CSE 551: Foundations of Algorithms

Module 4: Divide and Conquer

Ali Altunkaya
Spring 2024

Outline

1. Basics of Divide & Conquer technique
2. Applications of Divide and Conquer
    1. MergeSort
    2. Closest Pair of Points
    3. Integer Multiplication (Karatsuba version)
    4. Matrix Multiplication (Strassen version)
    5. Maximum Subarray Sum
3. Effects of sub-problem size and number of recursive calls
4. Conclusions

Module 4 Highlights

Motivation for Divide & Conquer:

- Usually we can decrease the running time complexity an algorithm, from $O(n^2)$ to O(n logn) by applying Divide & Conquer.

Huge gain when n is larger.

If n = 1,000 then:

- Brute force: $n^2 = 1000^2$ = 1 million
- Divide Conquer: n logn = 1000*log2(1000) =~ 10,000
- 1 second vs 100 seconds (e.g. waiting for a click on website)

If n = 1,000,000 then:

- Brute force: $n^2 = 1{,}000{,}000^2 = 10^{12}$ = 1 trillion
- Divide Conquer: n logn = 1,000,000*log2(1,000,000) =~ 20,000,000
- 1 vs 50,000 (e.g. analyzing big data)

Module 4 Highlights

Divide & Conquer is not an algorithm. It is a technique or method that you can apply to a problem and develop a new algorithm.

It has 3 steps:

1-) Divide the problem (input data) into sub-problems (smaller datasets) recursively, until you can't divide it anymore.

- Usually we divide it into two recursively. (subproblem size = n/2)
- Make sure both sub-problems have roughly the same size.

2-) Solve the sub-problems.

3-) Combine (merge) the sub-solutions recursively, into the single solution for the input. While combining sub-solutions, you may need to process borderline data (some data from both sub-problems, but close to the borderline) too. Example:

- Borderline data processing not needed: MergeSort, … etc.
- Borderline data processing needed: Closest pair of points, Maximum Subarray Sum etc ...

Module 4 Highlights

Usually we use recurrence relations to analyze the running times of Divide & Conquer algorithms. Because:

Recurrence Relation: describes a function in terms of its value on smaller inputs.

Three methods of solving recurrence relations:

- Recursion Tree method
- Induction method
- Telescoping method

Module 4 Highlights

Induction

$$base-case: T(1)=0$$
$$inductive\ hypothesis: T(n)=nlog_2(n)$$
$$goal: show\ that\ T(2n)=2nlog_2(2n)$$

start from the recurrence relation:

$$T(2n)=2T(n)+2n$$
$$T(2n)=2nlog_2(n)+2n$$
$$T(2n)=2nlog_2(2n*\frac{1}{2})+2n$$
$$T(2n)=2n(log_2(2n)+log_2\frac{1}{2})+2n \quad \Rightarrow \quad log_2(\frac{1}{2})=-1$$
$$T(2n)=2n(log_2(2n)-1)+2n$$
$$T(2n)=2nlog_2(2n)-2n+2n$$
$$T(2n)=2nlog_2(2n)$$

Module 4 Highlights

Telescoping: Proof by dividing n, finding complexity for smallest.

$$T(n) = 2\mathrm{T}(n/2) + n$$

$$\frac{T(n)}{n} = \frac{2\mathrm{T}(n/2)}{n} + \frac{n}{n}$$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1 \quad \Rightarrow \quad \frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$

$$\frac{T(n)}{n} = \frac{T(n/4)}{n/4} + 1 + 1$$

.....

$$\frac{T(n)}{n} = \frac{T(n/n)}{n/n} + 1 + \ldots + 1$$

$$\frac{T(n)}{n} = 0 + 1 + \ldots + 1 \quad \Rightarrow \quad \textit{height of tree is} \log_2(n)$$

$$\frac{T(n)}{n} = \log_2(n)$$

$$T(n) = n\log_2(n)$$

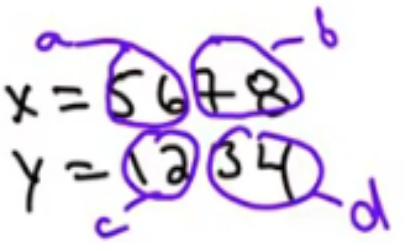
Module 4 Highlights

Closest pair of points: Finding the closest pair of points in a 2-d space.

