Hamming-code in Python 3.6

Robin van de Griend, Thomas Koopman, Tim Waroux

20 juni 2017

1 Taakverdeling

2 Ontwerp van de code

We hebben er voor gekozen om gebruik te maken van verschillende modules, we hebben voor belangrijke functies een apart bestand aangemaakt. Dit is beter voor de leesbaarheid en maakt het makkelijker om te werken in de code. Hieronder volgt een toelichting voor de verschillende modules.

2.1 matrix.py

Om te beginnen met het programmeren van de Hammingcode, hebben we ervoor gekozen om eerst een klasse Matrix te definieren. Dit staat eveneens aangegeven in de opdracht die we gevolgd hebben. Hier is al onze code opgebasseerd omdat we bij onze hammingcode gebruik maken van de matrixvermenigvuldiging.

2.2 strconv.py

Onze stringconvert bestand is van essentieel belang voor onze hammingcode. De functies die gedefinieerd staan in stringconvert zorgen ervoor dat een string, met name de teskt die mensen willen versturen of opslaan, omzet naar een lijst met matrices. Hierdoor kunnen we ze gebruiken in het Hamming bestand, waar onze Hammingcode wordt uitgevoerd.

2.3 hamming.py

In dit bestand staat ons algoritme voor de hammingcode. Om te beginnen staat er in de bestand het belangrijkste gedefinieerd van onze hele code. Namelijk de generator en pariteitcheck matrix. In dit bestand staan verder de functies waarmee we pariteit kunnen toevoegen aan bits, eventuele 1 bits fouten kunnen vinden en als deze er zijn herstellen, verder is er nog een functie waarmee we de pariteit van de bits kunnen verwijderen om later in interface.py de string van 1-en 0'en terug te vertalen naar iets wat wij kunnen lezen.

2.4 errormaker.py

Dit is een klein bestand waarin we 2 functies hebben gedefinieerd: Errormaker, de functie waarmee we random 1 bits fouten kunnen maken in een lijst met

vectoren waar pariteitbits aaantoegevoegd zijn, en Errormakerstring de functie waar we in een string van 1-en 0'en random 1 bits fouten kunnen maken.

2.5 interface.py

Dit is het bestand dat de gebruiker van ons programma moet opstarten. In dit bestand komen alle functies van de andere bestanden samen en worden ze samengebruikt om berichten te encoderen en decoderen, en bij het decoderen eventueel een aantal 1 bits fouten toe tevoegen om te laten zien dat het programma deze fouten kan herstellen.

3 Complexiteit

We drukken de complexiteit uit in de lengte van de boodschap n. Of dit in bits of in tekens is, maakt niet uit omdat elk teken 8 bits is, en in grote O notatie maakt dit niet uit.

Alle functies behalve encodeentiremessage, repairentiremessage, destroyallparitybits en decodeentiremessage nemen niet de boodschap als argument. Deze zijn dus O(1).

encodeentiremessage is O(n) want message is een lijst van n matrices (elk karakter is 1 matrix), en je loopt 1 keer door de lijst heen, en voert elke keer een constant aantal operaties uit. Een lege lijst new_message maken en dit returnen hangt niet van n af en is dus constant.

Het geval repairentiremessage is analoog.

Het geval destroyallparitybits is analoog.

4 Verbeterpunten en mogelijke uitbreidingen van de code

De Matrix klasse hebben is een zelfgeschreven library waarvan de interface nog verbetert zou kunnen worden. Zo komt het vaak voor in de code waar we lijst-operaties willen uitvoeren op vectoren, een matrix met 1 rij. Op het moment moet je dan de waarden van de matrix opvragen en opslaan als lijst om vervolgens weer opnieuw een matrix aan te moeten maken met de nieuwe lijst van waarden. Zoals de interface nu is geschreven ontstaan er situaties als deze:

```
code_pairs = [(codelist[i].values[0],
              codelist[i + 1].values[0])
              for i in range(0, len(codelist) - 1, 2)]
```

Dit is een stukje code uit `strconv.py` waar we een lijst van matrices `codelist` uit willen pakken en voor elke matrix een lijst-operatie willen uitvoeren. In feite zijn maken we hier nergens gebruik van de Matrix-klasse maar toch staan onze codes opgeslagen in een matrix omdat we matrix-vermenigvuldiging gebruiken in het hamming-algoritme.

Veel beter zou het zijn om een externe library te gebruiken als numpy, waar matrices en vectoren een intutieve interface hebben. Ook is de code sneller omdat veel rekenintensieve functies in C zijn gemplementeerd. Verder zou functionaliteit in `strconv.py` in de Matrix-klasse gebouwd kunnen worden. Het gaat

immers om het omzetten van matrices naar andere objecttypes of vice versa. Op die manier houden we functionaliteit die bij matrices hoort bij elkaar.

We zouden Hamming-codes algemener kunnen implementeren. Een Hamming-code wordt volledig vastgelegd door zijn controle-matrix, de genererende matrix kan immers hieruit bepaald worden. We zouden een code kunnen schrijven die voor algemene (n,k,d)-Hamming codes de $n \times d$ controle-matrix maakt en hieruit de genererende matrix vindt. Op deze manier hoeven we geen matrices expliciet te definiren en kunnen we ze äs needed"genereren.