Disjoint Set Union (Union Find)

243 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.

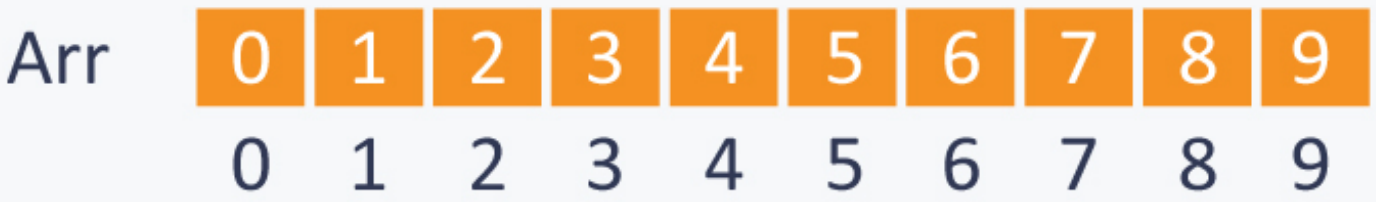
Assumption: A and B objects are connected only if Arr[A] = Arr [B].

Now how we will implement above operations :

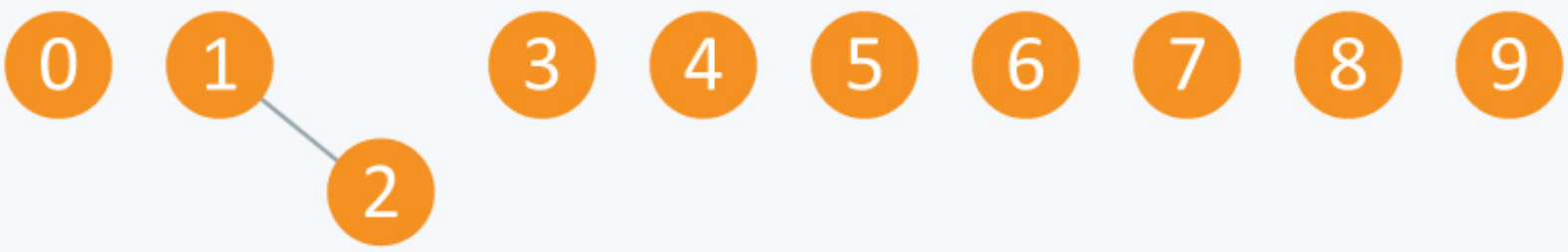
Find (A, B) - check if Arr[A] is equal to Arr[B] or not. Union (A, B) - Connect A to B and merge the components having A and B by changing all the elements ,whose value is equal to Arr[A], to Arr[B].

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

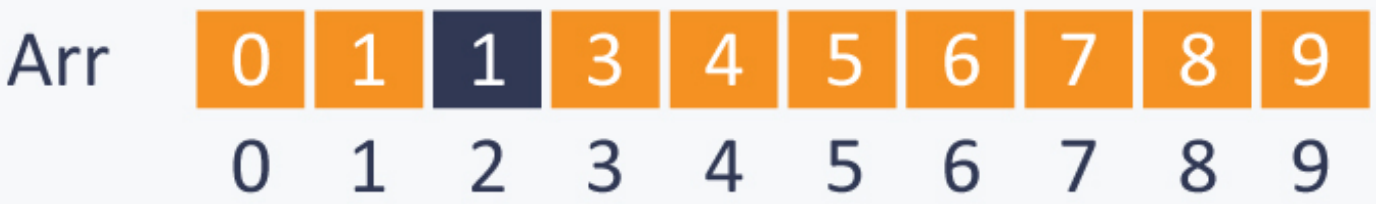
When each subset contains only single element, the array Arr is:



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



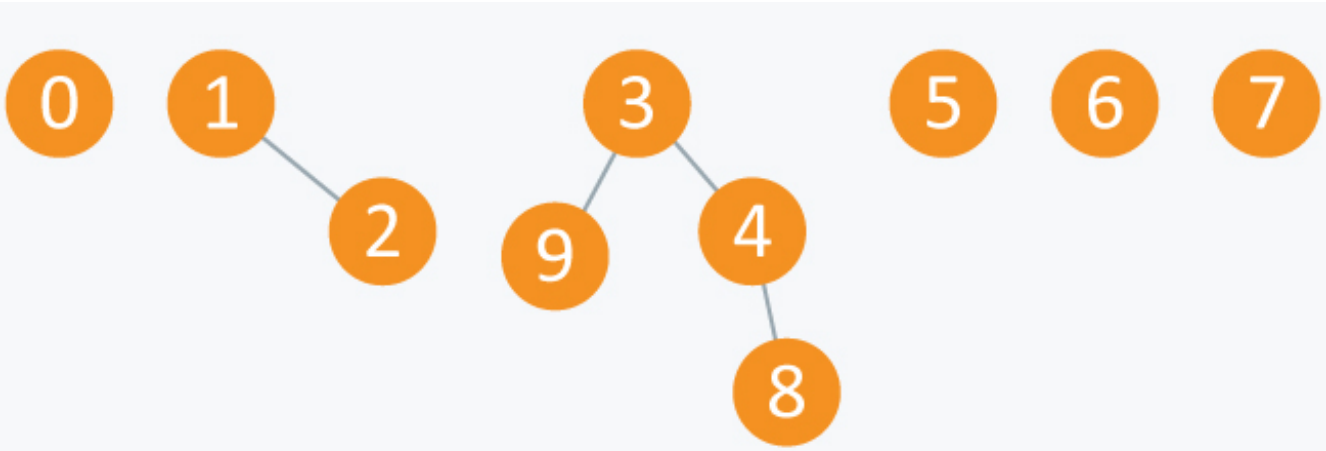
Arr will be:



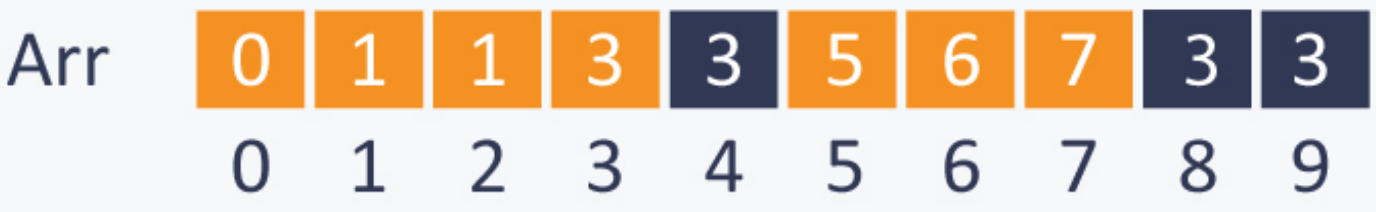
2) Union(4, 3)

3) Union(8, 4)

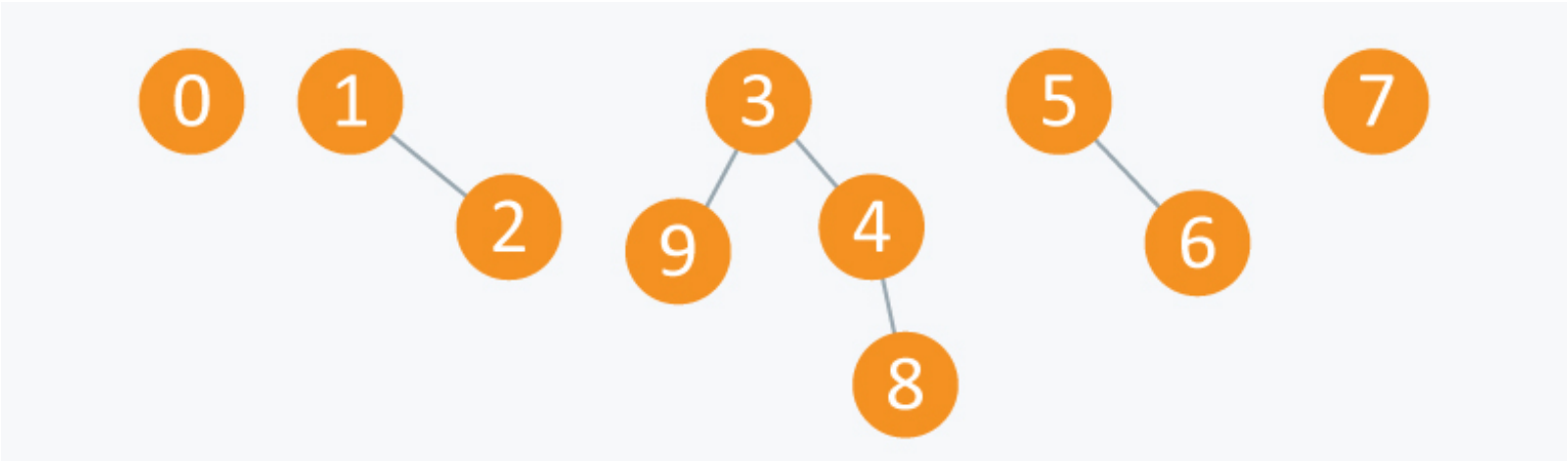
4) Union(9, 3)



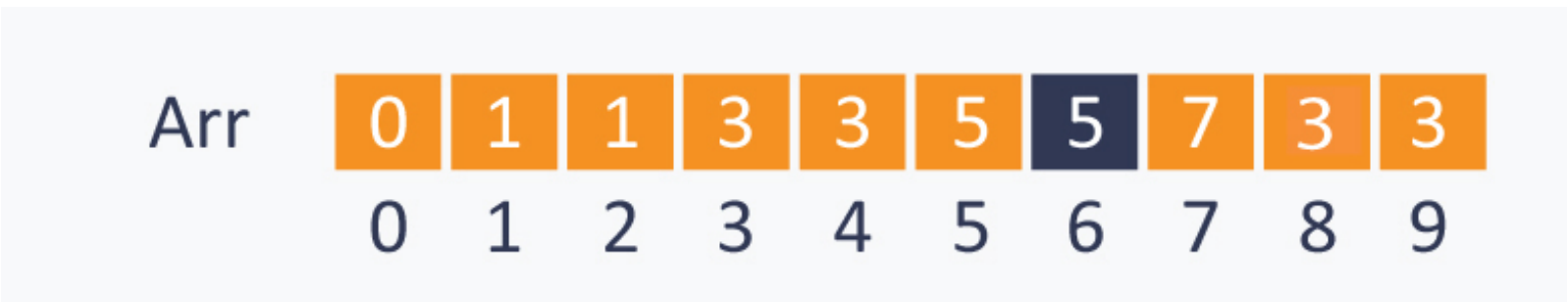
Arr will be:



5) Union(6, 5)



Arr will be:



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.**

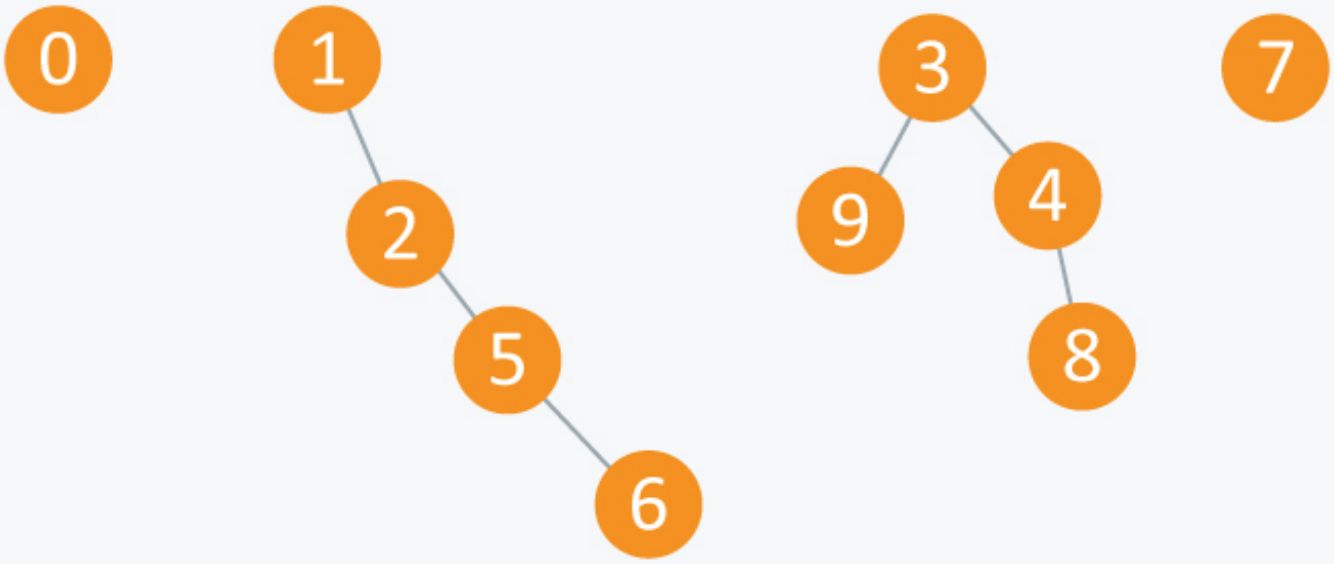
From this, we can infer that Union-Find data structure is useful in Graphs for performing various operations like connecting nodes, finding connected components etc.

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.

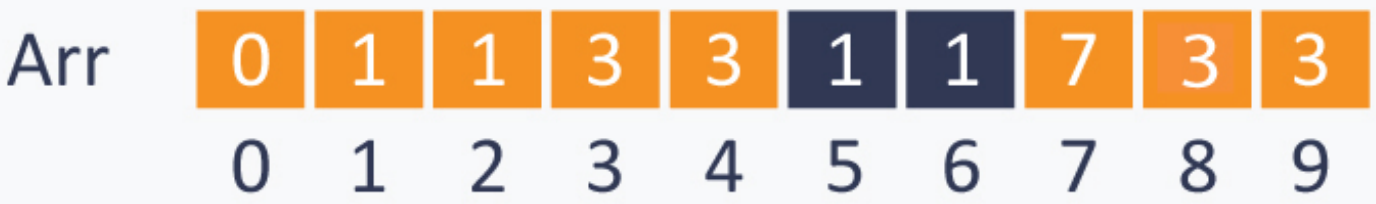
When we see above operations in terms of components, then :

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:



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:

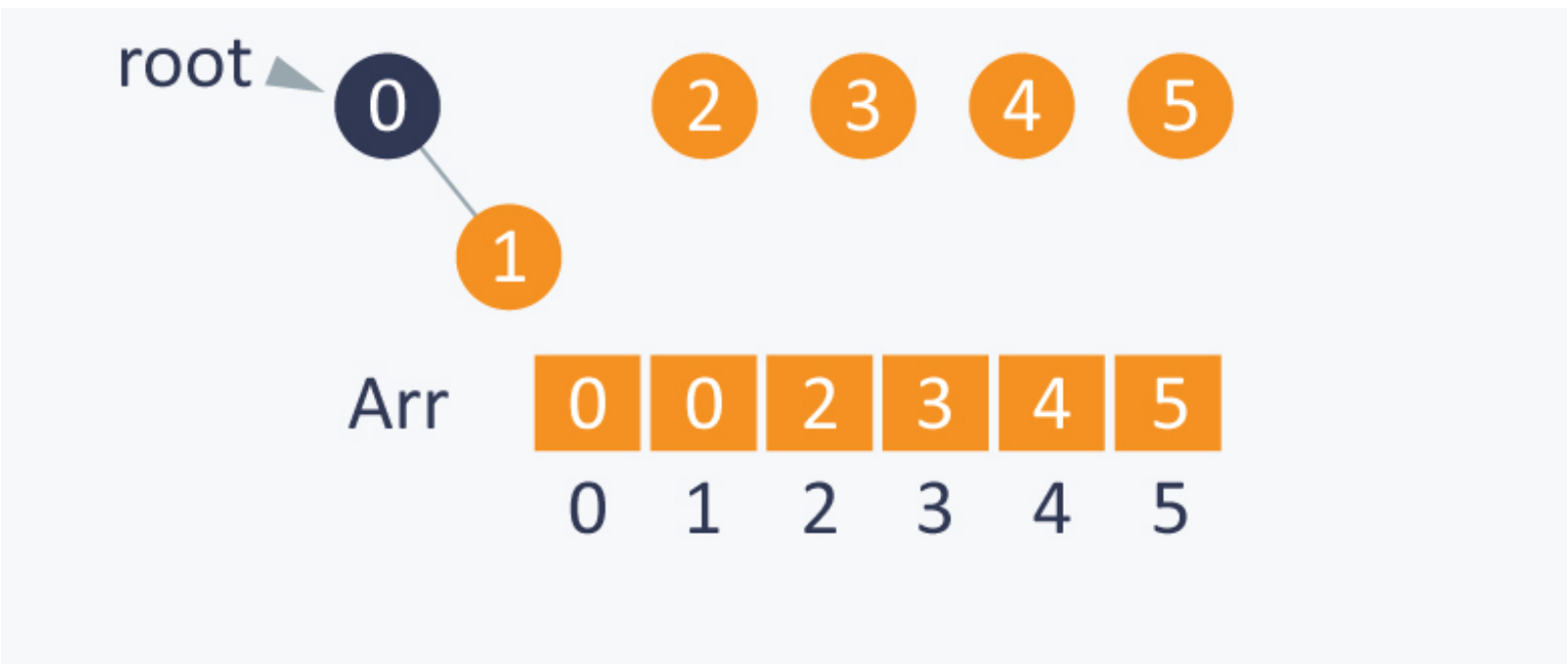
Idea: Arr[A] is a parent of A.

