Disjoint class set: an efficient data structure to solve the equivalence problem.

Equivalence relation:

For the relation R: for every pair of elements (a, b) where a, b is an element of S, aRb is either true or false. If aRb is true, then a is related to b.

-should satisfy 3 properties: reflexive, symmetric, transitive.

Reflexive: aRa for all a in S.

Symmetric: aRb if and only if bRa.

Transitive: aRb and bRc, this implies aRc.

EX:

Is the relation <= an equivalence relation?

1. a <= a
2. a <= b does not mean b <= a unless b = a.
3. a <= b <= c, a <= c.

Not an equivalence relation.

Electrical connectivity relationship: all connections are by metal wire.

a // a, reflexive.

b // a is symmetric.

a // b, b // c, a // c. True.

Is an equivalence relation.

Equivalence relation is defined for set {a1, a2, a3, a­4, a5}. There are 25 (n­2) pairs of elements.

However, if a1 ~ a2, a3~a4, a5~a1, a4~a5, all elements are related.

Equivalence class: of an element a in S is the subset of S that contains all the elements that are related to a.

Equivalence class forms a partition of S.

Make partitions S1, S2, S3, S4. They are equivalence classes.

If S1 U S2 = null, then they are not related.

Say a is in S2 and b is in S3, then the find operation will return the name of the equivalence class the element belongs to.

find(a) = S2, find(b) = S3, since they are not in the same equivalence class, and the elements are related, must merge the two sets (union).

Use a tree to represent each set. Since each element of the tree has the same root, use root value as the name of the set.

When the algorithm starts, each set has 1 element. Not a binary tree nor a binary search tree, does not need to have left and right children, but have reference to its parent.

EX:

0, 1, 2, 3, 4, 5, 6, 7 is in S.

[0][1][2][3][4][5][6][7]

Make the parent link of one tree’s root link to the root node of the other tree.

Make new root after union (x, y) to be x.

union(4, 5): [4] 🡨 [5] Both 4 and 5 now belong to the set 4.

union(6, 7): [6] 🡨 [7] Both 6 and 7 now belong to the set 6.

union(4, 6): [4] 🡨 [5] Set 6 is not part of set 4.

🡨 [6] 🡨 [7]

union(1, 0): [1] 🡨 [0] Both 1 and 0 now belong to set 1.

x = find(5) = 4 (the set 4, because 5 is the child of 4).

union(1, x) = [1] 🡨 [0]

🡨 [4] 🡨 [5]

🡨 [6] 🡨 [7]

S[i] = -1 if i is the root.

The array for the previous example:

S[-1 -1 -1 -1 -1 4 4 6]

0 1 2 3 4 5 6 7

4 is the root. [4] 🡨 [5]

🡨 [6] 🡨 [7]

2’s root is 2, 6’s root is 4. 2 and 6 are related, use union.

x = find(6), returns 4.

union(2, x);

public DisjointSet(int numOfElements) {

s = new int[numOfElements];

}

public int find(int x) {

if (s[x] < 0) {

return x;

} else {

return find(s[x]);

}

}

public void union(int x, int y) {

s[y] = x;

}

Height becomes an issue.