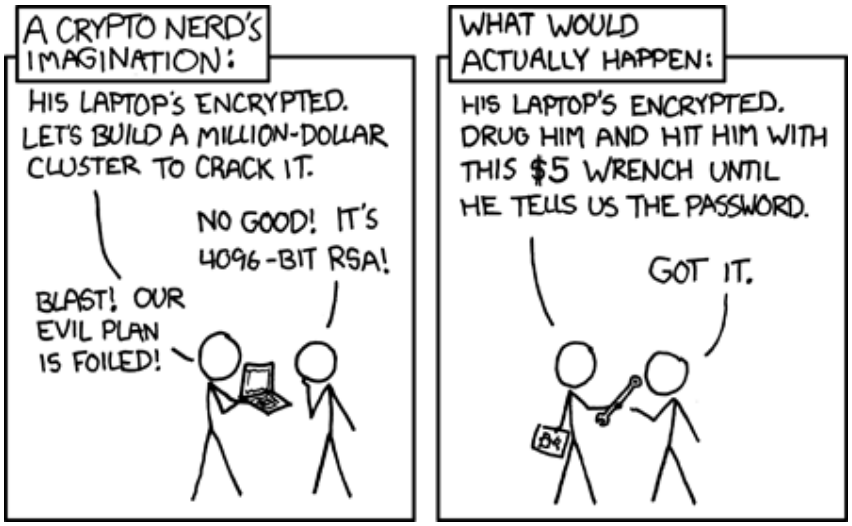
Universiteit van Amsterdam

Steven de Rooij, Danilo de Goede, Julian Verweij

Opdracht 3: Vigenère



1 Introductie

Het klassieke *Vigenère cijferschrift* (`https://en.wikipedia.org/wiki/Vigenere_cipher`) werd in de 16de eeuw bedacht door *Giovan Battista Bellaso*. Langer dan 300 jaar dacht men dat dit cijfer onbreekbaar was, totdat *Friedrich Kasiski* in de 19de eeuw een methode publiceerde om het cijferschrift te kraken. In deze opdracht zal je een dergelijke methode implementeren.

Het aangeleverde framework bevat het begin van een programma dat uiteindelijk verschillende operaties moet ondersteunen. We gaan in deze opdracht dat programma stap voor stap uitbreiden totdat het uiteindelijk berichten versleuteld met Vigenère volledig kan kraken.

2 Deel 1: Het Caesar cijferschrift

Voordat we aan Vigenère beginnen, kijken we eerst naar een eenvoudigere variant: het *Caesar cijferschrift*. Hierbij wordt een bericht versleuteld door alle karakters drie posities op te schuiven, waarbij we

van de ‘z’ weer terug gaan naar de ‘a’, alsof de letters in een cirkel aan elkaar zijn geplakt. Een ‘a’ wordt dus een ‘d’, een ‘b’ wordt een ‘e’, een ‘y’ wordt een ‘b’, etcetera.

a	b	c	d	e	f	g	h	i	j	k	l	m	...	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	...	z	a	b	c

Table 1: Versleuteling met het Caesar cijferschrift.

In het programma `vigenere.c` kun je in de `main` functie de vijf operaties zien die het programma uiteindelijk moet ondersteunen. Bekijk de functies `main` en `lees_bestand` even goed; de trucs die in die code worden gebruikt zul je tijdens dit practicum vaker kunnen toepassen. Maar het geeft niet als je nog niet van alles kunt zien hoe het precies werkt, want het omgaan met bestanden en `enum` en dergelijke zal tijdens latere colleges nog in meer detail worden behandeld.

