Notes

Disjoint Set Union (Union Find)

282 Code Monk Disjoint-sets Union Find

The efficiency of an algorithm sometimes depends on using an efficient data structure. A good choice of data structure can reduce the execution time of an algorithm and Union-Find is a data structure that falls in that category.

Let’s say, you have a set of N elements which are partitioned into further subsets, and you have to keep track of connectivity of each element in a particular subset or connectivity of subsets with each other. To do this operation efficiently, you can use Union-Find Data Structure.

Let’s say there are 5 people A, B, C, D E. A is a friend of B, B is a friend of C and D is a friend of E. As we can see:

1) A, B and C are connected to each other.
2) D and E are connected to each other.

So we can use Union Find Data Structure to check whether one friend is connected to another in a direct or indirect way or not. We can also determine the two different disconnected subsets. Here 2 different subsets are {A, B, C} and {D, E}.

You have to perform two operations here :

Union (A, B) - connect two elements A and B. Find (A, B) - find, is there any path connecting two elements A and B

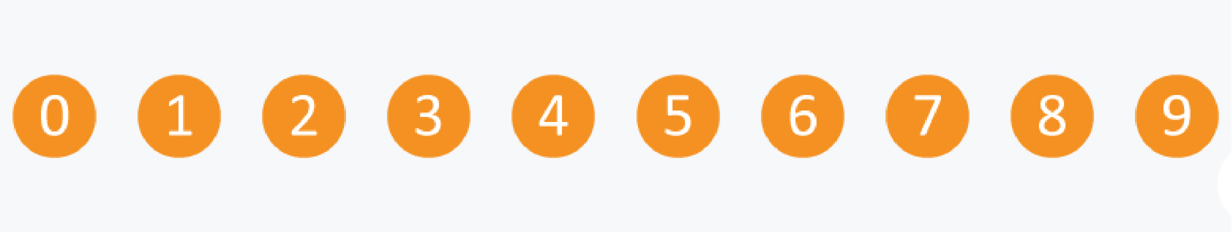
Example: You have a set of elements S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Here you have 10 elements (N = 10).We can use an array **Arr** to manage the connectivity of elements. Arr[ ] indexed by elements of set, having size of N (as N elements in set) and can be used to manage the above operations.

Assumption: A and B objects are connected only if Arr[A] = Arr [B].

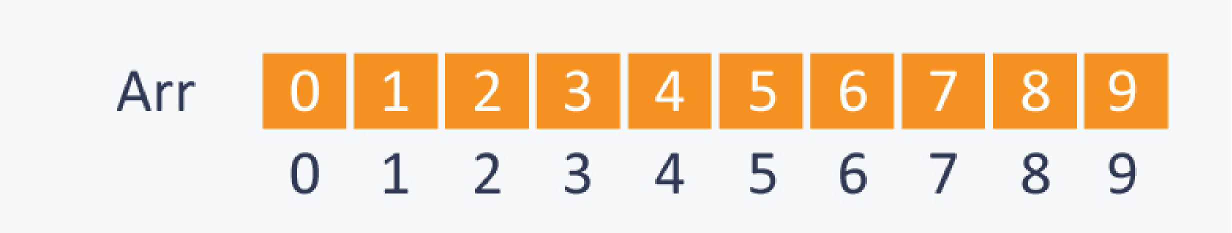
Now how we will implement above operations :

Find (A, B) - check if Arr[A] is equal to Arr[B] or not. Union (A, B) - Connect A to B and merge the components having A and B by changing all the elements ,whose value is equal to Arr[A], to Arr[B].

Initially there are 10 subsets and each subset has single element in it.

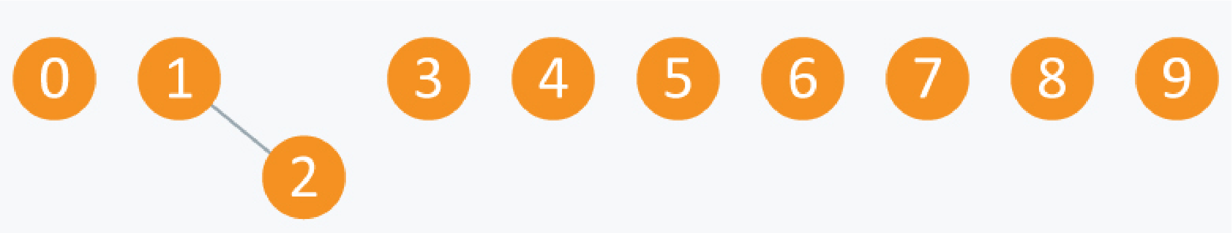


When each subset contains only single element, the array Arr is:

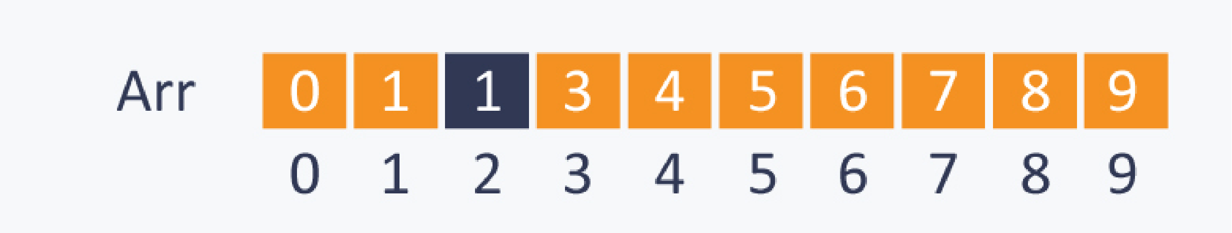


Arr	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

Let’s perform some Operations: 1) Union(2, 1)

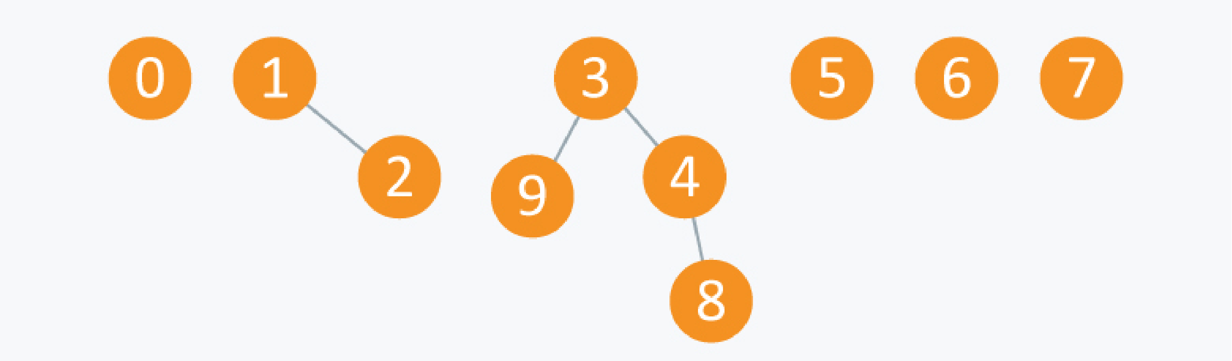


Arr will be:

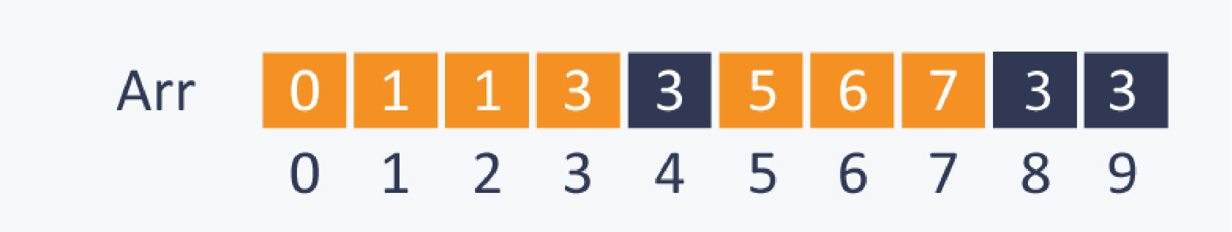


Arr	0	1	1	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

2) Union(4, 3)
3) Union(8, 4)
4) Union(9, 3)

