Project 6 (Traveling Salesperson Problem) Checklist

Prologue

Project goal: implement two greedy heuristics to find good (but not optimal) solutions to the *traveling salesperson problem* (TSP) — given N points in the plane, the goal of a traveling salesperson is to visit all of them (and arrive back home) while keeping the total distance traveled as short as possible

Files

- ⇝ `project6.pdf` (project description)
- ⇝ `project6.zip` (starter files for the problems, `report.txt` file for the project report, and `run_tests.py` file to test your solutions)

Problems

Student

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Instructor

Please summarize the project description for the students before you walk them through the rest of this checklist document.

Problems

Problem 1. (*Tour Data Type*) Implement a data type `Tour` that represents the sequence of points visited in a TSP tour, and supports the following API:

method	description
`Tour()`	create an empty tour t
`t.show()`	write the tour t to standard output
`t.draw()`	draw the tour t to standard draw
`t.size()`	the number of points in the tour t
`t.distance()`	the total distance of the tour t
`t.insertNearest(p)`	insert the point p to the tour t using the nearest neighbor heuristic
`t.insertSmallest(p)`	insert the point p to the tour t using the smallest increase heuristic

Hints

- ⇝ We represent a tour as a list of `Point` objects; the first element of the list is the point that the salesperson visits first and the last element is the point that the salesperson visits last before returning to the first point and completing the tour
- ⇝ Instance variable
  - ⇝ The tour, `_tour` (`list` of `Point` objects)

Problems

- `Tour()`
  - Initialize instance variable appropriately
- `t.show()`
  - Enumerate and write (one per line) each point in the tour
- `t.draw()`
  - Draw the tour by connecting the second point to the first, the third point to the second, and so on, and finally connecting the last point to the first — use `drawTo()` from `Point` to connect (ie, draw a line) between two points
- `t.size()`
  - Return the length of the tour
- `t.distance()`
  - Return the total tour distance calculated as the sum of: the distance of the second point to the first, distance of the third point to the second, and so on, and finally the distance of the last point to the first — use `distanceTo()` from `Point` to calculate the distance between two points

Problems

- `t.insertNearest(p)`
  - Find the index `0 <= i < tour size` such that `_tour[i].distanceTo(p)` is the smallest
  - Insert `p` (using the `insert()` method from `list`) into `_tour` at index `i + 1`
- `t.insertSmallest(p)`
  - For each `1 <= i <= tour size`, let `x = _tour[i - 1]` and `y = _tour[i % tour size]`
  - The increase in the current tour distance that results if `p` is inserted at `i` is given by `distance(x, p) + distance(y, p) - distance(x, y)`
  - Find the index `i` that results in the smallest such increase
  - Insert `p` (using the `insert()` method from `list`) into `_tour` at index `i`

Problems

Test data files (`*.txt`) and reference solutions (`*.ans`) are available under the `data` directory

The input format begins with two integers w and h, followed by pairs of $x \in (0, w)$ and $y \in (0, h)$ coordinates; for example

```
$ cat data/tsp10.txt
600 600

110.0 225.0
161.0 280.0
325.0 554.0
490.0 285.0
157.0 443.0
283.0 379.0
397.0 566.0
306.0 360.0
343.0 110.0
552.0 199.0
```

Problems

We provide the following text and visual client programs that you may use to test and debug your code:

- `nearest_insertion.py` is a text-based client that reads in points from standard input, runs the *nearest neighbor* heuristic, and prints to standard output the resulting tour along with its distance and the number of points.

```
$ python3 nearest_insertion.py < data/tsp10.txt
```

- `nearest_insertionv.py` is a visual client that reads in points from standard input, runs the *nearest neighbor* heuristic, prints to standard output the distance of the resulting tour along with the number of points, and displays the tour on standard draw.

```
$ python3 nearest_insertionv.py < data/tsp10.txt
```

- `smallest_insertion.py` is a text-based client that reads in points from standard input, runs the *smallest increase* heuristic, and prints to standard output the resulting tour along with its distance and the number of points.

```
$ python3 smallest_insertion.py < data/tsp10.txt
```

- `smallest_insertionv.py` is a visual client that reads in points from standard input, runs the *smallest increase* heuristic, prints to standard output the distance of the resulting tour along with the number of points, and displays the tour on standard draw.

```
$ python3 smallest_insertionv.py < data/tsp10.txt
```

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include

- ⇝ Time (in hours) spent on the project
- ⇝ Difficulty level (1: very easy; 5: very difficult) of the project
- ⇝ A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- ⇝ Acknowledgement of any help you received
- ⇝ Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files

- ⇝ Make sure your programs meet the style requirements by running the following command on the terminal

```
$ pycodestyle <program>
```

  where `<program>` is the `.py` file whose style you want to check

- ⇝ Make sure your programs meet the input and output specifications by running the following command on the terminal

```
$ python3 run_tests.py -v [<items>]
```

  where the optional argument `<items>` lists the exercises/problems (`Exercise1`, `Problem2`, etc.) you want to test, separated by spaces; all the exercises/problems are tested if no argument is given

- ⇝ Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- ⇝ Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit

1. `tour.py`
2. `report.txt`