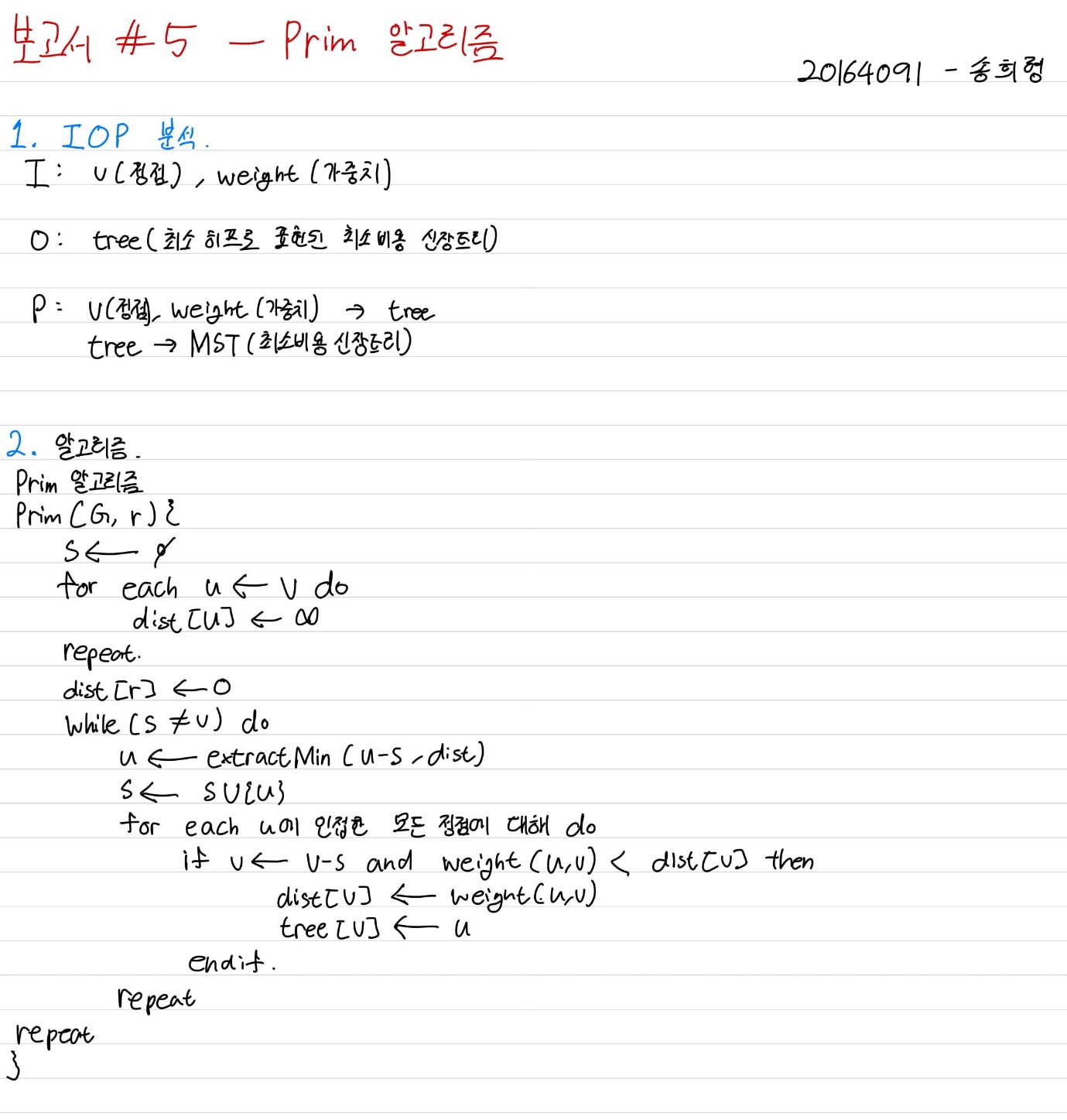
데이터구조 2

보고서 #5

(Prim 알고리즘)

20164091

송희령



3. 코드

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 2

#define MAX\_VERTICES 100

#define INF 1000

typedef struct GraphType {

    int n; //정점의 개수

    int weight[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

typedef struct {

    int key;      // 간선의 최소가중치

    int selected; // 선택여부

    int parent;   //부모

    int self;     //자신 노드

} element;

코드 실행에 필요한 헤더와 그래프, 노드, 변수들을 정의하였습니다.

// 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

}

최소히프 구현시 삽입할 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;

}

최소 히프 구현시 삭제할 함수 입니다.

