Program Structures & Algorithms

Spring 2022

Assignment No. 3 - Height-weighted Quick Union with Path Compression

Name: Tanya Shah
(NUID): 002988713

- **Tasks**

1. Implement the required methods in UF_HWQUPC.java.
   a) Updated doPathCompression() method to assign grandparent's value to the parent.
   b) Updated mergeComponents() method to point the shorter root to the taller root.
   c) Updated find() method to perform path compression.
2. Created a UF_HWQUPC_RandomPairTestClient that generates a random integer to get initial number of sites and a count() method that returns the number of connections based on the given number of sites.
3. Determine the relationship between number of objects (***n***) – **sites**; and the number of pairs generated (***m***) – **connections.**

- **Relationship Conclusion**

Data gathered from the experiments as shown in the screenshots below depicts that the relationship between number of sites(***N***) and number of connections(***M***) is:

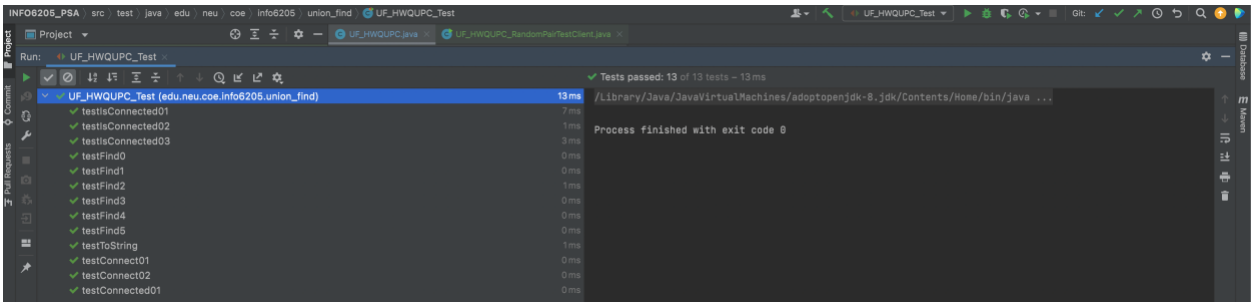
M = NlogN / 2 ***Time Complexity - O(logN)***

When we consider increasing value of number of sites, we can see the above relationship holds good.

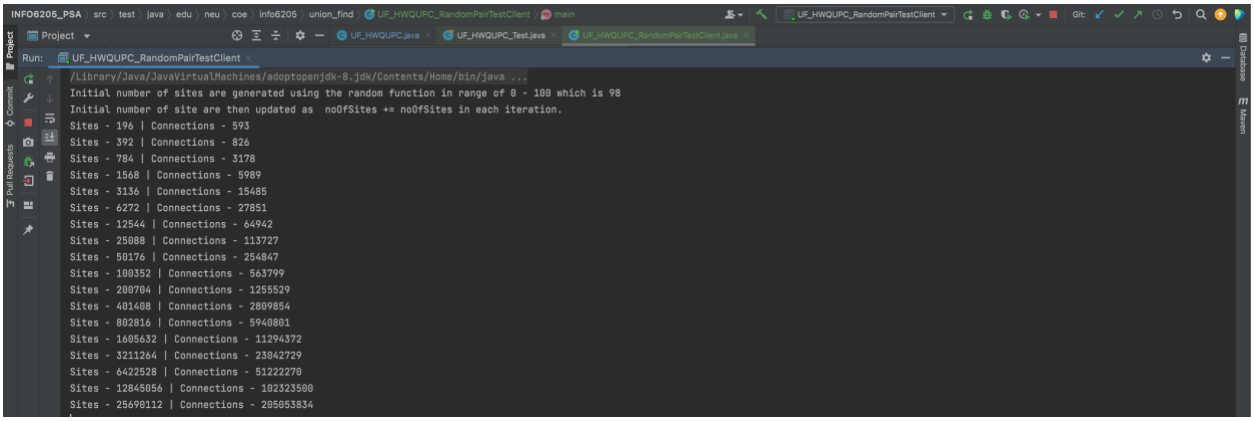
In this algorithm we can conclude that the initial operations would be expensive since we need to traverse full height of the tree and link every node to the root but it gets more cost effective when it reaches closer to the root to the point where every node is a direct descendant of the root.