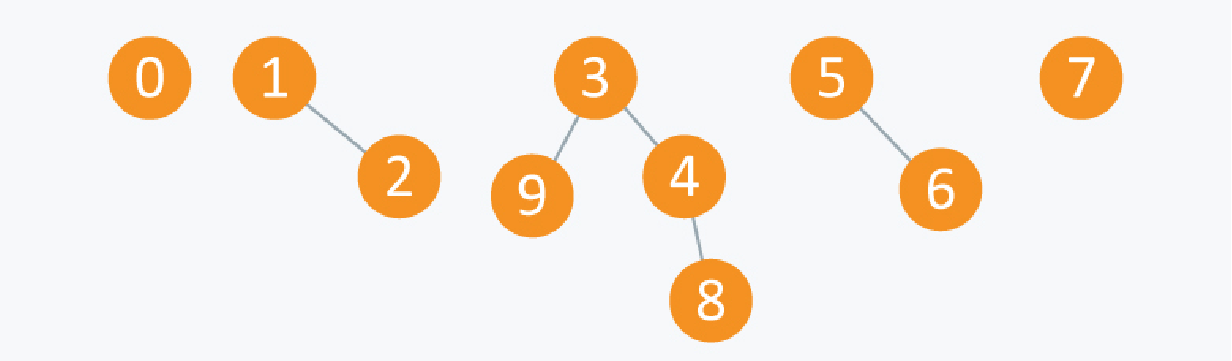


Arr will be:

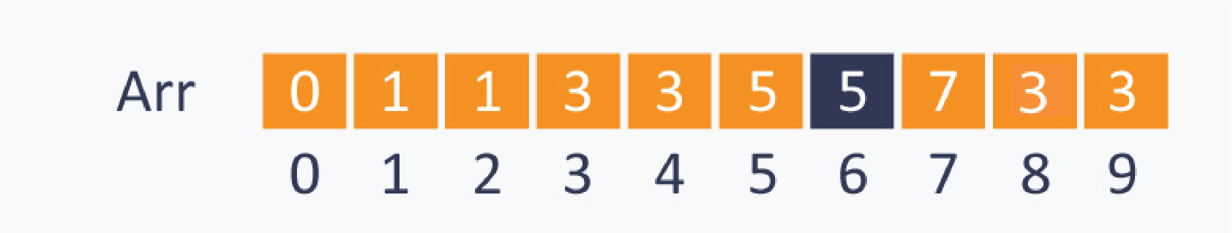


Arr	0	1	1	3	3	5	6	7	3	3
	0	1	2	3	4	5	6	7	8	9

5) Union(6, 5)



Arr will be:



Arr										
	0	1	1	3	3	5	5	7	3	3
	0	1	2	3	4	5	6	7	8	9

After performing some operations of Union(A ,B), you can see that now there are 5 subsets. First has elements {3, 4, 8, 9}, second has {1, 2}, third has {5, 6}, fourth has {0} and fifth has {7}. All these subsets are said to be Connected Components.

One can also relate these elements with nodes of a graph. The elements in one subset can be considered as the nodes of the graph which are connected to each other directly or indirectly, therefore each subset can be considered as **connected component.**

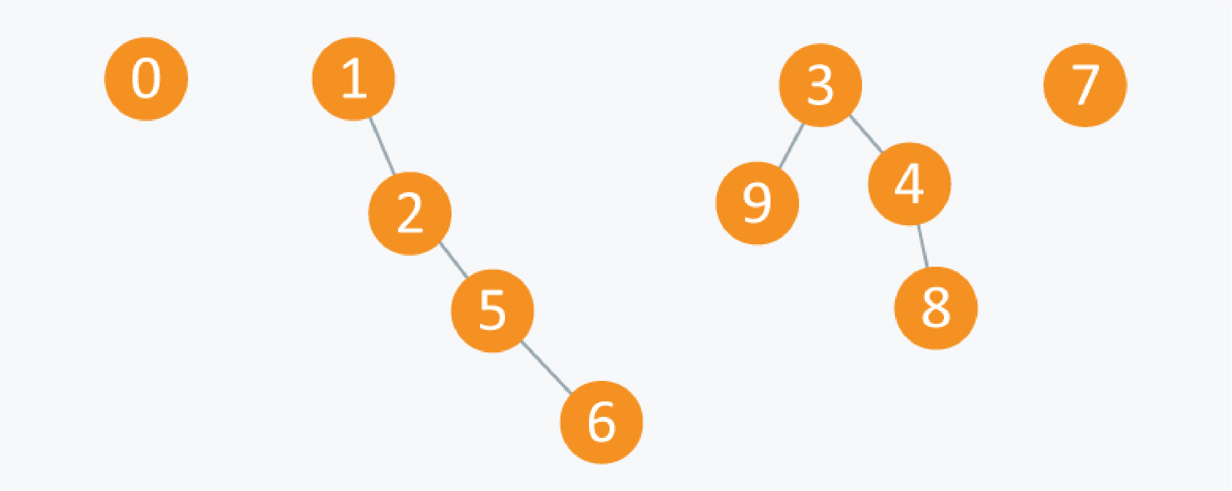
From this, we can infer that Union-Find data structure is useful in Graphs for performing various operations like connecting nodes, finding connected components etc.

Let's perform some Find(A, B) operations. 1) Find (0, 7) - as 0 and 7 are disconnected ,this will gives false result.
2) Find (8, 9) -though 8 and 9 are not connected directly ,but there exist a path connecting 8 and 9, so it will give us true result.

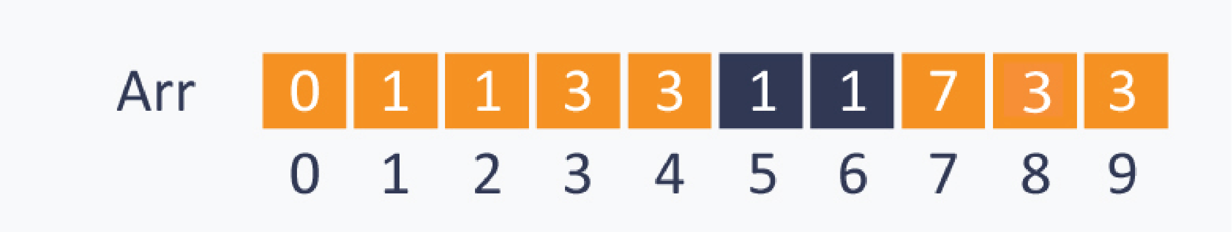
When we see above operations in terms of components, then :

Union(A, B) - Replace components containing two objects A and B with their union.
Find(A, B) - check if two objects A and B are in same component or not.

So if we perform operation Union(5, 2) on above components, then it will be :



Now the Arr will be:



Arr	0	1	1	3	3	1	1	7	3	3
	0	1	2	3	4	5	6	7	8	9

Implementation:

Initially there are N subsets containing single element in each subset, so to initialize array we will use **initialize ( ) function.**

```
void initialize( int Arr[ ], int N)
{
    for(int i = 0;i<N;i++)
    Arr[ i ] = i ;
}
//returns true,if A and B are connected, else it will return false.
 bool find( int Arr[ ], int A, int B)
{
if(Arr[ A ] == Arr[ B ])
return true;
else
return false;
}
//change all entries from Arr[ A ] to Arr[ B ].
void union(int Arr[ ], int N, int A, int B)
{
    int TEMP = Arr[ A ];
for(int i = 0; i < N;i++)
    {
    if(Arr[ i ] == TEMP)
    Arr[ i ] = Arr[ B ];
```

```
        }
}
```

As loop in Union function iterates through all the N elements for connecting two elements. So performing this operation on N objects will take **O(N^2)** time, which is quite inefficient.

