숭실대학교 Soongsil University

알고리즘 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

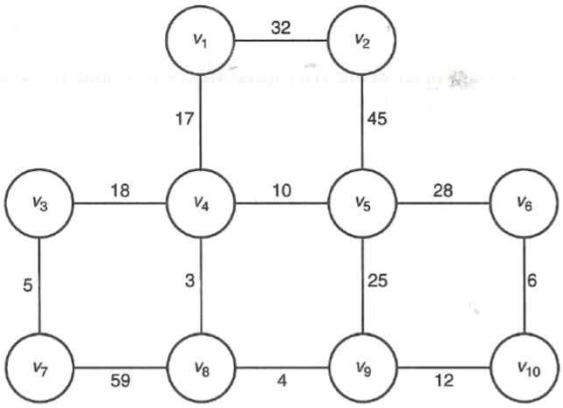
과목	알고리즘 2021
과제명	Prim Algoritm과 Kruskal Algoritm을 이용한 MST찾기
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20162449 김상현 [307]
제출일	2021년 11월 4일

차 례

1. 프로젝트 동기 및 목적

프림 알고리즘을 이해하고 MST (최소 신장 트리)를 구할 수 있도록 구현한다.
크루스칼 알고리즘을 이해하고 MST (최소 신장 트리)를 구할 수 있도록 구현한다.

2. 설계 구현 아이디어 (2번 문제)



주어진 그래프를 프로그래밍 상에서 구현하기 위해서 행렬로 표현한다.

변환 후 인접행렬로 나타냄 ->

```
int weight[MAX_VERTICES_PRIM][MAX_VERTICES_PRIM] = {// 신장트리의 거리와 모양을 배열로 표현, INF는 바로갈 수 있는 경로가 없음을 뜻한다.
 {0  ,32 ,INF,17 ,INF,INF,INF,INF,INF,INF},
 {32, 0, INF, INF, 45, INF, INF, INF, INF, INF},
{ INF,INF,0  ,18 ,INF,INF,5  ,INF,INF,INF },
{ 17 ,INF,18 ,0  ,10 ,INF,INF,3  ,INF,INF },
{ INF,45 ,INF,10 ,0  ,28 ,INF,INF,25 ,INF },
{ INF,INF,INF,INF,28 ,0  ,INF,INF,INF,6 },
{ INF,INF,5  ,INF,INF,INF,0  ,59 ,INF,INF },
{ INF,INF,INF,3  ,INF,INF,59 ,0  ,4  ,INF },
{ INF,INF,INF,INF,25 ,INF,INF,4  ,0  ,12 },
{ INF,INF,INF,INF,INF,6  ,INF,INF,12 ,0 }
};
```

먼저 **프림 알고리즘**을 간단히 그림으로 이해 해 보려고 한다. (뒷장)

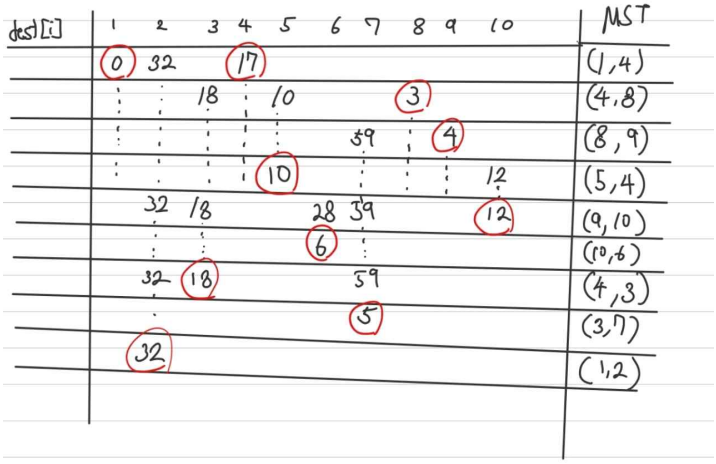
정점의 개수 만큼의 크기의 dist[] 배열을 선언한다. 정점간의 거리를 저장하는데 더 길이가 짧은 간선이 존재한다면 업데이트 하게 된다.

1. dest[] 배열은 시작 정점에 대해서 0으로 초기화 하고 나머지 모든 값을 INF 값으로 넣어준다.

2. 시작 정점에서 이어져 있는 모든 간선들의 가중치를 dist배열에 입력한다. 이제 배열에 들어있는 값 중 제일 작은 값을 찾고 그 가중치를 선택한다.

3. 선택한 가중치에 해당하는 정점에서 이어져 있는 간선들의 가중치를 다시 dist 배열에 입력한다. 이 때 이미 선택이 되었던 정점의 경우는 넣어도 의미 없으므로 넣지 않도록 하자. 또한 dist배열에 이미 들어있는 값이 새롭게 찾은 가중치의 값보다 큰 경우 dist배열의 값을 업데이트 해준다.