HeapType \*prim(GraphType \*g, HeapType \*tree) {

    int i, s = 0, p = 0; //s-시작정점, p-부모저장

    element tmp;

    tree->heap\_size = 0;

    printf("----요소 입력----\n");

    for (int k = 0; k < g->n; k++) {

        tmp.key = g->weight[s][k];

        tmp.selected = 0;

        tmp.self = k;

        tmp.parent = MAX\_VERTICES;

        insert\_min\_heap(tree, tmp);

        printf("%d %d %d %d\n", tmp.self, tmp.key, tmp.parent, tmp.selected);

    }

    printf("\n----가중치 최소비용신장트리 변화단계----\n");

    printf("  노드번호, 가중치, 부모노드, 선택여부\n");

    for (int sta = 0; sta < g->n; sta++) {

        element \*tmp\_store = (element \*)malloc(sizeof(element) \* 4);

        HeapType \*tree2 = (HeapType \*)malloc(sizeof(HeapType) \* 4);

        HeapType \*tree\_init = (HeapType \*)malloc(sizeof(HeapType));

        tree2->heap\_size = 0;

        int tmp\_num = 0;

        for (int j = 0; j < g->n; j++) {

            tmp = delete\_min\_heap(tree);

            if (tmp.selected == 0 && tmp.key != INF)

                break;

            tmp\_store[tmp\_num++] = tmp;

        }

        tmp.selected = 1;

        tmp.parent = p;

        p = tmp.self;

        printf("\n");

        insert\_min\_heap(tree, tmp);

        for (int j = 0; j < tmp\_num; j++) {

            insert\_min\_heap(tree, tmp\_store[j]);

        }

        for (i = 0; i < g->n; i++) {

            element af = delete\_min\_heap(tree);

            insert\_min\_heap(tree2, af);

        }

        tree = tree2;

        tree2 = tree\_init;

        for (i = 0; i < g->n; i++) {

            for (int j = 0; j < g->n; j++) {

                if (tree->heap[i + 1].self == j)

                    if (tree->heap[i + 1].key > g->weight[p][j])

                        if (g->weight[p][j] && tree->heap[i + 1].selected != 1)

                            tree->heap[i + 1].key = g->weight[p][j];

            }

        }

        for (i = 0; i < g->n; i++) {

            printf("%d %d %d %d | ", tree->heap[i + 1].self, tree->heap[i + 1].key,

                   tree->heap[i + 1].parent, tree->heap[i + 1].selected);

        }

    }

    return tree;

}

프림 알고리즘을 구현한 함수입니다. 프림 알고리즘 실행 이후 배열 트리를 반환하도록 하였습니다.

void print(HeapType \*tree) {

    printf("\n\n-----간선 출력 시작-----\n");

    for (int i = 0; i < tree->heap\_size; i++) {

        printf("%d. 점 : %d, 가중치 : %d\n", i, tree->heap[i + 1].self, tree->heap[i + 1].key);

    }

}

int main(void) {

    //1번 예시

    /\*GraphType g = {7,

                   {{0, 29, INF, INF, INF, 10, INF},

                    {29, 0, 16, INF, INF, INF, 15},

                    {INF, 16, 0, 12, INF, INF, INF},

                    {INF, INF, 12, 0, 22, INF, 18},

                    {INF, INF, INF, 22, 0, 27, 25},

                    {10, INF, INF, INF, 27, 0, INF},

                    {INF, 15, INF, 18, 25, INF, 0}}};\*/

    //2번 예시

    GraphType g = {8,

                   {{0, 10, INF, 6, INF, INF, INF, 1},

                    {10, 0, 4, INF, INF, 2, INF, INF},

                    {INF, 4, 0, 11, INF, 7, INF, INF},

                    {6, INF, 11, 0, INF, INF, INF, 3},

                    {INF, INF, INF, INF, 0, 5, INF, 8},

                    {INF, 2, 7, INF, 5, 0, 9, INF},

                    {INF, INF, INF, INF, INF, 9, 0, 12},

                    {1, INF, INF, 3, 8, INF, 12, 0}}};

    HeapType \*tree = (HeapType \*)malloc(sizeof(HeapType) \* 4);

    tree = prim(&g, tree);

    print(tree);

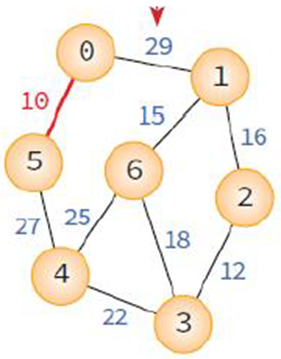
    free(tree);

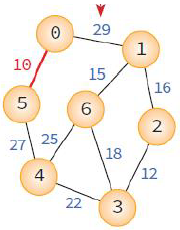
    return 0;

}

Prim 알고리즘통해 나온 겱과를 출력할 print함수와

실행을 할 main함수 입니다.