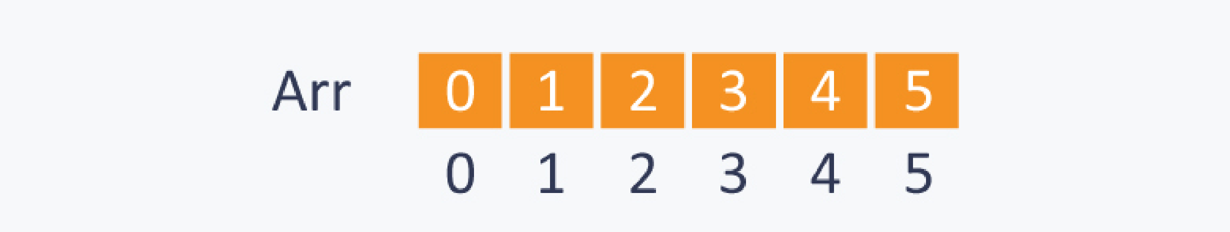
Let’s try another approach:

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

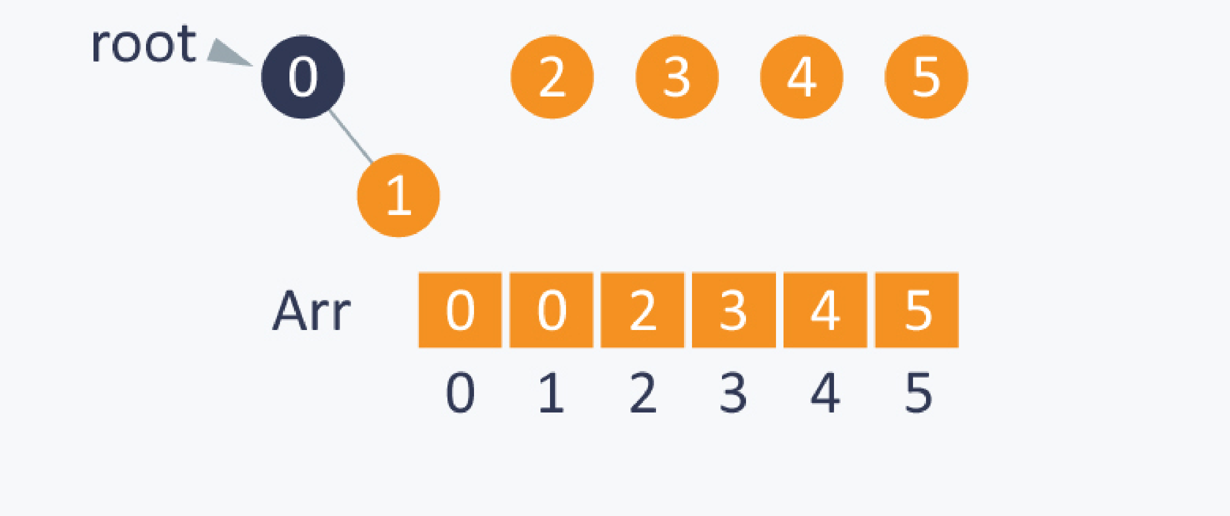
We can consider a **root element** of each subset, which is a only special element in that subset having itself as the parent. Let’s say R is a root element, then Arr[R] = R.

To make it more clear,let’s take a subset S = {0, 1, 2, 3, 4, 5}

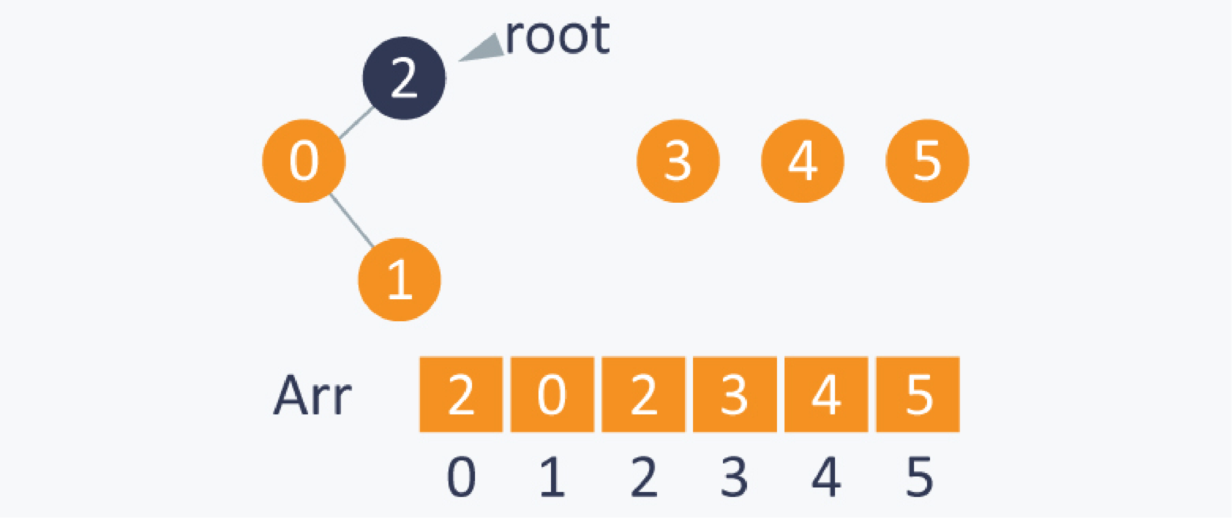
Initially each element is the root of itself in all subsets, as Arr[i] = i, where i is element in the set, therefore root(i) = i.



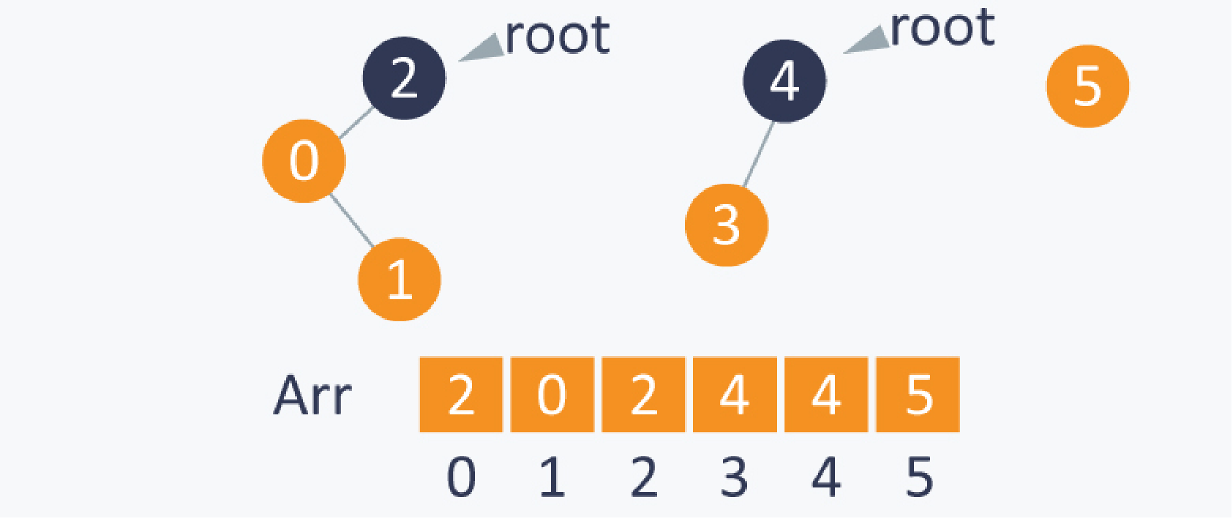
Performing Union(1, 0) will connect 1 to 0 and will set root(0) as the parent of root(1). As root(1) = 1, and root(0) = 0, therefore value of Arr[1] will be changed from 1 to 0. It will make 0 as a root of subset containing elements {0, 1}.



