CSCI 136 Assignment – TSP

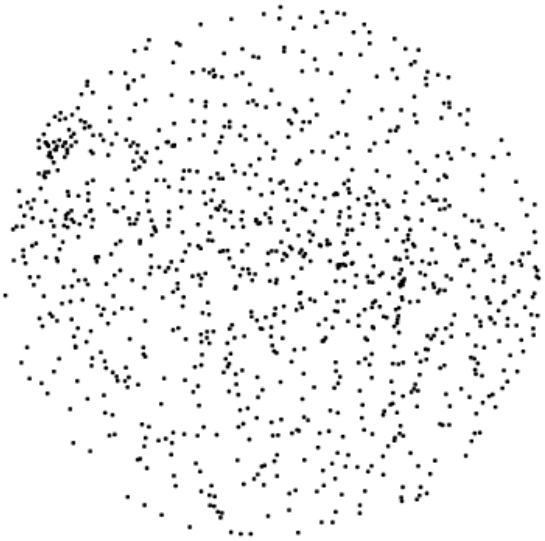
January 18th, 2023

Due Date: Sunday, 1/29/2023 11:59PM

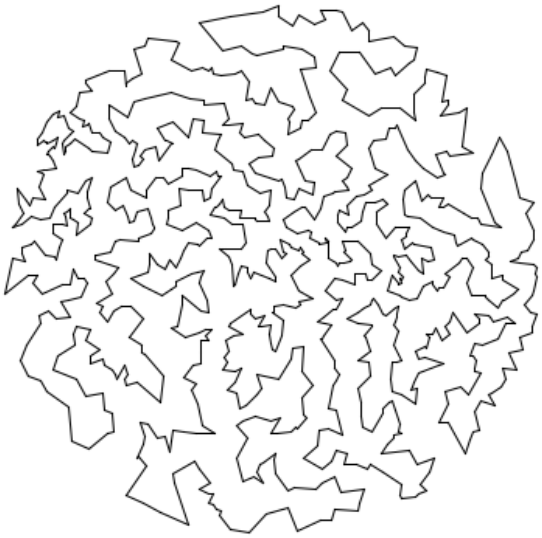
Points: 40

Topic: Linked Lists, the Traveling Salesman Problem

Given N points, the goal of the traveling salesman in the Traveling Salesman Problem is to visit all points (and arrive back at the start) while minimizing the total distance traveled. In this assignment, you will implement one straightforward (but not great) heuristic algorithm and two greedy heuristics to find good (but not optimal) solutions to the traveling salesman problem (TSP).



1,000 points



optimal tour

The importance of the TSP does not arise from an overwhelming demand of salespeople to minimize their travel distance, but rather from a wealth of other applications such as vehicle routing, circuit board drilling, VLSI design, robot control, X-ray crystallography, machine scheduling, and computational biology. It also serves as a "hard" problem that is of significance for theory of computation, algorithm design and other computer science topics.

The traveling salesman problem is a notoriously difficult combinatorial optimization problem. In principle, one could enumerate all possible tours and pick the shortest one; in practice, the number of tours is so staggeringly large (N factorial) that this approach is useless. For large N, no one knows an efficient method that can find the shortest possible tour for any given set of points. However, many methods have been studied, even though they are not guaranteed to produce the best possible tour. Such methods are called heuristics.

Your main task in this assignment is to implement the *in-order*, *nearest-neighbor* and *smallest-increase* insertion heuristics for building a tour incrementally. Start with a one-point tour (from the first point back to itself) and iterate the following process until there are no points left.

- In Order heuristic: Read in the next point and add it to the end of the current tour, after the previous point. This typically will not result in an efficient route, but it is a good starting point.
- Nearest Neighbor heuristic: Read in the next point and add it to the current tour after the point in the tour to which it is closest. If there is more than one point to which it is closest, insert it after the first such point.
- Smallest Increase heuristic: Read in the next point and add it to the current tour after the point where it results in the least possible increase in the tour length. If there is more than one point, insert it after the first such point you discover.

In this assignment, you will implement these heuristics to find paths through a set of points. In the process, you will gain experience with linked lists and learn about a famous computer science problem.

After following all instructions below, please submit your Python script (`Tour.py`) and a writeup file (`Tour.docx`) in Moodle. Upload these as individual files – do not zip them. Be sure the (main) file has a header section that includes the file name, your name, credits, a description of the file, the class and other useful information. Make sure your class and all methods are thoroughly commented.

