



순천향대학교
SOON CHUN HYANG
UNIVERSITY

자료구조 실습 HW 10

자료구조2 실습

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 11월 09일

| 목 차 |

1. Kruskal 최소 비용 신장 트리

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

2. Prim 최소 비용 신장 트리

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

1.1 문제 분석

■ Kruskal의 MST 알고리즘

- 405 페이지에 프로그램 11.8을 참고하여 Kruskal의 최소 비용 신장 트리 프로그램을 작성하고 아래와 같이 가장 최소 비용으로 도달할 때의 비용을 결과로 출력하시오.
 - data.txt에서 간선 및 가중치를 가져와 사용
 - 이미 입력되어있는 간선의 경우 중복되는 간선임을 출력하고 제외

```
C:\WINDOWS\system32\cmd.exe
>> 데이터 입력
간선 0-3 추가 완료
간선 0-5 추가 완료
간선 1-4 추가 완료
간선 1-5 추가 완료
간선 2-5 추가 완료
간선 2-7 추가 완료
간선 3-0 이미 추가된 간선입니다. -- 제외
간선 3-4 추가 완료
간선 3-7 추가 완료
간선 5-1 이미 추가된 간선입니다. -- 제외
간선 5-7 추가 완료

>> 과정
간선 2-7 : 2
간선 1-5 : 4
간선 1-4 : 5
간선 2-5 : 6
간선 5-7 : 7 - 사이클 생성으로 제외
간선 3-4 : 8
간선 3-7 : 9 - 사이클 생성으로 제외
간선 0-5 : 11
간선 0-3 : 13 - 사이클 생성으로 제외

< 필요한 최소 비용 36 >
계속하려면 아무 키나 누르십시오 . . .
```

파일(F)	편집(E)	서식(O)	보기(V)
0 3 13			
0 5 11			
1 4 5			
1 5 4			
2 5 6			
2 7 2			
3 0 19			
3 4 8			
3 7 9			
5 1 6			
5 7 7			

이번 1번 과제는 Kruskal의 최소 비용 신장 트리를 이용해 데이터를 받아 해결하는 과제이다. 제일 중요한 부분은 사이클이 아니면서 최소를 찾아야 하는 알고리즘을 이용해 프로그램을 제작해야 한다. 또한 조건에 간선이 겹칠 경우 제외하기 때문에 별도로 구분하는 부분이 필요하다. 총 값을 더한 후 과정을 표시해 주는 부분도 필요하다.

우선 간선 구조체를 만들어 정수형 변수 시작, 도착 정점과 가중치 변수를 만든다

그래프 구조체에서는 정점의 개수와 간선 배열을 만들어 준다.

초기화 함수에서는 부모 노드를 초기화 하는 함수를 만들어 정점의 개수만큼 부모 노드를 초기화 시켜준다

집합 반환 함수는 curr 번째를 기준으로 초기화 상태 일 경우 반환하는 반환 함수를 만들어 준다.

두개의 원소의 집합을 합치는 함수가 필요하다. 2개의 루트를 찾은 후(루트는 위에서 사용한 반환함수 이용) 두 노드가 다를 경우 두 노드를 합친다.

그래프 초기화 함수를 만들어 준다. 그래프 초기화 함수는 정점의 크기를 0으로 설정하고 설정한 값만큼 반복하면서 배열의 구조체의 시작 정점, 도착 정점, 가중치를 모두 초기화 한다

간선 삽입 연산은 받은 값들을 배열에 순서에 따른 간선 구조체에 값을 넣어 준 후 그래프의 정점을 하나 증가시키게 한다.

퀵정렬에 사용할 compare 함수를 만들어 x, y 구조체를 만들고 가중치의 차를 리턴 한다

제일 중요한 Kruskal 알고리즘 함수는 필요한 변수들을 선언하고 구조체를 선언한다. 집합을 초기화 하고 퀵정렬을 통해 정렬을 한 후 간선의 수가 정점의 개수보다 크거나 같을 때까지 반복을 진행한다. 반복문 안에서는 간선을 가리키게 한 후 정점의 집합을 찾은 후 속한 집합이 다를 경우 두개의 집합을 합치는 방식으로 반복을 한다. 아닐 경우는 사이클이라는 뜻이므로 그냥 종료해주고 간선의 수를 한 개 증가시키는 방식으로 진행한다.

메인 함수에서는 기본적인 파일 포인터, 그래프 포인터와 필요한 변수들을 선언한 후 파일을 읽어 온 후 임시로 만든 간선 배열에 값들을 하나씩 저장하면서 간선 삽입을 진행한다. 만약 간선이 겹칠 경우는 배열 안에 값이 있기 때문에 제외하고 아닐 경우는 배열에 값들을 추가시킨 후 간선 삽입을 한다.

