What is a Disjoint set data structure?

Two sets are called **disjoint sets** if they don't have any element in common, the intersection of sets is a null set.

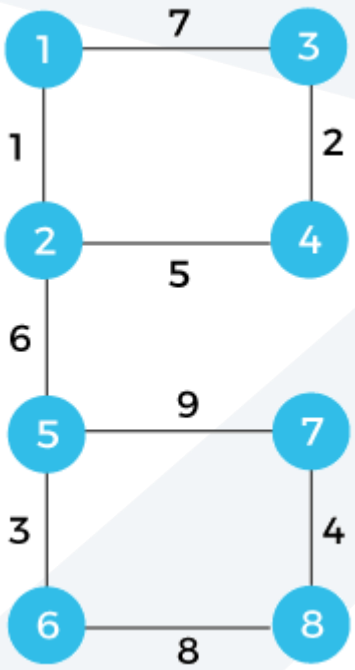
A data structure that stores non overlapping or disjoint subset of elements is called disjoint set data structure. The disjoint set data structure supports following operations:

- Adding new sets to the disjoint set.
- Merging disjoint sets to a single disjoint set using **Union** operation.
- Finding representative of a disjoint set using **Find** operation.
- Check if two sets are disjoint or not.

The disjoint set data structure is also known as union-find data structure and merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets. The various operations can be performed on the disjoint subsets. In this case, we can add new sets, we can merge the sets, and we can also find the representative member of a set. It also allows to find out whether the two elements are in the same set or not efficiently.

How can we detect a cycle in a graph?

We will understand this concept through an example. Consider the below example to detect a cycle with the help of using disjoint sets.



U = {1, 2, 3, 4, 5, 6, 7, 8}

Each vertex is labelled with some weight. There is a universal set with 8 vertices. We will consider each edge one by one and form the sets.

First, we consider vertices 1 and 2. Both belong to the universal set; we perform the union operation between elements 1 and 2. We will add the elements 1 and 2 in a set s1 and remove these two elements from the universal set shown below:

s1 = {1, 2}

The vertices that we consider now are 3 and 4. Both the vertices belong to the universal set; we perform the union operation between elements 3 and 4. We will form the set s3 having elements 3 and 4 and remove the elements from the universal set shown as below:

s2 = {3, 4}

The vertices that we consider now are 5 and 6. Both the vertices belong to the universal set, so we perform the union operation between elements 5 and 6. We will form the set s3 having elements 5 and 6 and will remove these elements from the universal set shown as below:

s3 = {5, 6}

The vertices that we consider now are 7 and 8. Both the vertices belong to the universal set, so we perform the union operation between elements 7 and 8. We will form the set s4 having elements 7 and 8 and will remove these elements from the universal set shown as below:

s4 = {7, 8}

The next edge that we take is (2, 4). The vertex 2 is in set 1, and vertex 4 is in set 2, so both the vertices are in different sets. When we apply the union operation, then it will form the new set shown as below:

s5 = {1, 2, 3, 4}

The next edge that we consider is (2, 5). The vertex 2 is in set 5, and the vertex 5 is in set s3, so both the vertices are in different sets. When we apply the union operation, then it will form the new set shown as below:

s6 = {1, 2, 3, 4, 5, 6}

Now, two sets are left which are given below:

s4 = {7, 8}

s6 = {1, 2, 3, 4, 5, 6}

The next edge is (1, 3). Since both the vertices, i.e.,1 and 3 belong to the same set, so it forms a cycle. We will not consider this vertex.

The next edge is (6, 8). Since both vertices 6 and 8 belong to the different vertices s4 and s6, we will perform the union operation. The union operation will form the new set shown as below:

s7 = {1, 2, 3, 4, 5, 6, 7, 8}

The last edge is left, which is (5, 7). Since both the vertices belong to the same set named s7, a cycle is formed.