- Why are we dividing region into two pieces?
  - Usually we divide the input data into two parts, to make sure both parts are equal in size.
  - Also, since it is recursive, eventually we will divide it into 4,8,16 parts later right!
- We divide the δ-strip into 12 squares of length ½ δ. Why do we compare with the next 11 values?
  - The diagonal of squares will be ½δ < δ. And each area can have at most 1 point.
  - We use 3 rows, because rowLength = ½ δ

Integer Multiplication (Karatsuba)

Example



Step1: Compute $a \cdot c = 672$

Step 2: Compute $b \cdot d = 2652$

Step 3: Compute $(a+b)(c+d) = 134 \cdot 46 = 6164$

Step 4: Compute ③ − ② − ① = 2840

Step 5:

$$\begin{array}{r} 6720000 \\ 2652 \\ 284000 \\ \hline 7006652 \end{array} = (1234)(5678)$$

https://www.youtube.com/watch?v=JCbZayFr9RE&ab_channel=StanfordAlgorithms

Q1: In Karatsuba multiplication, did we divide data into 2 or 4? What is data? What is n?

Q2: In Strassen matrix multiplication, did we divide data into 2 or 4? What is data? What is n?

Module 4 Highlights

What are the effects of these parameters to the complexity of a Divide&Conquer algorithm? Especially for the combine (merge) step.

Assume our initial problem size is n. What happens if:

$O\left(n^{\log_i k}\right)$

i: size of sub-problems: n/2 or n/3 or n/4

k: number of recursive calls: 2 or 3 or 4

Note: if you make too-many recursive-calls, you may not get any benefit from divide&conquer method. So one of our goals is also to reduce the number of recursive calls. Through reuse or some other clever strategies...

- Classic multiplication: If we divide data by 2 and make 4 recursive calls: $O\left(n^{\log_2 4}\right)=O\left(n^2\right)$

- Karatsuba multiplication: Divide data by 2 and make 3 recursive calls: $O\left(n^{\log_2 3}\right)=O\left(n^{1.56}\right)$

- Strassen multiplication: Divide data by 2 and make 7 recursive calls: $O\left(n^{\log_2 7}\right)=O\left(n^{2.81}\right)$

Question: If Karatsuba multiplication is better than classic one, why don't we use it in computers? Computer hardwares or low-level operations (such as Assembly etc). Does it have any use in real-world?

Question: Can Strassen's matrix multiplication work if the size of matrix is not the power of 2? For example 48x48 matrices or 34x34 matrices? If yes, how?

Module 4 Highlights

Maximum Subarray Sum: Given an array of integers (contains both + or -), find the sum of contiguous subarray which has the largest sum.

- Example: {-2, -5, 6, -2, -3, 1, 5, -6}
- Solution: {-2, -5, 6, -2, -3, 1, 5, -6} → 7

Solution 1: Brute-Force : $O(n^2)$

Solution 2: Divide & Conquer: O(nlogn)

- Return the max of these three:
  - Max subarray sum in left half (recursive call)
  - Max subarray sum in right half (recursive call)
  - Max subarray sum that crosses midpoint

Solution 3: Kadane's algorithm: O(n)

- But it uses an extra array, so a classic trade-off between running-time complexity and space complexity.

Module 4 Highlights

Conclusions:

Applying Divide&Conquer method to a new problem is not very easy. Dividing the data might be easier, but especially combining (or merging) the results might be more challenging.

If you search Google Scholar using “Divide and Conquer algorithm for ...”, you will see many papers. Each paper probably develop a new Divide and Conquer algorithm for an existing/new problem.

You can try to apply Divide and Conquer technique to a new or existing problem/method, and if it works, probably it may even be sufficient for a Masters Thesis in Computer Science.

Thanks

&

Questions