CS 466/666, Spring 2006
Timothy Chan

Assignment 1 (due May 25 Thursday noon)

Please read http://www.student.cs.uwaterloo.ca/~cs466/policies.html first for general instructions.

1. [*25 marks*] Given a directed graph $G = (V, E)$ with n vertices and m edges ($m \geq n$), consider the problem of deciding whether G contains a (directed) cycle of length 4. In this question, you will explore one "high-low" approach to solve this problem. You will first examine special cases or subroutines in parts (a)–(f) below before giving the final algorithm in parts (g)–(h).

   Fix a parameter d. Let H be the set of all vertices of degree $\geq d$ and let L be the set of all vertices of degree $< d$. Let $c_H[u, v] =$ true iff there exists $w \in H$ such that $(u, w) \in E$ and $(w, v) \in E$. Similarly, let $c_L[u, v] =$ true iff there exists $w \in L$ such that $(u, w) \in E$ and $(w, v) \in E$.

   (a) [*4 marks*] Show that we can compute $c_H[u, v]$ for all $u, v \in H$ in $O((m/d)^{2.81})$ time.

   (b) [*3 marks*] Using (a), show that we can decide whether G contains a cycle $v_1v_2v_3v_4$ with $v_1, v_2, v_3, v_4 \in H$ in $O((m/d)^{2.81})$ time.

   (c) [*4 marks*] Show that we can compute $c_L[u, v]$ for all $u, v \in V$ in $O(md + n^2)$ time.

   (d) [*2 marks*] Using (c), show that we can decide whether G contains a cycle $v_1v_2v_3v_4$ with $v_2, v_4 \in L$ in $O(md + n^2)$ time. [Remark: it is possible to eliminate the $O(n^2)$ term, but you are not required to.]

   (e) [*2 marks*] Using both (a) and (c), show that we can decide whether G contains a cycle $v_1v_2v_3v_4$ with $v_1, v_2, v_3 \in H$ and $v_4 \in L$ in $O(md + (m/d)^{2.81} + n^2)$ time.

   (f) [*4 marks*] Show that we can decide whether G contains a cycle $v_1v_2v_3v_4$ with $v_1, v_2 \in L$ in $O(md^2 + n^2)$ time.

   (g) [*3 marks*] Using (b), (d), (e), and (f), conclude that we can decide whether G contains a cycle of length 4 in $O(md^2 + (m/d)^{2.81} + n^2)$ time.

   (h) [*3 marks*] What value of d would you choose to get the best asymptotic running time in (g)?

2. [*12 marks*] Consider the following problem: given a set S of n points in 2D, compute a farthest pair, i.e., two points $p, q \in S$ such that the Euclidean distance between p and q is largest. Prove the following lower bound result: this problem requires $\Omega(n \log n)$ worst-case time, for algorithms that access the input points only through "algebraic primitive operations" (testing the sign of fixed-degree algebraic/polynomial functions).

   You may (and should) use the following known 1D lower bound result on the SetIntersection problem: given two sets A and B of real numbers between 0 and 1 with $n = |A| + |B|$, deciding whether A and B contain a common element (i.e., whether $A \cap B \neq \emptyset$) requires $\Omega(n \log n)$ worst-case time, for algorithms that access the input numbers only through algebraic primitive operations.

Vague Hint: use a reduction (which direction?). Choose points on the circle $x^2 + y^2 = 1$ of radius 1, with elements of A (or B) somehow corresponding to points on the upper (or lower) semi-circle. If the farthest pair p, q has distance exactly equal to 2, what can you say about the x-coordinates of p and q? [Remark: the $\Omega(n \log n)$ lower bound for SETINTERSECTION can be proved by Ben-Or's technique (mentioned in class), but you do not need to know this proof.]

3. [*13 marks*] The *median* of a set S of n elements (real numbers) is an element of rank $\lfloor n/2 \rfloor$. (The *rank* of x refers to the number of elements smaller than x.) It is known (e.g., see [CLRS, Ch9] or any algorithms textbook) that the median can be computed in $O(n)$ time.

   In this question, we study a dynamic version of the problem: how to maintain an approximate median under insertions of new elements. For our purposes, an *approximate median* is defined as an element of rank between $0.4n$ and $0.6n$.

   Consider the following simple "lazy" strategy: don't update the median for the next $0.1n$ elements; after we have seen $0.1n$ elements, recompute the median from scratch (by the $O(n)$ algorithm mentioned above). The precise pseudocode is given below. Initially, $n_0 = n = \text{count} = y = 0$.

   insert(x):
   $n = n + 1$, $A[n] = x$, count $=$ count $+ 1$
   if count $\geq 0.1 n_0$ then {
       $y =$ the (exact) median of $A[1], \ldots, A[n]$
       $n_0 = n$, count $= 0$
   }
   return y

   (a) [*2 marks*] Show that insert() always returns an approximate median of $A[1], \ldots, A[n]$.

   (b) [*1 marks*] What is the worst-case running time of insert()?

   (c) [*4 marks*] Determine the amortized running time of insert(). Use a direct argument that avoids potentials.

   (d) [*6 marks*] Redo part (c), this time using the potential method.