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.

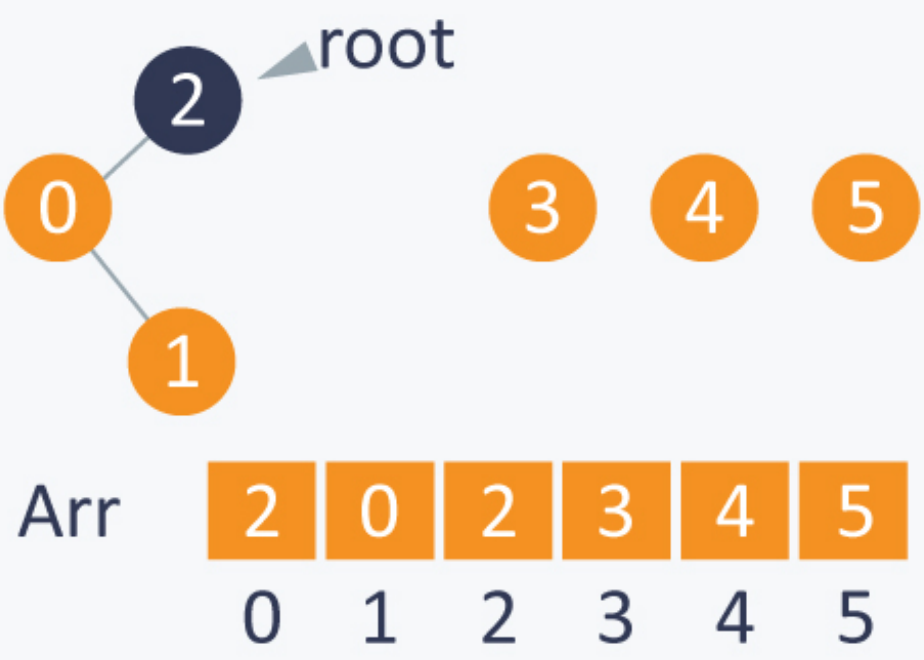
To make it more clear,let's take a subset S = {0, 1, 2, 3, 4, 5}

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.

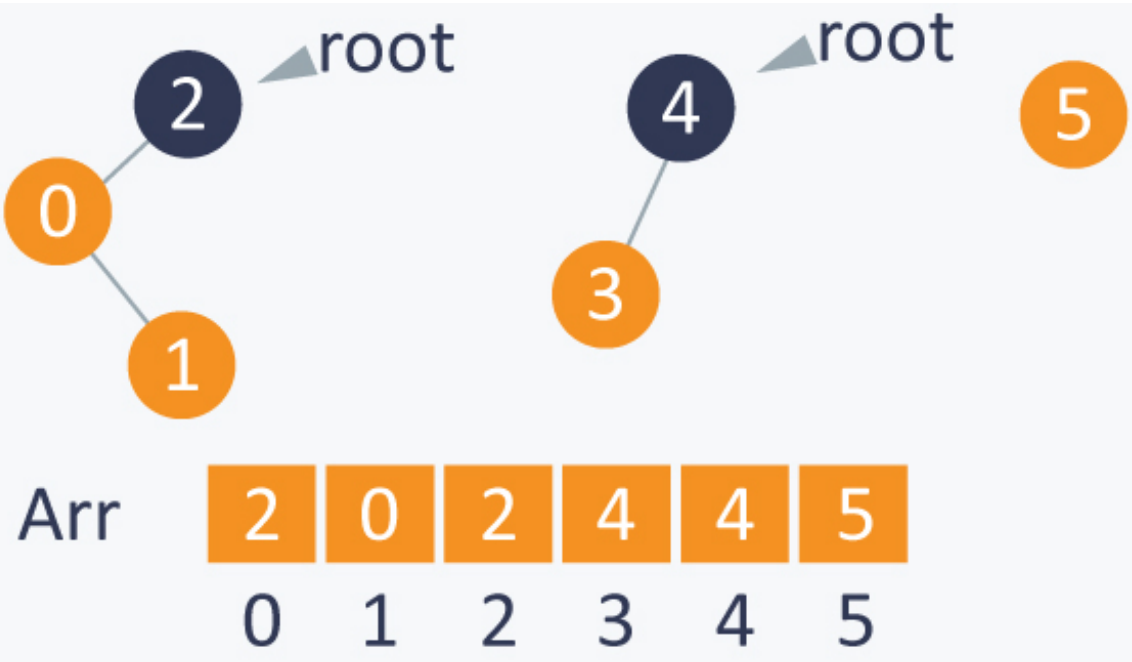
Performing Union(1, 0) will connect 1 to 0 and will set root(0) as the parent of root(1). As root(1) = 1, and root(0) = 0, therefore value of Arr[1] will be changed from 1 to 0. It will make 0 as a root of subset containing elements {0, 1}.