4. 실행결과

 -1번 예시-

7

6

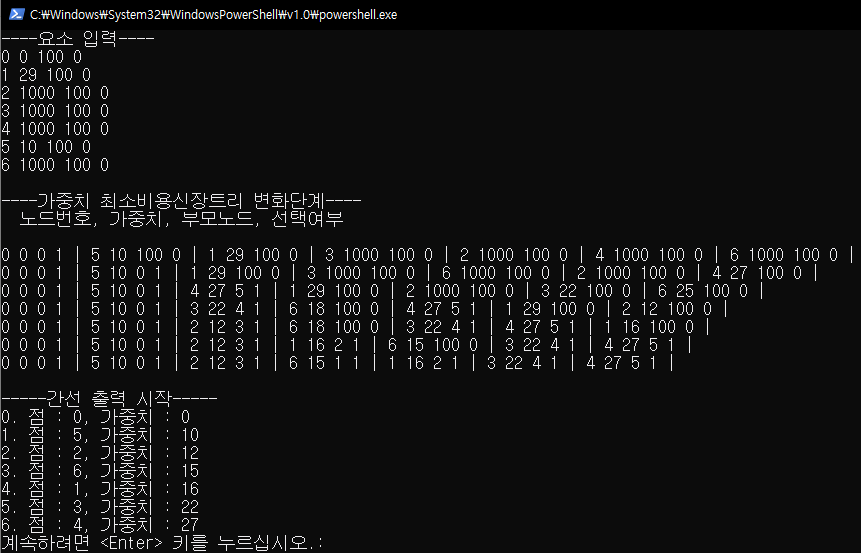
5

4

3

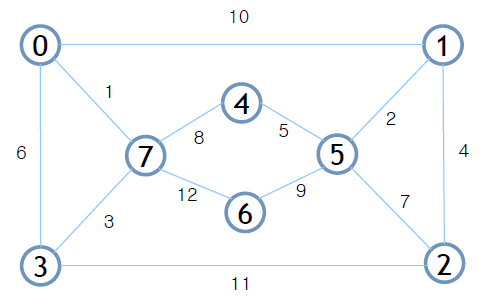
2

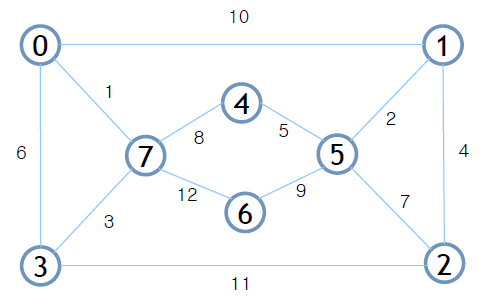
1



시작점(0)에서부터 변화하는 가중치에 따라 최소의 가중치를 선택하며 진행하는 것을 확인할 수 있습니다.

또한 사이클 발생 없이 온전히 찾은 것 또한 알 수 있습니다.

