CSCI-550 **Project 2**

Minimum Bottleneck Spanning Tree

Galaxy trader Joe has to visit N planets that promise high revenue. He has not upgraded his space ship, so he must be careful how he chooses his route. He can recharge fuel at every planet, but his ship's load capacity is limited: the more fuel he will take, the fewer goods he can carry. That is why Joe wants to choose the route that minimizes the total fuel that he must carry between any two planets (and of course, amount of fuel needed depends on the distance between two planets). Returning to the already visited place is not a problem for Joe since he always has new goods to offer when he comes back to the visited planet. Help Joe to find the length of the longest distance between two planets that he will ever have to travel.

Asteroid fields prevent to travel between certain pairs of planets, so Joe must choose only between accessible routes.

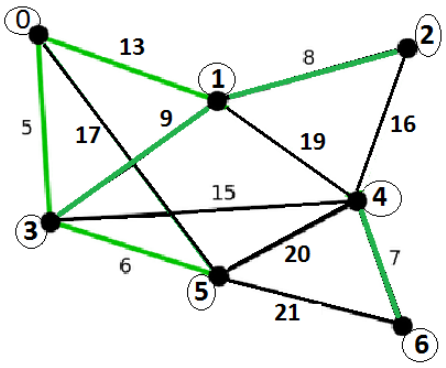
Input:

The first line contains two integers: N and M. N is the number of planets, and M is the number of accessible routs (undirected edges).
The following M lines have three integers: ***x***, ***y*** and ***w***, where ***x*** and ***y*** are planets' IDs (0, 1, 2,…, N-1) such that there is an accessible route between ***x*** and ***y***, and ***w*** is the length of the route in space miles.

Output:

A single integer ***L***, the length of the longest route between a pair of planets that Joe must travel printed in the format:
<L><endl>

Sample Input:




```
7 12
0 1 13
0 3 5
0 5 17
1 2 8
1 3 9
1 4 19
2 4 16
3 4 15
3 5 6
4 5 20
4 6 7
5 6 21
```

Sample Output:
15

Specification Requirements:

You may assume that all weights are distinct.

1. Your algorithm must run in O(M)-time. This means that you need to use adjacency lists for undirected graph: if an edge (u, v) appears in Input, then Adj[u] will have edge (u, v), and Adj[v] will have edge (v, u).

2. Use vector< list< edge > > data structure to implement an adjacency lists. You can use a ***class "edge"*** or a ***struct "edge"***, where edge (u,v) must hold at least these four values: v, w, u_original, v_original. If Adj[u] contains edge(v, w, u_original, v_original), that means there is an edge from u to v with weight w (length of the route), and u_original and v_original are the original names of this edge. In this way, you will practice using *list* from standard library.

3. I will provide some code that is supposed to save you time to program Project 2 so that it runs in O(M) time, where M is the number of edges. For example, I will provide function
int select (vector<int> &v, int start, int fin, int k)
given a vector of weights (integers, lengths of edges), and given the middle index ***k***, this function calculates and returns the k-th weight when considering weights in increasing order. Here, ***start*** and ***fin*** are the starting and last indices of the current range of integers inside vector ***v***.
For example, given {7, 8, 5, 3, 9} (not sorted integers), this algorithm will return 7, the midean of integers taken in increasing order. This algorithm runs in linear time (not in Nlog(N)-time). When given even-number of integers, {7, 8, 5, 3, 9, 2}, the median will be chosen as the greatest integer in the first half, i.e. 5, in this case.
To pass ***k*** as a parameter, please use this code:

```
if(v.size() % 2 == 0){
        k = (v.size() / 2 ) - 1;
}
else
        k = v.size() / 2;
```

4. I might include some skeleton code for the recursive function ***mbst***, it is up to you whether to use it or not.

5. It is up to you whether to include comments or not.

Grading Rubric:
1. Your grade will depend on how many tests your program passes: there are 5 tests, each is worth 60 points. Total points available for this project is 300pts.

2. There is up to 300pts of Extra Credit available for this project (60pts per test).

Submission:
Submit a single file ***proj2.cpp*** to ***turnin*** system.