Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)

A *disjoint-set data structure* is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. A *union-find algorithm* is an algorithm that performs two useful operations on such a data structure:

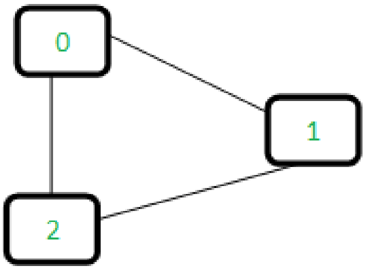
Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

Union: Join two subsets into a single subset.

In this post, we will discuss an application of Disjoint Set Data Structure. The application is to check whether a given graph contains a cycle or not.

Union-Find Algorithm can be used to check whether an undirected graph contains cycle or not. Note that we have discussed an algorithm to detect cycle. This is another method based on *Union-Find*. This method assumes that graph doesn't contain any self-loops.
We can keep track of the subsets in a 1D array, let's call it parent[].

Let us consider the following graph:



For each edge, make subsets using both the vertices of the edge. If both the vertices are in the same subset, a cycle is found.

Initially, all slots of parent array are initialized to -1 (means there is only one item in every subset).

```
0   1   2
-1 -1  -1
```

Now process all edges one by one.

Edge 0-1: Find the subsets in which vertices 0 and 1 are. Since they are in different subsets, we take the union of them. For taking the union, either make node 0 as parent of node 1 or vice-versa.

```
0   1   2    <----- 1 is made parent of 0 (1 is now representative of subset {0, 1})
1  -1  -1
```

Edge 1-2: 1 is in subset 1 and 2 is in subset 2. So, take union.

```
0   1   2    <----- 2 is made parent of 1 (2 is now representative of subset {0, 1, 2})
1   2  -1
```

Edge 0-2: 0 is in subset 2 and 2 is also in subset 2. Hence, including this edge forms a cycle.

How subset of 0 is same as 2?
0->1->2 // 1 is parent of 0 and 2 is parent of 1

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Based on the above explanation, below are implementations:

C/C++

```
// A union-find algorithm to detect cycle in a graph
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// a structure to represent an edge in the graph
struct Edge
{
    int src, dest;
};

// a structure to represent a graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph =
           (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge =
        (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

// A utility function to find the subset of an element i
int find(int parent[], int i)
{
    if (parent[i] == -1)
        return i;
    return find(parent, parent[i]);
}

// A utility function to do union of two subsets
void Union(int parent[], int x, int y)
{
    int xset = find(parent, x);
    int yset = find(parent, y);
    parent[xset] = yset;
}

// The main function to check whether a given graph contains
// cycle or not
int isCycle( struct Graph* graph )
{
    // Allocate memory for creating V subsets
    int *parent = (int*) malloc( graph->V * sizeof(int) );

    // Initialize all subsets as single element sets
    memset(parent, -1, sizeof(int) * graph->V);

    // Iterate through all edges of graph, find subset of both
    // vertices of every edge, if both subsets are same, then
    // there is cycle in graph.
    for(int i = 0; i < graph->E; ++i)
    {
        int x = find(parent, graph->edge[i].src);
        int y = find(parent, graph->edge[i].dest);

        if (x == y)
            return 1;

        Union(parent, x, y);
    }
    return 0;
}

// Driver program to test above functions
int main()
{
    /* Let us create the following graph
         0
        |  \
        |    \
        1-----2 */
    int V = 3, E = 3;
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;

    // add edge 1-2
    graph->edge[1].src = 1;
    graph->edge[1].dest = 2;

    // add edge 0-2
    graph->edge[2].src = 0;
    graph->edge[2].dest = 2;

    if (isCycle(graph))
        printf( "graph contains cycle" );
    else
        printf( "graph doesn't contain cycle" );

    return 0;
}
```