- `Tour.py`
- `Tour.docx`

Grading:

This assignment is worth 40 points. You will be graded according to the following criteria:

Grade Item	Points
Program Compiles and Runs	2
Comments on All Classes, Methods and Header	4
Implemented __init__() Correctly	2
Implemented insertInOrder(p) Correctly	4
Implemented show() Correctly	2
Implemented draw() Correctly	2
Implemented size() Correctly	2
Implemented distance() Correctly	2
Implemented insertNearest(p) Correctly	5
Implemented insertSmallest(p) Correctly	5
Write Up of Results across Several Test Files	10
Total	**40**

You can use the included `TestTSP.py` script to test your code. Download this file and run it, as follows:

```
> python TestTSP.py
```

from the (Anaconda) command prompt.

The results `TestTSP.py` script will show if your code works and may point to areas where you need to do more work. The `TestTSP.py` script assumes you have named your program and classes as specified, and that all methods are implemented as described below.

Getting Started

Install pygame

To get `StdDraw` to work (see below), you need to have `pygame` installed. It should be installed for our use in the lab(s). If you find otherwise, please let me know as soon as possible. It does not install by default as part of the Anaconda Distribution or most Python installs. So, you will need to install it on your personal computer to get it to work with Anaconda.

The following worked for me to `pygame` installed:

- Open the Anaconda Prompt as Administrator. Right-click on it in the Start menu and select "Run as Administrator".
- At the command prompt, run:

```
> pip install pygame
```

To check if StdDraw is available, try the following in a Python script or in a Python console:

```
import StdDraw
```

Download and Extract TSPStudent.zip

Create a folder and unzip the included `TSPStudent.zip` into that folder. This archive contains data files as well class files you will use. Note that in the assignment you only need to create the `TSP.py` script, as the rest of the programs are already completed, and they should not be modified.

`TSPStudent.zip` includes the following files:

- Point.py (defines the Point class)
- color.py (defines a Color class for use by StdDraw.py)
- StdDraw.py (defines drawing functions)
- InOrderInsertion.py (client for In Order heuristic)
- NearestInsertion.py (client for Nearest Neighbor heuristic)
- SmallestInsertion.py (client for Smallest Insertion Cost heuristic)
- TestTSP.py (test program)
- tsp10.txt (10-point TSP problem)
- tsp100.txt (100-point TSP problem)
- tsp1000.txt (1000-point TSP problem)
- usa13509.txt (13509-city USA TSP problem)

Point Class (Point.py)

The `Point` data type represents a point in the plane, as described by the following class. This class has been implemented for you - you do not need to modify it.

```
class Point (2D point data type)
-----------------------------------------------------------------

# create the point (x, y)
__init__(float x, float y)

# return string representation
string toString()

# draw point using standard draw
draw()

# draw line segment between the two points
drawTo(Point that)

# return the Euclidean distance between the two points
float  distanceTo(Point that)
```

A `Point` object can return a string representation of itself, draw itself (using StdDraw), draw a line segment from itself to another point using standard draw, and calculate the Euclidean distance between itself and another point.

Tour Class and Node Class (Tour.py)

Name your script `Tour.py`. and implement the following class. See below for descriptions of what the class and its methods should do. It is important that your code implements the class as specified. Test your methods as you write them.

Your task is to create a `Tour` data type that represents the sequence of points visited in a TSP tour. Represent the tour as a circular linked list of nodes, one for each point. A circular linked list is one where the last data item in the list points back to the first one, rather than containing a `null` (or `None`) pointer. Each `Node` will contain a `Point` and a reference to the next `Node` in the tour.

Within `Tour.py`, define a class `Node` as follows:

```
class Node:
    def __init__(self):

       # This will hold the data, in this case a 2D point
         self.p = None

       # This will be the pointer to the next node
         self.next = None
```

The `Tour` class must implement the following:

```
class Tour:

       # create an empty tour
       __init__()

       # print the tour to standard output
       show()

       # draw the tour to standard draw
       draw()

       # number of points on tour
       int size()

       # return the total distance of the tour
       float distance()

       # insert p using in order heuristic
       insertInOrder(Point p)

       # insert p using nearest neighbor heuristic
       insertNearest(Point p)

       # insert p using smallest increase heuristic
       insertSmallest(Point p)
```

Each `Tour` object should be able to print its constituent points to standard output (`print` statements), draw its points (using `StdDraw`), count its number of points, compute its total length (distance), and insert a new point using any of the three heuristics. The constructor creates an empty tour (a linked list with no nodes).

Input and Testing

The input file format will begin with two whole numbers *w* and *h*, followed by pairs of *x*- and *y*-coordinates. All *x*-coordinates will be floating point numbers between 0 and *w*; all *y*-coordinates will be whole numbers between 0 and *h*.

