# General Problem Description

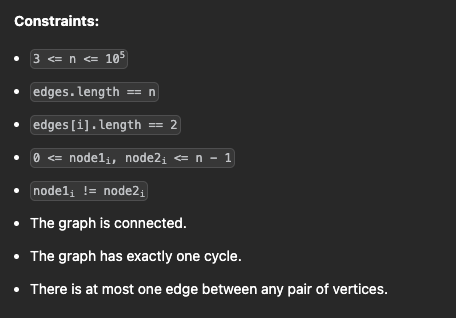
Leetcode 2204

You are given a positive integer n representing the number of nodes in a connected undirected graph containing exactly one cycle. The nodes are numbered from 0 to n - 1 (inclusive).

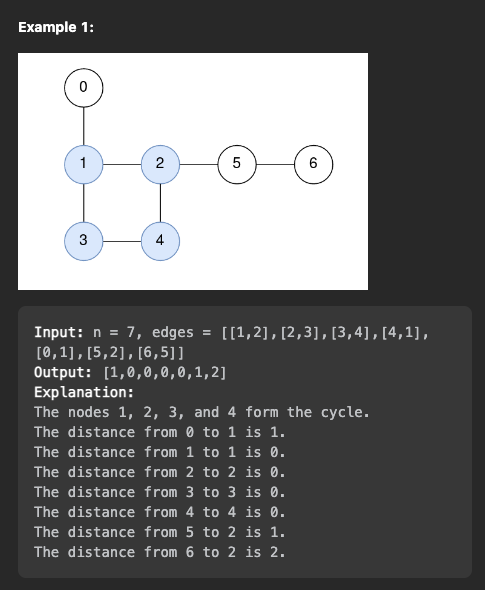
You are also given a 2D integer array of edges, where edges[i] = [node1\_i , node2\_i] denotes that there is a bidirectional edge connecting node1\_i and node2\_i in the graph.

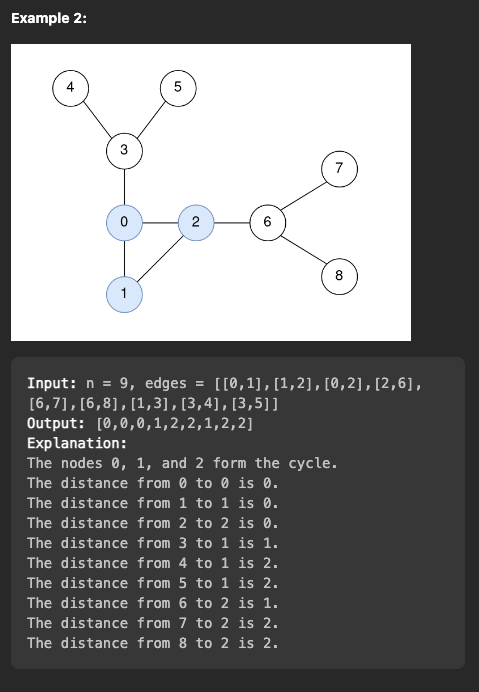
The distance between two nodes a and b is defined to be the minimum number of edges that are needed to go from a to b.

Return an integer array answer of size n, where answer[i] is the minimum distance between the ith node and any node in the cycle.



# Sample Input





# Proposed Algorithm

## Description:

Step 1: find all nodes in cycle, put in list? Use some version of kruskal’s

Step 2: adjacent list of nodes. breadth first search on the cycle nodes.

Step 3: breadth first search using cycle to apply a level to every other node

## Edge Cases:

Input: root = []

Output: -1

## Test Cases:

Input: n=3, edges=[[0,1],[1,2],[0,2]]

Output: [0,0,0]

Input: n = 4, edges = [[0,2], [1,2], [0,1], [0,3]]

Output: [0, 0, 0, 1]

Input: n = 9, edges = [[0,1],[1,2],[0,2],[2,6],[6,7],[6,8],[1,3],[3,4],[3,5]]

Output: [0,0,0,1,2,2,1,2,2]

## Time Complexity:

O(Edges.size())

## Space Complexity:

O(Vertices.size())

# Implementation of Algorithm (simple pseudo code, list of functions definitions etc)

def modified\_kruskals (vector<vector<int>> edges) {

// complexity is O(|E| lg |E|) or O(|E| lg |V|) if using binary heap for PriorityQueue

for ( auto v : g.vertices() ) {

vertex.setID = Set::makeSet();

}

int N = g.vertices().size();

Vector<Edges> T;

while ( T.size() < N-1 ) {

Edge e = Q.extractMin();

if ( Set::findSet(e.u.setID) == Set::findSet(e.v.setID) ) { // i.e., in the same set

return findSet(e.u.setID) // add edge e to solution

}

}

}

def adjacentNodes(edges){

vector adjacentNodes; // node -> list of adjacent nodes

for edge in edges{

edge[0].append(edge[1])

edge[1].append(edge[0])

}

return adjacentNodes

}

cycleNodes = modified\_kruskals(edges)

adjacentNodes = adjacentNodes(edges)

queue = cycleNodes

res = [-1] \* n

lvl = 0

while queue{

newQ = new queue()

while queue{

e = popleft(queue)

newQ.append(adjacentNodes[e])

res[e] = lvl

}

lvl += 1

queue = newQ

}

return res

# Advantages/Disadvantages of Your Algorithm and Any Other Comments

Advantages: relatively quicker runtime as opposed to other solutions

Disadvantages: we need extra space to store information, like adjacent nodes, nodes in the cycle.