Om te oefenen gaan we nu eerst het Caesar cijferschrift implementeren. Een sessie zou er zo uit kunnen zien:

```
% cat voorbeelden/bericht1.txt
Tussen droom en daad staan wetten in de weg, en praktische bezwaren.
% ./vigenere versleutel_caesar voorbeelden/bericht1.txt
wxvvhqgurrphqgddgvwddqzhwwhqlqghzhjhqsudnwlvfkhehczduhq
% ./vigenere versleutel_caesar voorbeelden/bericht1.txt > test.txt
% ./vigenere ontsleutel_caesar test.txt
tussendroomendaadstaanwettenindewegenpraktischebezwaren
```

Het versleutelen gaat als volgt in zijn werk:

1. De main functie ontcijfert welke operatie wordt gevraagd en roept de functie `opdracht_versleutel_caesar` aan.
2. Deze functie leest eerst het tekstbestand in met de functie `lees_en_versimpel`. Vervolgens wordt het bericht versleuteld en het resultaat afgedrukt met `printf`.
3. De functie `lees_en_versimpel` gebruikt eerst `lees_bestand` om het bericht in te lezen. Vervolgens moet het bericht worden opgeschoond en klaargemaakt om versleuteld te worden. Dat houdt in dat alle spaties en andere tekens uit het bericht verwijderd worden zodat alleen de letters overblijven, en van alle hoofdletters worden kleine letters gemaakt.

- **Opdracht (1pt).** Implementeer de functies `lees_en_versimpel`, `opdracht_versleutel_caesar` en `opdracht_ontsleutel_caesar`. Controleer dat je programma compileert en dat de functies goed werken.

Hieronder een aantal hints:

- In C is `char` (gek genoeg eigenlijk) een numeriek type. Je kunt letters dus verschuiven door er simpelweg een getal bij op te tellen. Je moet natuurlijk wel opletten dat het goed gaat als je voorbij de ‘z’ of voor de ‘a’ uitkomt. Je kunt hiervoor eventueel de modulo operatie `%` gebruiken, maar het gaat ook met `if`-statements.
- Je moet in deze opdracht werken met arrays. Arrays worden in C nooit zomaar gekopieerd naar een andere plaats in het geheugen. Dat heeft als gevolg dat een functie de meegegeven array altijd kan wijzigen. Daarom kunnen in functies arrays dus zowel als input en als output worden gebruikt. Het gebruik van `const` maakt code niet alleen veiliger, het is ook een hint die aangeeft of het de bedoeling is dat de functie dat array wijzigt.
- De arrays in deze opdracht bevatten strings. Een string is een array van `char` met een `'\0'` erin die het einde aanduidt. Denk erom dat je die laatste `'\0'` nergens vergeet, want dat lijdt tot vervelende errors!

3 Deel 2: Het Vigenère cijferschrift

In het Vigenère cijferschrift worden net zoals bij het Caesar schrift alle letters een stukje doorgeschoven, maar niet alle letters gaan nu even ver. Er wordt gebruik gemaakt van een zogenaamde *sleutel* die bepaalt hoe ver elke letter wordt doorgeschoven. De letters van de sleutel worden een voor een gebruikt om te bepalen hoe ver een letter uit het oorspronkelijke bericht moet worden verschoven, waarbij een 'a' in de sleutel staat voor geen verschuiving, 'b' voor +1, 'c' voor +2, enzovoorts. Zo krijg je een soort optelsom van letters.

Bijvoorbeeld, stel we gebruiken als sleutel `elsschot`. Dan wordt `voorbeelden/bericht1.txt` als volgt versleuteld:

```
tussendroomendaadstaanwettenindewegenpraktischebezwaren
elsschotelsschotelsschotelsschotelsschotelsschotelsscho
------------------------------------------------------- +
xfkkgurkszewpkothdlscukxxewfkurxapywpwftoeakeosuikostlb
```

Je ziet dat de verschuiving van de eerste letter `t` wordt bepaald door de eerste letter van de sleutel `e`, wat een verschuiving van vier stappen betekent. Vier letters verder in het alfabet vanaf de `t` vinden we de `x`, wat dus de eerste letter van de cijfertekst is. Voor de tweede letter wordt `l` bij `u` opgeteld, een verschuiving van 11 stappen. Daarbij komen we voorbij de `z`, maar na doordraaien is het resultaat een `f`. Enzovoort.