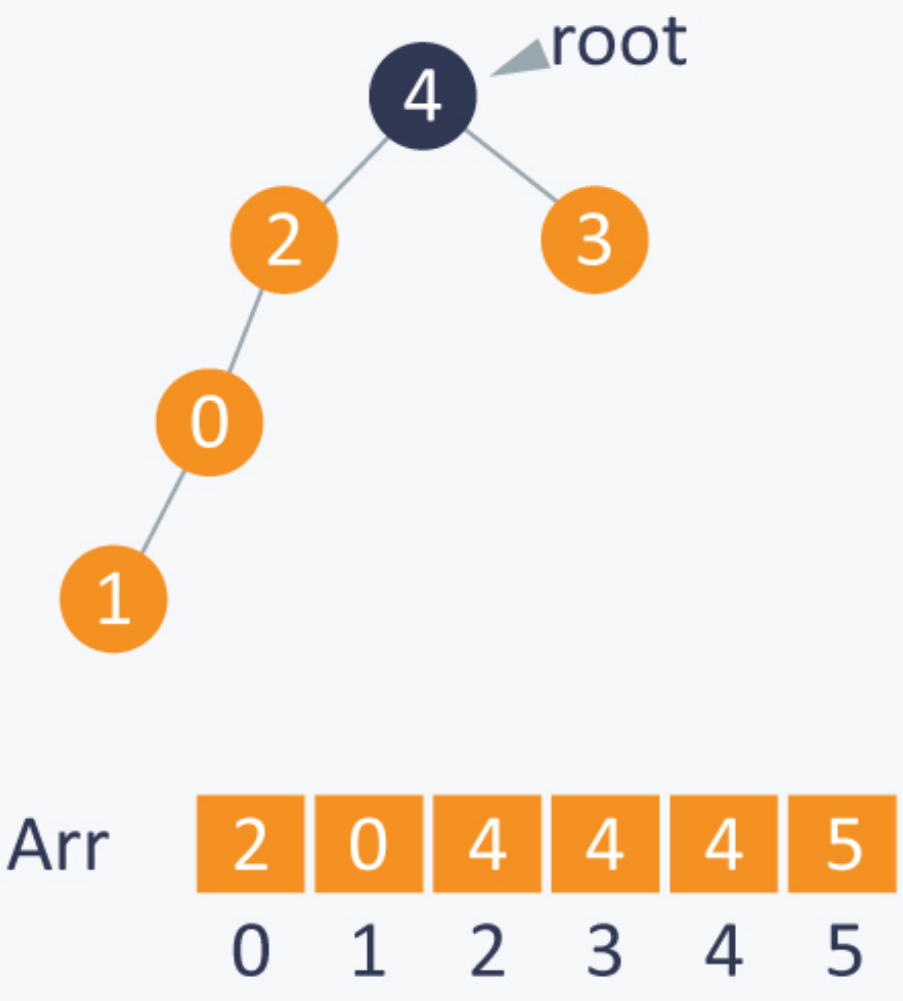
Now performing Union(0, 2), will indirectly connect 0 to 2, by setting root(2) as the parent of root(0). As root(0) is 0 and root(2) is 2, therefore it will change value Arr[0] from 0 to 2. Now 2 will be the root of subset containing elements {2, 0, 1}.



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;
}
```

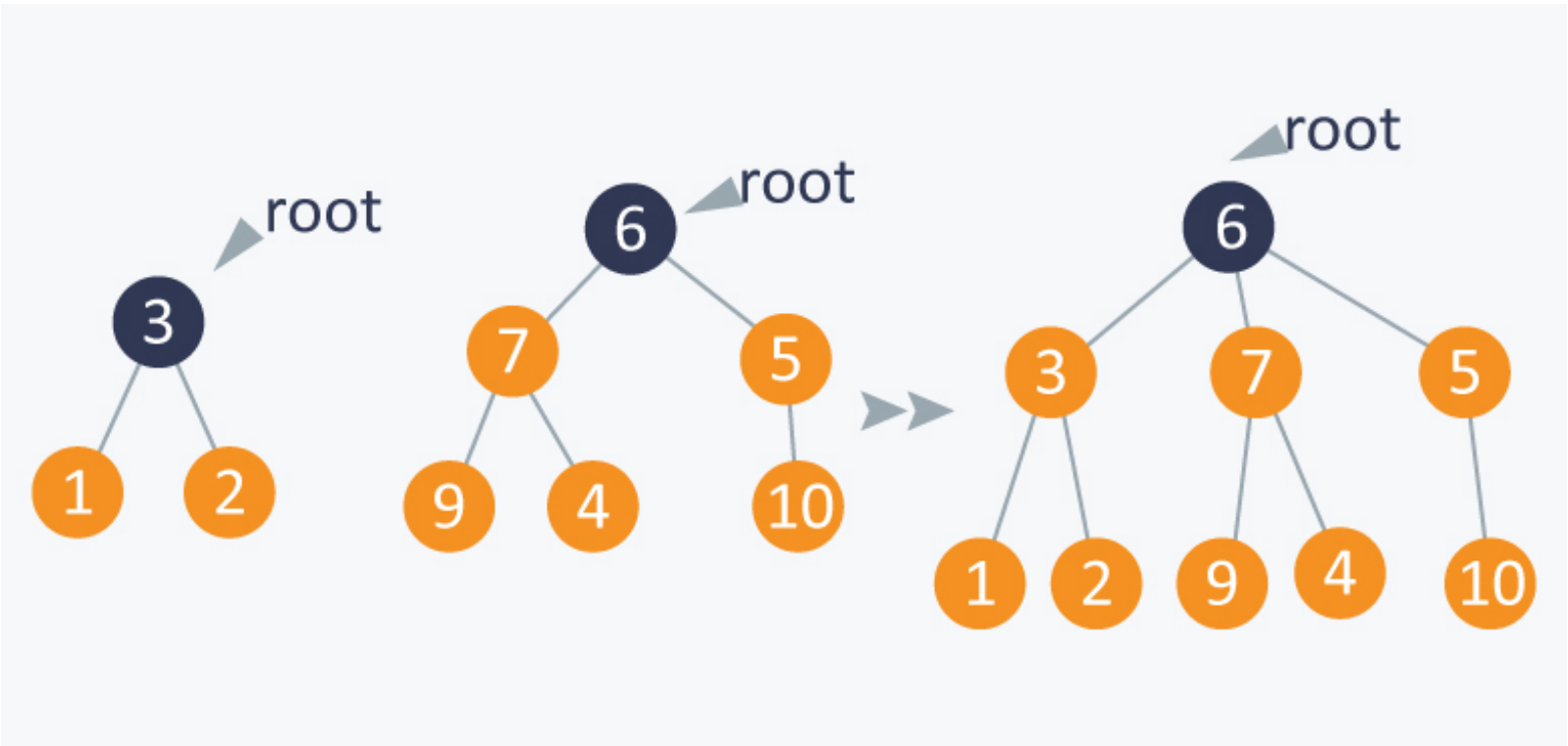
Now as you can see, in worst case, this idea will also take linear time in connecting 2 elements and even in finding that if two elements are connected or not, it will take linear time.
Another disadvantage is that while connecting two elements, we do not check which subset has more element than other 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);
```

