CS 466/666, Spring 2009
Timothy Chan

Assignment 1 (due May 27 Wednesday 5pm)

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

1. [*20 marks*] Let $G = (V, E)$ be an undirected graph with n vertices, where each vertex v has a cost $c[v]$ (a real number). In this question, you will give an algorithm that finds a *cheapest triangle*, i.e., three vertices $u, v, w \in V$ minimizing $c[u] + c[v] + c[w]$ such that $uv, uw, wv \in E$. (If no triangle exists, the output is undefined.) Fix a parameter $d \leq n$ to be determined later.

   (a) [*3 marks*] Given an $n \times d$ matrix A an $d \times n$ matrix B, first show that we can multiply A and B in $O((n/d)^2 \cdot d^{2.376})$ time. [Use the fact that we can multiply two $d \times d$ matrices in $O(d^{2.376})$ time.]

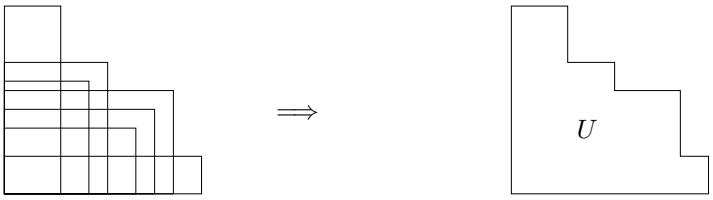
   (b) [*7 marks*] Let $v_1, \ldots, v_n$ be the vertices of V in *increasing* order of cost. Divide V into $\lceil n/d \rceil$ groups $G_1, \ldots, G_{\lceil n/d \rceil}$ each with at most d vertices, where $G_k = \{v_{(k-1)d+1}, \ldots, v_{\min\{kd,n\}}\}$. For every $u, v \in V$, let k_{uv} be the smallest k such that there exists a vertex $w \in G_k$ with $uw, wv \in E$. Give an algorithm that computes all the k_{uv}'s in total time $O((n/d)^3 \cdot d^{2.376})$. [Hint: use part (a).]

   (c) [*6 marks*] For every $u, v \in V$, let b_{uv} be the minimum of $c[w]$ over all $w \in V$ with $uw, wv \in E$. (If no such w exists, then $b_{uv} = \infty$.) Give an algorithm that computes all the b_{uv}'s in total time $O((n/d)^3 \cdot d^{2.376} + dn^2)$. [Hint: use (b).]

   (d) [*2 marks*] What is the best choice of d (asymptotically)?

   (e) [*2 marks*] Conclude that we can solve the cheapest triangle problem in time asymptotically better than n^3.

2. [*24 marks*] Consider the following problem which we will call UNION: we are given a set R of n rectangles, all having the origin as the lower-left vertex, and we want to compute the union U of these rectangles (which is a polygon). More formally, the input is a list of $2n$ real numbers $x_1, y_1, \ldots, x_n, y_n > 0$, where the i-th rectangle has vertices $(x_i, y_i), (x_i, 0), (0, 0), (0, y_i)$. The output is the sequence of vertices of the polygon U in left-to-right order.



   (a) [*4 marks*] Show that if the x-coordinates have already been sorted in decreasing order ($x_1 > x_2 > \cdots > x_n$), then the UNION problem can be solved in $O(n)$ time.

(b) [*14 marks*] For the general UNION problem (where the coordinates are not sorted), give an algorithm that runs in $O(n \log h)$ time where h is the output size. [Hint: The idea should be similar to an algorithm from class.]

(c) [*6 marks*] Prove an $\Omega(n \log n)$ lower bound for the UNION problem. [Hint: use a reduction involving sorting.]

3. [*16 marks*] Fix k. Consider the problem of maintaining a set S of n elements ($n > k$), to support two operations:

   - S.top-k() returns the k largest elements of S (these k elements can be reported in any order);
   - S.insert(x) inserts a new element x to S.

   By explicitly maintaining a sorted ordering of S in a balanced search tree, it is possible to solve the problem in $O(k)$ time for top-k() and $O(\log n)$ time for insert(). However, this data structure requires $O(n)$ space. In this question, you will explore a different data structure that uses only $O(k)$ space.

   Specifically, we maintain a subset $Q \subseteq S$ stored in a linked list. Initially, $Q = \emptyset$. The two operations are implemented as follows:

   S.top-k():
   1. return the k largest elements in Q

   S.insert(x):
   1. insert x to Q
   2. if $|Q| = 2k$ then
   3. compute $Q' =$ the k largest elements in Q
   4. reset $Q \leftarrow Q'$

   (a) [*3 marks*] Argue that this data structure indeed uses $O(k)$ space at all times.

   (b) [*4 marks*] Show that top-k() and line 3 of insert() can be done in $O(k)$ time. [Hint: you may use the known fact that the k-th largest element of a set Q can be found in linear $O(|Q|)$ time; e.g., see [CLRS, Ch9]. How would you use this subroutine to find the first k largest elements of Q?]

   (c) [*9 marks*] Prove that the amortized cost of insert() is $O(1)$ by the potential method. [Hint: the "obvious" choice for the potential should work. Consider two cases and in both cases, express the cost of insert() in terms of the change in potential.]

   [Bonus (3 marks): give a method that supports top-k() in $O(k)$ time and insert() in $O(1)$ *worst-case* (rather than amortized) time.]