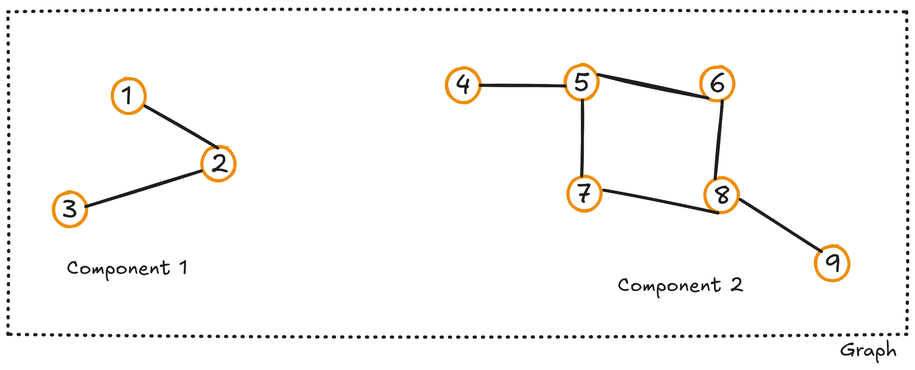
Cycle detection in undirected Graphs using BFS



Graph

Adj List

1 -> 2
2 -> 3
3 -> 2
4 -> 5
5 -> 4,6,7
6 -> 5,8
7 -> 5,8
8 -> 6,7,9
9 -> 8

Visited

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->

Parent

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->
9 ->

Queue

```cpp
#include<unordered_map>
#include<queue>
#include<list>
using namespace std;


bool isCyclicBFS( int src, unordered_map< int, bool > &visited,  unordered_map<int, list<int>> &adj){

    unordered_map<int,int> parent;

    parent[src] = -1; // Initially we took the parent as - 1 of the source node
    visited[src] = 1;
    queue<int> q;
    q.push(src);

    while(!q.empty()){
        int front = q.front();
        q.pop();

        for( auto neighbour : adj[front]){
            if( visited[neighbour] == true && neighbour != parent[front] ){
                return true;
            }
            else if(!visited[neighbour]){
                q.push(neighbour);
                visited[neighbour] = 1;
                parent[neighbour] = front;
            }
        }
    }

    return false;
}

string cycleDetection (vector<vector<int>>& edges, int n, int m)
{
    // creating the adjacency list
    unordered_map<int, list<int>> adj;
    for( int i = 0; i < m; i++){
        int u = edges[i][0];
        int v = edges[i][1];

        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    // make the visited array
    unordered_map< int, bool > visited;


    // to handle the disconnected components, we apply this for loop
    for( int i = 1; i <= n; i++){
        if(!visited[i]){
            bool ans = isCyclicBFS(i, visited, adj);
            if( ans == 1){
                return "Yes";
            }
        }
    }

    return "No";

}
```

For the source node, parent of the source node is -1, visited of that node is 1. We initialize a queue for that and we pushed that source node into it.

The loop will run till the queue is empty , take the front element of that q and pop it, Now for that front element, look at the neighbors of that, if the neighbor is not visited, push that neighbor into the queue, mark its visited as true and push it into the queue

This is the condition for cycle, just one line that if the particular neighbor of the front is visited but it is not the parent of front element, cycle is there.

We created the adjacency list first from the function and then we created the map which shows visited or not visited nodes

This for looping is done to handle the disconnected components, we will start with 1 and then will apply the cyclic bfs check in that, 1 will take care of the 1 2 3 connection via cyclic bfs function and return true if the cycle exists. When the queue of 1 gets empty for i = 2 and i =3 visited is true so they do not go inside for loop. As soon as i hits 4, again bfs is called and 4 5 6 7 8 9 manage themselves in bfs and when queue gets empty and i hits 5 , it does not go inside for loop as it is already marked as true in visited array. Similarly with 6 7 8 9.

This for loop is only to take care of the two components and that too specifically from source node 1 and source node 4.

Dry Run and the Code Logic

Adj List

1 -> 2
2 -> 3
3 -> 2
4 -> 5
5 -> 4,6,7
6 -> 5,8
7 -> 5,8
8 -> 6,7,9
9 -> 8

Visited

1 -> true
2 -> true
3 -> true
4 -> true
5 -> true
6 -> true
7 -> true
8 -> true

Parent

1 -> -1
2 -> 1
3 -> 2
4 -> -1
5 -> 4
6 -> 5
7 -> 5
8 -> 6
9 ->

Queue

8
7
6
5
4
3
2
1

- We first made the adjacency list with the information of edges, nodes and vertices and we also made the visited array

- Inside the for loop we started with i = 1, and the visited[1] DNE or is False, so we go ahead, we called cyclic BFS function on it, if it returns true, we return Yes.

