Divide and Conquer
Master MOSIG - Algorithms and Program Design

Marc Tchiboukdjian - Denis Trystram

13-11-2009

Course

Objectives

- Use the divide and conquer approach to design efficient algorithms.
- Analyze the cost of divide and conquer algorithms.

To Remember

Divide and conquer approach. The divide and conquer paradigm involves 3 steps :

- **Divide:** Break the problem into several subproblems that are similar to the original problem but smaller in size.
- **Conquer:** Solve the subproblems recursively.
- **Combine:** Combine these solutions to create a solution to the original problem.

Suppose the divide step yields a subproblems of size $1/b$ the size of the original problem. If the cost of the divide step is $D(n)$ and the cost of the combine step if $C(n)$, the cost of the algorithm $T(n)$ solves the following recurrence:

$$T(n) = \begin{cases} \Theta(1) \text{ if } n \leq c \\ aT(n/b) + D(n) + C(n) \text{ otherwise} \end{cases}$$

The next paragraphs show 3 methods to solve such recurrences.

Substitution method. Guess the form of the solution and use mathematical induction to find the constants and show that the solution works.

Recursion tree. Draw a recursion tree and label each node with the cost of the corresponding subproblem. Sum the costs within each level and then sum the per-level costs to determine the total cost of all levels of the recursion.

Master theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined by the recurrence

$$T(n) = aT(n/b) + f(n)$$

Then $T(n)$ can be bounded as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Insight:

1. If $f(n)$ is polynomially smaller than $n^{\log_b a}$ then the cost is dominated by the cost in the leaves.
2. If $f(n)$ is within a polylog factor of $n^{\log_b a}$ but not smaller then the cost if $n^{\log_b a} \lg^k n$ at each level and there are $\Theta(\lg n)$ levels.
3. If $f(n)$ is polynomially greater than $n^{\log_b a}$ then the cost is dominated by the cost of the root.

Example

Using the master theorem

- $T(n) = 5T(n/2) + \Theta(n^2)$.
  Compare $n^{\log_2 5}$ and n^2. Since $\log_2 5 - \epsilon = 2$ for some constant $\epsilon > 0$, use Case 1 to get $T(n) = \Theta(n^{\log_2 5})$.
- $T(n) = 27T(n/3) + \Theta(n^3 \lg n)$.
  Compare $n^{\log_3 27} = n^3$ to $n^3 \lg n$. Use Case 2 with $k = 1$ to get $T(n) = \Theta(n^3 \lg^2 n)$.
- $T(n) = 5T(n/2) + \Theta(n^3)$.
  Compare $n^{\log_2 5}$ to n^3. Now $\log_2 5 + \epsilon = 3$ for some constant $\epsilon > 0$. Check f: $af(n/b) = 5(n/2)^2 = 5n^3/8 \leq cn^3$ for $c = 5/8 < 1$. Use case 3 to get $T(n) = \Theta(n^3)$.

Merge sort

The merge sort algorithm follows the divide and conquer paradigm. It operates as follows.

- Divide: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each. Cost is $D(n) = \Theta(1)$.
- Conquer: Sort the two subsequences recursively using merge sort. Cost is $2T(n/2)$.
- Combine: Merge the two sorted subsequences to produce the sorted answer. Cost is $C(n) = \Theta(n)$.

Thus, $T(n)$ solves the following recurrence.

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1 \\ 2T(n/2) + \Theta(n) \text{ if } n > 1 \end{cases}$$

Using master theorem case 2, we get $T(n) = \Theta(n \lg n)$.

Karatsuba multiplication

The basic step of Karatsuba's algorithm is a formula that allows us to compute the product of two large numbers x and y using 3 multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts.

Let x and y be represented as n-digit strings in base 2. For $m = \lfloor n/2 \rfloor$, one can split the 2 given numbers as follows

$$\begin{aligned} x &= x_1 2^m + x_0 \\ y &= y_1 2^m + y_0 \end{aligned}$$

The product is then

$$xy = (x_1 2^m + x_0)(y_1 2^m + y_0) = z_2 2^{2m} + z_1 2^m + z_0$$

where

$$\begin{aligned} z_2 &= x_1 y_1 \\ z_1 &= x_1 y_0 + x_0 y_1 \\ z_0 &= x_0 y_0 \end{aligned}$$

These formulas require 4 multiplications of $n/2$-digit numbers and thus the cost is

$$T(n) = 4T(n/2) + \Theta(n) = \Theta(n^2)$$

Using these formulas

$$\begin{aligned} z_2 &= x_1 y_1 \\ z_0 &= x_0 y_0 \\ z_1 &= (x_1 + y_0)(y_1 + y_0) - z_2 - z_0 \end{aligned}$$

we only have 3 multiplications and thus the cost is

$$T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

The best know integer multiplication algorithm has $\mathrm{O}(n \log n 2^{\mathrm{O}(\log^* n)})$ complexity (Fürer 2007).

