Disjoint Sets II

By Zander Havgaard & Thomas Høpfner-Dahl

Results

We solved this exercise using java 8. The java program DS2.java can be executed using any of the three Union Find classes from the algs4 library, UnionFindUF, QuickUnionUF and WeightedQuickUnionUF respectively. We chose to test all three to find out which one would be the most optimum. To keep the code simple, switching between implementations is done by commenting/uncommenting lines 20 through 22 in DS2.java.

The test input was generated using a script that generates a .txt file containing the desired amount of operations over a specified range. The two numbers in each operation were chosen randomly between 0 and the top of the range. The distribution of operations were 2 queries for each union.

The table on the following page will present actual run times of running the algorithms on different combinations of ranges (n) and number of operations (m), along with extrapolations on how long we estimate the algorithms would take with even larger numbers based on the run times recorded.
The run times were recorded by saving the current time at the start of the program using System.currentTimeMillis() and again at the end of the program and then subtracting them.

All of the numbers written in red are extrapolated, the black numbers are actual run times.
We have not been able to extrapolate fully for the quick-union algorithm as we have gotten consistent results not fitting into any model for calculating not processed input. E.g. with 1 mil. Operations we went from 7.5 seconds for a range of 10.000 to 135 seconds for a range of 100.000, but then went back to 1.6 seconds for a range of 1 mil. These results does not make sense and can in our opinion not be extrapolated from. We expect the problem being in our benchmarking framework and that the JIT compiler is not executing the program properly but fetching from the cache (We would love to get feedback on this).

Our experiments have clearly shown that the Quick-find algorithm is efficient when working with a small range, as it can handle large amounts of operations efficient on a small range. Quick-union performs worse then quick-find the more operations that has to be performed, but can handle large ranges a lot efficient. The weighted quick-union overall performs the best as it is the most efficient algorithm with both large and small ranges as well as small and large amounts of operations. For a 1 billion range with 1 billion operations our calculations suggest that it would take more than 3 years for the quick-find and around 38 minutes for the weighted quick-union algorithm, thus we would choose the latter.

Quick-find	10^3	10^4	10^5	10^6	10^7	10^8	10^9	
Time Elapsed, Secs	Operations: 1000	Operations: 10.000	Operations: 100.000	Operations 1.000.000	Operations. 10.000.000	Operations: 100.000.000	Operations: 1.000.000.000	
Range: 1.000	0.068	0.151	0.277	1,273.000	10,183	ca 101,83 sek	ca 1018,3 sek	
Range: 10.000	0.079	0.157	0.333	1,331.000	10,913	ca 109,13 sek	ca 1091,3 sek	
Range: 100.000	0.084	0.212	1.080	7,593.000	96,264	ca 800 sek	ca 8000 sek	
Range: 1.000.000	181.000	1.076	9,398.000	91,117.000	ca 911 sek	ca 9111 sek	ca 91111 sek	
Range 10.000.000	1.042	9,598.000	94,431.000	ca 944 sek	ca 9943 sek	ca 99431 sek	ca 994311 sek	
Range 100.000.000	6.252	86,382	849,879	9,441	99431	994311	9943111	
Range 1.000.000.000	37.512	777,438	7,648,911	94,411	994311	9943111	99431111	<- (AROUND 3 YEARS)

quick-union	10^3	10^4	10^5	10^6	10^7	10^8	10^9
Time Elapsed, Secs	Operations: 1000	Operations: 10.000	Operations: 100.000	Operations 1.000.000	Operations. 10.000.000	Operations: 100.000.000	Operations: 1.000.000.000
Range: 1.000	0.064	0.150	0.323	1,761	15,823		
Range: 10.000	0.068	0.154	0.721	7,584	81,411		
Range: 100.000	0.077	0.143	0.356	135,915			
Range: 1.000.000	0.072	0.154	0.321	1,673			
Range 10.000.000	0,1	0.159	0.365	1,725			
Range 100.000.000	0,11	0.163	0.370	1,801			
Range 1.000.000.000	0,12	0.165	0.375	1,901			

Weighted Quick union	10^3	10^4	10^5	10^6	10^7	10^8	10^9	
Time Elapsed, Secs	Operations: 1000	Operations: 10.000	Operations: 100.000	Operations 1.000.000	Operations. 10.000.000	Operations: 100.000.000	Operations: 1.000.000.000	
Range: 1.000	0.047	0.135	0.311	1.329	12.225	97.8	1564.8	
Range: 10.000	0.052	0.137	0.299	1.394	12.595	100.76	1612.16	
Range: 100.000	0.056	0.140	0.300	1.524	13.663	109.304	1748.864	
Range: 1.000.000	0.057	0.148	0.341	1.787	15.173	121.384	1942.144	
Range 10.000.000	0.093	0.171	0.372	1.833	15.945	127.56	2040.96	
Range 100.000.000	0.103	0.201	0.385	1.933	16.945	135.56	2168.96	
Range 1.000.000.000	0.113	0.245	0.401	2.033	17.945	143.56	2296.96	<- (38 minutes)