4. 위 과정을 반복하고 (총 정점의 개수 - 1) 만큼 반복하게 되면 모든 정점을 선택하였고 종료하게 된다.



<- 직접 손으로 표현한 MST 과정

```
int get_min_vertex(int n)
{
    int v, i; // 정점의 정보를 저장할 변수 v, 반복문을 위한 변수 i

    for (i = 0; i < n; i++)
    {
        if (selected[i] == FALSE) {
            v = i;  // 아직 선택되지 않은 정점의 번호를 v에 저장, 각 함수 실행별 0부터 n - 1까지 차례대로 저장된다.
            break;
        }
    }

    // 위에서 선택된 정점이 과연 최소거리를 지니고 있는 정점인지를 확인한다.
    for (i = 0; i < n; i++)
    {
        // 선택되지 않은 정점들을 순회하면서 최소거리를 가진 정점을 찾아낸다.
        if (selected[i] == FALSE && (dist[i] < dist[v]))
            v = i;  // 더 적은 거리가 존재한다면 해당 정점을 저장한다.
    }

    return(v);  // 최소의 거리를 갖는 정점이 선택됐으므로 정점 번호를 리턴한다.
}
```

get_min_vertex() : 각 반복마다 최소 거리를 갖는 정점을 찾는 함수

```
void prim(int s, int n){
    make_arr(n);
    int i, u, v;
    for (u = 0; u < n; u++){  // dist배열과 selected배열의 정보를 초기화
        dist[u] = INF;
        selected[u] = FALSE;
    }
    dist[s] = 0;  // 시작정점과 시작정점간의 거리는 0이다. 자기자신을 순환하는 경로는 없다고 가정한다.
    int min = 0;
    for (i = 0; i < n; i++){
        // 리턴된 정점 번호를 u에 저장한다. u는 최소거리를 가지는 정점.
        u = get_min_vertex(n);
        selected[u] = TRUE; //  해당 정점을 선택했다고 표시.

        for (v = 0; v < n; v++) { // 새로운 정점에서 얻은 이어진 가중치 정보 dist배열 업데이트
            // 선택된 u 정점을 기준으로 정점(u)과 연결되어 있는 정점까지의 거리를 각각 비교.
            if (weight[u][v] != INF) {  // 정점 u와 연결이 되어있고
                // 아직 선택되지 않았으며 해당 변(weight[u][v])의 길이가 기존의 dist[v] 값보다 작다면
                if (selected[v] == FALSE && weight[u][v] < dist[v])
                    dist[v] = weight[u][v]; // dist[v]의 값을 갱신.
            }
        }
        min += dist[u]; // 최소거리 산출
        print_set(dist[u]);
        printf("선택한 가중치: %d\t-->", dist[u]);
        printf("현재 최소값 : %d\n", min);
    }
    printf("\n최소값 : %d\n", min);
}
```

prim() 함수 : get_min_vertex() 함수에서 찾은 정점에서 얻는 새로운 가중치 정보를 이미 선택한 정점이 아니고 기존의 dist 배열 값보다 작으면 업데이트 해준다.

다음은 Kruscal 알고리즘이다. 크루스칼 알고리즘의 가장 기초의 개념은 간선 가중치를 오름차순으로 정렬하여 가중치가 낮은 정점부터 차례로 연결 시키는 것이다. 물론 제일 중요한 것은 정점을 연결하였을 때 아직 연결하지 못한 정점들이 남아 있을 때는 사이클이 생기면 안된다는 것이다.

먼저 모든 간선들을 가중치와 시작 정점, 끝 정점을 모두 int 형식을 담을 수 있는 구조체에 담는다.

```
typedef struct {
    int src; //시작정점 번호
    int dest; //끝 정점 번호
    int weight; //가중치
}forSort;

forSort arr[MAX_VER] = {0}; // 간선 별로 시작 정점과 끝 정점 , 가중치를 저장하기 위한 구조체 배열
```

이제 C library 함수 qsort() 함수를 이용하여 arr 구조체 배열을 weight (가중치) 기준으로 오름차순 정렬한다.

```
while (edge_count < (n - 1)) {
    if (i < index) {
        min = arr[i].weight;
        uset = find_set(arr[i].src);
        vset = find_set(arr[i].dest);
        if (uset != vset) {
            edge_count++;
            union_set(uset, vset);
            mincost += min;
            print_set(min);
            printf("선택한 가중치: %d\t-->", min);
            printf("현재 최소값 : %d\n", mincost);
        }
        i++;
    }
}
```

Kruscal() 함수에서의 최종 반복문 :

- find_set() 함수를 이용하여 arr[] 구조체에 저장되어있는 시작 정점, 끝 정점 두 개의 정점의 제일 위에 있는 부모 노드를 찾게 된다.
- union_set() 함수를 이용하여 두 개가 속한 집합이 다를 때 두 집합을 합치고 최소 비용 트리에 포함하여 가중치 값을 계산하는데 이용된다.

```
int find_set(int vertex) {
    int parent_value, p, i = -1;
    for (i = vertex; (parent_value = parent[i]) >= 0; i = parent_value)
        ;    // 원소의 parent를 찾아감
    p = i; //parent를 발견해냄.
    for (i = vertex; (parent_value = parent[i]) >= 0; i = parent_value)
        parent[i] = p;  // 원소의 parent
    return p;
}
```

find_set() 함수 : 사이클이 형성되는지 확인하는 함수이다.