1.2 소스 코드

```
1  // =====
2  // 제작기간 : 21년 11월 3일 ~ 21년 11월 8일
3  // 제작자 : 20204005 김필중
4  // 프로그램명 :Kruskal의 최소 비용 신장 트리 알고리즘
5  // =====
6
7  // 필요한 헤더 파일을 선언한다
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 // 오류 방지 구문
12 #pragma warning(disable : 4996)
13
14 // 필요한 정의를 해준다
15 #define TRUE 1
16 #define FALSE 0
17
18 #define MAX_VERTICES 100 // 그래프 총 크기
19 #define INF 1000 // INF 정의
20
21 // 부모 노드 생성
22 int parent[MAX_VERTICES];
23
24 // 간선을 나타내는 구조체
25 typedef struct Edge
26 {
27     int start, end, weight; // 시작 정점, 도착 정점, 가중치 선언
28 }Edge;
29
30 // 그래프 구조체
31 typedef struct GraphType
32 {
33     int n; // 간선의 개수
34     struct Edge edges[2 * MAX_VERTICES]; // 총 크기의 2배 만큼 간선 구조체 배열을 생성한다
35 } GraphType;
36
37 // 초기화
38 void set_init(int n)
39 {
40     for (int i = 0; i < n; i++) // 받은 n의 크기만큼 반복한다
41         parent[i] = -1; // i번째 부모 노드를 -1로 초기화 한다
42 }
43
44 // curr가 속하는 집합을 반환한다.
45 int set_find(int curr)
46 {
47     if (parent[curr] == -1) // 만약 curr 번째 부모노드가 초기화 상태라면
48         return curr; // 반환한다
49     while (parent[curr] != -1) // 만약 curr 번째 부모노드가 초기화 상태일때까지 반복을 한다
50         curr = parent[curr]; // curr 번째의 부모 노드의 값을 curr로 한다
51     return curr; // curr를 반환한다
52 }
53
54 // 두 개의 집합이 속한 집합을 반환한다
```

```

54 // 두개의 원소가 속한 집합을 합친다.
55 void set_union(int a, int b)
56 {
57     int root1 = set_find(a); // 노드 a의 루트를 찾는다.
58     int root2 = set_find(b); // 노드 b의 루트를 찾는다.
59     if (root1 != root2) // 두 노드가 다를경우
60         parent[root1] = root2; // 두 노드를 합한다
61 }
62
63 // 그래프 초기화
64 void graph_init(GraphType* g)
65 {
66     g->n = 0; // 정점의 개수를 0개로 설정한다
67     for (int i = 0; i < 2 * MAX_VERTICES; i++) // 2 * MAX_VERTICES 만큼 반복한다
68     {
69         // i번째 시작 정점 ~ 끝 정점, 가중치를 모두 초기화 한다
70         g->edges[i].start = 0;
71         g->edges[i].end = 0;
72         g->edges[i].weight = INF;
73     }
74 }
75
76 // 간선 삽입 연산
77 void insert_edge(GraphType* g, int start, int end, int w)
78 {
79     g->edges[g->n].start = start; // 받은 값을 시작 정점에 넣는다
80     g->edges[g->n].end = end; // 받은 값을 도착 정점에 넣는다
81     g->edges[g->n].weight = w; // 받은 값을 가중치에 넣는다
82     g->n++; // 그래프 정점을 하나 증가시킨다.
83 }
84
85 // qsort()에 사용되는 함수
86 int compare(const void* a, const void* b)
87 {
88     // x와 y 구조체를 생성한다
89     struct Edge* x = (struct Edge*)a;
90     struct Edge* y = (struct Edge*)b;
91     // x의 가중치에서 y의 가중치를 뺀 값을 리턴한다
92     return (x->weight - y->weight);
93 }
94

```

```

// kruskal의 최소 비용 신장 트리 프로그램
void kruskal(GraphType *g)
{
    int edge_accepted = 0; // 현재까지 선택된 간선의 수
    int uset, vset; // 정점 u와 정점 v의 집합 번호
    struct Edge e; // 간선 구조체 e 생성
    int result = 0; // 총 값을 0으로 선언한다

    set_init(g->n); // 집합 초기화
    qsort(g->edges, g->n, sizeof(struct Edge), compare); // 퀵정렬을 호출한다

    int i = 0;
    while (edge_accepted < (g->n - 1)) // 현재까지 선택된 간선의 수가 총 정점의 개수보다 크거나 같을때까지 반복한다
    {
        e = g->edges[i]; // i번째 간선을 e를 가리키게한다
        uset = set_find(e.start); // 정점 u의 집합 번호
        vset = set_find(e.end); // 정점 v의 집합 번호
        if (uset != vset) // 서로 속한 집합이 다르면
        {
            printf("간선 %d - %d : %d\n", e.start, e.end, e.weight);
            set_union(uset, vset); // 두개의 집합을 합친다.
            result += e.weight; // 총 값에 더한다
        }
        else // 아닐경우 사이클이므로 프린트
        {
            printf("간선 %d - %d : %d -- 사이클 생성으로 제외\n", e.start, e.end, e.weight);
        }
        i++; // i값 증가
        edge_accepted++; // 간선의 수 하나 증가
    }

    printf("\n< 필요한 최소 비용 : %d >\n", result); // 최소 비용 출력
}

```

```

129 int main(void)
130 {
131     FILE *fp; // 파일 포인터 선언
132     GraphType *g; // 그래프 포인터 선언
133     Edge check[100]; // 겹치는 간선이 있는지 확인하기 위한 임시 구조체 선언
134     int temp1, temp2, temp3; // start, end, weight 값을 저장하기 위한 변수 선언
135     int ch; // 체크 확인 하기 위한 변수
136     int i = 0;
137
138     fp = fopen("data01.txt", "r"); // 파일을 오픈한다
139
140     if (fp == NULL) // 오픈에 실패할 경우 종료한다
141     {
142         printf("파일 오픈 실패\n");
143         return 0;
144     }
145
146     g = (GraphType *)malloc(sizeof(GraphType)); // 그래프를 동적할당을 해준다
147     graph_init(g); // 그래프를 초기화 한다
148
149     printf(">> 데이터 입력\n");
150     while (!feof(fp)) // 파일 끝까지 반복을 한다
151     {
152         ch = 0; // 0이면 새로운 간선 1이면 이미 있는 간선
153         fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // start, end, weight 값을 읽어온다
154         for (int k = 0; k < 101; k++)
155         {
156             // 만약 start, end 값이 한번이라도 나왔다면 ch를 1로 만든다
157             if (((temp1 == check[k].start) && (temp2 == check[k].end)) || ((temp2 == check[k].start) && (temp1 == check[k].end)))
158             {
159                 ch = 1;
160             }
161         }
162         if (ch == 1) // ch가 1이면 이미 있는 간선이므로 넘긴다
163         {
164             printf("간선 %d - %d 은 이미 추가된 간선입니다 --- 제외\n", temp1, temp2);
165         }
166         else // 아닐경우
167         {
168             // 값들을 구조체에 저장한다
169             check[i].start = temp1;
170             check[i].end = temp2;
171             check[i].weight = temp3;
172             // 간선 삽입 연산을 진행한다
173             insert_edge(g, temp1, temp2, temp3);
174             printf("간선 %d - %d 추가 완료\n", temp1, temp2);
175             i++;
176         }
177     }
178 }

```



```

179
180     printf("\n\n>> 과정\n");
181     kruskal(g); // Kruskal 알고리즘을 호출한
182     free(g); // 동적할당을 해제한다
183     fclose(fp); // 파일을 닫는다
184     return 0;
185 }

