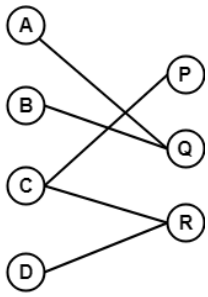
Data Structures and Algorithms (CS F211)
Second Semester 2018-2019
Lab Sheet 2

1. Given a weighted, directed graph, write a C program to find the cycle having minimum weight. Assume that the graph is given as user input in the form of an adjacency matrix, contains no self-loops and edge weights are positive whole numbers greater than zero. The number of nodes and edges can be taken as user inputs. The graph can contain a maximum of 1000 nodes. Your program should be able to identify all of possible minimum weight cycles in the graph. In case, the graph contains no cycle, your program should print the message "No cycles present".

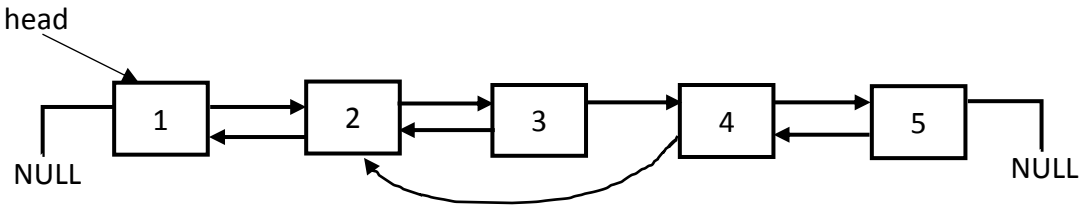
2. The diameter of a graph is defined as $\max_{u, v} d(u, v)$ between any two vertices (u, v), where d(u,v) is the shortest path between u and v. To find the diameter of a graph, you have to first find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph. Given an undirected, unweighted graph, write a C program to find the diameter of the graph. The graph is given as user input in the form of an adjacency matrix. The number of nodes and edges can be taken as user inputs. The graph can contain cycles, self-loops and a maximum of 1000 nodes. In addition to printing the diameter, your program should also print all shortest paths (in case there are more than one) between every pair of vertices. Note that for an unweighted graph, the length of the shortest path between a pair of vertices is the total number of edges lying in that path.

3. A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V. The diagram below shows an example bipartite graph.

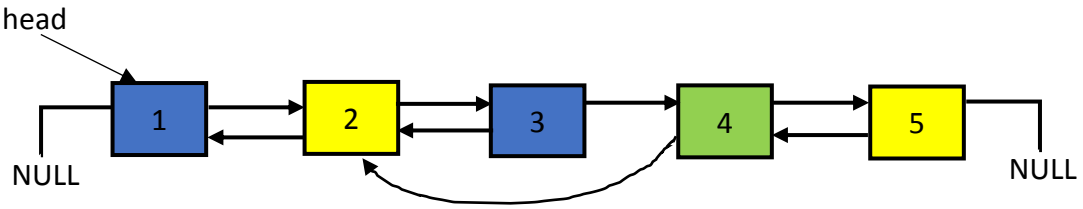


Given an undirected, unweighted graph, write a C program to determine whether the graph is bipartite or not. The graph is given as user input in the form of an adjacency matrix. The number of vertices and edges can be taken as user input. The graph can contain a maximum of 1000 nodes. Note that your program should be able to tell if the graph is bipartite or not.

4. Suppose you are given a doubly linked list. Each node has an id, a next pointer and a previous pointer. The previous pointer of a node **X** can either point to the immediately preceding node (case 1) or a node lying beyond the immediately preceding node (case 2). The list can contain one or more loops, i.e., the previous pointer of one or more nodes (except for the first node) can point to a node lying beyond the immediately preceding node. The loop never involves previous pointers satisfying case 1. However, the next pointer of every node points to the immediately next neighbouring node except for the last node whose next pointer is NULL. The previous pointer of the first node is always NULL. There is a head pointer pointing to the first node of the list. There is no tail pointer. The diagram below shows an example of such a list. The number inside each node indicates the node id.



Given such a list as user input, write a C program to determine whether the list contains any loops or not. If yes, print all such loops. Irrespective of whether the list contains loops or not, finally, assign colours to each of the nodes in such a manner that no two adjacent nodes are assigned the same colour and the colouring is done using minimum number of colours. While determining if two nodes are adjacent or not, you will have to consider both next and previous pointers (satisfying both case 1 and case 2).
For the linked list shown above, the output will be:
A loop exists 2 → 3 → 4 → 2 (Here, 2 → 3 → 2, 2 → 3 → 4 → 5 → 4 → 2 etc. are not considered as loops).
The list can be coloured using 3 colours.
A sample colouring is shown below.

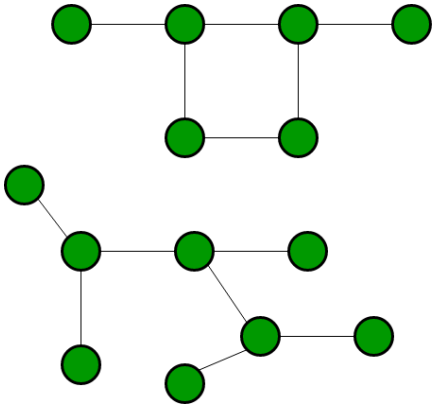


The number of colours as well as the colours assigned to the individual nodes should be printed.
The linked list is given as user input. However, the total number of nodes cannot be taken as user input. The input will be given in the following tuple format (X, Y, Z):

- X - id of the new node
- Y - id of the node where the previous pointer of X will point
- Z - id of the node where the next pointer of X will point