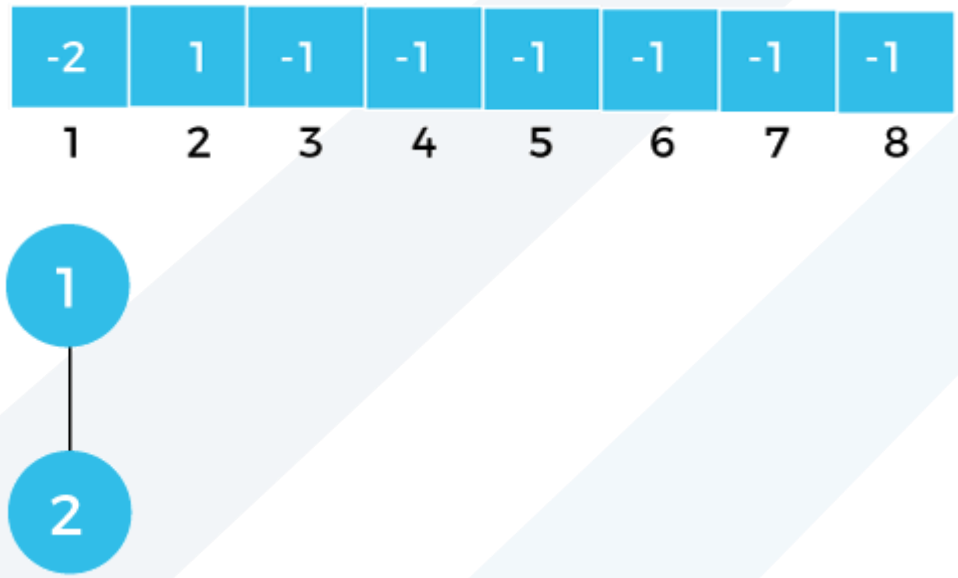
How can we detect a cycle with the help of an array?

Consider the below graph:

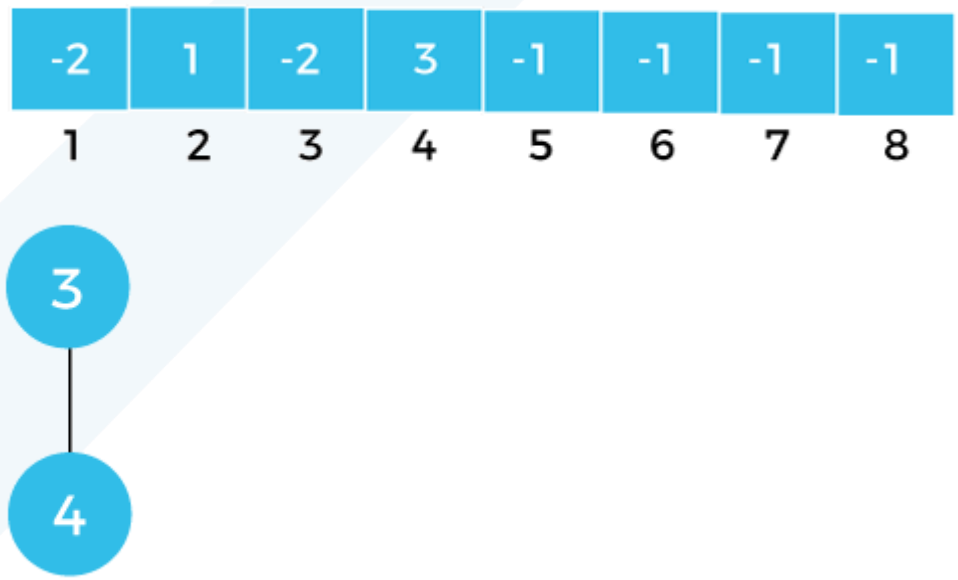
The above graph contains the 8 vertices. So, we represent all these 8 vertices in a single array. Here, indices represent the 8 vertices. Each index contains a -1 value. The -1 value means the vertex is the parent of itself.

Now we will see that how we can represent the sets in an array.

