Colin Weinshenker

CS303

Project 04

All files were compiled and run on the “Astro” machine in the McGlothlin CS lab. The compiler options used were the ones provided in the makefile.

0. The test() function in union\_find0.cpp tests the different class objects appropriately because the use of virtual functions allows the compiler to recognize and call the methods from less abstract classes. Without virtual functions, the test() function in union\_find1.cpp recognizes all calls to onion() and find() as calls to the Union\_Find() class object underlying the various classes. Thus the union\_find1.cpp test() function fails because it attempts to test non-Union\_Find objects as Union\_Find objects.

1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Trial 1  100000 sites  200000 edges | Trial 2  50000  sites  100000  edges | Trial 3  200000  sites  400000  edges | Trial 4  400000  sites  800000  edges | Trial 5  50000  sites  5000000  edges | Trial 6 50000  Sites  10000000 |
| Union Quick Find | 10995ms | 2798ms | 43171ms | 178534ms | 455ms | 112ms |
| Union Quick Find (Connected) | 3ms | 1ms | 6ms | 22ms | 636ms | 126ms |
| Quick Union | 13135ms | 3162ms | 62491ms | 204467ms | 106571ms | 20739ms |
| Quick Union  (Connected) | 61623ms | 14910ms | 29459ms | 96336sms | 217319ms | 41715ms |
| Weighted Quick Union | 9ms | 4ms | 21ms | 62ms | 1622ms | 309ms |
| Weighted Quick Union  (Connected | 13ms | 6ms | 29ms | 66ms | 2632ms | 493ms |
| Weighted Quick Union w/ P.C. | 9ms | 4ms | 18ms | 58ms | 1057ms | 210ms |
| Weighted Quick Union w/ P.C. (Connected) | 9ms | 5ms | 17ms | 49ms | 1885ms | 375ms |

The doubling tests (trials 1-4) show that the runtimes for quick find union and quick union union increased runtime proportional to n^2 (i.e., doubling the number of sites and edges causes the function to take four times as long), while the runtimes for weighted quick union and weighted quick union with path compression increased roughly proportional to n (i.e., doubling the number of sites and edges causes the function to take only twice as long). The last two tests check implementation performance with higher density graphs of one hundred times more edges than sites and twenty times more edges than sites. As expected, increasing the density of the graph greatly increases the performance of quick find union and greatly decreases the performances of weighted quick union and weighted quick union with path compression. The performance of the weighted quick unions was roughly linear time, as expected. Weighted quick union performed upwards of one thousand times faster than quick find and quick union when tested on large numbers of sites. The path compression version performed just a hair faster than regular weighted quick union. Unweighted quick union surprised by slightly underperforming, compared to quick find.

2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Trial 1  100000 sites, 200000 edges | Trial 2  50000  sites  100000  edges | Trial 3  200000  sites  400000  edges | Trial 4  400000  sites  800000  edges | Trial 5  5000  sites  5000000  edges | Trial 6  5000  sites  1000000 |
| Union Quick Find | 11106ms | 2758ms | 44942ms | 181719ms | 42ms | 31ms |
| Union Quick Find (Connected) | 0ms | 0ms | 0ms | 0ms | 0ms | 0ms |
| Quick Union | 14255ms | 3440ms | 65464ms | 219617ms | 13566ms | 2682ms |
| Quick Union  (Connected) | 63546ms | 15412ms | 298783ms | 1148529ms | 23959ms | 4792ms |
| Weighted Quick Union | 7ms | 3ms | 17ms | 36ms | 120ms | 24ms |
| Weighted Quick Union  (Connected) | 8ms | 4ms | 19ms | 40ms | 155ms | 31ms |
| Weighted Quick Union w/ P.C. | 7ms | 3ms | 17ms | 54ms | 122ms | 25ms |
| Weighted Quick Union w/ P.C. (Connected) | 8ms | 4ms | 18ms | 45ms | 148ms | 30ms |

The relationships between the algorithms’ runtimes are the same as those in union\_find0.cpp, with two notable exceptions: 1) union\_find1.cpp quick find can confirm a graph is connected in less than a millisecond and 2) the performances of the weighted quick unions greatly increased (see trials 5 and 6).

