Program Structures & Algorithms  
Spring 2022  
Assignment No. 3(WQUPC)

`

Name: Vamsi Krishna Poluru  
(NUID): 002924538

•Task

Step 1:  
(a) Implement height-weighted Quick Union with Path Compression. For this, you will flesh out the class UF\_HWQUPC. All you have to do is to fill in the sections marked with // TO BE IMPLEMENTED ... // ...END IMPLEMENTATION.

(b) Check that the unit tests for this class all work. You must show "green" test results in your submission (screenshot is OK).

Step 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. If you prefer, you can create a main program that doesn't require any input and runs the experiment for a fixed set of n values. Show evidence of your run(s).

Step 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). Justify your conclusion in terms of your observations and what you think might be going on.

Solution file code:

HWQUPC\_SOLUTION.Java

package edu.neu.coe.info6205.union\_find;  
  
import java.util.\*;  
  
public class HWQUPC\_Solution {  
  
 static int *randomPairsGenerated*=0;  
 public static void main(String[] args)  
 {  
 int[] testdata=new int[50];  
 int numberOfRuns=200,out=0;  
  
 for(int i=0;i<testdata.length;i++)  
 testdata[i]=i\*10000;  
  
 System.*out*.println("Number of Elements\t"+"Number of Randomly Generated Pairs");  
 for(int i=1;i<testdata.length;i++)  
 {  
 *randomPairsGenerated*=0;  
 for (int j = 0; j < numberOfRuns; j++)  
 {  
 out = *count*(testdata[i]);  
 }  
  
 System.*out*.println("Number of elements: "+testdata[i]+", Number of Connections needed are "+out + " and Number of the pairs: "+(*randomPairsGenerated* / numberOfRuns));  
 }  
 }  
  
  
 public static int count(int i)  
 {  
 int connectionCount=0;  
 UF\_HWQUPC unionfind=new UF\_HWQUPC(i,true);  
 Random random= new Random();  
 while(unionfind.components()>1)  
 {  
 int a= random.nextInt(0,i);  
 int b= random.nextInt(0,i);  
 *randomPairsGenerated*++;  
 if(!unionfind.isConnected(a,b)){  
 unionfind.union(a,b);  
 connectionCount++;  
 }  
 }  
 return connectionCount;  
 }  
}

UF\_HWQUPC.Java

Methods to be filled are filled and present below.

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

private void mergeComponents(int i, int j) {  
 if(height[i]<height[j]) {  
 updateParent(i,j);  
 updateHeight(j, i);  
 }  
 else {  
 updateParent(j,i);  
 updateHeight(i, j);  
 }  
}

private void doPathCompression(int i) {  
 parent[i]=parent[parent[i]];  
}

•Output screenshot

Test file output screenshot:

Graphical user interface, text, application, email

Description automatically generated

Solution file output screenshots:

Graphical user interface, application, table

Description automatically generated

Graphical user interface, application, table

Description automatically generated

•Relationship Conclusion

Have filled the methods and created a main method and static count method to execute the test scenarios mentioned in the problem statement on UF\_HWQUPC.Java. Have created an array of size 50 with numbers ranging from 10000 to 500000 numbers are ran for 200 times and mean is calculated for all the runs. The number of connections needed to make **the set of N elements completely connected is always N-1**. By plotting the mean vs number of random elements needed to make the whole set connected we will be seeing that the graph plotted below looks somewhere near to **Linearithmic.** As the number of elements increase so does the number of connections and the number of a random elements required is increasing proportionally.

Below in the evidence, I have provided a table with the calculation and observations from the values generated

The values of the number of randomly generated sets are almost equal to half of the number of elements times the logarithmic of number of elements.

Number of Elements is N

Number of Connections is M

Number of Random Generated Pairs is R

**M= N-1**

**R= (1/2)\* N\*log N**

•Evidence / Graph  
Below is the graph plotted with the generated values.

