// На основе обхода списка узлов найти узел с заданным значением в графе и исключить его.

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

// A structure to represent an adjacency list node

struct AdjListNode

{

int dest;

struct AdjListNode\* next;

};

// A structure to represent an adjacency list

struct AdjList

{

struct AdjListNode\* head;

};

// A structure to represent a graph. A graph is an array of adjacency lists.

// Size of array will be V (number of vertices in graph)

struct Graph

{

int V;

struct AdjList\* array;

};

// A utility function to create a new adjacency list node

struct AdjListNode\* newAdjListNode(int dest)

{

struct AdjListNode\* newNode =(struct AdjListNode\*) malloc(sizeof(struct AdjListNode));

newNode->dest = dest;

return newNode;

}

// A utility function that creates a graph of V vertices

struct Graph\* createGraph(int V)

{

struct Graph\* graph =(struct Graph\*) malloc(sizeof(struct Graph));

graph->V = V;

// Create an array of adjacency lists. Size of array will be V

graph->array =(struct AdjList\*) malloc(V \* sizeof(struct AdjList));

// Initialize each adjacency list as empty by making head as NULL

int i;

for (i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Adds an edge to an undirected graph

void addEdge(struct Graph\* graph, int src, int dest, int h)

{

// Add an edge from src to dest. A new node is added to the adjacency list of src. The node is added at the beginning

struct AdjListNode\* newNode = newAdjListNode(dest);

if (src != h && dest != h)

{

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

}

// Since graph is undirected, add an edge from dest to src also

newNode = newAdjListNode(src);

if (src != h && dest != h)

{

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

}

// A utility function to print the adjacency list representation of graph

void printGraph(struct Graph\* graph)

{

int v;

for (v = 1; v <= graph->V; ++v)

{

struct AdjListNode\* pCrawl = graph->array[v].head;

printf("\n Список смежности вершины %d\n head ", v);

while (pCrawl)

{

printf("-> %d", pCrawl->dest);

pCrawl = pCrawl->next;

}

printf("\n");

}

}

// Driver program to test above functions

int main()

{

setlocale(LC\_ALL, "russian");

// create the graph given in above fugure

int V,k,h;

int dest, src;

cout << "кол-во вершин: ";

cin >> V;

struct Graph\* graph = createGraph(V);

cout << "кол-во связей: ";

cin >> k;

cout << "Исключаемая вершина: ";

cin >> h;

for (int i = 0; i < k; i++)

{

cout << "Введите вершину ";

cin >> src;

cout << "Введите вершину с которой она связана ";

cin >> dest;

addEdge(graph, src, dest,h); //Adding an edge

}

// print the adjacency list representation of the above graph

printGraph(graph);

return 0;

}