20121277 김주호 HW6 보고서

1번.

- 사용할 전역 변수

```
int visited[MAX_VERTICES];
FILE* fp_input;
FILE* fp_output_DFS;
FILE* fp_output_BFS;
```

- int visited[MAX_VERTICES]

그래프 탐색 시에 방문한 vertex를 표시하는 일차원 배열

- FILE* fp_input

input.txt 파일과 연결하기 위한 FILE 구조체 포인터. input.txt 파일을 읽기 모드로 연다.

- FILE* fp_output_DFS

output_DFS.txt 파일과 연결하기 위한 FILE 구조체 포인터. output_DFS.txt 파일을 쓰기 모드로 연다. 텍스트 파일에 DFS 탐색 결과를 저장한다.

- FILE* fp_output_BFS

output_BFS.txt 파일과 연결하기 위한 FILE 구조체 포인터. output_BFS.txt 파일을 쓰기모드로 연다. 텍스트 파일에 BFS 탐색 결과를 저장한다.

- 사용할 자료구조와 ADT

- Graph

Graph 자료구조

```
typedef struct GraphNode {
        int vertex;
        struct GraphNode * link;
}GraphNode;

typedef struct GraphType {
        int E_num; // edge의 개수
        GraphNode* adj_list[MAX_VERTICES];
}GraphType;
```

다음은 Graph 자료구조의 ADT를 나타낸다.

- void graph_init()

그래프를 초기화하는 함수

input : GraphType* g

return : 없음

- void insert_vertex()

그래프 g에 vertex를 삽입한다.

input : GraphType* g, int v

return : 없음

- void insert_edge()

그래프 g에 edge를 삽입한다.

input : GraphType* g, int start, int end

return : 없음

● Queue

BFS 탐색 시에 활용할 queue 자료구조를 연결리스트로 선언하였다.

Queue 자료구조

```
typedef int element;
typedef struct QueueNode {
        element data;
        struct QueueNode * link;
}QueueNode;

typedef struct LinkedQueueType {
        QueueNode*front, * rear;
}LinkedQueueType;
```

- void init_Q()

Queue를 초기화한다.

input : LinkedQueueType* q

return : 없음

- int is_empty()

Queue의 공백여부를 판단한다.

input : LinkedQueueType* q

return : Queue가 비어있다면 1, Queue가 비어있지 않다면, 0

- void enqueue(LinkedQueueType* q, element data)

Queue에 data를 삽입하는 함수이다.

input : LinkedQueueType* q

return : 없음

- element dequeue(LinkedQueueType* q)

Queue에 저장된 data를 삭제하는 함수이다. 선입선출의 형태로 자료가 삭제된다.

input : LinkedQueueType* q

return : Queue에 저장된 data

- 함수 설명

- void dfs()

parameter를 통해 입력받은 그래프에 대해 DFS 탐색을 시행하여, 방문하는 vertex의 번호를 텍스트 파일에 저장하는 함수이다. fprintf()를 실행한 뒤, 해당 함수를 recursive하게 불러, DFS 탐색을 구현하였다.
input : GraphType* g, int v
return : 없음

- void bfs()

Queue 자료구조를 이용하여, BFS 탐색을 구현하였다. 하나의 vertex와 연결된 다른 vertex들을 Queue 자료구조에 enqueue 하였다가, 꺼내어 출력하는 방식으로 Level Order Traversal을 구현하였다.
input : GraphType* g, int v
return : 없음

- void main()

주어진 과제에서는 2가지 작업을 수행하여야 한다.
① input.txt 파일에 저장되어있는 인접 행렬을 읽어 프로그램 내에 인접 리스트 자료구조로 저장한다.
② 저장한 Graph에 대해 dfs 탐색과 bfs 탐색을 수행하여, 이를 프로그램 외부의 output_DFS.txt 파일, output_BFS.txt 파일에 한 줄에 하나의 connected component가 출력되도록 각각 저장한다.

작업 ①을 위해서, fp_input FILE 구조체 포인터를 이용하여 파일을 읽기 모드로 연 뒤, 다음의 반복문을 실행한다. 해당 소스코드는 다음과 같다.

```
if (fscanf(fp_input, "%d", &V_num) == -1) { // size 읽음
                printf("scanf err!");
                return ;
        }

for (int i = 0; i < V_num; i++) {
     for (int j = 0; j < V_num; j++) {
          if (fscanf(fp_input, "%d", &tmp) == -1) { // 숫자 읽음
```

```
            printf("scanf err!");
            return ;
        }
        if (tmp != 0) {
            insert_edge(g, i, j);
        }
    }
}
```

텍스트파일의 가장 첫 줄에 입력된 숫자인 size를 읽어들여 V_num에 저장한다. 그 뒤 인접 행렬의 1로 표시(연결되었음을 나타냄)된 부분의 열과 행을 읽어, 이를 edge의 양 끝 vertex로 생각한다. 해당 데이터를 프로그램 내 인접 리스트로 표현한 그래프의 edge로 삽입한다(insert_edge(g, i, j)).

