 null이 아닐때까지 반복하면서 새로운 변수에 노드의 정점의 값으로 정해 준 후 진입 차수를 감소시키고 동적할당한 값이 0일 경우 푸시한다. 그 후 다음 정점으로 이동한다. 여기서 별개로 마지막인지 아닌지를 통해 wn을 넣어서 출력하는지 아닌지를 판단하는 부분도 추가로 넣어준다.

2.2 소스 코드

```
1 //=====
2 // 제작일자: 21년 11월 17일 ~ 11월 22일
3 // 제작자: 20204005 김필중
4 // 프로그램명: 위상 정렬 알고리즘
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 필요한 변수들을 정의한다
13 #define TRUE 1
14 #define FALSE 0
15 #define MAX_VERTICES 50
16
17 // 오류 방지 구문을 선택한다
18 #pragma warning(disable : 4996)
19
20 // 그래프 노드 구조체를 선언한다
21 typedef struct GraphNode
22 {
23     int vertex; // 정점 번호
24     struct GraphNode *link; // 그래프 링크
25 } GraphNode;
26
27 // 그래프 구조체를 선언한다
28 typedef struct GraphType {
29     int n; // 정점의 개수
30     GraphNode *adj_list[MAX_VERTICES]; // 그래프 배열 생성
31 } GraphType;
32
33 // 이름 저장 구조체
34 typedef struct name {
35     char nam[100]; // 이름
36 } name;
37
38 // 이름 저장 배열
39 name test[MAX_VERTICES];
40
41 // 방문 배열
42 int visited[MAX_VERTICES];
43
44 // 그래프 초기화
45 void graph_init(GraphType *g)
46 {
47     int v; // 필요한 변수 생성
48     g->n = 0; // 정점의 개수를 0개로 만든다
49     for (v = 0; v < MAX_VERTICES; v++) // MAX_VERTICES만큼 반복한다
50         g->adj_list[v] = NULL; // 그래프 배열을 NULL로 초기화 한다
51 }
52
53 // 정점 사이 연결
```

```

53 // 정점 삽입 연산
54 void insert_vertex(GraphType *g, int v)
55 {
56     // 정점의 개수가 최대를 넘을 경우 오류 발생
57     if (((g->n) + 1) > MAX_VERTICES)
58     {
59         fprintf(stderr, "그래프: 정점의 개수 초과");
60         return;
61     }
62     // 아닐경우는 정점 1개 추가
63     g->n++;
64 }
65 // 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.
66 void insert_edge(GraphType *g, int u, int v)
67 {
68     // 그래프 노드 생성
69     GraphNode *node;
70     // 받은 간선이 정점의 개수보다 많으면 오류 발생
71     if (u >= g->n || v >= g->n)
72     {
73         fprintf(stderr, "그래프: 정점 번호 오류");
74         return;
75     }
76     // 노드를 동적할당을 해준다
77     node = (GraphNode *)malloc(sizeof(GraphNode));
78     node->vertex = v; // node 정점에 v를 삽입한다
79     node->link = g->adj_list[u]; // 노드 링크가 그래프의 u번째 배열을 가리키게 한다
80     g->adj_list[u] = node; // u번째 배열의 그래프를 node로 한다
81 }
82
83 // 최대 스택 크기 변수를 정의한다
84 #define MAX_STACK_SIZE 100
85 // element 변수를 int로 정한다
86 typedef int element;
87
88 // 스택 구조체를 선언한다
89 typedef struct
90 {
91     element stack[MAX_STACK_SIZE]; // element형 스택 배열을 MAX_STACK_SIZE 만큼 만든다
92     int top; // 맨 위 변수
93 } StackType;
94
95 // 스택 초기화 함수
96 void init(StackType *s)
97 {
98     s->top = -1; // top변수를 -1로 정한다
99 }
100

```

```

94
95 // 스택 초기화 함수
96 void init(StackType *s)
97 {
98     s->top = -1; // top변수를 -1로 정한다
99 }
100
101 // 공백 상태 검출 함수
102 int is_empty(StackType *s)
103 {
104     return (s->top == -1); // 만약 top이 -1이면 공백 true 아닐경우는 false
105 }
106
107 // 포화 상태 검출 함수
108 int is_full(StackType *s)
109 {
110     return (s->top == (MAX_STACK_SIZE - 1)); // 만약 top이 최대치 - 1 같을 경우 true 반환 아닐경우는 false 반환
111 }
112
113 // 삽입함수
114 void push(StackType *s, element item)
115 {
116     // 만약 스택이 꽉차면 오류 발생
117     if (is_full(s))
118     {
119         fprintf(stderr, "스택 포화 에러\n");
120         return;
121     }
122     // 아닐 경우는 하나를 증가시키고 item 값 삽입
123     else
124         s->stack[++(s->top)] = item;
125 }
126
127 // 삭제함수
128 element pop(StackType *s)
129 {
130     // 만약 스택이 비어있으면 오류 발생
131     if (is_empty(s))
132     {
133         fprintf(stderr, "스택 공백 에러\n");
134         exit(1);
135     }
136     // 아닐 경우 스택 하나를 리턴하고 top 값 감소
137     else
138         return s->stack[(s->top)--];
139 }
140

