Assignment 4—Due Friday, December 8

Reminder: no late homeworks accepted! No exceptions!

"The Traveling Salesman Problem"

BACKGROUND. The *traveling salesman problem* (TSP) is the most famous problem of combinatorial optimization. It has been studied for more than 150 years, and still we still we don't know how to solve this problem on many instances of quite modest size. In this assignment you'll solve a small instance of the TSP.

In the TSP you are given a graph G, each edge having some (real-valued) *cost*. You are asked to find a cheapest cycle which passes through each vertex exactly once. Typically one thinks of the vertices as cities, the costs as the distances between them, and the solution being the path which a traveling salesman should take in order to visit each city once, ending where he starts, and minimizing his commute. A bit more formally, given an undirected[1] graph $G = (V, E)$ and a cost function $c : E \to \mathsf{R}$, where $V = (1, 2, \ldots, n)$, you must find a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ for which

$$c^* = c_{\pi(1)\pi(2)} + c_{\pi(2)\pi(3)} + c_{\pi(3)\pi(4)} + \ldots + c_{\pi(n-1)\pi(n)} + c_{\pi(n)\pi(1)}$$

is minimized. In the version of TSP which you'll address the costs will be the distances of points in the plane. This is sometimes called *Euclidean* TSP, or (slightly more generally) TSP with the *triangle inequality.*

WHAT YOU MUST DO. Your program will read in the location of some set of cities. The format of this data will be: the number of cities, n, followed by n triples of numbers, each triple consisting of the name of the city (a string), its x-coordinate (a real number), and its y-coordinate (another real number). Whitespace is ignored. Given this data, you must find a minimal-cost tour which visits each and every city. The cost of going from one city to another is simply the distance between them. When you find your tour, report what it is and what it costs.

You must find a *cheapest* tour. The algorithm of your program must effectively be providing a proof that there is none shorter.

To solve this problem use some version of (least cost) *branch-and-bound.* We will briefly discuss branch-and-bound in lecture and in discussion section, but the idea is quite simple: in the *state space tree* of TSP solutions (i) explore the tree in a good order— expand more promising nodes before you expand less promising ones; (ii) prune the tree whenever you can— when you can determine that all solutions rooted at a node have a cost which is higher than the minimum cost, don't bother to expand that node. Of course (i) and (ii) are only heuristics for making your algorithm run faster, but what you finally find using these heuristics is guaranteed to be optimal.

[1] It makes perfect sense for G to be a directed graph, too. But your solution will be for the undirected case.

There are several ways to realize a branch-and-bound approach to TSP. Attached to this assignment I reprint from an old algorithms book[2] a section on branch-and-bound solutions to the TSP. My guess is that you will have the best luck using the first technique described in this reading.

A potentially useful fact I don't recall being mentioned in the Horowitz and Sahni reading is that a minimum spanning tree T^* always has cost $t^* \leq c^*$ (why?).

COLLABORATION. For this homework, you may work in pairs. You might find that fun. Only pairs— no bigger groups. If you do work with a partner, the two of you turn in **one** writeup and get **one** grade. I advise you to split things up so that both partners do design and coding. Whether you work alone or with a partner isn't a factor in your grade on the assignment.

WHAT TO TURN IN. Your writeup should carefully explain how you solved this problem and why you made those choices. Spend time on your English-language writeup; it is an important component of what you turn in. Your program itself should be as efficient as you can reasonably make it. Achieve this efficiency with good algorithms and data structures and not by writing poorly-abstracted code which you think (probably wrongly) will run faster.

Run your program on the data provided to you in `~rogaway/public_html/classes/110/tsp`. This dataset has 15 cities. Your program should work for any number of cities, but it is expected that your solution will too long when the number of cities gets large. It is entirely possible that I miss-guessed and 15 cities is already too large. If so, no problem: just prune my test case to the first n' cities, where n' is the largest number of cities your program can handle in a reasonable amount of time. In a class like this "reasonable" means at most a couple minutes on the CSIF machines; please don't hog our shared-environment CPU resources.

Do your final run on the DECs, using Ted's `AutoTimer` class to demonstrate how fast your program runs.

Investigate the running time of your program experimentally. Estimate how long it would take your program to solve a 25-city TSP instance. A 50-city TSP instance.

The program is worth the usual 8% of your grade, but occasionally it's a bit more significant: because this is the only one of your programs which I have in hand when assigning grades, I sometimes glance at this when I can't decide a grade based on the numeric scores.

[2] E. Horowitz and S. Sahni, *Fundamental of Computer Algorithms,* Computer Science Press, 1978.