Lab 11

COMP9021, Session 2, 2015

1 ☞ Median and priority queues

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 derive from `PriorityQueue`, and add the right comparison function.

This implementation of `priority_queue.py` could contain

```
if __name__ == '__main__':
    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)
    print(max_pq._data[ : max_pq._length + 1])
    print(min_pq._data[ : min_pq._length + 1])
    for i in range(len(L)):
        print(max_pq.delete_top_priority(), end = ' ')
    print()
    for i in range(len(L)):
        print(min_pq.delete_top_priority(), end = ' ')
    print()
```

in which case testing this class would yield:

```
[None, 17, 16, 15, 13, 15, 5, 14, 13, 6, 14, 11, 2, 4, 4, 8, 12, 9, 4, 5, 9]
[None, 2, 4, 4, 5, 11, 4, 5, 9, 6, 13, 15, 13, 17, 8, 14, 15, 12, 14, 9, 16]
17 16 15 15 14 14 13 13 12 11 9 9 8 6 5 5 4 4 4 2
2 4 4 4 5 5 6 8 9 9 11 12 13 13 14 14 15 15 16 17
```

With this in place, the implementation of `median.py` could contain

```
if __name__ == '__main__':
    L = [2, 1, 7, 5, 4, 8, 0, 6, 3, 9]
    values = Median()
    for e in L:
        values.insert(e)
        print(values.median(), end = ' ')
    print()
```

in which case testing this class would yield:

```
2 1.5 2 3.5 4 4.5 4 4.5 4 4.5
```

2 A generalised priority queue

Reimplement `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. This implementation of `priority_queue.py` could contain

```
if __name__ == '__main__':
    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 i in range(8):
        print(pq.delete(), end = ' ')
    print()
    print(pq.is_empty())
```

in which case testing this class would yield:

```
B H C D A G E F
True
```