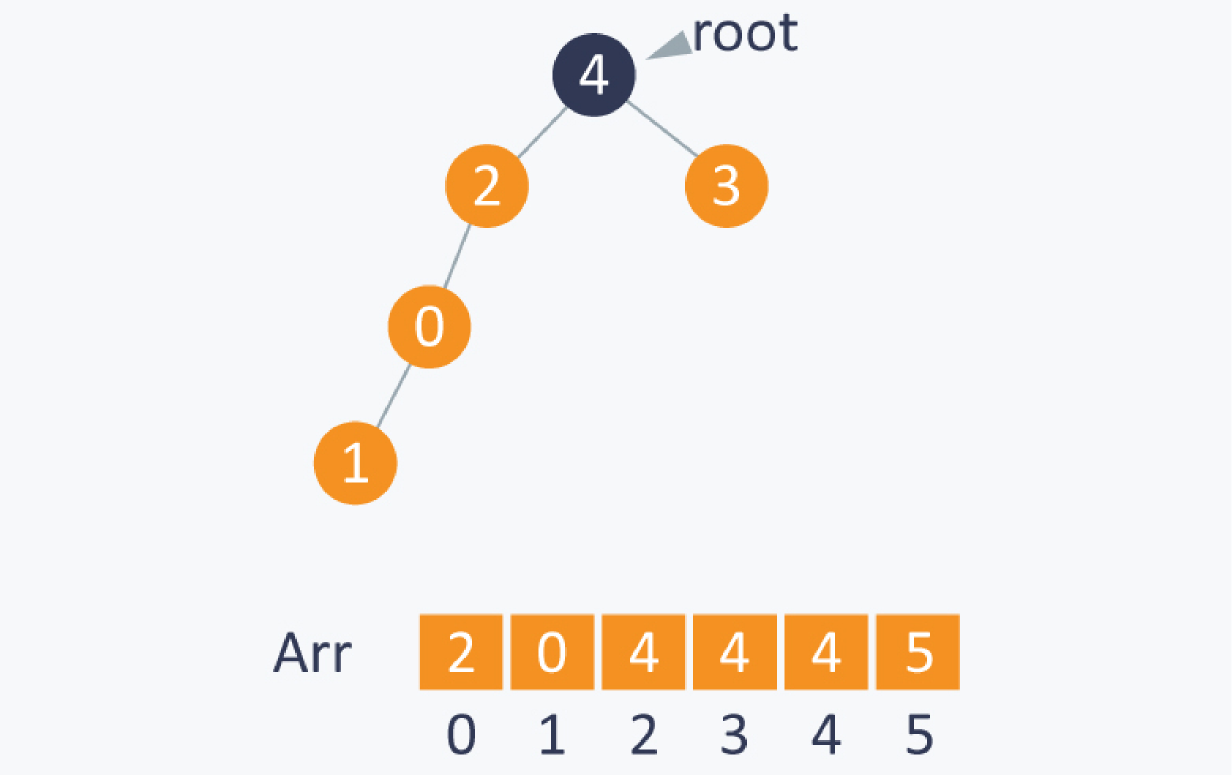
Now performing Union(0, 2), will indirectly connect 0 to 2, by setting root(2) as the parent of root(0). As root(0) is 0 and root(2) is 2, therefore it will change value Arr[0] from 0 to 2. Now 2 will be the root of subset containing elements {2, 0, 1}.



Similarly Union(3, 4) will indirectly connect 3 to 4, by setting root(4) as the parent of root(3). As root(3) is 3 and root(4) is 4, therefore it will change value of Arr[3] from 3 to 4. It will make 4 as a root of subset containing elements {3, 4}.



Performing Union(1, 4) will indirectly connect 1 to 4, by setting root(4) as the parent of root(1). As root(4) is 4 and root(1) is 2, therefore it will change value of Arr[2] from 2 to 4. It makes 4 as root of set containing elements {0, 1, 2, 3, 4}.



After each step you can observe the change in array Arr also.
After performing required Union(A, B) operations, we can easily perform the Find(A, B) operation to check whether A and B are connected or not. It can be checked by calculating roots of both A and B. If roots of A and B are same, that means both A and B are in same subset and are connected.

Now how to calculate root of a element ?
As we know that Arr[i] is the parent of i (where i is the element of set),then the root of i is **Arr[Arr[Arr[......Arr[i]......]]]** until Arr[i] is not equal to i. Simply we can run a loop until we get a element which is a parent of itself.

Note: This can be only done when there is no cycle in the elements of subset, otherwise loop will run infinitely.

Find(1, 4) - 1 and 4 have same root as 4, therefore it means they are connected and this operation will give true as a result.

Find(3, 5) - 3 and 5 do not have same root, as root(3) is 4 and root(5) is 5. It means they are not connected and it will give false as a result.

Implementation:

As initially all the elements are parent of itself,which can be done using initialize function discussed above.

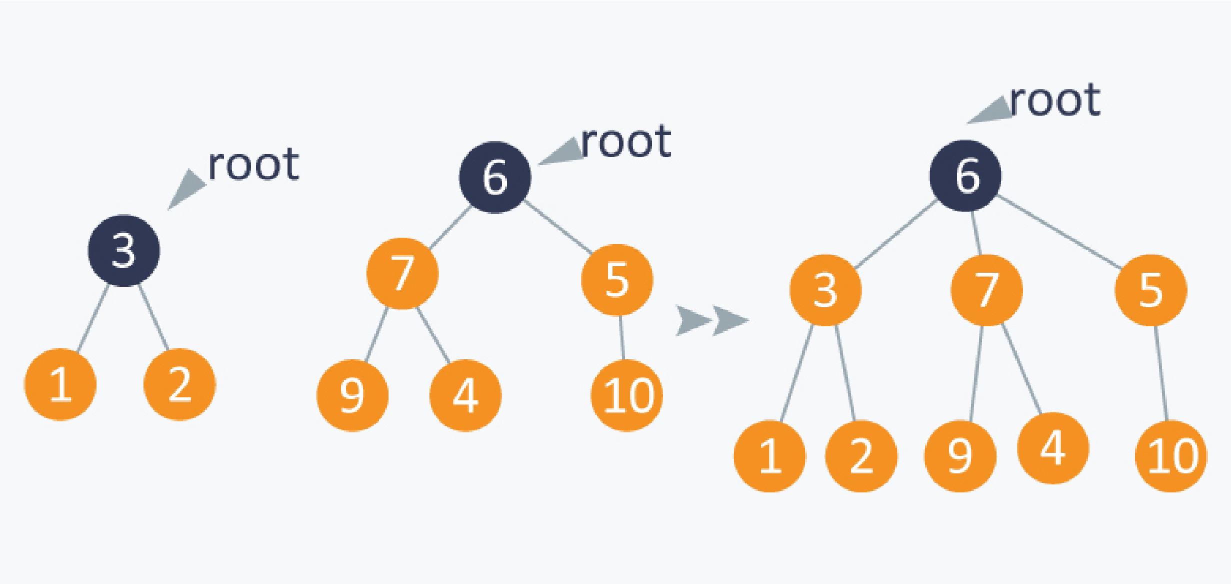
```
//finding root of an element.
int root(int Arr[ ],int i)
{
    while(Arr[ i ] != i)           //chase parent of current element until
it reaches root.
    {
     i = Arr[ i ];
    }
    return i;
}

/*modified union function where we connect the elements by changing the
root of one of the element */

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 ;        //setting parent of root(A) as root(B).
}
bool find(int A,int B)
{
    if( root(A)==root(B) )        //if A and B have same root,means they
are connected.
    return true;
    else
    return false;
}
```

Now as you can see, in worst case, this idea will also take linear time in connecting 2 elements and even in finding that if two elements are connected or not, it will take linear time. Another disadvantage is that while connecting two elements, we do not check which subset has more element than other and sometimes it creates a big problem as in worst case we have to perform approximately linear time operations.

We can avoid this, by keeping the track of size of each subset and then while connecting two elements, we can connect the root of subset having smaller number of elements to the root of subset having larger number of elements.

Example:

Here if we want to connect 1 and 5, then we will connect the root of Subset A (subset which contains 1) will be connected to root of Subset B (contains 5), this is because Subset A contains less number of elements than of Subset B.



