Introduction to Disjoint Set (Union-Find Data Structure) - GeeksforGeeks

Source: https://www.geeksforgeeks.org/union-find/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Introduction to Disjoint Set (Union-Find Data Structure) Last Updated : 17 Jan, 2026 Two sets are called disjoint sets if they don't have any element in common. The disjoint set data structure is used to store such sets. It supports following operations: Merging two disjoint sets to a single set using Union operation. Finding representative of a disjoint set using Find operation. Check if two elements belong to same set or not. We mainly find representative of both and check if same. Consider a situation with a number of persons and the following tasks to be performed on them: Add a new friendship relation , i.e. a person x becomes the friend of another person y i.e adding new element to a set. Find whether individual x is a friend of individual y (direct or indirect friend) Examples: We are given 10 individuals say, a, b, c, d, e, f, g, h, i, j Following are relationships to be added: a <-> b b <-> d c <-> f c <-> i j <-> e g <-> j Given queries like whether a is a friend of d or not. We basically need to create following 4 groups and maintain a quickly accessible connection among group items: G1 = {a, b, d} G2 = {c, f, i} G3 = {e, g, j} G4 = {h} Try it on GfG Practice Find whether x and y belong to the same group or not, i.e. to find if x and y are direct/indirect friends. Partitioning the individuals into different sets according to the groups in which they fall. This method is known as a Disjoint set Union which maintains a collection of Disjoint sets and each set is represented by one of its members. To answer the above question two key points to be considered are: How to Resolve sets? Initially, all elements belong to different sets. After working on the given relations, we select a member as a representative . Check if 2 persons are in the same group? If representatives of two individuals are the same, then they are friends. Data Structures used are: Array: An array of integers is called Parent[] . If we are dealing with N items, i'th element of the array represents the i'th item. More precisely, the i'th element of the Parent[] array is the parent of the i'th item. These relationships create one or more virtual trees. Tree: It is a Disjoint set . If two elements are in the same tree, then they are in the same Disjoint set . The root node (or the topmost node) of each tree is called the representative of the set. There is always a single unique representative of each set. A simple rule to identify a representative is if 'i' is the representative of a set, then Parent[i] = i . If i is not the representative of his set, then it can be found by traveling up the tree until we find the representative. Operations on Disjoint Set Data Structures: 1. Find: The task is to find representative of the set of a given element. The representative is always root of the tree. So we implement find() by recursively traversing the parent array until we hit a node that is root (parent of itself). 2. Union: The task is to combine two sets and make one. It takes two elements as input and finds the representatives of their sets using the Find operation, and finally puts either one of the trees (representing the set) under the root node of the other tree. C++ #include <iostream> #include <vector> using namespace std ; class UnionFind { vector < int > parent ; public : UnionFind (int size) { parent . resize (size); // Initialize the parent array with each // element as its own representative for (int i = 0 ; i < size ; i ++) { parent [i] = i ; } } // Find the representative (root) of the // set that includes element i int find (int i) { // If i itself is root or representative if (parent [i] == i) { return i ; } // Else recursively find the representative // of the parent return find (parent [i]); } // Unite (merge) the set that includes element // i and the set that includes element j void unite (int i , int j) { // Representative of set containing i int irep = find (i); // Representative of set containing j int jrep = find (j); // Make the representative of i's set // be the representative of j's set parent [irep] = jrep ; } }; int main () { int size = 5 ; UnionFind uf (size); uf . unite

