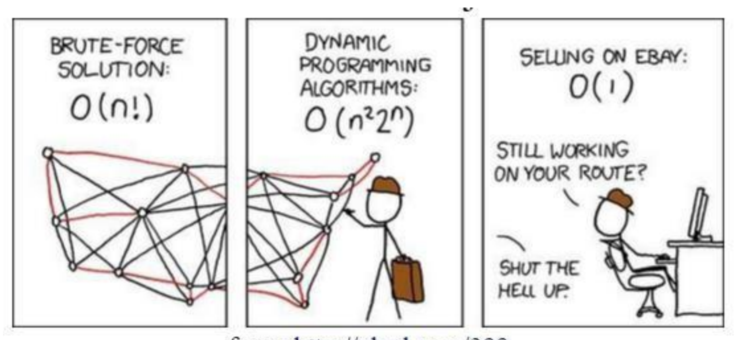
CS 325 HW 6: The Travelling Salesman Problem (TSP)



from: http://xkcd.com/399

In this assignment you will have fun trying out ideas to solve a very hard problem: The Traveling Salesman Problem (TSP).

You are given a set of n cities and for each pair of cities c_1 and c_2, the distances between them $d(c_1, c_2)$. Your goal is to find an ordering (called a tour) of the cities so that the distance you travel is minimized. The distance you travel is the sum of the distances from the first city in the ordering to the second city, plus the distance second city to the third city, and so on until you reach the last city, and then adding the distance from the last city to the first city. For example if the cities are Seattle, Portland, Corvallis and Boise. The total distance traveled visiting the cities in this order is:

d(tour) = d(Seattle,Portland) + d(Portland, Corvallis) + d(Corvallis,Boise) + d(Boise, Seattle)

In this project, you will only need to consider the special case where the cities are locations in a 2D grid (given by their x and y coordinates) and the distance between two cities $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$ is given by their Euclidean distance. To avoid floating point precision problems in computing the squareroot, we will always round the distance to the nearest integer. In other words you will compute the distance between cities c_1 and c_2 as:

$$(c_1, c_2) = \text{nearestint}\left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\right)$$

For example, if the three cities are given by the coordinates $c_1 = (0, 0)$, $c_2 = (1, 3)$, $c_3 = (6, 0)$, then a tour that visits the cities in order $c_1 \to c_2 \to c_3 \to c_1$ has the distance

$$d(tour) = d(c_1, c_2) + d(c_2, c_3) + d(c_3, c_1)$$

where

$$\begin{aligned}(c_1, c_2) &= \text{nearestint}\left(\sqrt{(0-1)^2 + (0-3)^2}\right)\\ &= \text{nearestint}\left(\sqrt{(-1)^2 + (-3)^2}\right)\\ &= \text{nearestint}\left(\sqrt{1+9}\right)\\ &= \text{nearestint}\left(\sqrt{10}\right)\\ &= \text{nearestint}(3.1622\ldots)\\ &= 3\end{aligned}$$

$$
\begin{aligned}
(c_2, c_3) &= \text{nearestint}\left(\sqrt{(1-6)^2 + (3-0)^2}\right) \\
&= \text{nearestint}\left(\sqrt{(-5)^2 + (3)^2}\right) \\
&= \text{nearestint}\left(\sqrt{25+9}\right) \\
&= \text{nearestint}\left(\sqrt{34}\right) \\
&= \text{nearestint}(5.8309 \ldots) \\
&= 6
\end{aligned}
$$

$$
\begin{aligned}
(c_3, c_1) &= \text{nearestint}\left(\sqrt{(6-0)^2 + (0-0)^2}\right) \\
&= \text{nearestint}\left(\sqrt{(6)^2 + (0)^2}\right) \\
&= \text{nearestint}\left(\sqrt{36+0}\right) \\
&= \text{nearestint}\left(\sqrt{36}\right) \\
&= \text{nearestint}(6) \\
&= 6
\end{aligned}
$$

So that d(tour) = 3 + 6 + 6 = 15.

HW 6 - Specification

You will research and implement an algorithm for finding an approximate solution for the TSP problem. You should provide pseudocode for the algorithm that is implemented and give the running time. There is much literature on methods to "solve" TSP please cite any sources you use. TSP is not a problem for which you will be able to easily find optimal solutions. Your goal is to find an approximate solution or the "best solution" you can find in a reasonable amount of time. One possible algorithm would perform a depth-first traversal on a MST for a Euclidean graph. See JE Algorithms http://jeffe.cs.illinois.edu/teaching/algorithms/notes/J-approx.pdf

Your program must:

- Be named written in C, C++ or Python and named, tsp.c, tsp.cpp or tsp.py
- Accept problem files on the command line.
- Name the output file as the input file's name with .tour appended (for example input **tsp_example_1.txt** will output **tsp_example_1.txt.tour**).
- Compile/Execute correctly and without debugging on the flip server according to specifications and any documentation you provide.
- You can use any programming language you want to that runs on the flip server.

Input specifications: A problem instance will always be given to you as a text file.

- The first line of the file is the number of cities n
- The following lines each define a city and have three numbers separated by white space.
    - The first number is the city identifier
    - The second number is the city's x-coordinate

  - The third number is the city's y-coordinate.

Output specifications:

- You must output your solution into another text file with n+1 lines, where n is the number of cities.
- The first line is the length of the tour your program computes.
- The next n lines should contain the city identifiers in the order they are visited by your tour.
- Each city must be listed exactly once in this list.
- This is the certificate for your solution and your solutions will be checked. If they are not valid you will not receive credit for them.

Example instances:

I have provided you with six example files. They are available on Canvas and are provided according to the input specifications.

• **tsp_example_[*].txt**	Input files
• **tsp_example_[*].txt.tour**	Example output is provided for tsp_example_0.

The optimal tour lengths are:

- tsp_example_0 = 14,
- tsp_example_1 = 108159,
- tsp_example_2 = 2579
- tsp_example_3 = 5333,
- tsp_example_4 = 7411,
- tsp_example_5 = 23001.

You should use these values to evaluate your algorithm. For full credit it is required that you have a ratio of 1.50. That is
(your tour length)/(optimal tour length) <= 1.50, and runs in less than 30 seconds.

Note: The 1.50 bound is only required for the specific test cases that I give you. I am not requiring proof of a 1.50-approximation algorithm. Some 2-approximation algorithms will satisfy the 1.50 bound on these specific instances.

Verifying:

You should verify that your program outputs tours that contain all the cities and has the total distance computed correctly. The verifying program **tsp-verifier.py** can be used as follows to verify that your solutions:

```
python tsp-verifier.py   inputfilename  solutionfilename
```

Project Report (Canvas):

You will submit a project report containing the following:

- A description and pseudocode of your algorithm for solving the Traveling Salesman Problem.
- Theoretical running time and approximation bound.
- Summarize any other research you did.
- List your "best" tours for the tsp_example instances, the approximation ratios and the time it took to obtain these tours.

Program files (TEACH):

- In a .zip file include all program files, tour files and solution files, verifier and script
  - tsp.c, tsp.cpp or tsp.py
  - all tsp_example_*.txt file
  - all tsp_examble_*.txt.tour files
  - tsp-verifier.py and TSPAllVisitied.py
  - HW6.sh there is one script for all languages.