

# 자료구조2 실습

## 10주차 과제



제출일	21.11.09	전 공	컴퓨터소프트웨어공학과
과 목	자료구조 2 실습	학 번	20184612
담당교수	홍 민 교수님	이 름	김동민

# | 목 차 |

## 1. 그래프: Kruskal의 MST 알고리즘

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

## 2. 그래프: Prim의 MST 알고리즘

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

## 3. 그래프를 하며 느낀 점

## 1. 그래프

# Kruskal의 MST 알고리즘

---

- 405 페이지의 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 입력되어 있는 간선의 경우 중복되는 간선임을 출력하고 제외

### 1.1 문제 분석

조건 1 중복되는 간선이 있다면 출력하고 제외함

조건 2 Kruskal 알고리즘을 사용하는 과정을 출력함

조건 3 data.txt파일에는 간선과 가중치가 있고 직접 생성하여 입력받음

- 이 문제는 Kruskal알고리즘을 이용하여 최소 비용 신장 트리를 구하는 문제이다. Kruskal알고리즘은 선택할 때마다 그 순간 가장 좋다고 생각되는 것을 선택함으로써 최종적인 해답에 도달하는 방법인 탐욕적인 방법을 이용하여 프로그래밍을 한다. Kruskal알고리즘은 그래프의 간선들을 가중치의 오름차순으로 정렬한후 정렬된 간선들의 리스트에서 사이클을 형성하지 않는 간선을 찾아서 현재의 최소 비용 신장 트리의 집합에 추가한다.

Kruskal 알고리즘에서는 사이클이 형성되지 않아야 하기 때문에 간선의 양 끝 정점이 같은 집합에 속해있는지 검사하는데, 이를 위해 union\_find알고리즘을 사용한다. union\_find를 구현하기 위해서 1차원 배열을 이용하고 그 배열은 부모노드의 인덱스를 저장한다. 배열의 값이 -1이면 부모노드가 없다.

간선을 삽입할 때는 같은 간선이 들어온다면 이를 출력한 후 삽입하지 않고 kruskal알고리즘을 수행할 때는 중간에 사이클이 생성된다면 사이클을 생성하는 정점을 출력한 후 사이클을 생성한다는 메시지를 출력한다.

## 1.2 소스 코드

```
/*
 * 학번 : 20184812
 * 학과 : 컴퓨터소프트웨어공학과
 * 이름 : 김종민
 * 파일명 : Kruskal의 MST 알고리즘 프로그램
 */
#define _CRT_SECURE_NO_WARNINGS

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

#define TRUE 1
#define FALSE 0
#define INF 1000 //간선이 없을 때 값
int *parent; //부모노드

void set_init(int n); //부모노드 초기화 함수
int set_find(int curr); //curr가 속하는 집합 반환 함수
void set_union(int a, int b); //두개의 집소가 속한 집합을 합치는 함수

struct Edge { //간선을 나타내는 구조체
    int start, end, weight; //시작, 끝, 가중치
};

typedef struct GraphType {
    int n; //간선의 개수
    struct Edge *edges; //간선 배열
} GraphType;

void Graph_init(GraphType *g, int num); //그래프를 초기화하는 함수
void insert_edge(GraphType *g, struct Edge e); //간선을 삽입하는 함수
int compare(const void *a, const void *b); //qsort에 사용되는 함수
int kruskal(GraphType *g); //kruskal 최소 비용 신장 트리 함수
void Delete(GraphType *g); //그래프 삭제 함수

int main() {
    FILE *fp;
    struct Edge e; //에이저를 입력받을 변수
    GraphType *g; //그래프 변수
    int cost; //kruskal 최소 비용
    int count = 0; //파일 데이터의 개수

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open\n");
        return 0;
    }
    while (!feof(fp)) { //파일의 데이터 개수를 구함
        fscanf(fp, "%d%d%d", &e.start, &e.end, &e.weight);
        count++; //데이터 개수 증가
    }
    g = (GraphType *) malloc(sizeof(GraphType)); //그래프 생성
    Graph_init(g, count); //그래프 초기화
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &e.start, &e.end, &e.weight);
        insert_edge(g, e); //그래프에 간선 삽입
    }
    cost = kruskal(g); //kruskal 최소 비용 구함
    printf("최소 비용 : %d\n", cost); //최소 비용 출력

    Delete(g); //그래프 삭제
    fclose(fp); //파일 포인터 닫음
    return 0;
}

void set_init(int n) { //부모노드 초기화
    int i;
    parent = (int *) malloc(sizeof(int) * n); //부모 노드를 n개로 출력할당
    for (i = 0; i < n; i++)
        parent[i] = -1; //부모노드를 -1로 초기화
}

int set_find(int curr) { //curr가 속하는 집합을 반환
    if (parent[curr] == -1) //단일 부모노드가 -1이면
        return curr; //curr리턴
    while (parent[curr] != -1) curr = parent[curr]; //부모노드가 -1이 될 때까지 반복
    return curr; //부모 노드의 값을 curr에 담음
}

void set_union(int a, int b) { //두개의 집소가 속한 집합을 합침
    int root1 = set_find(a); //노드 a의 부모를 찾음
    int root2 = set_find(b); //노드 b의 부모를 찾음
    if (root1 != root2) //두 부모를 합침
        parent[root1] = root2;
}
```

```

void Graph_init(GraphType* g, int num) { //그래프 초기화
    int i;
    g->n = 0;
    g->edges = (struct Edge*)malloc(sizeof(struct Edge)*(num + 2)); //num개의 2바이트 간선 배열 할당
    for (i = 0; i < num + 2; i++) { //초기화
        g->edges[i].start = 0;
        g->edges[i].end = 0;
        g->edges[i].weight = INF; //가중치 값을 INF로 초기화
    }
}

void insert_edge(GraphType* g, struct Edge e) { //간선 삽입
    int i;
    for (i = 0; i < g->n; i++) { //배열에 간선이 이미 존재하는지 확인
        if ((g->edges[i].end == e.start && g->edges[i].start == e.end) || (g->edges[i].start == e.start && g->edges[i].end == e.end)) {
            printf("간선 %d - %d은 이미 추가된 간선입니다. -- 제거 %d", e.start, e.end);
            return; //반환 존재한다면 함수 리턴
        }
    }
    g->edges[g->n].start = e.start; //존재하지 않는다면 간선 삽입
    g->edges[g->n].end = e.end;
    g->edges[g->n].weight = e.weight;
    printf("간선 %d - %d 추가됨\n", e.start, e.end);
    g->n++; //간선의 개수 증가
}

int compare(const void* a, const void* b) { //qsort에 사용되는 함수
    struct Edge* x = (struct Edge*)a;
    struct Edge* y = (struct Edge*)b;
    return (x->weight - y->weight);
}

int kruskal(GraphType* g) { //kruskal 최소 비용 신장 트리 함수
    int edge_accepted = 0; //현재까지 선택된 간선의 수
    int uset, vset; //집합 u와 집합 v의 집합 번호
    struct Edge e; //한쌍의 간선을 저장할 변수
    int sum = 0; //최소 비용을 할당 변수

    set_init(g->n); //집합 초기화

    qsort(g->edges, g->n, sizeof(struct Edge), compare); //트 정렬 함수

    printf("크루스칼 최소 신장 트리 알고리즘\n");
    int i = 0;
    while (edge_accepted < (g->n)) { //선택된 간선의 수가 정렬된 개수보다 작다면 반복
        e = g->edges[i];
        uset = set_find(e.start); //집합 u의 집합 번호
        vset = set_find(e.end); //집합 v의 집합 번호

        if (uset != vset) { //서로 속한 집합이 다르다면
            printf("간선 %d - %d : %d\n", e.start, e.end, e.weight);
            edge_accepted++;
            set_union(uset, vset); //두 집합을 합친다.
            sum += e.weight; //가중치를 더함
        }
        else { //반대 간선이 사이클을 형성한다면
            edge_accepted++;
            printf("간선 %d - %d : %d - 사이클 생성으로 제거\n", e.start, e.end, e.weight);
        }
        i++;
    }
    return sum; //최소 비용 반환
}

void Delete(GraphType* g) { //그래프 삭제 함수
    free(perent); //부모 노드 삭제
    free(g->edges); //간선 변수 삭제
    free(g); //그래프 변수 삭제
}

