Chapter 35

Exercises - Miscellaneous

By Sariel Har-Peled, December 9, 2012[1] Version: 1.0

35.1 Data structures

35.1.1 Furthest Neighbor

[**20 Points**]

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in the plane.

(a) [**10 Points**] A *partition* $\mathcal{P} = (S, T)$ of P is a decomposition of P into two sets $S, T \subseteq P$, such that $P = S \cup T$, and $S \cap T = \emptyset$.

Describe a *deterministic*[2] algorithm to compute $m = O(\log n)$ partitions $\mathcal{P}_1, \ldots, \mathcal{P}_m$ of P, such that for any pair of distinct points $p, q \in P$, there exists a partition $\mathcal{P}_i = (S_i, T_i)$, where $1 \leq i \leq m$, such that $p \in S_i$ and $q \in T_i$ or vice versa (i.e., $p \in T_i$ and $q \in S_i$). The running time of your algorithm should be $O(n \log n)$.

(b) [**10 Points**] Assume that you are given a black-box $\mathcal{B}$, such that given a set of points Q in the plane, one can compute in $O(|Q| \log |Q|)$ time, a data-structure $\mathcal{X}$, such that given any query point w in the plane, one can compute, in $O(\log |Q|)$ time, using the data-structure, the furthest point in Q from w (i.e., this is the point in Q with largest distance from w). To make things interesting, assume that if $w \in Q$, then the data-structure does not work.

Describe an algorithm that uses $\mathcal{B}$, and such that computes, in $O(n \log^2 n)$ time, for *every* point $p \in P$, its furthest neighbor f_p in $P \setminus \{p\}$.

[1]This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit `http://creativecommons.org/licenses/by-nc/3.0/` or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

[2]There is a very nice and simple randomized algorithm for this problem, you can think about it if you are interested.

35.1.2 Free lunch.

[**10 Points**]

1. [**3 Points**] Provide a *detailed* description of the procedure that computes the *longest ascending subsequence* in a given sequence of n numbers. The procedure should use only arrays, and should output together with the length of the subsequence, the subsequence itself.

2. [**4 Points**] Provide a data-structure, that store pairs (a_i, b_i) of numbers, such that an insertion/deletion operation takes $O(\log n)$ time, where n is the total number of elements inserted. And furthermore, given a query interval $[\alpha, \beta]$, it can output in $O(\log n)$ time, the pair realizing

$$\max_{(a_i,b_i)\in S, a_i \in [\alpha,\beta]} b_i,$$

   where S is the current set of pairs.

3. [**3 Points**] Using (b), describe an $O(n \log n)$ time algorithm for computing the longest ascending subsequence given a sequence of n numbers.

35.2 Divide and Conqueror

35.2.1 Divide-and-Conquer Multiplication

1. [**5 Points**] Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications. (Hint: One of the multiplications is $(a + b) \cdot (c + d)$.)

2. [**5 Points**] Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound n that run in time $\Theta(n^{\lg 3})$. The first algorithm should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is odd or even.

3. [**5 Points**] Show that two n-bit integers can be multiplied in $O(n^{\lg 3})$ steps, where each step operates on at most a constant number of 1-bit values.

35.3 Fast Fourier Transform

35.3.1 3sum

Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the *Cartesian sum* of A and B, defined by

$$C = \{x + y : x \in A \text{ and } y \in B\}.$$

Note that the integers in C are in the range from 0 to $20n$. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Show that the problem can be solved in $O(n \lg n)$ time. (Hint: Represent A and B as polynomials of degree at most $10n$.)

35.3.2 Common subsequence

Given two sequences, $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$ of real numbers, We want to determine whether there is an $i \geq 0$, such that $b_1 = a_{i+1}, b_2 = a_{i+2}, \ldots, b_m = a_{i+m}$. Show how to solve this problem in $O(n \log n)$ time with high probability.

35.3.3 Computing Polynomials Quickly

In the following, assume that given two polynomials $p(x), q(x)$ of degree at most n, one can compute the polynomial remainder of $p(x) \bmod q(x)$ in $O(n \log n)$ time. The ***remainder*** of $r(x) = p(x) \bmod q(x)$ is the unique polynomial of degree smaller than this of $q(x)$, such that $p(x) = q(x) * d(x) + r(x)$, where $d(x)$ is a polynomial.