Several test data files are available. As an example, `tsp1000.txt` contains the following data:

```
% type tsp1000.txt
775 768
185.0411 457.8824
247.5023 299.4322
701.3532 369.7156
563.2718 442.3282
144.5569 576.4812
535.9311 478.4692
383.8523 458.4757
329.9402 740.9576

...

254.9820 302.2548
```

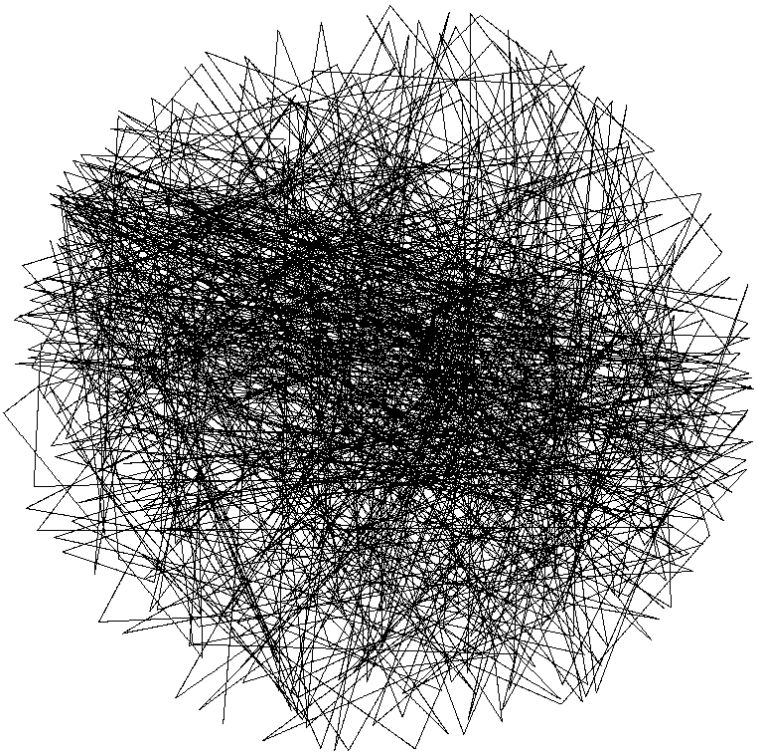
After implementing `Tour.py`, you can use the client programs `InOrderInsertion.py`, `NearestInsertion.py`, and `SmallestInsertion.py` to read in the points from a data file, run the associated heuristic, print the resulting tour and its distance to standard output, and draw the resulting tour using `StdDraw`. Your code will implement the methods to carry this out, as described above, and should be able to do this using only the methods in the `Point` class.

If you wish to test the drawing part in a test `main` function, you will need to import `StdDraw` in your `Tour.py` code. If you implement a test `main` function, please ensure that you do so in a way that will not interfere with importing `Tour.py`. This function should be defined inside `Tour.py` but not inside the `Tour` class. It should be defined so that it will only be executed if the `Tour.py` script is executed stand-alone, not imported into another script. (See prior separate slides and article about Defining Main Functions for guidance on this.)

Example Runs:

```
% python InOrderInsertion.py tsp1000.txt

Tour distance = 327693.76217093336
(185.0411 457.8824)
(247.5023 299.4322)
(701.3532 369.7156)
(563.2718 442.3282)
(144.5569 576.4812)
(535.9311 478.4692)
(383.8523 458.4757)
(329.9402 740.9576)
...
(254.9820 302.2548)
```



```
% python NearestInsertion.py tsp1000.txt

Tour distance = 27868.710634854797
(185.0411, 457.8824)
(198.3921, 464.6812)
(195.8296, 456.6559)
(216.8989, 455.126)
(213.3513, 468.0186)
(241.4387, 467.413)
(259.0682, 473.7961)
(221.5852, 442.8863)
...
(264.57, 410.328)
```

% python SmallestInsertion.py tsp1000.txt

```
Tour distance = 17265.628155352584
(185.0411, 457.8824)
(195.8296, 456.6559)
(193.0671, 450.2405)
(200.7237, 426.3461)
(200.5698, 422.6481)
(217.4682, 434.3839)
(223.1549, 439.8027)
(221.5852, 442.8863)
...
(186.8032, 449.9557)
```

You should test your code by running all methods against the included test files. Of course, it will take less time to run on data will smaller numbers of points. Your code should be able to find tours for tsp1000.txt (1000-point TSP problem) within a reasonable amount of time (less than a minute). Your code will take a longer time (minutes) running on usa13509.txt (13509-city USA TSP problem).

