Домашна работа № 2 по Функционално програмиране
Специалност Информационни системи, 1-ви курс
2017/2018 учебна година

Заданието за домашното включва пет задачи, последната, от които не е задължителна (но е най-интересна и ви позволява да направите сравнително добра, макар и малко опростена емулация на машината Енигма).

Можете (препоръчително е) да използвате функциите за работа със знакове от модула Data.Char, както и базовите функции за работа със списъци от стандартната прелюдия (Prelude).

ВАЖНО:

Крайният срок за предаване на домашните работи е ***18.04.2018 г.*** (сряда).

Решенията ви трябва да са готови за компилиране и автоматично тестване. Важно е писмените работи да бъдат добре форматирани и да съдържат коментари на ключовите места.

Предайте решенията на четирите задачи в един файл с наименование **hw2_<FN>.hs**, където **<FN>** е Вашият факултетен номер.

Приятна работа и успех!

Задача 1. Нормализация на входните данни

Енигма, както повечето криптиращи машини от това време, е разполагала с клавиатура със само 26-те главни букви от латинската азбука. Затова, преди да бъдат криптирани, всички съобщения трябвало да бъдат приведени в т. нар. нормален вид: всички числени стойности бивали изписвани словом, всички малки букви ставали главни, а интервалите и пунктуационните знакове били премахвани или заменяни с кодови комбинации от главни букви (напр. интервалът бил заменян с X и т. н.)

Напишете функция **`normalize message`**, която нормализира входното съобщение. Правилата за нормализация са следните:

- Всички малки букви стават главни.
- Ако съобщението съдържа цифри, функцията връща грешка.
- Всички останали знакове се игнорират.

Примери:

```
normalize "Attack London tomorrow at ten a.m." = "ATTACKLONDONTOMORROWATTENAM"
normalize "Attack London tomorrow at 10 a.m." = error “digits not allowed”
```

Задача 2. Цезаров шифър

Цезаровият шифър е един от най-простите и най-стари методи за криптиране на съобщения. Първото му известно използване е от Юлий Цезар по време на кампаниите му в Галия, откъдето идва и неговото име. Идеята на Цезаровия шифър е проста: вземаме съобщението, което искаме да шифроваме, и заместваме всяка от буквите в него с буквата, отместена с определен брой позиции в азбуката. Например, ако отместването е 3, то тогава ‘A’ -> ‘D’, ‘B’ -> ‘E’, ‘C’ -> ‘F,’ … , ‘X’ -> ‘A’, ‘Y’ -> ‘B’, ‘Z’ -> ‘C’.

а). Напишете функция **`encode alphabet ch offset`**, която приема списък от знакове alphabet, знак ch и отместване offset и връща знака от alphabet, отместен на offset от ch (по модул дължината на списъка). Функцията encode трябва да работи както с положително, така и с отрицателно отместване и да връща грешка, ако ch не е елемент на alphabet.

N.B. Не е задължително буквите в alphabet да са подредени от ‘A’ до ‘Z’, т.е. **НЕ** може да разчитате на функциите **ord** и **chr!**

Примери:
```
encode ['A'..'Z'] 'A' 1 = 'B'
encode ['A'..'Z'] 'C' 2 = 'E'
encode ['A'..'Z'] 'Z' 3 = 'C'
encode ['A'..'Z'] 'A' (-1) = 'Z'
encode ['A'..'Z'] 'C' (-2) = 'A'
encode ['A'..'Z'] 'Z' (-3) = 'W'
encode ['A'..'Z'] '@' 1 = error “unsupported symbol: @”
```

б). Напишете функция **`encrypt alphabet offset normalized`**, която приема азбука alphabet, отместване offset и съобщение ***в нормализиран вид*** и връща съобщението, криптирано със съответното отместване.

Пример:
```
encrypt ['A'..'Z'] 5 "ATTACKLONDONTOMORROWATTENAM" = "FYYFHPQTSITSYTRTWWTBFYYJSFR"
```

в). Напишете функция **`decrypt alphabet offset ecrypted`**, която приема отместване offset и съобщение, криптирано с това отместване, и връща оригиналното съобщение ***в нормализиран вид***. Можете да използвате факта, че декриптирането на Цезаров шифър с отместване offset е еквивалентно на криптиране с отместване -offset.

Пример:
```
decrypt ['A'..'Z'] 5 "FYYFHPQTSITSYTRTWWTBFYYJSFR" = "ATTACKLONDONTOMORROWATTENAM"
```

Задача 3. Атака на Цезаровия шифър

Една от основните слабости на Цезаровия шифър се състои в това, че броят на възможните шифри е ограничен до броя на ротациите на буквите в азбуката минус едно. Това прави Цезаровия шифър податлив на т. нар. brute force атака, т. е. атака, която генерира всички възможни дешифровки на кодираното съобщение.

а). Напишете функцията **crackall alphabet encrypted**, която връща списък от всички възможни дешифровки на кодираното съобщение enccrypted.