```

## 1.3 소스 코드 분석

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

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

#define TRUE 1
#define FALSE 0
#define INF 1000 //간선이 없을 때 길이
int *parent; //부모노드

struct Edge { //간선을 나타내는 구조체
    int start, end, weight; //시작, 끝, 가중치
};

typedef struct GraphType {
    int n; //간선의 개수
    struct Edge *edges; //간선 배열
}GraphType;
```

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

2. 부모 노드의 인덱스를 저장하는 parent변수를 선언한다. parent변수는 전역변수로 선언하고 포인터로 선언되어 데이터의 개수에 맞게 동적할당하여 사용하고, 정점이 같은 집합에 속해있는지 확인하는 union\_find연산에 사용한다.

3. 간선을 나타내는 구조체를 선언한다. 구조체 안의 변수는 시작과 끝 정점의 값을 나타내는 start, end와 간선의 가중치를 나타내는 weight변수가 있다.

4. 그래프를 나타내는 GraphType 구조체를 선언한다. 구조체 안에는 간선의 개수를 나타내는 n과 struct Edge 자료형으로 간선을 저장하는 변수인 edges가 포인터로 선언되었다. 포인터는 데이터의 개수에 맞게 동적할당하여 사용한다.

```

void set_init(int n);    //부모노드 초기화 함수
int set_find(int curr); //curr가 속하는 집합 반환 함수
void set_union(int a, int b); //두개의 원소가 속한 집합을 합치는 함수

void Graph_init(GraphType*g, int num);    //그래프를 초기화하는 함수
void insert_edge(GraphType*g, struct Edge e); //간선을 삽입하는 함수
int compare(const void*a, const void*b);    //qsort에 사용되는 함수
int kruskal(GraphType*g);    //kruskal 최소 비용 신장 트리 함수
void Delete(GraphType*g);    //그래프 삭제 함수

```

##### 5. 필요한 함수들을 선언한다.

- set\_init함수는 parent변수를 초기화 하는 함수이다.
- set\_find함수는 curr가 속하는 집합을 반환하는 함수이다.
- set\_union함수는 두 개의 원소가 속한 집합을 합치는 함수이다.
- Graph\_init함수는 그래프의 변수들을 동적할당하고, 초기화하는 변수이다.
- insert\_edge함수는 그래프에 간선을 삽입하는 함수이다.
- compare함수는 qsort에 사용되는 함수로 비교함수를 나타낸다.
- kruskal함수는 kruskal알고리즘을 이용하여 최소 비용을 찾는 함수이다.
- Delete함수는 그래프와 parent 변수를 삭제하는 함수이다.

```

int main() {
    FILE *fp;
    struct Edge e;    //데이터를 입력받을 임시변수
    GraphType *g;    //그래프 변수
    int cost;    //kruskal로 구한 최소 비용
    int count = 0;    //파일 데이터의 개수

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open\n");
        return 0;
    }
    while (!feof(fp)) {    //파일의 데이터 개수를 구함
        fscanf(fp, "%d%d%d", &e.start, &e.end, &e.weight);
        count++;    //데이터 개수 증가
    }
    g = (GraphType*)malloc(sizeof(GraphType)); //그래프 생성
    Graph_init(g, count);    //그래프 초기화
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &e.start, &e.end, &e.weight);
        insert_edge(g, e);    //그래프에 간선 삽입
    }
    cost = kruskal(g);    //kruskal로 최소 비용을 구함
    printf("필요한 최소 비용 : %d\n", cost);    //최소 비용 출력

    Delete(g);    //그래프 삭제
    fclose(fp);    //파일포인터 닫음
    return 0;
}

```



6. 필요한 변수들을 선언한다.

- fp는 파일포인터, e는 데이터를 입력받을 임시변수, g는 그래프 변수이다.
- cost는 최소 비용을 저장할 변수, count는 파일의 개수를 셀 변수이다.

7. data.txt파일을 읽기형식으로 open한다. open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 임시 구조체 변수 e에 입력을 받고 데이터의 개수인 count를 증가시킨다.

8. 그래프 변수g를 동적할당하여 생성하고, 그래프를 Graph\_init함수로 초기화하는데, count값을 전달하여 초기화한다.

9. rewind함수로 파일포인터를 다시 처음으로 되돌리고 파일을 다시 입력받는다. 그리고 입력받는 간선의 정보를 insert\_edge함수를 호출하여 간선을 삽입한다.

10. 간선이 모두 삽입되었다면 kruskal함수를 호출하여 최소 비용을 찾는다. kruskal함수는 최소 비용을 함수 내에서 계산하고 그 값을 반환한다. 메인함수에서는 cost에 그 값을 저장하고 출력한다.

11. 모든 출력이 완료 되었다면 동적할당을 모두 해제할 수 있도록 한다. Delete함수를 이용하여 동적할당한 그래프의 정보를 모두 해제하도록 한다. 그리고 fclose로 파일포인터를 닫고 프로그램을 종료한다.

```
void set_init(int n) { //부모노드 초기화
    int i;
    parent = (int*)malloc(sizeof(int)*n); //부모 노드를 n값으로 동적할당
    for (i = 0; i < n; i++)
        parent[i] = -1; //부모노드를 -1로 초기화
}

int set_find(int curr) { //curr가 속하는 집합을 반환
    if (parent[curr] == -1) //만약 부모노드가 -1이라면
        return curr; //curr리턴
    while (parent[curr] != -1) curr = parent[curr]; //부모노드가 -1이 될때까지 반복
    return curr; //부모 노드의 값을 curr에 담음
}

void set_union(int a, int b) { //두개의 원소가 속한 집합을 합침
    int root1 = set_find(a); //노드 a의 루트를 찾음
    int root2 = set_find(b); //노드 b의 루트를 찾음
    if (root1 != root2) //두 루트를 합함
        parent[root1] = root2;
}
```



12. set\_init함수는 parent변수를 초기화 하는 함수이다.

먼저 parent포인터를 입력받은 n값으로 동적할당한 후 만들어진 배열을 -1로 모두 초기화한다. 이 배열은 부모 노드의 인덱스를 저장하고 배열의 값이 -1이면 부모노드가 없다는 의미이다.

13. set\_find함수는 curr가 속하는 집합을 반환하는 함수이다.

만약 parent변수의 curr인덱스 값이 -1이라면 curr을 리턴하고 -1이 아니라면 -1이 나올 때까지 반복하면서 curr을 업데이트하며 부모를 찾아간다.

14. set\_union함수는 두 개의 원소가 속한 집합을 합치는 함수이다.

입력받은 a, b의 루트를 찾고 만약 이 두 루트가 다르다면 두 루트를 합한다.

```
void Graph_init(GraphType*g, int num) { //그래프 초기화
    int i;
    g->n = 0;
    g->edges = (struct Edge*)malloc(sizeof(struct Edge)*(num * 2)); //num값의 2배로 간선 배열 동적할당
    for (i = 0; i < num * 2; i++) { //초기화
        g->edges[i].start = 0;
        g->edges[i].end = 0;
        g->edges[i].weight = INF; //가중치 값을 INF로 초기화
    }
}
```

15. Graph\_init함수는 그래프의 정보를 초기화하는 함수이다.

정점의 개수를 나타내는 n을 0으로 초기화하고 edges변수를 입력받은 num값의 2배로 동적할당한다.

그리고 for문을 이용하여 모든 간선 구조체의 정보를 초기화하는데 start와 end는 0으로 초기화하고 weight가중치는 INF로 초기화한다.