작업 ②를 위해서 모든 node에 대해 dfs 혹은 bfs 탐색을 수행하되, 방문이 된 노드에 대해서는 탐색을 건너뛰도록 코드를 구현한다. visited[] 배열을 전역 변수로 선언하여, dfs 혹은 bfs 탐색시, 방문이 된 노드는 방문이 되었다는 표시(1)를 한다. (visited[v] = true)

- 전체 코드

```
#include <stdio.h >
#include <stdlib.h >
#define MAX_VERTICES 1000
#define true 1
#define false 0

int visited[MAX_VERTICES];
FILE* fp_input;
FILE* fp_output_DFS;
FILE* fp_output_BFS;

//// Graph 자료구조
typedef struct GraphNode {
        int vertex;
        struct GraphNode * link;
}GraphNode;
typedef struct GraphType {
        int E_num;
        GraphNode* adj_list[MAX_VERTICES];
}GraphType;

//// Queue 자료구조
```

```c
typedef int element;
typedef struct QueueNode {
	element data;
	struct QueueNode * link;
}QueueNode;
typedef struct LinkedQueueType {
	QueueNode*front, * rear;
}LinkedQueueType;

//// Queue
void init_Q(LinkedQueueType * q);
int is_empty(LinkedQueueType * q);
void enqueue(LinkedQueueType * q, element data);
element dequeue(LinkedQueueType* q);
//// Graph
void graph_init(GraphType * g);
void insert_vertex(GraphType * g, int v);
void insert_edge(GraphType * g, int start, int end);
//// dfs, bfs
void dfs(GraphType * g, int v);
void bfs(GraphType * g, int v);

////////////////////////////// Queue ////////////////////////////////
void init_Q(LinkedQueueType * q) {
	q->front = q ->rear =0;
}
int is_empty(LinkedQueueType * q) {
	return (q ->front ==NULL);
}
void enqueue(LinkedQueueType * q, element data) {
	QueueNode* tmp = (QueueNode *)malloc(sizeof(QueueNode));
	if (tmp ==NULL) {
		printf("alloc err!₩n");
		return;
	}
	tmp->data = data;
	tmp->link =NULL;
	if (is_empty(q)) {
		q->front = q ->rear = tmp;
	}
	else {
		q->rear ->link = tmp;
		q->rear = tmp;
	}
}
element dequeue(LinkedQueueType* q) {
	QueueNode* tmp = q ->front;
	element data;
	if (tmp ==NULL) {
```

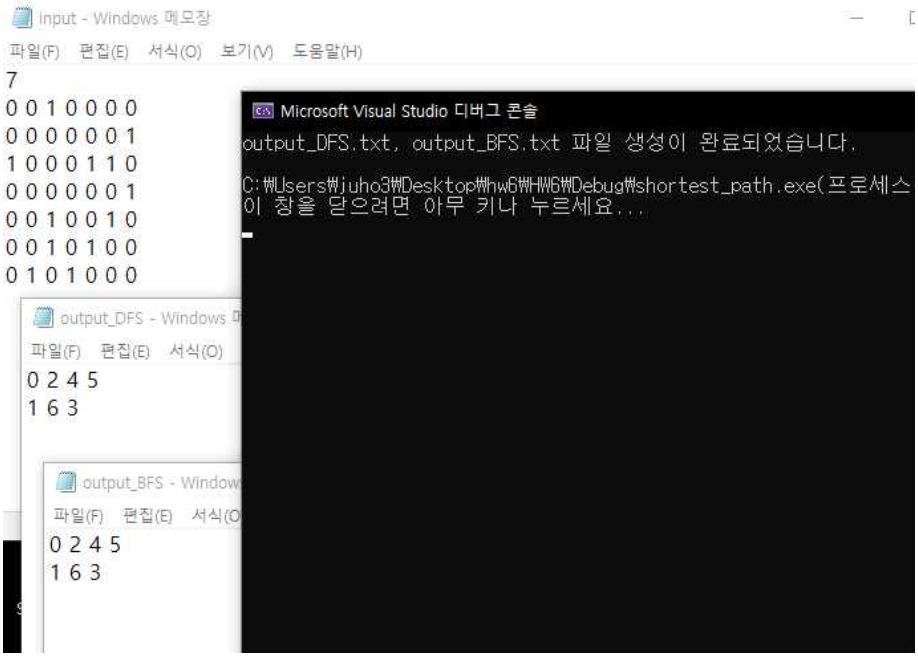
```
                printf("alloc err!₩n");
                return 0;
        }
        if (is_empty(q)) {
                printf("stack is empty!₩n");
                return 0;
        }
        else {
                data = tmp ->data;
                q->front = q ->front ->link;
                if (q ->front ==NULL) {
                        q->rear =NULL;
                }
                free(tmp);
                return data;
        }
}
////////////////////////////////////////////////////////////////////

////////////////////////////// Graph ////////////////////////////////
// 그래프 초기화
void graph_init(GraphType * g) {
        int v;
        g->E_num =0;
        for (v =0; v < MAX_VERTICES; v ++) {
                g->adj_list[v] =NULL;
        }
}
// 정점 삽입 연산
void insert_vertex(GraphType * g, int v) {
        if ((g ->E_num) +1 > MAX_VERTICES) {
                printf("그래프 : 정점 개수 초과₩n");
                return;
        }
        g->E_num ++;
}
// 간선 삽입 연산, e를 s의 인접리스트에 삽입한다.(s->e 연결)
void insert_edge(GraphType * g, int start, int end) {
        GraphNode* new_node = (GraphNode *)malloc(sizeof(GraphNode));
        if (new_node ==NULL) {
                printf("alloc err!₩n");
                return;
        }
        if (start >= g ->E_num ||end >= g ->E_num) {
                printf("그래프 : 정점 번호 오류₩n");
                return;
        }
        new_node->vertex =end;
        new_node->link =NULL;
```

