1st Part: Implement priority queue

Based on given PQInterface interface, you will implement your priority queue (MyPQ) and heap (MyHeap). The PQInterface interface is below:

public interface PQInterface<E> {

}

void insert(E entry); void clear(); E removeMin(); E min();

int size(); boolean isEmpty();

This interface has the methods’ description of priority queue data structure. As you see in the class, elements in the priority queue consist of entries which are key and value pairs. Therefore, we give Entry interface for you and you should implement it as well for the second phase of the project.

public interface Entry<K,V> extends Comparable<K>{ K getKey();

V getValue(); void setKey(K key);

}

void setValue(V Value);

MyPQ implementation:

Priority policy of this implementation will be the entry which has minimum key. Simply, you keep your priority queue always sorted so that removeMin() or min() method will access first entry of the priority queue . Therefore, when you insert an entry you should add it in correct place that does not breaks order. For example:

Insert((25,a))

Insert((30,b))

Insert((10,b))

Insert((28,c))

MyPQ class is below. Since it has unimplemented methods, it will give some errors so you should implement all methods in the PQInterface interface.

public class MyPQ<E> implements PQInterface<E>{

private ArrayList<E> entryList;

public MyPQ() { entryList = new ArrayList<E>();

(25,a)

|  |  |  |  |
| --- | --- | --- | --- |
| (25,a) | | (30,b) | |
| (10,b) | (25,a) | (30,b) | |
| (10,b) | (25,a) | (28,c) | (30,b) |

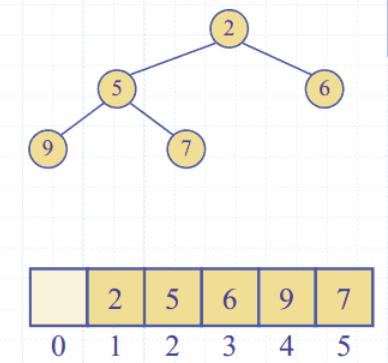
   

} }

MyHeap Implementation:

MyHeap class will be based on vector based heap implementation that is told in the class. It implements also PQInterface. The difference between MyHeap and MyPQ is their positioning. In MyHeap, the entries should be poisoned according to heap order.

So, when you insert new node or remove minimum node you should preserve the rule of heap tree structure in the arraylist.



MyHeap class is below. Since it has unimplemented methods, it will give some errors so you should implement all methods in the PQInterface interface.

import java.util.ArrayList;

public class MyHeap<E> implements PQInterface<E> {

private ArrayList<E> entryList;

public MyHeap(){ entryList = new ArrayList<E>();

} }

2nd Part: Implement Huffman Coding

You will implement Huffman Coding algorithm with using MyHeap and MyPQ data structure that you implement. Firstly, we give brief information about what is Huffman Coding and then we will explain how you will implement.

Huffman Coding Algorithm: Huffman encoding is a way to assign binary codes to symbols that reduces the overall number of bits used to encode a typical string of those symbols. For example, if you use letters as symbols and have details of the frequency of occurrence of those letters in typical strings, then you could just encode each letter with a fixed number of bits, such as in ASCII codes.

You can do better than this by encoding more frequently occurring letters such as e and a, with smaller bit strings; and less frequently occurring letters such as q and x with longer bit strings with using Huffman Coding Algorithm.

Example: aaabbbccdeeeee (letter, frequency)

Choose least frequent two node: (c and d) and combine it. (add their frequencies)

c,2 d,1

Choose least frequent two node from second level. (cd and b) and combine them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a,3 | b,3 | c,2 | d,1 | e,5 |

cd,3 a,4 b,3 e,5



cd,3 c,2 bcd,6 b,3 d,1

a,4 e,5

Choose least frequent two node from third level. (a and e) and combine them

cd,3 c,2 bcd,6 b,3 d,1

a,4 ae ,10

e,5

Choose least frequent two node from fourth level. (ae and bcd) and combine them

bcd=6

ae =10

cd=3 b=3

a=4 e=5

c=2 d=1

abcde = 16

Assign 0 to all left children and 1 to all right children.

0 c,2 cd,3

bcd,6 d,1 0 b,31

0

|  |
| --- |
| 1 |
|  |
| 0 |

abcde , 16

1

a,4 e,5

1

ae ,10

Check the path from the leafs to the root to find the code. So the characters are encoded as:

a=01, b=10, c=000, d=100, e = 11 And the encoded version of the text aaabbbccdeeeee is 0101011010100000001001111111111

Classic coding:

In classic coding, each character is encoded by fixed size code. For example, the above example has 5 different characters. So we can express each 5 character with using 3 ( bits. So one possibility is:

a=000, b=001, c=010, d=011, e=100 So the coded text aaabbbccdeeeee is 000000000001001001010010011100100100100100 As you see this coding is longer than Huffman coding so it is better to use Huffman coding.