```


1.3 소스 코드 분석

```
1 // =====
2 // 제작기간 : 21년 11월 3일 ~ 21년 11월 8일
3 // 제작자 : 20204005 김필중
4 // 프로그램명 :Kruskal의 최소 비용 신장 트리 알고리즘
5 // =====
6
7 // 필요한 헤더 파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 // 오류 방지 구문
12 #pragma warning(disable : 4996)
13
14 // 필요한 정의를 해준다
15 #define TRUE 1
16 #define FALSE 0
17
18 #define MAX_VERTICES 100 // 그래프 총 크기
19 #define INF 1000 // INF 정의
20
```

1. 필요한 헤더 파일을 선언한다

2. 오류 방지 구문을 선언한다

3. 필요한 정의를 해준다. 그래프의 총 크기와 INF 를 정의한다

```
// 부모 노드 생성
int parent[MAX_VERTICES];
```

4. 부모 노드를 생성한다. 부모 노드는 Kruskal 알고리즘에서 사용한다

```
// 간선을 나타내는 구조체
typedef struct Edge
{
    int start, end, weight; // 시작 정점, 도착 정점, 가중치 선언
}Edge;
```

5. 간선 구조체를 선언한다. 구조체에는 시작 정점, 도착 정점, 가중치 선언 변수를 선언한다

```

// 그래프 구조체
typedef struct GraphType
{
    int n; // 간선의 개수
    struct Edge edges[2 * MAX_VERTICES]; // 총 크기의 2배 만큼 간선 구조체 배열을 생성한다
} GraphType;

```

6. 그래프 구조체를 선언한다.

7. 구조체에는 간선의 개수, 위에서 선언한 간선 구조체의 배열을 선언한다. 배열의 크기는 정의한 크기의 2배 크기로 잡아준다

```

// 초기화
void set_init(int n)
{
    for (int i = 0; i < n; i++) // 받은 n의 크기만큼 반복한다
        parent[i] = -1; // i번째 부모 노드를 -1로 초기화 한다
}

```

8. 초기화 함수를 만들어 준다.

9. 받은 정점의 개수만큼 반복하면서 부모 노드를 -1로 모두 초기화 시킨다

```

// curr가 속하는 집합을 반환한다.
int set_find(int curr)
{
    if (parent[curr] == -1) // 만약 curr 번째 부모노드가 초기화 상태라면
        return curr; // 반환한다
    while (parent[curr] != -1) // 만약 curr 번째 부모노드가 초기화 상태일때까지 반복을 한다
        curr = parent[curr]; // curr 번째의 부모 노드의 값을 curr로 한다
    return curr; // curr를 반환한다
}

```

10. 집합 반환 함수를 생성한다.

11. 만약 curr번째 부모 노드가 초기화 상태 즉 -1이면 그대로 반환한다

12. 아닐 경우는 계속해서 초기화 상태일때까지 값을 반복해서 찾는다

13. 그 후 curr를 반환한다

```

// 두개의 원소가 속한 집합을 합친다.
void set_union(int a, int b)
{
    int root1 = set_find(a); // 노드 a의 루트를 찾는다.
    int root2 = set_find(b); // 노드 b의 루트를 찾는다.
    if (root1 != root2) // 두 노드가 다를경우
        parent[root1] = root2; // 두 노드를 합한다
}

```

14. 두개의 원소의 집합 합치는 함수를 만든다
15. 노드 a, b의 루트를 위에서 만든 함수를 호출해서 찾는다
16. 만약 2개의 노드가 다를 경우 두개의 노드를 연결해준다.

```

// 그래프 초기화
void graph_init(GraphType* g)
{
    g->n = 0; // 정점의 개수를 0개로 설정한다
    for (int i = 0; i < 2 * MAX_VERTICES; i++) // 2 * MAX_VERTICES 만큼 반복한다
    {
        // i번째 시작 정점 ~ 끝 정점, 가중치를 모두 초기화 한다
        g->edges[i].start = 0;
        g->edges[i].end = 0;
        g->edges[i].weight = INF;
    }
}

```

17. 그래프 초기화 함수를 만들어 준다.
18. 정점의 개수를 0개로 만든 후 배열의 크기만큼 반복한다
19. 시작 정점, 마지막 정점, 가중치를 모두 초기화 시켜준다

```

// 간선 삽입 연산
void insert_edge(GraphType* g, int start, int end, int w)
{
    g->edges[g->n].start = start; // 받은 값을 시작 정점에 넣는다
    g->edges[g->n].end = end; // 받은 값을 도착 정점에 넣는다
    g->edges[g->n].weight = w; // 받은 값을 가중치에 넣는다
    g->n++; // 그래프 정점을 하나 증가시킨다.
}

```

20. 간선 삽입 연산 함수를 만든다

21. 받은 값을 시작 정점, 끝 정점, 가중치에 삽입 한 후, 그래프 정점을 하나 증가시킨다.

```
// qsort()에 사용되는 함수
int compare(const void* a, const void* b)
{
    // x와 y 구조체를 생성한다
    struct Edge* x = (struct Edge*)a;
    struct Edge* y = (struct Edge*)b;
    // x의 가중치에서 y의 가중치를 뺀 값을 리턴한다
    return (x->weight - y->weight);
}
```

22. 퀵정렬에 사용할 함수를 만든다

23. x,y 간선 구조체를 생성한다.

24. x의 가중치에서 y의 가중치를 뺀 값을 리턴을 한다

```
// kruskal의 최소 비용 신장 트리 프로그램
void kruskal(GraphType *g)
{
    int edge_accepted = 0; // 현재까지 선택된 간선의 수
    int uiset, vset; // 정점 u와 정점 v의 집합 번호
    struct Edge e; // 간선 구조체 e 생성
    int result = 0; // 총 값을 0으로 선언한다

    set_init(g->n); // 집합 초기화
    qsort(g->edges, g->n, sizeof(struct Edge), compare); // 퀵정렬을 호출한다
}
```

