Algorithms and Data Structures

Charles A. Wuethrich

Bauhaus-University Weimar - CogVis/MMC

June 6, 2019

Space Reduction/ Divide and conquer

- Space reduction mechanisms
  - Range searching
    - Elementary Algorithms (2D)
    - Raster Methods
  - Shell sort
- Divide and conquer
  - Quicksort
  - The closest points problem
  - Polynomial arithmetics
  - Fast Fourier Transform (FFT)

Space reduction mechanisms

- The idea here is pretty simple: our problem is complex, and the data we have to look at is too big
- This makes the execution of any solution finding mechanism complicated, because solution cases grow with the number of data
- While looking at two objects makes available cases at most 4, if we have a large number of objects, the possible cases to explore grow like 2^N the number of objects
- So, reducing the objects to consider simplifies both reasoning as well as the tractability of the problem
- There are two ways to simplify object space:
  - By partitioning the space in a fixed number of parts
  - By partitioning recursively

Range searching

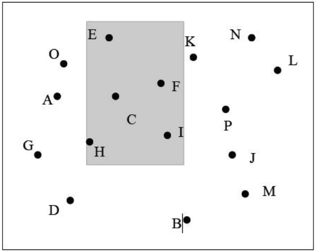
- As an example, let us consider the problem of range searching:
  - Search many times on same data set
  - Algorithms usually work in 2 steps:
    1. Prepare data for efficient search
    2. Do the actual search

Range searching (1-D)

- For the 1-dimensional problem, there is no big need of a space partition
  - Presorting produces the desired effect
- The 1D Problem: Given a set of numbers, find all numbers which lie a certain interval [a,b]
- Sol. 1:
  - Sort data
  - Do binary search for a and b
  - Pick all elements in between
- Sol. 2
  - Build a binary search tree (same as a heap)
  - Do recursive tree traversing to find where a and b are
  - Pop out in between values on the tree

Range searching (2D)

- In 2D, this problem becomes: Find points in rectangle $a_x \leq x \leq b_x$ and $a_y \leq y \leq b_y$



Range searching (n-D)

- In n-D: Given a set of points in n dimensional space, find all points such that they satisfy a certain property
- Examples:
    - All stars $<$ 50 light years far from the sun (1D)
    - All people between 21 and 31 earning between 50000 and 700000 EUR (2D)
    - ... and tall betw. 160 and 170 cm and female (4D)

Range searching (2D)

- First solution: use of 1-D algorithm on each direction
- Projection methods: first search fitting x values, then y values.
    - all “wrong” y values (=having wrong x) are left out from search

Raster Methods

- IDEA: Subdivide the whole domain in squares, and list all points in each square of the raster
- When the points in a rectangle R have to be found, just check only against squares intersecting with R
- Problem: size of the raster squares?
    - If grid too big, then too many points per square
    - If grid too small, then too many squares to be searched
- Sol: choose number of squares as a constant fraction of number of points, so that denser regions get denser squares

Raster Methods

- More precisely, if
  - max = max span between the points
  - N = number of points
  - M = wished number of points per square
- then size of square size = nearest integer to $\sqrt{\frac{N}{M}}$

- Efficiency:
  - Average case: linear with number of points in range
  - Worst case: linear with number of points overall

Shell sort

- Among all sorting methods, shell sort is one method which partitions the space
- This according to a fixed scheme
- The data is subdivided into 3h+1 subsets (h=1,2,...)

Divide and conquer

- Partition your space until you have reached an atomically easy to decide case
- This technique has been applied to many different problems:
  - From sorting in Quicksort
  - To the search of the closest points in a 2D set
  - To the multiplication of polynomes
  - To algorithms for computing the discrete Fourier Transform of a function

Divide and conquer

- Raster methods work quite well, they partition the space so that it is more tractable
- However, the single undersets are still difficult to manage
- The complexity of the choice of how much to subdivide is left to observations “a posteriori”.
- Simpler it would be if we reduced the problem to the bone, where no observation is needed.
- Divide and conquer mechanisms do exactly this: the space is recursively subdivided until it is “atomic”

