14057 - Savev David

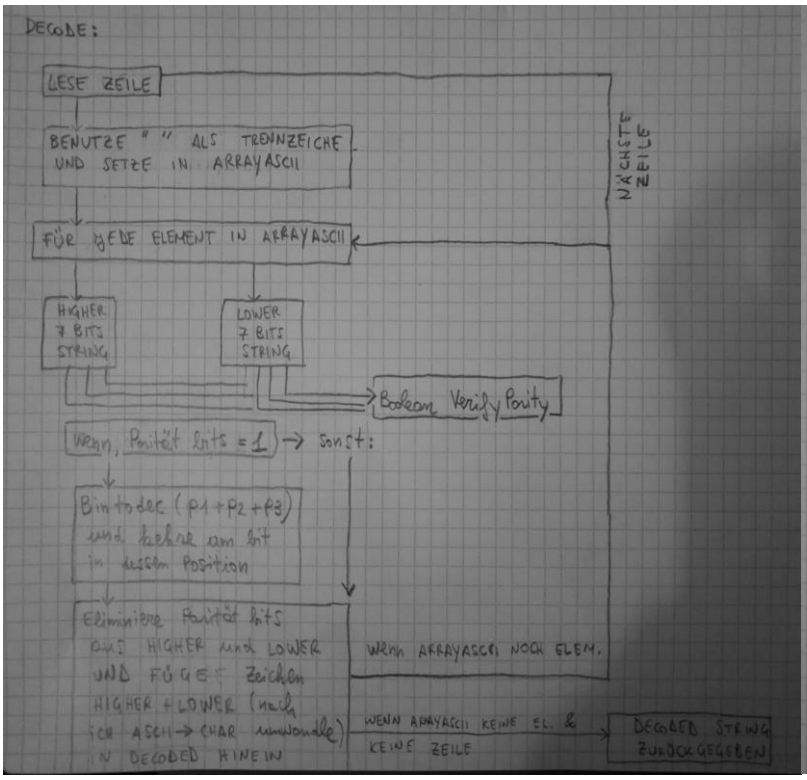
Aufgabe nr. 1 – VERTEILTE SYSTEME

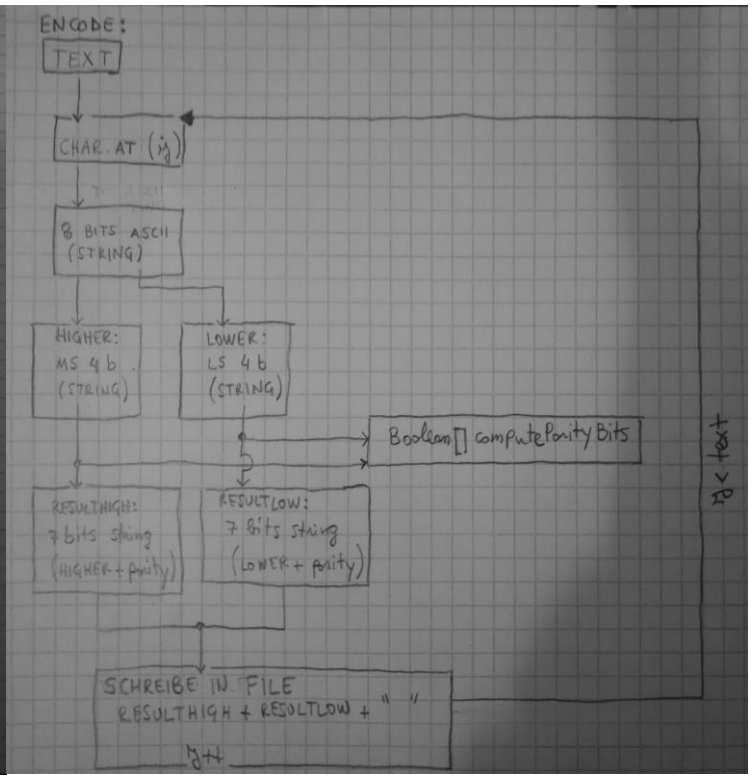
In meiner Lösung habe ich zwei statische Haupt-Methoden die sich **encode** und **decode** nennen implementiert; in meiner Lösung findet man auch andere 5 private Methoden (computeParityBits, verifyParity, boolToString, setCharAt, charToBoolean) die ich hauptsächlich für Berechnungen und Konversionen mehrmals benutze.
Man kann bemerken, dass ich viel String-Manipulationen benutzt habe; so kann man den Problem schneller lösen (z.B. es ist leicht Parität-bits im String konvertieren und in eine ASCII-String in die richtige Positionen hineinfügen).

Hauptsächlich, in **encode**, konvertiere ich jedes einzelne Zeichen vom Text das ich kodieren möchte in ASCII-Format (6 oder 7 bits) und ich füge „00“ als erste Zeichen wenn wir 6 bits haben oder „0“ wenn wir 7 bits haben → um 8 bits zu erhalten.
Dann, breche ich jedes ASCII-Format Zeichen in zwei Strings (low und high) mit Größe 4 bits; die werden an die Methode computeParityBits geschickt, die jedes Zeichen in die String in Boolean konvertiert (mit hilfe des charToBoolean Methode) und die XOR Berechnungen ausführt. Dann wird eine Booleanische Array mit dem Status von die 3 Parität bits züruckgegeben.
Die 3 Parität bits werden dann in Zeichen konvertiert (boolToString), mit die original String in die korrekte Positionen verketteten und in den File geschrieben.
Die Komplexität der encode Manipulation ist Linear (Θ = n).

In **decode**, wird jede Reihe aus den File gelesen, und dann jede einzelne kodierte ASCII String (14 Zeichen) wird in eine Array von Strings gefügt.
Dann für jede String in die Array, breche ich sie in zwei Teile (7 Zeichen) und mit der Methode checkParity verifiziere ich den Status von den 3 Parität bits und ich gebe eine Booleanische status zurück; hier, wenn ich mindestens eine von denen „TRUE“ ist, haben wir ein Fehler und ich konvertiere die 3 Status Sequenz in String und dann in Dezimal (ich benutze die Methoden boolToString und Integer.ParseInt) und mit Hilfe der Methode setCharAt benutze ich diese Dezimal Nummer um den bit in dessen Position zu umkehren (wenn „0“ → „1“, wenn „1“ → „0“).
Hier, nehme ich die substring von der bearbeiteten String (ohne Parität bits), verkette String high und String low, verwandle die ASCII-String in Char und ich füge den Zeichen ins decoded String; wenn wir jede Zeile gelesen haben, können wir decoded String zurückgeben.
Die Komplexität der decode Manipulation ist Θ = n^2.

Test.java enthält die Main Methode und ein Test ist schon implementiert.





Den Hamming(7,4) Kode kann ein Fehler von 1 bit in eine 7 bit bestehende Sequenz korrigieren; in unserem Beispiel besteht jedes Zeichen aus 14 Hamming-kodierte bits (und es bedeutet zwei verkettete Hamming-Kodes) und zwei Fehlern können korrigiert sein (natürlicher weiße eine in die 7 MSB, den andere bei die 7 LSB).