Let $p(x) = \sum_{i=0}^{n-1} a_i x^i$ be a given polynomial.

1. [**4 Points**] Prove that $p(x) \bmod (x - z) = p(z)$, for all z.

2. [**4 Points**] We want to evaluate $p(\cdot)$ on the points $x_0, x_1, \ldots, x_{n-1}$. Let

$$P_{ij}(x) = \prod_{k=i}^{j} (x - x_k)$$

and

$$Q_{ij}(x) = p(x) \bmod P_{ij}(x).$$

Observe that the degree of Q_{ij} is at most $j - i$.

Prove that, for all x, $Q_{kk}(x) = p(x_k)$ and $Q_{0,n-1}(x) = p(x)$.

3. [**4 Points**] Prove that for $i \leq k \leq j$, we have

$$\forall x \quad Q_{ik}(x) = Q_{ij}(x) \bmod P_{ik}(x)$$

and

$$\forall x \quad Q_{kj}(x) = Q_{ij}(x) \bmod P_{kj}(x).$$

4. [**8 Points**] Given an $O(n \log^2 n)$ time algorithm to evaluate $p(x_0), \ldots, p(x_{n-1})$. Here $x_0, \ldots, x_{n-1}$ are n given real numbers.

35.4 Union-Find

35.4.1 Linear time Union-Find,

[**20 Points**]

1. [**2 Points**] With path compression and union by rank, during the lifetime of a Union-Find data-structure, how many elements would have rank equal to $\lfloor \lg n - 5 \rfloor$, where there are n elements stored in the data-structure?

2. [**2 Points**] Same question, for rank $\lfloor (\lg n)/2 \rfloor$.

3. [**4 Points**] Prove that in a set of n elements, a sequence of n consecutive FIND operations take $O(n)$ time in total.

4. [**2 Points**]
   Write a non-recursive version of FIND with path compression.

5. [**6 Points**] Show that any sequence of m MAKESET, FIND, and UNION operations, where all the UNION operations appear before any of the FIND operations, takes only $O(m)$ time if both path compression and union by rank are used.

6. [**4 Points**] What happens in the same situation if only the path compression is used?

35.4.2 Off-line Minimum

[**20 Points**]

The *off-line minimum problem* asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations INSERT and EXTRACT-MIN. We are given a sequence S of n INSERT and m EXTRACT-MIN calls, where each key in $\{1, 2, \ldots, n\}$ is inserted exactly once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array *extracted*$[1 \ldots m]$, where for $i = 1, 2, \ldots, m$, *extracted*$[i]$ is the key returned by the ith EXTRACT-MIN call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

1. [**4 Points**]
   In the following instance of the off-line minimum problem, each INSERT is represented by a number and each EXTRACT-MIN is represented by the letter E:

$$4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5.$$

   Fill in the correct values in the *extracted* array.

2. **[8 Points]**
   To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by

   $I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}$,

   where each E represents a single EXTRACT-MIN call and each I_j represents a (possibly empty) sequence of INSERT calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following.

```
OFF-LINE-MINIMUM($m$,$n$)
1  for $i \leftarrow 1$ to $n$
2      do determine $j$ such that $i \in K_j$
3         if $j \neq m + 1$
4            then extracted[$j$] $\leftarrow i$
5                 let $l$ be the smallest value greater than $j$ for which set $K_l$ exists
6                 $K_l \leftarrow K_j \cup K_l$, destroying $K_j$
7  return extracted
```

   Argue that the array *extracted* returned by OFF-LINE-MINIMUM is correct.

3. **[8 Points]**
   Describe how to implement OFF-LINE-MINIMUM efficiently with a disjoint-set data structure. Give a tight bound on the worst-case running time of your implementation.

35.4.3 Tarjan's Off-Line Least-Common-Ancestors Algorithm

[20 Points]

The *least common ancestor* of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the *off-line least-common-ancestors problem*, we are given a rooted tree T and an arbitrary set $P = \{\{u, v\}\}$ of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P.