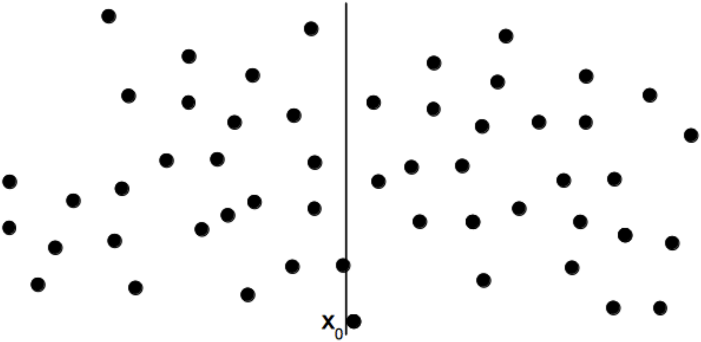
Quicksort

- Let us revisit quicksort:
  - It took the array to sort
  - Chose a pivot
  - Separated into lower and higher elements (one comparison) per element
  - Divided the array in 2: lower or equal than pivot, and the rest
  - Reapplied the pivot technique to subsets
  - UNTIL we have single elements (trivial case)

The closest points problem

- Given a set of points, find the two points lying closest to each other.
- Simplest solution: compare all pairs$O(N^2)$
- The algorithm we will present has worst case analysis of $O(NlogN)$
- From the $logN$ one can deduce that the method is divide and conquer
- So this is how we do it:

The closest points problem

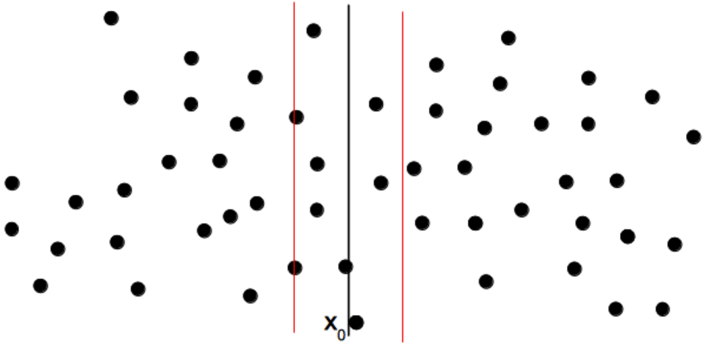


Idea: we subdivide the points in two, according to their x coordinate ($x = x_0$)

The closest points problem

- The closest points are either closest in one of the two subdivisions, or across them
- Let's call the distance between the closest points of the left hand side d_{Lmin}, and on the right hand side d_{Rmin}.
- Both d_{Lmin} and d_{Rmin} are computed by recursion.
- If instead the minimum distance is between one point of the left hand side and one of the right hand side how do I check it?
- Obviously I cannot check all N^2 pairs!

The closest points problem



The points must be in the strip $|x_0 - Min(d_{Lmin}, d_{Rmin})|$

The closest points problem

- If the points are sorted by y coord, then one needs to check only if the current point lies within the strip of points with distance $< Min(d_{Lmin}, d_{Rmin})$
- This results in a strip with y distance $< Min(d_{Lmin}, d_{Rmin})$
- Scan points in increasing y, and compute all distances with all other points in the strip $<$ min in the other partition and being below the curr. point
- Such points are not many, and the more the points, the smaller the strip

The closest points problem

- The problem is that recursive sorting WRT y takes time, i.e. $O(Nlog^2N)$.
- However, a good example for doing this was Mergesort.
- The set of points is subdivided, then calls itself recursively to sort again y coords AND to find nearest pair, and then shuffles results, to find complete sorting on y.
- This avoids double sorting, and brings computational time down to $O(NlogN)$

Polynomials arithmetics

- One application of divide and conquer methods is in polynomials algebra
- Here we want to do algorithms for the addition and multiplication of
  - Polynomials
  - Long integers
  - Matrices

Polynomials arithmetics

- Polynomials are represented as arrays: $\sum_{i=0,...,N} P_i X^i - P[0, ..., N]$

SUM: piece of cake, simply add on same components

Multiplication: how does this work? Not too different! Remember which components give which exponent

Polynomial arithmetics

- By list implementation of polynomials more difficult, but also feasible.
- Here, of course each node has attached to it its degree, so that many coefficients may fall out without taking up space
- Resulting algorithms do a list traversing

- This takes however quite some time... it's a quadratic algorithm.