Strassen matrix multiplication

The naive algorithm to multiply two matrices has cost $\Theta(n^3)$. Using Strassen's formula that allow us to compute the product of two $n \times n$ matrices using 7 multiplications of $n/2 \times n/2$ matrices, we can build a divide and conquer algorithm with $\mathrm{O}(n^{2.81})$ complexity.

$$P = M.N = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \cdot \begin{pmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{pmatrix}$$

Using the formula

$$\begin{aligned} P_{11} &= M_{11}N_{11} + M_{12}N_{21} \\ P_{12} &= M_{11}N_{12} + M_{12}N_{22} \\ P_{21} &= M_{21}N_{11} + M_{22}N_{21} \\ P_{22} &= M_{21}N_{12} + M_{22}N_{22} \end{aligned}$$

we get

$$T(n) = 8T(n/2) + \Theta(n^2) = \Theta(n^3)$$

Using the Strassen's formula

$$\begin{aligned} P_{11} &= X_1 + X_4 - X_5 + X_7 \\ P_{12} &= X_3 + X_5 \\ P_{21} &= X_2 + X_4 \\ P_{22} &= X_1 + X_3 - X_2 + X_6 \end{aligned}$$

where

$$\begin{aligned} X_1 &= (M_{11} + M_{22})(N_{11} + N_{22}) \\ X_2 &= (M_{21} + M_{22})N_{11} \\ X_3 &= M_{11}(N_{12} - N_{22}) \\ X_4 &= M_{22}(N_{21} - N_{11}) \\ X_5 &= (M_{11} + M_{12})N_{22} \\ X_6 &= (M_{21} - M_{11})(N_{11} + N_{12}) \\ X_7 &= (M_{12} - M_{22})(N_{21} + N_{22}) \end{aligned}$$

we get

$$T(n) = 7\,T(n/2) + \Theta(n^2) = \Theta(n^{log_2(7)}) = \Theta(n^{2.81})$$

The best known matrix multiplication algorithm has $\mathrm{O}(n^{2.376})$ complexity (Coppersmith and Winograd 1990).

Convex hull

The convex hull of a set Q of points is the smallest convex polygon H for which each point in Q is either on the boundary of H or in its interior.

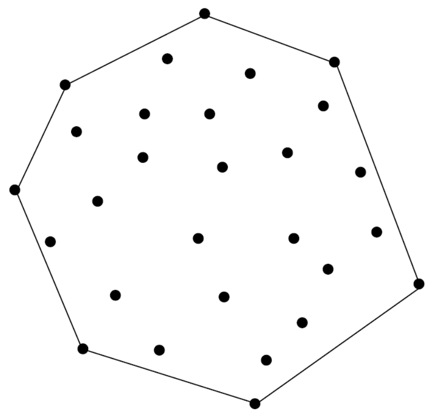

Figure 1: Convex hull

Let Q be a set of n points in the plane. To compute the convex hull of Q using a divide and conquer algorithm, we can proceed as follow.

`ConvexHull(`Q`)`

1. If $|Q| \leq 3$, then compute the convex hull by brute force in $\Theta(1)$ time and return.
2. Otherwise, partition the point set Q into two sets A and B, where A consists of half the points with the lowest x-coordinates and B consists of half of the points with the highest x-coordinates.
3. Recursively compute $H_A = CH(A)$ and $H_B = CH(B)$.
4. Merge the two hulls into a common convex hull, H, by computing the upper and lower tangents for H_A and H_B and discarding all the points lying between these two tangents.

`UpperTangent(`H_A`,`H_B`)`

1. Let a be the rightmost point of H_A.
2. Let b be the leftmost point of H_B.
3. While ab is not a upper tangent for H_A and H_B do
   (a) While ab is not a upper tangent to H_A do $a = a - 1$ (move a counterclockwise).

(b) While ab is not a upper tangent to H_B do $b = b + 1$ (move b clockwise).