```
void insert_edge(GraphType*g, struct Edge e) { //간선 삽입
    int i;
    for (i = 0; i < g->n; i++) { //배열에 간선이 이미 존재하는지 확인
        if ((g->edges[i].end == e.start && g->edges[i].start == e.end) || (g->edges[i].start == e.start && g->edges[i].end == e.end)) {
            printf("간선 %d - %d은 이미 추가된 간선입니다.-- 제외 \n", e.start, e.end);
            return; //만약 존재한다면 함수 리턴
        }
    }
    g->edges[g->n].start = e.start; //존재하지 않는다면 간선 삽입
    g->edges[g->n].end = e.end;
    g->edges[g->n].weight = e.weight;
    printf("간선 %d - %d 추가완료\n", e.start, e.end);
    g->n++; //간선의 개수 증가
}
```

16. insert\_edge함수는 그래프에 간선을 삽입하는 함수이다.

먼저 삽입 전에 같은 간선이 있는지 확인한다. 정점의 개수만큼 for문으로 반복하며 이미 입력되어있는 간선과 입력받은 간선을 비교한다. 만약 같은 간선이 있다면 함수를 바로 return 한다.

17. 겹치는 간선이 없다면 edges변수에 간선을 삽입한다. g->n은 정점의 개수를 나타내기 때문에 edges의 g->n인덱스에 데이터를 저장한 후 추가된 간선을 출력한다. 그리고 간선의 개수인 g->n을 증가시킨다.

```
int compare(const void*a, const void*b) { //qsort에 사용되는 함수
    struct Edge *x = (struct Edge*)a;
    struct Edge* y = (struct Edge*)b;
    return (x->weight - y->weight);
}
```

18. compare함수는 qsort에 사용되는 함수로, 정렬을 할 때 배열의 자료형이 제각각이기 때문에 비교방식을 설정하기 위해 사용한다.

19. 입력받은 void포인터 변수 a와 b를 struct Edge 포인터로 변환한 뒤 값을 x, y에 삽입한다.

그리고 x->weight에서 y->weight를 뺀 값을 리턴하는데 만약 x->weight가 더 크다면 양수가 반환되고 y->weight가 더 크다면 음수가 반환되고 두가중치가 같다면 0이 반환될 것이다.

```
int kruskal(GraphType*g) { //kruskal 최소 비용 신장 트리 함수
    int edge_accepted = 0; //현재까지 선택된 간선의 수
    int uset, vset; //정점 u와 정점 v의 집합 번호
    struct Edge e; //반환 받은 값을 저장할 변수
    int sum = 0; //최소 비용을 합할 변수

    set_init(g->n); //집합 초기화

    qsort(g->edges, g->n, sizeof(struct Edge), compare); //퀵 정렬 함수

    printf("\n크루스칼 최소 신장 트리 알고리즘\n");
    int i = 0;
    while (edge_accepted < (g->n)) { //선택된 간선의 수가 정점의 개수보다 작다면 반복
        e = g->edges[i];
        uset = set_find(e.start); //정점 u의 집합 번호
        vset = set_find(e.end); //정점 v의 집합 번호

