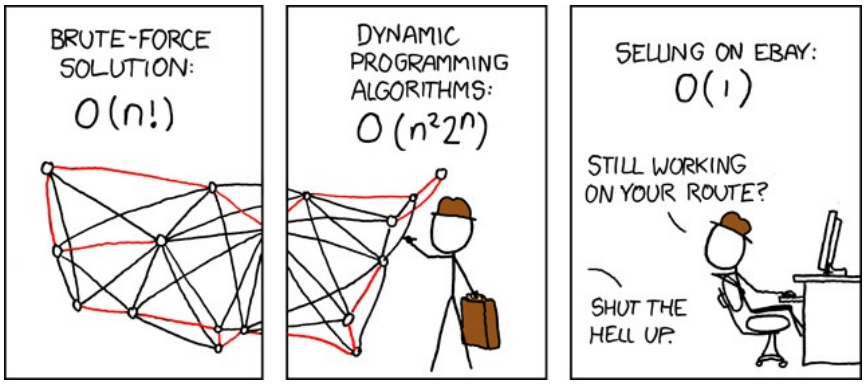
CS325 TSP Project



from: http://xkcd.com/399

This project is rather open-ended, and I hope you will have fun trying out ideas to solve a very hard problem called the traveling salesperson problem or TSP.

You are given a set of n cities and, for each pair of cities c1 and c2, the distance between them d(c1, c2). Your goal is to find an ordering (called a tour) of the cities so that the distance you travel is minimized. The distance your travel is the sum of the distance from the first city in your ordering to the second plus the distance from the second city in your ordering to the third and so on until you reach the last city and finally add the distance from the last city to the first city. For example, say the cities are Chicago, New York, New Orleans and San Francisco. The total distance traveled visiting the cities in this order is:

d(Chicago, New York) + d(New York, New Orleans) + d(New Orleans, San Francisco) + d(San Francisco, Chicago)

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 distances between two cities c1 = (x1, y1) and c2 = (x2, y2) is given by their Euclidean distance. So that we need not worry about floating point precision problems in computing the square-root (and so that everyone will get the same answer), we will always round this distance to the nearest integer. In other words, you will compute the distance between city c1 and c2 as:
$\mathrm{d(c1,\ c2)} = \mathrm{nearestInteger}\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

For example, if three cities are given by the coordinates (0, 0), (1, 3), (3, 1), then a tour that visits these cities in this order has distance:
$nint(\sqrt{10}) + nint(\sqrt{8}) + nint(\sqrt{10}) = nint(\sqrt{3.16...}) + nint(\sqrt{2.82...}) + nint(\sqrt{3.16...}) = 9$

Project specification

Your group is to design and implement a method for finding the best tour you can. TSP is not a problem for which you will be able to easily find optimal solutions. It is that difficult. But your goal is to find the best solution you can in a certain time frame. You may want to start with some local search heuristics as described in section 9.3.1:
http://www.cs.berkeley.edu/~vazirani/algorithms/chap9.pdf

Beyond that you may:
• read anything about how to solve the traveling salesperson problem (you must cite any resources you use),
• use which ever programming language you want (so long as all group members are comfortable with it and I am easily capable of running it),

You may not:
• use existing implementations or subroutines for your submitted project,
• extensive libraries (if you are unsure, ask in the discussion forum for this project),
• use other people's code (other than that of your group members),

Your program must accept problem instances and options on the command line (if impossible in your language of choice, then contact me about it), needs to explicitly have a usage statement if any options are required (or recommended), and needs to name the output file named as the input file's name with .tour appended.
(for example, an input file named tsp_example_1.txt, needs to have an output called tsp_example_1.txt.tour):

1. You need have the ability to use command line arguments for the full execution of the program so that it can be run automatically with little effort (I tune some scripts to your problem and let it run on all the cases in sequence),

2. If I have to go digging through your code to understand an error, then you may lose points depending on the cause,

3. If your code does not compile on my desktop at home or the Flip server, then you will likely lose points,

4. This course may mostly be about learning how to get theoretical algorithms to run as actual problem solutions, but I will not deprive you the experience working with requirements, other people, and multiple systems (you will likely thank me if you get a software design, engineering, programming, or analyst position and then need to work with others regularly).

Input specification:

1. A problem instance (a particular list for the input) will always be given to you as a text file.
   (Note: this means a text encoded file, not necessarily a file ending with .txt)

2. Each line defines a city and each line has three numbers separated by white space,
   (Note: be careful with this requirement, as I may format my input files to be easy for a human to read by using columns with white space padding the values)

   a. The first number is the city identifier,
   b. The second number is the city's x-coordinate,
   c. The third number is the city's y-coordinate.

Output specification:

1. You must output your solution into another text file with n +1 lines where n is the number of cities,

2. The first line is to be the length of your tour you find,

3. The next n lines should contain the city identifiers in the order they are visited by your tour.
   a. Each city must be listed exactly once in this list,
   b. 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: We have provided you with three example instances. They are available in the directory http://web.engr.oregonstate.edu/~jessjo/CS325/TSP/, tsp_example_?.txt are provided according to the input specifications, tsp_example_?.txt.tour are example outputs corresponding to these three example cases. You should use the output instances to test that you are computing distances correctly. My best tour lengths for test cases 1, 2 and 3 are 108159, 2579, and 1573084, respectively (clearly not the values listed in the tours given...). You should use these values to judge how good your algorithm is.

