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

You are given a set of \( n \) cities and, for each pair of cities \( c_1 \) and \( c_2 \), the distance 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 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 travelled visiting the cities in this order is:

\[
d(\text{Chicago, New York}) + d(\text{New York, New Orleans}) + d(\text{New Orleans, San Francisco}) + d(\text{San Francisco, Chicago})
\]

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 \( c_1 = (x_1, y_1) \) and \( c_2 = (x_2, y_2) \) is given by their Euclidean distance rounded to the nearest integer. In other words, you will compute the distance between city \( c_1 \) and \( c_2 \) as:

\[
d(c_1, c_2) = \text{nearest integer} \left( \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \right)
\]

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

\[
nint(\sqrt{10}) + nint(\sqrt{8}) + nint(\sqrt{10}) = nint(3.16\ldots) + nint(2.82\ldots) + nint(3.16\ldots) = 9
\]
Project specification

Your group (up to 3 people) 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 find optimal solutions. It is that difficult. But your goal is to find the best solution you can. 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](http://www.cs.berkeley.edu/~vazirani/algorithms/chap9.pdf)

Beyond that you may:

- read as much as you want about how to solve the travelling salesperson problem (so long as you cite any resources you use)
- use which ever programming language you want (so long as all group members are comfortable with it)

You may not use:

- existing implementations or subroutines
- extensive libraries (if you are unsure, check with me)
- other people's code (other than that of your group members)

Input specification  A problem instance (a particular input) will always be given to you as a text file. Each line defines a city and each line has three numbers separated by a white space. The first number is the city identifier, the second number is the city’s x-coordinate and the third number is the city’s y-coordinate.

Output specification  You must output your solution into another text file with \( n + 1 \) lines where \( n \) is the number of cities. The first line is to be the length of your tour you find. 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  We will provide you with some example instances. They are available in the directory [http://classes.engr.oregonstate.edu/eecs/winter2014/cs325-001/TSP/](http://classes.engr.oregonstate.edu/eecs/winter2014/cs325-001/TSP/) example-input-* are provided according to the input specifications. example-output-* are example outputs corresponding to these three example cases. You should use the output instances to test that you are computing distances correctly. The optimal tour lengths for test cases 1, 2 and 3 are 108159, 2579, and 1573084, respectively. You can use these values to judge how good your algorithm is.

Testing  A verifying procedure tsp-verifier.py is given in the same directory that we will use to verify your solutions. Usage to test example instance 1 is:

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

The verifier only verifies that your output contains a correct total distance value, not the optimality of the output. You should test that your program report the correct total distance.

Test instances

On March 14th at 4 PM, we will make available 3 test instances on blackboard. By March 15th at 4 PM, you will be required to submit 3 separate text files according to the output specification and corresponding to each of these test instances. These files should be called test1, test2, test3 and will be submitted to the TEACH subdirectory of one team member.

The deadline for this portion of the project is hard. No exceptions will be made without prior approval from Prof. Fern.
Project report

You will submit a project report on March 16th by 11:59PM. The project report may only be up to 2 pages in length in no less than 11pt font. In this report you must describe the ideas behind your algorithm as completely as is possible.

Grading rubric

60% of your project grade will be determined by your project report. You will be judged on clarity and creativity.

40% 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.

10% and 5% extra points will be given to the best and second best performing teams respectively.

Check list prior to submission

1. Does your program correctly compute tour lengths?

2. Does your program meet the output specification?

3. Did you check that you produce solutions that verify correctly?