        if (uset != vset) { //서로 속한 집합이 다르다면
            printf("간선 %d - %d : %d\n", e.start, e.end, e.weight);
            edge_accepted++;
            set_union(uset, vset); //두 집합을 합친다.
            sum += e.weight; //가중치를 더함
        }
        else { //만약 간선이 사이클을 형성한다면
            edge_accepted++;
            printf("간선 %d - %d : %d - 사이클 생성으로 제외\n", e.start, e.end, e.weight);
        }
        i++;
    }
    return sum; //최소 비용 반환
}
```

20. kruskal함수는 kruskal알고리즘을 이용하여 최소 비용을 계산하는 함수이다. kruskal함수는 탐욕적인 방법을 이용하여 선택할 때마다 그 순간 가장 좋다고 생각되는 것을 선택한다.

21. 먼저 필요한 변수들을 선언하고 set\_init함수로 집합을 초기화한다.

그리고 퀵정렬 함수인 qsort함수를 사용하여 그래프의 간선을 가중치의 오름차순으로 정렬한다.

qsort함수 : 왼쪽 값들과 오른쪽 값들을 각각 따로 재귀를 통해 분할정복하는 알고리즘이다.

퀵 정렬 함수에는 정렬할 배열 또는 메모리의 주소, 요소 개수, 요소 크기, 비교함수를 넣어준다 (stdlib.h헤더파일에 선언되어 있다).

22. 선택된 간선의 수를 나타내는 edge\_accepted변수가 정점의 개수보다 작다면 계속 반복한다. 오름차순으로 정렬된 간선의 정보를 임시변수 e에 저장한 후 set\_find함수로 정점이 같은 집합에 포함되어있는지 확인한다.

23. 만약 두 집합이 다르다면 간선을 출력한 후 edge\_accepted변수를 증가시키고 set\_union함수를 통해 두 집합을 합친다. 그리고 가중치의 합을 나타내는 sum에 현재 가중치를 합친다.

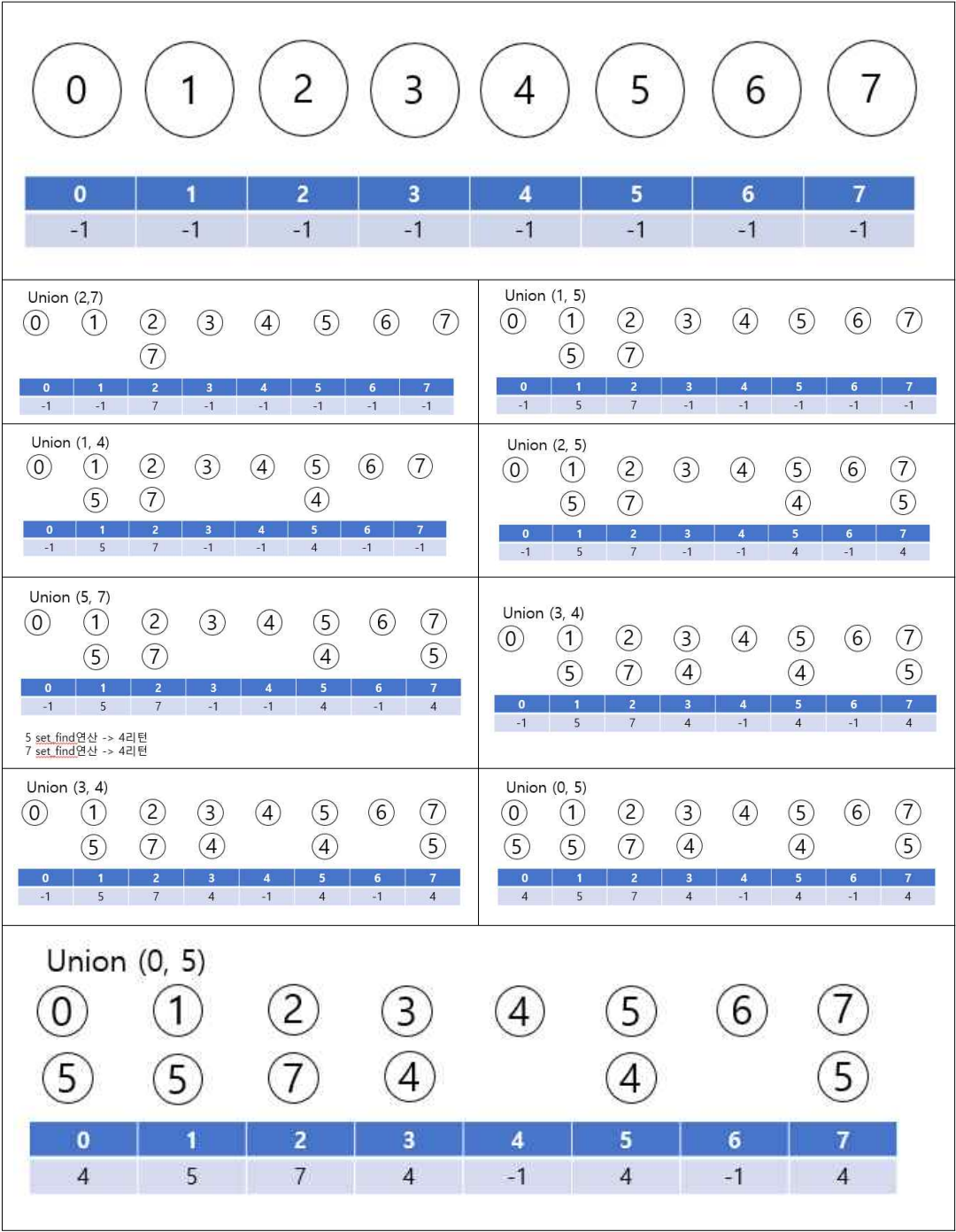
두 집합이 같다면 사이클을 형성한다는 의미이므로 가중치를 더하지 않고 사이클을 형성한다는 메시지를 출력한다.

24. 모든 반복을 마치면 sum에는 kruskal 알고리즘으로 구한 최소비용이 저장되어 있다. kruskal함수는 반환 형식이 int이기 때문에 최소 비용을 리턴해준다.

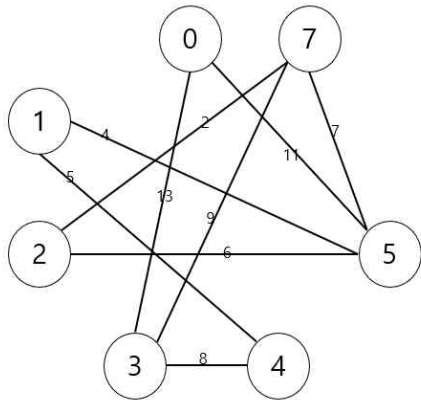
```
void Delete(GraphType*g) { //그래프 삭제 함수
    free(parent); //부모 노드 삭제
    free(g->edges); //간선 변수 삭제
    free(g); //그래프 변수 삭제
}
```

25. Delete함수는 그래프와 parent변수를 모두 메모리 해제하는 함수이다. 동적할당한 parent변수를 메모리 해제하고 간선 변수 edges와 그래프 변수 g를 모두 메모리 해제한다.

set\_union의 과정



kruskal알고리즘의 진행과정



오름차순으로 정렬한 edges 배열

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

<b>2, 7 간선 연결</b>	<b>1, 5 간선 연결</b>	<b>1, 4 간선 연결</b>
<b>2, 5 간선 연결</b>	<b>5 7 간선이 사이클 생성</b>	<b>3 4 간선 연결</b>
<b>3 7 간선이 사이클 생성</b>	<b>0 5 간선 연결</b>	<b>0 3 간선이 사이클 생성</b>



## 1.4 실행 창

- kruskal알고리즘 최소 비용 결과

간선 0 - 3 추가완료	
간선 0 - 5 추가완료	
간선 1 - 4 추가완료	
간선 1 - 5 추가완료	
간선 2 - 5 추가완료	
간선 2 - 7 추가완료	
간선 3 - 0은 이미 추가된 간선입니다.-- 제외	
간선 3 - 4 추가완료	
간선 3 - 7 추가완료	
간선 5 - 1은 이미 추가된 간선입니다.-- 제외	
간선 5 - 7 추가완료	
크루스칼 최소 신장 트리 알고리즘	
간선 2 - 7 : 2	0 3 13
간선 1 - 5 : 4	0 5 11
간선 1 - 4 : 5	1 4 5
간선 2 - 5 : 6	1 5 4
간선 5 - 7 : 7 - 사이클 생성으로 제외	2 5 6
간선 3 - 4 : 8	2 7 2
간선 3 - 7 : 9 - 사이클 생성으로 제외	3 0 19
간선 0 - 5 : 11	3 4 8
간선 0 - 3 : 13 - 사이클 생성으로 제외	3 7 9
필요한 최소 비용 : 36	5 1 6
C:\Users\dmk46\OneDrive\바탕 화면\2학기 수업자료 인해 종료되었습니다.	5 7 7
이 창을 닫으려면 아무 키나 누르세요.	

## 1.5 느낀 점

이번 문제는 Kruskal 알고리즘을 이용하여 최소 비용 신장 트리를 구하는 문제였습니다. 처음 보는 알고리즘들을 보며 처음엔 이해가 잘 되지 않는 부분들이 많았지만 하나하나 알고리즘이 수행되는 과정을 그려보며 레포트를 쓰니 조금이나마 쉽게 이해할 수 있었던 것 같습니다. 특히 parent라는 배열을 사용하여 부모 노드의 인덱스를 저장한다는 것과 union\_find연산에 대해 이해가 잘 안되는 부분이 많았지만 이번 과제를 계기로 이해할 수 있었던 것 같습니다.

Kruskal 알고리즘은 먼저 쿼정렬함수 qsort를 이용하여 간선의 정보를 오름차순 정렬하였습니다. qsort함수 또한 처음 봤기 때문에 직접 구글링을 통해 함수를 이용하는 방법을 찾아보았습니다. 이를 통해 qsort함수에 필요한 매개변수와 compare함수가 필요하다는 것 등을 배울 수 있었고 앞으로 다른 문제에 이 함수를 사용할 일이 생긴다면 사용할 수 있게 되었습니다.

이번 최소 비용 신장 트리 알고리즘에서 처음으로 탐욕적인 방법이라는 것을 배울 수 있었습니다. 아직 많이 익숙하지는 않지만 관련된 알고리즘을 더 열심히 공부하여 Kruskal알고리즘이 필요할 때마다 사용할 수 있도록 노력해야겠다고 다짐했습니다.



## 2. 그래프

### Prim의 MST 알고리즘

---

- 412페이지에 프로그램 11.9를 참고하여 Prim의 최소 비용 신장트리 프로그램을 작성하여 테스트 하시오.
- data.txt에서 그래프의 정보를 가져오게 수정  
(  $x\ y\ w \rightarrow x, y$  정점 /  $w$  가중치 )
- 동적할당 이용

#### 2.1 문제 분석

조건 1 Prim 알고리즘으로 정점을 찾아가는 과정을 출력함

조건 2 그래프에 대해서 data.txt파일들을 직접 생성하여 입력받음

조건 3 그래프 간선 정보를  $n \times n$  행렬에 저장

- 이 문제는 Prim알고리즘을 사용하여 최소비용 신장 트리를 구하는 문제이다. Prim알고리즘은 시작 정점에서부터 출발하여 신장 트리 집합을 단계적으로 확장해나가는 방법이다. 시작 단계에서는 시작 정점만이 신장 트리 집합에 포함된다. Prim의 방법은 앞 단계에서 만들어진 신장 트리 집합에, 인접한 정점들 중에서 최저 간선으로 연결된 정점을 선택하여 트리를 확장한다.

Kruskal알고리즘은 간선을 기반으로 하는 알고리즘인 반면 Prim알고리즘은 정점을 기반으로 하는 알고리즘이다. 또한 Kruskal은 무조건 최저 간선만을 선택했지만 Prim알고리즘은 이전단계에서 만들어진 신장 트리를 확장해나가는 방식이다.

Prim을 구현하기 위해선 정점에서 정점까지의 거리를 담고있는 distance 변수가 필요하다. 처음에는 시작 정점의 노드만 값이 0이고 나머지는 무한대의 값을 가진다. 후에 정점들이 트리 집합에 추가되면서 distance값은 변경된다.

이 문제에서는 형식에 맞게 출력하기 위해 방문하는 과정을 임시 배열에 저장한 후 그 과정을 출력하여 사용자가 볼 수 있도록 프로그래밍 한다. 그리고 추가로 출력과 함께 가중치를 더하여 출력한다.

## 2.2 소스 코드

