```
    for (int i=0; i<n; i++){
loop ↲   parent [i] = i;
    }    size (i] =1;}
```

Initializes an empty union- find data structure with n elements Through n-1 initially, e ach element is in its own set. n is n- umber of elements.

```
public int Count () {
    return count; }
```

Retuon the number of sets ; between 1 and n.

```
public int find (intP) {
    validate (P);
  while (P!= parent [P])
    P= parent [P];
    return P; }
```

It retuons The cononical element of The set containing element p.

DATE: ASSIGNMENT # 4 (M) (T) (W) (T) (F) (S)

-1 Anatomy of weighted Quick-Union

```
public class weighted Quick
Union UF {
```

The weighted Guck union UF class represent a union-find data Type. In support classic union find operations along with a count operation that returns The total number of sets.

```
private int [] parent;
```

Parent [i] = Parent of i

```
Private int [] size;
```

size [i] = number of elements is subtree rooted it i

```
Private int count;
```

to count number of components

```
public weighted Quick union UF (int n) {
    count = n;
parent = new int [n];
 size = new int [n];
```

parameter p an element Throws illegal argument exception unless 0<=p<n

@ Depreciated

```
public boolean connected (int p, int q) {
    return find (p) = find (q); }
```

p one element

q The other element

It return true if p and q are in The same set. if not it returns false Throws illegal arguments exception unless both 0<=pn and 0<=q<n

Depreciated => Replace with two cell to (find (int))

```
private void validates (int p) {
    int n= parent . length;
  if (p<0 || p >=n) {
Throw new illegal Argument
Exception ("index"+' is not
        between 0 and"+(n-1); }
}
```

It is used to validate That p is a valid index.

void does not return value.

int ⇒ is a a data Type.

used a loop to check validation of P.

```
Public void union (intp, int q) {
    int root p = find (p);
    int root q = find (q);
    if (root p == root q) return;
```

Merges The set containing element p with The set containing element q.

P⇒ one element

q⇒ The other element

Throw illegal argument exception

unless both $0 <= p < n$ and $0 <= q < n$

```
if (size [root p] < size [root Q] {
    parent [root p] = root Q;
   Size [root Q] += size [root p] ;
 }
else {
   parent [root Q] = root p;
    Size [root p] += size [root Q];
 }
    Count --;
}
```

In This roots are being compare to each other. smaller roots are being compared to longer and large with smaller root This size code wake smaller root point to large one

```
Public static void main (string [] arg
int n = StdIn.read Int ();
```

weighted Quick union UF uf= new weighted Quick union UF (int).

```
  while (!StdIn. is Empty () {
   int p= StdIn. read Int ();
   int q = Stdin . read Int ();
  if (uf. find (p)= = uf. find (q) continue;
   uf .union (p,q);
   StdOut . print (n(p+" "+q); }
   Stdout . print In (uf. count()+ "count");
  }
 }
```

Reads an integer n and a sequence of pairs of integers between a and n-1 from standard input, where each integer in the pair represents Some element. If the elements are in diff sets. merge two sets and print to standard output.