It will balance the tree formed by the above operations. We call this operation as **weighted_union operation** .

Implementation:

Initially the size of each subset will be one as each subset will have only one element and we can initialize it in the initialize function discussed above:

size[ ] array will keep track of size of each subset.

```
//modified initialize function:
void initialize( int Arr[ ], int N)
{
    for(int i = 0;i<N;i++)
    {
Arr[ i ] = i ;
size[ i ] = 1;
}
}
```

root() and **find()** function will be same as above .

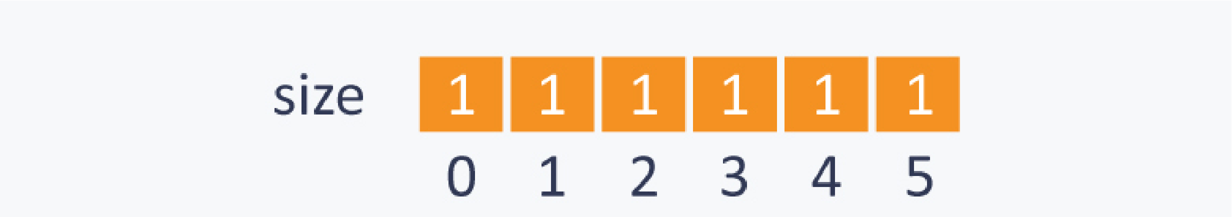
Union function will be modified as we will connect two subsets according to the number of elements in subset.

//modified union function

```
void 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];
}


}
```

Example:

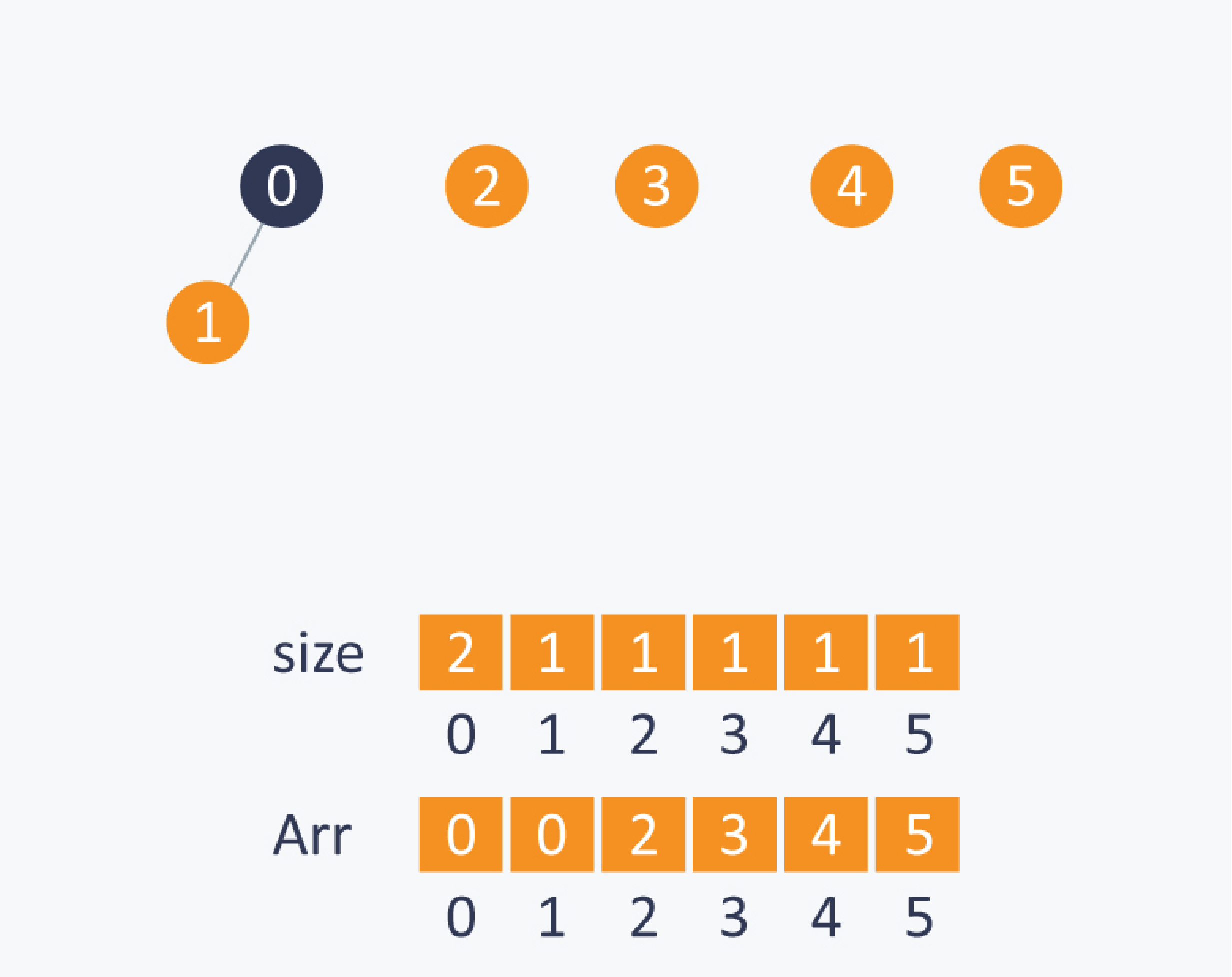
You have a set S = {0, 1, 2, 3, 4, 5} Initially all the subsets have a single element and each element is a root of itself. Initially size[ ] array will be :



size	1	1	1	1	1	1
	0	1	2	3	4	5

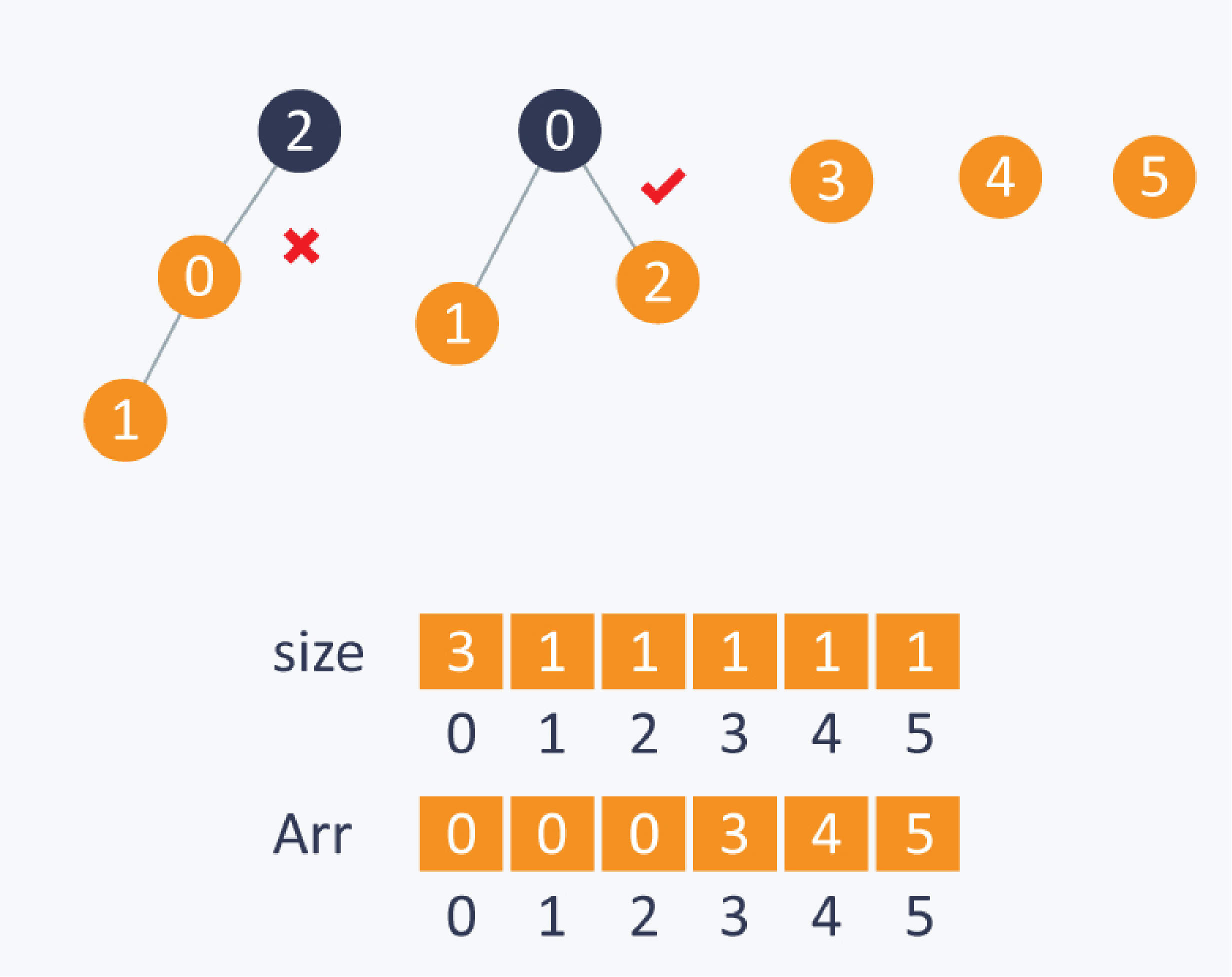
Perform Union(0, 1). Here we can connect any root of any element with root of other one as both the element's subsets have same size and then we will update the respective size.