```
//+
//학번 : 20154612
//학과 : 컴퓨터소프트웨어공학부
//이름 : 김동진
//파일명 : Prim의 MST 프로그램
//+
#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; // 인접 행렬 그래프 표현
}GraphType;

int *selected; //정점이 방문되었는지 알아볼 변수
int *distance; //가장 가까운 정점까지의 거리 변수

void init(GraphType *g, int num); //그래프, selected, distance 변수 초기화 함수

void insert_vertex(GraphType *g, int num); //정점 삽입 함수
void insert_edge(GraphType *g, int start, int end, int weight); //간선 삽입 함수 (무방향 그래프)

int get_min_vertex(int n); //최소 dist[v] 값을 가진 정점 반환 함수
void prim(GraphType *g, int s); //Prim 최소 비용을 신장 트리 함수
void Delete(GraphType *g); //그래프 삭제 함수

int main() {
    FILE *fp;
    GraphType *g;
    int max = INT_MIN; //max에 int의 최소값 삽입
    int start, end, weight; //에이저를 입력받을 변수
    int i;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &start, &end, &weight);
        if (max < start) max = start; //start와 end중에서 가장 큰 정점의 값을 찾을
        if (max < end) max = end;
    }
    g = (GraphType *) malloc(sizeof(GraphType)); //그래프 생성
    max++;
    init(g, max); //max값으로 초기화

    for (i = 0; i < max; i++) { //정점 삽입
        insert_vertex(g, max);
    }
    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &start, &end, &weight);
        insert_edge(g, start, end, weight); //그래프에 간선 삽입
    }
    prim(g, 0); //prim 최소 비용을 신장 트리의 함수 호출 (0부터 시작)

    Delete(g); //그래프 삭제 함수
    fclose(fp);
    return 0;
}

void init(GraphType *g, int num) { //초기화 함수
    int i, j;
    g->n = 0;
    g->weight = (int **) malloc(sizeof(int *) * num); //num만큼 weight를 할당
    selected = (int *) malloc(sizeof(int) * num); //num만큼 selected를 할당
    distance = (int *) malloc(sizeof(int) * num); //num만큼 distance를 할당
    for (i = 0; i < num; i++) {
        g->weight[i] = (int *) malloc(sizeof(int) * num); //그래프 행을 할당
        selected[i] = 0;
        distance[i] = 0;
        for (j = 0; j < num; j++) {
            g->weight[i][j] = INF; //그래프의 값을 INF로 초기화
        }
    }
}
```

```

void insert_vertex(GraphType* g, int num) { //정점을 삽입하는 함수
    if ((g->n) + 1 > num) {
        for(int i=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) {
        for(int i=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.
}

int get_min_vertex(int n) { //최소 dist[v] 값을 갖는 정점을 반환하는 함수
    int v, i;
    for (i = 0; i < n; i++) { //0부터 정점의 개수까지 반복
        if (!selected[i]) { //선택된 정점인지 확인
            v = i; //현재 인덱스를 v에 할당 break;
            break;
        }
    }
    for (i = 0; i < n; i++) { //0부터 정점의 개수까지 반복
        //정점이 방문되지 않았고, 현재 가중치보다 더 작은 가중치가 있다면 v를 현재이름
        if (!selected[i] && (distance[i] < distance[v])) v = i; //가장 작은 간선중에 최소 distance를 반환
    }
    return v; //v리턴
}

void prim(GraphType* g, int s) { //Prim 최소 비용 신장 트리 함수
    int i, u, v;
    int *ary;
    int sum = 0;

    ary = (int*)malloc(sizeof(int)*g->n); //메모리를 할당할 ary변수 할당함
    printf("Prim의 최소 비용 신장 트리 프로그램-MAIN");
    printf(">> 과정\n");
    for (u = 0; u < g->n; u++)
        distance[u] = INF; //초반 distance[]를 INF로 초기화
    distance[s] = 0; //시작 정점의 distance[]에 0 삽입
    for (i = 0; i < g->n; i++) {

        u = get_min_vertex(g->n); //최소 distance를 u에 저장

        selected[u] = TRUE; //현재 정점을 방문했다고 표시
        if (distance[u] == INF) return; //distance[]이 INF이면 return

        ary[i] = u; //ary에 최소 값을 가진 u를 삽입
        sum += distance[u]; //sum에 distance[]를 더해나감
        for (v = 0; v < g->n; v++) {
            if (g->weight[u][v] != INF) { //weight[]이 INF가 아니라면
                //현재 정점이 방문되지 않았고 weight[]이 distance보다 작다면
                if (!selected[v] && g->weight[u][v] < distance[v]) {
                    distance[v] = g->weight[u][v]; //distance[]를 weight로 현재이름
                }
            }
        }
    }
    printf("%d >> ", i);
    for (v = 0; v < i; v++) {
        printf("%d ", ary[v]); //i의 개수만큼 정점 출력
    }
    printf("\n", sum); //결과 출력
    // sum = 0;
    free(ary);
}

void Delete(GraphType* g) { //메모리 삭제 함수
    int i;
    for (i = 0; i < g->n; i++) {
        free(g->weight[i]); //그래프 행 삭제
    }
    free(g->weight); //그래프 행 삭제
    free(g); //그래프 삭제
    free(distance); //distance변수 메모리 해제
    free(selected); //selected변수 메모리 해제
}

```

## 2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    파일 명: Prim의 MST프로그램
*/
#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;    // 인접 행렬 그래프 포인터
}GraphType;

int *selected;      //정점이 방문되었는지 담아놓는 변수
int *distance;      //가중치가 저장되어있는 변수
```

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

2. 그래프 구조체 GraphType을 선언한다. 그래프 구조체에는 정점의 개수 n과 가중치가 저장되어있는 인접행렬 이중포인터 weight변수를 가진다.

3. 정점이 방문되었는지 확인하는 변수인 selected와 가중치를 저장할 distance변수를 선언한다.

```
void init(GraphType*g, int num);          //그래프, selected, distance 변수 초기화 함수

void insert_vertex(GraphType*g, int num); //정점 삽입 함수
void insert_edge(GraphType* g, int start, int end, int weight); //간선 삽입 함수(무방향)

int get_min_vertex(int n);                //최소 dist[v] 값을 갖는 정점 반환함수
void prim(GraphType*g, int s);            //Prim 최소 비용 신장 트리 함수
void Delete(GraphType*g);                 //그래프 삭제 함수
```

4. 필요한 함수들을 선언한다.

- init함수는 그래프와 selected, distance변수를 초기화하는 함수이다.
- insert\_vertex함수는 인접 행렬의 정점을 삽입하는 함수이다.
- insert\_edge함수는 그래프에 간선을 삽입하는 함수이다.
- get\_min\_vertex함수는 가장 작은 distance값을 반환하는 함수이다.
- prim함수는 Prim알고리즘을 사용하여 최소 비용을 찾는 함수이다.
- Delete함수는 동적할당한 부분을 모두 메모리 해제하는 함수이다.

```

int main() {
    FILE *fp;
    GraphType *g;
    int max = INT_MIN; //max에 int의 최소값 삽입
    int start, end, weight; //데이터를 입력받을 임시변수
    int i;

    fp = fopen("data.txt", "r");
    if (!fp) {
        printf("file not open");
        return 0;
    }
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &start, &end, &weight);
        if (max < start)max = start; //start와 end중에서 가장 큰 정점의 값을 찾음
        if (max < end)max = end;
    }
    g = (GraphType*)malloc(sizeof(GraphType)); //그래프 생성
    max++;
    init(g, max); //max값으로 초기화

    for (i = 0; i < max; i++) { //정점 삽입
        insert_vertex(g, max);
    }

    rewind(fp);
    while (!feof(fp)) {
        fscanf(fp, "%d%d%d", &start, &end, &weight);
        insert_edge(g, start, end, weight); //그래프에 간선 삽입
    }
    prim(g, 0); //prim 최소 비용 신장트리 함수 호출 (0부터 시작)

    Delete(g); //그래프 삭제 함수
    fclose(fp);
    return 0;
}

void init(GraphType *g, int size) { //초기화 함수

```

5. 필요한 변수들을 선언한다.

- fp는 파일포인터, g는 그래프변수, max는 가장 큰 정점을 찾기 위한 변수이다.
- start, end, weight는 파일 데이터를 입력받을 임시변수이다.

max는 가장 큰 정점을 찾아야하기 때문에 int의 최소 값을 삽입한다.

6. data.txt파일을 읽기형식으로 open한다. open하고 만약 파일이 존재한다면 파일 끝까지 데이터를 입력받는데, 그 과정에서 start와 end중에서 가장 큰 정점의 값을 찾는다.

