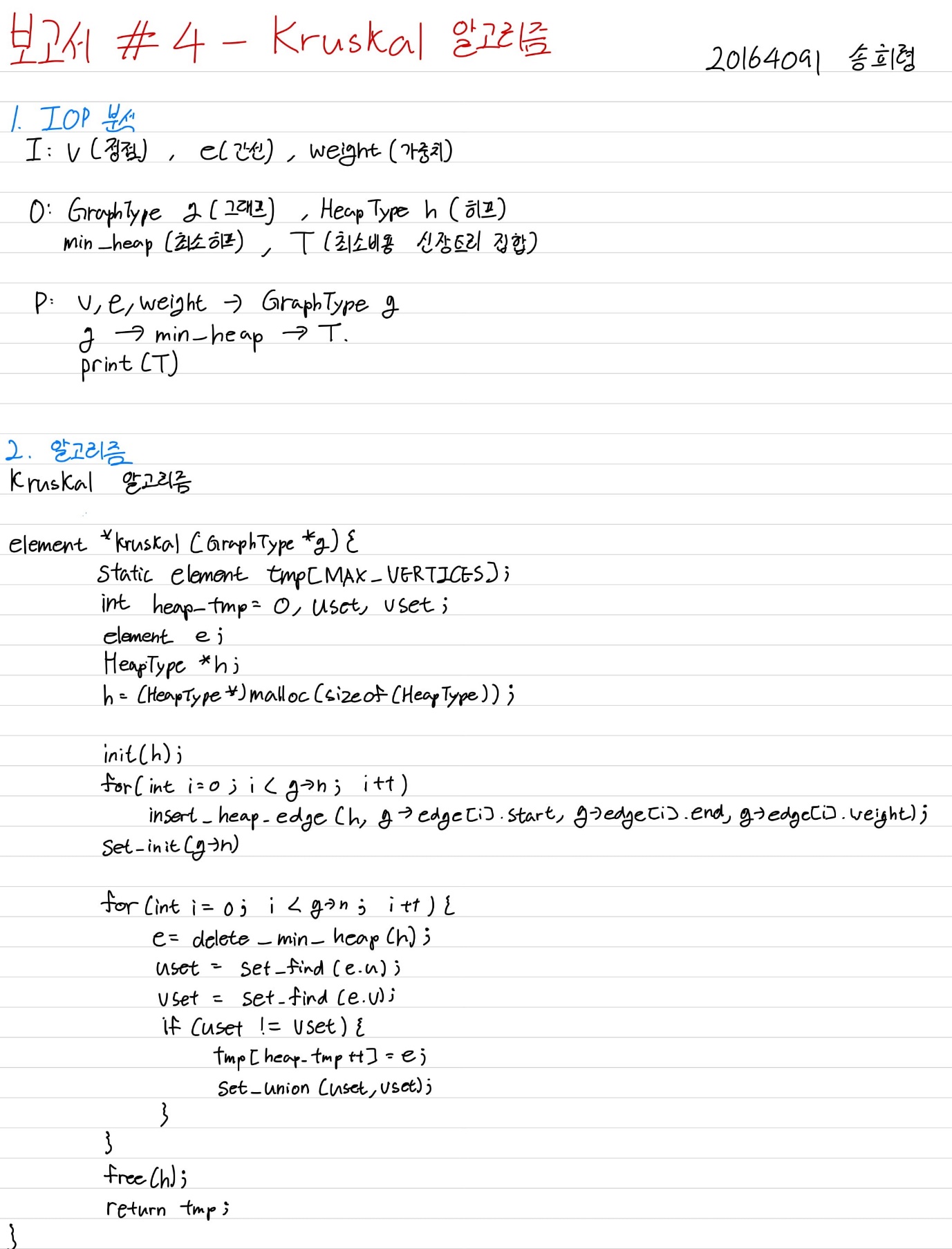
데이터구조 3

보고서 #4

(Kruskal 알고리즘)

20164091

송희령



3. 코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#define MAX\_VERTICES 100

#define INF 1000

#include <stdio.h>

#include <stdlib.h>

int parent[MAX\_VERTICES];

void set\_init(int n) {

    for (int i = 0; i < n; i++)

        parent[i] = -1;

}

사용할 헤더와 define매크로 설정입니다.

Parent를 이용하여 kruscal 알고리즘 정리를 합니다.