Evaluation+Interpolation of Polynomials

- Let us now take a completely different approach.

- Following problem: evaluate in x
  $p(x) = x^4 + 3x^3 - 6x^2 + 2x + 1$
- One can do it directly
- However, most commonly used form is to group parentheses:
  $p(x) = x(x(x(x + 3) - 6) + 2) + 1$
- This is called the Horn scheme

Evaluation+Interpolation of Polynomials

- Note that some acceleration can be achieved even in the computation of the powers of x:
    - One can compute some basic powers of x, for example its powers of 2
    - And then use them to compute the powers in between (through binary representation of exponent)

Multiplication of Polynomials

- The algorithm we presented required quadratic time
- One can use divide and conquer:
  - Subdivide polynomial of degree N in N/2 higher order and lower order exponentials
    $p_1(x) = p_0 + p_1 x + ... + p_{N/2-1} x^{(n/2)-1}$
    $p_h(x) = p_{n/2} + p_{(n/2)+1} ... + p_{N-1} x^{(n/2)-1}$

Multiplication of Polynomials

- The original polynomial becomes
  $p(x) = p_l(x) + x^{N/2} p_h(x)$
  Similarly,
  $q(x) = q_l(x) + x^{N/2} q_h(x)$
- And we obtain that $p(x)q(x) =$
  $p_l(x)q_l(x) + (p_l(x)q_h(x) + q_l(x)p_h(x))x^{N/2} + p_h(x)q_h(x)x^N$
- Here 3 multiplications, let's write more clear:

Multiplication of Polynomials

- Set:
    - $r_l = p_l(x)q_l(x), r_h(x) = p_h(x)q_h(x)$, and
      $r_m(x) = (p_l(x) + p_h(x))(q_l(x) + q_h(x))$
      and we have
      $p(x)q(x) = r_l(x) + (r_m(x) - r_l(x) - r_h(x))x^{N/2} + r_h(x)x^N$
    - These are polynomials of lower order
- Continue recursively, until end
- Divide and conquer

Multiplication of Polynomials

- Summary: the multiplication of 2 polynomials of degree N can be split in 3 subproblems of degree $N/2$,
- with the addition of a couple of sums of polynomials
- Speed? $\sim N^{lg3} = N^{1.58}$

Fast Fourier Transform (FFT)

- Fourier transforms are very popular in signal processing
- Complex mathematical background
- Basically, expression of a function as linear sum of *basic functions*
- Applications: from Image and Sound Filtering, to Data Compression, to Signal Analysis:
  we use it for multiplying polynomials

FFT preliminaries

- Idea: a polynomial of degree N-1 is determined by N points
- If 2 polynomials of degree N-1 are multiplied, we obtain a polynomial of degree 2N-2
- The new polynomial is determined through 2N-1 points
- This leaves open ONE point. This point can be determined by multiplying the values of the factoring polynomials in that point.

FFT preliminaries

- Thus, we can multiply 2 polynomials p and q of degree N-1 by doing:
    - Compute the values of p and q in 2N-1 points (equal for p and q)
    - Multiply obtained values
    - Find out polynomial passing through resulting points

FFT preliminaries

- Example: Let$p(x) = 1 + x + x^2, q(x) = 2 - x + x^2$
- Compute p and q at $x = -2, -1, 0, 1, 2$:
  $p : [3, 1, 1, 3, 7]$ and $q : [8, 4, 2, 2, 4]$
- Multiply:$p * q = [24, 4, 2, 6, 28]$
- Use Lagrange formula to determine $p * q$:

FFT preliminaries

$$p * q = +24\frac{x+1}{-2+1}\frac{x-0}{-2-0}\frac{x-1}{-2-1}\frac{x-2}{-2-2} +4\frac{x+2}{-1+2}\frac{x-0}{-1-0}\frac{x-1}{-1-1}\frac{x-2}{-1-2}$$
$$+2\frac{x+2}{0+2}\frac{x-1}{0+1}\frac{x-1}{0-1}\frac{x-2}{0-2} +6\frac{x+2}{1+2}\frac{x+1}{1+1}\frac{x-0}{1-1}\frac{x-2}{1-2} +28\frac{x+2}{2+2}\frac{x+1}{2+1}\frac{x-0}{2-0}\frac{x-2}{2-1}$$
$$\Rightarrow 2 + x + 2x^2 + x^4$$