Пример:

```
crackall ['A'..'Z'] "FYYFHPQTSITSYTRTWWTBFYYJSFR" =
["EXXEGOPSRHSRXSQSVVSAEXXIREQ","DWWDFNORQGRQWRPRUURZDWWHQDP","CVVCEMNQPFQPVQOQTTQY
CVVGPCO","BUUBDLMPOEPOUPNPSSPXBUUFOBN","ATTACKLONDONTOMORROWATTENAM","ZSSZBJKNMCNM
SNLNQQNVZSSDMZL","YRRYAIJMLBMLRMKMPPMUYRRCLYK","XQQXZHILKALKQLJLOOLTXQQBKXJ","WPPW
YGHKJZKJPKIKNNKSWPPAJWI","VOOVXFGJIYJIOJHJMMJRVOOZIVH","UNNUWEFIHXIHNIGILLIQUNNYHU
G","TMMTVDEHGWHGMHFHKKHPTMMXGTF","SLLSUCDGFVGFLGEGJJGOSLLWFSE","RKKRTBCFEUFEKFDFII
FNRKKVERD","QJJQSABEDTEDJECEHHEMQJJUDQC","PIIPRZADCSDCIDBDGGDLPIITCPB","OHHOQYZCBR
CBHCACFFCKOHHSBOA","NGGNPXYBAQBAGBZBEEBJNGGRANZ","MFFMOWXAZPAZFAYADDAIMFFQZMY","LE
ELNVWZYOZYEZXZCCZHLEEPYLX","KDDKMUVYXNYXDYWYBBYGKDDOXKW","JCCJLTUXWMXWCXVXAAXFJCCN
WJV","IBBIKSTWVLWVBWUWZZWEIBBMVIU","HAAHJRSVUKVUAVTVYYVDHAALUHT","GZZGIQRUTJUTZUSU
XXUCGZZKTGS"]
```

б). След като сме генерирали всички възможни дешифровки, бихме могли лесно да намерим най-вероятните от тях, използвайки факта, че някои кратки думи, напр. **the, at, on**, се срещат много често в английския език.

За тази цел най-напред напишете функция **substring sub str**, която проверява дали поднизът sub се среща в низа str.

Пример:

```
substring "Haskell" "Haskell Curry" = True
substring "Curry"   "Haskell Curry" = True
substring "Turing"  "Haskell Curry" = False
```

в). Използвайте функциите от предишните две подточки, за да напишете функцията **crackcandidates alphabet commonwords encrypted**, която приема списък с често срещани думи и криптирано съобщение и връща списък с потенициални вероятни разшифровки.

Една разшифровка се смята за вероятна, ако съдържа поне една от думите от списъка с често срещани думи.

Пример:

```
crackcandidates ['A'..'Z'] ["THE","AND","AT","ON","IS"]
            "FYYFHPQTSITSYTRTWWTBFYYJSFR" = ["ATTACKLONDONTOMORROWATTENAM"]
```

Задача 4: Polysubstitution cypher (шифър с множествено заместване)

Един от простите начини да се справим със слабостта на Цезаровия шифър е да разбием съобщението на блокове от по няколко знака и да криптираме всеки от тях с различен Цезаров шифър, отместен с определена стъпка спрямо предишния.

а). Напишете функция **`polyencrypt alphabet offset step blockSize normalized`**, която приема азбука alphabet, първоначално отместване offset, стъпка step и размер на блока blockSize, както и съобщение в нормализиран вид, и връща криптирано съобщение, първите blockSize знака на което са криптират с отместване offset, следващите blockSize знака - с отместване offset + step, и т. н.

Пример:
```
polyencrypt ['A'..'Z'] 5 1 7 "ATTACKLONDONTOMORROWATTENAM" =
      "FYYFHPQUTJUTZUTVYYVDHBBMVIU"
```

б). Напишете функция **`polydecrypt alphabet offset step blockSize encrypted`**, която декриптира съобщението от предишната подточка.

Пример:
```
polydecrypt ['A'..'Z'] 5 1 7 "FYYFHPQUTJUTZUTVYYVDHBBMVIU" =
      "ATTACKLONDONTOMORROWATTENAM"
```

Задача 5: Емулация на Енигма

Един от основните компоненти на Енигма е система от ротори, всеки от които може да се моделира като polysubstitution cypher от предната задача. Резултатът от всеки от роторите се предава като вход на следващия. Резултатът от последния ротор е криптираното съобщение.

а). Напишете функция **`enigmaencrypt alphabet rotors normalized`**, която приема азбука alphabet, списък от ротори (offset, step, blockSize) и съобщение в нормализиран вид и връща криптираното от роторите съобщение.

Пример:
```
enigmaencrypt ['A'..'Z'] [(5,1,1),(7,2,10),(13,3,25)]
             "ATTACKLONDONTOMORROWATTENAM" = "ZTUCFOQUULZZGCBEIJHQXRSEOFS"
```

б). Напишете функция **`enigmadecrypt alphabet rotors normalized`**, която приема азбука, списък от ротори и криптирано съобщение и връща оригиналното съобщение.

Пример:
```
enigmadecrypt ['A'..'Z'] [(5,1,1),(7,2,10),(13,3,25)]
             "ZTUCFOQUULZZGCBEIJHQXRSEOFS" = "ATTACKLONDONTOMORROWATTENAM"
```