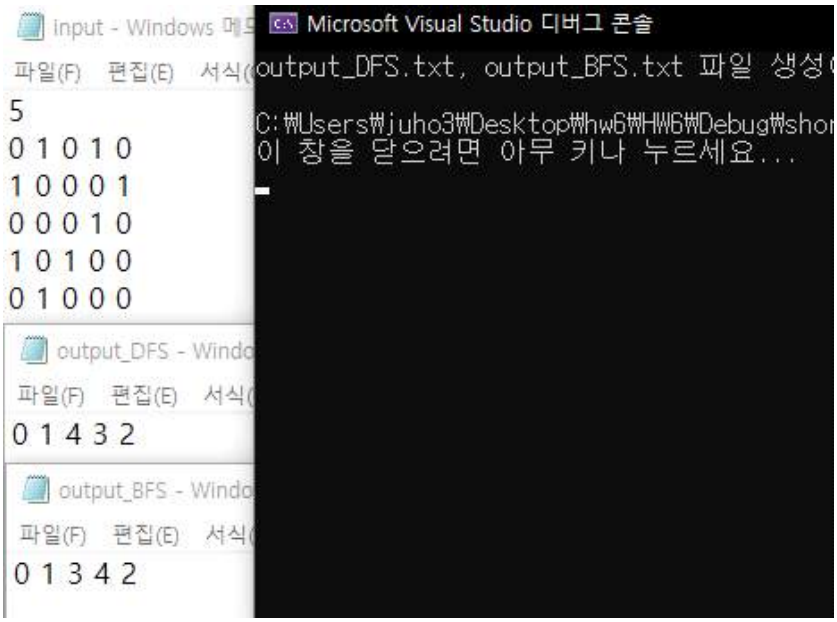
```
        GraphNode* ptr = g ->adj_list[start];
        if (ptr ==NULL) {
                g->adj_list[start] = new_node;
        }
        else {
                for (; ptr ->link; ptr = ptr ->link);
                ptr->link = new_node;
        }
}
// 출력(for test)
void print_adj_list(GraphType * g) {
        for (int i =0; i < g ->E_num; i ++) {
                GraphNode* ptr = g ->adj_list[i];
                printf("정점 %d의 인접리스트 ", i);
                while (ptr !=NULL) {
                        printf("-> %d", ptr ->vertex);
                        ptr = ptr ->link;
                }
                printf("₩n");
        }
}
////////////////////////////////////////////////////////////////

//////////////////////////// dfs, bfs ////////////////////////////
void dfs(GraphType * g, int v) {
        GraphNode* w;
        visited[v] =true;
        fprintf(fp_output_DFS, "%d ", v);
        for (w = g ->adj_list[v]; w; w = w ->link) {
                if (!visited[w ->vertex]) {
                        dfs(g, w->vertex);
                }
        }
}
void bfs(GraphType * g, int v) {
        GraphNode* w;
        LinkedQueueType q;
        init_Q(&q);

        visited[v] =true;
        fprintf(fp_output_BFS, "%d ", v);
        enqueue(&q, v);
        while (!is_empty(&q)) {
                v = dequeue(&q);
                for (w = g ->adj_list[v]; w; w = w ->link) {
                        if (!visited[w ->vertex]) {
                                visited[w->vertex] =true;
                                fprintf(fp_output_BFS, "%d ", w ->vertex);
                                enqueue(&q, w ->vertex);
```

```
                        }
                }
        }
}
////////////////////////////////////////////////////////////////

void main() {
        int V_num;
        int tmp;

        GraphType* g = (GraphType *)malloc(sizeof(GraphType));
        graph_init(g);
        if ((fp_input = fopen("input.txt", "r")) ==NULL) {
                printf("file open err!₩n");
                return;
        }
        if ((fp_output_DFS = fopen("output_DFS.txt", "w")) ==NULL) {
                printf("file open err!₩n");
                return;
        }
        if ((fp_output_BFS = fopen("output_BFS.txt", "w")) ==NULL) {
                printf("file open err!₩n");
                return;
        }
        if (fscanf(fp_input, "%d", &V_num) ==-1) { // size 읽음
                printf("scanf err!");
                return ;
        }
        for (int i =0; i < V_num; i ++) {
                insert_vertex(g, i);
        }
        for (int i =0; i < V_num; i ++) {
                for (int j =0; j < V_num; j ++) {
                        if (fscanf(fp_input, "%d", &tmp) ==-1) { // 숫자 읽음
                                printf("scanf err!");
                                return ;
                        }
                        if (tmp !=0) {
                                insert_edge(g, i, j);
                        }
                }
        }

        for (int i =0; i < V_num; i ++) {
                if (!visited[i]) {
                        dfs(g, i);
                        fprintf(fp_output_DFS, "₩n");
                }
        }
```

