Домашна работа № 4 по Функционално програмиране
специалност „Компютърни науки“, II курс, II поток
2022/2023 учебна година

Решенията трябва да са готови за автоматично тестване. Важно е програмният код да бъде добре форматиран и да съдържа коментари на ключовите места. Предайте решенията си в един архив с наименование hw4_<FN>.zip, където <FN> е Вашият факултетен номер, който включва

- *файл с наименование hw4_<FN>.hs, съдържащ решенията на двете задачи;*
- *файловете с тестови случаи sample.txt и input.txt.*

Домашните работи се предават като изпълнение на съответното задание в курса по ФП в Moodle (https://learn.fmi.uni-sofia.bg/course/view.php?id=8504) най-късно до 23:55 ч. на 20.01.2023 г. (петък).

Приятна работа и успех!

Задача 1 (продължение на втората задача от третото домашно).

Докато прибирали шейната след тазгодишния полет, елфите на Дядо Коледа забелязали, че четирицифрените седемсегментни дисплеи не работят добре; вероятно са се повредили от високите температура по време на полета. Те помолили доброволци от целия свят за помощ - без тези дисплеи Дядо Коледа няма да може да разбере дали се движи навреме и дали не са се появили други проблеми с шейната по време на следващия полет.

Всяка цифра на седемсегментен дисплей се изобразява чрез включване или изключване на кой да е от седемте сегмента от `a` до `g`:

```
  0:          1:          2:          3:          4:
 aaaa        ....        aaaa        aaaa        ....
b    c      .    c      .    c      .    c      b    c
b    c      .    c      .    c      .    c      b    c
 ....        ....        dddd        dddd        dddd
e    f      .    f      e    .      .    f      .    f
e    f      .    f      e    .      .    f      .    f
 gggg        ....        gggg        gggg        ....

  5:          6:          7:          8:          9:
 aaaa        aaaa        aaaa        aaaa        aaaa
b    .      b    .      .    c      c    b      b    c
b    .      b    .      .    c      c    b      b    c
 dddd        dddd        ....        dddd        dddd
.    f      e    f      .    f      e    f      .    f
.    f      e    f      .    f      e    f      .    f
 gggg        gggg        ....        gggg        gggg
```

Следователно, за да се изобрази `1`, ще бъдат включени само сегменти `c` и `f`; останалото ще бъде изключено. За изобразяване на `7` ще бъдат включени само сегменти `a`, `c` и `f`.

Проблемът е, че сигналите, които управляват сегментите, са се разместили за всеки дисплей. Главният компютър на шейната все още се опитва да показва числа, като произвежда изходни сигнали на сигнални проводници от `a` до `g`, но тези проводници са свързани към сегменти **произволно**. Още по-лошо: проводниците/сегментните връзки се смесват отделно за всеки четирицифрен дисплей! (Всички цифри **в конкретен дисплей** обаче използват едни и същи връзки.)

С други думи, от това, че само сигналните проводници `b` и `g` са включени, не следва, че **сегментите** `b` и `g` са включени. Единствената цифра, която използва два сегмента, е `1`, следователно това, че само сигналните проводници `b` и `g` са включени, означава, че сегменти `c` и `f` са включени. Обаче само с тази информация все още не е ясно кой проводник (`b`/`g`) отива към кой сегмент (`c`/`f`). За целта ще трябва да съберете повече информация.

За всеки дисплей помощникът наблюдава променящите се сигнали за известно време, отбелязва **всичките десет уникални модела на сигнала**, които вижда, и след това записва една единствена **четирицифрена изходна стойност**. Използвайки моделите на сигнала, трябва да може да се определи кой модел на коя цифра съответства.

Например, ето какво може да се види в един запис в бележките на доброволен помощник :

```
acedgfb cdfbe gcdfa fbcad dab cefabd cdfgeb eafb cagedb ab | cdfeb fcadb
cdfeb cdbaf
```

(Записът тук е разположен на два реда; в действителните бележки той ще бъде на един ред.)

Всеки запис се състои от десет **уникални модела на сигнал**, разделител (`|`) и накрая **четирицифрената изходна стойност**. В рамките на един запис се използват същите проводници/сегментни връзки (но не знаете как те са свързани). Моделите на сигнала съответстват на десетте различни начина, по които компютърът на шейната се опитва да изобрази цифра, използвайки текущите кабелни/сегментни връзки. Тъй като `7` е единствената цифра, която използва три сегмента, **`dab`** в горния пример означава, че за изобразяване на `7` сигналните линии `d`, `a` и `b` са включени. Тъй като `4` е единствената цифра, която използва четири сегмента, **`eafb`** означава, че за изобразяване на `4` сигналните линии `e`, `a`, `f` и `b` са включени.

Използвайки тази информация, трябва да се определи коя комбинация от сигнални проводници съответства на всяка от десетте цифри. След това може да се декодира четирицифрената изходна стойност.

Нека разгледаме отново познатия запис:

```
acedgfb cdfbe gcdfa fbcad dab cefabd cdfgeb eafb cagedb ab |
cdfeb fcadb cdfeb cdbaf
```

След внимателен анализ става ясно, че връзките между сигналните проводници и сегментите имат смисъл само в следната конфигурация:

```
 dddd
e    a
e    a
 ffff
g    b
g    b
 cccc
```

Уникалните модели на сигнала ще съответстват на следните цифри:

- `acedgfb: 8`
- `cdfbe: 5`
- `gcdfa: 2`
- `fbcad: 3`
- `dab: 7`
- `cefabd: 9`
- `cdfgeb: 6`
- `eafb: 4`
- `cagedb: 0`
- `ab: 1`

След това четирите цифри на изходната стойност могат да бъдат декодирани:

- `cdfeb: 5`
- `fcadb: 3`
- `cdfeb: 5`
- `cdbaf: 3`

Следователно изходната стойност за този запис е **`5353`**.