첫 for 문에서는 parent[]의 값이 음수가 나올 때까지 확인한다. 음수가 나오면 해당 정점을 포함한 트리의 최상위 부모라는 뜻이기에 그 값을 저장한다.

두 번째 for문에서는 그 최상위 부모에 연결되어 있는 노드들의 parent[]를 모두 저장한 값으로 바꾸어 최상

위 부모를 알려준다. 또한 해당 저장 값을 리턴하여 최상위 정점의 index를 반환한다.

```
void union_set(int r1, int r2) {
    if (num[r1] < num[r2]) {
        parent[r1] = r2;   //r2 = 부모노드가 될것  r1  = 자식
        num[r2] += num[r1]; // r2의 갯수가 r1의 집합의 갯수만큼 들어남.
    }
    else {
        parent[r2] = r1;
        num[r1] += num[r2];
    }
}
```

union_set() 함수 : 두 트리를 연결하기 위해서 실행되는 함수이다.

연결시킬 두 개의 정점 r1과 r2 중 집합 내부의 정점의 수가 더 큰 쪽을 기준으로 해서 그쪽에 작은 트리를 연결시킨다. 또한 연결되는 쪽의 정점의 부모를 연결할 정점으로 바꾸어 주어야하고, 연결된 트리의 집합의 개수도 두 정점의 num의 값을 더해준다.

3. 수행결과 (문제 1번 / 문제 2번 / 문제 3번)

문제 1번 .

