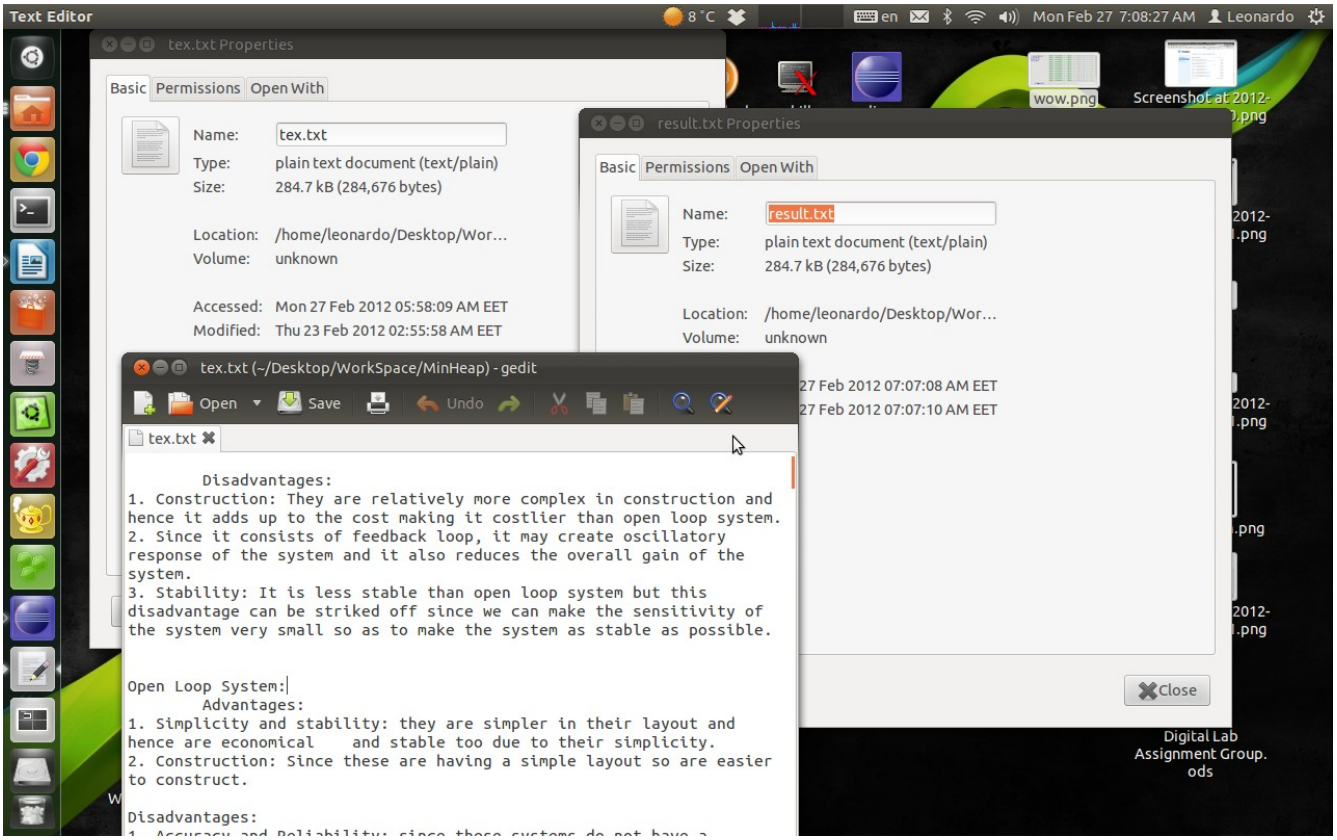
Lab 2 Report
(MinHeap + Huffman Encoding)

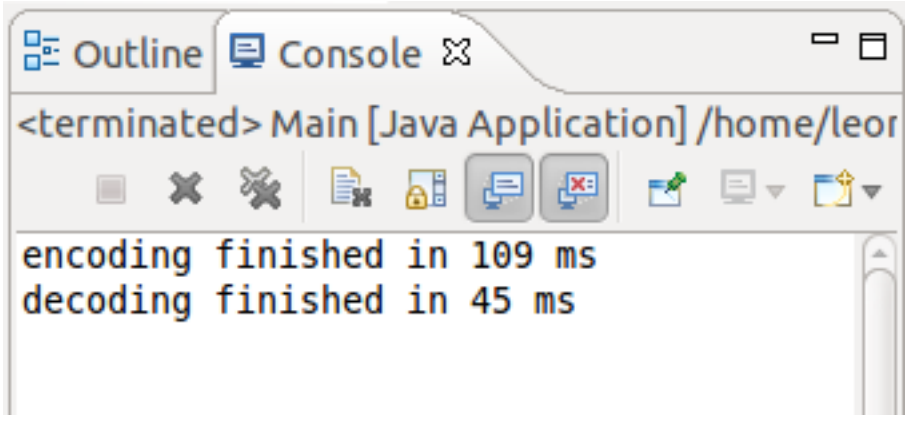
Ashraf Saleh Mohamed Aly (20)

