Task 4: Graph Edge Addition Validation

Given a directed graph, write a function that adds an edge between two nodes and then checks if the graph still has no cycles. If a cycle is created, the edge should not be added.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_NODES 100

// Structure to represent a node in the adjacency list
struct Node {
    int dest;
    struct Node* next;
};

// Structure to represent the adjacency list of a graph
```

```
struct AdjList {
    struct Node* head;
};

// Structure to represent the graph
struct Graph {
    int numVertices;
    struct AdjList* array;
};

// Create a new node
struct Node* createNode(int dest) {
    struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// Create a graph with a given number of vertices
```

```
struct Graph* createGraph(int numVertices) {
    struct Graph* graph = (struct
Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;
    graph->array = (struct AdjList*)malloc(numVertices *
sizeof(struct AdjList));
    for (int i = 0; i < numVertices; ++i)
        graph->array[i].head = NULL;
    return graph;
}

// Add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    struct Node* newNode = createNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
}

// Utility function to perform Depth-First Search (DFS)
bool isCyclicUtil(int v, bool visited[], bool recStack[],
struct Graph* graph) {
```

```
    if (visited[v] == false) {
        // Mark the current node as visited and add it to the recursion stack
        visited[v] = true;
        recStack[v] = true;

        struct Node* adjNode = graph->array[v].head;
        while (adjNode != NULL) {
            int dest = adjNode->dest;
            if (!visited[dest] && isCyclicUtil(dest, visited, recStack, graph))
                return true;
            else if (recStack[dest])
                return true;
            adjNode = adjNode->next;
        }
    }
    // Remove the vertex from recursion stack
    recStack[v] = false;
    return false;
}
```

```
// Function to check if adding an edge between two nodes creates a cycle
bool isCyclic(struct Graph* graph, int src, int dest) {
    // Add the edge to the graph
    addEdge(graph, src, dest);

    // Initialize arrays to keep track of visited vertices and recursion stack
    bool* visited = (bool*)malloc(graph->numVertices * sizeof(bool));
    bool* recStack = (bool*)malloc(graph->numVertices * sizeof(bool));
    for (int i = 0; i < graph->numVertices; i++) {
        visited[i] = false;
        recStack[i] = false;
    }

    // Perform Depth-First Search (DFS) to check for cycles
    for (int i = 0; i < graph->numVertices; i++)
```

```
        if (isCyclicUtil(i, visited, recStack, graph))
            return true;

    // Remove the edge from the graph as it does not
create a cycle
    struct Node* temp = graph->array[src].head;
    graph->array[src].head = temp->next;
    free(temp);

    return false;
}

int main() {
    // Create a graph with 4 vertices
    struct Graph* graph = createGraph(4);

    // Add edges to the graph
    addEdge(graph, 0, 1);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
```

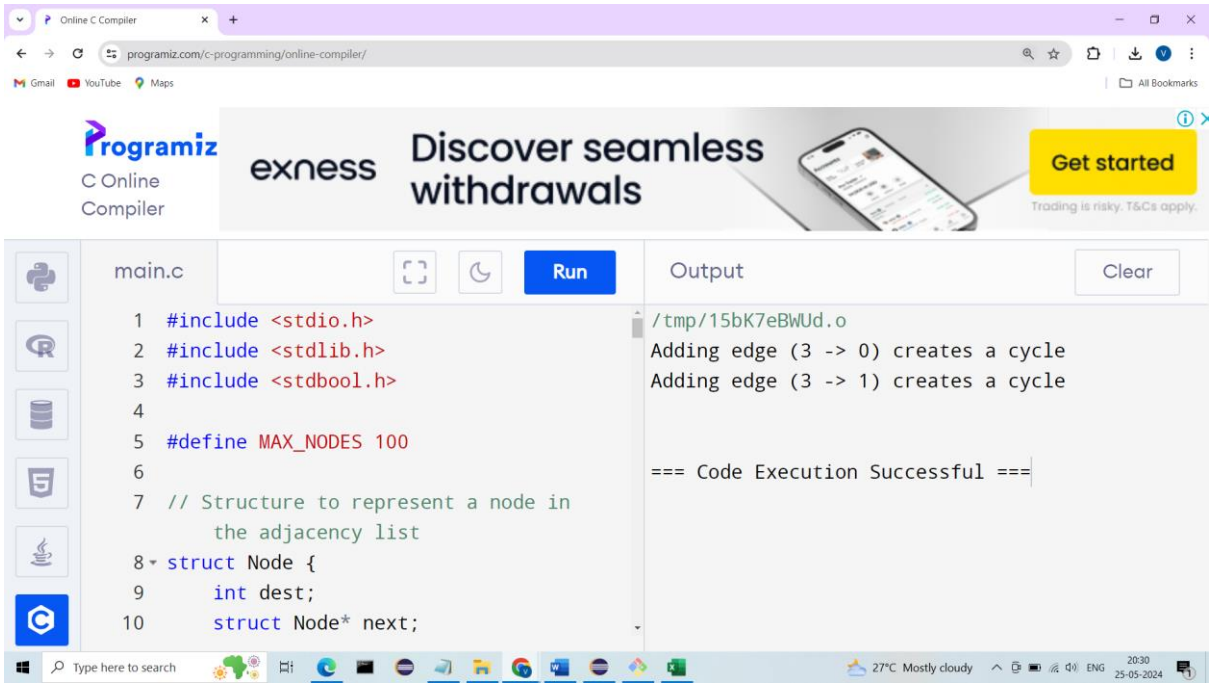
```
    // Check if adding an edge between two nodes creates a cycle
    int src = 3, dest = 0;
    if (isCyclic(graph, src, dest))
        printf("Adding edge (%d -> %d) creates a cycle\n", src, dest);
    else
        printf("Adding edge (%d -> %d) does not create a cycle\n", src, dest);

    // Adding an edge that creates a cycle
    src = 3, dest = 1;
    if (isCyclic(graph, src, dest))
        printf("Adding edge (%d -> %d) creates a cycle\n", src, dest);
    else
        printf("Adding edge (%d -> %d) does not create a cycle\n", src, dest);

    return 0;
```

}

Output: -


Online C Compiler
programiz.com/c-programming/online-compiler/
Gmail
YouTube
Maps
All Bookmarks
Programiz
C Online
Compiler
exness
Discover seamless withdrawals
Get started
Trading is risky. T&Cs apply.
main.c
Run
Output
Clear
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 #define MAX_NODES 100
6
7 // Structure to represent a node in the adjacency list
8 struct Node {
9 int dest;
10 struct Node* next;
/tmp/15bK7eBWUd.o
Adding edge (3 -> 0) creates a cycle
Adding edge (3 -> 1) creates a cycle
=== Code Execution Successful ===
Type here to search
27°C Mostly cloudy
ENG
20:30
25-05-2024