June 16th, 2020

Exercises for Programming Practice 2

Note:
- ✓ Do not create “module-info.java”, when you create a Java Project.
- ✓ Do not set Package name in the window “New Java Class”.
- ✓ Do not use the title of exercise for “contents of the program”. Think about “contents of the program” yourself.

The deadline for submitting the programs is 18:00 on June 18th, 2020.

The purpose of Exercise 12 - 14 is complete the Java program “HeapPriorityQueue.java” that is implementation of a heap-based priority queue using Tuple class which is for an item with key and value.

Use HeapTest class to check the behavior of HeapPriorityQueue class.

The classes “HeapTest.java” and “Tuple.java” can be obtained from Week 7 page Resource of “Programming Practice 2” course, manaba+R.

Tuple class is used for an item with key and value as shown in Fig. 1. It has only two variables: int key and String element. These values can be referenced by two getter methods.

ArrayList is used as an array-based representation of the heap in HeapTest class as follows:

```
ArrayList<Tuple> data = new ArrayList<Tuple>();
```

```
public class Tuple {
    int key;
    String element;

    public Tuple(int key_, String element_) {
        key = key_;
        element = element_;
    }

    public int getKey() {
        return key;
    }

    public String getElement() {
        return element;
    }
}
```

Fig. 1 Node class.

The method setData of HeapTest class is shown in Fig. 2. In this method, the heap represented by a binary tree in FIg. 3 is set to the array-based representation.

```
public ArrayList<Tuple> setData(){
    ArrayList<Tuple> data = new ArrayList<Tuple>();
    data.add(new Tuple(4, "C"));
    data.add(new Tuple(5, "A"));
    data.add(new Tuple(6, "Z"));
    data.add(new Tuple(15, "K"));
    data.add(new Tuple(9, "F"));
    data.add(new Tuple(7, "Q"));
    data.add(new Tuple(20, "B"));
    data.add(new Tuple(16, "X"));
    data.add(new Tuple(25, "J"));
    data.add(new Tuple(14, "E"));
    data.add(new Tuple(12, "H"));
    data.add(new Tuple(11, "S"));
    data.add(new Tuple(13, "W"));
    return data;
}
```

Fig. 2 The method setData of HeapTest class.

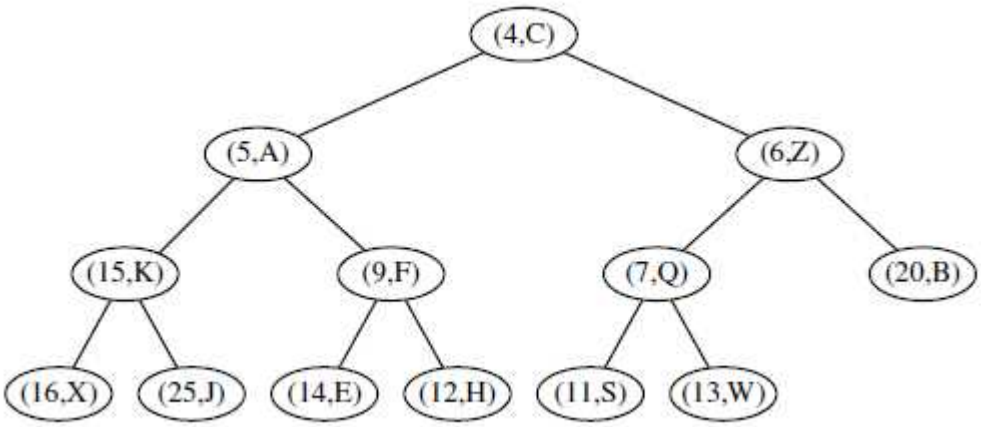

Fig. 3 An example of the heap represented by a binary tree.