Sample Runs:

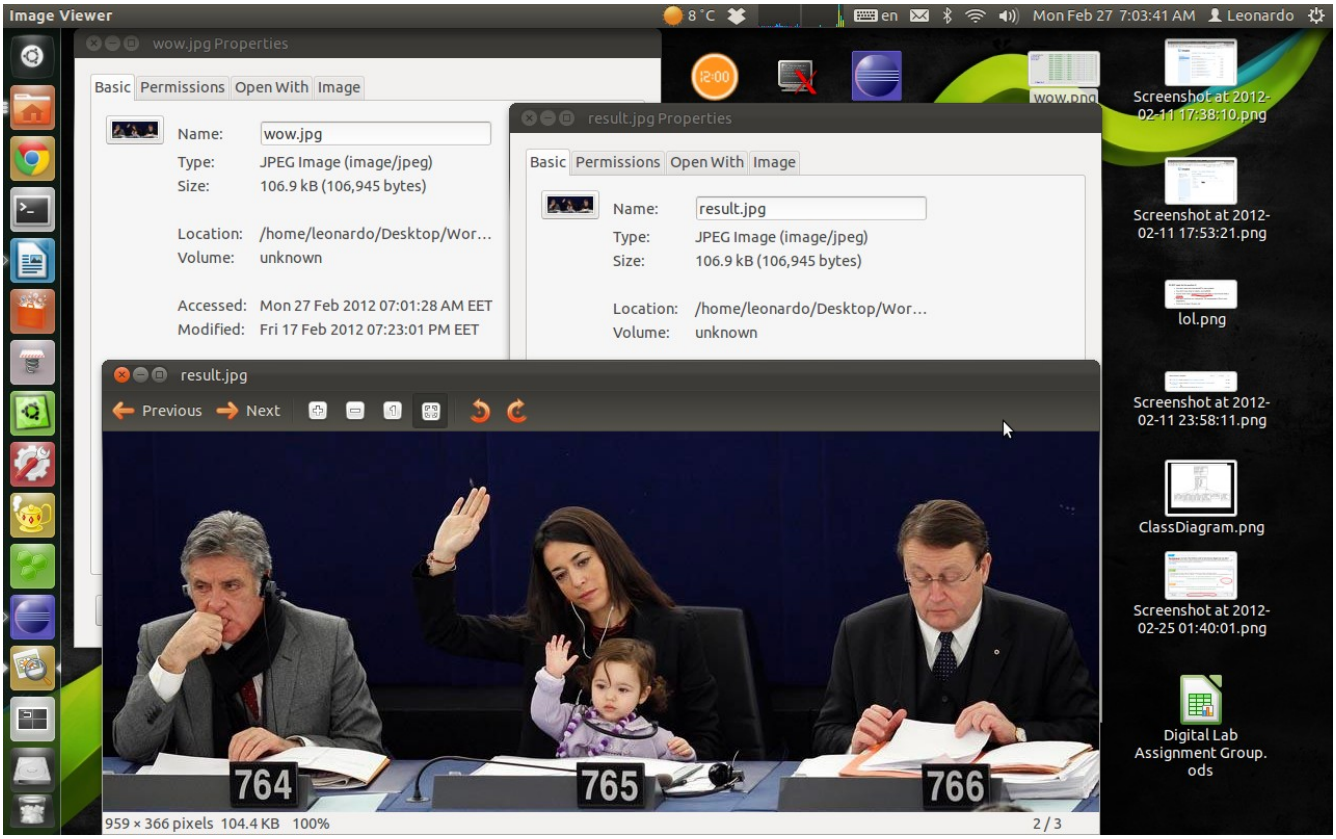
text file :



