Union-Find Problem

Given a set of N objects

- Union: connect two objects
- find: Is there a path connecting the two objects?

Efficiency

Algorithm	Init	Union	Find
Quick-Find(Eager)	N	N	1
Quick-Union(Lazy)	N	N	N ← (Worst-Case)
Weighted quick-union	N	lg N	lg N

Further Improvements:

- Bottleneck is root()
- Quick-union with path compression

Eager Approach (Quick-find)

Data Structure

- Object: integer array **id[]** of length **N**
- Interpretation: **id[i]** is the component to which **i** belongs
- Connections: **u** and **v** are connected iff they have the same **id**

In [17]:

```
# Union-Find Eager approach (Quick Find)
class UnionFind_Eager:

    def __init__(self,N):
        self.numNode = N
        self.idArray = []
        for i in range(self.numNode):
            self.idArray.append(i)

    def union(self, u,v):
        uid = self.idArray[u]
        vid = self.idArray[v]
        for i in range(self.numNode):
            if self.idArray[i] == uid:
                self.idArray[i] = vid

    def find(self, u,v):
        return self.idArray[u] == self.idArray[v]
```

In [18]:

```
## Test Code
N = 9
myEager = UnionFind_Eager(N)
```

```
myEager.union(0,5)
myEager.union(1,2)
myEager.union(5,6)
myEager.union(2,7)
myEager.union(8,3)
myEager.union(3,4)
myEager.union(4,8)

print(myEager.idArray)

## Test Find
print(f"0, 5 -> {myEager.find(0,5)}")
print(f"0, 6 -> {myEager.find(0,6)}")
print(f"5, 6 -> {myEager.find(5,6)}")
print(f"1, 7 -> {myEager.find(1,7)}")
print(f"8, 3 -> {myEager.find(8,3)}")

print(f"3, 8 -> {myEager.find(3,8)}")

## False case
print(f"0, 1 -> {myEager.find(0,1)}")
```

```
[6, 7, 7, 4, 4, 6, 6, 7, 4]
0, 5 -> True
0, 6 -> True
5, 6 -> True
1, 7 -> True
8, 3 -> True
3, 8 -> True
0, 1 -> False
```

Lazy Approach (Quick-Union)

Data Structure

- Object: integer array **id[]** of length **N**
- Interpretation: **id[i]** is the parent of **i**
- Connections: **u** and **v** are connected iff they have the same **root**

In [19]:

```
# Union-Find Lazy approach (Quick Union)
class UnionFind_lazy:

    def __init__(self,N):
        self.numNode = N
        self.idArray = []
        for i in range(self.numNode):
            self.idArray.append(i)

    def root(self, i):
        while i != self.idArray[i]:
            i = self.idArray[i]
        return i

    def union(self, u,v):
        r_u = self.root(u)
        r_v = self.root(v)
        self.idArray[r_u] = r_v

    def find(self, u,v):
        return self.root(u) == self.root(v)
```

In [20]:
```
## Test Code
N = 9
myLazy = UnionFind_lazy(N)

myLazy.union(0,5)
myLazy.union(1,2)
myLazy.union(5,6)
myLazy.union(2,7)
myLazy.union(8,3)
myLazy.union(3,4)
myLazy.union(4,8)

print(myLazy.idArray)

## Test Find
print(f"0, 5 -> {myLazy.find(0,5)}")
print(f"0, 6 -> {myLazy.find(0,6)}")
print(f"5, 6 -> {myLazy.find(5,6)}")
print(f"1, 7 -> {myLazy.find(1,7)}")
print(f"8, 3 -> {myLazy.find(8,3)}")

print(f"3, 8 -> {myLazy.find(3,8)}")

## False case
print(f"0, 1 -> {myLazy.find(0,1)}")
```

```
[5, 2, 7, 4, 4, 6, 6, 7, 3]
0, 5 -> True
0, 6 -> True
5, 6 -> True
1, 7 -> True
8, 3 -> True
3, 8 -> True
0, 1 -> False
```

Weighted Quick-Union

Idea

- Modify quick-union to avoid tall trees
- Additional data structure: Keep track of **size** of each tree
- Union: link smaller tree to root of larger tree

Data Structure

- Object: integer array **id[]** of length **N**, with **id[i]** being the **parent of i**
- Additional array **size[]**, with **size[i]** being the number of objects in the tree rooted in **i**

In [15]:
```
# Union-Find Weighted quick-union
class UnionFind_weighted_lazy:

    def __init__(self,N):
        self.numNode = N
        self.idArray = []
        self.sizeArray = []
        for i in range(self.numNode):
            self.idArray.append(i)
            self.sizeArray.append(i)

    def root(self, i):
        while i != self.idArray[i]:
            i = self.idArray[i]
        return i

    def union(self, u,v):
        r_u = self.root(u)
        r_v = self.root(v)
        if r_u == r_v:
            return
        if self.sizeArray[r_u] < self.sizeArray[r_v]:
            self.idArray[r_u] = r_v
            self.sizeArray[r_v] += self.sizeArray[r_u]
        else:
            self.idArray[r_v] = r_u
            self.sizeArray[r_u] += self.sizeArray[r_v]

    def find(self, u,v):
        return self.root(u) == self.root(v)
```

In [16]:

```
## Test Code
N = 9
myLazyWeighted = UnionFind_weighted_lazy(N)

myLazyWeighted.union(0,5)
myLazyWeighted.union(1,2)
myLazyWeighted.union(5,6)
myLazyWeighted.union(2,7)
myLazyWeighted.union(8,3)
myLazyWeighted.union(3,4)
myLazyWeighted.union(4,8)

print(myLazyWeighted.idArray)

## Test Find
print(f"0, 5 -> {myLazyWeighted.find(0,5)}")
print(f"0, 6 -> {myLazyWeighted.find(0,6)}")
print(f"5, 6 -> {myLazyWeighted.find(5,6)}")
print(f"1, 7 -> {myLazyWeighted.find(1,7)}")
print(f"8, 3 -> {myLazyWeighted.find(8,3)}")

print(f"3, 8 -> {myLazyWeighted.find(3,8)}")

## False case
print(f"0, 1 -> {myLazyWeighted.find(0,1)}")
```

```
[5, 2, 7, 8, 8, 6, 6, 7, 8]
0, 5 -> True
0, 6 -> True
5, 6 -> True
1, 7 -> True
8, 3 -> True
3, 8 -> True
0, 1 -> False
```

In [ ]: