Université de Haute-Alsace
Université franco-azerbaïdjanaise

Master DSAI
2020-2021

Project-Mode applied Programming in Python

Practical Work : A route planner

Goals

Build a Graphical User Interface for a route planner
Load CSV data
Implement search algorithms
Implement heuristics

Instructions

Use Python at 100%, Python comes with lot of tools to make your life easy, use them.

Think "object", use object programming at your advantage.

Code neatly with well-chosen variables and functions/methods names. Add useful comments to your code in order to be able to understand it in some days.

Try to respect coding styleguides. I advise you to follow python styleguide PEP8 [1] or Google Python Styleguide [2].

Simple is beautiful. Do not try to code complicated, keep it simple, it will be more efficient and less error-prone.

Think before you code, take some time to draw/write your idea on a sheet. The tinking time before you code will save you a lot of debugging time after.

Read the documentation links that I give you. When using a new library, a coder spends lot of time on its documentation to understand how it works. There is an expression dedicated to this : RTFM which means 'Read The F... Manual'.

Read carefully my instructions, they are here to help you (or try to ;-)).

---

1 Graphical User Interface

1.1 Let's create our graphical window and display the map

First, we will create a window showing the map of France in a canvas like the screenshot below.

1. PEP8 : https://www.python.org/dev/peps/pep-0008/
2. Google Python Styleguide : https://google.github.io/styleguide/pyguide.html

Canvas sizes depends on image size. Proposed images comes from Wikipedia.
— France map size 1066 × 1024
— France map size 799 × 768
— France map size 499 × 480

1.2 Load the CSV data

On the website, you got two CSV files, one `towns.csv` containing french towns that are capital of a department and an other `road.csv` containing roads between capitals or neighbor departments.

Load them, using the module `csv`[3] and put them in an efficient data structure. My advice would be to create classes town and road and put objects in adapted collections (list, dictionnaries, tuples, ...). Check that everything is ok by printing your collections.

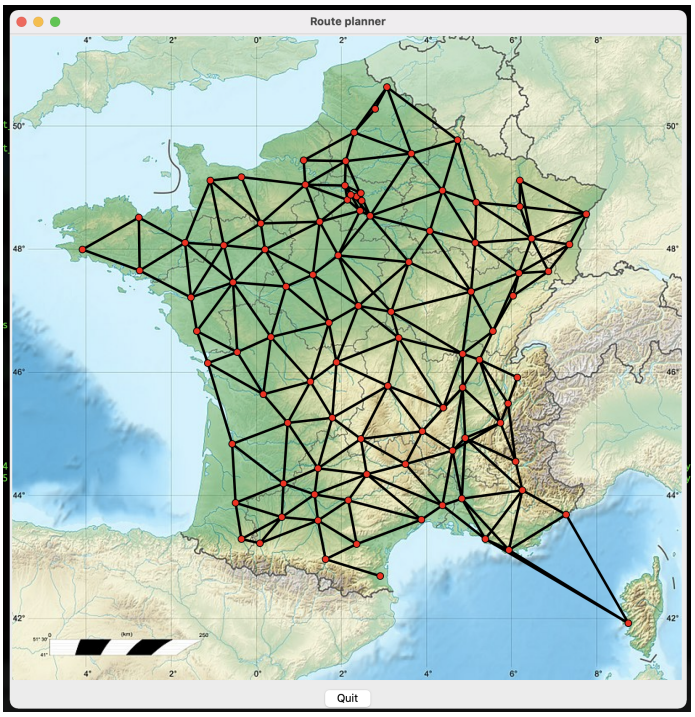
⚠Note that french departments are designed by number named `dept_id` in `towns.csv`. These are the same key as `town1` and `town2` in `road.csv`.

⚠In `road.csv`, distance is in kilometers and time in minutes.

1.3 Display towns and roads

Now, we want to display the towns and the roads in the canvas, as shown in the screenshot below.

3. Module csv : https://docs.python.org/3/library/csv.html



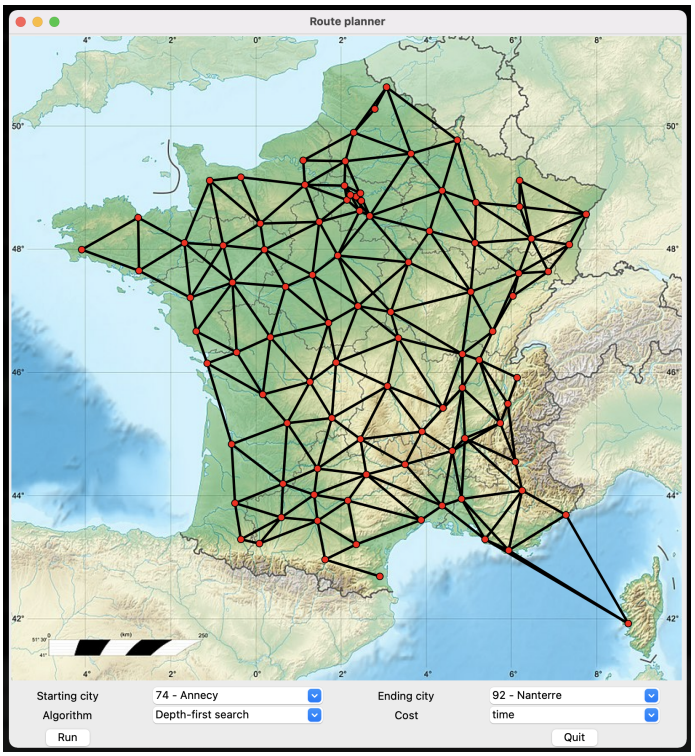
To do this properly, you need the coordinates of the edge of our map which are indicated below :

— North edge has latitude 51.5
— South edge has latitude 41
— West edge has longitude -5.8
— East edge has longitude 10

Using the edge coordinates you are able to convert town coordinates in longitude/latitude to pixel coordinates.

1.4 Add combobox and run button

Add combobox[4] for choosing starting and ending cities, but also search algorithm and heuristic if needed. You have also to add a "Run" button and our GUI will be ready. An illustration is shown below.



4. combobox : https://docs.python.org/3.8/library/tkinter.ttk.html#combobox

As shown in the screenshot, please provide random initialisation of current combobox choice.

2 Implementation of the algorithms

2.1 Implement Breadth-first search

Using the algorithm given in the lecture slides, implement the Breadth-first search.

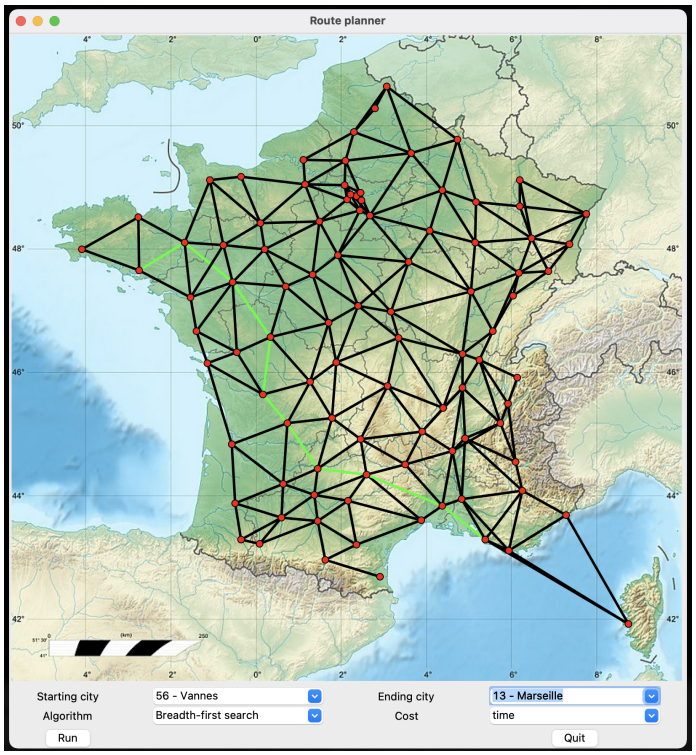
⚠ You will need to use `Queue` [5]

⚠ You will also have to create a callback function for the Run button which will launch the search.