Huffman Coding with Priority Queue:

HuffmanCoding class is given. You should implement all methods of it as described below. In addition you should implement the interface Entry with given class HuffmanEntry.

import

import

import import

public

java.io.BufferedWriter;

java.io.FileWriter;

java.io.IOException; java.util.ArrayList;

class HuffmanCoding {

private ArrayList<HuffmanEntry> huffEntries; //This list holds the nodes (parent or child) that you create

coding

private PQInterface<HuffmanEntry> huffPQ; //This PQ is used used for Huffman algorithm

public HuffmanCoding(PQInterface<HuffmanEntry> huffPQ){ this.huffPQ = huffPQ;

}

public void encoding(){ int i = 1;

FileWriter outFile = null; BufferedWriter outBuff = null; try{

outFile = new FileWriter("Huffman.txt");

outBuff = new BufferedWriter(outFile); }catch (IOException e){

System.err.println("Error: " + e.getMessage());

}

* //  Start loop
* //  write entries in the heap with desired format.

* //  encode huffEntries according to algorithm that is
* //  given in the project description.
* //  increment i.
* //  End loop  } public String getCode(Character c){

// returns assigned code to the c

} }

Encode [ encode()] Repeat the following process until you have 1 entries in the heap defined in the HuffmanCoding class:

* 

1)

2) 3)

4)

5)

Write all pair of entries in the heap to the text file “Huffman.txt” with following format: Step i: (key1,value1) (key2,value2) ... (keyn,valuen) Remove from the huffPQ the two entries with the smallest frequencies. Combine those two entries into a single entry whose name is parent entry. Add information to the parent entry which is one of the two entries is its left child and the other is right child and encode links between parent and child as described below.

* Assign 0 the link between parent and its left child.
* Assign 1 the link between parent and its right child. Add removed entries and new parent entries are in huffEntries defined in the HuffmanCoding  class. Add the parent entry to the myPQ.

Getting assigned code: [getCode(String c)] You should return the code of c (assigned bits of its parents and grandparents)

For example, it should return “100” if I call getCode(“d”) according to above example of Huffman Coding.

Testing your code

1) Read the given input file “input.txt” and assign the text to the static variable text in the TestHuffman class.

2) Calculate frequencies of each character including space.

Each character and its frequencies will construct entry where value is character and key is frequencies.

3) You have two PQInterface object and insert all the entries to them.

PQInterface<HuffmanEntry> myHeap = new MyHeap<HuffmanEntry>(); PQInterface<HuffmanEntry> myPQ = new MyPQ<HuffmanEntry>();

* 4)  You find the assigned codes to each character and write it to “coding.txt”. Then you will encode the given text according to assigned codes.
* 5)  Then you do classic coding as described above and write the each assigned codes to each character along with encoded text to “coding.txt”. For example for the above example of the “coding.txt” should be written following format. (Note that the order of character is not important)  Huffman Codding: a=01 b=10 c=000  d=100 e=11 0101011010100000001001111111111 The size of the coded text is 31. Classic coding: a=000, b=001 c=010 d=011 e=100 000000000001001001010010011100100100100100 The size of the coded text is 42.

import java.io.BufferedWriter; import java.io.FileWriter; import java.io.IOException;



public class static

public

TestHuffman { String text;

static void main(String[] args) {

* //  Read input data from the file: input.txt
* //  for each character find its frequencies in the text
* //  then create HuffmanEnrty object
* //  and put it in the the following priority queues  PQInterface<HuffmanEntry> myHeap = new MyHeap<HuffmanEntry>(); PQInterface<HuffmanEntry> myPQ = new MyPQ<HuffmanEntry>();

coding.txt

// write the classic code version (classicCodedTxt) to

outBuff.write("Classic Coding:"); outBuff.newLine(); outBuff.write(classicCodedTxt); outBuff.newLine(); outBuff.write("The size of the coded text is " +

HuffmanCoding code1 = new HuffmanCoding(myHeap); HuffmanCoding code2 = new HuffmanCoding(myPQ);

code1.encoding(); code2.encoding();

try

{

FileWriter outFile = new FileWriter("coding.txt"); BufferedWriter outBuff = new BufferedWriter(outFile); outBuff.write("Huffman Coding:"); String huffCodedTxt = null;

// Find the assigned codes of each character // and encode the text according to it then // assign it to huffCodedTxt variable.

// write assigned codes of each character to the coding.txt // in addition write the huffCodedTxt to the file.

outBuff.newLine(); outBuff.write(huffCodedTxt); outBuff.newLine(); outBuff.write("The size of the coded text is " +

huffCodedTxt.length()); String classicCodedTxt = classicEncoding();



classicCodedTxt.length()); outBuff.close();

}

catch (IOException e) {

System.err.println("Error: " + e.getMessage());

}

//compare if code1 and code2 find the same codes for each character.

}

public static int findFrequency(String c){

* //  find frequency of c in text
* //  and return it  }



} }

public static String classicEncoding(){



// // // //

encode the text according to classic encoding (same length codes) that is describe the project

and return the coded text