Not a great improvement(best algo. seen before works in N^2), but on right track

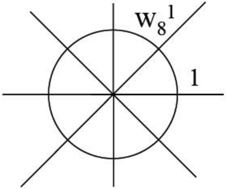
Complex numbers

- Consider $i = \sqrt{-1}$ imaginary number
- $i^2 = -1$
- Complex number: $z = a + ib$ (wh. $a = Rez$, $b = Imz$ real numbers)
- $(a + ib) + (c + id) = (a + c) + i(b + d)$
- $(a + ib) * (c + id) = (ac - bd) + (ad + bc)i$
- Sometimes multipl. two complex numbers the Re or Im disappear
- So for example $[\frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}]^8 = 1$

Complex numbers

- These are called roots of the unity
- NB: For all N, $\exists$ N complex numbers z ST $z^N = 1$
- One of them, w_N , is called main unit root and the others are ST they are $w_N^k (k = 0..N-1)$

- For example: eighth roots are w_8^0 , w_8^1 , w_8^2 , w_8^3 , w_8^4 , w_8^5 , w_8^6 , w_8^7
- Where $w_8^0 = 1$, and w_8^1 =main root
- Also, $w_N^{\frac{N}{2}}$ =1 for N even



Computing unit roots

- Next task: compute unit roots
- Why? We want to write a procedure to compute a polynomial of degree N-1 for the N-th unit root.
- Such procedure transforms the N coefficients of the polynomial into the N values, derived from all computations of the polynomials from N values
- We use a divide and conquer strategy

Computing unit roots

- We subdivide into even and odd powers

$$p(x) = \sum_{i=0,\ldots,N} p_i x^i = \sum_{i=0,\ldots,N/2} p_{2i} x^{2i} +$$

$$x * \sum_{i=0,\ldots,N/2} p_{2i+1} x^{2i+1} = p_e(x^2) + x p_o(x^2)$$

- The strategy is a divide and conquer one
- Note that powers of a unit root are unit roots
- To compute a polynomial of N coeff. in N points, we subdivide in 2 with N/2 coeff.

Computing unit roots

- The new polynomials need N/2 points, and we subdivide further until the case is clear, i.e. till N=2 and $p_0 + p_1 x$ has to be computed
- this requires a power of 2 exponent for the exponential
- To summarize, in the end one can use this to interpolate polynomials

Interpolating polynomials

- The resulting algorithm does the necessary computations
    - Compute polynomials for the (2N-1) unit root
    - Multiply by the values obtained for each point
    - Interpolate to obtain result, by computing the polynomial for the computed value as defined from the (2N-1) unit root, which look like:

$$w_N^i = \cos\left(\frac{2\pi j}{N+1}\right) + i\sin\left(\frac{2\pi j}{N+1}\right)$$

Interpolating polynomials

- Efficiency: 2N lg N + O(N)
- This algorithm is very important: it can be used to compute the Fourier Transform of a signal
- Where here, signal can be:
  - A picture, where the grey values of the picture are the function values
  - This can be used for example for filtering

Discrete Fourier Transform

- Consider a periodic signal F(x) sample at N values $x_0, \dots, x_{N-1}$ where N is a power of 2
- The discrete Fourier Transform is given by the formula
  $$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$
  where k is an integer between 0 and N-1
- The DFT can be computed by first computing the DFT of the even numbers x_{2m} and then of the odd numbers x_{2m+1}
- This can be done recursively, until a simple case is reached

Discrete Fourier Transform

- The separation of odd and even parts is done like this:

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

- Notice that the first and the second sums are DFTs themselves, but for half the sample points: in fact, if we set M=N/2 we get

$$X_k = \sum_{m=0}^{M-1} x_{2m} e^{-\frac{2\pi i}{N} mk} + e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{M-1} x_{2m+1} e^{-\frac{2\pi i}{N} mk}$$

  and it is easy to recognize that the two factors are DFTs themselves with half the number of samples

Discrete Fourier Transform

- We have thus converted the problem of computing the DFT of N samples into two different DFTs of half the size of simples.
- This we have here too a divide and conquer technique, but here the algorithm works in interleaved separation (odd/even numbers)
- One goes on subdividing until the elements are two, for which the DFT is computed directly