The outline of “HeapPriorityQueue.java” is as follows.

```
import java.util.ArrayList;

public class HeapPriorityQueue {
    ArrayList<Tuple> data = null;
    Tuple tp;

    public HeapPriorityQueue(ArrayList<Tuple> data_) {
        data = data_;
    }

    public void showQueue() {
        // Complete this part

    public int parent(int j) {
        // Complete this part
    }

    public int left(int j) {
        // Complete this part
    }

    public int right(int j) {
        // Complete this part
    }

    public int len() {
        // Complete this part
    }

    public void add(int key, String element) {
        // Complete this part

    public void swap(int i, int j) {
        // Complete this part
}

    public void upheap(int j) {
        System.out.print("upheap: ");
        showQueue();
        // Complete this part
}

    public Tuple remove_min() {
        // Complete this part
    }

    public void downheap(int j) {
        int left, right, small_child;

        System.out.print("downheap: ");
        showQueue();
        // Complete this part
    }
}
```

Exercise 12 (file name “HeapPriorityQueue.java”)

Create the following methods of “HeapPriorityQueue.java”:

- ✓ public void showQueue() that display the contents of data (Array<Tuple>) with keys and elements in parentheses as shown in Flg. 5.
- ✓ pubic int len(): return a length of data.
- ✓ public int parent(int j)
- ✓ public int left(int j)
- ✓ public int right(int j)

For the last three methods, refer to the following document.

The array-based representation of a binary tree (Section 8.3.2) is especially suitable for a complete binary tree T. We recall that in this implementation, the elements of T are stored in an array-based list A such that the element at position p in T is stored in A with index equal to the level number $f(p)$ of p, defined as follows:

- If p is the root of T, then $f(p) = 0$.
- If p is the left child of position q, then $f(p) = 2f(q) + 1$.
- If p is the right child of position q, then $f(p) = 2f(q) + 2$.

The method ex12 of HeapTest is shown in Fig. 4. When using the ex12 method, uncomment “//ht.ex12()” in the main method of HeapTest.java.

Submit “HeapPriorityQueue.java” that satisfies Exercise 12. The execution result of method ex12 of HeapTest using it is as shown in Fig. 5.

```
public void ex12() {
    data = setData();
    HeapPriorityQueue hpq = new HeapPriorityQueue(data);
    hpq.showQueue();
    System.out.println("the parent of position 4 is position " + hpq.parent(4));
    System.out.println("the left node of position 1 is position " + hpq.left(1));
    System.out.println("the right node of position 1 is position " + hpq.right(1));
    System.out.println("the size of Heap is " + hpq.len());
}
```

Fig. 4 The method ex12 of “HeapTest.java”.

```
(4, C) (5, A) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W)
the parent of position 4 is position 1
the left node of position 1 is position 3
the right node of position 1 is position 4
the size of Heap is 13
```

Fig. 5 The execution result of method ex12 of “HeapTest.java”.

Exercise 13 (file name “HeapPriorityQueue.java”)

Create the following methods of “HeapPriorityQueue.java”:

- ✓ public void add(int key, String element)
- ✓ public void upheap(int j)
- ✓ public void swap(int i, int j)

Please refer Python program as shown in Fig. 6.

The method ex13 of HeapTest is shown in Fig. 7. When using the ex13 method, uncomment “//ht.ex13()” in the main method of HeapTest.java..

Submit “HeapPriorityQueue.java” that satisfies Exercise 13. The execution result of method ex13 of HeapTest using it is as shown in Fig. 8.

```
def add(self, key, value):
  """Add a key-value pair to the priority queue."""
  self._data.append(self._Item(key, value))
  self._upheap(len(self._data) - 1)          # upheap newly added position

def _upheap(self, j):
  parent = self._parent(j)
  if j > 0 and self._data[j] < self._data[parent]:
    self._swap(j, parent)
    self._upheap(parent)                     # recur at position of parent

def _swap(self, i, j):
  """Swap the elements at indices i and j of array."""
  self._data[i], self._data[j] = self._data[j], self._data[i]
```