⚠ You will also have to handle some information display, such as putting in green the path given by the search method.

⚠⚠⚠ This is a wide and complex task so cut it in simple problems and tackle them one by one!

At the end, you should obtain something looking similar to the screenshot below :



2.2 Some graphic and functionnal improvments

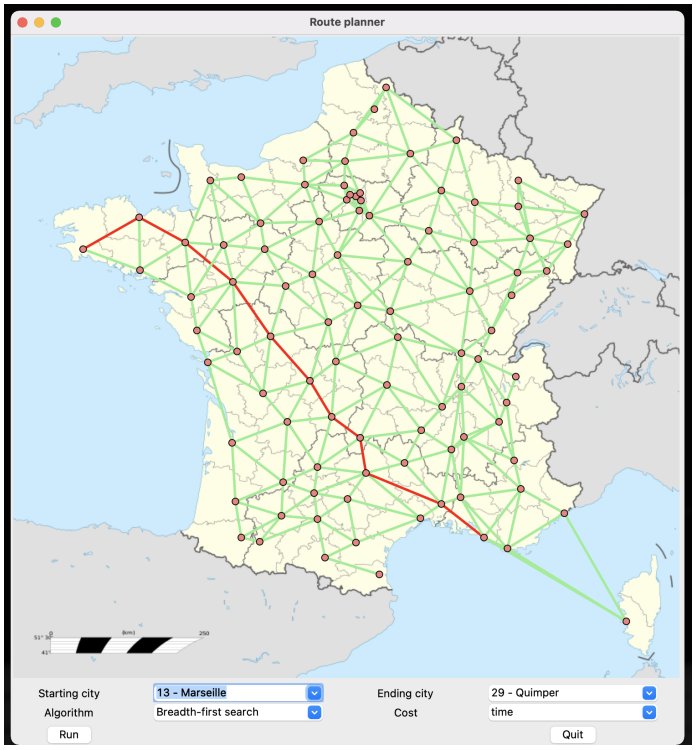
The colors chosen and shown in the previous screenshot are not well-chosen ... We can improve this by changing them and setting some global variables for each color type so it can be changed easily.

Now, we got another problem with this implementation, because you can click on the Run button several times and launch concurrently the same search, so we have to block the button once it is cliked and unblock it when the computation is over.

Do not forget to reput in normal colors the road once you run a new path search.

5. Queue : https://docs.python.org/3/library/queue.html

Moreover, you can change the map to use the administrative map as shown below with well-chosen colors.



2.3 Implement Depth-first search

Implement the Depth-first search, adapting the Depth-Limited Search for a recursive version or an iterative one if you want to take inspiration from BFS.

⚠ You will need to use `LifoQueue` [6]

2.4 Implement Uniform cost search

Using the algorithm given in the lecture slides, implement the uniform cost search.

⚠ You will need to use `PriorityQueue` [7]

⚠ You will need to modify class `Node` so it can be compared.

2.5 Implement Greedy search

Using the lecture slides, implement the greedy search.

You will have to defined the 'As the crow flies' distance.

2.6 Implement A*

Using the lecture slides, implement A*.

⚠ It will be tricky to manage the already traveled distance and the heuristic!

6. LifoQueue : https://docs.python.org/3/library/queue.html
7. PriorityQueue : https://docs.python.org/3/library/queue.html

2.7 Metrics and display

Add some metrics and display them on the window.

⚠Distance of the path, time of the path, computation time, number of nodes explored, ...

3 Bonus

You already finished? Congratulations, now you can handle the cost in time instead of only in distance.