Lab 9A

Priority queue

[Optional]

Author: Ragnar Nohre, moved to English Johannes Schmidt

This is the second of four optional labs. The optional labs can give together up to 10 *bonus points*. If all regular labs are passed by the general deadline, the accumulated bonus points will count as a bonus on the exam. The scale is that 10 bonus points correspond to 10% of the exam points. This *bonus rule* is only applicable to the first exam (December), and only if the exam is passed without the bonus points.

In order for this lab to qualify for bonus points, you must present and submit your solution before the first exam.

The 10 bonus points are distributed among the optional labs as follows:
Lab8A - 3 points
Lab9A - 1 point
Lab9B - 3 points
Lab11A - 3 points

In this lab you train your skills in using a priority queue. This as a preparation for the next optional lab 9B.

0.1 Task 1

From *Datastructures and Algorithms* we know what a *priority queue* is (and that it can be implemented with a min-heap). In the python module `queue` there is a priority queue class that we want to get familiar with.

Consider the following code. The function `test1()` creates a priority queue (pq) and inserts a number of *tuples* into the queue. To see how these are ordered we call `printAndPop(pq)` which prints all elements in the priority's order. The function also removes the all the elements from the priority queue, because there is no other way to iterate over the elements of a priority queue (cf. min-heap).

Test-run the following.

```
import queue

def printAndPop(pq):
    while pq.qsize()>0:
        print( pq.get() )


def test1():
    print("running test 1")

    pq = queue.PriorityQueue()

    pq.put( (4.0, 10) )
    pq.put( (2.0, 8) )
    pq.put( (5.0, 2) )
    pq.put( (1.5, 8) )
    pq.put( (4.0, 8) )
    pq.put( (1.0, 8) )

    printAndPop( pq)

test1()
```

Note that the queue "sorts" according to the tuple's 1st element and that the *smallest* number has highest priority. Note further that two of the

tuples have the same value as 1st element (4.0), and that (4.0, 8) is output before (4.0, 10).

0.1.1 Introducing another type of tuple

Modify the function `test1()` by adding the following line before the call to `printAndPop()`:

```
pq.put( (3.0, (1,2)) )
```

That line inserts thus a tuple with a more complicated structure.

Test-run the program and note that the new tuple ends up at the right place in the output.

0.1.2 Crash!

Modify again the function `test1()` by adding the folowing line between the previously added line and the call to `printAndPop()`

```
pq.put( (2.0, (1,2)) )
```

Test-run! If you have not modified any other values the program will *crash! Explain why it crashes now, but did not before.* Google if you need to. Answer!

0.2 Task 2

We have seen how to create and use a priority queue, and that problems can occur if one inserts objects of different types. The easiest fix to this is to create a new object type (a `class`). For the priority queue to be able to sort objects of the new type, the new class must define a method called `__lt__` (where lt stands for *less than*).
Add the following class definition to your code:

```
class Node:
    def __init__(self, prio, data):
        self.prio = prio
        self.data = data

    def __lt__(self, other):
        return self.prio<other.prio
```

Modify then `test1()` such that it inserts into the queue `Node`-objects instead of tuples (e.g. transform `pq.put( (3.0, (1,2) ) )` to `pq.put( Node(3.0, (1,2) ) )`.

Test-run the program. Note that the output of the `Node`-object is not very meaningful. This can be fixed in two ways. Either one adapts `printAndPop()` appropriately, or one defines a method that can convert a `Node`-object directly to a string. The latter method is more elegant and future proof, so we prefer it. Define the following method in your `Node`-class:

```
def __str__(self):
      return "({} {})".format(self.prio, self.data)
```

If you now test-run again everything should be fine and nice.

0.3 Examination

Submit your python file on Canvas and present your code and program to your lab assistant. Be able to explain what is marked with *Answer!*.