De bedoeling is dat je het programma zodadelijk ongeveer hetzelfde kunt gebruiken voor Vigenère als eerder voor Caesar. Een sessie zou er dan zo uit kunnen zien:

```
% ./vigenere versleutel voorbeelden/bericht1.txt elsschot
xfkkgurkszewpkothdlscukxxewfkurxapywpwftoeakeosuikostlb
% cat voorbeelden/cijfertekst1.txt
xfkkgurkszewpkothdlscukxxewfkurxapywpwftoeakeosuikostlb
% ./vigenere ontsleutel voorbeelden/cijfertekst1.txt elsschot
tussendroomendaadstaanwettenindewegenpraktischebezwaren
```

Je kunt ook controleren dat `vigenere versleutel <bericht> d` hetzelfde doet als `vigenere versleutel_caesar <bericht>`.

Anders dan hiervoor splitsen we nu de versleutel- en ontsleutelfuncties allebei in tweeën. De functie `main` roept bijvoorbeeld `opdracht_versleutel` aan. Die maakt gebruik van een hulpfunctie `versleutel` om het daadwerkelijke werk te verrichten, en drukt daarna het gewenste resultaat af voor de gebruiker.

- **Opdracht (0.5pt)** Implementeer de functies `versleutel` en `ontsleutel`. Implementeer ook de functies `opdracht_versleutel` en `opdracht_ontsleutel`.

Hints:

- Denk weer om de afsluitende `'\0'`!
- Gebruik voor de functie `versimpel` eventueel de functies `isalpha` en `tolower` uit de standaard library. (Je kunt in je terminal `man isalpha` en `man tolower` intypen om meer informatie te krijgen over deze functies, en ook om te zien welke headers je met `#include` moet toevoegen als je deze functies wilt gebruiken.)

4 Deel 3: De sleutel kraken.

Nu ontwikkelen we stap voor stap een methode om, gegeven alleen de cijfertekst, de sleutel te achterhalen. We doen dit door telkens met een kandidaatsleutel de cijfertekst te proberen te ontcijferen (met de functie `ontsleutel` die je eerder hebt gemaakt). We kijken dan naar de frequenties van de letters in het resultaat. Als die frequenties *lijken* op de frequenties in een stuk normale Nederlandse tekst, dan is er een goede kans dat de kandidaatsleutel lijkt op de correcte sleutel. We zoeken vervolgens een voor een de letters van de kandidaatsleutel die het beste werken. Om te weten wat de frequenties zijn in (min of meer) normale Nederlandse tekst, zullen we het bestand `voorbeeld/bint.txt` gebruiken. Dit is een fragment

uit de novelle *Bint* van Bordewijk; ons programma zal eenvoudigweg tellen hoe vaak alle letters daarin voorkomen.

- **Opdracht (0.5pt).** Implementeer eerst de functie `bepaal_frequenties` waarmee je de frequenties van de letters in een string kunt meten. De frequentie van een letter is het aantal keer dat de letter voorkomt in de string, gedeeld door het totaal aantal letters (dus de som van alle frequenties moet gelijk zijn aan 1).
- **Opdracht (0.5pt).** Maak nu de functie `opdracht_frequenties` om je functie te testen. Deze laatste functie leest een opgegeven bestand in,versimpelt het, telt de frequenties, en drukt vervolgens het resultaat af zoals in het onderstaande voorbeeld, met een precisie van drie cijfers achter de komma. (Gebruik `man printf` om te zien hoe je getallen afgerond kunt afdrukken). Doe het volgende experiment om te zien dat het precies goed werkt:

```
./vigenere frequenties voorbeelden/bint.txt
a: 0.072
b: 0.014
c: 0.014
d: 0.055
e: 0.187
...
y: 0.000
z: 0.017
```

- **Opdracht (0.5pt).** Implementeer nu de functie `gelijkenis`, die de gelijkenis meet tussen twee arrays met frequenties. Als we de frequenties in de eerste array $p_1, p_2, \ldots, p_{26}$ noemen, en in de tweede array $q_1, q_2, \ldots, q_{26}$, dan is een redelijke maat voor de gelijkenis $p_1q_1 + p_2q_2 + \ldots + p_{26}q_{26}$.
- **Opdracht (0.5pt).** Implementeer om je functie te testen de functie `opdracht_vergelijk`, die twee bestanden inleest, van allebei de frequenties telt, en daarna de gelijkenis tussen de twee frequentiearrays afdrukt, opnieuw tot drie cijfers achter de komma. Controleer dat de uitvoer van je programma overeenkomt met het onderstaande voorbeeld.

```
% ./vigenere vergelijk voorbeelden/bericht1.txt voorbeelden/bint.txt
0.078
% ./vigenere vergelijk voorbeelden/cijfertekst1.txt voorbeelden/bint.txt
0.042
```

(Je kunt nu dus zien dat de gelijkenis tussen twee Nederlandse tekstbestanden veel hoger is dan de gelijkenis tussen een cijfertekst en een stuk Nederlandse tekst, zoals je zou verwachten.)

