Disjoint Sets ADT

1. Quick Union (Slow Find)

Makeset()

Find() // without path compression

Union()

*#include*<bits/stdc++.h>

using *namespace* std;

*int\** makeset(*int* size)

{

*int* \*temp = new *int*[size];

*for*(*int* i=0; i<size; i++)

    {

        temp[i] = i;

    }

*return* temp;

}

*int* find(*int* *\**arr, *int* size, *int* ele)

{

*if*(ele<=0 || ele>=size)

    {

*return* -1;

    }

*if*(arr[ele] == ele)

    {

*return* ele;

    }

*return* find(arr, size, arr[ele]);

}

*void* Union(*int* *\**arr, *int* size, *int* ele1, *int* ele2)

{

*if*(find(arr, size, ele1) == find(arr, size, ele2))

    {

*return*;

    }

    arr[ele1] = arr[ele2];

}

*int* ele\_in\_same\_set(*int* *\**arr, *int* size, *int* ele1, *int* ele2)

{

*int* a = find(arr, size, ele1);

*int* b = find(arr, size, ele2);

*if*( a == b)

    {

*return* a;

    }

*else*{

*return* -1;

    }

}

*void* print(*int* *\**arr, *int* size)

{

*for*(*int* i=0 ;i<size; i++)

    {

        cout<<arr[i]<<" ";

    }

}

*int* main()

{

*int* \*arr = makeset(10);

    Union(arr,10, 2, 3);

    Union(arr, 10, 4,5);

    Union(arr, 10, 1, 3);

    print(arr, 10);

    cout<<endl<<find(arr, 10, 4);

    cout<<endl<<ele\_in\_same\_set(arr, 10, 1, 9);

*return* 0;

}

1. Quick Union (Fast Find)

Makeset()

Find\_by\_compression()

Union\_by\_size()

Union\_by\_height()

*#include*<bits/stdc++.h>

using *namespace* std;

*void* makeset(*int* *\**parent, *int* *\**rank, *int* s)

{

*for*(*int* i=0; i<s; i++)

    {

        parent[i] = i;

        rank[i] = 1;

    }

}

*int* find\_by\_path\_compression(*int* *\**parent, *int* s, *int* ele)

{

*if*(ele<0 || ele>=s)

    {

*return* -1;

    }

*if*(parent[ele]  == ele)

    {

*return* ele;

    }

*else*

    {

*return* (parent[ele] = find\_by\_path\_compression(parent, s, parent[ele]));

    }

}

*void* union\_by\_height(*int* *\**parent, *int* *\**rank, *int* s,*int* ele1, *int* ele2)

{

*int* root1 = find\_by\_path\_compression(parent, s, ele1);

*int* root2 = find\_by\_path\_compression(parent, s, ele2);

    cout<<root1<<" "<<rank[root1]<<" "<<root2<<" "<<rank[root2]<<endl;

*if*(( root1 == root2) || (root1==-1 || root2==-1)) *//both values are equal and not in same set*

    {

*return*;

    }

*if*(rank[root2] < rank[root1])

    {

        parent[root2] = root1;

    }

*else* *if* (rank[root1] < rank[root2]){

        parent[root1] = root2;

    }

*else*{

        parent[root1] = root2;

        rank[root2]++;

    }

}

*void* union\_by\_weight(*int* *\**parent, *int* *\**rank, *int* s, *int* ele1, *int* ele2)

{

*int* root1 = find\_by\_path\_compression(parent, s, ele1);

*int* root2 = find\_by\_path\_compression(parent, s, ele2);

    cout<<"root 1: "<<root1<<" size:"<<rank[root1]<<" root2: "<<root2<<" size:"<<rank[root2]<<endl;

*if*((root1 ==root2) || (root1==-1 || root2==-1) )

    {

*return*;

    }

*if*(rank[root1] < rank[root2])

    {

        parent[root1] = parent[root2];

        rank[root2] += rank[root1];

    }

*else*

    {

        parent[root2] = parent[root1];

        rank[root1] += rank[root2];

    }

}

*void* print(*int* *\**parent, *int* *\**rank, *int* s)

{

    cout<<"index:   ";

*for*(*int* i=0 ;i<s; i++)

    {

        cout<<i<<" ";

    }

    cout<<endl;

    cout<<"parent:  ";

*for*(*int* i=0 ;i<s; i++)

    {

        cout<<parent[i]<<" ";

    }

    cout<<endl;

    cout<<"size:    ";

*for*(*int* i=0 ;i<s; i++)

    {

        cout<<rank[i]<<" ";

    }

    cout<<endl;

}

*int* main()

{

*int* s=6;

*int* \*parent, \*rank;

    parent = new *int*[s];

    rank = new *int*[s];

    makeset(parent , rank, s);

    print(parent, rank, s);

*//---------Union by height rank on basis of height*

    union\_by\_height(parent, rank, s, 1,2); *// Attach 1 to 2*

    union\_by\_height(parent, rank, s, 0, 3);

    union\_by\_height(parent, rank, s, 1, 3);

    print(parent, rank, s);

*//---------- Union by weight (rank on basis of size*

    union\_by\_weight(parent, rank,s, 0,1); *//Attach 2 to 1*

    union\_by\_weight(parent, rank,s, 1,2);

    union\_by\_weight(parent, rank,s, 3,2);

    print(parent, rank, s);

*// union\_by\_weight(arr, size, 3,0);*

*return* 0;

}