// curr가 속하는 집합을 반환한다

int set\_find(int curr) {

    if (parent[curr] == -1)

        return curr;

    while (parent[curr] != -1)

        curr = parent[curr];

    return curr;

}

// 두개의 원속가 속한 집합을 합친다.

void set\_union(int a, int b) {

    int root1 = set\_find(a);

    int root2 = set\_find(b);

    if (root1 != root2)

        parent[root1] = root2;

}

Kruskal 알고리즘을 적용 연산 시킬 함수들입니다.

struct Edge { // 간선을 나타내는 구조체

    int start, end, weight;

};

typedef struct GraphType {

    int v; // 정점의 개수

    int n; // 간선의 개수

    struct Edge edges[MAX\_VERTICES];

} GraphType;

// 그래프 초기화

void graph\_init(GraphType \*g) {

    g->v = 0;

    g->n = 0;

    for (int i = 0; i < MAX\_VERTICES; i++) {

        g->edges[i].start = 0;

        g->edges[i].end = 0;

        g->edges[i].weight = INF;

    }

}

간선과 정점의 개수를 나타낼 엣지와 그래프 표현 입니다.

void insert\_vertex(GraphType \*g) {

    if (((g->v) + 1) > MAX\_VERTICES) {

        fprintf(stderr, "그래프 : 정점의 개수 초과");

        return;

    }

    g->v++;

}

// 간섭 삽입 연산

void insert\_edge(GraphType \*g, int start, int end, int w) {

    if (start >= g->v || end >= g->v || start < 0 || end < 0) {

        fprintf(stderr, "그래프 : 정점 번호 오류\n");

        return;

    }

    g->edges[g->n].start = start;

    g->edges[g->n].end = end;

    g->edges[g->n].weight = w;

    g->n++;

}

간선과 정점의 삽입을 수행할 함수입니다.

void gen\_graph(GraphType \*g) {

    int tmp\_v, max\_n = 0;

    printf("정점의 개수를 입력해 주세요 : ");

    scanf("%d", &tmp\_v);

    if (tmp\_v <= 1 || tmp\_v > MAX\_VERTICES)

        printf("입력오류");

    for (int i = 0; i < tmp\_v; i++)

        insert\_vertex(g); //간선 입력

    for (int i = 0; i <= tmp\_v - 1; i++)

        max\_n += i; //최대 간선 개수

    for (int i = 0; i < max\_n; i++) {

        int a, b, c;

        printf("간선의 각 정점과 비용을 입력해 주세요(취소 = 0 0 0) : ");

        scanf("%d %d %d", &a, &b, &c);

        if (a == 0 && b == 0 && c == 0)

            break;

        insert\_edge(g, a, b, c);

    }

}

정점의 개수를 삽입하고 간선을 입려해줄 함수 입니다.

typedef struct

{

    int key; // 간선의 가중치

    int u;   // 정점 1

    int v;   // 정점 2

} element;

typedef struct

{

    element heap[MAX\_VERTICES];

    int heap\_size;

} HeapType;

// 초기화 함수

void init(HeapType \*h) { h->heap\_size = 0; }

최소힙 구현을 위해 히프타입을 정의하고 초기화 해 줄 함수를 구현하였습니다.

// 삽입 함수

void insert\_min\_heap(HeapType \*h, element item) {

    int i;

    i = ++(h->heap\_size);

    //  트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

    while ((i != 1) && (item.key < h->heap[i / 2].key)) {

        h->heap[i] = h->heap[i / 2];

        i /= 2;

    }

    h->heap[i] = item; // 새로운 노드를 삽입

}

최소힙 삽입 함수 입니다.

// 삭제 함수

element delete\_min\_heap(HeapType \*h) {

    int parent, child;

    element item, temp;

    item = h->heap[1];

    temp = h->heap[(h->heap\_size)--];

    parent = 1;

    child = 2;

    while (child <= h->heap\_size) {

        // 현재 노드의 자식노드중 더 작은 자식노드를 찾는다.

        if ((child < h->heap\_size) &&

            (h->heap[child].key) > h->heap[child + 1].key)

            child++;

        if (temp.key <= h->heap[child].key)

            break;

        // 한단계 아래로 이동

        h->heap[parent] = h->heap[child];

        parent = child;

        child \*= 2;

    }

    h->heap[parent] = temp;

    return item;

}

최소힙 삭제 함수 입니다.

void insert\_heap\_edge(HeapType \*h, int u, int v, int weight) {

    element e;

    e.u = u;

    e.v = v;

    e.key = weight;

    insert\_min\_heap(h, e);

}

힙의 간선 정보와 가중치 정보를 담은 함수 입니다.