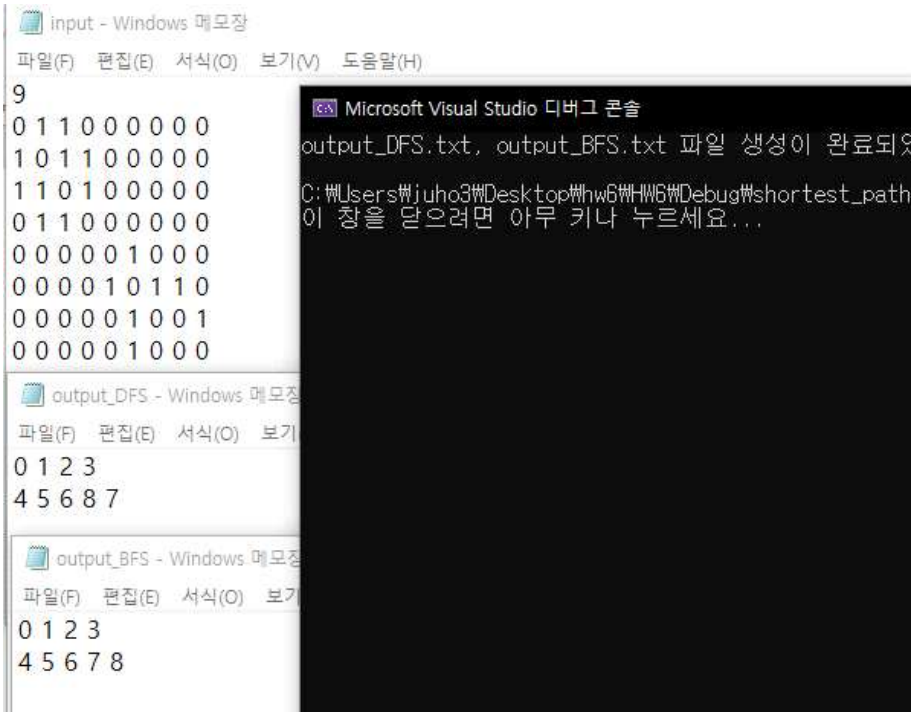
```
        for (int i =0; i < V_num; i ++) {
                visited[i] =false;
        }
        for (int i =0; i < V_num; i ++) {
                if (!visited[i]) {
                        bfs(g, i);
                        fprintf(fp_output_BFS, "₩n");
                }
        }
        printf("output_DFS.txt, output_BFS.txt 파일 생성이 완료되었습니다.₩n");
        free(g);
        fclose(fp_input); fclose(fp_output_DFS); fclose(fp_output_BFS);
}
```

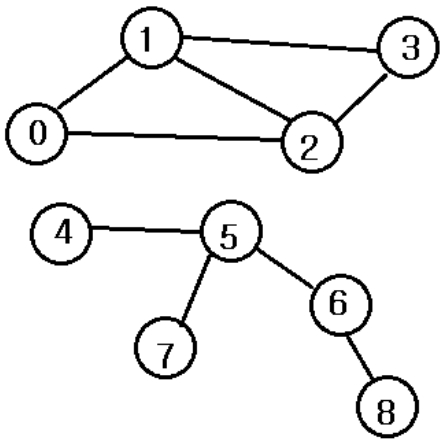
- 실행 결과





아래 test case는 입력에 해당하는 그래프 모양을 같이 그림으로 나타내었다.