25. Kruskal의 최소 비용 신장 트리 함수를 만든다.

26. 간선의 수, 정점 집합 번호, 간선 구조체 등 변수를 선언한다

27. 집합을 초기화 한 후 퀵정렬을 호출한다.

```

while (edge_accepted < (g->n - 1)) // 현재까지 선택된 간선의 수가 총 정점의 개수보다 크거나 같을때까지 반복한다
{
    e = g->edges[i]; // i번째 간선을 e를 가리키게한다
    uset = set_find(e.start); // 정점 u의 집합 번호
    vset = set_find(e.end); // 정점 v의 집합 번호
    if (uset != vset) // 서로 속한 집합이 다르면
    {
        printf("간선 %d - %d : %d\n", e.start, e.end, e.weight);
        set_union(uset, vset); // 두개의 집합을 합친다.
        result += e.weight; // 총 값에 더한다
    }
    else // 아닐경우 사이클이므로 프린트
    {
        printf("간선 %d - %d : %d -- 사이클 생성으로 제외\n", e.start, e.end, e.weight);
    }
    i++; // i값 증가
    edge_accepted++; // 간선의 수 하나 증가
}

printf("\n< 필요한 최소 비용 : %d >\n", result); // 최소 비용 출력

```

28. 현재까지 선택된 간선의 수가 총 정점의 개수 - 1보다 크거나 같을 때까지 반복을 진행한다

29. e는 i번째 간선을 가리키게 한다

30. uset과 vset 변수는 find 함수를 통해 값을 찾는다

31. 만약 서로 속한 집합이 다를 경우는 간선을 합친다.

32. 총 값에 가중치를 더한다

33. 아닐 경우는 사이클이므로 사이클임을 알리는 프린트를 한다.

34. 그 후 i값을 증가시키고 간선의 수를 하나를 추가한다.

```

FILE *fp; // 파일 포인터 선언
GraphType *g; // 그래프 포인터 선언
Edge check[100]; // 겹치는 간선이 있는지 확인하기 위한 임시 구조체 선언
int temp1, temp2, temp3; // start, end, weight 값을 저장하기 위한 변수 선언
int ch; // 체크 확인 하기 위한 변수
int i = 0;

fp = fopen("data01.txt", "r"); // 파일을 오픈한다

if (fp == NULL) // 오픈에 실패할 경우 종료한다
{
    printf("파일 오픈 실패\n");
    return 0;
}

g = (GraphType *)malloc(sizeof(GraphType)); // 그래프를 동적할당을 해준다
graph_init(g); // 그래프를 초기화 한다

```

35. 메인 함수에서는 우선 파일 포인터, 그래프 포인터를 선언한다
36. 겹치는 간선이 있는지 확인하기 위한 임시 구조체를 선언하고, 임시 저장 변수 3개를 만든다. 겹치는지 체크하기 위한 ch도 선언한다
37. 파일을 열고 파일 오픈 실패시 종료한다
38. 그래프를 동적할당하고 그래프를 초기화 한다.

```
printf(">> 데이터 입력\n");
while (!feof(fp)) // 파일 끝까지 반복을 한다
{
    ch = 0; // 0이면 새로운 간선 1이면 이미 있는 간선
    fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // start, end, weight 값을 읽어온다
    for (int k = 0; k < 101; k++)
    {
        // 만약 start, end 값이 한번이라도 나왔다면 ch를 1로 만든다
        if (((temp1 == check[k].start) && (temp2 == check[k].end)) || ((temp2 == check[k].start) && (temp1 == check[k].end)))
        {
            ch = 1;
        }
    }
    if (ch == 1) // ch가 1이면 이미 있는 간선이므로 넘긴다
    {
        printf("간선 %d - %d 은 이미 추가된 간선입니다 --- 제외\n", temp1, temp2);
    }
    else // 아닐경우
    {
        // 값들을 구조체에 저장한다
        check[i].start = temp1;
        check[i].end = temp2;
        check[i].weight = temp3;
        // 간선 삽입 연산을 진행한다
        insert_edge(g, temp1, temp2, temp3);
        printf("간선 %d - %d 추가 완료\n", temp1, temp2);
        i++;
    }
}
```

39. 파일을 끝까지 반복하고 새로운 간선 추가면 0, 아니면 1을 의미하게 우선 ch를 0으로 만든다.
40. 만약 한번이라도 나왔다면 ch를 1로 만든다. 그 후 이미 있는 간선이므로 넘긴다.
41. 아닐 경우는 간선을 구조체에 저장한 후 삽입 연산을 진행한다

```
printf("ㄱㄱ>> 과정ㄱ");  
kruskal(g); // Kruskal 알고리즘을 호출한다  
free(g); // 동적할당을 해제한다  
fclose(fp); // 파일을 닫는다  
return 0;
```

42. Kruskal 알고리즘을 호출하고 동적할당을 해제하고 파일을 닫은 후 종료한다

1.4 실행 결과

```
Microsoft Visual Studio 디버그 콘솔
>> 데이터 입력
간선 0 - 3 추가 완료
간선 0 - 5 추가 완료
간선 1 - 4 추가 완료
간선 1 - 5 추가 완료
간선 2 - 5 추가 완료
간선 2 - 7 추가 완료
간선 3 - 0 은 이미 추가된 간선입니다 --- 제외
간선 3 - 4 추가 완료
간선 3 - 7 추가 완료
간선 5 - 1 은 이미 추가된 간선입니다 --- 제외
간선 5 - 7 추가 완료

>> 과정
간선 2 - 7 : 2
간선 1 - 5 : 4
간선 1 - 4 : 5
간선 2 - 5 : 6
간선 5 - 7 : 7 -- 사이클 생성으로 제외
간선 3 - 4 : 8
간선 3 - 7 : 9 -- 사이클 생성으로 제외
간선 0 - 5 : 11
간선 0 - 3 : 13 -- 사이클 생성으로 제외

< 필요한 최소 비용 : 36 >

C:\Users\korca\Desktop\20204005_김필중_자료구조10
16 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

```
data01 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V)
0 3 13
0 5 11
1 4 5
1 5 4
2 5 6
2 7 2
3 0 19
3 4 8
3 7 9
5 1 6
5 7 7
```

1.5 느낀 점

처음 보는 알고리즘에 매우 복잡한 알고리즘이었다. 앞에 했던 그래프와는 다르
고 이건 가중치라는 새로운 요소가 등장함으로써 다양한 표현이 가능해지다는 사
실을 알게 되었다. 특히나 가중치를 활용하면 다양한 프로그램을 제작할 수도 있
고 Kruskal 알고리즘의 원리를 이번 과제를 하면서 계속 수정하고 공부를 할 수
있었다. 그래서 완벽히는 아니지만 반 이상은 이해하고 동작 원리를 알게 되었다.
추후 다른 방법으로도 만들 수 있지 않을까 라는 생각도 해보게 되는 과제였다.

2.1 문제 분석

■ Prim의 MST 알고리즘

- 412 페이지에 프로그램 11.9를 참고하여 Prim의 최소 비용 신장 트리 프로그램을 작성하여 테스트 하시오.
 - data.txt에서 그래프의 정보를 가져오게 수정
(x y w -> x,y 정점 / w 가중치)
 - 동적 할당 이용