```

```

141 // 위상정렬 함수
142 int topo_sort(GraphType *g, int num)
143 {
144     int i; // 필요한 변수 선언
145     StackType s; // 스택 s 선언
146     GraphNode *node; // 그래프 노드 선언
147
148     // 모든 정점의 진입 차수를 계산한다
149     int *in_degree = (int *)malloc(g->n * sizeof(int)); // in_degree 포인터를 정점의 개수 * int 크기만큼 동적할당을 해준다
150     // 초기화를 진행한다
151     for (i = 0; i < g->n; i++)
152         in_degree[i] = 0;
153     // 정점의 개수만큼 반복한다
154     for (i = 0; i < g->n; i++)
155     {
156         GraphNode *node = g->adj_list[i]; //정점 i에서 나오는 간선들을 저장
157         // 만약 node가 null 일 경우
158         while (node != NULL)
159         {
160             in_degree[node->vertex]++; // in_degree 크기 증가를 한다
161             node = node->link; // 노드를 연결한다
162         }
163     }
164     // 진입 차수가 0인 정점을 스택에 삽입한다
165     init(&s); // 스택 초기화
166     for (i = 0; i < g->n; i++) // 정점의 크기만큼 반복
167     {
168         if (in_degree[i] == 0) // in_degree값이 0이면
169             push(&s, i); // 값을 넣는다
170     }
171
172     // 위상 순서를 생성 한다
173     while (!is_empty(&s)) // 스택이 공백일때 까지 반복한다
174     {
175         int w;
176         w = pop(&s); // w에 맨 값을 넣어준다
177         if (w == num) // 만약 w가 마지막 이라면
178         {
179             if (visited[w] == TRUE) // w열에 방문했으면
180             {
181                 printf("정점 %d - %s", w, test[w].nam); //정점과 과목 출력
182                 node = g->adj_list[w]; //각 정점의 진입 차수를 변경
183                 // node가 null이 아닐때까지 반복한다
184                 while (node != NULL)
185                 {
186                     int u = node->vertex; // u에 노드의 정점의 값으로 정의한다
187                     in_degree[u]--; //진입 차수를 감소시킨다
188                     if (in_degree[u] == 0) // 만약 u번째 in_degree가 0이면
189                         push(&s, u); // 삽입한다
190                     node = node->link; // 다음 정점
191                 }
192             }
193         }

```

```

193     }
194     // 만약 마지막이 아닐경우
195     else
196     {
197         if (visited[w] == TRUE) // w열에 방문했으면
198         {
199             printf("정점 %d - %s\n", w, test[w].nam); //정점과 과목 출력 \n 포함해서
200             node = g->adj_list[w]; //각 정점의 진입 차수를 변경
201             // node가 null이 아닐때까지 반복한다
202             while (node != NULL)
203             {
204                 int u = node->vertex; // u에 노드의 정점의 값으로 정의한다
205                 in_degree[u]--; //진입 차수를 감소시킨다
206                 if (in_degree[u] == 0) // 만약 u번째 in_degree가 0이면
207                     push(&s, u); // 삽입한다
208                 node = node->link; // 다음 정점
209             }
210         }
211     }
212 }
213 free(in_degree); // 해제한다
214 printf("\n");
215 return (i == g->n); // 반환값이 1이면 성공, 0이면 실패
216 }

```

```

217 // 메인 함수
218 int main(void)
219 {
220     FILE *fp; // 파일 포인터 선언
221     GraphType *g; // 그래프 포인터 선언
222     // 필요한 변수 선언
223     char tmp[100];
224     int check;
225     int tmp1, tmp2;
226     int max = 0;
227
228     // 파일을 열고 실패하면 종료한다
229     fp = fopen("data02.txt", "r");
230
231     if (fp == NULL)
232     {
233         printf("파일 오픈 실패\n");
234         return;
235     }
236
237     printf("< 데이터 >\n");
238     printf("=====n");
239
240     // 파일 끝까지 반복한다
241     while (!feof(fp))
242     {
243         // 파일 맨 앞을 읽어서 v일 경우
244         fscanf(fp, "%s", tmp);
245         if (strcmp(tmp, "v") == 0)
246         {
247             fscanf(fp, "%d %s", &check, tmp); // 값을 읽는다
248             strcpy(test[check].nam, tmp); // test 배열에 이름을 넣는다
249             printf("%d - %s\n", check, tmp); // 출력한다
250             visited[check] = TRUE; // 방문 표시를 남긴다
251             // 만약 max 값 보다 크면 교체한다
252             if (max < check)
253                 max = check;
254         }
255         // 아닐 경우는 그냥 한줄을 읽는다
256         else
257         {
258             fgets(tmp, 100, fp);
259         }
260     }
261
262     printf("=====n");

```

```

262
263 // 파일 포인터를 맨앞으로 돌린다
264 rewind(fp);
265
266 // 그래프 동적할당
267 g = (GraphType *)malloc(sizeof(GraphType));
268 // 그래프 초기화
269 graph_init(g);
270
271 // 최대 크기 + 1만큼 반복
272 for (int i = 0; i < max + 1; i++)
273     insert_vertex(g, i);
274
275 printf("\n\n\n");
276
277 // 파일 끝까지 반복한다
278 while (!feof(fp))
279 {
280     // 맨 앞이 e 일 경우
281     fscanf(fp, "%s", tmp);
282     if (strcmp(tmp, "e") == 0)
283     {
284         fscanf(fp, "%d %d", &tmp1, &tmp2); // 값을 읽고
285         insert_edge(g, tmp1, tmp2); // 간선 삽입을 한다
286     }
287     // 아닐경우 한줄을 읽는다
288     else
289     {
290         fgets(tmp, 100, fp);
291     }
292 }
293
294
295 //위상 정렬
296 printf("< 위상 정렬 출력 >\n");
297 printf("=====\n");
298 topo_sort(g, max); // 위상 정렬 호출
299 printf("=====\n");
300 // 동적할당 해제, 파일 닫기, 종료
301 free(g);
302 fclose(fp);
303 return 0;
304 }