- The call is for bfs(1), 1. Write its parent as -1 (source node) 2. Make it visited 3. push it into the queue.

- The queue is not empty , front is -1 , and now remove that 1, it has neighbor as 2, check cyclic condition , not go, it is not visited so, 1. push it into the queue , mark it visited and mark its parent as 1

- Queue is not empty, front is 2, pop it out , and remove that. Look at the neighbors of 2, it is 3, cyclic condition not followed, push that 3 into the queue, mark it visited and mark its parent.

- The queue is not empty it has 3, so now make store it as front and remove it, look at the neighbors, 2 , cycle condition not met and 2 is already visited, so the loop ends and queue has also become empty, it returns False.

- Now go to the main function, if is not executed as this does not had cycle, now for loop moves to i = 2, it is visited, so loop ends it goes to i = 3, visited , loop ends , Now it goes to 4, it is not visited and cyclicdfs(4) is called and now we are working on the second component of our graph

- df(4) is called , make the parent as -1, mark visited as true, push it into the queue.

- Now the queue has 4 in it, its not empty, so store 4 as front and pop out the 4, look at the adjacency of 4, it is 5

- 5 does not follow the cycle condition, it is not visited so it goes inside the loop, mark it visited , mark its parent and push it into the queue

- Queue is not empty it has 5 in it , pop that out and store that 5 in front, look at the neighbors of front it is 4 6 7 , 4 does not follow cyclic condition it is also marked true, so dont go inside the loop,

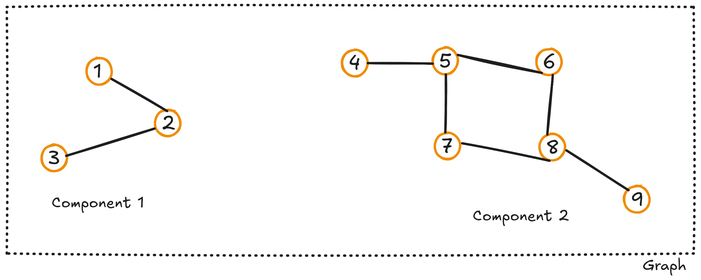
- 6 does not follow cyclic condition and is not visited , so make it visited mark the parent and push in into the queue

- 7 is another neighbor, does not follow the cyclic condition, and is not visited, so goes inside the loop, mark it visisted, mark its parent and push it into the queue.

- The queue is not empty , it has 6 and 7 in it , 6 is in front so pop it out and store it in the front, look at the neighbors of 6, it is 5 and 8, 5 does not follow cyclic condition and also visited so neglect it, for 8, it is not visited so mark it as visited, push it into the queue and mark its parents.

- The queue is not empty it has 7 and 8 in it , 7 is the front, pop it out and look at the neighbors of 7 it is 5 and 8, 5 does not follow the cyclic condition and is already visited so neglect it , 8 follows the cyclic condition, as it is visited and the parent of 8 is not 7, so return True. Loop ends.

Cycle detection in undirected Graphs using DFS



Graph

Adj List

1 -> 2
2 -> 3
3 -> 2
4 -> 5
5 -> 4,6,7
6 -> 5,8
7 -> 5,8
8 -> 6,7,9
9 -> 8

Visited

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->

Parent

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->
9 ->

```cpp
#include<unordered_map>
#include<queue>
#include<list>
using namespace std;

//TC : O(n)


bool isCyclicDFS( int node, int parent, unordered_map< int, bool > &visited,  unordered_map<int, list<int>> &adj){


    visited[node] = true;

    for( auto neighbor : adj[node]){
        if( !visited[neighbor]){
            bool cycleDetected = isCyclicDFS(neighbor, node, visited, adj);
            if(cycleDetected){
                return true;
            }
        }

        else if( neighbor != parent){
            //cycle present
            return true;
        }

    }

}

string cycleDetection (vector<vector<int>>& edges, int n, int m)
{
    // creating the adjacency list
    unordered_map<int, list<int>> adj;
    for( int i = 0; i < m; i++){
        int u = edges[i][0];
        int v = edges[i][1];

        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    // make the visited array
    unordered_map< int, bool > visited;

    // to handle the disconnected components, we apply this for loop
    for( int i = 1; i <= n; i++){
        if(!visited[i]){
            bool ans = isCyclicDFS(i, -1, visited, adj);
            if( ans == 1){
                return "Yes";
            }
        }
    }

    return "No";

}
```

continue to call for dfs if the neighbor is not visited and run the dfs for each neighbor, condition for the detection of the cycle is that the neighbor was visited which goes to else and the neighbor is not the parent node,

