**Ram Gopal Varma Alluri**

**002196886**

**INFO 6205 Program Structures and Algorithms**

**Assignment 3**

Task 1:

 Implement height-weighted Quick Union with Path Compression

Unit Test Results:

Graphical user interface, text

Description automatically generated

Task 2:

Using your implementation of UF\_HWQUPC, develop a UF ("union-find") client that takes an integer value n from the command line to determine the number of "sites." Then generates random pairs of integers between 0 and n-1, calling connected() to determine if they are connected and union() if not. Loop until all sites are connected then print the number of connections generated. Package your program as a static method count() that takes n as the argument and returns the number of connections; and a main() that takes n from the command line, calls count() and prints the returned value.

Source Code:

package edu.neu.coe.info6205.union\_find;  
  
import java.util.Random;  
import java.util.Scanner;  
  
public class UnionFind\_Client {  
  
  
 public static void main(String[] *args*) {  
  
 Scanner sc=new Scanner(System.in);  
  
 System.out.print("Enter No of test Cases : ");  
 int testCases=sc.nextInt(),i=1;  
 while (testCases>0)  
 {  
 System.out.println("Enter No of nodes for testCase "+ i);  
 int input=sc.nextInt();  
 int generatedPairs=**count**(input);  
 System.out.println("No of nodes = "+input+" Avg Generated-Pairs = "+generatedPairs);  
 testCases--;  
 i++;  
 System.out.println();  
 }  
  
 }  
  
 private static int[] generateRandomPairs(int *n*, Random *r*)  
 {  
 return new int[]{*r*.nextInt(*n*), *r*.nextInt(*n*)};  
 }  
  
 private static int count(int *i*) {  
 // considering average of 200  
 int connections=0;  
  
 Random random=new Random();  
 for(int t=1;t<200;t++) {  
 UF\_HWQUPC client = new UF\_HWQUPC(*i*, true);  
 {  
 int uf=0;  
 int c = 0;  
 while (client.components() > 1) {  
 int[] pairs = **generateRandomPairs**(*i*, random);  
 if (!(client.connected(pairs[0], pairs[1]))) {  
 client.union(pairs[0], pairs[1]);  
 }  
 c++;  
 }  
 connections += c;  
 }  
 }  
  
 return connections/200;  
 }  
}

Output :

Text

Description automatically generated

Task 3:

Determine the relationship between the number of objects (*n*) and the number of pairs (*m*) generated to accomplish this (i.e. to reduce the number of components from *n* to 1).

Conclusion:

The result was taken by generating random pairs to union all the nodes and computing the average by repeatedly running the count method 200 times. In most cases, the number of pairs generated has a similar trend to that of C \* N ln N (where ln is the natural logarithm of N),Where C **≡** 0.5 . Based on this, it can be demonstrated that in order to reduce the number of components from N to 1**, A = C \* N ln N + Z** (where ln is the natural logarithm of N) of connections are necessary. Where C, Z is some constant , N is the number of nodes and A is the result.

Evidence:

|  |  |  |  |
| --- | --- | --- | --- |
| **Initial Nodes** | **Generated Pairs** | **1/2 n ln n where n is natural logarithm of n** |  |
| **100** | 266 | 230.25850929940500 |  |
| **200** | 604 | 529.8317366548040 |  |
| **500** | 1693 | 1553.652024605550 |  |
| **700** | 2465 | 2292.8781172651900 |  |
| **1000** | 3673 | 3453.8776394910700 |  |
| **5000** | 22967 | 21292.982978540600 |  |
| **10000** | 49503 | 46051.701859880900 |  |
| **15000** | 75399 | 72118.5411006326 |  |
| **25000** | 133654 | 126582.88879812900 |  |
| **30000** | 164291 | 154634.2899096640 |  |

Chart, line chart

Description automatically generated

Source Code ([Link](https://github.com/alluriramgopalvarma/INFO6205))