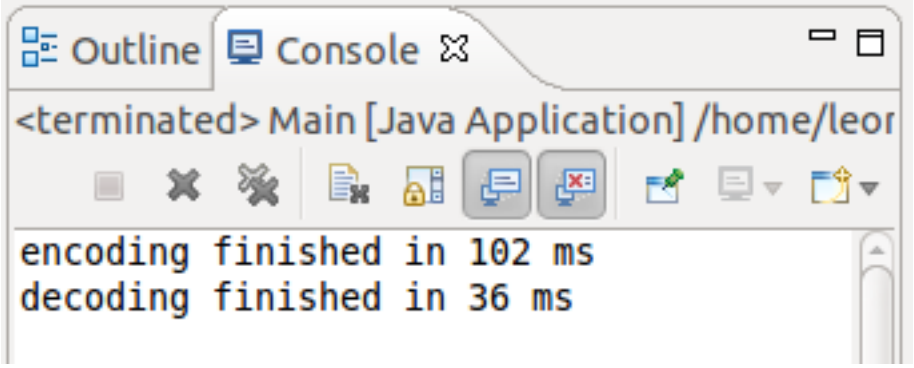
time



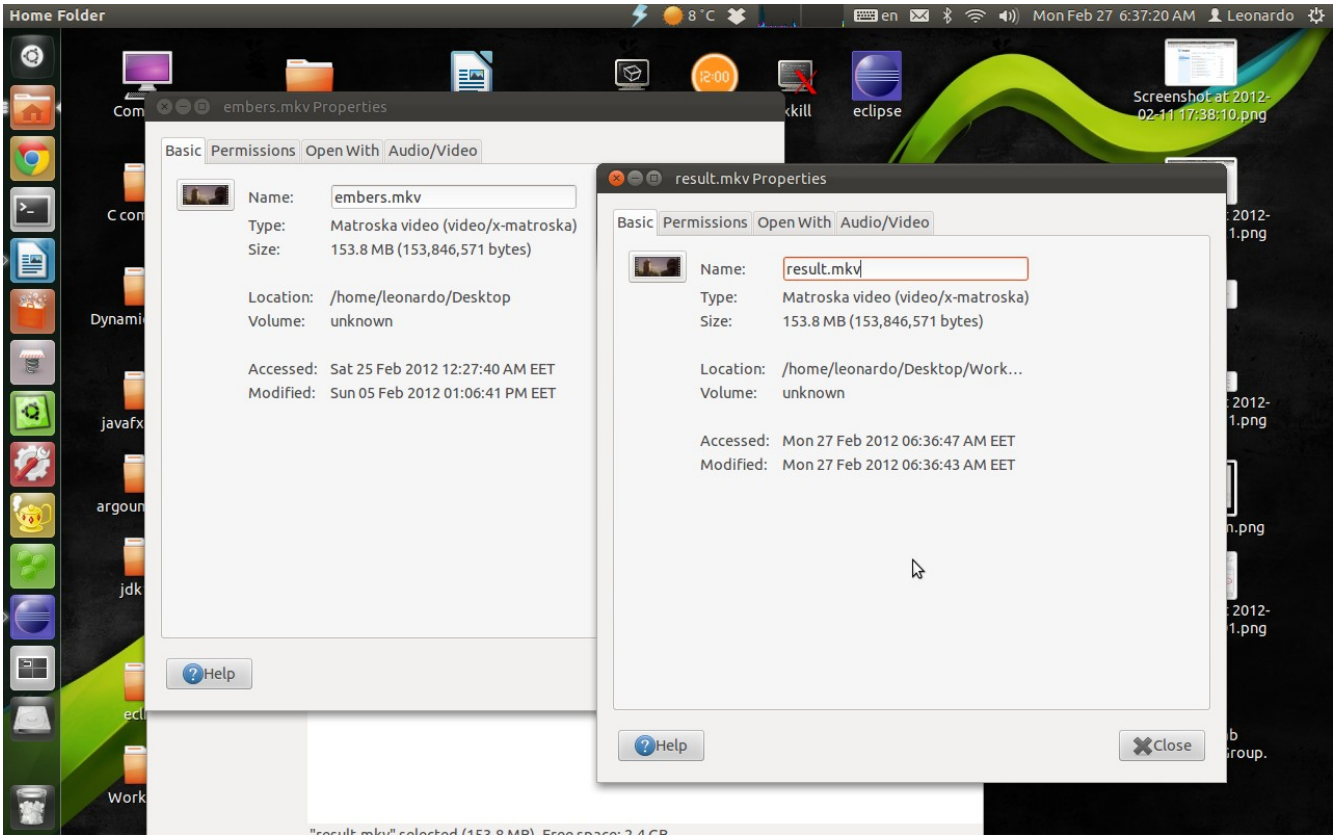
Picture:



