CMPSCI 311: Introduction to Algorithms

Lecture 11: Divide and Conquer III

Akshay Krishnamurthy

University of Massachusetts

Last Compiled: March 6, 2018

Announcements

- Midterms graded
  - Regrades?
  - Solutions
- Homework 3 due wednesday
- No quiz tonight, yes discussion

Divide and Conquer Recap

- Given a problem of an input of size n,
  - We generate (multiple) smaller instances of the problem
  - We solve each of these smaller instances
  - We use the solutions of the small instances to solve the original problem.
- Suppose that the first and third steps can be performed in $O(n^\alpha)$ time. If there are q smaller instances generated, each of size n/k, then the running time $T(n)$ of the algorithm satisfies the recurrence.

$$T(n) \le qT(n/k) + cn^\alpha$$

Divide and Conquer: Recurrences

- Suppose $T(n) \le qT(n/2) + n^\alpha$ and $T(1) \le 1$. Then:

$$T(n) = \begin{cases} O(n^\alpha) & \text{if } \alpha > \log_2 q \\ O(n^{\log_2 q}) & \text{if } \alpha < \log_2 q \\ O(n^\alpha \log n) & \text{if } \alpha = \log_2 q \end{cases}$$

- If you forget this formula just apply the "unrolling method":

$$\begin{aligned} T(n) &\le qT(n/2) + n^\alpha \\ &\le q(qT(n/4) + (n/2)^\alpha) + n^\alpha \\ &\le q(q(qT(n/8) + (n/4)^\alpha) + (n/2)^\alpha) + n^\alpha \\ &\le \ldots \end{aligned}$$

- Some example recurrence: $T(n) \le T(n/2) + 1$ and $T(n) \le 4T(n/2) + n$

Divide and Conquer Algorithms

- Mergesort, Maximum Subsequence Sum
- Integer Multiplication
- Minimum distance
- Today: Counting inversions

Minimum Distance Recap

- **Problem:** Given n distinct points $p_1, \ldots, p_n \in \mathbb{R}^2$, find

  minimum distance between any two points $= \min_{i \ne j} d(p_i, p_j)$

$$d(p, q) = \sqrt{(p[1] - q[1])^2 + (p[2] - q[2])^2}$$

Naive algorithm takes $O(n^2)$
But we can do $O(n \log_2 n)$.

Minimum Distance Algorithm

- Divide points P with a vertical line into P_L and P_R where $|P_L| = |P_R| = n/2$
- Recursively find minimum distance within P_L and P_R:

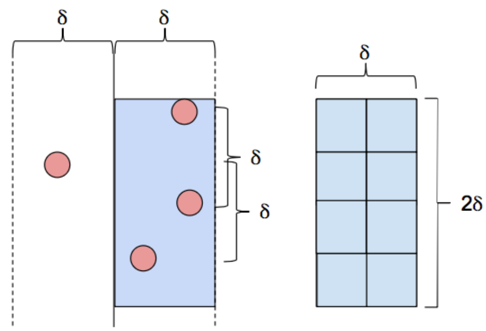
$$\delta_L = \min_{p,q \in P_L : p \neq q} d(p,q) \quad \delta_R = \min_{p,q \in P_R : p \neq q} d(p,q)$$

- Compute $\delta_M = \min_{p \in P_L, q \in P_R} d(p,q)$ and return

$$\min(\delta_L, \delta_R, \delta_M)$$

- Key idea: Can make step 3 $O(n)$-time.
  - Proof requires "packing" argument.

Packing picture



Counting Inversions

- Consider a music recommendation system that works as follows
  - When you join the service, they ask you to rank n songs
  - Based on this ranking, they identify people with similar music preferences
- How to measure "similar"in a large database? Count inversions
  - My ranking is: $1, 2, \ldots, n$
  - Your ranking is: $a_1, a_2, \ldots, a_n$ $(a_i \in \{1, \ldots, n\})$
  - An *inversion* is a pair (i, j) where $i < j$ but $a_i > a_j$.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

- Inversions at $B - D$ and $C - D$.