```

2.3 소스 코드 분석

```
1  //=====
2  // 제작일자: 21년 11월 17일 ~ 11월 22일
3  // 제작자: 20204005 김필중
4  // 프로그램명: 위상 정렬 알고리즘
5  //=====
6
7  // 필요한 헤더파일을 선언한다
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 // 필요한 변수들을 정의한다
13 #define TRUE 1
14 #define FALSE 0
15 #define MAX_VERTICES 50
16
17 // 오류 방지 구문을 선택한다
18 #pragma warning(disable : 4996)
```

1. 필요한 헤더파일을 선언한다
2. 필요한 변수들을 정의한다
3. 오류 방지 구문을 선언한다

```

19
20 // 그래프 노드 구조체를 선언한다
21 typedef struct GraphNode
22 {
23     int vertex; // 정점 번호
24     struct GraphNode *link; // 그래프 링크
25 } GraphNode;
26
27 // 그래프 구조체를 선언한다
28 typedef struct GraphType {
29     int n; // 정점의 개수
30     GraphNode *adj_list[MAX_VERTICES]; // 그래프 배열 생성
31 } GraphType;
32
33 // 이름 저장 구조체
34 typedef struct name {
35     char nam[100]; // 이름
36 } name;
37
38 // 이름 저장 배열
39 name test[MAX_VERTICES];
40
41 // 방문 배열
42 int visited[MAX_VERTICES];
43

```

4. 그래프 노드 구조체를 선언한다
5. 노드 구조체 안에 정점의 번호 변수와 그래프 자기 참조 링크를 선언한다
6. 그래프 구조체를 선언한다.
7. 정점의 개수와 그래프 배열을 포인터로 선언한다
8. 이름 저장 구조체를 선언하고 구조체 안에 이름 변수를 저장한다
9. 이름 저장 배열, 방문 배열을 만들어 준다

```

44 // 그래프 초기화
45 void graph_init(GraphType *g)
46 {
47     int v; // 필요한 변수 생성
48     g->n = 0; // 정점의 개수를 0개로 만든다
49     for (v = 0; v < MAX_VERTICES; v++) // MAX_VERTICES만큼 반복한다
50         g->adj_list[v] = NULL; // 그래프 배열을 NULL로 초기화 한다
51 }
52

```

10. 그래프 초기화 함수를 만들어서 필요한 변수를 생성한다.
11. 정점의 개수를 0개로 초기화하고 그래프의 배열을 null로 초기화한다


```

53 // 정점 삽입 연산
54 void insert_vertex(GraphType *g, int v)
55 {
56     // 정점의 개수가 최대를 넘을 경우 오류 발생
57     if (((g->n) + 1) > MAX_VERTICES)
58     {
59         fprintf(stderr, "그래프: 정점의 개수 초과");
60         return;
61     }
62     // 아닐 경우는 정점 1개 추가
63     g->n++;
64 }

```

12. 정점 삽입 연산 함수를 만든다

13. 조건에 걸리지 않는다면 정점 1개를 추가하는 함수이다

```

65 // 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.
66 void insert_edge(GraphType *g, int u, int v)
67 {
68     // 그래프 노드 생성
69     GraphNode *node;
70     // 받은 간선이 정점의 개수보다 많으면 오류 발생
71     if (u >= g->n || v >= g->n)
72     {
73         fprintf(stderr, "그래프: 정점 번호 오류");
74         return;
75     }
76     // 노드를 동적할당을 해준다
77     node = (GraphNode *)malloc(sizeof(GraphNode));
78     node->vertex = v; // node 정점에 v를 삽입한다
79     node->link = g->adj_list[u]; // 노드 링크가 그래프의 u번째 배열을 가리키게 한다
80     g->adj_list[u] = node; // u번째 배열의 그래프를 node로 한다
81 }
82

```

14. 간선 삽입 연산 함수를 만들어 준다.

15. 그래프 노드를 생성하고 조건을 체크해서 오류 체크를 진행한다

16. 만약 아닐 경우는 노드를 동적할당을 하고 node 정점에 v를 삽입한다

17. 노드 링크가 그래프의 u번째 배열을 가리키게 하고 u 번째 배열 그래프를 node로 한다

```

83 // 최대 스택 크기 변수를 정의한다
84 #define MAX_STACK_SIZE 100
85 // element 변수를 int로 정한다
86 typedef int element;
87
88 // 스택 구조체를 선언한다
89 typedef struct
90 {
91     element stack[MAX_STACK_SIZE]; // element형 스택 배열을 MAX_STACK_SIZE 만큼 만든다
92     int top; // 맨 위 변수
93 } StackType;

```

18. 최대 스택 크기 변수를 정의하고 element를 정의한다

19. 스택 구조체를 만들어서 element형 스택 배열을 선언하고 top 변수를 선언한다

```

95 // 스택 초기화 함수
96 void init(StackType *s)
97 {
98     s->top = -1; // top변수를 -1로 정한다
99 }
100
101 // 공백 상태 검출 함수
102 int is_empty(StackType *s)
103 {
104     return (s->top == -1); // 만약 top이 -1이면 공백 true 아닐경우는 false
105 }
106
107 // 포화 상태 검출 함수
108 int is_full(StackType *s)
109 {
110     return (s->top == (MAX_STACK_SIZE - 1)); // 만약 top이 최대치 - 1 같을 경우 true 반환 아닐경우는 false 반환
111 }
112