// kruskal의 최소비용 신장트리 프로그램

element \*Kruskal(GraphType \*g) {

    printf("kruskal 시작\n");

    static element tmp[MAX\_VERTICES];

    int heap\_tmp = 0, uset, vset;

    element e;

    HeapType \*h; // 최소 히프

    h = (HeapType \*)malloc(sizeof(HeapType));

    init(h); // 히프 초기화

    for (int i = 0; i < g->n; i++)

        insert\_heap\_edge(h, g->edges[i].start, g->edges[i].end, g->edges[i].weight);

    set\_init(g->n);

    for (int i = 0; i < g->n; i++) {

        e = delete\_min\_heap(h); // 최소 히프에서 삭제

        uset = set\_find(e.u);   // 정점 u의 집합 번호

        vset = set\_find(e.v);   // 정점 v의 집합 번호

        if (uset != vset) {     // 서로 속한 집합이 다르면

            tmp[heap\_tmp++] = e;

            set\_union(uset, vset); // 두개의 집합을 합친다.

        }

    }

    free(h);

    return tmp;

}

Kruskal 알고리즘의 구현 입니다.

void print(element T[]) { //출력

    printf("결과 출력\n");

    for (int i = 0; i < MAX\_VERTICES; i++) {

        if (T[i].key == 0 && T[i].u == 0 && T[i].v == 0)

            break;

        printf("(%d, %d), %d\n", T[i].v, T[i].u, T[i].key);

    }

    printf("출력 종료\n");

}

int main() {

    GraphType \*g;

    g = (GraphType \*)malloc(sizeof(GraphType)); //그래프생성

    graph\_init(g);                              //그래프 초기화

    gen\_graph(g);                               //그래프 데이터 입력

    element \*T = Kruskal(g);

    print(T);

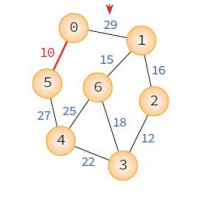
    free(g);

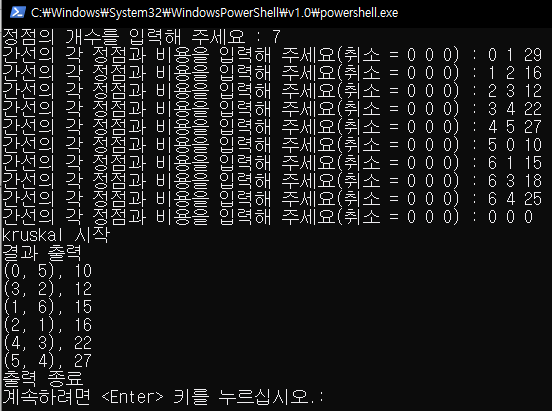
    return 0;

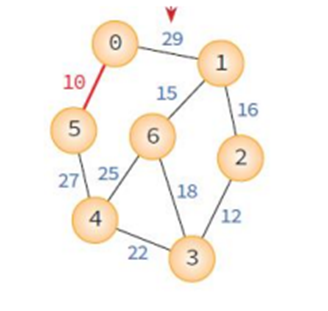
}

Kruscal 알고리즘을 통해 나온 최소비용신장트리값을 구할 메인함수와 출력을 담당할 프린트 함수 입니다.

4. 실행 결과

-1번 실행 예제-



n개(7)의 정점을 가지는 그래프에서 n-1개(6)개의 간선이 표현되고, 사이클은 존재하지 않으며 가중치의 합이 최소인 것을 확인할 수 있습니다.

