**Program Structures & Algorithms**

**Spring 2022**

**Assignment No. 3**

Name:Deepali Kasture

(NUID): 001586375

* **Task**

1. Implement height-weighted Quick Union with Path Compression.
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."
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).

* **Output screenshot**

**Text

Description automatically generated**

**Code:**

*/\*\*  
 \* Returns the component identifier for the component containing site {****@code*** *p}.  
 \*  
 \** ***@param*** *p the integer representing one site  
 \** ***@return*** *the component identifier for the component containing site {****@code*** *p}  
 \** ***@throws*** *IllegalArgumentException unless {****@code*** *0 <= p < n}  
 \*/*public int find(int p) {  
 validate(p);  
 int root = p;  
 // TO BE IMPLEMENTED  
 while (root != parent[root]) {  
 if (pathCompression) {  
 doPathCompression(root);  
 }  
 root = parent[root];  
 }  
 //  
 return root;  
 //END  
}

private void mergeComponents(int i, int j) {  
 // *FIXME make shorter root point to taller one* // END  
  
 if (i == j) return;  
 //according to the height update the parent  
 if (height[j] > height[i]) {  
 updateParent(i, j);  
 updateHeight(j, i);  
 } else {  
 updateParent(j, i);  
 updateHeight(i, j);  
 }  
 }  
  
 */\*\*  
 \* This implements the single-pass path-halving mechanism of path compression  
 \*/* private void doPathCompression(int i) {  
 // *FIXME update parent to value of grandparent* // END  
 parent[i] = parent[parent[i]];  
 }

package edu.neu.coe.info6205.union\_find;  
  
import java.util.Random;  
import java.util.Scanner;  
  
public class Client {  
  
 public static int count(int n) {  
 //count the number of connections for n objects for random values of a and b  
 UF\_HWQUPC uf = new UF\_HWQUPC(n);  
 Random r = new Random();  
 int x = 0;  
 while (uf.components() > 1) {  
 int a = r.nextInt(n);  
 int b = r.nextInt(n);  
  
 uf.connect(a, b);  
 x++;  
 }  
 return x;  
 }  
  
 public static void main(String[] args) {  
 //Take input n  
 System.*out*.println("please enter number for number of objects:");  
 Scanner scanner = new Scanner(System.*in*);  
 int n = scanner.nextInt();  
 //get number of connections  
 System.*out*.println("The number of objects are " + n + " and the number of connections are " + *count*(n));  
//step 3  
 System.*out*.println("Determine the relationship between the number of objects (n) and the number of pairs (m)");  
  
 for (int i = 1000; i < 140000; i \*= 2) {  
 int total = 0;  
 // test count 5 times for an average number  
 for (int j = 0; j < 5; j++) {  
 total += *count*(i);  
 }  
 int avg = total / 5;  
 System.*out*.println("The number of objects are " + i + " and the number of pairs are " + avg);  
 }  
 }  
  
}

* **Relationship Conclusion**

m : The number of pairs

n : The number of objects

Relationship between the number of objects (n) and the number of pairs (m)

**m = ½ n Log(n)**

* **Evidence / Graph**

**Input:150**

|  |  |  |
| --- | --- | --- |
| **Number of Objects** | **Number of Pairs** | **(n \*log(n)/2)** |
| 1000 | 3418 | 3454 |
| 2000 | 7775 | 7601 |
| 4000 | 17222 | 16588 |
| 8000 | 36673 | 35949 |
| 16000 | 80020 | 77443 |
| 32000 | 167232 | 165976 |
| 64000 | 366888 | 354132 |
| 128000 | 812723 | 752626 |

1. **Result:**

-Run Client.java. Enter a number from command line as input.

-To test the relationship between m and n we use higher values.

-We passed n=150. Ran the count function 5 times to get average number of its pairs

-X axis is number of objects.

-Y axis is number of pairs

-Blue line represents number of objects Vs number of pairs

-Orange line represents calculated o/p Vs number of objects

* **Unit tests result**

