CS 182
Advanced Topics in Algorithms
Spring 2004
Problem Set 5b
Due Monday, February 23 in class

These problems are best done in sequence. Some problems rely on their predecessors!

1. **[30 Points] The DC Tree Algorithm!** In class we examined the online DC algorithm for the real line with the Euclidean distance metric. Recall that in this algorithm if the request is outside the convex hull of the servers, we shuttle the nearest server to serve the request. If the request is inside the convex hull, we shuttle the two servers adjacent to the request towards the request at equal speed (at least one of the two servers then arrives to serve the request).

   Now, let's extend the algorithm to trees. Consider a tree made of line segments. The distance between two points in this infinite metric space is the total Euclidean length of the (unique) path between those two points in the tree. We say that a server s is a *neighbor* of a request r if there is no other server on the (unique) path from s to r. When two servers located in the same spot are neighbors of the request, we consider one of them (arbitrarily) to be a neighbor and the other one not to be a neighbor.

   The algorithm **DC Tree** does the following: For each request, all servers which are neighbors of the request move toward the request at the same constant speed.

   Notice that when the tree degenerates to be a line, algorithm DC Tree becomes the same as algorithm DC. Notice one weird thing: As a group of servers are moving towards a request, some of them may get "passed" by other servers coming in from different limbs of the tree. In this way, a moving server can suddenly stop being a neighbor of the request! When this happens, this server stops moving.

   Show that DC Tree is k-competitive by generalizing the proof that we showed in class. Use the same potential function. There will be a bit more algebra, but it's not bad. Any yucky terms in your algebra should cancel out nicely.

2. **[5 Points] Laziness is Good.** This problem is a freebie! We talked about this in class and this is just your chance to explain it in your own words.

   A k-server algorithm is said to be *lazy* if it has the following properties: If there is already a server located at the request point, no servers are moved. If there is no server located at the request point, the algorithm moves one robot to that request point. No other movements are ever made. Note that DC and DC Tree are not lazy algorithms! Explain briefly how any non-lazy algorithm can be converted into a lazy algorithm which incurs no additional cost.

3. **[15 Points] A Neat Application of the DC Tree Algorithm.** Now consider a finite metric space which is a weighted graph with N vertices. That is, the requests and the servers can only be located at the N vertices. The edge weights, of course, satisfy the triangle inequality. In this problem you will show that there is a $(N-1)k$-competitive online algorithm for this k-server problem.

(a) First we will need a little technical lemma. Consider a weighted graph G and a minimum spanning tree T for G. Let (u, v) be an edge of weight d in G. Note that edge (u, v) may or may not be in T. However, prove that there must be a path from u to v in T whose total weight does not exceed $(N-1)d$.

(b) Now describe a $(N-1)k$-competitive online algorithm for this problem. Remember, the metric space is finite! The servers and requests must be at the vertices of the graph.

4. **[10 Points] Meshes.** Consider k servers on a $\sqrt{N} \times \sqrt{N}$ grid in which adjacent grid points are at distance 1 from one another. Again, this is a finite metric space. The servers and requests therefore may only move from vertex to vertex. Show how to find a $(\sqrt{N}+1)k$-competitive algorithm for such a mesh.

5. **[10 Points] DC Tree and Paging!** Consider a paging problem with a slow memory of size N and a fast memory of size k. We can embed this paging problem into a tree as follows: The tree is a star with a central vertex and N additional vertices, $v_1, \ldots, v_N$ each of which is connected by an edge with weight $\frac{1}{2}$ to the central vertex. The vertices $v_1, \ldots, v_N$ correspond to the N pages of slow memory and are called *page vertices*. If one of the k servers is located on a page vertex, this corresponds to that page being in the fast memory. We'll assume that the k-servers are initially distributed among the N page vertices representing an initial selection of k pages in fast memory. Notice that moving a server from one vertex in $v_1, \ldots v_N$ to another costs 1, which is the cost of a page fault in the paging problem.

   The DC Tree algorithm (the pure non-lazy version) on this graph corresponds to a paging algorithm which we know and love (and saw in class). Which one? Explain.