If we connect 1 to 0 and make 0 as a root and then size of 0 will change from 1 to 2.



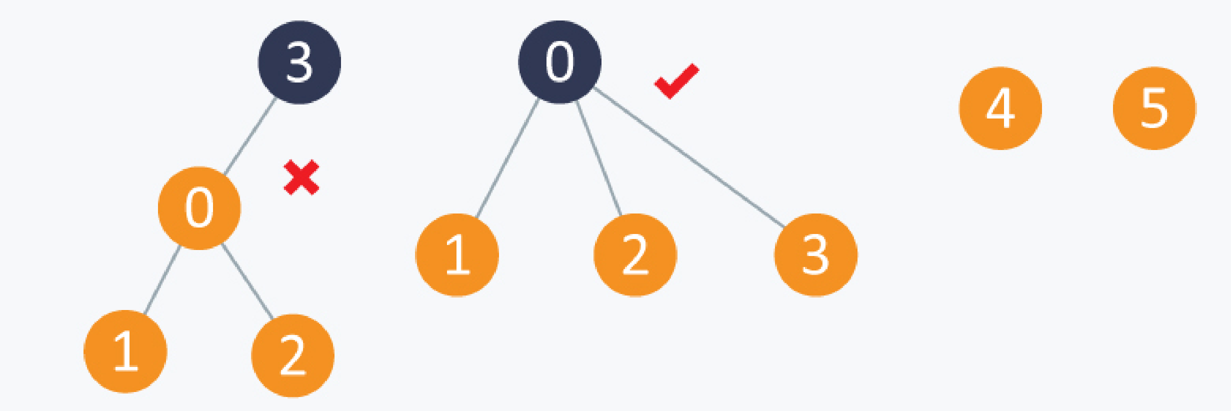
size	2	1	1	1	1	1
	0	1	2	3	4	5
Arr	0	0	2	3	4	5
	0	1	2	3	4	5

While performing Union(1, 2), we will connect root(2) with root(1) as subset of 2 has less number of elements than number of elements in subset of 1.



size	3	1	1	1	1	1
	0	1	2	3	4	5
Arr	0	0	0	3	4	5
	0	1	2	3	4	5

Similarly in Union(3, 2), it will connect root(3) to root(2) as subset of 3 has less number of element than number of elements in subset of 2.



size	4	1	1	1	1	1
	0	1	2	3	4	5
Arr	0	0	0	0	4	5
	0	1	2	3	4	5

Maintaining a **balance tree**, will reduce complexity of union and find function from **N** to **log$_2$N.**

Can we improve more ?

Idea: Union with path compression : While computing the root of A, set each i to point to its grandparent (thereby halving the path length), where i is the node which comes in between path, while computing root of A.

```
// modified root function.

int root (int Arr[ ] ,int i)
{
    while(Arr[ i ] != i)
    {
        Arr[ i ] = Arr[ Arr[ i ] ] ;
i = Arr[ i ];
    }
return i;
}
```

When we use Weighted-union with path compression it takes **log * N** for each union find operation,where **N** is the number of elements in the set.

log *N is the iterative function which computes the number of times you have to take log of N till the value of N doesn’t reaches to 1.

log*N is much better than log N, as its value reaches at most up to 5 in the real world.

Applications:

1) As explained above, Union-Find is used to determine the connected components in a graph. We can determine whether 2 nodes are in the same connected component or not in the graph. We can also determine that by adding an edge between 2 nodes whether it leads to cycle in the graph or not.
We learned that we can reduce its complexity to a very optimum level, so in case of very large and dense graph, we can use this data structure.

2) It is used to determine the cycles in the graph. In the **Kruskal’s Algorithm**, Union Find Data Structure is used as a subroutine to find the cycles in the graph, which helps in finding the minimum spanning tree.(Spanning tree is a subgraph in a graph which connects all the vertices and spanning tree with minimum sum of weights of all edges in it is called minimum spanning tree).

Practice Problems:
1) Panda and Combination

2) Monk's birthday treat

Solve Problems

Tweet

COMMENTS □ **SORT BY:** Relevance □

Login/Signup to Comment

Naivedya Bansal 14 juillet 2015 at 17:19
What is log *n exactly ? How did the complexity become log *n? Please explain.
□ 6 votes

Pulkit Gupta 14 juillet 2015 at 21:42
log*n is log(log(......log(n))) (recursive log that many times till the number becomes 1). it's an extremely slowly growing function you can notice if n = 2^65536 ,log*(n) = 5.
notice in the path compression step you link the node to its grandparent ,ultimately decreasing the levels int the tree by bringing nodes closer to the root(tree gets flattened)
□ 40 votes

Prateek Garg □ Author 15 juillet 2015 at 09:01
Yes, pulkit is right.
□ 5 votes

Naivedya Bansal 15 juillet 2015 at 09:49
Thanks :)
□ 4 votes

Arun Prasad 15 juillet 2015 at 15:33

?

```
void weighted-union(int Arr[ ],int size[ ],int A,int B)
{
int root_A = root(A);
int root_B = root(B);
//EDIT 1:
if(root_A == root_B)
return;
// This will avoid if two sets are already unioned, even if root of both are same, the size increases
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];
}
}
```

□ 12 votes

Harsh Paliwal 14 juin 2017 at 08:38

Why is there Only one parameter in root() Function?

□ 0 votes

Shivam Tanay 29 décembre 2015 at 08:02

great Article :P

□ 9 votes

Sarvagya Agarwal 18 juillet 2015 at 02:22

why did we stop at grand-parent . why not grand-grand-parent . that would be even faster . or am i wrong ?

□ 3 votes

Reeshabh Kumar Ranjan 19 août 2018 at 07:59

@Prateek Garg please clear this doubt.

□ 0 votes

Reeshabh Kumar Ranjan 19 août 2018 at 09:57

I understood it while implementing DSU. You need to update the parent of each element in the path up to the top. If you skip an element in between, its parent will not be updated.

□ 1 vote

Kisan Thapa 7 février 2020 at 15:05

The main purpose of path compression is to shorten the length from root to other elements. If we cut grand-grand parent and attach to the root, the cut part would be long and it will not reduce length.

□ 0 votes

Naivedya Bansal 14 juillet 2015 at 18:00

The practice problems that are mentioned I was able to solve using bfs only. How are they to be done with the union find data structure?