7. max값을 구했다면 그래프를 동적할당하여 생성하고 그래프를 초기화하는데, 만약 정점의 최대값이 7이라면 8로 동적할당을 해야 하기 때문에 max값을 1증가시켜 초기화한다.

그리고 max값만큼 for문을 이용해 반복하며 정점을 삽입한다. 여기서 정점 개수는 최대 개수인 max를 넘을 수 없으므로 이를 매개변수로 전달한다.



8. rewind함수를 이용하여 파일포인터를 다시 처음으로 되돌리고, 간선을 삽입한다. start, end, weight변수에 파일 데이터를 입력받고, 간선 삽입함수 insert\_edge함수를 호출하여 매개변수로 이를 전달한다.

9. 간선이 모두 삽입되었다면 prim함수를 호출하여 Prim알고리즘을 수행한다. prim함수는 그래프와 시작 정점을 매개변수로 전달한다.

10. prim함수로 최소 비용을 구했다면 Delete함수를 호출하여 동적할당된 부분을 모두 메모리 해제해준다. 그래프의 정보와 selected, distance 모두 해제한다. 그리고 fclose로 파일포인터를 닫고 프로그램을 종료한다.

```
void init(GraphType*g, int num) { //초기화 함수
    int i, j;
    g->n = 0;
    g->weight = (int**)malloc(sizeof(int)*num); //num값으로 weight동적할당
    selected = (int*)malloc(sizeof(int)*num); //num값으로 selected동적할당
    distance = (int*)malloc(sizeof(int)*num); //num값으로 distance동적할당
    for (i = 0; i < num; i++) {
        g->weight[i] = (int*)malloc(sizeof(int)*num); //그래프 열 동적할당
        selected[i] = 0;
        distance[i] = 0;
        for (j = 0; j < num; j++) {
            g->weight[i][j] = INF; //그래프의 값을 INF로 초기화
        }
    }
}
```

11. init함수는 그래프와 selected, distance변수를 초기화하는 함수이다. 먼저 그래프의 정점의 개수를 나타내는 g->n을 0으로 초기화한다. selected와 distnace변수를 num값으로 동적할당하고 weight이중포인터도 num값으로 동적할당한다.

12. for문을 num까지 반복하며 weight의 열을 동적할당해주고 selected와 distance값을 0으로 초기화한다. 그리고 weight의 값은 INF로 초기화한다.

```
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
}
```

13. insert\_vertex함수는 그래프에 정점을 삽입하는 함수이다.

만약 삽입한 후 정점의 개수인  $g \rightarrow n+1$ 이 메인에서의 정점 개수 num값보다 크면 함수를 종료한다. 여기서 num값은 메인에서 구한 max값을 나타낸다. 삽입이 끝나면 정점의 개수인  $g \rightarrow n$ 을 증가시킨다.

14. insert\_edge함수는 방향그래프로 간선을 삽입하는 함수이다. 매개변수로 그래프 g와 시작 인덱스인 start와 끝 인덱스인 end를 전달받는다.

인덱스를 먼저 정점의 개수  $g \rightarrow n$ 과 비교한다. 만약 정점의 개수보다 인덱스의 값이 작다면 행렬에 값을 삽입한다.

15. 이 함수는 무방향 그래프로 삽입하기 때문에 두 방향모두 삽입해야 한다. 그렇기 때문에 start행 end열의 값과 end행 start열의 값에 삽입을 하는데, 삽입할 때는 가중치 값을 나타내는 weight를 배열에 삽입한다.

```
int get_min_vertex(int n) { //최소 dist[v] 값을 갖는 정점을 반환하는 함수
    int v, i;

    for (i = 0; i < n; i++) { //0부터 정점의 개수까지 반복
        if (!selected[i]) { //만약 방문되지 않았다면
            v = i; //현재 인덱스를 v에 담고 break;
            break;
        }
    }

    for (i = 0; i < n; i++) { //0부터 정점의 개수까지 반복
        //정점이 방문되지 않았고, 현재 가중치보다 더 작은 가중치가 있다면 v를 업데이트
        if (!selected[i] && (distance[i] < distance[v])) v = i; //가능한 간선중에 최소 distance를 찾음
    }

    return v; //v리턴
}
```

16. get\_min\_vertex함수는 방문되지 않은 정점 중에 가장 작은 distance값의 인덱스를 반환하는 함수이다.

먼저 for문으로 정점의 개수만큼 반복하며 정점이 방문되었는지 확인한다.

만약 방문되지 않은 정점이라면 v에 인덱스 값을 담고 break로 반복을 종료한다.

17. 정점이 방문되지 않았고 현재 구한 가중치보다 더 작은 가중치가 있다면 v를 업데이트하며 for문으로 반복한다.

결과적으로 가장 작은 distance의 인덱스 번호를 찾아 반환할 수 있다.



```

void prim(GraphType*g, int s) { //Prim 최소 비용 신장 트리 함수
    int i, u, v;
    int *ary;
    int sum = 0;

    ary = (int*)malloc(sizeof(int)*g->n); //순서를 저장할 ary변수 동적할당
    printf("-Prim의 최소 비용 신장 트리 프로그램 -\n\n");
    printf(">> 과정\n");
    for (u = 0; u < g->n; u++)
        distance[u] = INF; //모든 distance값을 INF로 초기화
    distance[s] = 0; //시작 정점의 distance값에 0 삽입
    for (i = 0; i < g->n; i++) {

        u = get_min_vertex(g->n); //최소 distance를 u에 저장

        selected[u] = TRUE; //현재 정점을 방문되었다고 표시
        if (distance[u] == INF) return; //distance값이 INF이면 return

        ary[i] = u; //ary에 최소 값을 가진 u를 삽입
        sum += distance[u]; //sum에 distance값을 더해나감
        for (v = 0; v < g->n; v++) {
            if (g->weight[u][v] != INF) { //weight값이 INF가 아니라면
                //현재 정점이 방문되지 않았고 weight값이 distance보다 작다면
                if (!selected[v] && g->weight[u][v] < distance[v]) {
                    distance[v] = g->weight[u][v]; //distance값을 weight로 업데이트
                }
            }
        }
        printf("%d >> ", i+1);
        for (v = 0; v <= i; v++) {
            printf("%d ", ary[v]); //i의 개수만큼 정점 출력
        }
        printf(": %d\n", sum); //길이 출력
        // sum = 0;
    }
    free(ary);
}

```

18. prim함수는 Prim의 MST알고리즘을 사용하여 최소비용을 구하는 함수이다. 먼저 정점을 방문할 순서를 담을 ary포인터를 정점의 개수만큼 동적할당하여 생성한다.

그리고 가중치를 저장할 distance변수를 모두 INF로 초기화한 후 시작정점의 distance값에 0을 삽입한다.

19. for문으로 정점의 개수만큼 반복한다. 먼저 get\_min\_vertex함수를 통해 가장 작은 distance값을 반환받는다. 가장 처음에는 작은 값이 distance[s]로, 값이 0이기 때문에 0이 반환될 것이다.

그리고 그 정점을 방문되었다고 표시하기 위해 selected변수에 TRUE를 삽입한다.

20. 정점 방문 순서를 저장할 ary변수에 방문한 정점을 삽입하고, 최소 비용을 나타내는 sum변수에 가중치 값 distance를 더한다.

21. 현재 정점에서 갈 수 있는 정점 중 간선의 가중치가 INF가 아니라면, 그리고 현재 정점이 방문되지 않았고 그 가중치가 distance보다 작다면 distance를 업데이트해준다.

22. 모든 삽입이 끝나면 형식에 맞게 I의 개수만큼 for문을 반복하며 방문한 정점 ary를 순서대로 출력한다.

모든 정점을 출력한 후 최소 비용인 sum을 출력한다.