```

20. 스택 초기화 함수를 만들어서 top변수를 -1로 초기화 시킨다

21. 공백 상태 검출 함수를 만들어서 만약 top이 -1이라면 공백이므로 true 반환한다

22. 포화 상태 검출 함수를 만들어서 만약 top이 최대치 -1의 값과 같을 경우 포화이므로 true를 반환한다

```

113     // 삽입함수
114 void push(StackType *s, element item)
115 {
116     // 만약 스택이 꽉차면 오류 발생
117     if (is_full(s))
118     {
119         fprintf(stderr, "스택 포화 에러\n");
120         return;
121     }
122     // 아닐 경우는 하나를 증가시키고 item 값 삽입
123     else
124         s->stack[++(s->top)] = item;
125 }
126

```

23. 삽입 함수를 만든다

24. 만약 스택이 포화이면 오류를 출력하고 아닐 경우는 stack 배열 top+1 번째에 item 값을 넣는다

```

127     // 삭제함수
128 element pop(StackType *s)
129 {
130     // 만약 스택이 비어있으면 오류 발생
131     if (is_empty(s))
132     {
133         fprintf(stderr, "스택 공백 에러\n");
134         exit(1);
135     }
136     // 아닐 경우 스택 하나를 리턴하고 top 값 감소
137     else
138         return s->stack[(s->top)--];
139 }
140

```

25. 삭제 함수는 우선 스택이 공백인지 확인하고 공백이면 오류를 출력한다

26. 아닐 경우는 s->top번째의 스택 배열의 값을 리턴하고 하나 감소시킨다

```

141 // 리턴 값
142 int topo_sort(GraphType *g, int num)
143 {
144     int i; // 필요한 변수 선언
145     StackType s; // 스택 s 선언
146     GraphNode *node; // 그래프 노드 선언
147
148     // 모든 정점의 진입 차수를 계산한다
149     int *in_degree = (int *)malloc(g->n * sizeof(int)); // in_degree 포인터를 정점의 개수 * int 크기만큼 동적할당을 해준다
150     // 초기화를 진행한다
151     for (i = 0; i < g->n; i++)
152         in_degree[i] = 0;
153     // 정점의 개수만큼 반복한다
154     for (i = 0; i < g->n; i++)
155     {
156         GraphNode *node = g->adj_list[i]; //정점 i에서 나오는 간선들을 저장
157         // 만약 node가 null 일 경우
158         while (node != NULL)
159         {
160             in_degree[node->vertex]++; // in_degree 크기 증가를 한다
161             node = node->link; // 노드를 연결한다
162         }
163     }
164     // 진입 차수가 0인 정점을 스택에 삽입한다
165     init(&s); // 스택 초기화
166     for (i = 0; i < g->n; i++) // 정점의 크기만큼 반복
167     {
168         if (in_degree[i] == 0) // in_degree 값이 0이면
169             push(&s, i); // 값을 넣는다
170     }
171 }

```

27. 위상 정렬 함수를 만든다

28. 필요한 변수를 선언하고 스택, 그래프 노드를 선언한다

29. in_degree 변수를 포인터로 선언한다. 정점의 개수 * int의 크기만큼 동적할당을 해준다

30. in_degree 변수를 0으로 초기화를 한다

31. 정점의 개수만큼 반복하면서 node에 정점 i에서 나오는 간선들을 저장한다

32. 만약 node가 null이 아닐 때 까지 반복하면서 in_degree의 정점 번호 배열의 크기를 증가시키고 노드를 연결한다.

33. 스택을 초기화 하고 정점의 크기만큼 반복하면서 in_degree 값이 0이면 i 를 푸시한다

```

// 위상 순서를 생성 한다
while (!is_empty(&s)) // 스택이 공백일때 까지 반복한다
{
    int w;
    w = pop(&s); // w에 뺀 값을 넣어준다
    if (w == num) // 만약 w가 마지막 이라면
    {
        if (visited[w] == TRUE) // w열에 방문했으면
        {
            printf("정점 %d - %s", w, test[w].nam); //정점과 과목 출력
            node = g->adj_list[w]; //각 정점의 진입 차수를 변경
            // node가 null이 아닐때까지 반복한다
            while (node != NULL)
            {
                int u = node->vertex; // u에 노드의 정점의 값으로 정의한다
                in_degree[u]--; //진입 차수를 감소시킨다
                if (in_degree[u] == 0) // 만약 u번째 in_degree가 0이면
                    push(&s, u); // 삽입한다
                node = node->link; // 다음 정점
            }
        }
    }
}

```

34. 스택이 공백일때까지 반복하면서 w에 팝한 값을 넣어준다
35. 만약 w가 마지막 값일 경우, w번째 배열에 방문했을 경우 정점과 과목을 wn 없이 출력하고 각 정점의 진입 차수를 변경한다.
36. node가 null이 아닐때까지 반복하면서 u를 노드의 정점의 값으로 한다
37. in_degree[u]의 값을 감소 시키고 만약 u번째 in_degree 값이 0이면 u를 삽입하고 다음 정점을 가리키게 한다

```

// 만약 마지막이 아닐경우
else
{
    if (visited[w] == TRUE) // w열에 방문했으면
    {
        printf("정점 %d - %s\n", w, test[w].nam); //정점과 과목 출력 \n 포함해서
        node = g->adj_list[w]; //각 정점의 진입 차수를 변경
        // node가 null이 아닐때까지 반복한다
        while (node != NULL)
        {
            int u = node->vertex; // u에 노드의 정점의 값으로 정의한다
            in_degree[u]--; //진입 차수를 감소시킨다
            if (in_degree[u] == 0) // 만약 u번째 in_degree가 0이면
                push(&s, u); // 삽입한다
            node = node->link; // 다음 정점
        }
    }
}
}

```

38. 만약 마지막이 아니라면 위와 차이점은 \n 없이 출력하게 하는것이다.

```

free(in_degree); // 해제한다
printf("\n");
return (i == g->n); // 반환값이 1이면 성공, 0이면 실패

