BU CLA CS 530: Analysis of Algorithms
Solutions for Mid-Term Test

Problem 1 (5 points).

This is problem 2 in problem set 4. The idea is to use KMP-MATCHER in its original form, without any modification, but the input is now: TT (for input text) and T' (for input pattern). "TT" means two copies of T concatenated, for a total length of $2 \cdot n$. Now, T is a cyclic rotation of T' iff pattern T' occurs in text TT. Concatenating two copies of T takes time at most $\mathcal{O}(n)$, and since the running time of KMP-MATCHER on input TT and T' is $\mathcal{O}(|TT| + |T'|) = \mathcal{O}(n)$, the total running time is still $\mathcal{O}(n)$.

Problem 2 (5 points).

Let $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ be two rational polynomials with n variables. Let every variable in f be of degree at most k, and every variable in g be of degree at most ℓ. Define a new rational polynomial $h(x_1, \ldots, x_n) =_{\text{def}} f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n)$, so that $m = \max\{k, \ell\}$ is the highest degree of any variable in h. Now f and g are equal (i.e. evaluate to the same value for every possible assignment to the variables $x_1, \ldots, x_n$) iff h is identically 0. Here is the desired algorithm:

> We compute $h(\xi_1, \ldots, \xi_n)$ with integer values ξ_i chosen randomly and independently of each other according to the uniform distribution in the interval $[0, 2mn]$. If we don't get the value 0 we stop: f and g are not equal. If we get 0 we repeat the computation. If we get value 0 repeatedly 100 times, we stop and declare that f and g are equal.

The probability that this algorithm will erroneously assert that f and g are equal is less than 2^{100}.

Problem 3 (each part 5 points, total 15 points).

1. This is problem 3, part (a), in problem set 2. First, sort the set S in time at most $\mathcal{O}(n \log n)$. Second, for each entry y in S, perform binary search to find a number equal to $x - y$ in time at most $n \cdot \mathcal{O}(\log n) = \mathcal{O}(n \log n)$. The total time for the two phases is at most $\mathcal{O}(n \log n) + \mathcal{O}(n \log n) = \mathcal{O}(n \log n)$.

2. We need to find two numbers $v, w \in S$ that are "farthest apart". Let $v = \min(S)$ and $w = \max(S)$, each can be found in time $\mathcal{O}(n)$.

3. We need to find two numbers $v, w \in S$ that are "closest together". First sort S in $\mathcal{O}(n \log n)$ time (using heapsort or mergesort or any other sorting algorithm that works in $\mathcal{O}(n \log n)$ time in the worst-case). Then scan the sorted numbers for the two adjacent numbers with the minimum difference, in time at most $\mathcal{O}(n)$. Total time is $\mathcal{O}(n \log n)$.

Problem 4 (each part 5 points, total 15 points).

1. The total running time of FORD-FULKERSON is at most $\mathcal{O}(E \cdot |f^*|)$. But here $|f^*| = \mathcal{O}(V)$, because the flow is bounded by the capacity of any cut, and the capacity of the cut $(s, V - s)$ is $\mathcal{O}(V)$ because there are $\mathcal{O}(V)$ edges leaving the source s and each edge has capacity $= 1$. Hence, the worst-case running time in this case is $\mathcal{O}(E \cdot |f^*|) = \mathcal{O}(E \cdot \mathcal{O}(V)) = \mathcal{O}(VE)$.

2. Just execute one additional iteration of the FORD-FULKERSON algorithm. The new edge in E adds a new edge to the residual graph, so look for an augmenting path and update the flow if a path is found.

   The running time is $\mathcal{O}(V + E) = \mathcal{O}(E)$, whether you find this path by depth-first or breadth-first search.

   Only one iteration is needed because adding an edge of capacity $= 1$ increases the capacity of the min cut by at most 1. Thus the max flow (which equals the capacity of the min cut) increases by at most 1. Since every edge has capacity $= 1$, any augmentation increases the flow by 1, so only at most one augmentation is needed.

3. Let the removed edge be (u, v). If (u, v) had no flow, we don't need to do anything. If (u, v) carried flow, the network must be updated. The picture is something like:

   $$s \longrightarrow \cdots \longrightarrow u \Longrightarrow v \longrightarrow \cdots \longrightarrow t$$

   where the edge "$\Longrightarrow$" is deleted. There is now one more unit of flow coming into u than is going out (and one more unit of flow going out of v than is coming in). The idea is to try to reroute this unit of flow so that it goes out of u and into v via some other path, if possible. If this is not possible, we must reduce the flow from s to u and from v to t by one unit.

   Look for an augmenting path from u to v (not from s to t as we do at every iteration of the basic FORD-FULKERSON method). If there is such a path, augment the flow along that path. If there is no such path, reduce the flow from s to u by augmenting the flow from u to s. That is, find an augmenting path from u to s and augment the flow along that path. (There is definitely such a path, because there is flow from s to u.) Augment the flow from t to v similarly.

   The running time is again $\mathcal{O}(V + E) = \mathcal{O}(E)$, whether you find the two paths, from u to s and from t to v, by depth-first or breadth-first search.

Assaf Kfoury
Created: 95.03.14