CMPUT204
Department of Computing Science
University of Alberta

Homework Assignment #5
Due: Dec 8, 2004

Question 1. (15 marks)

Consider an edge weighted graph G with weights $w[\cdot]$, and a minimum spanning tree of the graph, T. Suppose for some edge (u, v) you decrease $w[u, v]$ to be w_*. Describe a procedure (pseudocode is unnecessary) that would find the new MST with the updated weight. Your algorithm must run faster than recomputing the MST from scratch, i.e., $o(|E|log|V|)$. Give the running time of your procedure.

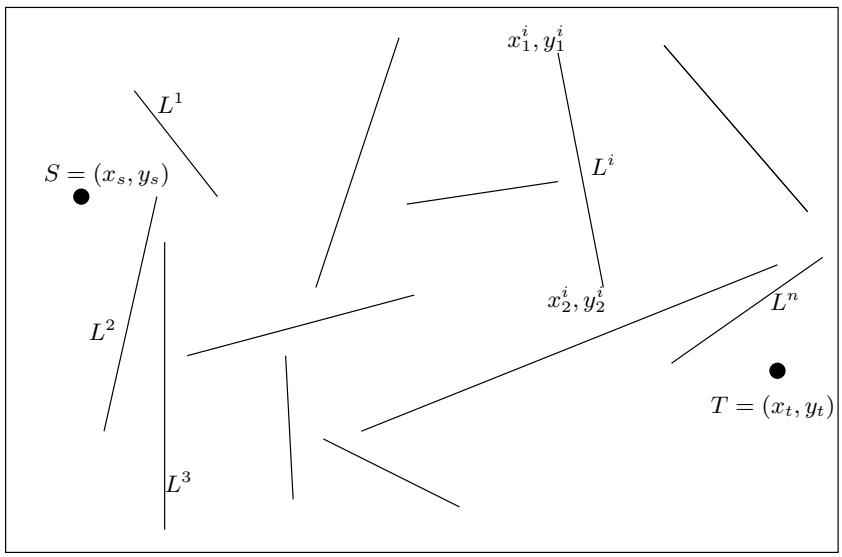
Solution: Suppose (u, v) is in T, then decreasing its weight will leave the minimum spanning tree unchanged. If (u, v) is not in T, then decreasing its weight may cause it to be in the new minimum spanning tree. Adding (u, v) into T will cause a single cycle. The edges on this cycle need to be examined and the edge with the largest weight (may be w_*) can be removed to create the new MST. Finding the cycle can be done with DFS from u and finding a back-edge that returns to u. The largest weight edge on this cycle can then be found and removed to create the new minimum spanning tree T^*. This takes at most $O(|E|)$ for the search (in a connected graph) and finding the edge in the cycle to be removed.

This was not required, but proof of correctness is fairly straightforward. The easiest way to see this is to consider Kruskal's algorithm, known to be correct. With the weight of (u, v) decreased it might get considered earlier and so might get added without creating a cycle. If it still creates a cycle then the MST remains unchanged (and the algorithm correctly returns the unchanged T). If it is added, then there will be some other edge (x, y) that is in T, but won't be added since it creates a cycle. The cycle it creates is exactly the cycle the above algorithm finds and examines, and since (x, y) finishes the cycle it must have the largest weight of any edge in the cycle. Kruskal's won't add the edge in the correct updated MST, and the above algorithm also will remove the edge, and so they result in identical trees.

Question 2. (20 marks) A robot is in a room with many walls (obstacles) and wants to go from the start point $S = (x_s, y_s)$ to target point $T = (x_t, y_t)$. There are a total of n walls $L^1, \ldots, L^n$ and each wall L^i is a straight line segment which runs between points x_1^i, y_1^i and x_2^i, y_2^i (see the example in the figure). Assume that the robot is provided with the cordinates of the walls as well as the start point and the target point. Describe an efficient algorithm for the robot which helps it to go from the start point to the end point in the shortest amount of time (assuming that the robot moves at a fixed speed). You may also make use of a geometry procedure *Intersect* that can answer in constant time if two line segments (identified by their endpoints) intersect. Analyse the running time of your algorithm.

Solution: First, we note that the shortest path from S to T consist of stragiht line segments that run between end-points of some of the walls. One easy explanation for this is: consider a rope running on the shortest path from S to T. Take the end-points of this rope and stretch. Since it cannot be shortened (because it is on a shortest path) it must be touching the end-points of a wall whenever there is a bend somewhere on the rope. To prove it mathematically, consider a bend-point y along the shortest path and assume (by way of contradiction) that it is not touching the end-point of one of the walls. Then you can "cut" that bend and make a shorter path.

So we can model the problem as a shortest path problem on a graph G. The vertices of G are just all the end-ponits of the walls plus one vertex v_S for S and one vertex v_T for T. Two vertices u and v are adjacent in G if the line segment connecting the corresponding points does not intersect with any of the walls (so you can go directly from the point corresponding to u to the point corresponding to v). We can compute this graph using the geometry procedure *Intersect*. The weight of an edge is just the Eucleadian distance of the corresponding points on the plane. Now we use Dijkstra's algorithm to compute a shorest path from v_S to v_T. The sequence of line segments connecting to the edges of this path form the shortest path from S to T.



Question 3. (15 marks) For each of the following problems, first prove that it belongs to NP. Then state whether you think the problem is NP-complete. Justify your answer by either proving it to be NP-compelete or giving good reason to suspect that it is not. You can assume the facts that the following problems are NP-complete: SAT, 3SAT, CLIQUE, IND-SET, HAM-PATH, HAM-CYCLE. Remember that it is widely believed that P≠NP.

a. MIN-LEAF-SPANNING-TREE
Input: A graph G and an integer k.
Question: Does G have a spanning tree with at most k leaves?

Solution: Given a set T of edges as a solution to this problem, first have to check that T is a spanning tree of G (is acyclic connected graph and contains every vertex). This clearly can be done by running a BFS or DFS on T. Then we count the number of vertices with degree 1 in T. All of these can be done in polytime. So this problem belongs to NP.

Consider this question with $k = 2$. It is essentially asking if you have a spanning tree with at most 2 leaves, i.e. a path that contains all the vertices (a Hamiltoninan path). So if we can solve this problem (in particular for $k = 2$) then we can solve the Ham-Path problem which we know is NP-complete. Thus this problem is NP-complete too.

b. 8-CLIQUE
Input: A graph G.
Question: Does G have a clique of size at least 8?

Solution: We show that the problem is in P (and therefore is in NP as well) and probably not NP-complete (because of the assumption that P≠NP). We consider every set of 8 vertices in G. There are at most $O(|V|^8)$ such sets (to be more precise there are $\binom{|V|}{8}$ of them). For each such set we can check in $O(1)$ if they form a clique. Thus we can solve this problem in polytime ($O(|V|^8)$).

c. MANY-3SAT
Input: A boolean formula Φ in 3-CNF and an integer k.
Question: Is there a truth assignment to Φ such that at least k clauses are satisfied?

Solution: Given a truth assignment we can easily check which cluases are satisfied and which are not. Then count the number of ones that are satisfied and compare with k. So this problem is in NP. Now we prove that it is NP-complete by reducing 3SAT to this problem. Given an instance I of 3SAT with m clauses, pass it on to MANY-3SAT with $k = m$. If the answer is yes, then I is satisfiable and if the answer is no then I is not satisfiable. So having a solver for MANY-3SAT we can use that to solve 3SAT (which is NP-complete).