Program Structures and Algorithms

Spring 2023(SEC –01)

NAME: Hinal Prabodh Patel

NUID: 002766180

**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.

**Relationship Conclusion:**

The relationship between the number of objects (n) and the number of pairs (m generated to reduce the number of components from n to 1 is:

m = f (n) =0.5 \* n \*In (n)

Where, m = number of pairs generated to reduce the number of components, n = number of objects

**Evidence to support the conclusion:**

To minimize the number of components from n to 1, let f(N) be the quantity of pairs (m) that were generated. We can compute the average number of pairs created to achieve this for each value of n by starting with an initial value of n equal to 100 and applying the doubling approach to determine the number of pairs (m) generated to reduce the number of components from n to 1.

The average number of pairs required to lower the component 1 for increasing values of n is quite close to 1/2 × n × In(n).

|  |  |  |
| --- | --- | --- |
| No of Objects(n) | No of Pairs(m) | 0.5\*n\*log(n) |
| 100 | 243 | 100 |
| 200 | 514 | 230.103 |
| 400 | 1030 | 520.411998 |
| 800 | 2259 | 1161.23599 |
| 1600 | 5646 | 2563.29599 |
| 3200 | 13353 | 5608.23997 |
| 6400 | 27666 | 12179.7759 |
| 12800 | 60966 | 26286.1438 |
| 25600 | 138240 | 56425.4716 |
| 51200 | 298240 | 120557.311 |
| 102400 | 691510 | 256527.358 |

**Graphical Representation:**

The diagrams below illustrate the outcome of charting the data from the preceding table on a standard scale with the number of objects (n) on the x-axis and the number of pairings (m) produced to decrease the number of components from n to 1 on the y-axis.

**Unit Test Screenshots:**

Text

Description automatically generated

Code Snippets:

find() method implementation:

Text

Description automatically generated

mergeComponents() and doPathCompression() method implementation :

Text

Description automatically generated