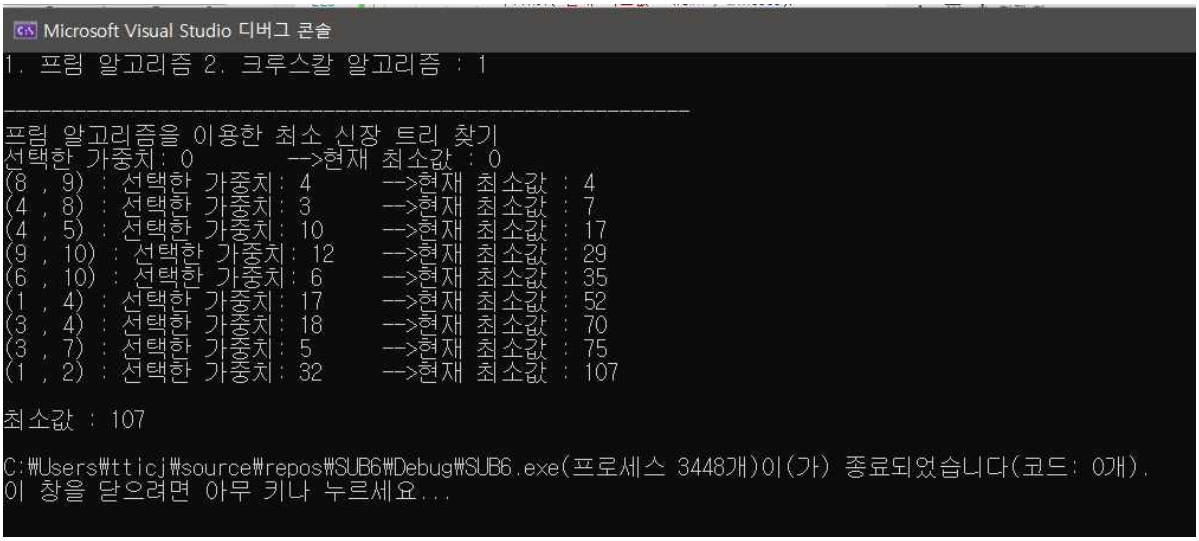
```
Microsoft Visual Studio 디버그 콘솔
1. 프림 알고리즘 2. 크루스칼 알고리즘 : 1

------------------------------------------------------------
프림 알고리즘을 이용한 최소 신장 트리 찾기
선택한 가중치 : 0        -->현재 최소값 : 0
(1 , 4) : 선택한 가중치 : 17     -->현재 최소값 : 17
(4 , 8) : 선택한 가중치 : 3      -->현재 최소값 : 20
(8 , 9) : 선택한 가중치 : 4      -->현재 최소값 : 24
(4 , 5) : 선택한 가중치 : 10     -->현재 최소값 : 34
(9 , 10) : 선택한 가중치 : 12    -->현재 최소값 : 46
(6 , 10) : 선택한 가중치 : 6     -->현재 최소값 : 52
(3 , 4) : 선택한 가중치 : 18     -->현재 최소값 : 70
(3 , 7) : 선택한 가중치 : 5      -->현재 최소값 : 75
(1 , 2) : 선택한 가중치 : 32     -->현재 최소값 : 107

최소값 : 107

C:₩Users₩tticj₩source₩repos₩SUB6₩Debug₩SUB6.exe(프로세스 5328개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

* 처음 시작 정점이 포함된 트리는 가중치가 0이기 때문에 간선을 나타내는 시작 정점과 끝 정점이 없다.

문제 2번.

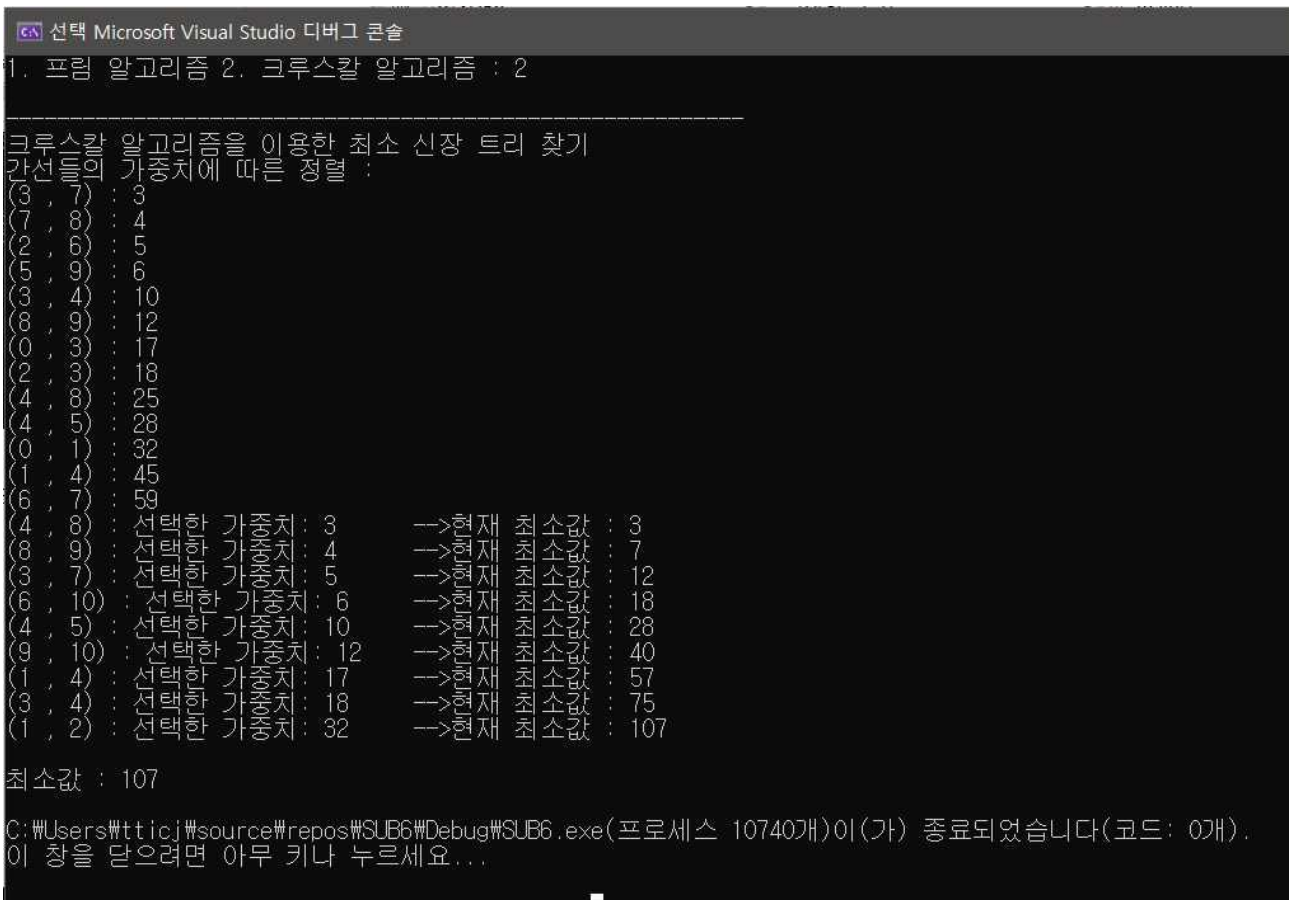
```
case 1:
    printf("프림 알고리즘을 이용한 최소 신장 트리 찾기₩n");
    //prim(0, 10);  // 정점 개수가 7개인 그래프에서 0번 정점을 출
    prim(8,10);
    break;
```

main() 함수에서 시작 정점을 8로 바꾸고 실행하였다. (뒷장)



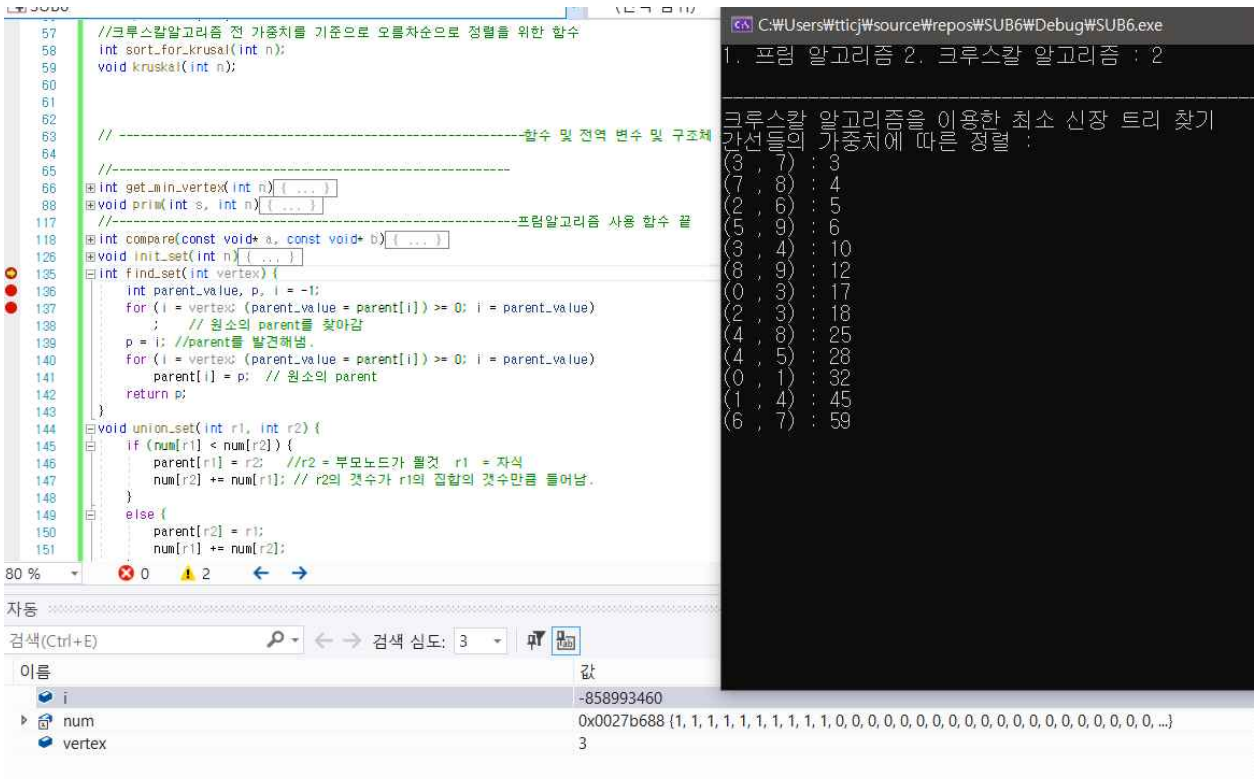
프림 알고리즘으로 찾아낸 최소 신장트리 (MST) 는 어떤 정점에서 시작해도 똑같다.

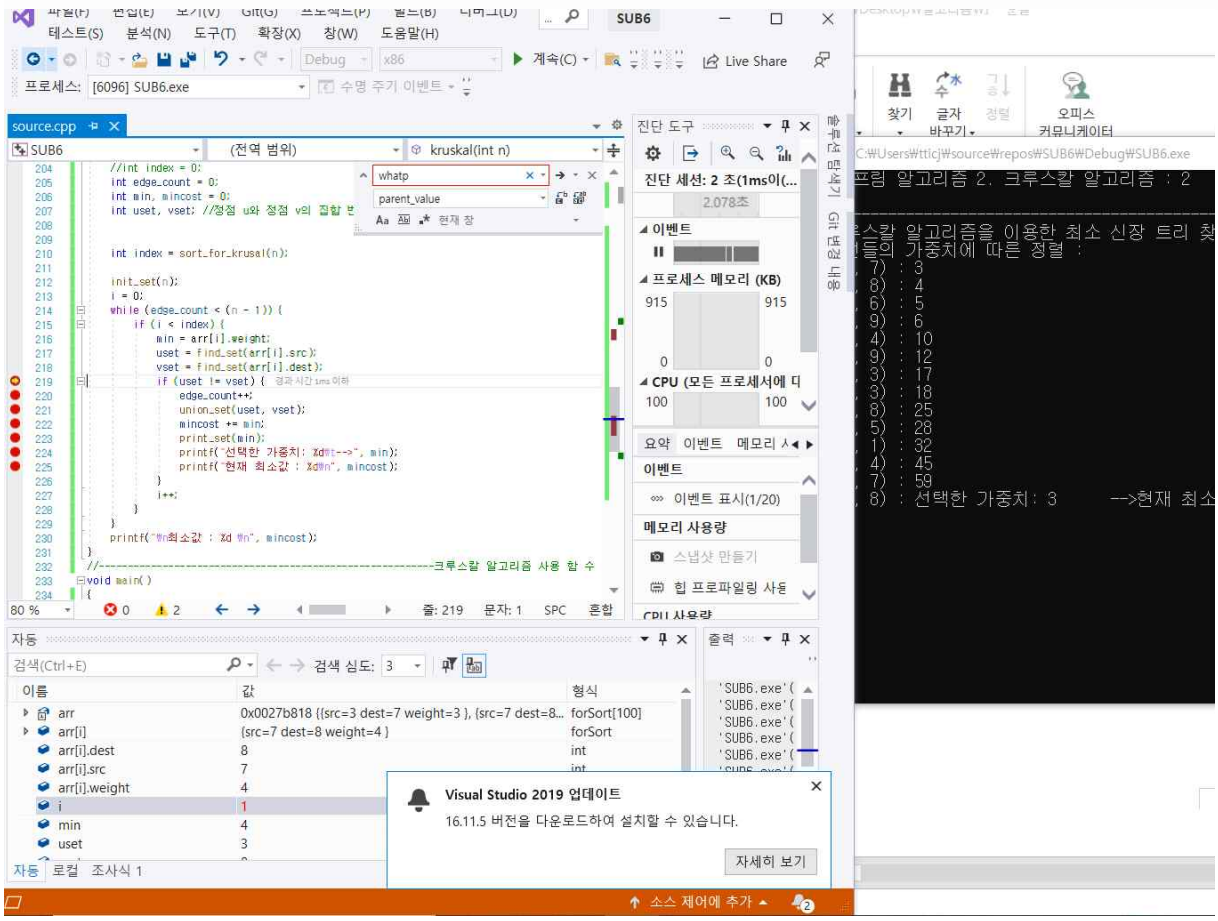
문제 3번.



* 가중치에 따른 오름차순 정렬한 arr[] 에 대해서도 출력하였다.

4. 디버깅


C:₩Users₩tticj₩source₩repos₩SUB6₩Debug₩SUB6.exe
1. 프림 알고리즘 2. 크루스칼 알고리즘 : 2
크루스칼 알고리즘을 이용한 최소 신장 트리 찾기
간선들의 가중치에 따른 정렬 :
(3 , 7) : 3
(7 , 8) : 4
(2 , 6) : 5
(5 , 9) : 6
(3 , 4) : 10
(8 , 9) : 12
(0 , 3) : 17
(2 , 3) : 18
(4 , 8) : 25
(4 , 5) : 28
(0 , 1) : 32
(1 , 4) : 45
(6 , 7) : 59
자동
검색(Ctrl+E)
검색 심도: 3
이름
값
i
-858993460
num
0x0027b688 {1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...}
vertex
3


SUB6
source.cpp
kruskal(int n)
Visual Studio 2019 업데이트
16.11.5 버전을 다운로드하여 설치할 수 있습니다.
자세히 보기

5. 소스 코드

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h >
#include <stdlib.h >
#define TRUE 1
#define FALSE 0
#define MAX_VER 100 //여기서는 모든 간선을 정렬전에 넣어야함으로 간선의 최대 수보다 크게 정의한다