2번.

- 사용할 전역 변수

```
int visited[MAX_VERTICES];
FILE* fp_input;
FILE* fp_output;
int parent[MAX_VERTICES];
```

- int visited[MAX_VERTICES]

그래프 탐색 시에 방문한 vertex를 표시하는 일차원 배열

- FILE* fp_input

input.txt 파일과 연결하기 위한 FILE 구조체 포인터. input.txt 파일을 읽기 모드로 연다.

- FILE* fp_output

output.txt 파일과 연결하기 위한 FILE 구조체 포인터. output.txt 파일을 쓰기 모드로 연다. 텍스트 파일에 DFS 탐색 결과와, MST의 cost를 저장한다.

- int parent[MAX_VERTICES]

kruskal 알고리즘 수행 시, edge에 부여된 weight가 작은 순서대로 새로운 edge를 선택한다. edge 하나는 vertex 2개(start_vertex, end_vertex)의 정보를 포함하게 되는데, 이 두 vertex가 같은 그룹에 속하게 되면, cycle이 만들어진다. 이는 '트리'라고 할 수 없으므로, 이런 상황을 배제해야 하는데, 두 vertex가 같은 그룹인지(부모가 같은지)에 대한 정보를 저장하는 일차원 배열이다.

- 사용할 자료구조와 ADT

● Graph

2번 과제에서 입력으로 주어지는 그래프는 1번과 달리 edge에 가중치가 주어지는 그래프이다. 따라서, edge가 단순히 vertex와 vertex의 연결성뿐만 아니라, weight에 대한 정보도 가지고 있어야 한다. 이를, 새로운 구조체로 정의하였다. 1번 문제에서 선언한 GraphType과는 달리, 추가로 edge에 대한 정보를 담고 있는 edges[]도 멤버 변수로 추가하였다.

Graph 자료구조

```
typedef struct Edge {
        int start, end, weight;
}Edge;

typedef struct GraphNode {
        int vertex;
        struct GraphNode * link;
}GraphNode;

typedef struct GraphType {
        int V_num; // vertex의 개수
        int E_num; // edge의 개수
        struct GraphNode * adj_list[MAX_VERTICES]; // vertex에 대한 정보
        struct Edge edges[MAX_VERTICES * MAX_VERTICES]; // edge에 대
한 정보
}GraphType;
```

다음은 Graph 자료구조의 ADT를 나타낸다.

- void graph_init()

그래프를 초기화하는 함수. edges 배열의 start, end, weight에 각각 0, 0, inf를 할당한다.

input : GraphType* g

return : 없음

- void insert_vertex()

그래프 g에 vertex를 삽입한다.

input : GraphType* g, int v

return : 없음

- void insert_edge()

그래프 g에 edge를 삽입한다. 문제 1과는 달리, weight가 parameter로 들어오게

된다.
input : GraphType* g, int start, int end, int w
return : 없음

- 함수 설명

- void set_init()

입력받은 정수 n 크기만큼 parent[] 배열의 원소 값을 -1로 초기화한다.
input : int n
return : 없음

- int set_find()

입력받은 정수 curr에 대해 이 curr가 어떤 그룹에 속하는지를 알려주게 된다. parent[curr]이 -1이 될 때까지, 계속하여 parent를 추적한다. 만일 어떤 두 정수 a, b에 대하여, set_find(a) == set_find(b)라면, a와 b가 같은 조상을 가졌다고 즉, 같은 집합 내의 원소라고 판정한다.
input : int curr
return : curr 원소의 최종 조상 number

- void set_union()

int a가 속한 집합과 int b가 속한 집합을 합친다. a의 조상(=root1)을 추적하고, b의 조상(=root2)을 추적하여, 이 둘이 같지 않다면, root1을 root2에 종속시키면서, 관계성을 갖게한다(parent[root1] = root2). 즉, 두 집합이 합쳐지게 된다.
input : int a, int b
return : 없음

- void insertion_sort()

kruskal 알고리즘을 수행하기 위해서는, 그래프의 edge들을 weight가 작은 순서대로 정렬해야 하고, 작은 순서대로 하나씩 선택해야 한다. 이를 위해 edge들을 정렬하는 함수가 필요한데, 이를 삽입 정렬 알고리즘으로 구현하였다.
input : Edge arr[], int size
return : 없음