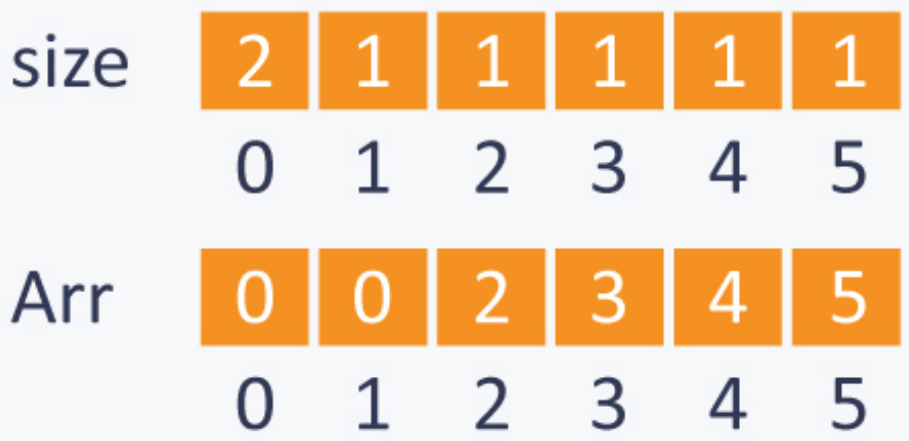
```
    int root_B = root(B);
    if(size[root_A] < size[root_B ])
    {
Arr[ root_A ] = Arr[root_B];
size[root_B] += size[root_A];
}
    else
    {
Arr[ root_B ] = Arr[root_A];
size[root_A] += size[root_B];
}

}
```

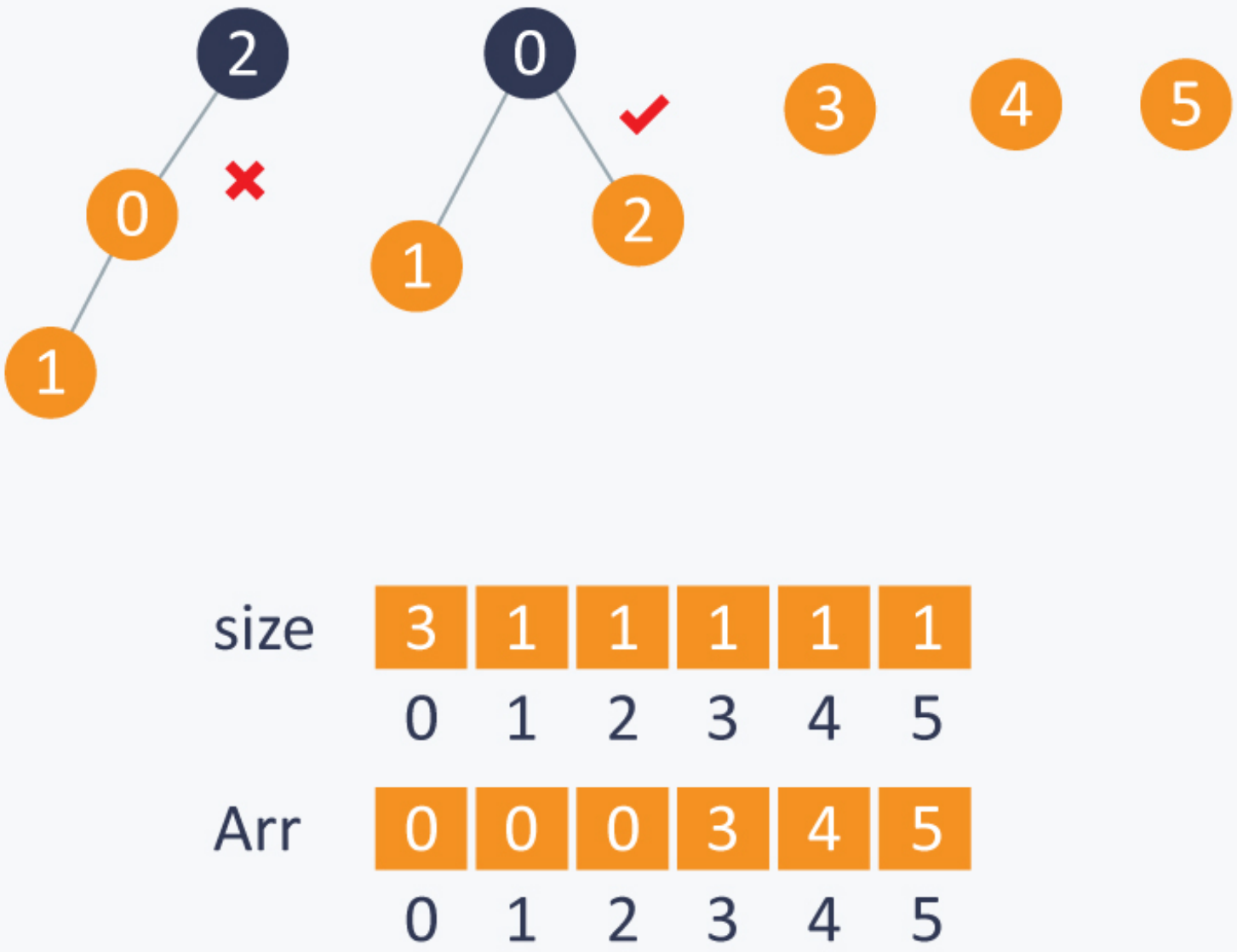
Example:

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 :

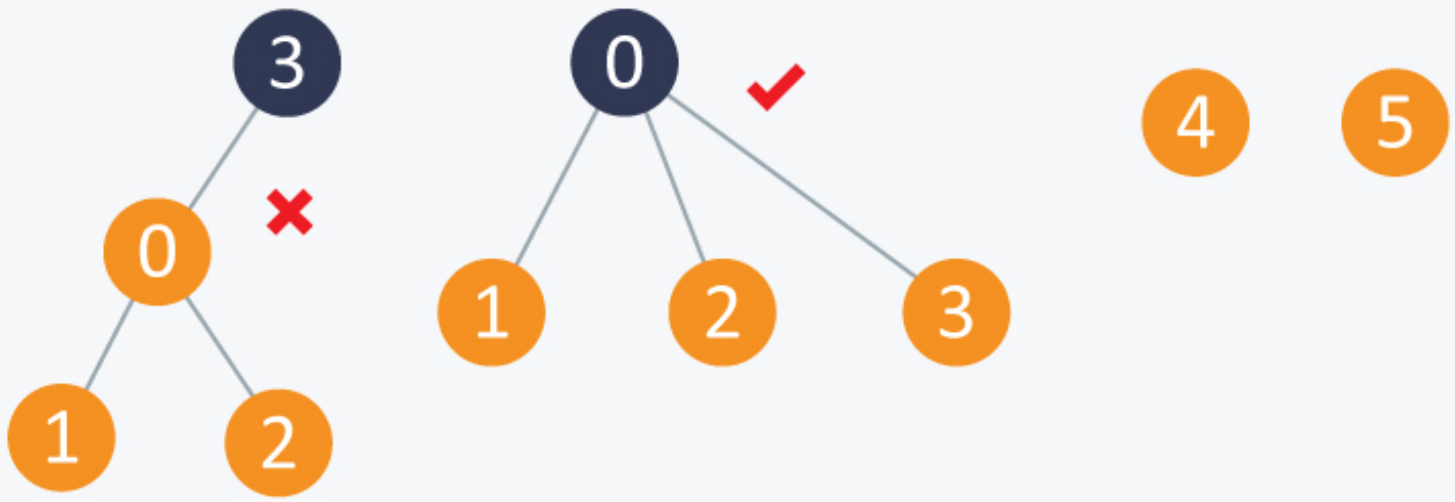
Perform Union(0, 1). Here we can connect any root of any element with root of other one as both the element's subsets have same size and then we will update the respective size. If we connect 1 to 0 and make 0 as a root and then size of 0 will change from 1 to 2.



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.



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.



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

COMMENTS (75) SORT BY: Relevance

Join Discussion...

Cancel Post

Naivedya Bansal 4 years ago

What is log *n exactly ? How did the complexity become log *n? Please explain.

4 votes · Reply · Message · Permalink

Pulkit Gupta 4 years ago

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)

32 votes • Reply • Message • Permalink

Prateek Garg Author 4 years ago

Yes, pulkit is right.

4 votes • Reply • Message • Permalink

Naivedya Bansal 4 years ago

Thanks :)

3 votes • Reply • Message • Permalink

Arun Prasad 4 years ago

```
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];
}
}
```

10 votes • Reply • Message • Permalink

Harsh Paliwal 2 years ago

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

0 votes • Reply • Message • Permalink

Shivam Tanay 4 years ago

great Article :P

9 votes • Reply • Message • Permalink

Sarvagya Agarwal 4 years ago

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

3 votes • Reply • Message • Permalink

Reeshabh Kumar Ranjan a year ago

https://www.hackerearth.com/users/prat33k/" target="_blank">@Prateek Garg please clear this doubt.

0 votes • Reply • Message • Permalink

Reeshabh Kumar Ranjan a year ago

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 • Reply • Message • Permalink

Naivedya Bansal 4 years ago

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 • Reply • Message • Permalink

dinkar gahoi 3 years ago

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 • Reply • Message • Permalink

Fubuki 2 years ago

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 • Reply • Message • Permalink

Nipun Mittal a year ago

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 • Reply • Message • Permalink

akash kandpal 10 months ago

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

0 votes • Reply • Message • Permalink

SOUMYA GHOSH 4 years ago

//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 • Reply • Message • Permalink

Prateek Garg Author 4 years ago

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 • Reply • Message • Permalink

SOUMYA GHOSH 4 years ago

So we could have done it recursively too?

