CS 681 Fall 2001 – Problem Set 2

Due in class Wednesday, October 3

(0) *Don't hand this part in.* Show that the greedy algorithm for finding a minimum-weight basis in a matroid works correctly even when the element weights are not all distinct.

(1) Recall that in the first lecture, we considered the problem of partitioning a set of points into k *clusters* so as to maximize the minimum distance between points in different clusters. Now let's consider a different clustering problem, in which we also must determine the "optimal" number of clusters, rather than being given a value k.

As input, we are given a set of points $P = \{p_1, p_2, \ldots, p_n\}$, together with a list of all $\binom{n}{2}$ distances among distinct pairs of points. Again, we'll assume only that $d(p_i, p_j) = d(p_j, p_i) > 0$ if $i \neq j$, and that $d(p_i, p_i) = 0$ for each i. We are also given a "scaling parameter" $\lambda > 0$.

Now, consider a partition of P into a collection of sets $\mathcal{C} = \{C_1, C_2, \ldots, C_\ell\}$. We define the minimum inter-cluster distance of $\mathcal{C}$ as in lecture: it is the minimum distance between a pair of points belonging to different clusters, and we denote it by $\mu(\mathcal{C})$. We then define the *quality* of $\mathcal{C}$ to be $|\mathcal{C}| + \lambda \cdot \mu(\mathcal{C})$ — in other words, it is the number of clusters in $\mathcal{C}$, plus λ times the minimum inter-cluster distance of $\mathcal{C}$.

The goal is to find a partition of maximum quality. Notice how the first term in the expression for quality rewards partitions with a large number of clusters; but the second term seeks clusters that are far apart, and hence it implicitly rewards partitions with a smaller number of clusters.

Give an algorithm with a running time of $O(n^2)$ to find a partition of maximum quality.

(2) Consider a data structure that we'll call a *delayed Fibonacci heap*. It is implemented just like a Fibonacci heap, but with the following modification: we cut a node x out of one of the constituent trees only after its *third* child has been cut out (rather than its second). The cutting operation may proceed recursively up the tree as before.

Is it the case that a delayed Fibonacci heap still has the same asymptotic cost for operations as a standard Fibonacci heap? You should answer this in one of the following two ways: Either show that a sequence of a `inserts`, b `merges`, c `delete-mins`, and d `decrements` still takes time $O(a+b+c \log n+d)$; or show that there are arbitrarily long sequences of operations that violate this bound.

(3) Given a graph $G = (V, E)$, we can consider the number of simple cycles it contains — here, by a *simple cycle*, we mean a cycle in which each node appears at most once. For example: a tree has no simple cycles; the complete graph on three nodes has exactly one simple cycle; and the complete graph on four nodes, after some calculation, can be seen to have exactly seven simple cycles (it has four of length three, and three of length four).

Suppose we are given a connected graph $G = (V, E)$ with n nodes and m edges.

(a) Consider the collection $\mathcal{F}_1$ of all sets $F \subseteq E$ such that the subgraph (V, F) has at most one simple cycle. Is the collection $(V, \mathcal{F}_1)$ a matroid? Give a proof of your answer.

(b) Consider the collection $\mathcal{F}_2$ of all sets $F \subseteq E$ such that the subgraph (V, F) has at most two simple cycles. Is the collection $(V, \mathcal{F}_2)$ a matroid? Give a proof of your answer.

(c) We define a *brace* of G to be a set of edges $F \subseteq E$ such that the subgraph (V, F) is connected and has exactly one simple cycle. Suppose that each edge in E has a positive weight. Give an algorithm with running time $O(m \log m)$ that finds a brace of G of minimum total edge weight.

(4) Suppose you are given a directed graph $G = (V, E)$ in which each edge has a cost of either 0 or 1. Also, suppose that G has a node r such that there is a path from r to each other node in G. You are also given an integer k. Give a polynomial-time algorithm that either constructs an arborescence rooted at r of cost *exactly* k, or reports (correctly) that no such arborescence exists.