- int kruskal()

오름차순으로 정렬된 edges[]에서 앞에서부터 원소를 하나씩 선택하면서, cycle을 만들지 않는다면, 선택하는 greedy 알고리즘이다. 가장 최소인 것부터 greedy하게 선택하면 MST가 얻어진다. 새로운 edge를 선택할 때, 해당 edge의 start vertex와

end vertex가 같은 집합에 속해있는지 점검(이때, set_find()가 이용된다.)한다. 선택될 때마다, weight를 sum 변수에 쌓아 모이게 하여, cost를 누적하고, 이 값을 return한다.

input : GraphType* g

return : MST의 cost

- **전체 코드**

```
#include <stdio.h >
#include <stdlib.h >
#define true 1
#define false 0
#define inf 10000
#define MAX_VERTICES 100

typedef struct Edge {
        int start, end, weight;
}Edge;

typedef struct GraphNode {
        int vertex;
        struct GraphNode * link;
}GraphNode;

typedef struct GraphType {
        int V_num; // vertex의 개수
        int E_num; // edge의 개수
        struct GraphNode * adj_list[MAX_VERTICES]; // vertex에 대한 정보
        struct Edge edges[MAX_VERTICES * MAX_VERTICES]; // edge에 대
한 정보
}GraphType;

int visited[MAX_VERTICES];
GraphType* MST;
FILE* fp_input;
FILE* fp_output;

////////////////////// Union Find ///////////////////////
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;
```

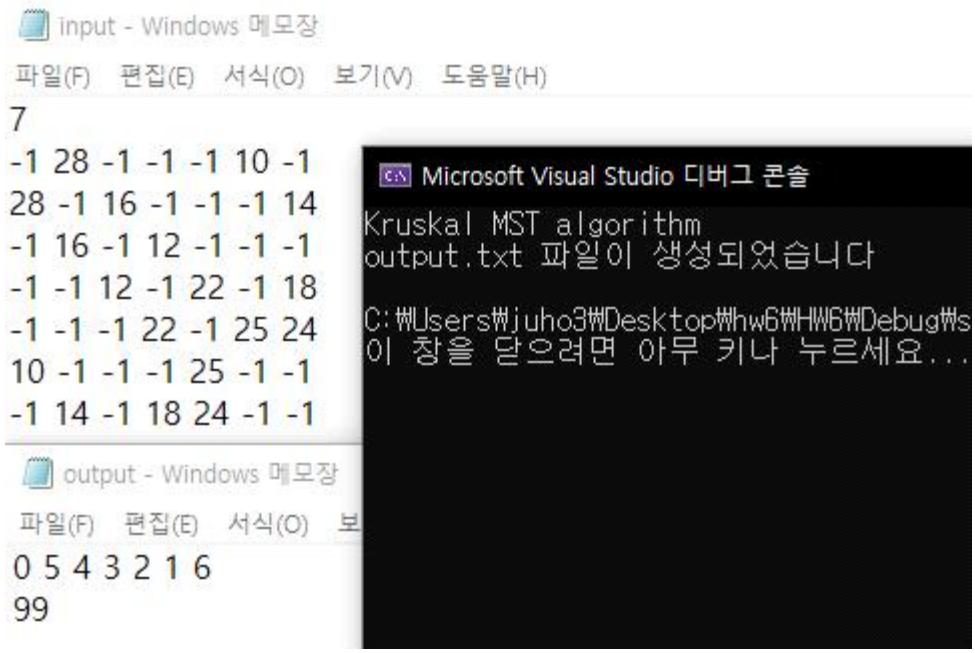
```
        while (parent[curr] !=-1) curr = parent[curr];
        return curr;
}
// 두개의 원소가 속한 집합을 합친다.
void set_union(int a, int b) {
        int root1 = set_find(a);
        int root2 = set_find(b);
        if (root1 != root2) {
                parent[root1] = root2;
        }
}
////////////////////////////////////////////////

////////////////// Kruskal ////////////////////
// 그래프 초기화
void graph_init(GraphType * g) {
        g->V_num =0;
        g->E_num =0;
        for (int v =0; v < MAX_VERTICES; v ++) {
                g->adj_list[v] =NULL;
        }
        for (int e =0; e < MAX_VERTICES * MAX_VERTICES; e ++) {
                g->edges[e].start =0;
                g->edges[e].end =0;
                g->edges[e].weight = inf;
        }
}
void insert_vertex(GraphType * g, int v) {
        if ((g ->V_num) +1 > MAX_VERTICES) {
                printf("그래프 정점 개수 초과!₩n");
                return;
        }
        g->V_num ++;
}
// 간선 삽입 연산
void insert_edge(GraphType * g, int start, int end, int w) {
        GraphNode* new_node;
        if (start >= g ->V_num ||end >= g ->V_num) {
                printf("그래프 정점 번호 오류!₩n");
                return;
        }
        new_node = (GraphNode *)malloc(sizeof(GraphNode));
        if (new_node ==NULL) {
                printf("alloc err!₩n");
                return;
        }

        new_node->vertex =end;
        new_node->link =NULL;
```

