CSC 373 H 1 Y — Summer 2006

University of Toronto — St. George Campus

Lecture Summary for Week 7

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

Tutorial this week: Counting Inversion [Section 5.3]

Integer Multiplication Problem [Section 5.5]

Input Two numbers in binary:

$$A = a_{n-1} \ \ldots \ a_0 \qquad B = b_{n-1} \ \ldots \ b_0$$

Output The product $A \cdot B$.

Note that

$$A = a_{n-1} \ \ldots \ a_0 = \sum_{i=0}^{n-1} a_i 2^i, \qquad B = b_{n-1} \ \ldots \ b_0 = \sum_{i=0}^{n-1} b_i 2^i \tag{1}$$

The "School" Algorithm

1. Compute the "table" with n rows, where row i is $R_i = A \cdot b_i 2^i$, for $0 \leq i \leq n-1$. In other words,
$$R_i = \begin{cases} 0 & \text{if } b_i = 0 \\ A \text{ shifted by } i \text{ bits} & \text{if } b_i = 1 \end{cases}$$
2. Compute the sum of n rows $R_0 + R_1 + \ldots + R_{n-1}$ by performing $n-1$ additions.

Note that in general the rows have $\Theta(O)$ bits, so adding two rows takes time $\Theta(n)$. Therefore adding n rows takes time $\Theta(n^2)$.

Divide-and-Conquer Algorithms We use the following observation. Assume that n is even. First, let A_1, A_0, B_1, B_0 be the $(n/2)$-bits numbers:

$$A_1 = a_{n-1} \ \ldots \ a_{n/2} \qquad A_0 = a_{n/2-1} \ \ldots \ a_0$$
$$B_1 = b_{n-1} \ \ldots \ b_{n/2} \qquad B_0 = b_{n/2-1} \ \ldots \ b_0$$

then

$$A = A_1 2^{n/2} + A_0 \qquad B = B_1 2^{n/2} + B_0$$

Now

$$A \cdot B = (A_1 \cdot B_1)2^n + (A_1 \cdot B_0 + A_0 \cdot B_1)2^{n/2} + A_0 \cdot B_0 \tag{2}$$

The First Attempt Assuming the length n is a power of 2. Using (2), we will compute $A \cdots B$ recursively as follows:

Multiply1(A, B)

1. If $n = 1$ Return $a_0 \cdot b_0$

2. Else
3. Compute A_1, A_0, B_1, B_0
4. $C \leftarrow$ Multiply1(A_1, B_1)
5. $D_1 \leftarrow$ Multiply1(A_1, B_0)
6. $D_2 \leftarrow$ Multiply1(A_0, B_1)
7. $E \leftarrow$ Multiply1(A_0, B_0)
8. Return $(C2^n + (D_1 + D_2)2^{n/2} + E)$.
9. End If

Running time Using the Master Theorem, the running time $T(n)$ satisfies

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

So $T(n) = \Theta(n^2)$. Assymptotically this does not improve on the "school algorithm" above.

The Second Attempt The improvement comes from the observation that

$$A_1 \cdots B_0 + A_0 \cdots B_1 = (A_1 + A_0) \cdot (B_1 + B_0) - (A_1 \cdots B_1 + A_0 \cdots B_0)$$

Thus the number of recursive calls is reduced to 3.

Multiply2(A, B)

1. If $n = 1$ Return $a_0 \cdot b_0$
2. Else
3. Compute A_1, A_0, B_1, B_0
4. $C \leftarrow$ Multiply2(A_1, B_1)
5. $E \leftarrow$ Multiply2(A_0, B_0)
6. $D \leftarrow$ Multiply2$(A_1 + A_0, B_1 + B_0) - (C + E)$
7. Return $(C2^n + D2^{n/2} + E)$.
8. End If

Running time The running time $T(n)$ of Multiply2 satisfies

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

The Master Theorem gives $T(n) = \Theta(n^{\log_2 3})$.

Clostest Pair of Points [Section 5.4]

Input A set P of n points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$.
Output A pair p_i, p_j of smallest distance.

For simplicity of the arguments below, we will assume that the points in P have distinct x-coordinates.

Brute-force algorithm Try all pairs. Running time: $\Theta(n^2)$.

Divide-and-Conquer: The idea

1. Divide P into two halves Q, R by a vertical line ℓ.
2. Compute recursively the closest pairs for Q and R
3. Compute the closest pair (p_i, p_j) where $p_i \in Q, p_j \in R$
4. Output the closest pair from step 2 and 3

There are several issues:

- How to compute Q and R efficiently ?
- How to carry out step 3 efficiently ?