Tour.docx (writeup)

Using no more than 2 pages, write and submit a summary MS-Word document named Tour.docx. Include the following in this document:

- Provide the results (length of path and run time) of each heuristic program (in order, nearest and smallest) for the included files (`tsp10.txt`, `tsp100.txt`, `tsp1000.txt`, `usa13509.txt`).
- Explain the relative performance of the algorithms. Which is best? Which is second best? Which is worst? Please explain, including why you think this happens.

Helpful hints:

Use the example data files that are provided. Use the smaller ones to develop and test your code thoroughly before moving on to the larger data files.

Do not assume that because your code works on one data file that it will work on others. Test and debug thoroughly.

Work incrementally. Build and test incrementally, starting with easier aspects. Add functionality incrementally, testing with each change.

Review the examples from our book for text file input closely. Choose an approach that makes sense and works for you. There are different ways to accomplish the task at hand. Some are easier than others.

Review the slides and article about Defining Main Functions. Make sure you understand what is called for with the main function for this lab and how to make it work correctly in the several situations in which it will be used.

Review other related class slides.

I'm not sure how to get started. What do you recommend? A good way to proceed in this assignment (and most programs) is to implement methods one at a time, starting with the simplest method first. After creating each method, write test code to see if the new method does what is supposed to. In this assignment, you can implement whatever test code you like in the main method in `Tour.py`. One way to proceed is to implement the `Tour` constructor, then write test code that creates an empty tour. Once this is working, implement the next simplest method (e.g. `insertInOrder` should come first and then `size` or `distance`). Test each method and make sure you get the right results, and move on.

Do I need to follow the prescribed API (class and method descriptions)? Yes, we will be testing the methods in the API directly. If your method has a different signature or does not behave as specified, you will lose points. You may add methods to your code if they help you in your implementation, but make sure you implement the specified methods correctly.

Do I need to implement a circular linked list? Yes. That is what was specified and the code is simpler using a circular linked list than using a null-terminated single-linked list.

What should my program do if the tour contains 0 points? The `size()` method should return 0; the `distance()` method should return 0.0; the `show()` method should write nothing to standard output; the `draw()` method should draw nothing to standard draw.

How do I represent positive infinity in Python? Use `float('inf')`. You may find this helpful when searching for the nearest point or smallest increase.

How long should my programs take to execute? It took a (rather slow) Linux machine 2 minutes to solve the `usa13509.txt` problem with `InOrderInsertion.py`, and 8 minutes to solve it using `SmallestInsertion.py`. It could take substantially less time if you have a faster computer. If your code takes much longer to run, figure out why and try to fix it.

Can I see the algorithms in action? It is instructive to watch the tour update after each insertion. `InOrderInsertiong.py`, `SmallestInsertion.py` and `NearestInsertion.py` take an additional command-line argument (time in milliseconds) that puts the program into animation mode. Use a number less than 1.0. Animating the drawing significantly slows down the execution time.

Can I use Python's built in linked list library? No. A goal of this assignment is for you to gain experience writing and using linked structures.

What's the largest TSP tour ever solved exactly? The "record" for the largest TSP problem ever solved exactly is a 85,900-point instance that arose from microchip design in the 1980s. It took over 136 CPU-years to solve. I have a data file for this if you are interested.

What is the length of the best known tour for tsp1000.txt? At the last check, Mark Tengi held the Princeton COS 126 record with a tour of length 15892.3. Using the Concorde TSP solver, a solution of length 15476.519 was found. Currently Montana Tech CSCI 136 does not have a record.

Submit Your Work

After following all instructions above, please submit your Python script (Tour.py) and a writeup file (Tour.docx) in Moodle. Upload these as individual files – do not zip them. Be sure the (main) file has a header section that includes the file name, your name, credits, a description of the file, the class and other useful information. Make sure your class and all methods are thoroughly commented.

- `Tour.py`
- `Tour.docx`

EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

From me:

1.

One can observe that any tour with paths that cross can be transformed into a shorter one with no crossing paths. How would you write code for an associated algorithm that searches for such situations and makes the associated transformations?

2.
Implement an even better heuristic than the ones given. There are many approaches. You could be famous if you can break the Princeton record or that from the Concorde TSP Solver:

https://www.math.uwaterloo.ca/tsp/concorde.html.

To be "better" at the very least, the tour length should be consistently less than that for the smallest insertion heuristic!

This assignment was developed by Bob Sedgewick and Kevin Wayne, Princeton's assignment page (Copyright © 2000 Robert Sedgewick), and modified by Michele Van Dyne and Doug Galarus.