□ 1 vote

dinkar gahoi 1 mai 2016 at 16:43

May be you can try this problem from SPOJ http://www.spoj.com/problems/FOXLINGS/
Though m not sure that union find is the only way for above problem

□ 2 votes

Shivam Sharma 11 décembre 2017 at 07:35

I don't think your first approach was right , it depends upon the order ,if it is -
2) Union(4, 3)

3) Union(8, 4)
then all is good bu if you reverse them -
2) Union(8, 4)
3) Union(4, 3)
it will become wrong as you are placing the minimum element as the value.

□ 3 votes

Nipun Mittal 9 juin 2018 at 02:39

There is a logical error in the weighted-union method. if (root_A == root_B) then they are already connected, so size[root_A] += size[root_B] or vice-versa should not be executed because it will wrongly double the size of the subset.

□ 2 votes

akash kandpal 7 novembre 2018 at 05:36

size here is the total number of nodes in a subset so it's fine I believe :) Am I correct ?

□ 0 votes

SOUMYA GHOSH 14 juillet 2015 at 15:27

```
//finding root of an element.
int root(int Arr[ ],int i)
{
while(Arr[ i ] != i) //chase parent of current element until it reaches root.
{
i = Arr[ i ];
}
return i;
}
```

can someone plz explain this way of finding root?

□ 0 votes

Prateek Garg □ Author 15 juillet 2015 at 09:19

only root element of any subset will have the Arr[i] equals to i. Lets say subset have 4 elements {4,5,6,7} and 7 is a root of subset.
4 is connected to 5 then Arr[4] = 5.
5 is connected to 7 , then Arr[5] = 7.
6 is connected to 7, then Arr[6] = 7.
and Arr[7] = 7.
Thus for finding root of 4, it will move from Arr[4] (i.e 5) to Arr[5] (i.e 7) and then to Arr[7] which is 7 only, hence we found a root.

□ 1 vote

SOUMYA GHOSH 15 juillet 2015 at 11:26

So we could have done it recursively too?

□ 0 votes

Sahil Sharma 3 janvier 2020 at 06:15

Yes.

```
int root (int arr[] , int i)
{
if(arr[i]==i) return i;
else return root(arr,arr[i]);
}
```

□ 0 votes

Farshid Nooshi 26 novembre 2017 at 16:42

cool

□ 1 vote

Ashwani Gautam 10 juillet 2015 at 08:10

in the 6th image from the top shouldn't it be arr[8]=3.?

□ 0 votes

?

Prateek Garg □ Author 13 juillet 2015 at 10:22

Yes. It is fixed. Thanks :)

□ 0 votes

Ravi Shankar 11 juillet 2015 at 06:11

Paragraph after 15th image : ie
"Find(1, 4) - 1 and 4 have same root as 2, therefore it means they are connected and this operation will give true as a result."
parent of 1 and 4 is 4 not 2.Please correct it.

□ 0 votes

6 LIVE EVENTS

Prateek Garg □ Author 13 juillet 2015 at 10:23

Fixed. Thanks.

□ 0 votes

Amar Kaswan 11 juillet 2015 at 07:10

[developer:ptk23] in 1st scenario where time complexity is O(n^2) if we check find(3,8) arr[3]=3 and arr[8]=4 then it will give false but (3,8) are connected indirectly as we perform Union(4, 3), Union(8, 4) and Union(9, 3) so {3,4,8,9} all are connected please clarify the doubt Thanks

□ 0 votes

Prateek Garg □ Author 13 juillet 2015 at 10:23

arr[8] will be 3. It is fixed.

□ 0 votes

Amar Kaswan 13 juillet 2015 at 17:30

you have updated the arr[8] but according to your algo of union still it will not come 3 it will be 4 only

```
void union(int Arr[ ], int N, int A, int B)
{
int TEMP = Arr[ A ];
for(int i = 0; i < N;i++)
{
if(Arr[ i ] == TEMP)
Arr[ i ] = Arr[ B ];
}
}
```

operation union (8,4)
currently array is 0 1 1 3 4 5 6 7 8 9
now please elaborate according to your algorithm

□ 0 votes

Prateek Garg □ Author 13 juillet 2015 at 17:39

Current array will not be the one as you have written above. Before operation union(8, 4) , you can see there is an another operation union(4, 3) which will change the array to 0 1 1 3 3 5 6 7 8 9 , and after that it will perform union(8, 4) which will give correct results with the same algorithm.

□ 0 votes

Amar Kaswan 14 juillet 2015 at 13:42

Thank you got it :)

□ 0 votes

Ramit Das 11 juillet 2015 at 09:22

Now how do we get back the subsets?

□ 0 votes

Prateek Garg □ Author 13 juillet 2015 at 10:25

Sorry, I didn't get you.

□ 0 votes

Ramit Das 13 juillet 2015 at 17:02

I was asking on how to get the individual sets from all the union-find structure. That is from the union-find structure after the union-finding operations how could I individually

?

get back the sets. I worked on it using coloring. It required a O(n) n being the size of the union-find array. Can we do better?

□ 0 votes

Prateek Garg □ Author 15 juillet 2015 at 09:23

Yes in worst case, it will always reach to O(N) .

□ 0 votes

6 LIVE EVENTS

Pulkit Mendiratta 13 juillet 2015 at 07:03

Amazing Article :)

□ 0 votes

Prateek Garg □ Author 13 juillet 2015 at 10:25

Thanks :)

□ 0 votes

SOUMYA GHOSH 14 juillet 2015 at 15:33

why Weighted-union with path compression akes log * N for each union ? Is the complexity calculated accordint to tree height?

□ 0 votes

Prateek Garg □ Author 15 juillet 2015 at 09:06

Yes the complexity depends on the height of tree. In path compression, in each iteration we are connecting node to its grand parent, thus reducing the level. As the maximum height will be log N, and each iteration we are jumping from node to its grandparent, then to the grandparent of grandparent, which makes the complexity to log* N .

□ 0 votes

jayasurya j 24 avril 2016 at 14:46

can we go back more than 2 steps ? i.e instead of going to its grandparents can we go to its parent of grandparent?

□ 0 votes

Mohamed Ayman 18 juin 2016 at 23:32

Yes, If It had been done recursively, we could have set the parent to the root directly.

□ 0 votes

Arun Prasad 14 juillet 2015 at 19:15

Hello,
Thanks for the Nice Tutorial,
The complexity of union and weighted union operation is same,isn't it??
And by using weighted-union , we make balanced binary tree, which will make reduce the complexity for further find operation on the set, is that the advantage of weighted-union over union operation?

□ 0 votes

Prateek Garg □ Author 15 juillet 2015 at 08:58

This comment has been deleted.

Arun Prasad 15 juillet 2015 at 10:02

Okay :)

□ 0 votes

Shubham Tandle 15 juillet 2015 at 05:49

im not able to recognise the lasst case of panda and combination

□ 0 votes

sathiyaseelan 15 juillet 2015 at 10:02

Hi , anyone tried Disjoint set based solution for Monk's birthday treat? I'm facing problem for the following test case
1
1 2

?

The output has to be 1(by selecting 2) But I'm getting 2. Since the root of both 1 & 2 contains 2. If I try to solve the porblem by checking arr[i] =i then I got problem with cylce scnarios.(the default one).

https://www.hackerearth.com/submission/2129539/

Any help appreciated.Thanks

□ 0 votes

6 LIVE EVENTS

Pulkit Gupta 15 juillet 2015 at 12:35

http://www.codechef.com/problems/FIRESC/
http://www.codechef.com/problems/GALACTIK
you can try these problems on DSU

□ 0 votes

Aditya Sharma 20 juillet 2015 at 21:08

Can you please explain why does the first(the following code) implementation takes o(n^2)

```
void union(int Arr[ ], int N, int A, int B)
{
int TEMP = Arr[ A ];
for(int i = 0; i < N;i++)
{
if(Arr[ i ] == TEMP)
Arr[ i ] = Arr[ B ];
}
}
```

□ 0 votes

Abhishek Chakraborty 21 juillet 2015 at 08:03

Great article.. Learned a lot, and even solved a TCS CodeVita question using this idea... :D

□ 0 votes

Preeti Nagal 30 juillet 2015 at 17:10

It's amazing.....