```
C:\WINDOWS\system32\cmd.exe
- Prim의 최소 비용 신장 트리 프로그램 -

>> 과정
1 >> 0 : 0
2 >> 0 5 : 10
3 >> 0 5 4 : 37
4 >> 0 5 4 3 : 59
5 >> 0 5 4 3 2 : 71
6 >> 0 5 4 3 2 1 : 87
7 >> 0 5 4 3 2 1 6 : 102

< 필요한 최소 비용 102 >
계속하려면 아무 키나 누르십시오 . . .
```

```
data - 메모장
파일(F) 편집(E) 서식(O) 보기(V)
0 1 29
0 5 10
1 2 16
1 6 15
2 3 12
3 4 22
3 6 18
4 5 27
4 6 25
```

2번 과제는 Prim의 최소 비용 신장 트리 알고리즘을 구현하는 과제이다. 가장 작은 가중치부터 찾아 정점을 찾아 이동하는 알고리즘이다.

먼저 필요한 정의를 해준 후 그래프의 구조체를 생성한다 그래프의 구조체는 정점의 개수와 2차원 배열을 생성한다, 선택, 거리 저장 정적 배열을 선언한다

그래프 초기화 함수를 만들어 행과 열을 반복해 INF로 모두 초기화를 한다

정점 삽입 연산 함수를 만들어 조건에 부합하면 정점을 하나 추가한다.

간선 삽입 연산 함수를 만들어 무방향 그래프로 받은 행과 열의 위치에 가중치를 저장한다.

최소 가중치를 갖는 정점을 반환하는 함수를 만든다 우선 정점의 개수만큼 반복을 하면서 만약 1번째가 선택되어있지 않으면 v 를 i 로 바꾼 후 반복문을 빠져나온다. 이후 정점 반복을 한 후 조건에 부합하면 반환하는 형식으로 진행한다

Prim 알고리즘을 만든다. 처음 값을 받은 후 정점의 개수만큼 우선 가중치 배열을 초기화를 한다. 받은 s 위치의 가중치를 0으로 바꾼 후 정점의 개수만큼 반복을 진행한다. 최소 가중치 값을 찾기 위해 호출하고 받은 값의 선택 배열을 활성화시킨다. 그 후 정점의 개수만큼 반복하면서 앞에서 받은 값의 행과 열의 값이 inf 가 아닐 경우 다시 한번 조건문을 통해 가중치를 변경하는 작업을 진행한다

메인 함수에서는 필요한 변수들을 선언하고 먼저 최대 행 혹은 열을 찾아 그만큼 반복을 해 정점 생성 함수를 호출한다. 추후 파일의 값을 읽어오고 간선 삽입 함수를 호출하고 Prim알고리즘을 0 정점부터 시작하게 한다

2.2 소스 코드

```
1 // =====
2 // 제작기간 : 21년 11월 3일 ~ 21년 11월 8일
3 // 제작자 : 20204005 김필중
4 // 프로그램명 : Prim의 최소 비용 신장 트리 알고리즘
5 // =====
6
7 // 필요한 헤더 파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 // 오류 방지 구문
12 #pragma warning(disable : 4996)
13
14 // 필요한 정의를 해준다
15 #define TRUE 1
16 #define FALSE 0
17 #define MAX_VERTICES 100
18 #define INF 1000L
19
20 // 그래프 구조체
21 typedef struct GraphType
22 {
23     int n; // 정점의 개수
24     int weight[MAX_VERTICES][MAX_VERTICES]; // 가중치 2차원 배열을 선언한다
25 } GraphType;
26
27
28 int selected[MAX_VERTICES]; // 선택을 했다는 배열을 선언한다
29 int distance[MAX_VERTICES]; // 거리를 저장하는 배열을 선언한다
30
31 // 그래프 초기화
32 void graph_init(GraphType* g)
33 {
34     g->n = 0; // 정점의 개수를 0으로 정의한다
35     int r, c;
36     for (r = 0; r < MAX_VERTICES; r++) // 행의 MAX_VERTICES 만큼 반복한다
37         for (c = 0; c < MAX_VERTICES; c++) // 열의 MAX_VERTICES 만큼 반복한다
38             g->weight[r][c] = INF; // INF로 모두 초기화 한다
39 }
40
41 // 정점 삽입 연산
42 void insert_vertex(GraphType* g, int v)
43 {
44     if (((g->n) + 1) > MAX_VERTICES) // 만약 현재 하나 추가한 정점의 개수가 최대 수를 넘으면 종료한다
45     {
46         fprintf(stderr, "그래프: 정점의 개수 초과");
47         return;
48     }
49     g->n++; // 아닐 경우 정점을 하나 추가한다
50 }
51
```

```

52 // 간선 삽입 연산
53 void insert_edge(GraphType* g, int start, int end, int weight)
54 {
55     if (start >= g->n || end >= g->n) // 만약 행 혹은 열이 최대를 넘기면 종료한다
56     {
57         fprintf(stderr, "그래프: 정점 번호 오류");
58         return;
59     }
60     // 아닐경우 무방향 그래프로 받은 값에대한 2차원 배열 위치에 가중치를 저장한다
61     g->weight[start][end] = weight;
62     g->weight[end][start] = weight;
63 }
64
65 // 최소 dist[v] 값을 갖는 정점을 반환
66 int get_min_vertex(int n)
67 {
68     int v, i;
69     for (i = 0; i < n; i++) // n보다 작을때 까지 반복을 한다
70     {
71         if (!selected[i]) // 만약 i가 선택되어있지 않다면
72         {
73             v = i; // v를 i로 바꾸고
74             break; // 반복문 빠져나오기
75         }
76     }
77     for (i = 0; i < n; i++) // n보다 작을때 까지 반복을 한다
78     {
79         if (!selected[i] && (distance[i] < distance[v])) // i가 선택되지 않았으면서 distance[i] 값이 distance[v] 값보다 작으면
80             v = i; // v를 i로 바꾸고
81     }
82     return (v); // 반환한다
83 }

```

```

80
81 // Prim 알고리즘
82 void prim(GraphType* g, int s)
83 {
84     int i, u, v; // 필요한 변수들을 선언
85     int print_list[100]; // 출력할 배열 생성
86     int total = 0; // 가중치의 총합 값을 저장할 변수 설정
87     for (u = 0; u < g->n; u++) // 정점의 개수만큼 반복한다
88     {
89         distance[u] = INF; // distance를 INF로 초기화한다
90         print_list[u] = 0; // 출력 배열을 초기화 한다
91     }
92
93     distance[s] = 0; // 받은 s위치의 distance를 0으로 바꾼다
94     for (i = 0; i < g->n; i++) // 정점의 개수 만큼 반복한다
95     {
96         u = get_min_vertex(g->n); // 최소 distance 값을 찾기 위해 정점의 개수를 보내고 호출한다
97         selected[u] = TRUE; // 받은 u값의 선택 배열에 true로 바꾼다
98         if (distance[u] == INF) // 만약 u번째 가중치 배열이 inf라면 리턴한다
99             return;
100         print_list[i] = u; // u를 출력 배열에 저장하고
101         printf("2d >> ", i);
102         for (int j = 0; j < i + 1; j++) // 반복하면서 프린트 한다
103         {
104             printf("%d ", print_list[j]);
105         }
106         total += distance[u]; // 총합 가중치에 u번째 가중치를 더한다
107         printf(": %d\n", total);
108         for (v = 0; v < g->n; v++) // v를 정점의 개수만큼 반복한다
109         {
110             if (g->weight[u][v] != INF) // 만약 2차원 배열의 u행 v열이 inf가 아니라면
111             {
112                 if (!selected[v] && g->weight[u][v] < distance[v]) // v번째가 선택되지 않고 2차원 배열의 u행 v열의 가중치가 distance[v]보다 작다면
113                     distance[v] = g->weight[u][v]; // v번째 가중치를 u행 v열의 가중치로 바꿔준다
114             }
115         }
116     }
117     printf("\n < 필요한 최소 비용 : %d >\n", total);
118 }

```

```

// 메인함수
int main(void)
{
    FILE *fp; // 파일 포인터를 선언한다
    GraphType *g; // 그래프 포인터를 선언한다
    int temp1, temp2, temp3; // 임시 저장 정수 변수를 생성한다
    int cnt = 0;

    // 파일을 오픈한다
    fp = fopen("data02.txt", "r");

    // 파일 오픈에 실패할 경우 종료한다
    if (fp == NULL)
    {
        printf("파일 오픈 실패\n");
        return 0;
    }

    while (!feof(fp)) // 파일 끝까지 반복한다
    {
        fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // 3개의 값을 읽어 온 후

        // 최대 값을 찾는다
        if (temp1 > cnt)
            cnt = temp1;
        if (temp2 > cnt)
            cnt = temp2;
    }

    rewind(fp); // 파일 포인터를 앞으로 돌린 후
    g = (GraphType *)malloc(sizeof(GraphType)); // 동적할당을 한다
    graph_init(g); // 그래프를 초기화 한다

    for (int i = 0; i < cnt+1; i++) // 최대값 + 1 만큼 동적할당을 한다
        insert_vertex(g, i);

    // 파일의 끝까지 반복을 한다
    while (!feof(fp))
    {
        fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // 파일의 3개의 값을 읽어 온 후
        insert_edge(g, temp1, temp2, temp3); // 간선 삽입 함수로 보낸다
    }

    printf("- Prim의 최소 비용 신장 트리 프로그램 -\n");
    printf(">> 과정\n");
    prim(g, 0); // Prim 알고리즘을 호출한다 0번 부터 시작

    free(g); // 동적할당을 해제하고
    fclose(fp); // 파일을 닫는다
    return 0;
}

```

2.3 소스 코드 분석

```
1 // =====
2 // 제작기간 : 21년 11월 3일 ~ 21년 11월 8일
3 // 제작자 : 20204005 김필중
4 // 프로그램명 :Kruskal의 최소 비용 신장 트리 알고리즘
5 // =====
6
7 // 필요한 헤더 파일을 선언한다
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 // 오류 방지 구문
12 #pragma warning(disable : 4996)
13
14 // 필요한 정의를 해준다
15 #define TRUE 1
16 #define FALSE 0
17 #define MAX_VERTICES 100
18 #define INF 1000L
19
```

1. 헤더 파일을 선언하고 오류 방지 구문을 선언한다
2. 필요한 정의를 해준다.

```
~
20 // 그래프 구조체
21 typedef struct GraphType
22 {
23     int n; // 정점의 개수
24     int weight[MAX_VERTICES][MAX_VERTICES]; // 가중치 2차원 배열을 선언한다
25 } GraphType;
26
27
28 int selected[MAX_VERTICES]; // 선택을 했다는 배열을 선언한다
29 int distance[MAX_VERTICES]; // 거리를 저장하는 배열을 선언한다
30
```

3. 그래프 구조체를 생성한다
4. 구조체에는 정점의 개수와 가중치 2차원 배열을 선언한다
5. 선택을 했다는 정적 배열을 선언한다
6. 가중치를 저장하는 정적 배열을 선언한다

```

0
1 // 정점 삽입 연산
2 void insert_vertex(GraphType* g, int v)
3 {
4     if (((g->n) + 1) > MAX_VERTICES) // 만약 현재 하나 추가한 정점의 개수가 최대 수를 넘으면 종료한다
5     {
6         fprintf(stderr, "그래프: 정점의 개수 초과");
7         return;
8     }
9     g->n++; // 아닐 경우 정점을 하나 추가한다
10 }
1

```

7. 정점 삽입 연산 함수를 만든다.

8. 현재 하나를 추가한 정점의 개수가 최대 수를 넘으면 종료한다

9. 아닐 경우 정점을 하나 추가한다