6번2번

2

5번2번

2

2번2번

2

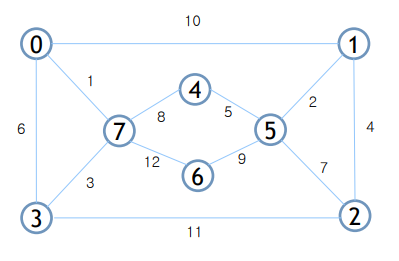
4번2번

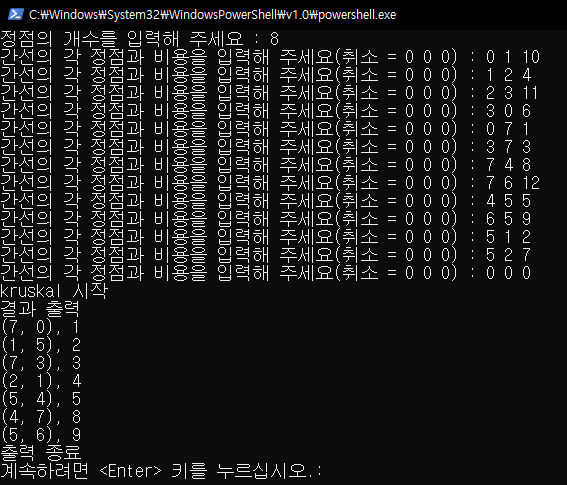
2

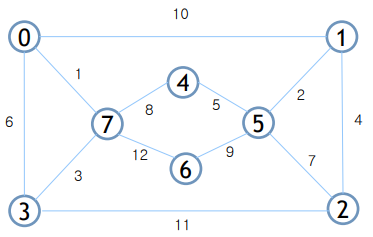
3번2번

2

1번

-2번 실행 예제-





.

7번2번

2

6번2번

2

5번2번

2

4번2번

2

3번2번

2

2번2번

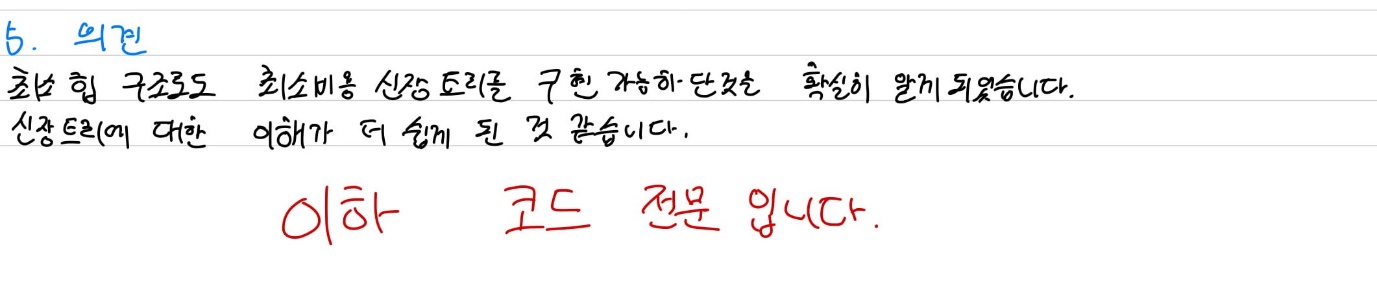
2

1번2번

2

역시 전과 마찬가지로 n개(7)의 정점을 가지는 그래프에서

n-1개(6)개의 간선이 표현되고, 사이클은 존재하지 않으며 가중치의 합이 최소인 것을 확인할 수 있습니다.



#define \_CRT\_SECURE\_NO\_WARNINGS

#define MAX\_VERTICES 100

#define INF 1000

#include <stdio.h>

#include <stdlib.h>

int parent[MAX\_VERTICES];

void set\_init(int n) {

    for (int i = 0; i < n; i++)

        parent[i] = -1;

}

// curr가 속하는 집합을 반환한다

int set\_find(int curr) {

    if (parent[curr] == -1)

        return curr;

    while (parent[curr] != -1)

        curr = parent[curr];

    return curr;

}

// 두개의 원속가 속한 집합을 합친다.

void set\_union(int a, int b) {

    int root1 = set\_find(a);

    int root2 = set\_find(b);

    if (root1 != root2)

        parent[root1] = root2;

}

struct Edge { // 간선을 나타내는 구조체

    int start, end, weight;

};

typedef struct GraphType {

    int v; // 정점의 개수

    int n; // 간선의 개수

    struct Edge edges[MAX\_VERTICES];

} GraphType;

// 그래프 초기화

void graph\_init(GraphType \*g) {

    g->v = 0;

    g->n = 0;

    for (int i = 0; i < MAX\_VERTICES; i++) {

        g->edges[i].start = 0;

        g->edges[i].end = 0;

        g->edges[i].weight = INF;

    }

}

// 정점삽입연산

void insert\_vertex(GraphType \*g) {

    if (((g->v) + 1) > MAX\_VERTICES) {

        fprintf(stderr, "그래프 : 정점의 개수 초과");

        return;

    }

    g->v++;

}

// 간섭 삽입 연산

void insert\_edge(GraphType \*g, int start, int end, int w) {

    if (start >= g->v || end >= g->v || start < 0 || end < 0) {

        fprintf(stderr, "그래프 : 정점 번호 오류\n");

        return;

    }

    g->edges[g->n].start = start;

    g->edges[g->n].end = end;

    g->edges[g->n].weight = w;

    g->n++;

