**Algorithmic solution for P1:**

Read inputs from file

Number of vertices: numV

Number of edges: numE

Vertex 1: v1

Vertex 2: v2

Weight: w

//Build Data Structure and set targer

For ( int i = 0; i < numE; i++){

Vertex 1s[] = v1;

Vertex 2s[] = v2;

Weight [] = w;

If (w<0){

Target = v1;

}

}

//Determine if negative cycle exists

Int [] dist;

For (int I = 0; i < numV; i++){

Dist[i] = Infinity;

//Initialize all distances to infinity

}

for (int i = 0; i <= numV; i++) {

for (int j = 0; j < Vertex 1s[].length; j++) {

int u = Vertex 1s[j];

int v = Vertex 2s[j];

//This is going to go through the entire cycle containing the target and come back to the target

if (dist[u] != INFINITY AND dist[u] + w < dist[v]){

dist[v] = dist[u] + w;

}

}

}

//Now after the cycle is complete, if there is any negative cycle exist, the target should have a negative value. If it has a positive value, that means there is no negative loop exist

If (dist[target] < 0)

Print(YES)

else

Print(NO)

**Algorithmic solution for P2:**

//By a set, I mean (u,v) from input

Read inputs

ReachSet = {0};

#F number of FE

UnReachSet = {1, 2, ..., #F};

SpanningTree = {};

//Find disconnection and add it to F

list L = v

while L nonempty

for (all vertices in F){

visit v

if disconnecting detected

mark w

add it to end of list

add edge vw to UnReachSet

}

//Calculating the minimum spanning tree

For (every vertex v ){

Use BFS to see if the vertices are disconnected from the rest of F

If (Indeed disconnected){

Find the closest path using BFS from either end of the edge to the closest v

UnReachSet.add( set )

}

}

Find

while ( UnReachSet ≠ empty )

{

Find edge e = (x, y) such that:

1. x ∈ ReachSet

2. y ∈ UnReachSet

3. e has smallest cost

SpanningTree = SpanningTree ∪ {e};

ReachSet = ReachSet ∪ {y};

UnReachSet = UnReachSet - {y};

}