#define MAX_VERTICES_PRIM 10  // 신장트리의 정점 개수
#define INF 1000L // 경로가 없는 INFINITE를 표현하기 위해 정의된 상수 INF
i n t weight[MAX_VERTICES_PRIM][MAX_VERTICES_PRIM] = {// 신장트리의 거리와 모양을 배열로 표현, INF는 바로갈 수 있는 경로가 없음을 뜻한다.
 {0  ,32 ,INF,17 ,INF,INF,INF,INF,INF,INF},
 {32,  0,  INF,  INF,  45,  INF,  INF,  INF,  INF,  INF},
{ INF,INF,0  ,18 ,INF,INF,5  ,INF,INF,INF },
{ 17 ,INF,18 ,0  ,10 ,INF,INF,3  ,INF,INF },
{ INF,45 ,INF,10 ,0  ,28 ,INF,INF,25 ,INF },
{ INF,INF,INF,INF,28 ,0  ,INF,INF,INF,6 },
{ INF,INF,5  ,INF,INF,INF,0  ,59 ,INF,INF },
{ INF,INF,INF,3  ,INF,INF,59 ,0  ,4  ,INF },
{ INF,INF,INF,INF,25 ,INF,INF,4  ,0  ,12 },
{ INF,INF,INF,INF,INF,6  ,INF,INF,12 ,0 }
};
i n t temp_weight[MAX_VERTICES_PRIM][MAX_VERTICES_PRIM];
int  selected[MAX_VERTICES_PRIM]; // 선택된 정점의 정보를 담을 배열, 선택이 됐는지 안됐는지를 표시한다.
int dist[MAX_VERTICES_PRIM]; // 최소의 거리 정보만을 담는 배열, 새로운 최소거리가 나올 때마다 갱신된다.
int parent[MAX_VER]; // 각 정점 부모의 노드가 저장된다 , 음수 이면 그 원소가 최상위 부오라는 뜻이고 음수와 함께 표시된 숫자가 자식의 수다.
int num[MAX_VER]; // 각 정점들의 집합의 개수를 저장한다.
typedef struct {
    int src; //시작정점 번호
    int dest; //끝 정점 번호
    int weight; //가중치
}forSort;
forSort arr[MAX_VER] = {0}; // 간선 별로 시작 정점과 끝 정점 , 가중치를 저장하기 위한 구조체 배열
// 간선들을 구조체 내에 저장하기 위한 함수
int make_arr(int n);
// 최소 dist[v]값을 갖는 정점을 반환
int get_min_vertex(int n);
// Prim, s는 시작 정점 , n 은 정점의 수
void prim(int s, int n);
// 오름차순 qsort 에 쓰이는 비교함수
int compare(const void * a, const void * b);
// 초기화 용 함수
void init_set(int n);
//원소의 루트 노드를 찾아가서 사이클이 형성되는지 확인하기위한 함수
int find_set(int vertex);
//사이클이 형성되지 않는 두개의 트리집합을 연결, 단 숫자가 더 작은 쪽이 큰 쪽의 자식으로 들어가는 것이 좋다.
void union_set(int r1, int r2);
//가중치를 가지고 해당 가중치를 가지는 정점 2개를 찾아서 프린트하는 함수
void print_set(int);
//크루스칼알고리즘 전 가중치를 기준으로 오름차순으로 정렬을 위한 함수
int sort_for_krusal(int n);
void kruskal(int n);
/                                                                                /
--------------------------------------------------------------------------함수 및 전역 변수 및 구조체 선언 끝