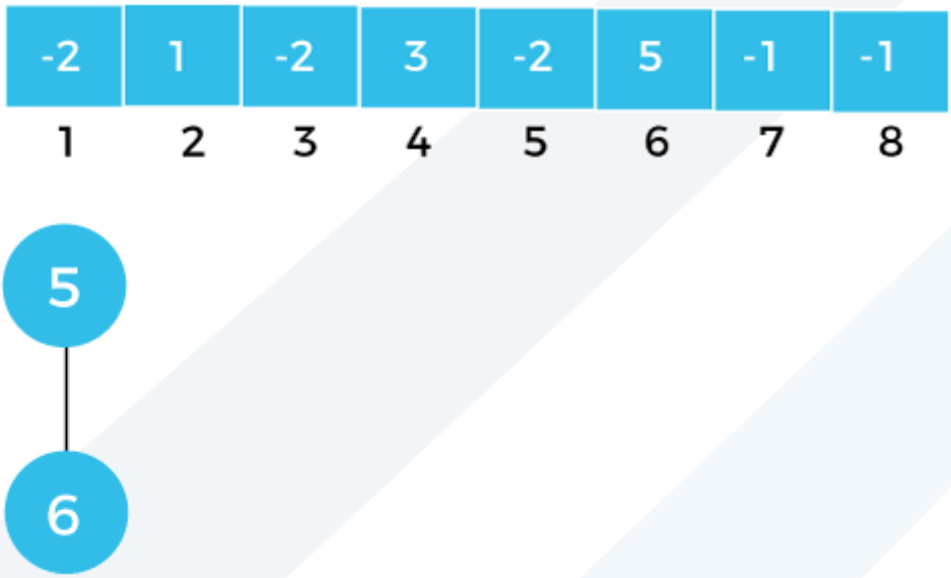
First, we consider the edge (1, 2). When we find 1 in an array, we observe that 1 is the parent of itself. Similarly, vertex 2 is the parent of itself, so we make vertex 2 as the child of vertex 1. We add 1 at the index 2 as 2 is the child of 1. We add -2 at the index 1 where '-' sign that the vertex 1 is the parent of itself and 2 represents the number of vertices in a set.



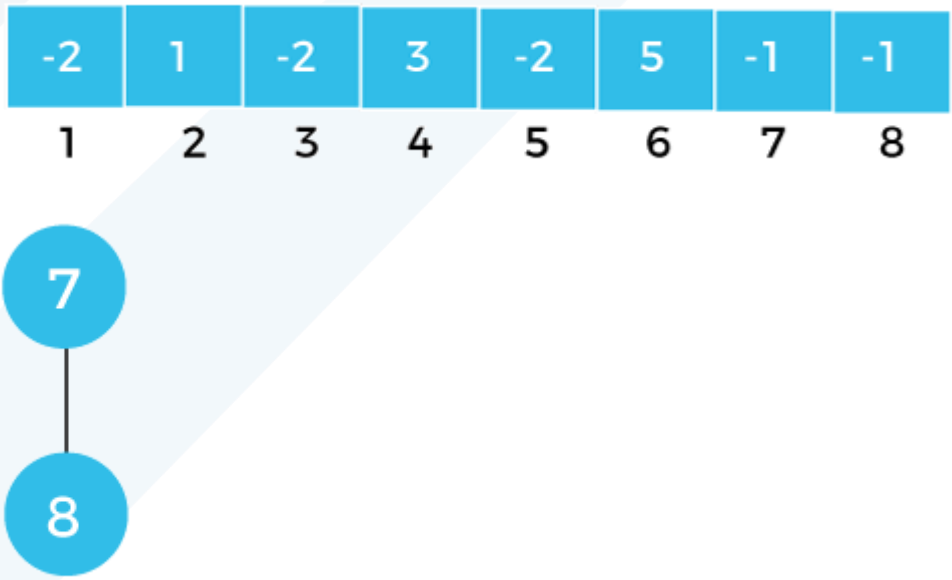
The next edge is (3, 4). When we find 3 and 4 in array; we observe that both the vertices are parent of itself. We make vertex 4 as the child of the vertex 3 so we add 3 at the index 4 in an array. We add -2 at the index 3 shown as below:



The next edge is (5, 6). When we find 5 and 6 in an array; we observe that both the vertices are parent of itself. We make 6 as the child of the vertex 5 so we add 5 at the index 6 in an array. We add -2 at the index 5 shown as below:

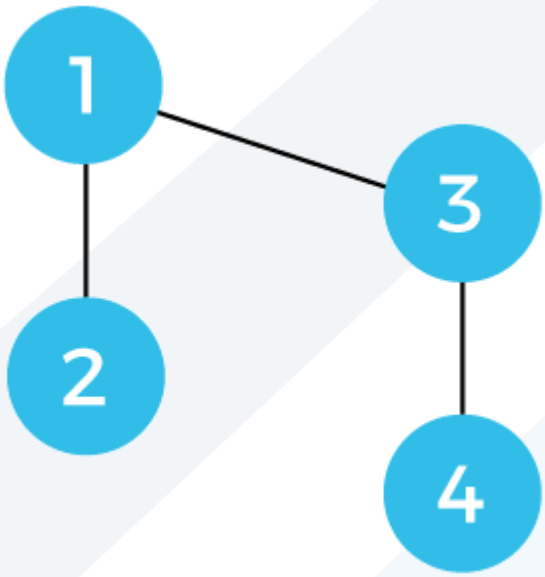


The next edge is (7, 8). Since both the vertices are parent of itself, so we make vertex 8 as the child of the vertex 7. We add 7 at the index 8 and -2 at the index 7 in an array shown as below:



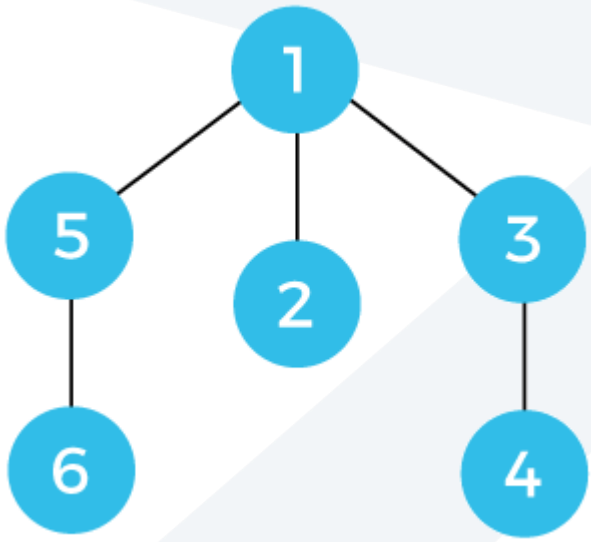
The next edge is (2, 4). The parent of the vertex 2 is 1 and the parent of the vertex is 3. Since both the vertices have different parent, so we make the vertex 3 as the child of vertex 1. We add 1 at the index 3. We add -4 at the index 1 as it contains 4 vertices.

Graphically, it can be represented as



The next edge is (2,5). When we find vertex 2 in an array, we observe that 1 is the parent of the vertex 2 and the vertex 1 is the parent of itself. When we find 5 in an array, we find -2 value which means vertex 5 is the parent of itself. Now we have to decide whether the vertex 1 or vertex 5 would become a parent. Since the weight of vertex 1, i.e., -4 is greater than the vertex of 5, i.e., -2, so when we apply the union operation then the vertex 5 would become a child of the vertex 1 shown as below:

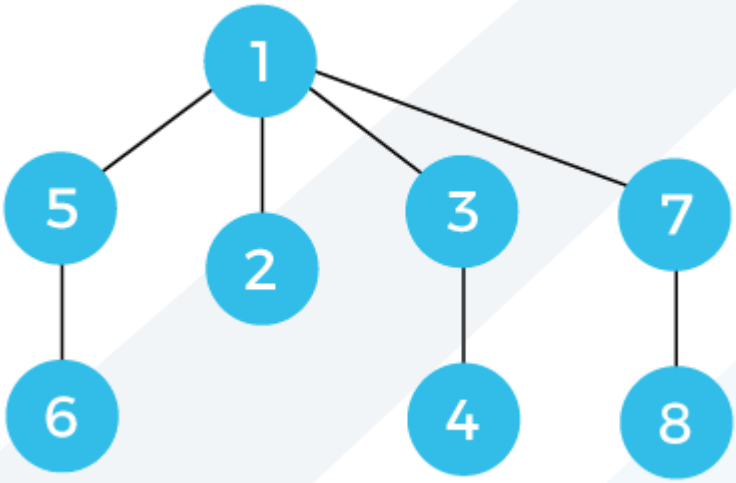
In an array, 1 would be added at the index 5 as the vertex 1 is now becomes a parent of vertex 5. We add -6 at the index 1 as two more nodes are added to the node 1.