The Values and the calculations for the relation between them

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Number of pairs | ln N | 1/2 N ln N | Number Of Connections |
| 10000 | 49088 | 9.210340372 | 46051.70186 | 9999 |
| 20000 | 105982 | 9.903487553 | 99034.87553 | 19999 |
| 30000 | 162852 | 10.30895266 | 154634.2899 | 29999 |
| 40000 | 221599 | 10.59663473 | 211932.6947 | 39999 |
| 50000 | 286988 | 10.81977828 | 270494.4571 | 49999 |
| 60000 | 352333 | 11.00209984 | 330062.9952 | 59999 |
| 70000 | 408138 | 11.15625052 | 390468.7682 | 69999 |
| 80000 | 472428 | 11.28978191 | 451591.2765 | 79999 |
| 90000 | 541791 | 11.40756495 | 513340.4227 | 8999 |
| 100000 | 602797 | 11.51292546 | 575646.2732 | 9999 |
| 110000 | 672848 | 11.60823564 | 638452.9605 | 109999 |
| 120000 | 737389 | 11.69524702 | 701714.8213 | 119999 |
| 130000 | 810265 | 11.77528973 | 765393.8324 | 129999 |
| 140000 | 871355 | 11.8493977 | 829457.8391 | 139999 |
| 150000 | 932611 | 11.91839057 | 893879.293 | 149999 |
| 160000 | 985320 | 11.98292909 | 958634.3275 | 159999 |
| 170000 | 1063384 | 12.04355372 | 1023702.066 | 169999 |
| 180000 | 1139705 | 12.10071213 | 1089064.092 | 179999 |
| 190000 | 1210603 | 12.15477935 | 1154704.038 | 189999 |
| 200000 | 1283905 | 12.20607265 | 1220607.265 | 199999 |
| 210000 | 1333443 | 12.25486281 | 1286760.595 | 209999 |
| 220000 | 1399088 | 12.30138283 | 1353152.111 | 219999 |
| 230000 | 1478242 | 12.34583459 | 1419770.978 | 229999 |
| 240000 | 1560054 | 12.3883942 | 1486607.304 | 239999 |
| 250000 | 1612944 | 12.4292162 | 1553652.025 | 249999 |
| 260000 | 1694871 | 12.46843691 | 1620896.798 | 259999 |
| 270000 | 1760425 | 12.50617724 | 1688333.927 | 269999 |
| 280000 | 1812979 | 12.54254488 | 1755956.284 | 279999 |
| 290000 | 1912190 | 12.5776362 | 1823757.249 | 289999 |
| 300000 | 1968315 | 12.61153775 | 1891730.663 | 299999 |
| 310000 | 2043456 | 12.64432758 | 1959870.774 | 309999 |
| 320000 | 2139804 | 12.67607627 | 2028172.204 | 319999 |
| 330000 | 2168526 | 12.70684793 | 2096629.909 | 329999 |
| 340000 | 2272858 | 12.7367009 | 2165239.152 | 339999 |
| 350000 | 2313375 | 12.76568843 | 2233995.476 | 349999 |
| 360000 | 2429674 | 12.79385931 | 2302894.676 | 359999 |
| 370000 | 2493155 | 12.82125828 | 2371932.783 | 369999 |
| 380000 | 2555269 | 12.84792653 | 2441106.041 | 379999 |
| 390000 | 2620795 | 12.87390202 | 2510410.894 | 389999 |
| 400000 | 2710362 | 12.89921983 | 2579843.965 | 399999 |
| 410000 | 2772857 | 12.92391244 | 2649402.05 | 409999 |
| 420000 | 2820457 | 12.94800999 | 2719082.098 | 419999 |
| 430000 | 2932779 | 12.97154049 | 2788881.205 | 429999 |
| 440000 | 3014469 | 12.99453001 | 2858796.601 | 439999 |
| 450000 | 3031104 | 13.01700286 | 2928825.644 | 449999 |
| 460000 | 3120220 | 13.03898177 | 2998965.807 | 459999 |
| 470000 | 3235397 | 13.06048797 | 3069214.674 | 469999 |
| 480000 | 3286520 | 13.08154138 | 3139569.932 | 479999 |
| 490000 | 3372914 | 13.10216067 | 3210029.364 | 489999 |
| 500000 | 3453276 | 13.12236338 | 3280590.844 | 499999 |

•Unit tests result

Graphical user interface, text, application, email

Description automatically generated