//--------------------------------------------------------------------------
int get_min_vertex(int n)
{
    int v, i; // 정점의 정보를 저장할 변수 v, 반복문을 위한 변수 i
    for (i =0; i < n; i ++)
    {
        if (selected[i] == FALSE) {
            v = i;  // 아직 선택되지 않은 정점의 번호를 v에 저장, 각 함수 실행별 0부터 n - 1까지 차례대로 저장된다.
            break;
        }
    }
    // 위에서 선택된 정점이 과연 최소거리를 지니고 있는 정점인지를 확인한다.
    for (i =0; i < n; i ++)
    {
        // 선택되지 않은 정점들을 순회하면서 최소거리를 가진 정점을 찾아낸다.
        if (selected[i] == FALSE && (dist[i] < dist[v]))
            v = i;  // 더 적은 거리가 존재한다면 해당 정점을 저장한다.
    }
    return(v);  // 최소의 거리를 갖는 정점이 선택됐으므로 정점 번호를 리턴한다.
}
void prim(int s, int n){
    make_arr(n);
    int i, u, v;
    for (u =0; u < n; u ++){   // dist배열과 selected배열의 정보를 초기화
        dist[u] = INF;
```

```
        selected[u] = FALSE;
    }
    dist[s] =0;  // 시작정점과 시작정점간의 거리는 0이다. 자기자신을 순환하는 경로는 없다고 가정한다.
    int min =0;
    for (i =0; i < n; i ++){
        // 리턴된 정점 번호를 u에 저장한다. u는 최소거리를 가지는 정점.
        u = get_min_vertex(n);
        selected[u] = TRUE; //  해당 정점을 선택했다고 표시.

        for (v =0; v < n; v ++) { // 새로운 정점에서 얻은 이어진 가중치 정보 dist배열 업데이트
            // 선택된 u 정점을 기준으로 정점(u)과 연결되어 있는 정점까지의 거리를 각각 비교.
            if (weight[u][v] != INF) {  // 정점 u와 연결이 되어있고
                // 아직 선택되지 않았으며 해당 변(weight[u][v])의 길이가 기존의 dist[v] 값보다 작다면
                if (selected[v] == FALSE && weight[u][v] < dist[v])
                    dist[v] = weight[u][v]; // dist[v]의 값을 갱신.
            }
        }
        min += dist[u]; // 최소거리 산출
        print_set(dist[u]);
        printf("선택한 가중치: %d₩t-->", dist[u]);
        printf("현재 최소값 : %d₩n", min);
    }
    printf("₩n최소값 : %d₩n", min);
}
//-----------------------------------------------------------프림알고리즘 사용함수 끝
int compare(const void * a, const void * b) {
    const forSort * m1 = (const forSort *)a;