- **Output screenshot**

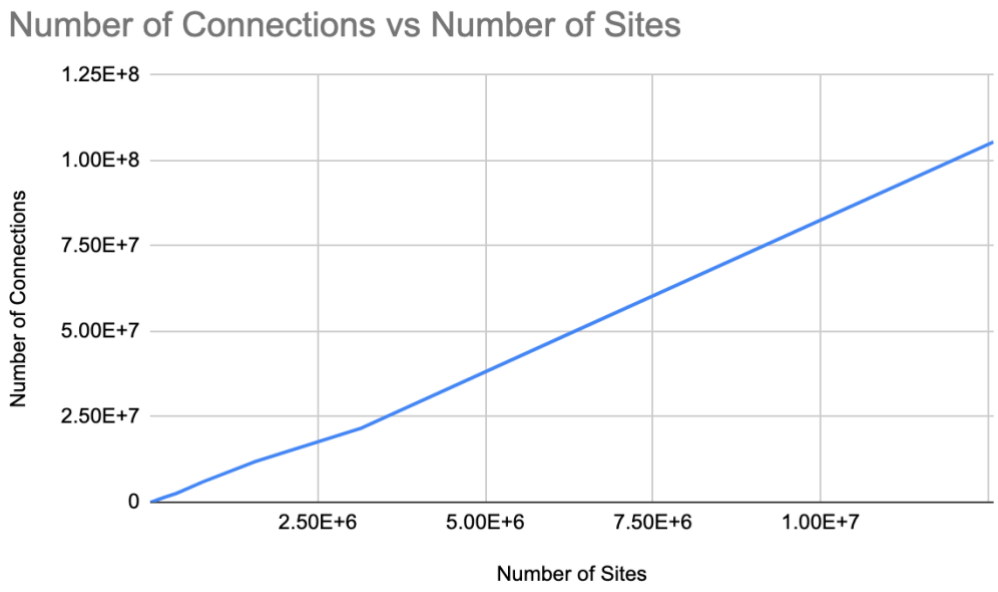
UF_HWQUPC_Test Output



UF_HWQUPC_RandomPairTestClient Output



- **Evidence / Graph**



- **Code Snapshots**

UF_HWQUPC_RandomPairTestClient

```
package edu.neu.coe.info6205.union_find;

import java.util.Random;

public class UF_HWQUPC_RandomPairTestClient {
    public static int getRandomInteger(int noOfSites) {
        int randomInt;
        Random random = new Random();
        if(noOfSites == 0) {
            randomInt = random.nextInt(100);
        } else {
            randomInt = random.nextInt(noOfSites);
        }
        return randomInt;
    }

    public static void main(String[] args) {
        //Random function to generate a random integer in range of 0 - 9
        int noOfSites = getRandomInteger(0);
        System.out.println("Initial number of sites are generated using the
random function in range of 0 - 100 which is " + noOfSites);
        System.out.println("Initial number of site are then updated as
noOfSites += noOfSites in each iteration.");

        for (int i = 0; i < 20; i++) {
            noOfSites += noOfSites;
            UF_HWQUPC uf = new UF_HWQUPC(noOfSites);
            int noOfConnections = count(uf, noOfSites);
            System.out.println("Sites - " + noOfSites + " | Connections - " +
noOfConnections);
        }
    }

    public static int count(UF_HWQUPC uf, int noOfSites) {
        int noOfConnections = 0;
        int i = 0, j = 0;
        while (uf.components() > 1) {
            i = getRandomInteger(noOfSites);
            j = getRandomInteger(noOfSites);
            uf.connect(i, j);
            noOfConnections++;
        }
        return noOfConnections;
    }
}
```

UF_HWQUPC Methods

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

```
private void mergeComponents(int i, int j) {
    // Make shorter root point to taller one
    int rootI = find(i);
    int rootJ = find(j);
    if (rootI == rootJ) {
        System.out.println("Same site is being accessed!");
    } else {
        if (height[rootI] > height[rootJ]) updateParent(rootJ, rootI);
        else if (height[rootI] < height[rootJ]) updateParent(rootI, rootJ);
        else {
            updateParent(rootJ, rootI);
            updateHeight(rootI,rootJ);
        }
    }
    // END
}
```

```
private void doPathCompression(int i) {
    // Update parent to value of grandparent
    parent[i] =  parent[parent[i]];
    // END
}
```