0 votes • Reply • Message • Permalink

farshid nooshi 2 years ago

cool

1 vote • Reply • Message • Permalink

Ashwani Gautam 4 years ago

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

0 votes • Reply • Message • Permalink

Prateek Garg Author 4 years ago

Yes. It is fixed. Thanks :)

0 votes • Reply • Message • Permalink

Ravi Shankar 4 years ago

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 • Reply • Message • Permalink

Prateek Garg Author 4 years ago

Fixed. Thanks.

0 votes • Reply • Message • Permalink

Amar Kaswan 4 years ago

[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 • Reply • Message • Permalink

Prateek Garg Author 4 years ago

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

0 votes • Reply • Message • Permalink

Amar Kaswan 4 years ago

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 • Reply • Message • Permalink

Prateek Garg Author 4 years ago

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 • Reply • Message • Permalink

Amar Kaswan 4 years ago

Thank you got it :)

0 votes • Reply • Message • Permalink

Ramit Das 4 years ago

Now how do we get back the subsets?

0 votes • Reply • Message • Permalink

Prateek Garg Author 4 years ago

Sorry, I didn't get you.

0 votes • Reply • Message • Permalink

Ramit Das 4 years ago

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 • Reply • Message • Permalink

Prateek Garg Author 4 years ago

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

0 votes • Reply • Message • Permalink

Pulkit Mendiratta 4 years ago

Amazing Article :)

0 votes • Reply • Message • Permalink

Prateek Garg Author 4 years ago

Thanks :)

0 votes • Reply • Message • Permalink

SOUMYA GHOSH 4 years ago

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

0 votes • Reply • Message • Permalink

Prateek Garg Author 4 years ago

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 • Reply • Message • Permalink

jayasurya j 3 years ago

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 • Reply • Message • Permalink

Mohamed Ayman 3 years ago

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

0 votes ● Reply ● Message ● Permalink

Arun Prasad 4 years ago

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 ● Reply ● Message ● Permalink

Prateek Garg Author 4 years ago

This comment has been deleted.

Reply ● Message ● Permalink

Arun Prasad 4 years ago

Okay :)

0 votes ● Reply ● Message ● Permalink

Shubham Tandle 4 years ago

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

0 votes ● Reply ● Message ● Permalink

sathiyaseelan 4 years ago

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 ● Reply ● Message ● Permalink

Pulkit Gupta 4 years ago

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

0 votes ● Reply ● Message ● Permalink

Aditya Sharma 4 years ago

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 ● Reply ● Message ● Permalink

Abhishek Chakraborty 4 years ago

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

0 votes ● Reply ● Message ● Permalink

Preeti Nagal 4 years ago

It's amazing.....

0 votes • Reply • Message • Permalink

hardik agrawal 4 years ago

very nicely written and explained! thanks.. :)

0 votes • Reply • Message • Permalink

Dhruv Kaushik 4 years ago

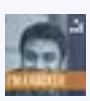
Great tutorial. But I think that the find function can be more optimized in a specific case. Suppose A and B are both present on same branch, then if while going from one element to its root (to find its root), if we encounter the second element in the path, then we can directly output yes and terminate 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 • Reply • Message • Permalink

Banipreet Singh Raheja 2 years ago

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 • Reply • Message • Permalink



Sandeep Ravindra 4 years ago

Nice article!

0 votes • Reply • Message • Permalink

Paras Avkirkar 4 years ago

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 • Reply • Message • Permalink

Nilesh Hirani 4 years ago

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

0 votes • Reply • Message • Permalink

Nilesh Hirani 4 years ago

Sorry I meant root() function

0 votes • Reply • Message • Permalink

Abhimanyu Singh Shekhawat 3 years ago

Very nice work!

0 votes • Reply • Message • Permalink

karandeep singh dhillon 3 years ago

we can use dfs also to implement this?

0 votes • Reply • Message • Permalink

Pintu Das 3 years ago

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 • Reply • Message • Permalink

Shubho Shaha Edited 3 years ago

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 • Reply • Message • Permalink

Rohit 3 years ago

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 • Reply • Message • Permalink

Vivek Singh 3 years ago

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

0 votes • Reply • Message • Permalink

Abhishek Agrawal 3 years ago

Good article mate!

0 votes • Reply • Message • Permalink

Psicodelico Demonio del Piano 2 years ago

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 • Reply • Message • Permalink

Fubuki 2 years ago

You should not do this , otherwise you can go to jail cause of the copyright !

0 votes • Reply • Message • Permalink

Bedir Tapkan Edited 2 years ago

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

0 votes • Reply • Message • Permalink

Sumit Saurav 2 years ago

Nice Article :_^_

0 votes • Reply • Message • Permalink

Aneesh Hiregange 2 years ago

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

0 votes • Reply • Message • Permalink

Rohit Rajak 2 years ago

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 • Reply • Message • Permalink

Engin öztürk 2 years ago

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 • Reply • Message • Permalink

ak3899 2 years ago

great tutorial on DSU!!

0 votes • Reply • Message • Permalink

Mrunal Khinvasara 2 years ago

Awesome Article thanks a tonne!!!

0 votes • Reply • Message • Permalink

Sanchit Agarwal a year ago

awesome article. loved it.

0 votes • Reply • Message • Permalink

Jishu Dohare 8 months ago

What a beautiful article, Bravo to the author.

0 votes • Reply • Message • Permalink

Saikat Ghosh 7 months ago

Very nice article explained from basics till the end.

The root function improves the more times it is called ?

0 votes • Reply • Message • Permalink

Anik Saha 3 months ago

/*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 • Reply • Message • Permalink

Prudhvi Kiran 21 days ago

Great tutorial and one can understand easily :)

0 votes • Reply • Message • Permalink

AUTHOR

Prateek Garg

SDE-1 at Flipk...

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 ...