    const forSort * m2 = (const forSort *)b;
    return m1 ->weight - m2 ->weight;
}
void init_set(int n) {
    int i;
    for (i =0; i < n; i ++) {
        //- : 정점, 음수의 숫자 : 집합의 갯수, 양수의 수 : 원소의 부모
        parent[i] =-1;
        // 각 정점들의 집합의 갯수
        num[i] =1;
    }
}
int find_set(int vertex) {
    int parent_value, p, i =-1;
    for (i = vertex; (parent_value = parent[i]) >=0; i = parent_value)
        ;    // 원소의 parent를 찾아감
    p = i; //parent를 발견해냄.
    for (i = vertex; (parent_value = parent[i]) >=0; i = parent_value)
        parent[i] = p;  // 원소의 parent
    return p;
}
void union_set(int r1, int r2) {
    if (num[r1] < num[r2]) {
        parent[r1] = r2;    //r2 = 부모노드가 될것 r1 = 자식
        num[r2] += num[r1]; // r2의 갯수가 r1의 집합의 갯수만큼 들어남.
    }
    else {
        parent[r2] = r1;
        num[r1] += num[r2];
    }
}
void print_set(int weight) {
    //printf("print_set에 인자로 들어온 weight : %d₩n", weight);
    if (weight ==0) {
        return;
    }
    for (int i =0; i < MAX_VER; i ++) {
        if (arr[i].weight == INF)
            break;
        if (arr[i].weight == weight) {
            printf("(%d , %d) : ", arr[i].src+1, arr[i].dest +1);
            break;
        }
    }
}
int make_arr(int n) {
    for (int i =0; i < MAX_VERTICES_PRIM; i ++) {
        for (int j =0; j < MAX_VERTICES_PRIM; j ++) {
            temp_weight[i][j] = weight[i][j];
        }
    }
    int index =0;
    for (int i =0; i < n; i ++) {
        for (int j =0; j < n; j ++) {
            if (temp_weight[i][j] !=0 && temp_weight[i][j] != INF) {
                arr[index].weight = weight[i][j];
                temp_weight[j][i] =0;
                arr[index].src = i;      // 해당 가중치의 정점 2개를 기억하기위한 배열
                arr[index].dest = j;
                //printf("(%d , %d) : %d₩n", arr[index].src+1,  arr[index].dest+1,
```

```
arr[index].weight);
                index++;