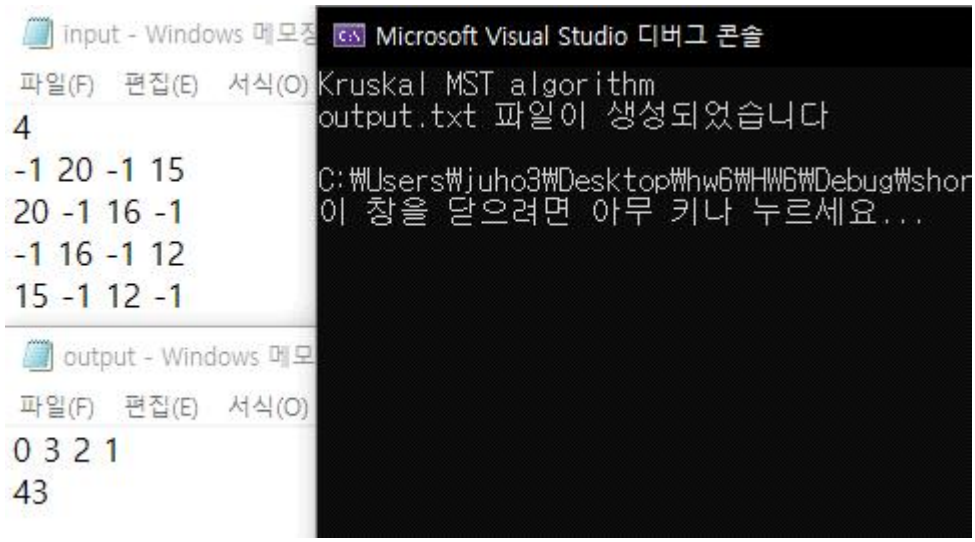
```
        GraphNode* ptr = g ->adj_list[start];
        if (ptr ==NULL) {
                g->adj_list[start] = new_node;
        }
        else {
                for (; ptr ->link; ptr = ptr ->link);
                ptr->link = new_node;
        }
        g->edges[g ->E_num].start = start;
        g->edges[g ->E_num].end =end;
        g->edges[g ->E_num].weight = w;
        g->E_num ++;
}
void insertion_sort(Edge arr[], int size) {
        int i, j;
        Edge key;
        for (i =1; i <size; i ++) {
                key = arr[i];
                for (j = i -1; j >=0 && arr[j].weight > key.weight; j --) {
                        arr[j +1] = arr[j];
                }
                arr[j +1] = key;
        }
}

// kruskal MST 프로그램
int kruskal(GraphType * g) {
        int edge_accepted =0; // 현재까지 선택된 간선의 수
        int uset, vset;      // 정점 u와 정점 v의 집합 번호
        struct Edge e;
        set_init(g->V_num); // 집합 초기화
        insertion_sort(g->edges, g ->E_num); // 정렬
        printf("Kruskal MST algorithm₩n");
        int i =0;
        int sum =0;
        while (edge_accepted < (g ->V_num -1)) {
                e = g ->edges[i];
                uset = set_find(e.start);
                vset = set_find(e.end);
                if (uset != vset) { // 서로 속한 집합이 다르면
                        //printf("edge (%d, %d) %d 선택₩n", e.start, e.end,
e.weight); // for test
                        insert_edge(MST, e.start, e.end, e.weight);
                        insert_edge(MST, e.end, e.start, e.weight);
                        sum += e.weight;
                        edge_accepted++;
                        set_union(uset, vset); // 두 개의 집합을 합친다.
                }
                i++;
```

```
        }
        return sum;
}

////////////////////////// dfs ////////////////////////////
void dfs(GraphType * g, int v) {
        GraphNode* w;
        visited[v] =true;
        fprintf(fp_output, "%d ", v);
        for (w = g ->adj_list[v]; w; w = w ->link) {
                if (!visited[w ->vertex]) {
                        dfs(g, w->vertex);
                }
        }
}

void main() {
        int V_num;
        int tmp;
        int cost;
        GraphType* g = (GraphType *)malloc(sizeof(GraphType));
        MST = (GraphType *)malloc(sizeof(GraphType));
        if ((fp_input = fopen("input.txt", "r")) ==NULL) {
                printf("file open err!");
                return;
        }
        if ((fp_output = fopen("output.txt", "w")) ==NULL) {
                printf("file open err!₩n");
                return;
        }
        if (fscanf(fp_input, "%d", &V_num) ==-1) { // size 읽음
                printf("scanf err!");
                return;
        }

        graph_init(g);
        for (int i =0; i < V_num; i ++) {
                insert_vertex(g, i);
        }
        for (int i =0; i < V_num; i ++) {
                for (int j =0; j < V_num; j ++) {
                        if (fscanf(fp_input, "%d", &tmp) ==-1) { // 숫자 읽음
                                printf("scanf err!");
                                return;
                        }
                        if (tmp !=-1) {
                                insert_edge(g, i, j, tmp);
                        }
                }
```

