Kruskal (MST): Really Special Subtree

Given an undirected weighted connected graph, find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and:

- There is only one exclusive path from a node to every other node.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- No cycles are formed

To create the Really Special SubTree, always pick the edge with smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

- Choose the edge that minimizes the sum $u + v + wt$ where u and v are vertices and wt is the edge weight.
- If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules.

For example, given the following edges:

u	v	wt
1	2	2
2	3	3
3	1	5

First choose $1 \rightarrow 2$ at weight 2. Next choose $2 \rightarrow 3$ at weight 3. All nodes are connected without cycles for a total weight of $3 + 2 = 5$.

Function Description

Complete the ***kruskals*** function in the editor below. It should return an integer that represents the total weight of the subtree formed.

kruskals has the following parameters:

- *g_nodes*: an integer that represents the number of nodes in the tree
- *g_from*: an array of integers that represent beginning edge node numbers
- *g_to*: an array of integers that represent ending edge node numbers
- *g_weight*: an array of integers that represent the weights of each edge

Input Format

The first line has two space-separated integers g_nodes and g_edges, the number of nodes and edges in the graph.

The next g_edges lines each consist of three space-separated integers g_from, g_to and g_weight, where g_from and g_to denote the two nodes between which the undirected edge exists and g_weight denotes the weight of that edge.

Constraints

- $2 \leq g\_nodes \leq 3000$
- $1 \leq g\_edges \leq \frac{N*(N-1)}{2}$
- $1 \leq g\_from, g\_to \leq N$
- $0 \leq g\_weight \leq 10^5$

**Note: ** If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

Output Format

Print a single integer denoting the total weight of the Really Special SubTree.

Current Buffer (saved locally, editable) C

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'kruskals' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts WEIGHTED_INTEGER_GRAPH g as parameter.
 */

/*
 * For the weighted graph, <name>:
 *
 * 1. The number of nodes is <name>_nodes.
 * 2. The number of edges is <name>_edges.
 * 3. An edge exists between <name>_from[i] and <name>_to[i]. The weight of the edge is <name>_weight[i].
 *
 */

int kruskals(int g_nodes, int g_edges, int* g_from, int* g_to, int* g_weight) {

}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char** g_nodes_edges = split_string(rtrim(readline()));

    int g_nodes = parse_int(*(g_nodes_edges + 0));
    int g_edges = parse_int(*(g_nodes_edges + 1));

    int* g_from = malloc(g_edges * sizeof(int));
    int* g_to = malloc(g_edges * sizeof(int));
    int* g_weight = malloc(g_edges * sizeof(int));

    for (int i = 0; i < g_edges; i++) {
        char** g_from_to_weight = split_string(ltrim(readline()));

        int g_from_temp = parse_int(*(g_from_to_weight + 0));
        int g_to_temp = parse_int(*(g_from_to_weight + 1));
        int g_weight_temp = parse_int(*(g_from_to_weight + 2));

        *(g_from + i) = g_from_temp;
        *(g_to + i) = g_to_temp;
        *(g_weight + i) = g_weight_temp;
    }

    int res = kruskals(g_nodes, g_edges, g_from, g_to, g_weight);

    // Write your code here.

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
```

```
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code