□ 0 votes

hardik agrawal 16 octobre 2015 at 19:53

very nicely written and explained! thanks.. :)

□ 0 votes

Dhruv Kaushik 17 octobre 2015 at 03:12

Great tutorial. But I think that the find function can be more optimized in a specific case. Suppose A and B are both present on same branch, then if while going from one element to its root (to find its root), if we encounter the second element in the path, then we can directly output yes and terminate function, instead of finding roots of both the element and then comparing. I think in this specific case. we can optimize it more (although the code will be a little more complex, as we cannot then use the root functions directly). Hope It is clear enough to make you understand what I want to say.....

□ 0 votes

> **Banipreet Singh Raheja** 25 octobre 2017 at 23:22
>
> still, the worst case of that case would be of linear time complexity. Suppose that your first element is the leaf and the second element is the root. Now you will be doing comparisons up till the root which would take the loop in linear complexity, however, with the help of log*(N) the time complexity would be lesser, as far as I can comprehend the article. Correct me if I am wrong, I am not sure but this is what I thought out of it.
>
> □ 0 votes

Sandeep Ravindra 1 décembre 2015 at 17:19

Nice article!

□ 0 votes

Paras Avkirkar 29 décembre 2015 at 07:33

Nice Article.
According to Wikipedia, (https://en.wikipedia.org/wiki/Disjoint-set_data_structure), inside the

?

union function, they maintain the size of only the bigger branch and not size of whole subset.
From Wikipedia --->

```
function Union(x, y)
xRoot := Find(x)
yRoot := Find(y)
if xRoot == yRoot
return
// x and y are not already in same set. Merge them.
if xRoot.rank < yRoot.rank
xRoot.parent := yRoot
else if xRoot.rank > yRoot.rank
yRoot.parent := xRoot
else
yRoot.parent := xRoot
xRoot.rank := xRoot.rank + 1
```

There they only increase the size of the root only when both the branches have equal size (there they refer size as 'rank') else the rank is kept same.
According to which method is efficient? Should we maintain rank of the maximum branch or size of whole subset?

□ 0 votes

Nilesh Hirani 29 décembre 2015 at 13:15

Rank is more efficient. Since Rank governs the time complexity of Find() function not size of the subset.

□ 0 votes

Nilesh Hirani 29 décembre 2015 at 13:18

Sorry I meant root() function

□ 0 votes

Abhimanyu Singh Shekhawat 6 avril 2016 at 12:42

Very nice work!

□ 0 votes

karandeep singh dhillon 11 juillet 2016 at 07:36

we can use dfs also to implement this?

□ 0 votes

Pintu Das 15 août 2016 at 12:43

Please update weighted_union function by changing two lines:

```
void 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 ] =root_B; // This is the change line 1
size[root_B] += size[root_A];
}
else
{
Arr[ root_B ] = root_A; // This is the change line 2
size[root_A] += size[root_B];
}
}
```

□ 0 votes

Shubho Shaha □ Edited 12 octobre 2016 at 07:03

I think this weighted-union implementation doesn't works for following case:

```
2 10
7 3
10 5
7 2
```

9 12
2 12
But overall very clearly described the whole process of DSU.

□ 0 votes

Rohit 21 décembre 2016 at 04:25

How to find the no. of elements in any given subset or total no. of elements present in a specific subset using disjoint union set algorithms. please help

□ 0 votes

Vivek Singh 18 janvier 2017 at 15:15

for loop in the union of 1st program is not required.

□ 0 votes

Abhishek Agrawal 26 janvier 2017 at 16:55

Good article mate!

□ 0 votes

Psicodelico Demonio del Piano 9 avril 2017 at 20:35

I've created a cpp (c++11) version, with helpful comments.
https://github.com/pianodaemon/cpp_magic/blob/master/algorithms_experiences/union_find/union_find.cpp

□ 0 votes

> **Shivam Sharma** 11 décembre 2017 at 07:36
>
> You should not do this , otherwise you can go to jail cause of the copyright !
>
> □ 0 votes

Bedir Tapkan □ Edited 6 mai 2017 at 17:54

Awesome ! On second code section, there is no array implemented in find function, although the other functions have it , fix please.

□ 0 votes

Sumit Saurav 24 août 2017 at 09:15

Nice Article :_^_

□ 0 votes

Aneesh Hiregange 18 septembre 2017 at 12:11

One more good problem on dsu -
https://szkopul.edu.pl/problemset/problem/neqccAalHI0t2ieiHEEgQHzy/site/?key=statement

□ 0 votes

Rohit Rajak 12 octobre 2017 at 06:11

how sort(p, p + edges) is being sorted. i know it is being sorted on weight but we didn't mentioned it. does that mean it automatically sort according first element of pair?

□ 0 votes

Engin öztürk 20 novembre 2017 at 07:02

I was looking for a nice tutorial that can teach me this algorithm and d.s. and thanks very much, now I understood and just used in a problem. Great tutorial!

□ 0 votes

ak3899 20 janvier 2018 at 18:41

great tutorial on DSU!!

□ 0 votes

Mrunal Khinvasara 6 février 2018 at 10:54

Awesome Article thanks a tonne!!!

□ 0 votes

Sanchit Agarwal 17 mai 2018 at 22:13

awesome article. loved it.

?

□ 0 votes

Jishu Dohare 24 décembre 2018 at 14:29

What a beautiful article, Bravo to the author.

□ 0 votes

Saikat Ghosh 5 février 2019 at 16:32

Very nice article explained from basics till the end.

The root function improves the more times it is called ?

□ 0 votes

Anik Saha 6 juin 2019 at 01:36

/*modified union function where we connect the elements by changing the root of one of the element */
After this shouldn't it be void union instead of int union()

□ 0 votes

Prudhvi Kiran 5 août 2019 at 20:53

Great tutorial and one can understand easily :)

□ 0 votes

Sushil Yadav □ Edited 9 janvier 2020 at 08:19

How can I print all the subsets in the most efficient way?

I know that we can use a map which will store root as key and list as value.

```
private static void printSets(int[] arr) {
Map<Integer, List<Integer>> map = new HashMap<>();
for(int i=0; i<arr.length; ++i) {
int root = find(arr, i); // uses compression
if(map.containsKey(root)) {
List<Integer> list = map.get(root);
list.add(i);
} else {
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(i);
map.put(root, list);
}
}
for(Integer key: map.keySet()) {
System.out.println(map.get(key));
}
}
```

□ 0 votes

Allen James Vinoy 9 février 2020 at 11:27

If the balanced tree formed after weighted-union is not binary then how is the complexity log base-2 N?

□ 0 votes

Swar Patel 16 février 2020 at 08:34

in weighted union we should check if(roota!=rootb) then do union otherwise not

□ 0 votes

6
LIVE EVENTS

□ **AUTHOR**

Prateek Garg

□ SDE-1 at Flipkart
□ BENGALURU
□ **7 notes**

?

TRENDING NOTES

Python Diaries Chapter 3 Map | Filter | For-else | List Comprehension
written by Divyanshu Bansal

Bokeh | Interactive Visualization Library | Use Graph with Django Template
written by Prateek Kumar

Bokeh | Interactive Visualization Library | Graph Plotting
written by Prateek Kumar

Python Diaries chapter 2
written by Divyanshu Bansal

Python Diaries chapter 1
written by Divyanshu Bansal

more ...

6
LIVE EVENTS

+1-650-461-4192

contact@hackerearth.com

Resources

Tech Recruitment Blog

Product Guides

Developer hiring guide

Engineering Blog

Developers Blog

Developers Wiki

Competitive Programming

Start a Programming Club

Practice Machine Learning

Solutions

Assess Developers

Conduct Remote Interviews

Assess University Talent

Organize Hackathons

Société

About Us

Press

Careers

Service & Support

Technical Support

Contact Us

Site Language: Français ▼

?