Condition for detection of cycle :
- for the particular node, neighbor was visited.
- neighbor is not the parent of the node.

make the adjacency list with the help of edges, nodes and vertices and also make the visited map

we are running for loop to incorporate the disconnected components of the graph, virtually this ensures that DFS cycle is checked on node 1 and node 4, as 1 2 3 are diconnectes from 4 5 6 7 8 9, so basically only 1 and 4 go inside the loop

Code Logic and the Dry Run

Adj List

1 -> 2
2 -> 3
3 -> 2
4 -> 5
5 -> 4,6,7
6 -> 5,8
7 -> 5,8
8 -> 6,7,9
9 -> 8

Visited

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->

Parent

1 ->
2 ->
3 ->
4 ->
5 ->
6 ->
7 ->
8 ->
9 ->

- Start for i = 1, 1 node is not visited so call the dfs for this function, df(1 , -1), this is source node that's why - 1 is the parent.

- df(1 , -1) is called, mark the 1 node as True, check for the neighbors of 1, which is 2, 2 is not visited so call df(2 , 1).

- df(2, 1) is called , mark the 2 node as True, check for the neighbor of 2 which is 1 and 3, 1 1 is visited and 1 is the parent of 2, so nothing happens now go to 3, 3 is not visited , so call df (3 , 2)

- df(3, 2) is called, mark the node 3 as true, neighbor of 3 is 2 , which is visited , 2 is the parent of 3 so nothing happens and df(3 ,2) returns to df(2,1) which returns to df (1,-1) and no cycle was detected.

- Main function loop goes for i = 2 , it is visited, so next is 3 which is also visited so next is 4.

- df (4, - 1) is called from the main function.

- df(4, -1) mark the 4 as true, and neighbor of 4 is 5 which is not visited, so call df(5, 4)

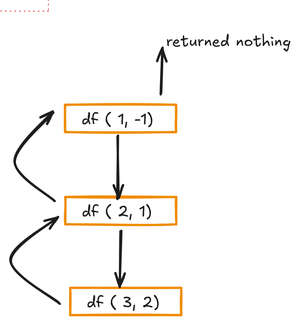
- df(5 , 4) mark the 5 as true, and neighbor of 5 is 4 6 7 . 4 is visited and 4 is parent of 5 , so nothing happens, df(6, 5) is called.

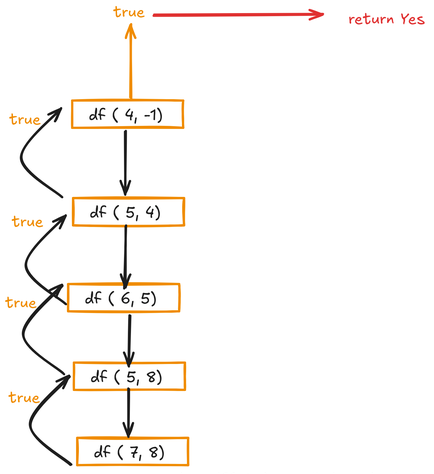
- df(6,5) is called , which marks 6 as true, neighbor of 6 is 5 and 8 , 5 is visited and is parent of 6 so nothing happens and dfs(8, 6) is called

- df(8,6) marks 8 as true and neighbor of 8 is 6 and 7, 6 is visted and is the parent of 8 so nothing happens, 7 is not visited df (7 , 8) is called

- df(7 , 8) is called, mark 7 as True and neighbor of 7 is 5 and 8 , 5 is visited, and now the 5 is not parent of 7 , so cycle exists which returns True

- This true propagates upward and the cycle is detected.





when df(7,8) is called, we had the neioghbor as 5 which was visited so the control went to the else if and it bacame true as 5 was not the parent of 7 but it was still visited , so this returned true.