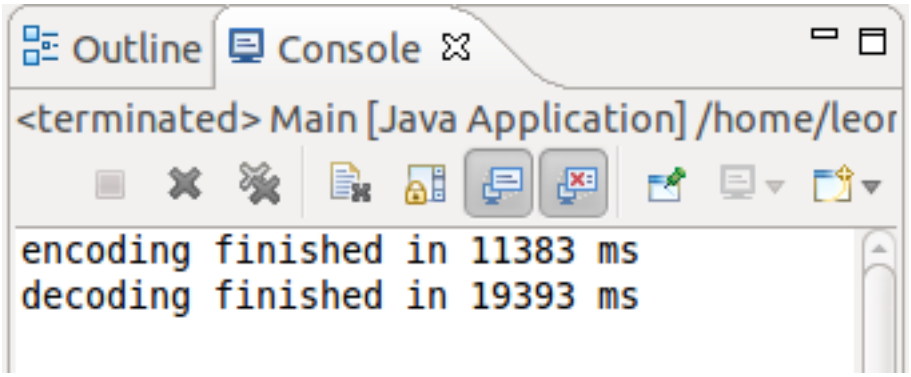
time



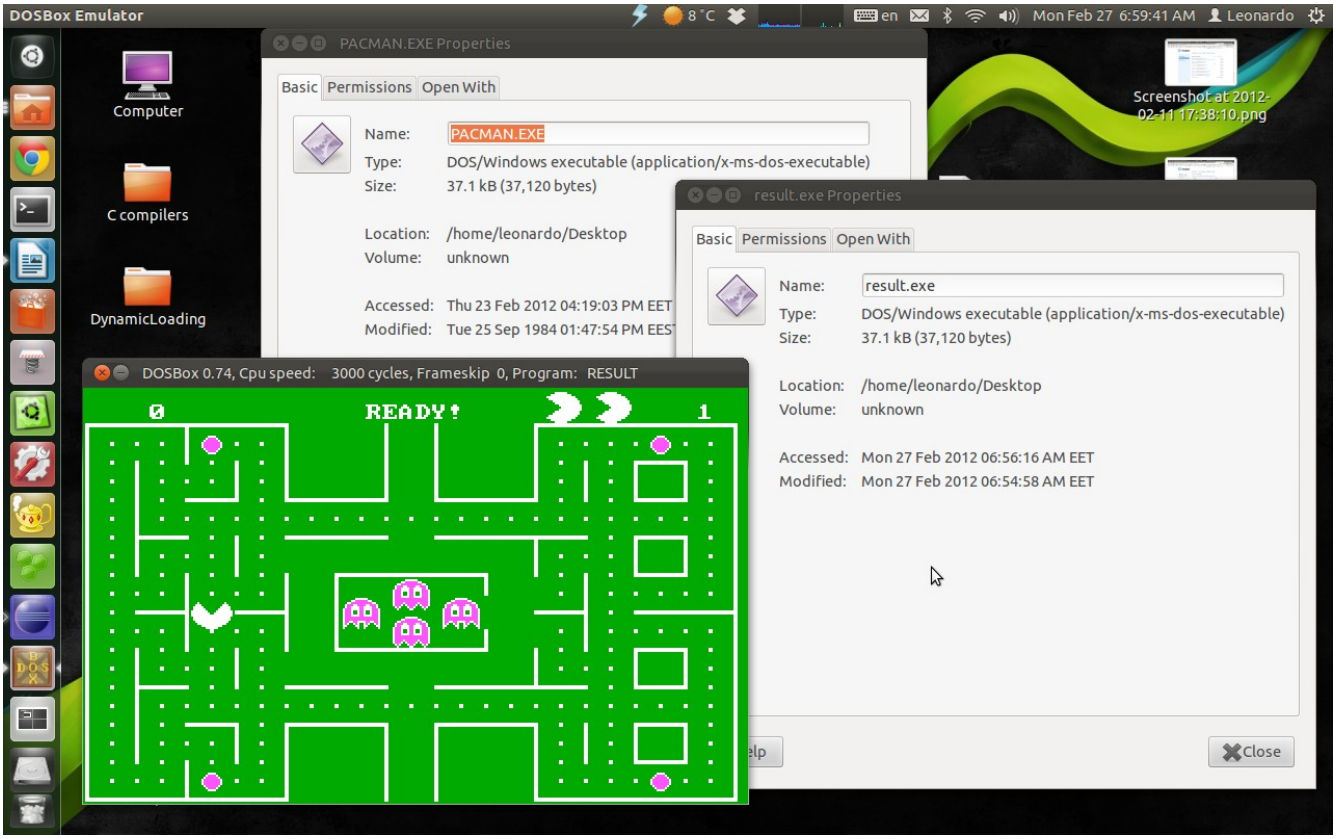
Video:


Home Folder
Mon Feb 27 6:37:20 AM
Leonardo
embers.mkv Properties
Basic
Permissions
Open With
Audio/Video
Name:
embers.mkv
Type:
Matroska video (video/x-matroska)
Size:
153.8 MB (153,846,571 bytes)
Location:
/home/leonardo/Desktop
Volume:
unknown
Accessed:
Sat 25 Feb 2012 12:27:40 AM EET
Modified:
Sun 05 Feb 2012 01:06:41 PM EET
Help
result.mkv Properties
Basic
Permissions
Open With
Audio/Video
Name:
result.mkv
Type:
Matroska video (video/x-matroska)
Size:
153.8 MB (153,846,571 bytes)
Location:
/home/leonardo/Desktop/Work…
Volume:
unknown
Accessed:
Mon 27 Feb 2012 06:36:47 AM EET
Modified:
Mon 27 Feb 2012 06:36:43 AM EET
Help
Close
eclipse
Screenshot at 2012-02-11 17:38:10.png

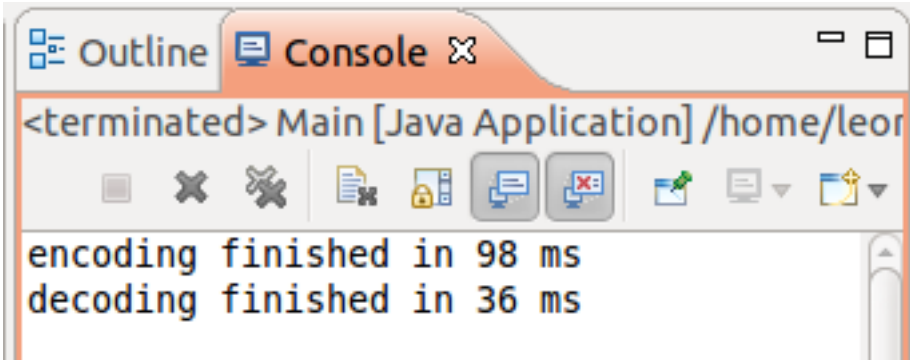
time


Outline
Console
<terminated> Main [Java Application] /home/leor
encoding finished in 11383 ms
decoding finished in 19393 ms

Executable:



time



Chosen Code Fragments :

MinHeap Class :

```
/**
	 * @param element
	 *            insert a new node then adjust the heap
	 */
	public void add(E element) {
		heap.add(element);
		int parent = (top - 1) / 2;
		int current = top;
		E temp;

		while (parent >= 0 && element.compareTo(heap.get(parent)) < 0) {
			temp = heap.get(current);
			heap.set(current, heap.get(parent));
			heap.set(parent, temp);
			current = parent;
			parent = (parent - 1) / 2;
		}
		top++;
	};


/**
	 * @param element
	 *            deletes the all the nodes containing the given element then
	 *            reheapify the heap
	 */
	public void remove(E element) {
		int i;

		while ((i = search(element)) >= 0) {
			heap.set(i, heap.get(--top));
			heap.remove(top);
			heapify(i);
		}

	}

/**
	 * @param i
	 *            heapify the sub tree whose root is at index i in the heap
	 */
	private void heapify(int i) {
		int minIndex;
		E temp;

		while ((i + 1) * 2 - 1 < top) {

			if ((i + 1) * 2 < top) {
				minIndex = heap.get((i + 1) * 2 - 1).compareTo(
						heap.get((i + 1) * 2)) < 0 ? (i + 1) * 2 - 1
						: (i + 1) * 2;

			} else if ((i + 1) * 2 - 1 < top) {
```

```
                        minIndex = (i + 1) * 2 - 1;
                  } else
                        break;
                  temp = heap.get(minIndex);
                  heap.set(minIndex, heap.get(i));
                  heap.set(i, temp);
                  i = minIndex;
            }
      }
```

