Problem Set I

6.897 Algorithms for Massive Data Sets

Due: Thursday, October 24

Grading policy:

- All problems are worth an equal number of points. This does *not* mean that all of them are equally easy or hard. If it helps, I believe that P1 is the easiest one, followed by P2, P3, P0, P5 and P4 (in the increasing order of difficulty).
- In your solutions, focus on main ideas, instead of writing extremely detailed answers. This is a *graduate* class, and I am not going to subtract points because of some small omissions/errors. Therefore, try to avoid writing long solutions. Each time you start another page, think about the rainforest.
- All algorithms can be randomized, with only a constant (say, 1/10) probability of success.

Problem 0: Approximate closest pair $\Rightarrow$ closest pair. Assume you are given an algorithm that, given n points $P = \{p_1 \dots p_n\}$ from $\Re^d$, reports a list L of pairs (p_i, p_j) of points such that:

- If (p_i, p_j) is the closest pair (w.r.t. the Euclidean norm) of distinct points in P, it is contained in L. Formally, $p_i \neq p_j$ is the closest pair of points if $\|p_i - p_j\|_2 = \min_{k \neq l} \|p_k - p_l\|_2$. Note that the closest pair does not have to be unique.
- All pairs (p_i, p_j) in L are such that the distance between them is at most twice the distance between the closest pair of distinct points in P.

Show that if the algorithm has running time $O(n+S)$ where S is the number of reported pairs, then you can use it to design an algorithm that reports all *closest pairs* of distinct points in P. The running time of the resulting algorithm should be $O(n)$ if the dimension d is constant.

Hint: The material from Lectures 2 and 4 should be helpful here.

Problem 1: Fuzzy min-hashing. In Lectures 5 and 6 we have seen a family $\mathcal{H}$ of hash functions $h : U \to S$ with the property that for any two sets $A, B \subset U$, we have

$$\Pr_{h\in\mathcal{H}}[h(A) = h(B)] = \frac{|A \cap B|}{|A \cup B|} \qquad (*)$$

As mentioned in Lecture 6 (slide 3), the approach can be extended to handle *multisets*, i.e., sets in which an element can occur many times. In particular, assume that the set union and set intersection operators are extended to multisets using the following "fuzzy logic" approach:

- The number of occurrences of x in A is denoted by $\mu_A(x)$. I.e., μ_A is the characteristic function of A.
- $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$
- $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$

Show how to redefine the family $\mathcal{H}$ in such a way that the property (*) holds even when A and B are multisets.

Problem 2: Median estimator. Prove the fact from Lecture 7, slide 7:

Fact: Let X be any real random variable with density function continuous over $(-\infty, \infty)$. For $k = C \log(1/\delta)/\epsilon^2$ (C large constant) let $s_1 \dots s_k$ be independently chosen from the same distribution as X. Then $M = median(s_1, \dots, s_k)$ satisfies

$$1/2 - \epsilon \le \Pr[X < M] \le 1/2 + \epsilon$$

with probability $\ge 1 - \delta$.

Hint 1: This fact follows quite easily from Chernoff bound. So, if your solution is overly complicated, chances are that it is wrong.

Hint 2: You can assume that k is odd, to make the $median(s_1, \dots, s_k)$ well-defined.

Problem 3: Locality-Sensitive Hashing in External Memory. In Lecture 6, we have seen that given an (r, cr, P_1, P_2)-sensitive family of hash functions, one can obtain an algorithm for the Near Neighbor problem (with parameters r and c) that performs $O(n^\rho)$ memory accesses and hash function evaluations per query, for $\rho = \frac{\log 1/P_1}{\log 1/P_2}$ (for simplicity, assume that each point can be stored in one memory cell). What happens if we want to use this algorithm when the data and the data structure needs to be stored in the external memory, say, with the block size B ? A naive implementation of LSH would require $O(n^\rho)$ I/O's per query.

Show how to tweak the parameters k and l in the LSH algorithm so that it uses $o(n^\rho)$ I/O's per query if $B = \omega(1)$. Try to get the number of I/O's to be as small as you can (asymptotically, don't worry about the constants). Your data structure should not use more than $O(n^{1+\rho})$ blocks of external memory (but the blocks do not have to be contiguous, if it simplifies your solution).

Problem 4: Closest pair of substrings. Given a string $T[1 \dots n]$ of 0's and 1's, and parameter $m \geq 1$, let $T_m^i = T[i \dots i+m-1]$. I.e., T_m^i is the i-th substring of T of length m. Let $T_m = \{T_m^1, \dots T_m^{n-m+1}\}$. For any $c > 1, r \geq 1$, the goal of the *approximate closest substring pair problem (ACSP)* is to design an algorithm that does the following: if there is a pair $T_m^i \neq T_m^j$ such that $D(T_m^i, T_m^j) \leq r$, the algorithm should output a pair $i' \neq j'$ such that $D(T_m^{i'}, T_m^{j'}) \leq cr$, where $D(\cdot,\cdot)$ is the Hamming distance.

Give an algorithm solving ACSP in time $O(n^{1+1/c}\log n)$. Note: the running time does *not* depend on m, this is not a typo.

Hint 1: Note the similarity between ACSP and the approximate near neighbor problem. In fact, as a warmup, you might want to find an $O(mn^{1+1/c})$-time algorithm first.

Hint 2: You can assume that the following problem can be solved in $O(n \log n)$ time (via FFT): given two sequences $A = a_1 \dots a_n$ and $B = b_1 \dots b_m$, compute all numbers $c_i = \sum_{j=1}^{m} a_{i+j-1} b_j$.

Problem 5: Consider the following heuristic for the k-center problem. The first center c^1 is chosen arbitrarily. The second center c^2 is chosen to maximize $D(c^1, c^2)$. The third center c^3 is chosen to maximize $\min[D(c^1, c^3), D(c^2, c^3)]$ and so on, i.e., at each stage we choose a new center to maximize its distance to the centers chosen so far.

1. Show that if we select $c_1 \dots c_k$ as above, then the maximum distance from any data point p to its closest center is at most twice the optimum (i.e., that the algorithm has the same approximation guarantee as the algorithm seen in Lecture 9).

2. Show that the algorithm can be implemented to run in $O(kn)$ time.