```

39. 동적할당을 해제한다

40. 반환 값이 1이면 성공 0이면 실패라는 의미로 반환한다

```

// 메인 함수
int main(void)
{
    FILE *fp; // 파일 포인터 선언
    GraphType *g; // 그래프 포인터 선언
    // 필요한 변수 선언
    char tmp[100];
    int check;
    int tmp1, tmp2;
    int max = 0;

    // 파일을 열고 실패하면 종료한다
    fp = fopen("data02.txt", "r");

    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return;
    }
}

```

41. 파일 포인터를 선언하고 그래프 포인터를 선언한다
42. 필요한 변수들을 선언한다
43. 파일을 열고 실패하면 종료하게 한다

```

// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 파일 맨 앞을 읽어서 v일 경우
    fscanf(fp, "%s", tmp);
    if (strcmp(tmp, "v") == 0)
    {
        fscanf(fp, "%d %s", &check, tmp); // 값을 읽는다
        strcpy(test[check].nam, tmp); // test 배열에 이름을 넣는다
        printf("%d - %s\n", check, tmp); // 출력한다
        visited[check] = TRUE; // 방문 표시를 남긴다
        // 만약 max 값 보다 크면 교체한다
        if (max < check)
            max = check;
    }
    // 아닐 경우는 그냥 한줄을 읽는다
    else
    {
        fgets(tmp, 100, fp);
    }
}
printf("=====\n");
// 파일 포인터를 맨앞으로 돌린다
rewind(fp);

```

44. 파일을 끝까지 반복한다
45. 파일 맨 앞을 읽은 후 v라면 값을 읽고 test 배열에 이름을 넣고 값을 출력한다
46. 방문 배열에 표시를 남기고 max 값 비교를 통해 최대값을 찾는다
47. 아닐 경우는 그냥 한 줄을 읽는다
48. 파일 포인터를 앞으로 돌린다


```

// 그래프 동적할당
g = (GraphType *)malloc(sizeof(GraphType));
// 그래프 초기화
graph_init(g);

// 최대 크기 + 1만큼 반복
for (int i = 0; i < max + 1; i++)
    insert_vertex(g, i);

printf("\n\n\n");

// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 맨 앞이 e 일 경우
    fscanf(fp, "%s", tmp);
    if (strcmp(tmp, "e") == 0)
    {
        fscanf(fp, "%d %d", &tmp1, &tmp2); // 값을 읽고
        insert_edge(g, tmp1, tmp2); // 간선 삽입을 한다
    }
    // 아닐경우 한줄을 읽는다
    else
    {
        fgets(tmp, 100, fp);
    }
}

```

49. 그래프를 동적할당을 하고 그래프를 초기화 한다
50. 최대 크기 +1 만큼 반복하면서 정점을 삽입한다
51. 파일 끝까지 반복하면서 만약 맨 앞이 e일 경우
52. 값을 읽고 간선 삽입을 한다
53. 아닐 경우는 한 줄을 읽게 한다

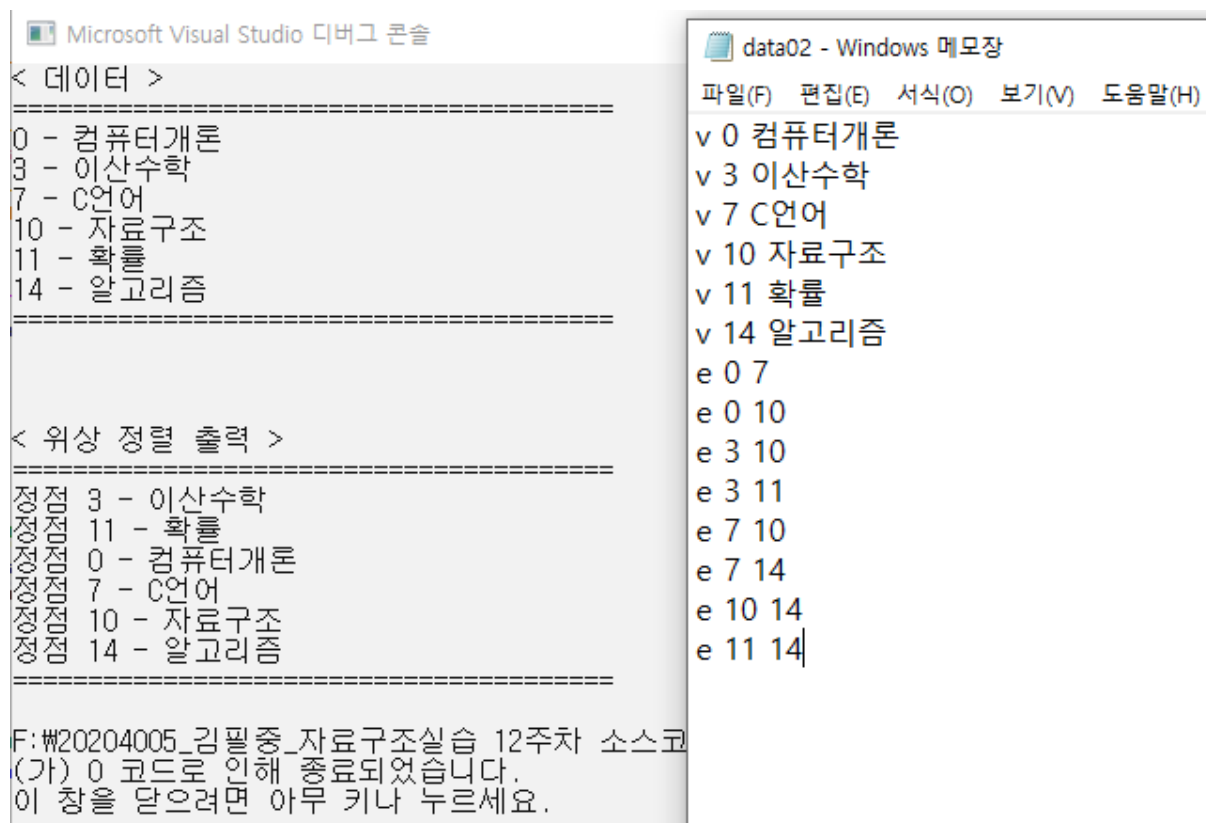
```

//위상 정렬
printf("< 위상 정렬 출력 >\n");
printf("=====\n");
topo_sort(g, max); // 위상 정렬 호출
printf("=====\n");
// 동적할당 해제, 파일 닫기, 종료
free(g);
fclose(fp);
return 0;

