PSA Assignment 4|Padma Anaokar |

Github: https://github.com/padmaanaokar/002727445\_PSA

**Part A:**

Code Changes in **UF\_HWQUPC.java:**

**Find method**

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

}

**mergeComponents method:**

**private void** mergeComponents(**int** i, **int** j) {  
 *//* ***FIXME make shorter root point to taller one*** *// END* **int** rootX = find(i);  
 **int** rootY = find(j);  
  
 **if** (**height**[rootX] < **height**[rootY]) {  
 **parent**[rootX] = rootY;  
 **height**[rootY] += **height**[rootX];  
 } **else if** (**height**[rootX] > **height**[rootY]) {  
 **parent**[rootY] = rootX;  
 **height**[rootX] += **height**[rootY];  
 } **else** {  
 **parent**[rootY] = rootX;  
 **height**[rootX]++;  
 }  
  
}

**doPathCompression Method:**

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

**Test Cases:**

Graphical user interface, text, application, email

Description automatically generated

**Part B:**

**Writing a main function for running a fixed set of n values. HWQUPC\_Solution.java**

**static int** *union*=0;  
 **public static void** main(String[] arg) {  
 *findConnections*(1000, 250, 1500);  
 }  
 **public static void** findConnections(**int** runs, **int** low, **int** high) {  
 **for** (**int** i = low; i <= high; i = i+low) {  
 **int** totalConn = 0;  
 **for** (**int** j = 0; j < runs; j++) {  
 totalConn+=*count*(i);  
 }  
 System.***out***.println(**"Connection generated for (N) "** + i + **" sites: "** + totalConn/runs);  
 System.***out***.println(**"Number of Unions required for "**+i+**" sites are :"**+*union*/runs);  
 *union*=0;  
 }  
 }  
 **public static int** count(**int** n) {  
 **int** pairs = 0;  
 UF\_HWQUPC unionFind = **new** UF\_HWQUPC(n);  
 Random r = **new** Random();  
 **while**(unionFind.components() > 1) {  
 **int** p = r.nextInt(n);  
 **int** q = r.nextInt(n);  
 pairs++;  
 **if** (!unionFind.isConnected(p, q)) {  
 unionFind.connect(p, q);  
 *union*++;  
 }  
 }  
 **return** pairs;  
 }

**Output:**

**Graphical user interface, text, application

Description automatically generated**

**Part C:**

Graph:

Unions Vs Pairs

|  |  |  |
| --- | --- | --- |
| N | Union | Pairs |
| 250 | 249 | 760 |
| 500 | 499 | 1691 |
| 750 | 749 | 2713 |
| 1000 | 999 | 3720 |
| 1250 | 1249 | 4826 |
| 1500 | 1499 | 5918 |

Chart, line chart

Description automatically generated

**Conclusion:**

Here we tried to average the number of pairs generated value by taking an average of 1000 runs for any given N value and we have plotted a graph for the same.

We can observe that the relationship between number of objects (n) and the number of pairs (m) generated is

**M = (N\*log(N))/2**

Also, the number of connections formed in union find to connect n objects is (n-1) because any connection after n-1 will result in a cycle.