Note that to compute the two halves Q and R, one way is to sort the given points in increasing order of the x-coordinates. Since the set of input points is fixed, it is better to sort them only once before calling the recursive procedure (instead of sorting them at every recursive call).

For the computation in step 3 above, note that for a set of n points, Q and R contain $n/2$ points each, so there are $(n/2)^2 = \Theta(n^2)$ pairs across Q and R. Suppose that in step 3 we simply check all of these $\Theta(n^2)$ pairs, then the running time $T(n)$ of the recursive procedure would satisfy

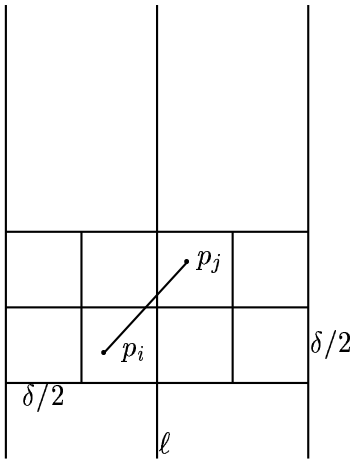
$$T(n) = 2T(n/2) + \Theta(n^2)$$

The Master Theorem gives us that $T(n) = \Theta(n^2)$. This is not (assymptotically) better than the brute-force algorithm above.

The improvement comes from the fact that step 3 can be carried in linear time.

First, let δ be the smallest distance from step 2. In step 3, we are only looking for pairs that have distance $< \delta$. Therefore we can focus on points in Q and R that have distance $< \delta$ from the dividing line ℓ.

Thus, let S be the set of points in P that have distance $< \delta$ from ℓ. Consider two point p_i, p_j in S that have distance $d(p_i, p_j) < \delta$. Then p_i, p_j must lie in a $\delta \times 2\delta$ box B.



Note that every two points in Q have distance $\geq \delta$. Similarly for every two points in R. Consequently, each $\frac{\delta}{2} \times \frac{\delta}{2}$ box in the $\delta \times 2\delta$ box B contains at most 1 point from Q and 1 point from R. So in B there are at most 4 points from Q and 4 points from R. Hence, in total, the $\delta \times 2\delta$ box B contains at most 8 points from S.

From this observation, suppose that we sort S in increasing order of y-coordinates, then the two points p_i and p_j cannot be more than 6 indices apart. Thus, once S has been sorted in this order, step 3 can be done in linear time.

The final issue is that sorting S takes time $\mathcal{O}(n \log(n))$, since S may contains all n points from P. As we did for step 2, here we will sort P in (increasing) y-coordinate before starting the recursive algorithm. This can be used to compute the sorted order of S in linear time.

The algorithm Closest-Pair(P)

1. $P_X \leftarrow P$ in increasing order of x-coordinate.
2. $P_Y \leftarrow P$ in increasing order of y-coordinate.
3. return Closest-Pair-Rec(P, P_X, P_Y).

Closest-Pair-Rec(P, P_X, P_Y)

1. If $|P| \leq 3$ then compute the closest pair in P by brute force.
2. Else
3. Compute line ℓ dividing P into Q, R
4. Compute Q_X, Q_Y and R_X, R_Y from P_X and P_Y.
5. $\langle q_0, q_1 \rangle$ = Closest-Pair-Rec(Q, Q_X, Q_Y)
6. $\langle r_0, r_1 \rangle$ = Closest-Pair-Rec(R, R_X, R_Y)
7. $\delta = min\{d(q_0, q_1), d(r_0, r_1)\}$
8. S = points in P within distance δ to ℓ in increasing order of y-coordinates
9. For each s in S,
10. Compute distances from s to the next 7 points
11. Let $\langle s_0, s_1 \rangle$ be the closest pairs in S
12. End For
13. return the closest of $\langle q_0, q_1 \rangle$, $\langle r_0, r_1 \rangle$ and $\langle s_0, s_1 \rangle$
14. End If

Exercise Verify that computing Q_X, Q_Y, R_X, R_Y (step 4) can be done in time $\mathcal{O}(n)$. Verify that computing S as in step 8 can also be done in time $\mathcal{O}(n)$.

Running time: The running time $T(n)$ for Closest-Pair-Rec satisfies

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

Hence $T(n) = \mathcal{O}(n \log(n))$. The sorting (steps 1 and 2) in the procedure Closest-Pair takes time $\Theta(n \log(n))$. Altogether, the running time is $\Theta(n \log(n))$.