```

54. 위상 정렬을 호출하고 동적할당을 해제한다
55. 파일닫고 종료한다

2.4 실행 결과



The image shows two windows from a Windows environment. The left window is the 'Microsoft Visual Studio 디버그 콘솔' (Microsoft Visual Studio Debug Console). It displays the following text:

```
< 데이터 >
=====
0 - 컴퓨터개론
3 - 이산수학
7 - C언어
10 - 자료구조
11 - 확률
14 - 알고리즘
=====

< 위상 정렬 출력 >
=====
정점 3 - 이산수학
정점 11 - 확률
정점 0 - 컴퓨터개론
정점 7 - C언어
정점 10 - 자료구조
정점 14 - 알고리즘
=====

F:\20204005_김필중_자료구조실습_12주차 소스코
(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

The right window is a 'data02 - Windows 메모장' (Notepad) window. It contains the following text:

```
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
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 느낀점

처음에는 위상 정렬이 생각보다 쉽다고 느꼈었다. 그 이유는 그림으로 그렸을 때 하나씩 제거하면서 차수를 줄여나가는 방식이 생각보다 쉬웠으나 막상 알고리즘을 만들어 보면서 느낀 것은 더 복잡하고 고려해야 할 상황이 많아 졌다는 것이다. 하지만 이 알고리즘을 이용하면 다양한 프로그램을 만들 수 있다는 것을 알 수 있다. 문제처럼 선 이수 과목을 받을 수 있고 아니면 게임이나 다른 분야에서도 적용이 가능하다고 생각했다. 아직 위상정렬은 더 공부해야겠다고 느꼈다.

3.1 문제 분석



선택 정렬

■ 선택 정렬 프로그램

- results_stpes.txt에는 우리대학교 SW중심대학 사업단에서 운영하고 있는 사업인 웰라이프 생활-실습형 BLEP(Bigdata-based Living lab Education Platform)의 데이터로, Fitbit을 이용하여 활동 데이터로 걸음수 정보가 저장되어 있다. 많이 걸음을 걸은 날부터 출력하도록 선택정렬을 이용하여 정렬 하시오.

- 동적 할당을 이용하여 데이터를 저장하시오.

C:\WINDOWS\system32\cmd.exe

** Selection Sort 정렬전 출력 **

날짜	걸음수
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	0

C:\WINDOWS\system32\cmd.exe

** Selection Sort 결과 출력 **

날짜	걸음수
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
2021-09-09	10361
2021-04-28	10243
2021-04-07	10174
2021-09-12	9828
2021-11-08	9823
2021-11-09	9645
2021-06-04	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
2021-06-01	8591
2021-04-11	8467
2021-04-12	8262
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

results_stpes - Windows 메모

파일(F) 편집(E) 서식(O) 보기

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

문제에서 데이터를 읽어서 걸음 순서대로 선택 정렬 알고리즘을 이용해 많은 순서대로 나열하는 것이 핵심이다. 우선 선택 정렬 알고리즘을 만들어 문제를 해결해야 하고 또한 동적할당을 해야하기 때문에 총 개수 만큼 동적할당을 해줘야 한다.

제일 먼저 스왑 기능을 정의한다

우선 리스트 구조체를 선언해서 날짜 저장 변수와 걸음 수 변수를 선언한다

Int형 변수를 선언하고 선택 정렬 함수를 만들어 준다.

선택 정렬 함수에서는 필요한 변수를 선언하고 반복을 하면서 여기서는 최대 값을 찾아야 하기 때문에 최대 값을 찾는 반복문을 진행한다. 최대값을 찾을 경우 값을 변경하고 스왑을 통해 걸음수를 변경한다 또한 날짜도 같이 변경해준다.