The next edge is (1,3). When we find vertex 1 in an array, we observe that 1 is the parent of itself. When we find 3 in an array, we observe that 1 is the parent of vertex 3. Therefore, the parent of both the vertices are same; so, we can say that there is a formation of cycle if we include the edge (1,3).

The next edge is (6,8). When we find vertex 6 in an array, we observe that vertex 5 is the parent of vertex 6 and vertex 1 is the parent of vertex 5. When we find 8 in an array, we observe that vertex 7 is the parent of the vertex 8 and 7 is the parent of itself. Since the weight of vertex 1, i.e., -6 is greater than the vertex 7, i.e., -2, so we make the vertex 7 as the child of the vertex and can be represented graphically as shown as below:

We add 1 at the index 7 because 7 becomes a child of the vertex 1. We add -8 at the index 1 as the weight of the graph now becomes 8.



The last edge to be included is (5, 7). When we find vertex 5 in an array, we observe that vertex 1 is the parent of the vertex 5. Similarly, when we find vertex 7 in an array, we observe that vertex 1 is the parent of vertex 7. Therefore, we can say that the parent of both the vertices is same, i.e., 1. It means that the inclusion (5,7) edge would form a cycle.

```
// C++ implementation of disjoint set
// in this implementation rank is height of tree representing the set

#include <bits/stdc++.h>
using namespace std;

class DisjSet {
    int *rank, *parent, n;

public:

    // Constructor to create and
    // initialize sets of n items
    DisjSet(int n)
    {
        rank = new int[n];
        parent = new int[n];
        this->n = n;
        makeSet();
    }

    // Creates n single item sets
```

```
    void makeSet()
    {
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i]=0;
        }
    }

    // Finds set of given item x
    int find(int x)
    {
        // Finds the representative of the set
        // that x is an element of
        if (parent[x] != x) {

            // if x is not the parent of itself
            // Then x is not the representative of
            // his set,
            parent[x] = find(parent[x]);

            // so we recursively call Find on its parent
            // and move i's node directly under the
            // representative of this set
        }

        return parent[x];
    }

// Finds the representative of the set that i
// is an element of.

#include <bits/stdc++.h>
using namespace std;

int find_with_path_compression(int i)
{

    // If i is the parent of itself
    if (Parent[i] == i) {

        // Then i is the representative
        return i;
    }
    else {

        // Recursively find the representative.
        int result = find(Parent[i]);

        // We cache the result by moving i's node
        // directly under the representative of this
        // set
        Parent[i] = result;

        // And then we return the result
        return result;
```

```
    }
}
    // Do union of two sets represented
    // by x and y.
    void Union(int x, int y)
    {
        // Find current sets of x and y
        int xset = find(x);
        int yset = find(y);

        // If they are already in same set
        if (xset == yset)
            return;

        // Put smaller ranked item under
        // bigger ranked item if ranks are
        // different
        if (rank[xset] < rank[yset]) {
            parent[xset] = yset;
        }
        else if (rank[xset] > rank[yset]) {
            parent[yset] = xset;
        }

        // If ranks are same, then increment
        // rank.
        else {
            parent[yset] = xset;
            rank[xset] = rank[xset] + 1;
        }
    }
};

// Driver Code
int main()
{

      // Function Call
    DisjSet obj(5);
    obj.Union(0, 2);
    obj.Union(4, 2);
    obj.Union(3, 1);

    if (obj.find(4) == obj.find(0))
        cout << "Yes\n";
    else
        cout << "No\n";
    if (obj.find(1) == obj.find(0))
        cout << "Yes\n";
    else
        cout << "No\n";

    return 0;
}
```