4. Return ab.

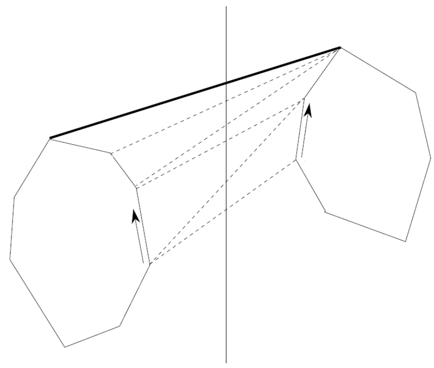

Figure 2: Convex hull

The divide step takes time $\Theta(n)$ (median finding), the combine step takes time $\Theta(n)$ (find upper and lower tangent), total complexity is

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$$

The best algorithm has complexity $\Theta(n \log h)$ where h is the number of points in the convex hull (T. Chan 1996).

Exercises

Recurrences

Solve the following recurrences.

1. $T(n) = 2T(n/2) + n^3$
2. $T(n) = T(9n/10) + n$
3. $T(n) = 16T(n/4) + n^2$
4. $T(n) = 7T(n/3) + n^2$
5. $T(n) = 7T(n/2) + n^2$
6. $T(n) = 2T(n/4) + \sqrt{n}$
7. $T(n) = T(n-1) + n$
8. $T(n) = T(\sqrt{n}) + 1$

9. $T(n) = T(n-1) + \lg n$
10. $T(n) = 2T(n/2) + n/\lg n$
11. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Quicksort

Suppose you are given an algorithm `median` which can find the median of n numbers in $\Theta(n)$ time. Devise a divide and conquer sort algorithm with $\Theta(n \log n)$ complexity.

Faster integer multiplication

Consider the integer multiplication problem: find the product of two n-bit integers x and y. We have seen in the class how to use divide-and-conquer algorithm to solve this problem in time $\Theta(n^{\lg_3})$. The key idea is to reduce a multiplication of two n-bit integers to three multiplications of two $(n/2)$-bit integers. Now, suppose we divide each integer into three equal parts, and then apply the divide-and-conquer algorithm. Show that we can reduce a single multiplication of two n-bit integers to 5 multiplications of two $(n/3)$-bit integers. What is the time complexity of such an algorithm?

QuickHull

The divide and conquer algorithm convex hull algorithm that we have seen in class can be viewed as a sort of generalization of Merge Sort. The algorithm that we will consider can be thought of as a generalization of the QuickSort sorting procedure. The resulting algorithm is called QuickHull.

Let E be the set of points and let P and Q be the points with minimum and maximum x-coordinate. Both P and Q are on the convex hull of E. Let E' and E'' be the two sets of points above and below line PQ. The convex hull of E is the concatenation of the convex hull of E' and the convex hull of E'' if we remove one copy of the edge PQ which is in both convex hulls. Let S the point of E' furthest away from line PQ. S is on the convex hull of E. All the points inside the triangle PSQ are not on the convex hull. Let E_1 the points above PS and E_2 the points above SQ. We can recursively compute the convex hull of E_1 and E_2.

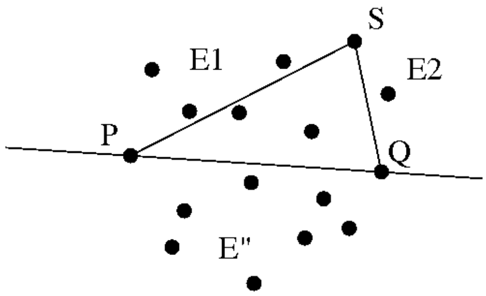

Figure 3: QuickHull

Using this idea, devise an algorithm to compute the convex hull of a set of points.

1. What is the worst case complexity of this algorithm?
2. Assume half of the points are discarded at each step (inside the triangle PSQ) and the remaining points are balanced between both sides (half in E_1 and half in E_2). What is the complexity in this case?

Fast exponentiation

You are given a real number x and a positive integer n. Give a naive algorithm to compute x^n. What is the complexity (number of multiplications) of your algorithm?

Devise a divide and conquer algorithm with better complexity. Give the recurrence verified by your algorithm. Hint: If n is even, $x^n = (x^{n/2})^2$.

Finding the missing integer

An array A of size n contains all the integers from 0 to n except one. It would be easy to determine the missing integer in linear time by using an auxiliary array B of size $n+1$ to record which numbers appear in A. In this problem, however, we cannot access an entire integer in A with a single operation. The elements of A are represented in binary, and the only operation we can use to access them is "fetch the jth bit of $A[i]$," which takes constant time.

Show that if we use only this operation, we can still determine the missing integer in O(n) time.