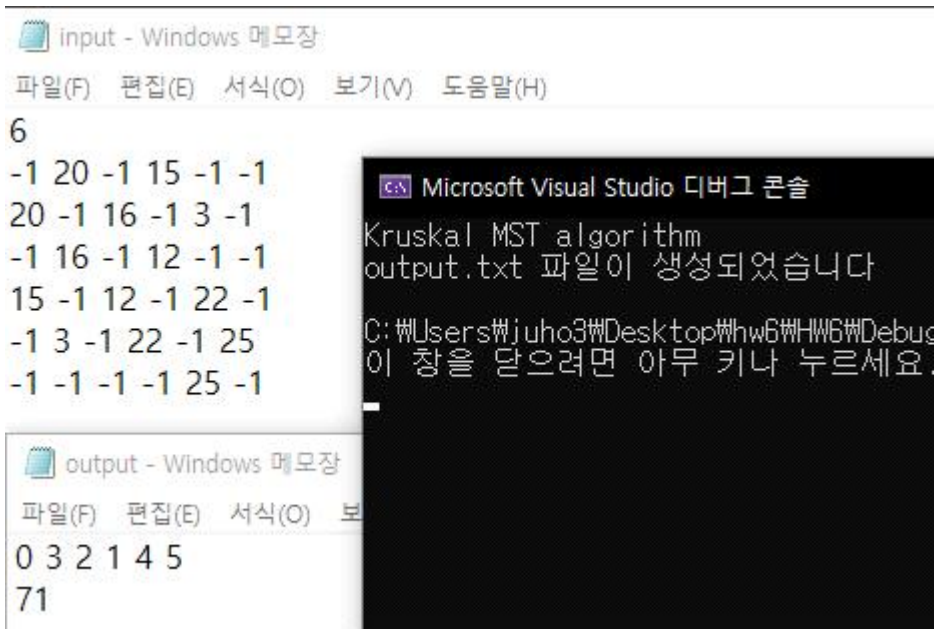
```
        }
        graph_init(MST);
        for (int i =0; i < V_num; i ++) {
                insert_vertex(MST, i);
        }
        cost = kruskal(g);
        dfs(MST, 0);
        fprintf(fp_output, "₩n%d", cost);
        free(g); free(MST);
        fclose(fp_input); fclose(fp_output);
        printf("output.txt 파일이 생성되었습니다₩n");
        return;
}
```

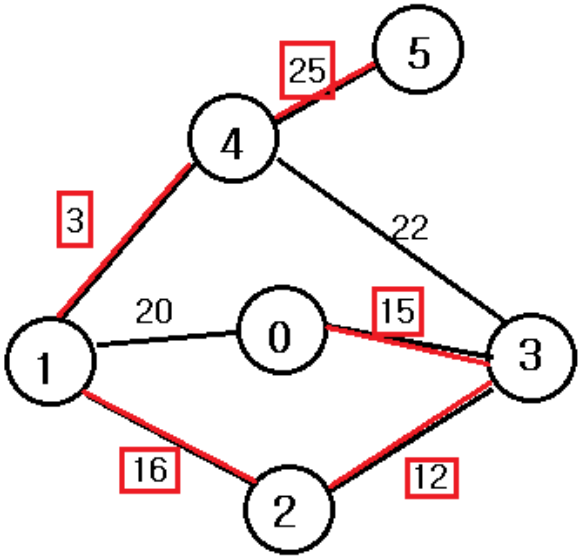
- 실행 결과





아래 test case는 입력에 해당하는 그래프의 그림과 함께 나타내었다. 빨간색으로 표시된 edge는 kruskal 알고리즘에 의해 선택된 edge들이다.




5
25
4
3
22
20
0
15
1
3
16
12
2