Dr. Ulf-Peter Schroeder

Paderborn, 12. Juli 2017
Abgabe: Keine!

Übungen zur Vorlesung

Methoden des Algorithmenentwurfs

SS 2017

Blatt 12

Aufgabe 32:
One of the (many) hard problems that arises in genome mapping can be forumlated in the following abstract way. We are given a set of n markers $\{\mu_1, \ldots, \mu_n\}$—these are positions on a chromosome that we are trying to map—and our goal is to output a linear ordering of these markers. The output should be consistent with a set of k constraints, each specified by a triple (μ_i, μ_j, μ_k), requiring that μ_j lie *between* μ_i and μ_k in the total ordering that we produce. (Not that this constraint does not specify which of μ_i or μ_k should come first in the ordering, only that μ_j should come between them).
Now it is not always possible to satsify all constraints simultaneously, so we wish to produce an ordering that satisfies as many as possible. Unfortunately, deciding whether there is an ordering that satisfies at least k' of the k constraints is an NP-complete problem (you don't have to prove this).
Give a constant $\alpha > 0$ (independent of n) and a randomized algorithm with the following property. If it is possible to satisfy k^* of the constraints, then the algorithm produces an ordering of markers for which the *expected* number of satisfied constraints is at least αk^*.

Aufgabe 33:
Let $G = (V, E)$ be an undirected graph with n nodes and m edges. For a subset $X \subseteq V$, we use $G[X]$ to denote the subgraph *induced* on X—that is, the graph whose node set is X and whose edge set consists of all edges of G for which both ends lie in X.
We are given a natural number $k \leq n$ and are insterested in finding a set of k nodes that induces a *dense* subgraph of G; we'll phrase this conretely as follows. Give an algorithm that produces, for a given natural number $k \leq n$, a set $X \subseteq V$ of k nodes with the property that the induced graph $G[X]$ has at least $\frac{mk(k-1)}{n(n-1)}$ edges. The algorithm should be randomized, it should always produce a correct answer, and it should run in *expected* polynomial time.