Lab 11

COMP9021, Session 2, 2016

1 ☞ A generalised priority queue

Write a program generalised_priority_queue.py that modifies priority_queue.py so as to insert pairs of the form *(datum, priority)*. If a pair is inserted with a datum that already occurs in the priority queue, then the priority is (possibly) changed to the (possibly) new value.

Next is a possible interaction.

```
$ python3
...
>>> from generalised_priority_queue import *
>>> pq = PriorityQueue()
>>> L = [('A', 13), ('B', 13), ('C', 4), ('D', 15), ('E', 9), ('F', 4), ('G', 5),
... ('H', 14),, ('A', 4), ('B', 11), ('C', 15), ('D', 2), ('E', 17), ('A', 8),
... ('B', 14), ('C',12), ('D', 9), ('E', 5), ('A', 6), ('B', 16)]
>>> for e in L:
...     pq.insert(e)
...
>>> for _ in range(8):
...     print(pq.delete(), end = ' ')
...
B H C D A G E F >>>
>>> print(pq.is_empty())
True
```

2 ☞ Special insertions into a priority queue

Write a program `special_insertions.py` that prompts the user to enter two nonnegative integer S and N and using `seed(S)` and `sample(range(N * 10), N)` (with `seed` and `sample` imported from the `random` module), randomly generates N distinct integers in the range $[0, 10N)$, inserts all these elements into a priority queue, and outputs a list L consisting of all those N integers, determined in such a way that:

- inserting the members of L from those of smallest index of those of largest index results in the same priority queue;
- L is preferred in the sense that the last element inserted is as large as possible, and then the penultimate element inserted is as large as possible, etc.

Next is a possible interaction.

```
$ python3 special_insertions.py
Enter 2 nonnegative integers: 0 3
The heap that has been generated is:
[None, 27, 12, 24]
The preferred ordering of data to generate this heap by successive insertions is:
[12, 24, 27]
$ python3 special_insertions.py
Enter 2 nonnegative integers: 0 5
The heap that has been generated is:
[None, 48, 24, 26, 2, 16]
The preferred ordering of data to generate this heap by successive insertions is:
[2, 26, 48, 16, 24]
$ python3 special_insertions.py
Enter 2 nonnegative integers: 0 7
The heap that has been generated is:
[None, 65, 53, 62, 33, 49, 5, 51]
The preferred ordering of data to generate this heap by successive insertions is:
[33, 49, 5, 53, 62, 51, 65]
$ python3 special_insertions.py
Enter 2 nonnegative integers: 0 10
The heap that has been generated is:
[None, 97, 61, 65, 49, 51, 53, 62, 5, 38, 33]
The preferred ordering of data to generate this heap by successive insertions is:
[5, 53, 62, 33, 38, 65, 97, 49, 51, 61]
$ python3 special_insertions.py
Enter 2 nonnegative integers: 0 15
The heap that has been generated is:
[None, 149, 130, 129, 107, 122, 124, 103, 66, 77, 91, 98, 10, 55, 35, 72]
The preferred ordering of data to generate this heap by successive insertions is:
[66, 91, 10, 107, 122, 35, 55, 77, 130, 98, 149, 124, 129, 72, 103]
```

3 Median and priority queues (optional)

Write a program `median.py` that implement a class `Median` that allows one to manage a list L of values with the following operations:

- add a value in logarithmic time complexity;
- return the median in constant time complexity.

One possible design is to use two priority queues: a max priority queue to store the half of the smallest elements, and a min priority queue to store the half of the largest elements. Both priority queues have the same number of elements if the number of elements in L is even, in which case the median is the average of the elements at the top of both priority queues. Otherwise, one priority queue has one more element than the other, and its element at the top is the median.

For the priority queue interface, reimplement `priority_queue.py` adding two classes, namely, `MaxPriorityQueue` and `MinPriorityQueue`, that both inherit from `PriorityQueue`, and add the right comparison function.

Next is a possible interaction.

```
$ python3
...
>>> from priority_queue import *
>>> max_pq = MaxPriorityQueue()
>>> min_pq = MinPriorityQueue()
>>> L = [13, 13, 4, 15, 9, 4, 5, 14, 4, 11, 15, 2, 17, 8, 14, 12, 9, 5, 6, 16]
>>> for e in L:
...     max_pq.insert(e)
...     min_pq.insert(e)
...
>>> max_pq._data[ : max_pq._length + 1]
[None, 17, 16, 15, 13, 15, 5, 14, 13, 6, 14, 11, 2, 4, 4, 8, 12, 9, 4, 5, 9]
>>> min_pq._data[ : min_pq._length + 1]
[None, 2, 4, 4, 5, 11, 4, 5, 9, 6, 13, 15, 13, 17, 8, 14, 15, 12, 14, 9, 16]
>>> for i in range(len(L)):...     print(max_pq.delete_top_priority(), end = ' ')
...
17 16 15 15 14 14 13 13 12 11 9 9 8 6 5 5 4 4 4 2 >>>
>>> for i in range(len(L)): print(min_pq.delete_top_priority(), end = ' ')
...
2 4 4 4 5 5 6 8 9 9 11 12 13 13 14 14 15 15 16 17 >>>


>>> from median import *
>>> L = [2, 1, 7, 5, 4, 8, 0, 6, 3, 9]
>>> values = Median()
```

```
>>> for e in L:
...     values.insert(e)
...     print(values.median(), end = ' ')
...
2 1.5 2 3.5 4 4.5 4 4.5 4 4.5 >>>
```