To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call LCA(*root*[T]). Each node is assumed to be colored WHITE prior to the walk.

```
LCA(u)
1   MakeSet(u)
2   ancestor[Find(u)] ← u
3   for each child v of u in T
4       do LCA(v)
5           Union(u, v)
6           ancestor[Find(u)] ← u
7   color[u] ← BLACK
8   for each node v such that {u, v} ∈ P
9       do if color[v] = BLACK
10              then print "The least common ancestor of" u "and" v "is" ancestor[Find(v)]
```

1. [**4 Points**] Argue that line 10 is executed exactly once for each pair $\{u, v\} \in P$.

2. [**4 Points**] Argue that at the time of the call LCA(u), the number of sets in the disjoint-set data structure is equal to the depth of u in T.

3. [**6 Points**] Prove that LCA correctly prints the least common ancestor of u and v for each pair $\{u, v\} \in P$.

4. [**6 Points**] Analyze the running time of LCA, assuming that we use the implementation of the disjoint-set data structure with path compression and union by rank.

35.4.4 Ackermann Function

[20 Points]

The Ackermann's function $A_i(n)$ is defined as follows:

$$A_i(n) = \begin{cases} 4 & \text{if } n = 1 \\ 4n & \text{if } i = 1 \\ A_{i-1}(A_i(n-1)) & \text{otherwise} \end{cases}$$

Here we define $A(x) = A_x(x)$. And we define $\alpha(n)$ as a pseudo-inverse function of $A(x)$. That is, $\alpha(n)$ is the least x such that $n \le A(x)$.

1. [**4 Points**] Give a precise description of what are the functions: $A_2(n)$, $A_3(n)$, and $A_4(n)$.

2. [**4 Points**] What is the number $A(4)$?

3. [**4 Points**] ***Prove*** that $\lim_{n\to\infty} \frac{\alpha(n)}{\log^*(n)} = 0$.

4. [**4 Points**] We define

$$\log^{**} n = \min \left\{ i \geq 1 \;\middle|\; \underbrace{\log^* \ldots \log^*}_{i \text{ times}} n \leq 2 \right\}$$

(i.e., how many times do you have to take $\log^*$ of a number before you get a number smaller than 2). ***Prove*** that $\lim_{n\to\infty} \frac{\sqrt{\alpha(n)}}{\log^{**}(n)} = 0$.

5. [**4 Points**] ***Prove*** that $\log(\alpha(n)) \leq \alpha(\log^{**} n)$ for n large enough.

35.5 Lower bounds

35.5.1 Sort them Up

[20 Points]

A sequence of real numbers $x_1, \ldots, x_n$ is k-mixed, if there exists a permutation π, such that $x_{\pi(i)} \leq x_{\pi(i+1)}$ and $|\pi(i) - i| \leq k$, for $i = 1, \ldots, n-1$.

1. [**10 Points**] Give a fast algorithm for sorting $x_1, \ldots, x_n$.

2. [**10 Points**] Prove a lower bound in the comparison model on the running time of your algorithm.

35.5.2 Another Lower Bound

[20 Points]

Let $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_k$ be k given sorted numbers, and let A be a set of n arbitrary numbers $A = \{a_1, \ldots, a_n\}$, such that $b_1 < a_i < b_k$, for $i = 1, \ldots, n$

The rank $v = r(a_i)$ of a_i is the index, such that $b_v < a_i < b_{v+1}$.

Prove, that in the comparison model, any algorithm that outputs the ranks $r(a_1), \ldots, r(a_n)$ must take $\Omega(n \log k)$ running time in the worst case.

35.6 Number theory

35.6.1 Some number theory.

[10 Points]

1. [**5 Points**] Prove that if $\gcd(m, n) = 1$, then $m^{\phi(n)} + n^{\phi(m)} \equiv 1 (\mathrm{mod} mn)$.

2. [**5 Points**] Give two distinct proofs that there are an infinite number of prime numbers.

35.6.2 Even More Number Theory

[10 Points]

Prove that $|P(n)| = \Omega(n^2)$, where $P(n) = \left\{(a,b) \;\middle|\; a,b \in \mathbb{Z}\ , 0 < a < b \le n, \gcd(a,b) = 1\right\}$.

