

Program Structures and Algorithms Spring 2023(SEC –1)

Assignment-4

Name: Naga Venkata Nishanth Sayana

NU ID: 002930970

Task:

The here is to implement height weighted quick union with path compression (HWQUPC). In this task, instead checking the weight of each tree, we would be checking the height of each tree and making the union based on it.

In the step 1 we have to implement the find(), mergeComponents(), and doPathCompression() methods.

Code Screenshots:

1)find method

```
public int find(int p) {  
    validate(p);  
    int root = p;  
    // FIXME  
    if(pathCompression == true)  
        doPathCompression(p);  
    while (root != parent[root]) {  
        root = parent[root];  
    }  
    // END  
    return root;  
}
```

/**

UFClient x

C:\Users\snvni\.jdk\openjdk-19\bin\java.exe ...

2) Merge Components

```
private void mergeComponents(int i, int j) {  
    // FIXME make shorter root point to taller one  
    if (i == j) return;  
  
    if (height[i] == height[j]) {  
        updateParent(j,i);  
        height[i] += 1;  
    }  
    else if(height[i] > height[j]){  
        updateParent(j,i);  
    }  
    else{  
        updateParent(i,j);  
    }  
    // END  
}
```

Client x

C:\Users\snvni\.jdk\openjdk-19\bin\java.exe ...

3)doPathCompression()

```
1 usage  xiaohuanlin *  
private void doPathCompression(int i) {  
    // FIXME update parent to value of grandparent  
    while(i != parent[i]){  
        parent[i] = parent[parent[i]];  
        i = parent[i];  
    }  
    // END  
}
```

UFClient x

C:\Users\snvni\.jdk\openjdk-19\bin\java.exe ...

In Step 2, we have to write a UFClient class(Union Find Client), which generates random pairs of integers between 0 to n-1, and checks if they are connected, if not it makes a union of them. I have a static

method count(int n) which takes n as an argument(No of Sites) and returns the total number of connections made(m).

```
UF_HWQUPC.java x UFClient.java x UF_HWQUPC_Test.java x
1 package edu.neu.coe.info6205.union_find;
2
3 import java.util.Random;
4
5 public class UFClient {
6
7     public static int count(int n){
8         int totalCount=0;
9         for(int i=0;i<1000;i++){
10             UF_HWQUPC uf_hwqupc=new UF_HWQUPC(n, pathCompression: true);
11             int pairCount=0;
12
13             while(uf_hwqupc.components()!=1){
14                 int[] pair=randomPair(n);
15                 pairCount++;
16                 if(!uf_hwqupc.connected(pair[0],pair[1])){
17                     uf_hwqupc.union(pair[0],pair[1]);
18                 }
19             }
20             totalCount+=pairCount;
21         }
22         return totalCount/1000;
23     }
24
25     @ 1 usage
26     private static int[] randomPair(int n){
27         Random r = new Random();
28         int p = r.nextInt(n);
29         int q = r.nextInt(n);
30         while(p == q){
31             q = r.nextInt(n);
32         }
33         return new int[]{p,q};
34     }
35 }
```

```
1 usage
@ private static int[] randomPair(int n){
    Random r = new Random();
    int p = r.nextInt(n);
    int q = r.nextInt(n);
    while(p == q){
        q = r.nextInt(n);
    }
    return new int[]{p,q};
}

public static void main(String[] args){
    int[] siteValues={100,200,400,800,1600,3200,6400,12800};

    for(int n:siteValues){
        int pairCount=count(n);

        System.out.println("No of Sites(n): "+n+" | No of connections generated(m): "+pairCount);
    }
}
```

UFClient x

C:\Users\snvni\jdk\openjdk-19\bin\java.exe ...

Output:

```
34
35 public static void main(String[] args){
36     int[] siteValues={100,200,400,800,1600,3200,6400,12800};
37
38     for(int n:siteValues){
39         int pairCount=count(n);
40
41         System.out.println("No of Sites(n): "+n+" | No of connections generated(m): "+pairCount);
42     }
43 }
44 }
```

Run: UFClient x

C:\Users\snvni\jdk\openjdk-19\bin\java.exe ...

No of Sites(n): 100 | No of connections generated(m): 258
No of Sites(n): 200 | No of connections generated(m): 586
No of Sites(n): 400 | No of connections generated(m): 1292
No of Sites(n): 800 | No of connections generated(m): 2909
No of Sites(n): 1600 | No of connections generated(m): 6364
No of Sites(n): 3200 | No of connections generated(m): 13839
No of Sites(n): 6400 | No of connections generated(m): 29911
No of Sites(n): 12800 | No of connections generated(m): 64260

Test Cases Screenshot:

The screenshot displays an IDE with two tabs: `UF_HWQUPC.java` and `UF_HWQUPC_Test.java`. The `UF_HWQUPC.java` tab is active, showing the following code:

```
78  * @return the component identifier for the component containing site {@code p}
79  * @throws IllegalArgumentException unless {@code 0 <= p < n}
80  */
81  * xiaohuanlin *
82  public int find(int p) {
83      validate(p);
84      int root = p;
85      // FIXME
86      if(pathCompression == true)
87          doPathCompression(p);
88      while (root != parent[root]) {
89          root = parent[root];
90      }
91      // END
92      return root;
93  }
94  /**
95  * Returns true if the the two sites are in the same component
```

The `UF_HWQUPC_Test.java` tab is also visible, showing the test results. The tests passed: 13 of 13 tests – 37 ms. The process finished with exit code 0.

Test Case	Duration
testIsConnected01	9 ms
testIsConnected02	6 ms
testIsConnected03	12 ms
testFind0	1 ms
testFind1	0 ms
testFind2	1 ms
testFind3	2 ms
testFind4	0 ms
testFind5	0 ms
testToString	6 ms
testConnect01	0 ms
testConnect02	0 ms
testConnected01	0 ms

Process finished with exit code 0

Observations:

Here I have taken n values starting from 100 to 12800 using doubling method and I have run the experiment for 1000 times each and calculated the average of number of connections made.

I have illustrated the results I have obtained below. Based on the values, I have been able to derive the relationship between n and m , that would be $m = (n/2) * \ln(n)$.

As we can see from the graphs below, the plot of m and $m = (n/2) * \ln(n)$ are very close.

No of Sites (n)	No of connections generated (m)	1/2 n ln(n)
100	258	230
200	586	529
400	1292	1198
800	2909	2673
1600	6364	5902
3200	13839	12913
6400	29911	28044
12800	64260	60526

No of connections generated (m) and $1/2 n \ln(n)$