모든 반복이 끝나면 동적할당한 ary를 메모리 해제해준다.

```
void Delete(GraphType*g) { //메모리 삭제 함수
    int i;
    for (i = 0; i < g->n; i++) {
        free(g->weight[i]); //그래프 열 삭제
    }
    free(g->weight); //그래프 행 삭제
    free(g); //그래프 삭제
    free(distance); //distance변수 메모리 해제
    free(selected); //selected변수 메모리 해제
}
```

23. Delete함수는 그래프와 distance, selected변수를 모두 메모리 해제하는 함수이다.

동적할당한 변수를 모두 메모리 해제하고 인접 행렬 포인터 weight와 그래프 변수 g를 모두 메모리 해제한다.

- 그래프 정보

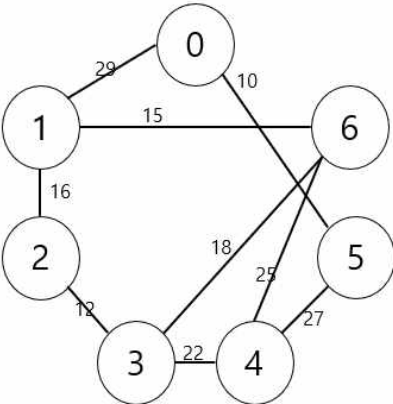
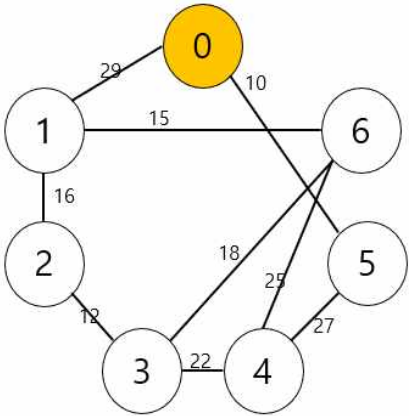
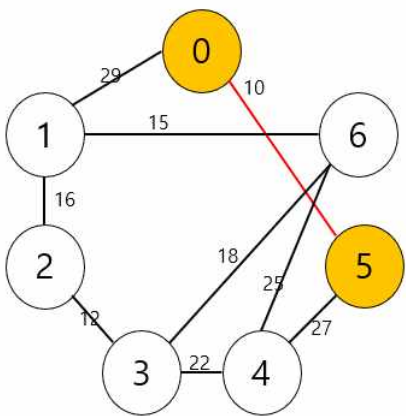
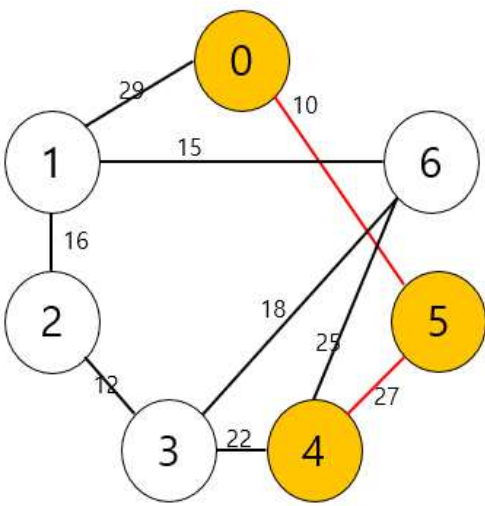
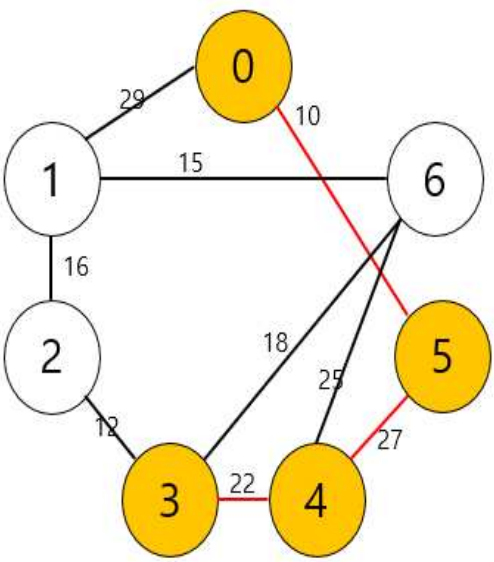
	0	1	2	3	4	5	6
0	INF	29	INF	INF	INF	10	INF
1	29	INF	16	INF	INF	INF	15
2	INF	16	INF	12	INF	INF	INF
3	INF	INF	12	INF	22	INF	18
4	INF	INF	INF	22	INF	27	25
5	10	INF	INF	INF	27	INF	INF
6	INF	15	INF	18	25	INF	INF

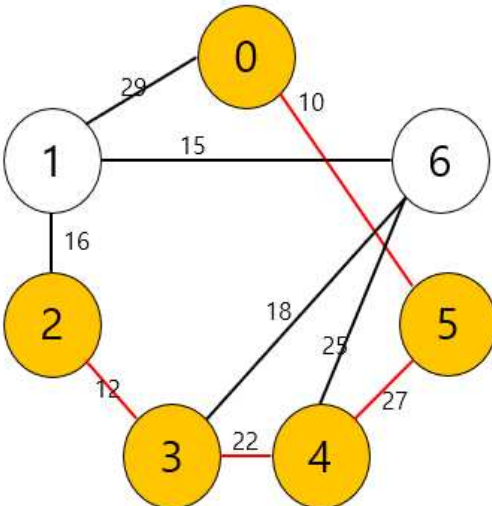
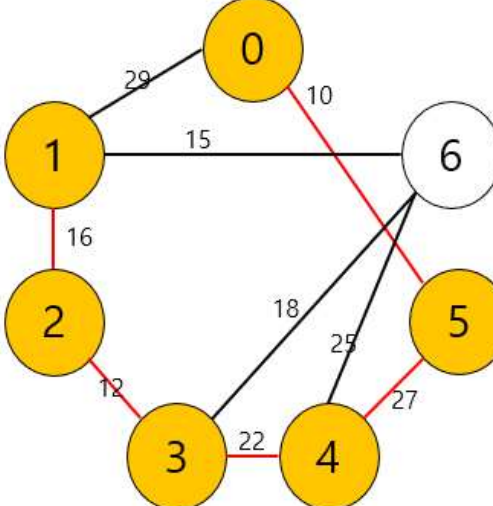
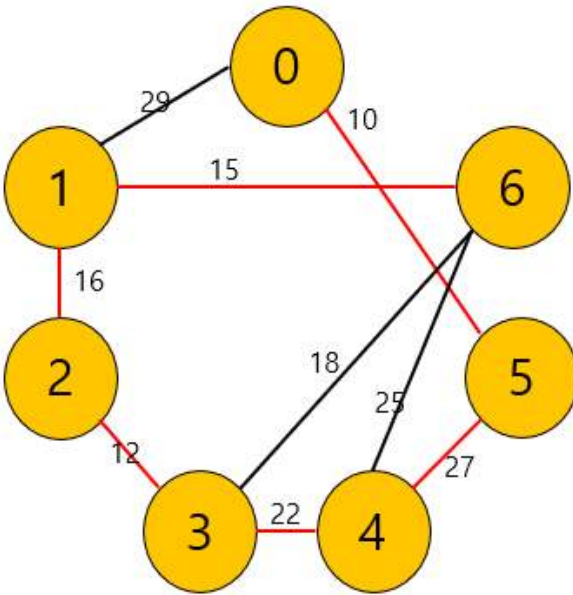
- selected 배열과 distance배열의 변화 과정

