

Problem Set 9

Due: Friday, November 13, 2009.

Collaboration policy: collaboration is *strongly encouraged*. However, remember that

1. You must write up your own solutions, independently.
2. You must record the name of every collaborator.
3. You must actually participate in solving all the problems. This is difficult in very large groups, so you should keep your collaboration groups limited to 3 or 4 people in a given week.
4. **No bibles. This includes solutions posted to problems in previous years.**

Problem 1. Treewidth.

- (a) In class we showed how the decision version of SAT is fixed-parameter tractable with respect to the treewidth. Show that *maximum satisfiability*, the problem of simultaneously satisfying as many clauses as possible, is also FPT with respect to treewidth. Your should need only a short paragraph to explain the change to the decidability algorithm and why it works.
- (b) Consider the (nonparameterized) *vertex cover* problem: given a graph G , find a minimum-size set of vertices that covers all edges (i.e., every edge has at least one endpoint in the vertex cover). Show that vertex cover is fixed-parameter tractable with respect to the treewidth parameter.

Problem 2. Here we will generalize from the Double Coverage algorithm for k -server on the line to k -server on trees. We say that a server s_i is a “neighbor” of a request if there is no other server on the path from s_i to the request.

Algorithm DC-TREE: At each time, all the servers neighboring the request are moving at a constant speed toward the request.

For analysis, we will use the same potential function used for the line: $\Phi = kM_{\min} + \sum_{\text{DC}}$, where M_{\min} is the minimum cost matching between OPT’s and DC’s servers, and $\sum_{\text{DC}} = \sum_{i < j} d(s_i, s_j)$. Note that the set of neighbors may decrease during the service process since a moving server may become “blocked” by other servers.

- (a) Show that **DC-TREE** is k -competitive.
Hint: Break the algorithm into phases, where in each phase the number of neighbors is fixed. Now consider the changes to each of the parts of the potential function within a phase.
- (b) Show that any algorithm for k -server on a tree can be used to solve the paging problem by modeling paging as k -server on a particularly simple tree. Note that k -server algorithm can place servers midway along edges, which doesn't make sense for paging, so you have to "reinterpret" it a bit to get a paging algorithm.
- (c) What standard paging algorithm do you get when you apply the above reduction using DC-TREE?

Problem 3. In many applications, one wants to do range searching among objects other than points. In this problem, we will see that we can reduce several problems of this flavor to normal orthogonal range searching.

- (a) Let S be a set of n axis-parallel rectangles in the plane (i.e., the sides of the rectangles are vertical and horizontal). We want to be able to report all rectangles in S that are completely contained in a query rectangle $[x : x'] \times [y : y']$. Describe a data structure for this problem that uses $O(n \log^3 n)$ storage and has $O(\log^4 n + k)$ query time, where k is the number of reported answers. (**Hint:** Transform the problem to an orthogonal range searching problem in some higher-dimensional space.)
- (b) Let P consist of a set of n polygons in the plane. Again, describe a data structure that uses $O(n \log^3 n)$ storage and has $O(\log^4 n + k)$ query time to report all polygons completely contained in the query rectangle, where k is the number of reported answers (note that as a special case, you can report containment of line segments).

NONCOLLABORATIVE Problem 4. Suppose you're implementing a video game in which the player can walk around a planar environment made up of walls, and at any time the screen displays only the walls that are (partially) visible by the player. More precisely, the player is modeled as a single point; the walls are modeled as noncrossing line segments; two points are *visible* if the line segment connecting them does not intersect any walls except at its endpoints; and a wall is *visible* from a point if at least one point on the wall is visible from the point. Give an $O(n \lg n)$ -time algorithm to compute the set of walls visible from the player. **Hint:** Use a line-sweep algorithm, but instead of sweeping a horizontal line, sweep a half-line around a point.

Problem 5. Given a set of n points in the plane, we wish to find the closest pair of points. We show how to do so in $O(n \log n)$ time by sweeping a vertical line past the points. At

any point in time, we will know the distance d between the closest pair of points behind the sweep line. We maintain a “strip of infinite height and of width d behind the sweep line, and the set of points inside the strip.

- (a) Argue that when the sweep line encounters a new point p , if p is one point in the closest pair behind the sweep line, then the other point in the closest pair is inside the strip—and in fact, in a particular portion of the strip quite close to the new point.
- (b) Argue that in fact, this portion of the strip can contain only a constant number of points.
- (c) Develop a data structure to associate with the sweep line so that these candidates for closest pair can be identified quickly ($O(\log n)$ time per event), and show how to maintain this data structure as the line sweeps ($O(\log n)$ time per event). Conclude an $O(n \log n)$ time bound for closest pair.
- (d) Does the first algorithm above generalize to any higher dimension k ? What is the time bound as a function of k ?
- (e) Alternatively, suppose you construct the Voronoi diagram on the points. Show how the closest pair can be identified in $O(n)$ time. This gives an alternative $O(n \log n)$ time algorithm in 2 dimensions.