Artificial Intelligence
Exercise 1

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

26. Oktober 2018

1 Programmieraufgabe: Tree Search

(The deadline for handing in your solution is Monday 2pm in the week of the tutorials)

In the repository you will find the directory `e01_graphsearch` with a couple of files. First there is `ex_graphsearch.py` with the boilerplate code for the exercise. The comments in the code define what each function is supposed to do. Implement each function and you are done with the exercise.

The second file you will find is `tests.py`. It consists of tests that check whether your functions do what they should. You don't have to care about this file, but you can have a look in it to understand the exercise better.

The next file is `data.py`. It consists of a very small graph and the S-Bahn net of Stuttgart as graph structure. It will be used by the test. If you like you can play around with the data in it.

The last file is `run_tests.sh`. It runs the tests, so that you can use the test to check whether you are doing right. Note that our test suite will be different from the one we hand to you. So just mocking each function with the desired output without actually computing it will not work. You can run the tests by executing:

```
$ sh run_tests.sh
```

If you are done implementing the exercise simply commit your implementation and push it to our server.

```
$ git add ex_graphsearch.py
$ git commit
$ git push
```

Task: Implement breadth-first search, uniform-cost search, limited-depth search, iterative deepening search and A-star as described in the lecture. All methods get as an input a graph, a start state, and a list of goal states. Your methods should return two things: the path from start to goal, and the fringe at the moment when the goal state is found (that latter allows us to check correctness of the implementation). The first return value should be the found `Node` (which has the path implicitly included through the parent links) and a `Queue` (one of the following: `Queue`, `LifoQueue`, `PriorityQueue` and `NodePriorityQueue`) object holding the fringe. You also have to fill in the priority computation at the `put()` method of the `NodePriorityQueue`.

Iterative Deepening and Depth-limited search are a bit different in that they do not explicitly have a fringe. You don't have to return a fringe in those cases, of course. Depth-limited search additionally gets a depth limit as input. A-star gets a heuristic function as input, which you can call like this:

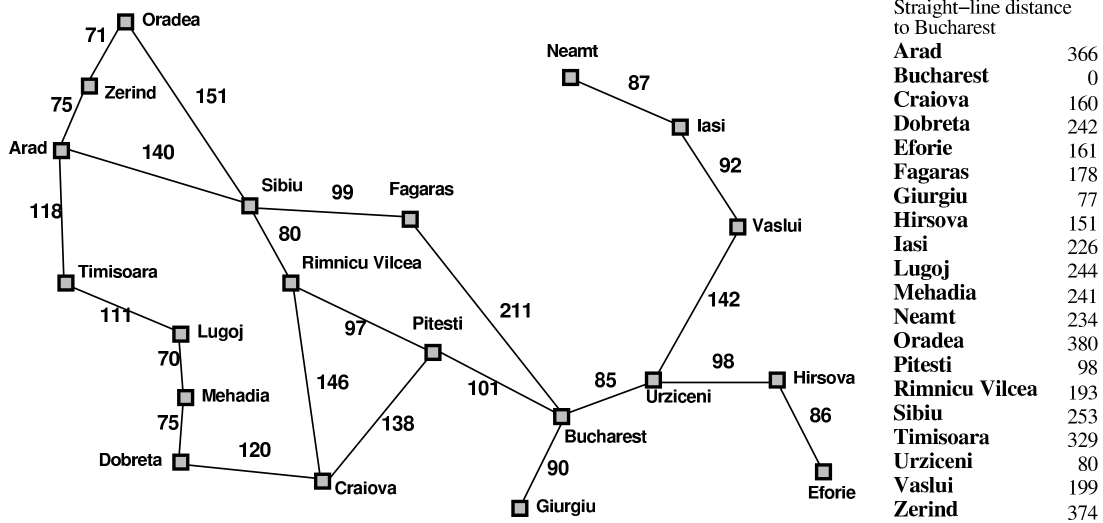
```
def a_star_search(graph, start, goal, heuristic):
    # ...
    h = heuristic(node.state, goal)
    # ...
```

Tips:

- For those used to IDEs like Visual Studio or Eclipse: Install PyCharm (Community Edition). Start it in the git directory. Perhaps set the Keymap to 'Visual Studio' (which sets exactly the same keys for running and stepping in the debugger). That's helping a lot.
- Use the data structure `Node` that is provided. It has exactly the attributes mentioned on slide 26.
- Maybe you don't have to implement the 'Tree-Search' and 'Expand' methods separately; you might want to put them in one little routine.

2 Votieraufgabe: A*-Suche

Betrachten Sie die Rumänien-Karte aus der Vorlesung:



- Verfolgen Sie den Weg von Lugoj nach Bukarest mittels einer A*-Suche und verwenden Sie die Luftlinien-Distanz als Heuristik. Geben Sie für jeden Schritt den momentanen Stand der *fringe* (Rand) an. Nutzen Sie folgende Notation für die fringe: $\langle (A : 0 + 366 = 366)(Z : 75 + 374 = 449) \rangle$ (d.h. $(\mathit{Zustand} : g + h = f)$).
- Geben Sie den mittels der A*-Suche gefundenen kürzesten Weg an.

3 Votieraufgabe: Beispiel für Tiefensuche

Betrachten Sie den Zustandsraum, in dem der Startzustand mit der Nummer 1 bezeichnet wird und die Nachfolgerfunktion für Zustand n die Zustände mit den Nummern $4n - 2$, $4n - 1$, $4n$ und $4n + 1$ zurück gibt. Nehmen Sie an, dass die hier gegebene Reihenfolge auch genau die Reihenfolge ist, in der die Nachbarn in `expand` durchlaufen werden und in die LIFO fringe eingetragen werden.

- Zeichnen Sie den Teil des Zustandsraums, der die Zustände 1 bis 21 umfasst.
- Geben Sie die Besuchsreihenfolge (Besuch=[ein Knoten wird aus der fringe genommen, goal-check, und expandiert]) für eine *beschränkte Tiefensuche mit Grenze 2* und für eine *iterative Tiefensuche*, jeweils mit Zielknoten 4, an. Geben Sie nach jedem Besuch eines Knotens den dann aktuellen Inhalt der *fringe* an. Die initiale fringe ist $\langle 1 \rangle$. Nutzen Sie für jeden Besuch in etwa die Notation:
  besuchter Zustand: $\langle$*fringe nach dem Besuch*
- Führt ein endlicher Zustandsraum immer zu einem endlichen Suchbaum? Begründen Sie Ihre Antwort.