Testing: A testing procedure tsp-verifier.py is available at http://web.engr.oregonstate.edu/~jessjo/CS325/TSP/ that we will use to verify your solutions. Usage to test example instance 1 is (requires TSPAllVisited.py):

```
python tsp-verifier.py example-input-1.txt example-output-1.txt
```

You should test that your outputs are correct.

Graded test instances:

On the morning of the day the assignment would normally be due (**Sunday morning unless I have changed my assignment schedule**), we will make available a number of test instances. The location may be posted on the project page (find this link to be ready for it when these cases are made available). Within 48 hours of these being posted, you will be required to submit an output file for each test instance according to the output specification and corresponding to each of these test instances as input files. These input files should be called test-input-1.txt, test-input-2.txt, test-input-3.txt, ... and your solutions should be submitted to TEACH subdirectory by a single member of the group (and no files should be submitted by other team members).

Note: Along with these test instances you need to finish and submit your project report and initial code submission as well.

This initial submission is a hard due date. You may have a chance to continue the project, but the initial findings must be turned in on time.

Project report

You will submit a project report along with your test instance solutions. The project report may only be up to 2 pages in length and with other formatting that the instructor is comfortable with (for Joseph that is: no less than 10pt font, no less than 0.5in margins, no larger than 8.5in by 11.0in digital paper). In this report you must describe the ideas behind your algorithm as completely as is possible (be succinct, take turns among group members reading the proposed drafts you create to ensure that it both makes sense and is complete).

Competition

We will also hold a "competition". The competition will require your program to find the best solution possible to one or more instances within a fixed amount of time (I have used 5 minutes per case in the past).

Termination procedure: The termination procedure (watch.py) is also provided in the directory http://web.engr.oregonstate.edu/~jessjo/CS325/TSP/. Two examples of how to write code that will work with this procedure are also given: simpleprimes.py and primes.py. Example usage:

```
# tries to sleep for 400 seconds, but will be terminated at the end of
# 300 seconds (5 minutes)

$ ./watch.py /bin/sleep 400

# finding prime numbers; a simple program that keeps writing
# the latest result to a file primes.txt - some what inefficient in that
# it writes the latest results each time a result is computed.

$ ./watch.py ./simpleprime.py

# finding prime numbers; continues calculations until a sigterm is received
# at which point, the primes.py writes a file last_prime.txt
# cleaner, and more efficient.

$ ./watch.py ./primes.py
```

Competition structure: There will be $\log_2(\text{classSize}) - 1$ required rounds and an approximately equal number of TSP tests. Everyone will compete in the first round on the first test. The best ½ of the projects will progress to the second round, the best ¼ teams to the third round and so on through the last round (where the final round will be between two projects). Ties will be broken by performance in earlier rounds. More than the number of teams given may progress to future rounds at the discretion of the judge. The competition instances will be of size $2^{\text{roundCount}} * 100$ city points (100, 200, 400, 800, 1600, 3200, more if needed...).

Grading rubric

30% of your project grade will be determined by your solutions to the test instances. You will be judged on how close your tour length is to that of the best possible solution. However, you will not be told what the optimal tour length is for the test instances. My current proposed formula (which may be altered somewhat depending on class performance) for this is (<my best tour> / <your best tour>) * 1.2; and yes, this means that you can then earn up to 120% of this section! Remember that doing well on this section (you have about 48 hours to run your code on the tests) does not necessarily mean you will do well in the competition because of the short duration of the competition tests.

50% of your project grade will be determined by your project report. You will be judged on clarity and creativity of your explanation for your algorithm and implementation.

20% of your project grade will be determined by your participation in the competition.

This will be awarded for full participation in finding a working, but not necessarily very short, solution to each competition test case (a penalty for each test case that your program cannot find a solution in the test time and more if it cannot find a solution in a reasonable time; determined by the time I have available to allow programs to run on my desktop or servers (only after they fail at 5 minutes will they be offloaded to other hardware to try a longer time, to keep the environment as similar as possible between tests)).

The benefit for winning through the rounds will be the right to brag to your fellow classmates that your program was more efficient or more robust than theirs.

Check list

1. Does your program correctly compute tour lengths for simple cases?

2. Does your program read input files and options all from the command line?

3. Does your program meet the output specification?

4. Did you check that you produce solutions that verify correctly (verification is easier after all, right?)?

5. Did you find solutions to the test instances by the final due date, which is 48 hours after the test cases are initially revealed?

6. Does your code run on flip (or on my desktop at home, Joseph has easy support for Java, C, C++, Python, and may be able to support other languages as well, so contact me if you are using a strange language)?

7. Have you checked the announcements on Blackboard for any updates before submission?

8. Have you (**after** checking the above requirements) submitted your project report, your solutions to the test cases, your source code for your program, and any comments you wish to include to TEACH?