Counting Inversions

- Question: With list of length n, maximum number of inversions?

	A	B	C	D	E
me	1	2	3	4	5
you	5	4	3	2	1

- Answer: $\Theta(n^2)$
- Brute force algorithm: Check all $n(n-1)/2$ possibilities
  - $\Theta(n^2)$ runtime.

Counting Inversions: Divide and Conquer

- Divide: Split list in two halves L, R
- Recurse: Count inversions in each half
- Combine: Count inversions (ℓ, r) with $\ell \in L, r \in R$.
- Return the sum

1	5	4	8	10	2	6	9	3	7

- Challenge: How to do combine step quickly?

The combine step?

- Challenge: How to do combine step quickly?
- What if L and R were sorted?

1	5	4	8	10	2	6	9	3	7

1	5	4	8	10
1	4	5	8	10

2	6	9	3	7
2	3	6	7	9

- Combine step: $4 + 4 + 2 + 2 + 1 = 13$.
- Can be done in $O(n)$ time!

Inversions Divide and Conquer

- Divide: Split list in two halves L, R
- Recurse: Count inversions in each half and sort each half!
- Combine: Count inversions (ℓ, r) with $\ell \in L, r \in R$.
- Return the sum and sorted list.
- Notes
  - Solve "harder" problem to make your life easier later.
  - Important: Count inversions *before* sorting!

Pseudocode

```
if length(Arr) ≤ 2 then                                         ▷ Base case
    run brute force algorithm return inversions and sorted list.
else
    middle = length(Arr)/2                                      ▷ Recursive Steps
    (c_ℓ, L) = CountAndSort(Arr[0:middle])
    (c_r, R) = CountAndSort(Arr[middle:length(Arr)])
    ℓ = 1, r = 1, C = [], c_m = 0                               ▷ Combine step
    while ℓ ≤ n/2, r ≤ n/2 do
        if L[ℓ] < R[r] then
            C.append(L[ℓ]), ℓ = ℓ + 1
        else
            c_m = c_m + (n/2 − ℓ + 1), C.append(R[r]), r = r + 1
        end if
    end while
    Return (c_ℓ + c_r + c_m, C)
end if
```

Runtime

- Two recursive calls of size $n/2$
- Combine step takes $O(n)$ times
- Recurrence:
$$T(n) \leq 2T(n/2) + cn$$
- Runtime: $O(n \log n)$ – same as merge sort.

Divide and Conquer Wrap-up

- Intution: Solve subproblems and combine together
  - Combine step can be tricky!
- Runtime analysis: Solving recurrence relations
- Other problems: Convolutions and FFT, Quicksort, Median find

Algorithm Design Techniques

- Greedy
- Divide and Conquer
- Dynamic Programming
- Network Flows

Divide and Conquer Recipe

- Devise recursive form for solution
- Implement recursion

Example. Compute sum of leaf weights for each internal node in k-ary tree. (From practice exam)

- Recursive form $w(v) = \sum_{u \text{ child of } v} w(u)$.

Dynamic Programming Recipe

- Devise recursive form for solution
- Observe that recursive implementation involves redundant computation. (Often exponential time)
- Design iterative algorithm that solves all subproblems without redundancy.

Example (From HW1)

Problem. Given array A of length n, compute matrix B with $B[i,j] = A[i] + \ldots + A[j]$ for $i < j$.

for $i = 1, 2, \ldots, n$ **do**
  for $j = i + 1, \ldots, n$ **do**
    Add up $A[i] + A[i+1] + ... + A[j]$. Store in $B[i,j]$.
  end for
end for

Running time: $\Theta(n^3)$.

Example (From HW1)

Problem. Compute B with $B[i,j] = A[i] + \ldots + A[j]$ for $i < j$.

$$B[i,j] = \begin{cases} B[i,j-1] + A[j] & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}$$

for $i = 1, 2, \ldots, n$ **do**
  $B[i,i] = 0$
  for $j = i + 1, \ldots, n$ **do**
    Add up $B[i,j-1] + A[j]$. Store in $B[i,j]$.
  end for
end for

Running time: $O(n^2)$