-2번 예제-



7=7=

6=

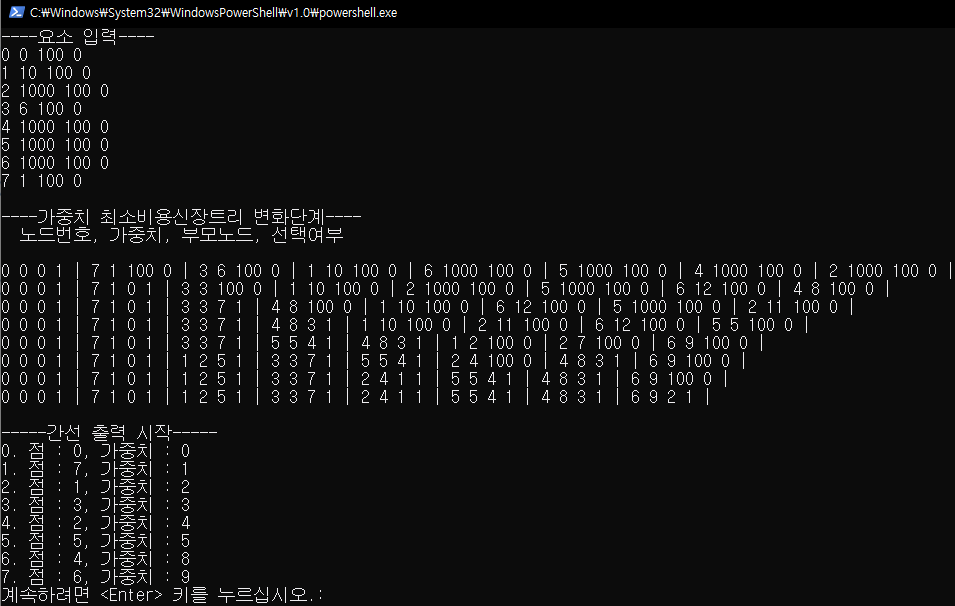
5=

4

3

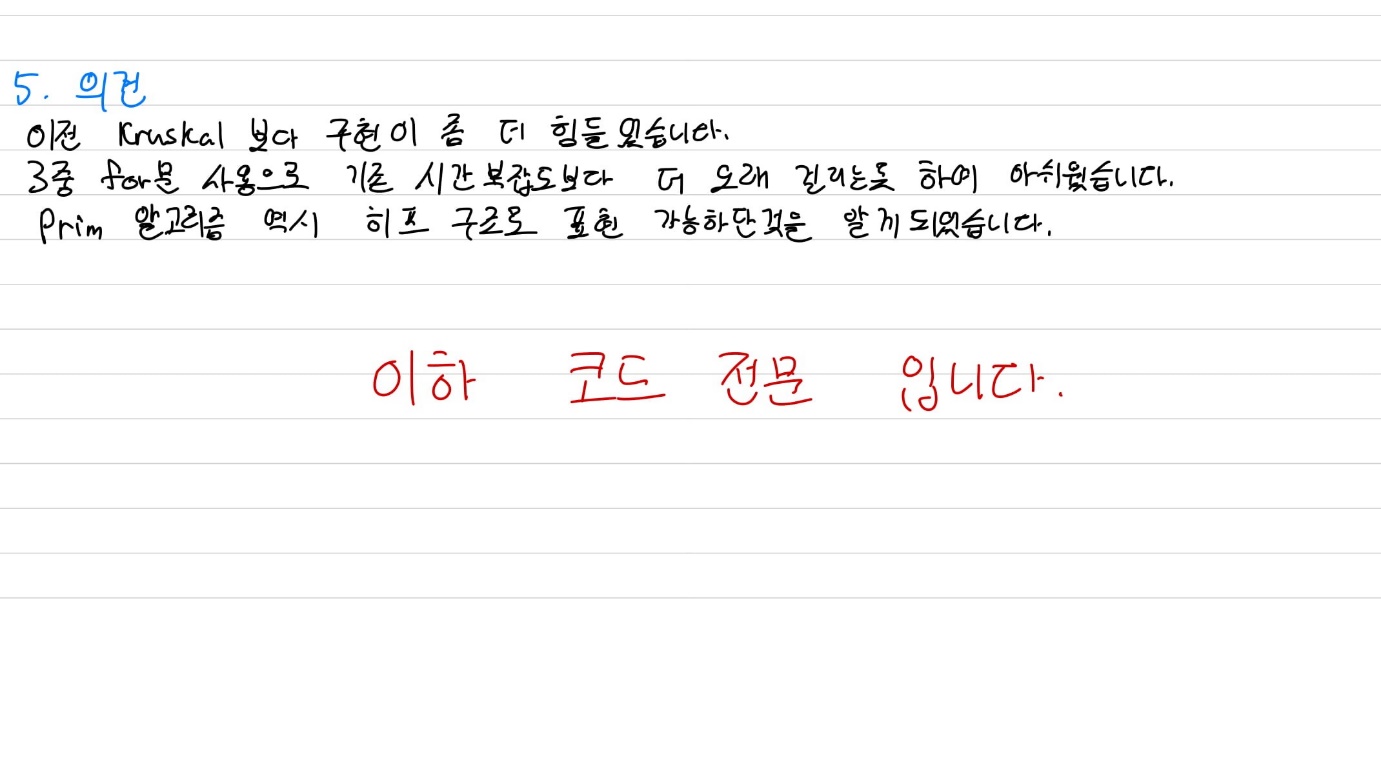
2

1



2번 예제 역시 시작점(0)에서부터 변화하는 가중치에 따라 최소의 가중치를 선택하며 진행하는 것을 확인할 수 있습니다.

마찬가지로 사이클 발생 없이 경로를 온전히 찾은 것 또한 알 수 있습니다.