Run on IDE

Java

```
// Java Program for union-find algorithm to detect cycle in a graph
import java.util.*;
import java.lang.*;
import java.io.*;

class Graph
{
    int V, E;    // V-> no. of vertices & E->no.of edges
    Edge edge[]; // /collection of all edges

    class Edge
    {
        int src, dest;
    };

    // Creates a graph with V vertices and E edges
    Graph(int v,int e)
    {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i=0; i<e; ++i)
            edge[i] = new Edge();
    }

    // A utility function to find the subset of an element i
    int find(int parent[], int i)
    {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    // A utility function to do union of two subsets
    void Union(int parent[], int x, int y)
    {
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }


    // The main function to check whether a given graph
    // contains cycle or not
    int isCycle( Graph graph)
    {
        // Allocate memory for creating V subsets
        int parent[] = new int[graph.V];

        // Initialize all subsets as single element sets
        for (int i=0; i<graph.V; ++i)
            parent[i]=-1;

        // Iterate through all edges of graph, find subset of both
        // vertices of every edge, if both subsets are same, then
        // there is cycle in graph.
        for (int i = 0; i < graph.E; ++i)
        {
            int x = graph.find(parent, graph.edge[i].src);
            int y = graph.find(parent, graph.edge[i].dest);

            if (x == y)
                return 1;

            graph.Union(parent, x, y);
        }
        return 0;
    }

    // Driver Method
    public static void main (String[] args)
    {
        /* Let us create following graph
         0
        |  \
        |    \
        1-----2 */
        int V = 3, E = 3;
        Graph graph = new Graph(V, E);

        // add edge 0-1
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;

        // add edge 1-2
        graph.edge[1].src = 1;
        graph.edge[1].dest = 2;

        // add edge 0-2
        graph.edge[2].src = 0;
        graph.edge[2].dest = 2;

        if (graph.isCycle(graph)==1)
            System.out.println( "graph contains cycle" );
        else
            System.out.println( "graph doesn't contain cycle" );
    }
}
```

