```c

/**
 * Implementation of Kruskal's algorithm to build an MST
 * Time Complexity: O(|E|.log|V|)
 *
 * Author: Mithusayel Murmu
 */

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

#define GRAPH_SZ 50
typedef enum { FALSE, TRUE } BOOL;

/** Disjoint-Set implementation. Uses rank and path compression */
typedef struct _DjointSet DjointSet;
typedef struct _DjointNode DjointNode;

struct _DjointSet {
    size_t size;    // Total disjoint sets available
    DjointNode *sets[GRAPH_SZ];
};
struct _DjointNode { int rank, key; DjointNode *parent; };

/* Creates a single element set with the given key {key} */
DjointNode * djoint_create_set(DjointSet *dset, const int key) {
    if (dset->size >= GRAPH_SZ) return NULL;

    DjointNode *node = (DjointNode*) malloc(sizeof(DjointNode));
    node->key = key; node->rank = 0;
    node->parent = node;
    // Insert the node as a set
    dset->sets[dset->size++] = node;

    return node;
}

/* Uses path compression for faster lookups */
DjointNode * djoint_find(DjointSet *dset, DjointNode *dnode) {
    if (dnode->parent != dnode) {
        // Flatten tree nodes
        dnode->parent = djoint_find(dset, dnode->parent);
    }
    return dnode->parent;
}

/* Uses union by rank to keep tree height short */
void djoint_union_set(DjointSet *dset, DjointNode *dnode1, DjointNode *dnode2) {
    DjointNode *p1 = djoint_find(dset, dnode1);
    DjointNode *p2 = djoint_find(dset, dnode2);
    if (p1 == p2) return;

    if (p1->rank < p2->rank) p1->parent = p2;
    else if (p2->rank < p1->rank) p2->parent = p1;
    else { p2->parent = p1; p1->rank++; }
}

DjointSet * djoint_create() {
    DjointSet *dset = (DjointSet*) malloc(sizeof(DjointSet));
    dset->size = 0; return dset;
}

void djoint_destroy(DjointSet *dset) {
    size_t i;
    for (i = 0; i < dset->size; ++i) free(dset->sets[i]);
    free(dset);
}

/** Rudimentary Graph definition */
typedef struct _Graph Graph;
typedef struct _GraphNode GraphNode;
typedef struct _GraphEdge GraphEdge;

/* Graph node indexes and weight */
struct _GraphEdge { int u, v, weight; };
struct _GraphNode { DjointNode *dnode_ref; };
struct _Graph {
    size_t vsize, esize;
    GraphNode nodeList[GRAPH_SZ];
    /* Maximum of |V|^2 edges */
    GraphEdge edgeList[GRAPH_SZ * GRAPH_SZ];
};

Graph * graph_create() {
    Graph *graph = (Graph *) malloc(sizeof(Graph));
    graph->vsize = graph->esize = 0;
    return graph;
}

#define _scand(n) scanf("%d", &(n))
void graph_input(Graph *graph) {
    int vs, asz, vi, i, j;

    _scand(vs); graph->vsize = vs;          // Number of vertices
    for (i = 0; i < vs; ++i) {
        _scand(asz);                        // Adjacency list size
        GraphNode node = { .dnode_ref = NULL };
        GraphEdge edge = { .u = i };

        for (j = 0; j < asz; ++j) {
            _scand(vi); edge.v = vi;        // Scan and set adjacent node's ID
            _scand(vi); edge.weight = vi;   // Scan and set edge weight for the adjacent node
            graph->edgeList[graph->esize++] = edge;
        }

        graph->nodeList[i] = node;
    }
}

/* Non-Decreasing comparator for two edges */
int graph_edge_comp(const void *edge1, const void *edge2) {
    return ((GraphEdge*)edge1)->weight - ((GraphEdge*)edge2)->weight;
}

/** Kruskal's algo implementation */
void kruskal_print_mst(Graph *graph, void (*callback)(GraphEdge)) {
    DjointSet *dset = djoint_create();

    size_t i;
    /* Create disjoint-set forest of vertices */
    for (i = 0; i < graph->vsize; ++i) {
        // Keep a reference to corresponding node in dis-joint set
        graph->nodeList[i].dnode_ref = djoint_create_set(dset, i);
    }

    /* Sort edge list in non-decreasing order. O(|E|.log|E|) */
    qsort(graph->edgeList, graph->esize, sizeof(GraphEdge), graph_edge_comp);

    /* Traverse through the edge list */
    for (i = 0; i < graph->esize; ++i) {
        GraphEdge edge = graph->edgeList[i];
        DjointNode *n1 = graph->nodeList[edge.u].dnode_ref;
        DjointNode *n2 = graph->nodeList[edge.v].dnode_ref;

        if (djoint_find(dset, n1) != djoint_find(dset, n2)) {
            /* Nodes don't belong to the same set, perform union */
            djoint_union_set(dset, n1, n2);
            callback(edge);
        }
    }

    djoint_destroy(dset);
}

static void print_utility(GraphEdge edge) { printf("(%d -> %d) ", edge.u, edge.v); }

/** Driver function */
int main(int argc, char const *argv[]) {
    Graph *graph = graph_create();

    printf("Enter graph data:\n");
    graph_input(graph);

    printf("\nMST result:\n");
    kruskal_print_mst(graph, print_utility);
    printf("\n"); free(graph);

    return 0;
}

```