EE1103-NUMERICAL METHODS-QUIZ 1 REPORT

EE23B102 - Sanjeev Subrahmaniyan S B

2023-09-23

1 Abstract and takeaways

This report details on my solutions for the problems in the quiz. I found the algorithms very interesting to think about and found the problems helpful in getting a better understanding of the utilization of structs, pointers, and arrays as linked lists. Figuring out the worst case scenarios were a little tricky for using small (10^5) sample sets resulted in very close run times, varying the most by about a second. These problems and the test cases that accompanied them emphasized the importance of dynamic memory allocation and permitted me to explore other features of my laptop such as using the terminal ulimit command. Also, while it was not specifically mentioned to use path compression, I have made use of it for it takes just one extra line of code, but still appears to be a good improvement on processing.

2 Problem 1 - Simple array based solution

In this problem I made use of the conventional array update method (disjoint sets) which led to a fairly straightforward but largely inefficient solution. Its runtimes for different test cases are presented in Table 1 in contrast to the other methods. A key observation here is that it runs at comparable speeds when the number of planets are lower, even when the number of requests are huge.

3 Problem 2 - Worst case for the implementation in problem 1

The worst case scenarios were trickly to identify. A common solution was one which aimed to run consequtive pairs, i.e. 0 1, 1 2, 2 3,... while I also pondered over the possibility of 1 0, 2 1, 3 2,... being the worse case. This was supported by the latter returning marginally greater runtimes on an average over many executions. However, interestingly, when the same sequence is extended to 10^6 and compared against the sample test file with the same length received, my sequence ran much faster ($10s against\ 10m$). This indicates that a random number sequence could actually force cache misses and hence have a longer runtime. I have however stuck to my latter case for 16 planets, partly due to the fact that it updates the values in the array O(n^2) times while the initial method updates O(n) times.

4 Problem 3 - Solution using structs, pointers and path compression

In this problem I used the method of structs and pointers and implemented them in a weighted union style with path compression. Path compression seemed to make the program faster as it directly links the element to the root of its parent element. An interesting observation was that although the use of path compression seems to be a significant improvement in logic, the difference in run times for samples of sizes (10^6) were small.

5 Problem 4 - Solution using arrays, weights and path compression

This solution was similar to the one implemented in the Problem 3, but instead used arrays and pointers for weights, roots, and indices. It was a simple implementation with efficiency slightly lower than that of the one in problem 3. The noteworthy observation was that this method seemed to perform relatively worse when the number of planets are small and the number of requests huge.

Table 1: Test case runtime vs sample size (NP=number of planets, NR=number of requests) for different problem implementations, rounded to nearest integer millisecond

Problem	$NP = 100$ $NR = 100$	$NP = 10^5$ $NR = 10^5$	$NP = 10^6$ $NR = 10^6$	$NP = 1000$ $NR = 10^7$	$NP = 10^7$ $NR = 10^7$
1	0 ms	7906 ms	841513 ms	828 ms	too long.
3	0 ms	17 ms	186 ms	1012 ms	35890 ms
4	0 ms	18 ms	220 ms	986 ms	57789 ms

6 Bibliography

I did all of the coding on my own and used the following sources to familiarize myself with the algorithm and problem I was dealing with:

1. https://regenerativetoday.com/union-find-data-structure-quick-find-algorithm/
2. https://algs4.cs.princeton.edu/15uf/ (special mention, very concise and easy to understand)
3. https://www.youtube.com/watch?v=DZKzDebT4gU

Aside from this numerous conversations occurred between me and my classmates, many of which helped me spot loopholes in algorithms and provoke thinking.