            }
        }
    }
    return index;
}
int sort_for_krusal(int n) {

    int index = make_arr(n);
    qsort(arr, index, sizeof(forSort), compare);
//-->arr배열에 저장된 가중치가 오름차순으로 정리
    printf("간선들의 가중치에 따른 정렬 :₩n");
    for (int i =0; i < index; i ++) {      // -->qsort확인
        if (arr[i].weight == INF)
            break;
        printf("(%d , %d) : %d₩n", arr[i].src, arr[i].dest, arr[i].weight);
    }
    return index;
}
void kruskal(int n) {
    int i, j;
    //int index = 0;
    int edge_count =0;
    int min, mincost =0;
    int uset, vset; //정점 u와 정점 v의 집합 번호
    int index = sort_for_krusal(n);
    init_set(n);
    i =0;
    while (edge_count < (n -1)) {
        if (i < index) {
            min = arr[i].weight;
            uset = find_set(arr[i].src);
            vset = find_set(arr[i].dest);
            if (uset != vset) {
                edge_count++;
                union_set(uset, vset);
                mincost += min;
                print_set(min);
                printf("선택한 가중치: %d₩t-->", min);
                printf("현재 최소값 : %d₩n", mincost);
            }
            i++;
        }
    }
    printf("₩n최소값 : %d ₩n", mincost);
}
//-----------------------------------------------------크루스칼 알고리즘 사용 함 수 끝
void main()
{
    int input;
    printf("1. 프림 알고리즘 2. 크루스칼 알고리즘 : "); scanf("%d", &input);

printf("₩n-----------------------------------------------------------₩n");
    switch (input)
    {
    case 1:
        printf("프림 알고리즘을 이용한 최소 신장 트리 찾기₩n");
        //prim(0, 10);  // 정점 개수가 7개인 그래프에서 0번 정점을 출발하여 얻을 수 있는 최소비용신장트리를 찾아라.
        prim(8,10);
        break;
    case 2 :
        printf("크루스칼 알고리즘을 이용한 최소 신장 트리 찾기₩n");
        kruskal(10);
        break;
    default: printf("1 혹은 2 의 숫자만 입력해주세요 ₩n");
        break;
    }



}
/*********************************************
 ** End Line
*********************************************/
```