메인 함수에서는 필요한 파일 포인터를 선언하고 변수들을 선언한다

리스트 구조체를 포인터로 선언한다, 그 후 데이터를 센 후 데이터의 크기만큼 동적할당을 해준다

파일을 읽으면서 값들을 리스트 배열에 잘 저장을 하면서 n을 계속 증가시킨다

3.2 소스 코드

```
1 //=====
2 // 제작일자: 21년 11월 17일 ~ 11월 22일
3 // 제작자: 20204005 김필중
4 // 프로그램명: 발걸음 데이터 선택 정렬
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 필요한 변수들을 정의한다
13 #define MAX_SIZE 200
14 #define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) ) // 스왑 정의
15
16 // 오류 방지 구문을 선택한다
17 #pragma warning(disable : 4996)
18
19 // 리스트 구조체를 선언한다
20 typedef struct list
21 {
22     char date[20]; // 날짜 저장 변수
23     int walk; // 걸음수 변수
24 }list;
25
26 // int형 n 변수 선언
27 int n;
28
29 // 선택 정렬 함수
30 void selection_sort(list l[], int n)
31 {
32     int i, j, least, temp; // 필요한 변수를 선언한다
33     char tmp[20]; // 날짜를 저장할 변수 선언
34     // i가 n-1 번 만큼 반복한다
35     for (i = 0; i < n - 1; i++)
36     {
37         least = i; // least 값을 i로 저장한다
38         for (j = i + 1; j < n; j++) // 최대값 탐색
39             if (l[j].walk > l[least].walk) // 만약 j의 걸음 수 보다 least 배열의 걸음수가 작을 경우
40                 least = j; // least의 값을 j로 바꾼다
41         SWAP(l[i].walk, l[least].walk, temp); // 스왑 실행
42         // 날짜도 같이 변경해준다
43         strcpy(tmp, l[i].date);
44         strcpy(l[i].date, l[least].date);
45         strcpy(l[least].date, tmp);
46     }
47 }
48
49 // 메인 함수
```

```

49 // 메인 함수
50 int main(void)
51 {
52     FILE *fp; // 파일 포인터 fp를 선언
53     // 필요한 변수들을 선언
54     int tmp_walk;
55     int n = 0;
56     int cnt = 0;
57     char tmp_date[20];
58
59     // list 구조체 포인터 선언
60     list *user;
61
62     // 파일을 열고 실패하면 종료한다
63     fp = fopen("data03.txt", "r");
64
65     if (fp == NULL)
66     {
67         printf("파일 오픈 실패\n");
68         return;
69     }
70
71     while (!feof(fp))
72     {
73         fscanf(fp, "%s %d", tmp_date, &tmp_walk); // 파일 값을 읽는다
74         cnt++; // cnt를 더한다
75     }
76     rewind(fp); // 파일 포인터를 앞으로 돌린다
77
78     user = (list *)malloc(sizeof(list) * cnt); // cnt의 크기만큼 동적할당을 해준다
79
80     printf("*** Selection Sort 정렬전 출력 **\n");
81     printf("=====n");
82     printf("    날짜   tt   걸음수\n");
83     printf("=====n");
84
85     // 파일 끝날때 까지 반복
86     while (!feof(fp))
87     {
88         fscanf(fp, "%s %d", tmp_date, &tmp_walk); // 파일 값을 읽는다
89         user[n].walk = tmp_walk; // n번째 리스트에 걸음수를 저장한다
90         strcpy(user[n].date, tmp_date); // n번째 리스트에 날짜를 저장한다
91         printf("    %s   %d\n", tmp_date, tmp_walk); // 출력한다
92         n++; // n을 더한다
93     }
94     printf("n\n=====n");
95     printf("*** Selection Sort 결과 출력 **\n");
96     printf("=====n");
97     printf("    날짜   tt   걸음수\n");
98     printf("=====n");
99
100    selection_sort(user, n); // 선택정렬 호출한다
101
102
103    for (int i = 0; i < n; i++)
104        printf("    %s   %d\n", user[i].date, user[i].walk);
105    printf("=====n");
106
107    // 파일을 닫는다
108    free(user);
109    fclose(fp);
110    return 0;
111 }

```

3.3 소스 코드 분석

```
1 //=====
2 // 제작일자: 21년 11월 17일 ~ 11월 22일
3 // 제작자: 20204005 김필중
4 // 프로그램명: 발걸음 데이터 선택 정렬
5 //=====
6
7 // 필요한 헤더파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 필요한 변수들을 정의한다
13 #define MAX_SIZE 200
14 #define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) ) // 스왑 정의
15
16 // 오류 방지 구문을 선택한다
17 #pragma warning(disable : 4996)
18
19 // 리스트 구조체를 선언한다
20 typedef struct list
21 {
22     char date[20]; // 날짜 저장 변수
23     int walk; // 걸음수 변수
24 }list;
25
26 // int형 n 변수 선언
27 int n;
```

1. 필요한 헤더파일을 선언한다
2. 필요한 변수들을 정의한다
3. 스왑 정의를 통해 값을 변경해 주는 매크로를 만들어 준다
4. 오류 방지 구문을 선언한다
5. 리스트 구조체를 정의하고 안에는 날짜 저장 변수와 걸음 수 변수를 저장한다
6. int형 변수를 선언한다

```

// 선택 정렬 함수
void selection_sort(list l[], int n)
{
    int i, j, least, temp; // 필요한 변수를 선언한다
    char tmp[20]; // 날짜를 저장할 변수 선언
    // i가 n-1 번 만큼 반복한다
    for (i = 0; i < n - 1; i++)
    {
        least = i; // least 값을 i로 저장한다
        for (j = i + 1; j < n; j++) // 최대값 탐색
            if (l[j].walk > l[least].walk) // 만약 j의 걸음 수 보다 least 배열의 걸음수가 작을 경우
                least = j; // least의 값을 j로 바꾼다
        SWAP(l[i].walk, l[least].walk, temp); // 스왑 실행
        // 날짜도 같이 변경해준다
        strcpy(tmp, l[i].date);
        strcpy(l[i].date, l[least].date);
        strcpy(l[least].date, tmp);
    }
}

```

7. 선택 정렬 함수를 만든다

8. 필요한 변수들을 선언하고 날짜를 저장할 변수를 선언한다

9. i가 n - 1만큼 반복하면서 least의 값을 i로 저장한다

10. 그 후 최대값 탐색을 위한 반복문을 진행하면서 만약 j의 리스트 걸음 수 보다 least 번째 배열의 걸음수가 작을 경우 least의 값을 j로 바꾼다