```

52 // 간선 삽입 연산
53 void insert_edge(GraphType* g, int start, int end, int weight)
54 {
55     if (start >= g->n || end >= g->n) // 만약 행 혹은 열이 최대를 넘기면 종료한다
56     {
57         fprintf(stderr, "그래프: 정점 번호 오류");
58         return;
59     }
60     // 아닐 경우 무방향 그래프로 받은 값에 대한 2차원 배열 위치에 가중치를 저장한다
61     g->weight[start][end] = weight;
62     g->weight[end][start] = weight;
63 }
64

```

10. 간선 삽입 연산 함수를 만든다.

11. 행 혹은 열이 최대를 넘기면 종료한다

12. 아닐 경우는 무방향 그래프 이므로 받은 값에 대한 2차원 배열 위치에 가중치를 저장한다

```

65 // 최소 dist[v] 값을 갖는 정점을 반환
66 int get_min_vertex(int n)
67 {
68     int v, i;
69     for (i = 0; i < n; i++) // n보다 작을때 까지 반복을 한다
70     {
71         if (!selected[i]) // 만약 i가 선택되어있지 않다면
72         {
73             v = i; // v를 i로 바꾸고
74             break; // 반복문 빠져나오기
75         }
76     }
77     for (i = 0; i < n; i++) // n보다 작을때 까지 반복을 한다
78     {
79         if (!selected[i] && (distance[i] < distance[v])) // i가 선택되지 않았으면서 distance[i] 값이 distance[v] 값보다 작으면
80             v = i; // v를 i로 바꾸고
81     }
82     return (v); // 반환한다
83 }
84

```

13. 최소 가중치를 값을 갖는 정점 반환 함수를 만든다

14. 변수 v , i 를 선언하고 정점보다 작을 때까지 반복을 한다
15. 만약 i 번째 선택 배열이 선택되어있지 않다면 v 를 i 로 바꾸고 반복문을 빠져 나온다.
16. 정점 보다 작을때까지 반복하면서 i 위치가 선택되지 않았으면서 i 번째 가중치 값이 v 번째 가중치의 값보다 작을 경우는 v 를 i 로 바꾸는 반복을 계속 한다
17. v 를 반환한다.

```
// Prim 알고리즘
void prim(GraphType* g, int s)
{
    int i, u, v; // 필요한 변수들을 선언
    int print_list[100]; // 출력할 배열 생성
    int total = 0; // 가중치의 총합 값을 저장할 변수 설정
    for (u = 0; u < g->n; u++) // 정점의 개수만큼 반복한다
    {
        distance[u] = INF; // distance를 INF로 초기화한다
        print_list[u] = 0; // 출력 배열을 초기화 한다
    }

    distance[s] = 0; // 받은 s위치의 distance를 0으로 바꾼다
}
```

18. Prim 알고리즘 함수를 만든다
19. 필요한 변수들을 선언하고 출력할 배열을 선언한다
20. 총합 값을 저장할 변수를 선언한다
21. 정점의 개수만큼 반복하면서 가중치 배열을 모두 INF 로 초기화하고 출력배열도 초기화 한다
22. 받은 s 위치의 가중치를 0으로 만든다


```

for (i = 0; i < g->n; i++) // 정점의 개수 만큼 반복한다
{
    u = get_min_vertex(g->n); // 최소 distance 값을 찾기 위해 정점의 개수를 보내고 호출한다
    selected[u] = TRUE; // 받은 u값의 선택 배열에 true로 바꾼다
    if (distance[u] == INF) // 만약 u번째 가중치 배열이 inf라면 리턴한다
        return;
    print_list[i] = u; // u를 출력 배열에 저장하고
    printf("%d >> ", i);
    for (int j = 0; j < i + 1; j++) // 반복하면서 프린트 한다
    {
        printf("%d ", print_list[j]);
    }
    total += distance[u]; // 총합 가중치에 u번째 가중치를 더한다
    printf("\n %d\n", total);
    for (v = 0; v < g->n; v++) // v를 정점의 개수만큼 반복한다
        if (g->weight[u][v] != INF) // 만약 2차원 배열의 u행 v열이 inf가 아니라면
            if (!selected[v] && g->weight[u][v] < distance[v]) // v번째가 선택되지 않고 2차원 배열의 u행 v열의 가중치가 distance[v]보다 작다면
            {
                distance[v] = g->weight[u][v]; // v번째 가중치를 u행 v열의 가중치로 바꿔준다
            }
}
printf("\n < 필요한 최소 비용 : %d >\n", total);

```

23. 정점의 개수만큼 반복한다

24. 최소 가중치를 찾기 위해 정점의 개수를 보내고 호출해 u값을 받는다

25. 받은 u값을 선택하고 만약 u번째 배열의 가중치의 값이 inf일경우 반환한다

26. u를 출력 배열에 저장한 후 1 만큼 반복하면서 출력을 한다

27. total 변수에 u번째 가중치의 값을 더해 준 후 출력한다

28. v를 정점의 개수만큼 반복하면서 만약 2차원 배열 u행 v열의 값이 INF가 아닐 경우, 다음으로 v번째 선택이 활성화되지 않았고 u행v열의 가중치의 값이 v번째 배열의 가중치의 값보다 작을 경우 v번째 가중치의 값을 현재 행과 열의 가중치의 값으로 바꿔준다

```

FILE *fp; // 파일 포인터를 선언한다
GraphType *g; // 그래프 포인터를 선언한다
int temp1, temp2, temp3; // 임시 저장 정수 변수를 생성한다
int cnt = 0;

// 파일을 오픈한다
fp = fopen("data02.txt", "r");

// 파일 오픈에 실패할 경우 종료한다
if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return 0;
}

```

29. 파일 포인터를 선언하고 그래프 포인터를 선언한다
30. 임시 저장 변수 3개를 선언하고 최대값을 저장할 CNT를 선언한다
31. 파일을 오픈하고 실패할 경우 종료한다

```

while (!feof(fp)) // 파일 끝까지 반복한다
{
    fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // 3개의 값을 읽어 온 후