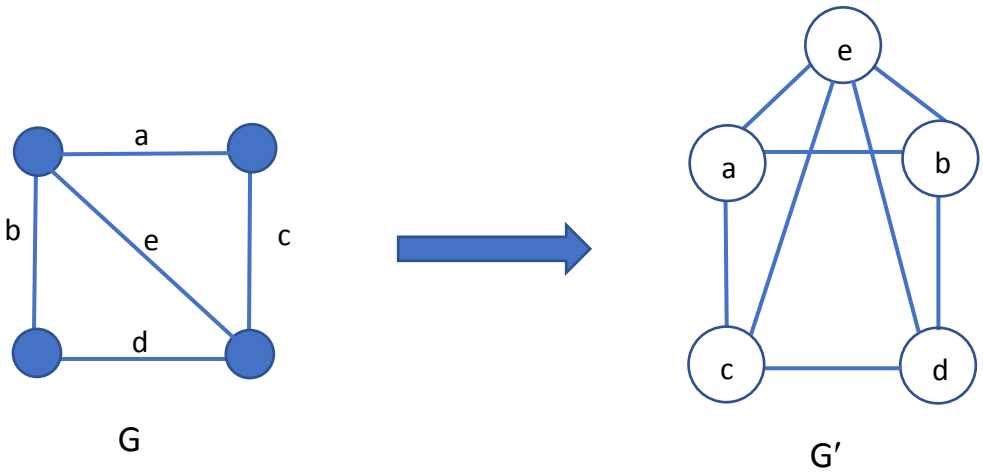
For the 1st node, the input will be (1, 0, 2) where 0 indicates the previous pointer of the 1st node is made NULL. Similar logic applies for the last node.
You can assume any colours for colouring the nodes. For a given linked list, several colourings can be possible using minimum number of colours. For eg., for the linked list shown above, the colour sequence could have been blue, yellow, blue, green, blue. It is sufficient to print only one such colouring sequence.

5. Suppose you are given a set of vertices V such that a factor f is associated with each vertex. For each vertex $u \in V$, f(u) denotes its factor. An unweighted, undirected graph G can be constructed using the vertices of V according to the following construction algorithm – for a pair of vertices, $u, v \in V$, the edge (u, v) is included in G, if $|f(u) - f(v)| \geq \delta$ where δ is a pre-defined threshold value. If the threshold condition is not satisfied, the edge is not included in G. Note that here, the absolute value of the difference between f(u) and f(v) have been considered. Given V and δ, write a C program to construct the graph G as per the above-mentioned algorithm. After graph construction, your program should determine the different connected components of G. Note that a connected component of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which are connected to no additional vertices in the supergraph. The number of connected components in the example graph shown below is 2.



The vertex set V and δ are given as user inputs. $|V| \leq 1000$, f and δ are always integers. You may take the size of V as user input also. Your program should determine the graph, the number of connected components and the actual components. Display the output graph and each connected component as separate adjacency matrices.

6. Suppose you are given an unweighted, undirected graph G, where V denotes the set of vertices and E denotes the set of edges. You have to construct an unweighted, undirected graph G′ from G in the following manner. For every edge in G, there is a vertex in G′. A pair of vertices u, v ∈ G′ are connected by an edge if the edges corresponding to u, v were incident (connected to) on the same vertex in G. The method of construction is illustrated in the diagram below.



After constructing G′, you have to determine if it can be split into at least two connected components by removing one single vertex and the edges incident on it. If yes, then identify that vertex and all the resulting connected components. If no (in case there exists no such vertex or can only be done using more than one vertex), then simply display the message "No such vertex exists". Write a C program for the above scenario. The graph is given as user input as an adjacency matrix. The number of vertices and edges may also be taken as user inputs. The graph can have a maximum of 1000 nodes.

7. There are ***N*** cities with ids ***{1, 2, 3, 4, … , N}*** and ***N-1*** roads connecting these cities. All roads are bi-directional in nature. Abhishek lives in City 1 so this can be considered as the source. There are ***M*** universities in various cities (there is no university in City 1). Abhishek gets accepted in all the universities, however, he will take admission in the university which is at minimum distance from his city. If two or more cities are at the same minimum distance, then he will accept the university which is located in the city with minimum id. No two universities are in the same city.

Input: First line consists of ***N***, i.e. number of cities. Next ***N-1*** lines follow the type ***u, v*** which denotes there is a road between ***u*** and ***v***. Next line consists of ***M***. Next ***M*** lines indicate ids of cities where universities are located.

Output: Print the id of the city with the university which Abhishek will accept.

Constraints: *2 <= **N** <= 100, 1 <= **u, v** <= **N**, 1 <= **M** <= (**N**-1)*

8. Given an undirected graph, find the total number of nodes connected at ***t*** distance away from each other. For example: Two nodes directly connected are at 1 distance connectivity. While the two nodes having a common node between each other without having direct connectivity are at 2 distance connectivity.

First line of input contains ***n*** (number of nodes) and second line of input contains ***e*** (number of edges). Next ***e*** lines will contain two integers ***u*** and ***v*** meaning that node ***u*** and ***v*** are connected to each other in undirected fashion. Next line contains the input ***t***. The output should display the total number of nodes connected at ***t*** distance away from each other.

9. Let us denote SumOfMatrix as a sum of all the elements of the matrix. Given a square matrix ***A*** of size ***n x n***, find the maximum of SumOfMatrix of all the square sub-matrices of ***A*** with size ***m x m*** where ***m <= n <= 100***. The elements of ***A*** lie in the range -1000 to 1000. ***n*** and the elements of the matrix are given as user input.

Hint - The graph problems are likely to be solved using BFS or DFS. Think about it!