Nu zijn we klaar om de sleutel te gaan zoeken. We initialiseren de sleutel eerst als een reeks a's. We gaan dan een voor een alle letters optimaliseren. We optimaliseren een letter door op die positie een voor een alle mogelijke letters uit te proberen, het bericht opnieuw te ontsleutelen met de nieuw gevormde kandidaatsleutel, en dan te kijken of de gelijkenis met de frequenties uit de Nederlandse taal is verbeterd.

- **Opdracht (1pt).** Implementeer de functie `zoek_sleutel` die de beste sleutel zoekt voor een gegeven cijfertekst, gegeven de gewenste sleutellengte en de frequenties van de taal van het oorspronkelijke bericht.
- **Opdracht (0.5pt).** Test de bovenstaande functie en voltooi de opdracht door de functie `opdracht_kraak` te schrijven. Deze functie zoekt de beste sleutel voor alle mogelijke sleutellengtes van 1 tot en met 16, en drukt de resultaten af zoals in de onderstaande voorbeelduitvoer.

```
% ../vigenere kraak b2_vigenere.txt bint.txt
Sleutel e: score=0.049
Sleutel ee: score=0.049
Sleutel see: score=0.049
Sleutel eeee: score=0.049
Sleutel eeree: score=0.049
Sleutel jeeere: score=0.050
```

```
Sleutel finkers: score=0.077
Sleutel inesjsbe: score=0.052
Sleutel eseneeoes: score=0.052
Sleutel enrebfsrja: score=0.052
Sleutel rebbfeefffe: score=0.052
Sleutel jnefessesnre: score=0.053
Sleutel eeenasrreswrn: score=0.055
Sleutel finkersfinkers: score=0.077
Sleutel ffsebskrxneefre: score=0.054
Sleutel enetjnoeiwrseexr: score=0.055
sleutel: finkers
bericht: debrandstoftankdieverlorenisdooreenamerikaansestraaljageristerechtgek
omeninhetdrentseklazienaveentweevrouwenmoestennaardedokteropwegdaarnaartoekreg
enzijdebrandstoftankophunhoofddaaropmoestenzijnaardedokterdebetreffendeartsist
erverantwoordinggeroependevrouwentroffenderhalvedeartsnietthuisenmoestenonverr
ichterzakeweernaarhuisterugkerenopwegnaarhuiskregenzijeentweedebrandstoftankop
hunhoofddaaropmoestenzijnaardedokterdepilootisterverantwoordinggeroepenopwegna
arzijnsuperieurenverloorhijtweebrandstoftanksinhetdrentseklazienaveenerwerdnie
mandgewondaanvankelijkwasernogsprakevantweevrouwendienaardedoktermoestenmaardo
oreenonsnietbekendeoorzaakisdieafspraakverzet
```

Test je programma uit op de overige cijfertekstbestanden in de directory `voorbeelden`, of eventueel op berichten die je eerst zelf versleutelt.

Noot: als je het leuk vindt je programma nog verder te verfijnen, kun je gebruik maken van de frequenties van *bigrammen* in plaats van de frequenties van enkele letters. Een bigram is twee opeenvolgende letters, dus bijvoorbeeld de string `hallo` bevat de bigrammen `ha`, `al`, `ll` en `lo`. Als de bigramfrequenties van het ontsleutelde bericht goed overeenkomen met de bigramfrequenties van de taal, dan is dat een nog betere aanwijzing dat je met het ontsleutelen op de juiste weg bent dan als je alleen kijkt naar de frequenties van enkele letters. Daardoor zijn nog kortere berichten te ontcijferen.

5 Inleveren

Gebruik het `make tarball` commando en lever deze tarball in.