11. 스왑을 실행 한다. 날짜도 같이 변경하는 구문을 진행한다

```

FILE *fp; // 파일 포인터 fp를 선언
// 필요한 변수들을 선언
int tmp_walk;
int n = 0;
int cnt = 0;
char tmp_date[20];

// list 구조체 포인터 선언
list *user;

// 파일을 열고 실패하면 종료한다
fp = fopen("data03.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return;
}

```

12. 파일 포인터 fp를 선언하고 필요한 변수들을 선언한다

13. list 구조체 포인터를 선언한다

14. 파일을 열고 실패하면 종료한다

```
while (!feof(fp))
{
    fscanf(fp, "%s %d", tmp_date, &tmp_walk); // 파일 값을 읽는다
    cnt++; // cnt를 더한다
}
rewind(fp); // 파일 포인터를 앞으로 돌린다
```

15. 파일이 끝날 때 까지 반복하면서 파일 값을 읽은 후 CNT를 더한다.

16. 파일 포인터를 맨 앞으로 옮긴다

```
user = (list *)malloc(sizeof(list) * cnt); // cnt의 크기만큼 동적할당을 해준다
```

17. 앞에서 선언한 user를 CNT의 크기만큼 동적할당을 해준다

```
// 파일 끝날때 까지 반복
while (!feof(fp))
{
    fscanf(fp, "%s %d", tmp_date, &tmp_walk); // 파일 값을 읽는다
    user[n].walk = tmp_walk; // n번째 리스트에 걸음수를 저장한다
    strcpy(user[n].date, tmp_date); // n번째 리스트에 날짜를 저장한다
    printf(" %s %d\n", tmp_date, tmp_walk); // 출력한다
    n++; // n을 더한다
}
```

18. 파일이 끝날 때 까지 반복하면서 파일 값을 읽고 날짜와 걸음 수를 리스트에 저장한다, 출력 후 N을 더한다

```

selection_sort(user, n); // 선택정렬 호출한다

for (int i = 0; i < n; i++)
    printf(" %s %tXd\n", user[i].date, user[i].walk);
printf("=====n");

// 파일을 닫는다
free(user);
fclose(fp);
return 0;

```

19. 선택 정렬을 호출한다
20. 정렬된 리스트를 출력한다
21. 동적할당을 해제하고 파일을 닫고 종료한다

3.4 실행 결과

Microsoft Visual Studio 디버그 콘솔		data03 - Windows 메모장	
** Selection Sort 정렬전 출력 **		파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)	
날짜	걸음수		
2021-04-07	10174	2021-04-07 10174	
2021-04-08	4829	2021-04-08 4829	
2021-04-09	8262	2021-04-09 8262	
2021-04-10	8864	2021-04-10 8864	
2021-04-11	8467	2021-04-11 8467	
2021-04-12	5120	2021-04-12 5120	
2021-04-13	10557	2021-04-13 10557	
2021-04-14	10594	2021-04-14 10594	
2021-04-15	8313	2021-04-15 8313	
2021-04-16	11314	2021-04-16 11314	
2021-04-17	8810	2021-04-17 8810	
2021-04-18	8798	2021-04-18 8798	
2021-04-19	11358	2021-04-19 11358	
2021-04-20	15456	2021-04-20 15456	
2021-04-21	5149	2021-04-21 5149	
2021-04-22	4754	2021-04-22 4754	
2021-04-23	11104	2021-04-23 11104	
2021-04-24	2128	2021-04-24 2128	
2021-04-25	876	2021-04-25 876	
2021-04-26	5739	2021-04-26 5739	
2021-04-27	4274	2021-04-27 4274	
2021-04-28	10243	2021-04-28 10243	
2021-04-29	4455		
2021-04-30	4637		
2021-05-01	8591		
2021-05-02	1265		

Microsoft Visual Studio 디버그 콘솔		data03 - Windows 메모장	
** Selection Sort 결과 출력 **		파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)	
날짜	걸음수		
2021-11-04	19259	2021-04-07 10174	
2021-11-05	17319	2021-04-08 4829	
2021-04-20	15456	2021-04-09 8262	
2021-10-18	13184	2021-04-10 8864	
2021-11-11	12536	2021-04-11 8467	
2021-04-19	11358	2021-04-12 5120	
2021-04-16	11314	2021-04-13 10557	
2021-04-23	11104	2021-04-14 10594	
2021-11-13	10621	2021-04-15 8313	
2021-04-23	10594	2021-04-16 11314	
2021-04-23	10557	2021-04-17 8810	
2021-10-17	10497	2021-04-18 8798	
2021-09-03	10361	2021-04-19 11358	
2021-04-28	10243	2021-04-20 15456	
2021-11-04	10174	2021-04-21 5149	
2021-09-12	9828	2021-04-22 4754	
2021-11-08	9823	2021-04-23 11104	
2021-11-03	9645	2021-04-24 2128	
2021-05-04	9640	2021-04-25 876	
2021-07-01	9215	2021-04-26 5739	
2021-11-12	8985	2021-04-27 4274	
2021-08-27	8884	2021-04-28 10243	
2021-10-18	8864		
2021-10-08	8855		
2021-11-08	8810		
2021-10-17	8798		

3.5 느낀점

앞에 진행한 2개 과제와는 다른 부분의 과제이다. 파일을 읽어서 값을 정렬하는 것은 버블 정렬만 해봤지만 이번엔 선택 정렬을 통해 한다는 것이 새로웠다. 아직 선택 정렬과 삽입 정렬이 비슷해서 헷갈리지만 계속 공부를 해야겠다 생각을 했다. 추후 나오는 퀵정렬이나 다른 정렬로도 이 데이터를 가지고 정렬을 해봐야겠다는 생각을 했다.