3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Trial 1  100000 sites 200000 edges | Trial 2  50000  sites  100000  edges | Trial 3  200000  sites  4000000  edges | Trial 4  400000  sites  800000  edges | Trial 5  5000  sites  5000000  edges | Trial 6  5000  sites  1000000  edges |
| Union Quick Find | 10994ms | 2753ms | 44669ms | 181719ms | 44ms | 32ms |
| Union Quick Find (Connected) | 0ms | 0ms | 0ms | 0ms | 0ms | 0ms |
| Quick Union | 14218ms | 3440ms | 65461ms | Omitted—too slow | 14036ms | 2665ms |
| Quick Union (Connected) | 63582ms | 15419ms | 301339ms | Omitted—  Too slow | 24933ms | 4752ms |
| Weighted Quick Union | 7ms | 3ms | 18ms | Omitted—  Too slow | 123ms | 25ms |
| Weighted Quick Union  (Connected) | 8ms | 3ms | 18ms | Omitted—  Too slow | 150ms | 30ms |
| Weighted Quick Union w/ P.C. | 6ms | 2ms | 15ms | 44ms | 50ms | 10ms |
| Weighted Quick Union w/ P.C. (Connected) | 2ms | 1ms | 5ms | 13ms | 33ms | 6ms |

The relationships between the algorithms’ runtimes are the same as those in union\_find1.cpp, although weighted quick union with path compression now performs almost three times faster than weighted quick union.

4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Trial 1  1000000 sites, 2000000 edges | Trial 2  2000000  sites  4000000  edges | Trial 3  4000000  sites  8000000  edges | Trial 4  8000000  sites  16000000  edges | Trial 5  5000000  sites  50000  edges |
| Weighted Quick Union (foo0) | 276ms | 782ms | 1739ms | 3771ms | 26ms |
| Weighted Quick Union (connected) (Foo0) | 307ms | 928ms | 2307ms | 4907ms | 9ms |
| Weighted Quick Union w/ P.C.  (Foo0) | 311ms | 821ms | 1806ms | 3827ms | 12ms |
| Weighted Quick Union w/ P.C. (Connected)(Foo0) | 282ms | 892ms | 2292ms | 4462ms | 10ms |
| Weighted Quick Union (foo1) | 267ms | 633ms | 1588ms | 3489ms | Omitted—too slow |
| Weighted Quick Union (connected) (Foo1) | 237ms | 656ms | 1695ms | 3767ms | Omitted—  Too slow |
| Weighted Quick Union w/ P.C. (foo1) | 267ms | 635ms | 1606ms | 3494ms | Omitted—too slow |
| Weighted Quick Union w/ P.C. (connected) (foo1) | 235ms | 662ms | 1711ms | 3738ms | Omitted—too slow |
| Weighted Quick Union (foo2) | 254ms | 641ms | 1447ms | 3190ms | Omitted—too slow |
| Weighted Quick Union (connected)  (Foo2) | 221ms | 667ms | 1602ms | 3523ms | Omitted—too slow |
| Weighted Quick Union w/ P.C. (foo2) | 227ms | 563ms | 1385ms | 2794ms | Omitted—too slow |
| Weighted Quick Union w/P.C. (foo2) | 94ms | 279ms | 706ms | 1488ms | Omitted—too slow |

Based on the above table, the rankings for each algorithm (least efficient to most efficient) are as follows:

Weighted quick union: union\_find0.cpp, union\_find1.cpp, union\_find2.cpp

Weighted quick union with path compression: union\_find0.cpp, union\_find1.cpp, union\_find2.cpp

5.

The use of virtual functions and fewer inline functions in the higher level classes causes union\_find0.cpp to have a huge overhead cost, making the union\_find0.cpp implementation the slowest. Union\_find1.cpp improves on union\_find0.cpp by removing virtual functions and giving Quick\_Find() and Quick\_Union() their own inline functions for connected(), which saves time by preventing them from calling methods in less abstract classes. Union\_find2.cpp keeps the improvements from union\_find1.cpp and adds inline find() functions *within the class bodies* of Quick\_Union, Weighted\_Quick\_Union, and Weighted\_Quick\_Union\_with\_Path\_Compression (in the respective header files). This makes the union\_find2.cpp implementation fastest.

Note that the above analysis applies only when the ratio of sites to nodes is relatively balanced. In the fifth test, with a high ratio of sites to nodes, the weighted quick find implementations in union\_find0.cpp actually run much faster than those in union\_find1.cpp and union\_find2.cpp because of the virtual functions.