<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>INF</td><td>INF</td><td>INF</td><td>INF</td><td>INF</td><td>INF</td></tr></table>	0	1	2	3	4	5	6	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	1	2	3	4	5	6	0	INF	INF	INF	INF	INF	INF	1. 시작 정점이 0부터 이므로 selected값은 TRUE이고 distance값은 0
0	1	2	3	4	5	6																							
TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE																							
0	1	2	3	4	5	6																							
0	INF	INF	INF	INF	INF	INF																							
<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>TRUE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>29</td><td>INF</td><td>INF</td><td>INF</td><td>10</td><td>INF</td></tr></table>	0	1	2	3	4	5	6	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	0	1	2	3	4	5	6	0	29	INF	INF	INF	10	INF	2. 0번 정점에서 갈 수 있는 간선중 가장 작은 가중치를 가진 인덱스->10 5번 인덱스 selected true distance 10 1번 distance 29로 업데이트
0	1	2	3	4	5	6																							
TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE																							
0	1	2	3	4	5	6																							
0	29	INF	INF	INF	10	INF																							
<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>FALSE</td><td>FALSE</td><td>FALSE</td><td>TRUE</td><td>TRUE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>29</td><td>INF</td><td>INF</td><td>27</td><td>10</td><td>INF</td></tr></table>	0	1	2	3	4	5	6	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	0	1	2	3	4	5	6	0	29	INF	INF	27	10	INF	3. 5번 정점에서 갈 수 있는 간선중 선택되지 않았고 가장 작은 가중치를 가진 인덱스-> 27 4번 인덱스 selected ture, distance 27
0	1	2	3	4	5	6																							
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE																							
0	1	2	3	4	5	6																							
0	29	INF	INF	27	10	INF																							

<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>FALSE</td><td>FALSE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>29</td><td>INF</td><td>22</td><td>27</td><td>10</td><td>25</td></tr></table>	0	1	2	3	4	5	6	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	0	1	2	3	4	5	6	0	29	INF	22	27	10	25	4. 4번 정점에서 갈 수 있는 간선 중 가장 작은 가중치를 가진 인덱스 -> 22 3번 인덱스 selected true, distance 22 6번 distance 25로 업데이트
0	1	2	3	4	5	6																							
TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE																							
0	1	2	3	4	5	6																							
0	29	INF	22	27	10	25																							
<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>FALSE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>29</td><td>12</td><td>22</td><td>27</td><td>10</td><td>18</td></tr></table>	0	1	2	3	4	5	6	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	0	1	2	3	4	5	6	0	29	12	22	27	10	18	5. 3번 정점에서 갈 수 있는 간선 중 가장 작은 가중치를 가진 인덱스-> 12 2번인덱스 selected TRUE distance 12 6번 distance 18로 업데이트
0	1	2	3	4	5	6																							
TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE																							
0	1	2	3	4	5	6																							
0	29	12	22	27	10	18																							
<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>FALSE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>16</td><td>12</td><td>22</td><td>27</td><td>10</td><td>18</td></tr></table>	0	1	2	3	4	5	6	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	0	1	2	3	4	5	6	0	16	12	22	27	10	18	6. 2번 정점에서 갈 수 있는 간선 중 가장 작은 가중치를 가진 인덱스 -> 16 1번 인덱스 selected true distance 16
0	1	2	3	4	5	6																							
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE																							
0	1	2	3	4	5	6																							
0	16	12	22	27	10	18																							
<table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td><td>TRUE</td></tr></table> <table><tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><td>0</td><td>16</td><td>12</td><td>22</td><td>27</td><td>10</td><td>15</td></tr></table>	0	1	2	3	4	5	6	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	0	1	2	3	4	5	6	0	16	12	22	27	10	15	7. 1번 정점에서 갈 수 있는 간선 중 가장 작은 가중치를 가진 인덱스->15 6번인덱스 selected true distance 15
0	1	2	3	4	5	6																							
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE																							
0	1	2	3	4	5	6																							
0	16	12	22	27	10	15																							

# Prim알고리즘 정점 탐색 과정

기본 그래프	
	
정점 0부터 시작	정점 0에서 갈 수 있는 정점 중 가장 작은 가중치를 가진 정점을 찾음 -> 10
	
갈 수 있는 가중치 -> 27, 29 가장 작은 가중치 -> 27	갈 수 있는 가중치->25, 22, 29 가장 작은 가중치 -> 22
	

<p>갈 수 있는 가중치 -&gt; 25, 18, 12, 29 가장 작은 가중치 -&gt; 12</p>	<p>갈 수 있는 가중치 -&gt; 16, 29, 18, 25 가장 작은 가중치 -&gt; 16</p>
	
<p>갈 수 있는 가중치 -&gt; 29, 15, 18, 25 가장 작은 가중치 -&gt; 15</p>	
	



## 2.4 실행 창

### - 정점 0부터 탐색

-Prim의 최소 비용 신장 트리 프로그램-		data - Wind
>> 과정		파일(F) 편집(E)
1 >> 0 : 0		0 1 29
2 >> 0 5 : 10		0 5 10
3 >> 0 5 4 : 37		1 2 16
4 >> 0 5 4 3 : 59		1 6 15
5 >> 0 5 4 3 2 : 71		2 3 12
6 >> 0 5 4 3 2 1 : 87		3 4 22
7 >> 0 5 4 3 2 1 6 : 102		3 6 18
<필요한 최소 비용 : 102>		4 5 27
C:\Users\wdmk46\OneDrive\바탕 화면\2학기		4 6 25
로 인해 종료되었습니다.		
이 창을 닫으려면 아무 키나 누르세요.		

### - 정점 6부터 탐색

-Prim의 최소 비용 신장 트리 프로그램-		data - Wind
>> 과정		파일(F) 편집(E)
1 >> 6 : 0		0 1 29
2 >> 6 1 : 15		0 5 10
3 >> 6 1 2 : 31		1 2 16
4 >> 6 1 2 3 : 43		1 6 15
5 >> 6 1 2 3 4 : 65		2 3 12
6 >> 6 1 2 3 4 5 : 92		3 4 22
7 >> 6 1 2 3 4 5 0 : 102		3 6 18
<필요한 최소 비용 : 102>		4 5 27
C:\Users\wdmk46\OneDrive\바탕 화면\2학기		4 6 25
로 인해 종료되었습니다.		
이 창을 닫으려면 아무 키나 누르세요.		



## 2.5 느낀 점

이번 문제는 Prim알고리즘을 사용하여 최소 비용 신장 트리를 구하는 문제였습니다. Kruskal알고리즘은 간선의 가중치가 가장 작은 간선부터 차례대로 길을 구했지만 Prim알고리즘은 어느 한 정점부터 시작하여 최소 비용을 찾는다는 점을 알 수 있었습니다.

이번 문제에서 가장 어려웠다고 생각하는 부분은 prim함수를 구현하는 부분이었습니다. 특히 `get_min_vertex`함수로 가장 작은 `distance`를 반환받고 이 값으로 `distance`값을 `weight`로 업데이트 한다는 것이 처음에는 이해가 잘 안되었는데, 다시 코드 하나하나 읽어보고 과정을 그림을 그리면서 해보니 알고리즘을 이해할 수 있었습니다. 또한 저번에 했던 인접 행렬 코드를 이용하여 프로그래밍을 하니 더 쉽게 할 수 있었던 것 같습니다.

이번 문제를 통해 Prim알고리즘에 대해 알 수 있었습니다. 앞으로 이 Prim알고리즘을 어디에 쓸 수 있을지 생각해보고 어떻게 이용할 수 있을지 생각해볼 계기가 되었습니다.

### 3. 그래프를 하며 느낀 점

이번 기회를 통해 탐욕적인 방법에 대해 배울 수 있는 계기가 되었습니다. 탐욕적인 방법에 대해 처음으로 배웠기 때문에 처음엔 잘 이해가 되지 않는 부분이 많았지만 교수님께서 라인by라인으로 하나하나 설명해 주셨기 때문에 확실히 이해할 수 있었던 것 같습니다. 더하여 레포트를 쓰며 직접 코드를 작성하니 이에 대해 더 잘 이해할 수 있었습니다.

이번에 배운 Kruskal알고리즘과 Prim알고리즘을 어디에 어떻게 사용할지 생각해보는 계기가 되었습니다. 앞으로 그래프의 정점을 모두 탐색하는 최소 비용을 찾기 위해서 어떤 알고리즘을 사용해야 더 이득이 될지 생각해보고 그에 맞는 알고리즘을 사용해야겠다고 생각했습니다.

탐욕적인 방법이라는 알고리즘에 아직 많이 익숙하지는 않지만 관련된 알고리즘을 더 열심히 공부하여 Kruskal알고리즘과 Prim 필요할 때마다 사용할 수 있도록 노력해야겠다고 다짐했습니다.