CS:3330 Spring 2017: Homework 8

Due at the start of class on Tue, April 18

Traveling Salesman Problem (TSP)

Given a set P of points (e.g., a bunch of cities) and pairwise "distances" between these points, the *Traveling Salesman Problem (TSP)* seeks to find a shortest tour that starts from a point and visits every point exactly once and returns to the starting point. TSP is computationally very difficult; the fastest algorithms for this problem run in exponential time and there are reasons to believe that faster algorithms do not exist for TSP. A special case of TSP is the *metric TSP* problem in which distances satisfy the triangle inequality, i.e., for any three points p_1, p_2, p_3, $distance(p_1, p_3) \leq distance(p_1, p_2) + distance(p_2, p_3)$. Even metric TSP is computationally difficult, but there are good approximation algorithms for metric TSP. In this homework you are required to implement a simple 2-approximation algorithm for metric TSP that is based on computing an MST.

The data set

We have posted a data file called `miles.dat`, which contains basic geographic data on 128 cities in North America. This data set was obtained from Donald Knuth's Stanford Graphbase. The file `miles.dat` contains, for each city, the following pieces of information: (i) name of the city, (ii) state of the city, (iii) latitude and longitude of the city, (iv) population of the city, and (v) distances to all other cities This is old data - from a 1949 Rand McNally mileage guide, so the population numbers are definitely out of date. The highway mileage may also be out of date, but the data serves our purposes quite nicely. Note that the data file specifies for each city, the distances to all cities before it in the file; this is sufficient to get pairwise distance between any pair of cities. To understand how the distances are specified take a look at the following first few lines of the data file. This data is telling us that Worcester, MA is 2964 miles from Yakima, WA, 1520 miles from Yankton, SD, and 604 miles from Youngstown, OH.

```
Youngstown, OH[4110,8065]115436
Yankton, SD[4288,9739]12011
966
Yakima, WA[4660,12051]49826
1513 2410
Worcester, MA[4227,7180]161799
2964 1520 604
```

Also note that latitudes and longitudes are specified without decimal points or N, S, E, W indicators: for example Youngstown, OH is at longitude 41.09972 N and latitude 80.64972 W and this is specified (approximately) as "[4110,8065]."

Your first coding task is to read from this data file and store the data is appropriate data structures.

MST algorithms

Your next coding task is to implement two of MST algorithms we studied in class: (i) Boruvka's Algorithm and (ii) Kruskal's Algorithm. Your implementations should be faithful implementations

of the algorithms discussed in class and should have asymptotic running time of $O(m \log n)$ on graphs with n vertices and m edges. For Boruvka's algorithm you need to use an $O(m+n)$-time graph traversal algorithm (BFS or DFS) in order to systematically examine all edges to determine safe edges. For Kruskal's algorithm you can use any one of the two Disjoint Set Union Find data structure implementations we studied in class: (a) reverse trees with union by depth or (b) shallow threaded trees with union by size.

MST to a Traveling Salesman tour

It is easy to see how to transform an MST to a Traveling Salesman tour whose total length is at most twice the length of an optimal tour. This would be a 2-approximation algorithm. Here is a fairly clear explanation of both the algorithm and the proof of why it yields a 2-approximation: `http://www.cs.tufts.edu/~cowen/advanced/2002/adv-lect3.pdf`.

Local search refers to the general algorithmic approach of examining solutions "near" a given solution to see if the given solution can be improved. The *2-OPT* algorithm is a local search algorithm that takes a given Traveling salesman tour and for every pair of edges $\{a, b\}$ and $\{c, d\}$ in the tour, checks to see if replacing these edges by $\{a, d\}$ and $\{c, b\}$ leads to an improved solution. The algorithm returns the cheapest solution found in this manner. Similarly, the *3-OPT* algorithm swaps out edge triples to see if improved solutions can be found.

Your next three coding tasks are to implement (i) the algorithm that takes as input an MST and returns a 2-approximate Traveling salesman tour, (ii) the 2-OPT algorithm, and (iii) the 3-OPT algorithm.

Experiments

1. Report the following statistics on the execution of Boruvka's algorithm on the input graph: (i) how many iterations does the algorithm take and (ii) what is the size of the smallest connected component at the end of each iteration. Recall that Boruvka's algorithm comes with the guarantee that in each iteration, the size of the smallest component, at least doubles. In practice, the size of connected components can grow much more quickly than this guarantee suggests. The point of this experiment is to understand this issue.

2. As we saw in class, the asymptotic performance of Kruskal's algorithm relies on the performance of the implementation of the Disjoint Set Union Find data structure.

   If you implemented this data structure as a reverse tree with Union by depth, then the depth of the deepest tree associated with a component is a good measure of the performance of this data structure. So in this case, report the maximum depth of a reverse tree after each iteration. Note that this quantity is 1 after the first iteration of Kruskal's algorithm and it will typically remain unchanged for a number of iterations and then increase by 1.

   If you implemented the Disjoint Set Union Find data structure as a shallow threaded tree with union by size, then the maximum number of times the "leader" pointer of a node is changed is a good measure of the performance of the data structure. So in this case, report the maximum number of times the "leader" pointer of a node is changed after each iteration. Note that this quantity is 1 after the first iteration of Kruskal's algorithm and it will typically remain unchanged for a number of iterations and then increase by 1.

It seems best to report these statistics in a plot with the x-axis being the iteration index and the y-axis being the measure of the data structure performance being reported.

3. Output the 127 edges in the computed MST and the weight of the computed MST.
4. After transforming the computed MST into a Traveling Salesman Tour, output the tour and its weight. So you're being asked to output the Traveling Salesman tour prior to using the 2-OPT and 3-OPT local search heuristics to improve the tour.
5. Output the Traveling Salesman Tour and its weight after using the 2-OPT local search heuristic.
6. Output the Traveling Salesman Tour and its weight after using the 2-OPT and 3-OPT local search heuristics.

What to Turn in

You are expected to turn in two items for this homework.

1. A single file named `hw8.ext` containing all your code. The extension "`.ext`" will of course depend on the programming language you use. Your code should be extensively documented. For example, functions should be preceded by comment blocks that tell the reader what the purpose of the function is, variable names should be suggestive of their purpose, etc. The file should also start with a listing of any bugs or issues with your code, sources you used, etc. You are not allowed to share code with classmates, but it is okay to use code available on the web, with appropriate attribution. For example, you might find it convenient to use a class that implements the *max-heap* data structure. This is fine, provide you carefully acknowledge your source. We will set up a dropbox on ICON for you to upload your code file. This needs to be done before class time on Tue, April 18th .
2. A single pdf file named `hw8.pdf` containing the results of all your experiments. You should upload this file onto the ICON dropbox *and* bring a printout to class on Tue, April 18th. This document should be well-formatted, should contain clear and concise text, and clearly and completely labeled plots.

Extra Credit Opportunities

1. Find an optimal Traveling Salesman tour for the given input. Given the current state of human knowledge, any algorithm that does this is bound to be some variant of brute force, taking exponential amount of time. But, there are some pretty sophisticated brute force implementations for the Traveling Salesman tour computation out there and you are welcome to download and use whatever you find. To get this extra credit, report the Traveling Salesman tour that has been computed, along with its weight. Then write a paragraph on how you found this tour, e.g., what software you downloaded, how you used it, and how long it took to find the tour.
2. Show on Google maps the MST and Traveling Salesman tours you computed for the experiments listed above. If you do this, then attach pictures of the MST and Traveling Salesman tours to your document `hw8.pdf`. You are welcome to go crazy and make a webpage, allow users to interact with these structures you're constructing, etc.