    // 최대 값을 찾는다
    if (temp1 > cnt)
        cnt = temp1;
    if (temp2 > cnt)
        cnt = temp2;
}

rewind(fp); // 파일 포인터를 앞으로 돌린 후
g = (GraphType *)malloc(sizeof(GraphType)); // 동적할당을 한다
graph_init(g); // 그래프를 초기화 한다

```

32. 파일 끝까지 반복하면서 3개의 값을 읽어오고 최대값을 찾는다.
33. 그 후 파일 포인터를 맨 앞으로 옮긴다
34. 그래프를 동적할당을 하고 그래프를 초기화 시킨다

```

for (int i = 0; i < cnt+1; i++) // 최대값 + 1 만큼 동적할당을 한다
    insert_vertex(g, i);

// 파일의 끝까지 반복을 한다
while (!feof(fp))
{
    fscanf(fp, "%d %d %d", &temp1, &temp2, &temp3); // 파일의 3개의 값을 읽어 온 후
    insert_edge(g, temp1, temp2, temp3); // 간선 삽입 함수로 보낸다
}

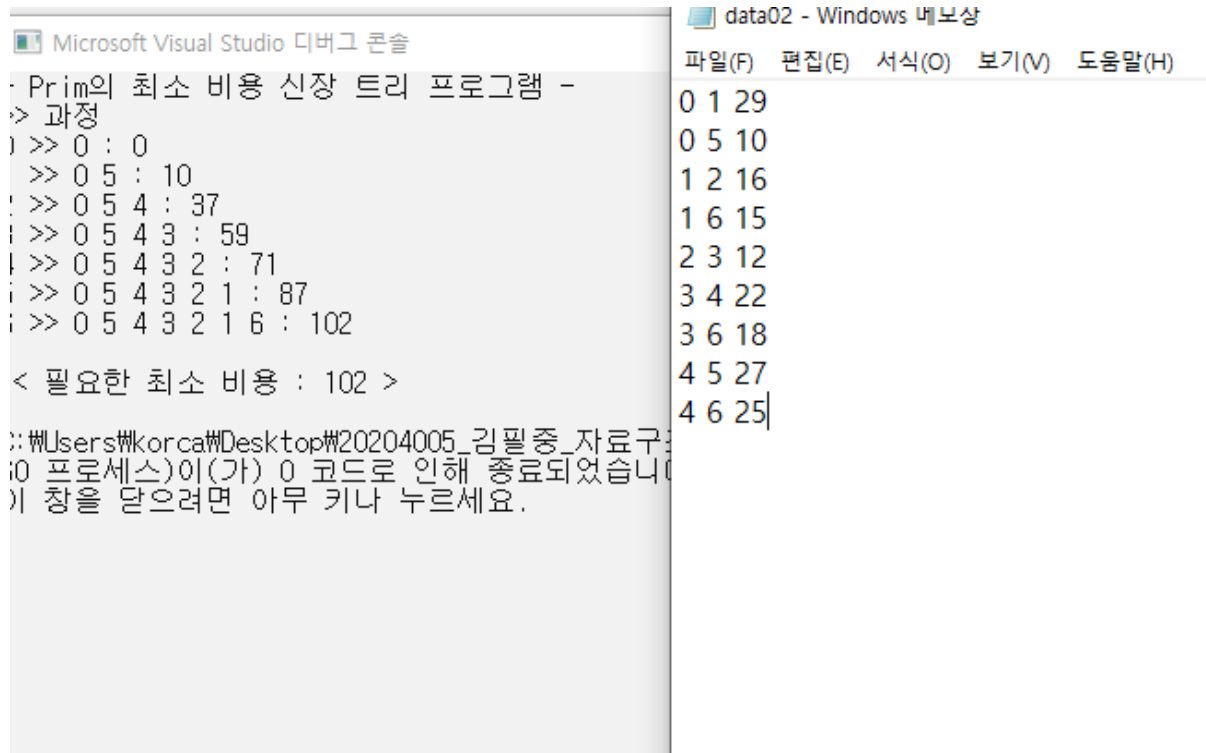
printf("- Prim의 최소 비용 신장 트리 프로그램 -\n");
printf(">> 과정\n");
prim(g, 0); // Prim 알고리즘을 호출한다 0번 부터 시작

free(g); // 동적할당을 해제하고
fclose(fp); // 파일을 닫는다
return 0;

```

35. 최대값 + 1만큼 정적 삽입 연산을 반복해 정점을 삽입해준다
36. 파일 끝까지 반복 하면서 3개의 값을 받아 3개의 값을 간선 삽입 함수를 호출한다
37. 그 후 0번부터 시작하는 Prim 알고리즘을 호출한다
38. 동적할당을 해제하고 파일을 닫은 후 종료한다

2.4 실행 화면



```
Microsoft Visual Studio 디버깅 콘솔
Prim의 최소 비용 신장 트리 프로그램 -
> 과정
>> 0 : 0
>> 0 5 : 10
>> 0 5 4 : 37
>> 0 5 4 3 : 59
>> 0 5 4 3 2 : 71
>> 0 5 4 3 2 1 : 87
>> 0 5 4 3 2 1 6 : 102

< 필요한 최소 비용 : 102 >

::\\Users\\korca\\Desktop\\20204005_김필중_자료구
:0 프로세스)이(가) 0 코드로 인해 종료되었습니다
이 창을 닫으려면 아무 키나 누르세요.
```

```
data02 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
0 1 29
0 5 10
1 2 16
1 6 15
2 3 12
3 4 22
3 6 18
4 5 27
4 6 25
```

2.5 느낀점

이번 과제는 앞에서 한 알고리즘보다 단순한듯 했으나 제일 큰 실수를 한 것이 그래프를 반으로 접었을 때 동일하기 때문에 무방향 그래프로 만들었어야 한다. 하지만 무방향 그래프가 아닌 방향 그래프로 만들었다가 계속 되는 출력값 오류로 인해 많은 시간을 소비했다는 것이 아쉬웠다. 그 후 교수님께서 말씀하신 의미를 이해하고 코드를 수정해 해결했다. Prim 알고리즘은 가장 중요한 부분은 역시 작은 값을 찾으면서 과정을 진행하는 것이 중요하다. 그래서 정점을 출력하는 과정을 보면서 그래프를 직접 그릴 수도 있다는 생각을 했다. 아직까지 그래프2 단원을 들어오면서 익숙하지 않지만 계속 반복하고 다양한 방법으로 다른 프로그램을 만들어 보면 조금 더 실력이 늘 수 있다고 생각한다.