(1 , 2); uf . unite (3 , 4); bool inSameSet = (uf . find (1) == uf . find (2)); cout << "Are 1 and 2 in the same set? " << (inSameSet ? "Yes" : "No") << endl ; return 0 ; } Java import java.util.Arrays ; public class UnionFind { private int [] parent ; public UnionFind (int size) { // Initialize the parent array with each // element as its own representative parent = new int [size] ; for (int i = 0 ; i < size ; i ++) { parent [i] = i ; } } // Find the representative (root) of the // set that includes element i public int find (int i) { // if i itself is root or representative if (parent [i] == i) { return i ; } // Else recursively find the representative // of the parent return find (parent [i]); } // Unite (merge) the set that includes element // i and the set that includes element j public void union (int i , int j) { // Representative of set containing i int irep = find (i); // Representative of set containing j int jrep = find (j); // Make the representative of i's set be // the representative of j's set parent [irep] = jrep ; } public static void main (String [] args) { int size = 5 ; UnionFind uf = new UnionFind (size); uf . union (1 , 2); uf . union (3 , 4); boolean inSameSet = uf . find (1) == uf . find (2); System . out . println ("Are 1 and 2 in the same set? " + inSameSet); } } Python class UnionFind : def __init__ (self , size): # Initialize the parent array with each # element as its own representative self . parent = list (range (size)) def find (self , i): # If i itself is root or representative if self . parent [i] == i : return i # Else recursively find the representative # of the parent return self . find (self . parent [i]) def unite (self , i , j): # Representative of set containing i irep = self . find (i) # Representative of set containing j jrep = self . find (j) # Make the representative of i's set # be the representative of j's set self . parent [irep] = jrep # Example usage size = 5 uf = UnionFind (size) uf . unite (1 , 2) uf . unite (3 , 4) in_same_set = (uf . find (1) == uf . find (2)) print ("Are 1 and 2 in the same set?" , "Yes" if in_same_set else "No") C# using System ; public class UnionFind { private int [] parent ; public UnionFind (int size) { // Initialize the parent array with each // element as its own representative parent = new int [size]; for (int i = 0 ; i < size ; i ++) { parent [i] = i ; } } // Find the representative (root) of the // set that includes element i public int Find (int i) { // If i itself is root or representative if (parent [i] == i) { return i ; } // Else recursively find the representative // of the parent return Find (parent [i]); } // Unite (merge) the set that includes element // i and the set that includes element j public void Union (int i , int j) { // Representative of set containing i int irep = Find (i); // Representative of set containing j int jrep = Find (j); // Make the representative of i's set be // the representative of j's set parent [irep] = jrep ; } public static void Main (string [] args) { int size = 5 ; UnionFind uf = new UnionFind (size); uf . Union (1 , 2); uf . Union (3 , 4); bool inSameSet = uf . Find (1) == uf . Find (2); Console . WriteLine ("Are 1 and 2 in the same set? " + (inSameSet ? "Yes" : "No")); } } JavaScript class UnionFind { constructor (size) { // Initialize the parent array with each // element as its own representative this . parent = Array . from ({ length : size }, (_ , i) => i); } find (i) { // If i itself is root or representative if (this . parent [i] === i) { return i ; } // Else recursively find the representative // of the parent return this . find (this . parent [i]); } unite (i , j) { // Representative of set containing i const irep = this . find (i); // Representative of set containing j const jrep = this . find (j); // Make the representative of i's set // be the representative of j's set this . parent [irep] = jrep ; } } // Example usage const size = 5 ; const uf = new UnionFind (size); // Unite sets containing 1 and 2, and 3 and 4 uf . unite (1 , 2); uf . unite (3 , 4); // Check if 1 and 2 are in the same set const inSameSet = uf . find (1) === uf . find (2); console . log ("Are 1 and 2 in the same set?" , inSameSet ? "Yes" : "No"); Output Are 1 and 2 in the same set? Yes The above union() and find() are naive and the worst case time complexity is linear. The trees created to represent subsets can be skewed and can become like a linked list. Following is an example of worst case scenario. The above operations can be optimized using Union by Rank and Path Compression. Comment Article Tags: Article Tags: Graph DSA Intellipaat Huawei union-find QA - Placement Quizzes-Data Interpretation graph-cycle Java-SecureRandom TCS-coding-questions B-Tree + 6 More