Fig. 6 Python program.

```
public void ex13() {
    data = setData();
    HeapPriorityQueue hpq = new HeapPriorityQueue(data);
    System.out.println("The first situation of heap is");
    hpq.showQueue();
    hpq.add(2, "T");
    System.out.println("The last situation of heap is");
    hpq.showQueue();
}
```

Fig. 7 The method ex13 of “HeapTest.java”.

```
The first situation of heap is
(4, C) (5, A) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W)
upheap: (4, C) (5, A) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W) (2, T)
upheap: (4, C) (5, A) (6, Z) (15, K) (9, F) (7, Q) (2, T) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W) (20, B)
upheap: (4, C) (5, A) (2, T) (15, K) (9, F) (7, Q) (6, Z) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W) (20, B)
upheap: (2, T) (5, A) (4, C) (15, K) (9, F) (7, Q) (6, Z) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W) (20, B)
The last situation of heap is
(2, T) (5, A) (4, C) (15, K) (9, F) (7, Q) (6, Z) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W) (20, B)
```

Fig. 8 The execution result of method ex13 of “HeapTest.java”.

Exercise 14 (file name “HeapPriorityQueue.java”)

public Typle remove_min()

public void downheap(int j)

Create the following methods of “HeapPriorityQueue.java”:

- ✓ public Typle remove_min()
- ✓ public void downheap(int j)

Please refer Python program as shown in Fig. 9.

The method ex14 of HeapTest is shown in Fig. 10. When using the ex14 method, uncomment “//ht.ex14()” in the main method of HeapTest.java.

Submit “HeapPriorityQueue.java” that satisfies Exercise 14. The execution result of method ex14 of HeapTest using it is as shown in Fig. 11.

```
def remove_min(self):
  """Remove and return (k,v) tuple with minimum key.

  Raise Empty exception if empty.
  """
  if self.is_empty( ):
    raise Empty('Priority queue is empty.')
  self._swap(0, len(self._data) - 1)        # put minimum item at the end
  item = self._data.pop( )                  # and remove it from the list;
  self._downheap(0)                         # then fix new root
  return (item._key, item._value)

def _downheap(self, j):
  if self._has_left(j):
    left = self._left(j)
    small_child = left                      # although right may be smaller
    if self._has_right(j):
      right = self._right(j)
      if self._data[right] < self._data[left]:
        small_child = right
    if self._data[small_child] < self._data[j]:
      self._swap(j, small_child)
      self._downheap(small_child)           # recur at position of small child
```

Fig. 9 Python program.

```
public void ex14() {
    data = setData();
    HeapPriorityQueue hpq = new HeapPriorityQueue(data);
    System.out.println("The first situation of heap is");
    hpq.showQueue();
    Tuple tu = hpq.remove_min();
    if(tu != null) {
        System.out.println("(" + tu.getKey() + ", "
                                + tu.getElement() + ") is removed");
    }
    System.out.println("The last situation of heap is");
    hpq.showQueue();
}
```

Fig. 10 The method ex14 of “HeapTest.java”.

```
The first situation of heap is
(4, C) (5, A) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S) (13, W)
downheap: (13, W) (5, A) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S)
downheap: (5, A) (13, W) (6, Z) (15, K) (9, F) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S)
downheap: (5, A) (9, F) (6, Z) (15, K) (13, W) (7, Q) (20, B) (16, X) (25, J) (14, E) (12, H) (11, S)
downheap: (5, A) (9, F) (6, Z) (15, K) (12, H) (7, Q) (20, B) (16, X) (25, J) (14, E) (13, W) (11, S)
(4, C) is removed
The last situation of heap is
(5, A) (9, F) (6, Z) (15, K) (12, H) (7, Q) (20, B) (16, X) (25, J) (14, E) (13, W) (11, S)
```

Fig. 11 The execution result of method ex14 of “HeapTest.java”.