CEN235 – DATA STRUCTURES
Fall – 2018

Lab 9: Symmetric Traveling Salesman Problem (TSP)

In this LAB, you are going to implement a program that solves symmetric traveling salesman problem.

Preliminary Work:

"The **traveling salesman problem** (TSP) is a prominent illustration of a class of problems in computational complexity theory. The problem can be stated as: Given a number of cities and the costs of traveling from one to the other, what is the cheapest roundtrip route that visits each city and then returns to the starting city?

The most direct answer would be to try all the combinations and see which one is cheapest, but given that the number of combinations is N!, this solution rapidly becomes impractical.

The problem is of considerable practical importance, and various approximation algorithms, which "quickly" yield "good" solutions with "high" probability, have been devised. However, the only way to be completely confident of finding the optimal route is to check every route, which is completely impractical for TSPs with more than 20 cities."

For this LAB, you are going to implement an optimum solver and one heuristic solver. Before starting the LAB, you can run this sample code for optimum solver with five cities. You can use this code or some parts of it directly in your program.

Note the following implementation details:

- Every tour must begin from Istanbul and end in Istanbul (Which is the first city in both files)
- Cities will be taken from gsp_turkiye.txt and gsp_dunya.txt. These text files include city name and country, followed by longitudinal *theta*-coordinate θ, and latitudinal *phi*-coordinate ϕ). In order to find distance between two cities, you must calculate:

$$D = R * \arccos(\cos(\phi_1)\cos(\phi_2)[\cos(\theta_1)\cos(\theta_2) + \sin(\theta_1)\sin(\theta_2)] + \sin(\phi_1)\sin(\phi_2))$$

where θ_1, θ_2 are the longitudes and ϕ_1, ϕ_2 are the latitudes of the two cities, and R is the Earth's radius, which is around 6400 kilometers.

Task-1: Optimum Solver [50pts]:

Write a command line program that takes as two arguments, a number *n* and a filename containing a list of cities with world coordinates and attempts to solve *TSP* on the first *n* cities in the file (e.g. "TSPOptimum 10 gsp_turkiye.txt"). When a solution is found, the shortest complete tour should be written to an output file (e.g. opt_gsp_turkiye_10.txt), together with the lengths of all the edges, and the total length of the tour.

There are several approaches to the complete solution program. The naive approach is to simply list all the permutations of *n* cities keeping the first city fixed and then count the total distance of a tour starting and ending in the first city. Then output the permutation whose total distance is minimal. More sophisticated approaches involve *branch and bound* and *cutting-plane* techniques.

You will use gsp_turkiye.txt for testing optimum solver. There are 55 locations listed. Start with n=6 and increase n by 2 each time (n=6, n=7, n=8, …). Your solver possibly will not be able to solve for large n (why?). Report the results and time passed executing the code in terms of seconds.

Task-2: Greedy Heuristic (Nearest Neighbor Algorithm) [50pts]:

Write a command line program that takes as two arguments, a number *n* and a filename containing a list of cities with world coordinates and attempts to find an approximate solution to *TSP* on the first *n* cities in the file using nearest neighbor algorithm (e.g. "TSPNNA 10 gsp_turkiye.txt"). When a solution is found, it will be written to an output file (e.g. nna_gsp_turkiye_10.txt), together with the lengths of all the edges, and the total length of the tour.

You will use gsp_turkiye.txt and gsp_dunya.txt for testing greedy solver. First solve for the n's for gsp_turkiye that you used in the optimum solver. Then solve for n=100, 200, 400, 800 and 1600 in gsp_dunya. Report the results and time passed executing the code in terms of seconds.

From Wikipedia, the free encyclopedia:

"The **nearest neighbor algorithm** was one of the first algorithms used to determine a solution to the traveling salesman problem. It is also a lot faster than testing every route and some other algorithms.

These are the steps of the algorithm:

1. Start with the initial city where the tour begins and add it to the initially empty list A.
2. Find the closest city (which should not be inside the list A) to the city added lastly to the list A.
3. Add this closest city to list A.
4. Repeat step 2 and 3 until all cities are added to the list A.

The nearest neighbor algorithm is easy to implement and can be done quickly, but it can sometimes miss shorter routes which are easily noticed with human hindsight. The results of the nearest neighbor algorithm should be checked before use, just in case such a shorter route has been missed.

In the worst case, the algorithm can compute tours that are by an arbitrary factor larger than the optimal tour. To be precise, for every constant r there is an instance of the traveling salesman problem such that the length of the tour computed by the nearest neighbor algorithm is greater than or equal to r times the length of the optimal tour."