#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 2

#define MAX\_VERTICES 100

#define INF 1000

typedef struct GraphType {

    int n; //정점의 개수

    int weight[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

typedef struct {

    int key;      // 간선의 최소가중치

    int selected; // 선택여부

    int parent;   //부모

    int self;     //자신 노드

} element;

typedef struct {

    element heap[MAX\_VERTICES];

    int heap\_size;

} HeapType;

// 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;

}

HeapType \*prim(GraphType \*g, HeapType \*tree) {

    int i, s = 0, p = 0; //s-시작정점, p-부모저장

    element tmp;

    tree->heap\_size = 0;

    printf("----요소 입력----\n");

    for (int k = 0; k < g->n; k++) {

        tmp.key = g->weight[s][k];

        tmp.selected = 0;

        tmp.self = k;

        tmp.parent = MAX\_VERTICES;

        insert\_min\_heap(tree, tmp);

        printf("%d %d %d %d\n", tmp.self, tmp.key, tmp.parent, tmp.selected);

    }

    printf("\n----가중치 최소비용신장트리 변화단계----\n");

    printf("  노드번호, 가중치, 부모노드, 선택여부\n");

    for (int sta = 0; sta < g->n; sta++) {

        element \*tmp\_store = (element \*)malloc(sizeof(element) \* 4);

        HeapType \*tree2 = (HeapType \*)malloc(sizeof(HeapType) \* 4);

        HeapType \*tree\_init = (HeapType \*)malloc(sizeof(HeapType));

        tree2->heap\_size = 0;

        int tmp\_num = 0;

        for (int j = 0; j < g->n; j++) {

            tmp = delete\_min\_heap(tree);

            if (tmp.selected == 0 && tmp.key != INF)

                break;

            tmp\_store[tmp\_num++] = tmp;

        }

        tmp.selected = 1;

        tmp.parent = p;

        p = tmp.self;

        printf("\n");

        insert\_min\_heap(tree, tmp);

        for (int j = 0; j < tmp\_num; j++) {

            insert\_min\_heap(tree, tmp\_store[j]);

        }

        for (i = 0; i < g->n; i++) {

            element af = delete\_min\_heap(tree);

            insert\_min\_heap(tree2, af);

        }

        tree = tree2;

        tree2 = tree\_init;

        for (i = 0; i < g->n; i++) {

            for (int j = 0; j < g->n; j++) {

                if (tree->heap[i + 1].self == j)

                    if (tree->heap[i + 1].key > g->weight[p][j])

                        if (g->weight[p][j] && tree->heap[i + 1].selected != 1)

                            tree->heap[i + 1].key = g->weight[p][j];

            }

        }

        for (i = 0; i < g->n; i++) {

            printf("%d %d %d %d | ", tree->heap[i + 1].self, tree->heap[i + 1].key,

                   tree->heap[i + 1].parent, tree->heap[i + 1].selected);

        }

    }

    return tree;

}

void print(HeapType \*tree) {

    printf("\n\n-----간선 출력 시작-----\n");

    for (int i = 0; i < tree->heap\_size; i++) {

        printf("%d. 점 : %d, 가중치 : %d\n", i, tree->heap[i + 1].self, tree->heap[i + 1].key);

    }

}

int main(void) {

    //1번 예시

    /\*GraphType g = {7,

                   {{0, 29, INF, INF, INF, 10, INF},

                    {29, 0, 16, INF, INF, INF, 15},

                    {INF, 16, 0, 12, INF, INF, INF},

                    {INF, INF, 12, 0, 22, INF, 18},

                    {INF, INF, INF, 22, 0, 27, 25},

                    {10, INF, INF, INF, 27, 0, INF},

                    {INF, 15, INF, 18, 25, INF, 0}}};\*/

    //2번 예시

    GraphType g = {8,

                   {{0, 10, INF, 6, INF, INF, INF, 1},

                    {10, 0, 4, INF, INF, 2, INF, INF},

                    {INF, 4, 0, 11, INF, 7, INF, INF},

                    {6, INF, 11, 0, INF, INF, INF, 3},

                    {INF, INF, INF, INF, 0, 5, INF, 8},

                    {INF, 2, 7, INF, 5, 0, 9, INF},

                    {INF, INF, INF, INF, INF, 9, 0, 12},

                    {1, INF, INF, 3, 8, INF, 12, 0}}};

    HeapType \*tree = (HeapType \*)malloc(sizeof(HeapType) \* 4);

    tree = prim(&g, tree);

    print(tree);

    free(tree);

    return 0;

}