COMPSCI 308 Design and Analysis of Algorithms Weekly Journal 2

Luyao Wang March 31 2024

This week, I delved deeper into the concepts of divide and conquer, the Master's theorem, and dynamic programming.

Divide and conquer is a problem-solving technique that involves three steps. First, we divide the problem into smaller subproblems of the same kind. Then, we solve these subproblems recursively using the same function, which is known as conquering or recurring. Finally, we combine the solutions of the subproblems to obtain the solution to the original problem.

We studied two popular sorting algorithms that employ the divide and conquer approach: Merge Sort and Quick Sort. In Merge Sort, the division step is trivial, and the conquering step takes $\Theta(n)$ time. Quick Sort, on the other hand, divides the problem in $\Theta(n)$ time through a partitioning process, and the conquering step is trivial. The worst-case time complexity of Quick Sort $O(n^2)$ occurs when partitioning produces one subproblem with $n-1$ elements and one with 0 elements. With an uneven split, it can achieve a time complexity of $O(n \log_k n)$, where k represents the base of the logarithm.

To analyze the time complexity of divide and conquer algorithms, I learned the substitution method, recursion tree method, and the Master's theorem. The Master's theorem provides a formula for determining the overall time complexity based on three cases and a regularity condition.

Furthermore, we explored the concept of dynamic programming. According to the definition from CLRS, dynamic programming is applicable when subproblems overlap, meaning they share common sub-subproblems. In dynamic programming, we solve each sub-subproblem only once and store the answer in a table to avoid recomputation. This approach is more efficient than divide and conquer, as the latter may repeat work by solving common sub-subproblems multiple times.

Dynamic programming involves solving small problems to obtain the optimal solution for a larger problem. It guarantees an optimal solution for the larger problem by leveraging the optimal solutions of its subproblems.

I learned two implementations of dynamic programming: Top-down with Memoization and Bottom-up through the example of rod cutting. Furthermore, we explored dynamic programming in the matrix multiplication problem, where we find an optimal way to parenthesize the multiplication of matrices.

Overall, this week's exploration of divide and conquer, the Master's theorem, and dynamic programming has deepened my understanding of these fundamental techniques in algorithm design. By applying these concepts, I can solve complex problems more efficiently.