}

void gen\_graph(GraphType \*g) {

    int tmp\_v, max\_n = 0;

    printf("정점의 개수를 입력해 주세요 : ");

    scanf("%d", &tmp\_v);

    if (tmp\_v <= 1 || tmp\_v > MAX\_VERTICES)

        printf("입력오류");

    for (int i = 0; i < tmp\_v; i++)

        insert\_vertex(g); //간선 입력

    for (int i = 0; i <= tmp\_v - 1; i++)

        max\_n += i; //최대 간선 개수

    for (int i = 0; i < max\_n; i++) {

        int a, b, c;

        printf("간선의 각 정점과 비용을 입력해 주세요(취소 = 0 0 0) : ");

        scanf("%d %d %d", &a, &b, &c);

        if (a == 0 && b == 0 && c == 0)

            break;

        insert\_edge(g, a, b, c);

    }

}

typedef struct

{

    int key; // 간선의 가중치

    int u;   // 정점 1

    int v;   // 정점 2

} element;

typedef struct

{

    element heap[MAX\_VERTICES];

    int heap\_size;

} HeapType;

// 초기화 함수

void init(HeapType \*h) { h->heap\_size = 0; }

// 히프 내용 출력 함수

void print\_heap(HeapType \*h) {

    int i;

    int level = 1;

    printf("\n===================");

    for (i = 1; i <= h->heap\_size; i++) {

        if (i == level) {

            printf("\n");

            level \*= 2;

        }

        printf("\t%d", h->heap[i].key);

    }

    printf("\n===================");

}

// 삽입 함수

void insert\_min\_heap(HeapType \*h, element item) {

    int i;

    i = ++(h->heap\_size);

    //  트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

    while ((i != 1) && (item.key < h->heap[i / 2].key)) {

        h->heap[i] = h->heap[i / 2];

        i /= 2;

    }

    h->heap[i] = item; // 새로운 노드를 삽입

}

// 삭제 함수

element delete\_min\_heap(HeapType \*h) {

    int parent, child;

    element item, temp;

    item = h->heap[1];

    temp = h->heap[(h->heap\_size)--];

    parent = 1;

    child = 2;

    while (child <= h->heap\_size) {

        // 현재 노드의 자식노드중 더 작은 자식노드를 찾는다.

        if ((child < h->heap\_size) &&

            (h->heap[child].key) > h->heap[child + 1].key)

            child++;

        if (temp.key <= h->heap[child].key)

            break;

        // 한단계 아래로 이동

        h->heap[parent] = h->heap[child];

        parent = child;

        child \*= 2;

    }

    h->heap[parent] = temp;

    return item;

}

void insert\_heap\_edge(HeapType \*h, int u, int v, int weight) {

    element e;

    e.u = u;

    e.v = v;

    e.key = weight;

    insert\_min\_heap(h, e);

}

// kruskal의 최소비용 신장트리 프로그램

element \*Kruskal(GraphType \*g) {

    printf("kruskal 시작\n");

    static element tmp[MAX\_VERTICES];

    int heap\_tmp = 0, uset, vset;

    element e;

    HeapType \*h; // 최소 히프

    h = (HeapType \*)malloc(sizeof(HeapType));

    init(h); // 히프 초기화

    for (int i = 0; i < g->n; i++)

        insert\_heap\_edge(h, g->edges[i].start, g->edges[i].end, g->edges[i].weight);

    set\_init(g->n);

    for (int i = 0; i < g->n; i++) {

        e = delete\_min\_heap(h); // 최소 히프에서 삭제

        uset = set\_find(e.u);   // 정점 u의 집합 번호

        vset = set\_find(e.v);   // 정점 v의 집합 번호

        if (uset != vset) {     // 서로 속한 집합이 다르면

            tmp[heap\_tmp++] = e;

            set\_union(uset, vset); // 두개의 집합을 합친다.

        }

    }

    free(h);

    return tmp;

}

void print(element T[]) { //출력

    printf("결과 출력\n");

    for (int i = 0; i < MAX\_VERTICES; i++) {

        if (T[i].key == 0 && T[i].u == 0 && T[i].v == 0)

            break;

        printf("(%d, %d), %d\n", T[i].v, T[i].u, T[i].key);

    }

    printf("출력 종료\n");

}

int main() {

    GraphType \*g;

    g = (GraphType \*)malloc(sizeof(GraphType)); //그래프생성

    graph\_init(g);                              //그래프 초기화

    gen\_graph(g);                               //그래프 데이터 입력

    element \*T = Kruskal(g);

    print(T);

    free(g);

    return 0;

}