Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Hamid Ghodselahi January 9, 2013

Algorithm Theory, Winter Term 2012/13
Problem Set 6

hand in by Wednesday, January 23, 2013

Exercise 1: Linear-Time Contention Resolution [6 Points]

In class, we looked at the following simple contention resolution problem. There are n processes that need to access a shared resource. Time is divided into time slots and in each time slot, a process i can access the resource if and only if i is the only process trying to access the resource. We have shown that if each process independently tries to access the resource with probability $1/n$ in each time slot, by time $O(n \log n)$, all processes can access the resource at least once with high probability. The goal of the exercise is to improve the algorithm and to get an $O(n)$ time algorithm under the following assumptions.

- As in the lecture, all the processes know n (the number of processes). In the algorithm of the lecture, this is needed because the probability $1/n$ for accessing the channel depends on n.
- If a process tries to access the resource in a time slot, the process afterwards knows whether the access was successful or not.

Give a randomized algorithm which guarantees that with probability at leat $1 - 1/n$, during the first $O(n)$ time slots, each of the n processes can access the channel once.

Hint: You can make use of the following fact. Consider a time interval consisting of at least $e^2 k$ time slots. During the time interval, there are at most k processes trying to access the channel and in each time slot, each of the at most k processes tries to access the channel with probability $1/k$. Then, with probability at least $1-e^{-k}$, at the end of the interval, at most $k/2$ of the processes have not succeeded to access the channel.

Exercise 2: Randomized Quicksort [8 Points]

One way to improve the Randomized Quicksort procedure is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. One common approach is the median-of-3 method: choose the pivot as the median (middle element) of a set of 3 elements randomly selected from the subarray. For this problem, let us assume that the elements in the input array $A\,[1..n]$ are distinct and that $n \geq 3$. We denote the sorted output array by $A'\,[1..n]$. Using the median-of-3 method to choose the pivot element x, define $p_i := \mathbb{P}(x = A'[i])$, i.e., p_i is the probability that the element of rank i is chosen as pivot.

1. Give an exact formula for p_i as a function of n and i for $i = 2, 3, ..., n-1$. (Note that $p_1 = p_n = 0$)
2. By what amount have we increased the likelihood of choosing the pivot as $x = A'\left[\left\lfloor \frac{n+1}{2} \right\rfloor\right]$, the median of $A[1..n]$, compared with the ordinary implementation? Assume that $n \to \infty$, and give the limiting ratio of these probabilities.
3. Argue intuitively that in the running time of quicksort, the median-of-3 method affects only the constant factor (compared to usual randomized quicksort variant discussed in the lecture).

Exercise 3: Running Time of the Contraction Algorithm [8 Points]

We have discussed the randomized contraction algorithm for the minimum cut problem in the lecture. When analyzing the algorithm, we have assumed that each contraction on a graph with n nodes can be done in time $O(n)$. Show that this is indeed possible. That is, give an appropriate (simple) data structure to store the current multi-graph G and an algorithm that contracts a uniformly random edge in time linear in the number of nodes of G.

Exercise 4: Randomized Minimum *s*-*t* Cuts? [8 Points]

Consider adapting the min-cut algorithm (Contraction Algorithm) to the problem of finding an s-t *min-cut* in an undirected graph. In this problem, we are given an undirected graph $G = (V, E)$ together with two distinguished nodes $s \in V$ and $t \in V$. An s-t cut is a partition $A \dot{\cup} B = V$ of V such that $s \in A$ and $t \in B$. The size of a cut (A, B) is the number of edges u, v for which $u \in A$ and $v \in B$; we seek an s-t cut of minimum size. As the algorithm proceeds, the vertex s may get merged into a new vertex as the result of an edge being contracted; we call this vertex the s-vertex (initially s itself). Similarly, we have a t-vertex. As we run the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex (this guarantees that in the end, we get an s-t cut).

(a) Show that there are graphs (not multi-graphs) in which the probability that this algorithm finds a minimum s-t cut is exponentially small.

(b) How large can the number of minimum s-t cuts in an instance be?