Encoder Class

```
public void write() throws IOException {
            BufferedOutputStream bw = new BufferedOutputStream(
                        new FileOutputStream(Main.compressed));
            br = new BufferedInputStream(new FileInputStream(Main.source));


            ObjectOutputStream ob = new ObjectOutputStream(bw);
            ob.writeLong(bytesNo);
            ob.writeObject(freq);

            int readBytes;
            long code;
            int len;
            byte[] outBuffer = new byte[Main.bufferSize];
            byte currentByte = 0;
            int leftLen = 8;
            int count = 0;
            int i = 0;

            readBytes = br.read(buffer);
            code = codes.get(buffer[i]);
            len = codelen.get(buffer[i]);
            while (true) {
                        if (len <= leftLen) {

                              code = code << (64 - len) ;
                              code = code >>> (64 - len) ;

                              currentByte = (byte) (currentByte | code<<(leftLen -
len));
                              leftLen = leftLen - len;


                              i++;
                              if (i >= readBytes)     {
                                    readBytes = br.read(buffer);
                                    if (readBytes == -1)
                                          break;
                                    i = 0;
                              }
                              code = codes.get(buffer[i]);
                              len = codelen.get(buffer[i]);


                              if (leftLen == 0) {
```

```
                                    if (count >= buffer.length) {
                                          bw.write(outBuffer);
                                          count = 0;
                                    }

                                    outBuffer[count++] = currentByte;
                                    currentByte = 0;
                                    leftLen = 8;


                              }


                        } else {
                              currentByte = (byte) ((currentByte ) | (code >> (len
- leftLen)));
                              code = code << 64 - len + leftLen;
                              code = code >>> 64 - len + leftLen;
                              len = len - leftLen;
                              leftLen = 0;


                              if (count >= buffer.length) {
                                    bw.write(outBuffer);
                                    count = 0;
                              }
                              outBuffer[count++] = currentByte;

                              currentByte = 0;
                              leftLen = 8;
                        }

            }

            outBuffer[count++] = (byte)(currentByte<<(leftLen));
            bw.write(outBuffer, 0, count);


            ob.close();
            bw.close();
      }


      String bin(byte b){
            String s = "";
            for (int i = 0; i < 8; i++) {
                  if((b&(1<<7-i))==0)
                  s+=0;
                  else
                        s+=1;
            }
            return s;
      }
```

Decoder Class

```
@SuppressWarnings("unchecked")
      public void read() throws IOException, ClassNotFoundException {

            BufferedInputStream br = new BufferedInputStream(new FileInputStream(
                        Main.compressed));
            BufferedOutputStream bw = new BufferedOutputStream(
                        new FileOutputStream(Main.dest + Main.ext));
            ObjectInputStream ob = new ObjectInputStream(br);
            long bytesNo = ob.readLong();

            freq = (long[]) ob.readObject() ;
            calcFreq();

            byte buffer[] = new byte[Main.bufferSize];
            byte outBuffer[] = new byte[Main.bufferSize];
            int n = 0;
            int i = 0;
            int j = 0;
            int count = 0;
            byte current = 0;
            AbstNode<Long> node = tree;

            n = br.read(buffer);
            current = buffer[i++];

            while (true) {

                  if (node.isLeaf()) {
                        if(--bytesNo<0)break;
                        outBuffer[count++] = ((Leaf<Long, Byte>)
node).getElement();
                        node = tree;
                        if (count >= outBuffer.length) {
                              count = 0;
                              bw.write(outBuffer);
                        }

                  } else {
                        if ((current & (1 << (7 - j))) == 0) {
                              node = ((Node<Long>) node).getLeft();
                              j++;
                        } else {
                              node = ((Node<Long>) node).getRight();
                              j++;
                        }
                  }

                  if (j >= 8) {
                        current = buffer[i++];
                        j = 0;

                  }

                  if (i >= buffer.length) {
                        i = 0;
                        n = br.read(buffer);
```

```
                if (n == -1)
                        break;
                }

        }
        //should be 0 but i decrement in the if statement
        if (bytesNo !=-1)
                count+=bytesNo+1;

        bw.write(outBuffer, 0, (int)(count));

        bw.close();
        ob.close();
        br.close();

}


public void calcFreq() {
        for (int i = 0; i < freq.length; i++) {
                if (freq[i] == 0)
                        continue;
                pq.add(new Leaf<Long, Byte>(freq[i], (byte) i));
        }

        AbstNode<Long> temp1;
        AbstNode<Long> temp2;

        while (pq.size() != 1) {
                temp1 = pq.poll();
                temp2 = pq.poll();

                tree = new Node<Long>(temp1.getKey() + temp2.getKey());
                tree.setChildren(temp1, temp2);
                pq.add(tree);
        }
        decode(0, pq.poll(), 0);
}
```