35.6.3 Yet Another Number Theory Question

[20 Points]

1. [**2 Points**] Prove that the product of all primes p, for $m < p \le 2m$ is at most $\binom{2m}{m}$.
2. [**4 Points**] **Using (a)**, prove that the number of all primes between m and $2m$ is $O(m/\ln m)$.
3. [**3 Points**] **Using (b)**, prove that the number of primes smaller than n is $O(n/\ln n)$.
4. [**2 Points**] Prove that if 2^k divides $\binom{2m}{m}$ then $2^k \le 2m$.
5. [**5 Points**] (Hard) Prove that for a prime p, if p^k divides $\binom{2m}{m}$ then $p^k \le 2m$.
6. [**4 Points**] **Using (e)**, prove that that the number of primes between 1 and n is $\Omega(n/\ln n)$. (Hint: use the fact that $\binom{2m}{m} \ge 2^{2m}/(2m)$.)

35.7 Sorting networks

35.7.1 Lower bound on sorting network

[10 Points]

Prove that an n-input sorting network must contain at least one comparator between the ith and $(i+1)$st lines for all $i = 1, 2, ..., n-1$.

35.7.2 First sort, then partition

Suppose that we have $2n$ elements $< a_1, a_2, ..., a_{2n} >$ and wish to partition them into the n smallest and the n largest. Prove that we can do this in constant additional depth after separately sorting $< a_1, a_2, ..., a_n >$ and $< a_{n+1}, a_{n+2}, ..., a_{2n} >$.

35.7.3 Easy points.

[20 Points]

Let $S(k)$ be the depth of a sorting network with k inputs, and let $M(k)$ be the depth of a merging network with $2k$ inputs. Suppose that we have a sequence of n numbers to be sorted and we know that every number is within k positions of its correct position in the

sorted order, which means that we need to move each number at most $(k-1)$ positions to sort the inputs. For example, in the sequence 3 2 1 4 5 8 7 6 9, every number is within 3 positions of its correct position. But in sequence 3 2 1 4 5 9 8 7 6, the number 9 and 6 are outside 3 positions of its correct position.
Show that we can sort the n numbers in depth $S(k) + 2M(k)$. (You need to prove your answer is correct.)

35.7.4 Matrix Madness

[**20 Points**]
We can sort the entries of an $m \times m$ matrix by repeating the following procedure k times:

1. Sort each odd-numbered row into monotonically increasing order.
2. Sort each even-numbered row into monotonically decreasing order.
3. Sort each column into monotonically increasing order.

1. [**8 Points**] Suppose the matrix contains only 0's and 1's. We repeat the above procedure again and again until no changes occur. In what order should we read the matrix to obtain the sorted output ($m \times m$ numbers in increasing order)? Prove that any $m \times m$ matrix of 0's and 1's will be finally sorted.
2. [**8 Points**] Prove that by repeating the above procedure, any matrix of real numbers can be sorted. [Hint:Refer to the proof of the zero-one principle.]
3. [**4 Points**] Suppose k iterations are required for this procedure to sort the $m \times m$ numbers. Give an upper bound for k. The tighter your upper bound the better (prove you bound).

35.8 Max Cut

35.8.1 Splitting and splicing

Let $G = (V, E)$ be a graph with n vertices and m edges. A *splitting* of G is a partition of V into two sets V_1, V_2, such that $V = V_1 \cup V_2$, and $V_1 \cap V_2 = \emptyset$. The cardinality of the split (V_1, V_2), denoted by $m(V_1, V_2)$, is the number of edges in G that has one vertex in V_1, and one vertex in V_2. Namely,

$$m(V_1, V_2) = \left|\left\{ e \;\middle|\; e = \{uv\} \in E(G), u \in V_1, v \in V_2 \right\}\right|.$$

Let $\int\backslash(G) = \max_{V_1} m(V_1, V_2)$ be the maximum cardinality of such a split. Describe a deterministic polynomial time algorithm that computes a splitting (V_1, V_2) of G, such that $m(V_1, V_2) \geq \int\backslash(G)/2$. (Hint: Start from an arbitrary split, and continue in a greedy fashion to improve it.)