Disjoint Set Union (DSU)

Union - Find

N elements

→ each element is a part of different set.

(A) (B) (C) (D) (E)

. . . . . n sets

→ keep track of connectivity of elements in a particular set.

→ Union - find data structure.

E. g. properties

Union (B, E)

① → B, E, C are connected.
② → B, E, and D are connected.

There are two different sets connected subset

$\{B, E, C\}$ $\{B, E, D\}$

① Union (B, E) → connect two elements A and B

② Find (E, D) → find, if there is any path connecting two elements A and B.

Ex → $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

10 elements.

Arr[] Indexed by elements & set, being a set id.

① Union (3, 4)

② Union (1, 2)

③ Union (1, 3)

④ Union (7, 8)

⑤ Union (7, 9)

⑥ Union (1, 5)

→ connected component

$\{1, 2, 3, 4\}$, $\{1, 5\}$, $\{5, 6\}$, $\{7, 8\}$, $\{7, 9\}$

→ relate these elements with nodes in a graph.

→ each subset can be considered as a connected component.

Find (A, B) → Bool / Integer

① Find (1, 4) → set connected

② Find (1, 7) → False

equivalence of connected sets:-

① Union (A, B) → Replace component containing two elements A and B with single component.

② Find (A, B) → check if two objects A and B are in same component or not.

Step (1)

(E, F)

Arr: 0 1 2 3 5 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Union (2, 3)

Union of $O(1)$

init (Arr[], int n) {

for (int i=0; i<n; i++) {

Arr[i] = i;

}

}

boolean find (int Arr[], int A, int B) {

if (Arr[A] == Arr[B]) {

return True;

}

else {

return false;

}

}

boolean union (int[] Arr, int n) {int A, int B) {

int temp = Arr[B];

for (int i=0; i<n; i++) {

if (Arr[i] == temp) {

Arr[i] = Arr[A];

}

}

}

$O(n^2)$

N - queries → N^2

Idea: Arr[A] is a parent of A.

Root element of each subset is the only operated element in that subset having itself as the parent.

Root of i is not element?

Arr[i] = i ?

Arr[i] = i

Root (i) = i

root (1) → 1

root (0) → 0

Arr[i] = 0

Arr[0] = 2

Arr[2] = 2

while (Arr[i] == i)

① Union (1, 3)

{0, 1}

② Union (5, 3)

Arr[5] [Arr[Arr[i]]] . . . .

{2, 4, 1}

③ Union (3, 4)

{3, 3}

④ Union (1, 4)

{0, 1, 2, 3, 4, 5}

How to calculate root element ??

int root (int Arr[], int i) {

while (Arr[i] != i) {

i = Arr[i];

}

return i;

}

$f \rightarrow O(1)$, $U \rightarrow O(N)$

$f \rightarrow O(N)$, $U \rightarrow O(N)$

Find (0, 5)

boolean find (int A, int B) {

if (root(A) == root(B)) {

return True;

}

else

return false;

}

int union (int Arr[], int A, int B) {

int root_A = root (Arr, A);

int root_B = root (Arr, B);

Arr [root_A] = root_B;

}

* We will need keep track of size of each subset and start union using that elements, we can connect the root of subset having smaller no. of elements to the root of having larger no. of elements.

size (B) = 6

weighted union operation.

Arr →

size →

initialize (int[] Arr, int N) {

for (int i = 0; i<N; i++) {

Arr[i] = i;

size[i] = 1;

}

}

weighted union (int Arr[], int size[], int A, int B) {

int root_A = root (A);

int root_B = root (B);

if (size [root_A] < size [root_B]) {

Arr [root_A] = Arr [root_B];

size [root_B] += size[root_A];

} else {

Arr [root_B] = Arr [root_A];

size [root_A] += size[root_B];

}

}

$A \rightarrow 2$

$B \rightarrow 4$

E.g. {4, 1, 2, 3, 5, 6, 7}

Arr →

size →

Maintaining a balanced tree, while weighted union and find function from N to $\log_2 N$

Union ← $\log_2 N$ + find

Applications:

① to determine connected components in a graph.

② to determine cycles in a graph.