Run on IDE

Python

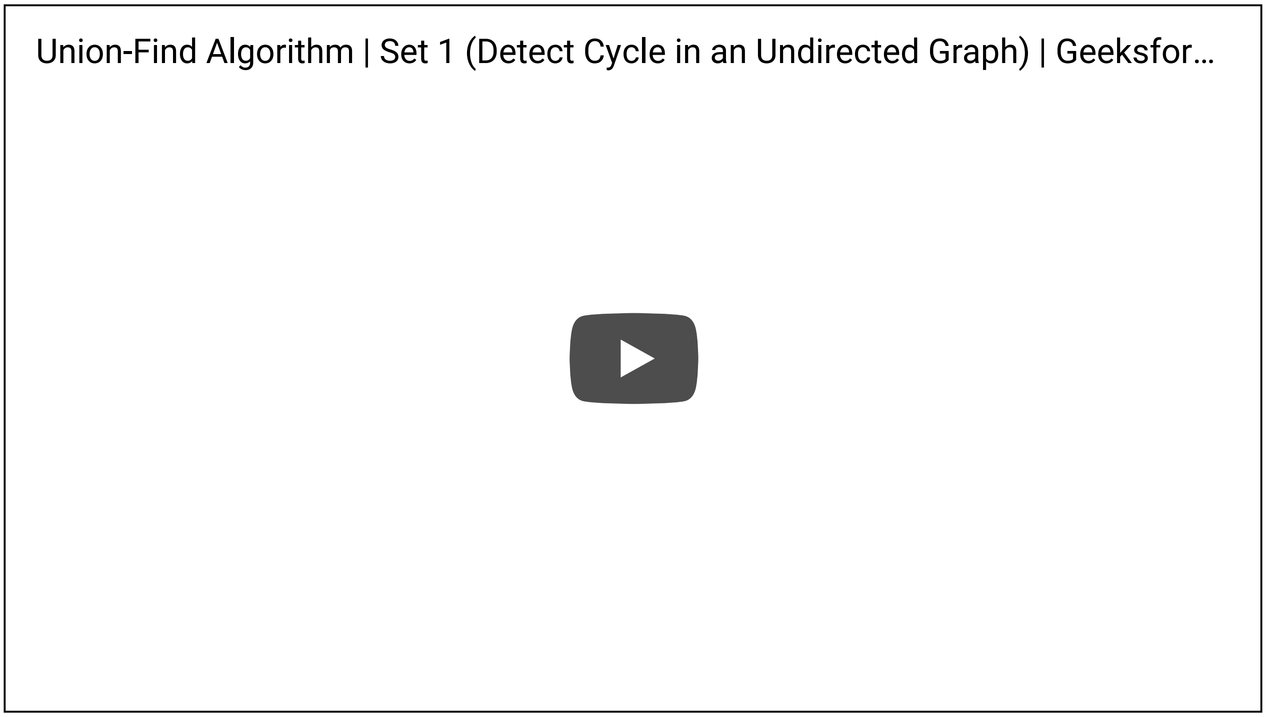
```
# Python Program for union-find algorithm to detect cycle in a undirected graph
# we have one egde for any two vertex i.e 1-2 is either 1-2 or 2-1 but not both

from collections import defaultdict

#This class represents a undirected graph using adjacency list representation
class Graph:

    def __init__(self,vertices):
        self.V= vertices #No. of vertices
        self.graph = defaultdict(list) # default dictionary to store graph


    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # A utility function to find the subset of an element i
    def find_parent(self, parent,i):
        if parent[i] == -1:
            return i
        if parent[i]!= -1:
             return self.find_parent(parent,parent[i])
    # A utility function to do union of two subsets
    def union(self,parent,x,y):
        x_set = self.find_parent(parent, x)
        y_set = self.find_parent(parent, y)
        parent[x_set] = y_set



    # The main function to check whether a given graph
    # contains cycle or not
    def isCyclic(self):

        # Allocate memory for creating V subsets and
        # Initialize all subsets as single element sets
        parent = [-1]*(self.V)

        # Iterate through all edges of graph, find subset of both
        # vertices of every edge, if both subsets are same, then
        # there is cycle in graph.
        for i in self.graph:
            for j in self.graph[i]:
                x = self.find_parent(parent, i)
                y = self.find_parent(parent, j)
                if x == y:
                    return True
                self.union(parent,x,y)


# Create a graph given in the above diagram
g = Graph(3)
g.addEdge(0, 1)
g.addEdge(1, 2)
g.addEdge(2, 0)

if g.isCyclic():
    print "Graph contains cycle"
else :
    print "Graph does not contain cycle "

#This code is contributed by Neelam Yadav
```

Run on IDE

Output:

```
graph contains cycle
```

Note that the implementation of *union()* and *find()* is naive and takes O(n) time in worst case. These methods can be improved to O(Logn) using *Union by Rank or Height*. We will soon be discussing *Union by Rank* in a separate post.



Union-Find Algorithm | Set 1 (Detect Cycle in an Undirected Graph) | Geeksfor...

Related Articles :

Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)
Disjoint Set Data Structures (Java Implementation)
Greedy Algorithms | Set 2 (Kruskal's Minimum Spanning Tree Algorithm)
Job Sequencing Problem | Set 2 (Using Disjoint Set)

This article is compiled by Aashish Barnwal and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Practice Tags : Graph union-find

Article Tags : Graph graph-cycle union-find

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content. Login to Improve this Article

Recommended Posts:

Kruskal’s Minimum Spanning Tree Algorithm | Greedy Algo-2
Detect cycle in an undirected graph
Detect Cycle in a Directed Graph
Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)
Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)
Minimum cost path from source node to destination node via an intermediate node
Print all the cycles in an undirected graph
Dominant Set of a Graph
Print the DFS traversal step-wise (Backtracking also)
Find alphabetical order such that words can be considered sorted

(Login to Rate)

2.9 Average Difficulty : **2.9/5.0**
Based on **153** vote(s)

Add to TODO List

Mark as DONE

Feedback Add Notes Improve Article

Basic Easy Medium Hard Expert

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY
About Us
Careers
Privacy Policy
Contact Us

LEARN
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE
Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE
Write an Article
Write Interview Experience
Internships
Videos