Следвайки същия процес за записите в следващия по-обемен пример:

```
be cfbegad cbdgef fgaecd cgeb fdcge agebfd fecdb fabcd edb |
fdgacbe cefdb cefbgd gcbe
edbfga begcd cbg gc gcadebf fbgde acbgfd abcde gfcbed gfec |
fcgedb cgb dgebacf gc
fgaebd cg bdaec gdafb agbcfd gdcbef bgcad gfac gcb cdgabef |
cg cg fdcagb cbg
fbegcd cbd adcefb dageb afcb bc aefdc ecdab fgdeca fcdbega |
efabcd cedba gadfec cb
aecbfdg fbg gf bafeg dbefa fcge gcbea fcaegb dgceab fcbdga |
gecf egdcabf bgf bfgea
fgeab ca afcebg bdacfeg cfaedg gcfdb baec bfadeg bafgc acf |
gebdcfa ecba ca fadegcb
dbcfg fgd bdegcaf fgec aegbdf ecdfab fbedc dacgb gdcebf gf |
cefg dcbef fcge gbcadfe
bdfegc cbegaf gecbf dfcage bdacg ed bedf ced adcbefg gebcd |
ed bcgafe cdgba cbgef
egadfb cdbfeg cegd fecab cgb gbdefca cg fgcdab egfdb bfceg |
gbdfcae bgc cg cgb
gcafb gcf dcaebfg ecagb gf abcdeg gaef cafbge fdbac fegbdc |
fgae cfgab fg bagce
```

изходната стойност на всеки запис може да бъде определена както следва:

- `fdgacbe cefdb cefbgd gcbe: 8394`
- `fcgedb cgb dgebacf gc: 9781`
- `cg cg fdcagb cbg: 1197`
- `efabcd cedba gadfec cb: 9361`
- `gecf egdcabf bgf bfgea: 4873`
- `gebdcfa ecba ca fadegcb: 8418`
- `cefg dcbef fcge gbcadfe: 4548`
- `ed bcgafe cdgba cbgef: 1625`
- `gbdfcae bgc cg cgb: 8717`
- `fgae cfgab fg bagce: 4315`

Сумата на всички изходни стойности в този по-голям пример е **`61229`**.

Доброволците помогнали на елфите да сбъднат желанията на децата през следващата Коледна нощ! За всеки запис определили всички проводници/сегментни връзки и декодирали четирицифрените изходни стойности.

Да се дефинира функция **`decode :: String -> Int`**, която получава последователност от записи и извежда сбора на всички изходни стойности.

Примери:

```
sample <- readFile "sample.txt"

input <- readFile "input.txt"

countUniques sample → 61229

countUniques input  → 983026
```

Задача 2.

Дадено е съобщение в шестнадесетично представяне. Първата стъпка от декодирането на съобщението е преобразуването на шестнадесетичното представяне в двоично. Всеки шестнадесетичен знак съответства на четири бита двоични данни:

```
0 = 0000   6 = 0110   B = 1011
1 = 0001   7 = 0111   C = 1100
2 = 0010   8 = 1000   D = 1101
3 = 0011   9 = 1001   E = 1110
4 = 0100   A = 1010   F = 1111
5 = 0101
```

Резултатът от т. нар. BITS предаване е един пакет, който може да съдържа други пакети. В шестнадесетичното представяне на този пакет може да присъстват няколко допълнителни бита със стойност `0` в края; те не са част от предаването и трябва да се игнорират.

Всеки пакет започва със стандартно заглавие: първите три бита кодират **версията** на пакета, а следващите три бита кодират **типа** пакет. Тези две стойности (версията и типът) са

неотрицателни цели числа; всички числа, кодирани във всеки пакет, се представят като двоични (с най-значимия бит на първа позиция). Например, версия, кодирана като двоичната последователност `100`, представя числото `4`.

Пакетите от тип `4` представляват **пакети-литерали**. Всеки пакет-литерал кодира едно двоично число. Това става, като двоичното число се допълва с водещи нули, докато дължината му стане кратна на `4`, след което се разделя на групи от по четири бита. Всяка група започва с бит със стойност `1`. Изключение е последната група, която започва с `0`. Тези групи от по `5` бита следват непосредствено заглавието на пакета. Например, шестнадесетичният низ `D2FE28` се превръща в:

```
110100101111111000101000
VVVTTTAAAAABBBBBCCCCC___
```

Под всеки бит има етикет, посочващ неговата цел:

- Трите бита, означени с `V` (`110`), са версията на пакета, `6`.
- Трите бита, означени с `T` (`100`), са типът на пакета, `4`, което означава, че пакетът е пакет-литерал, кодиращ двоично число.
- Петте бита, означени с `A` (`10111`), започват с `1` (следователно това не е последната група) и съдържат първите четири бита от числото `0111`.
- Петте бита, означени с `B` (`11110`), започват с `1` (следователно това не е последната група) и съдържат още четири бита от числото `1110`.
- Петте бита, означени с `C` (`00101`), започват с `0` (следователно това е последната група, край на пакета) и съдържат последните четири бита от числото `0101`.
- Трите немаркирани бита с `0` в края са допълнителни заради шестнадесетичното представяне и трябва да се игнорират.

И така, този пакет представлява литерал с двоично представяне `011111100101`, което е `2021` в десетична система.

Всеки друг тип пакет (всеки пакет с тип, различен от `4`) представлява **пакет-оператор**, който извършва някакво изчисление на един или повече подпакети, съдържащи се в него. За момента конкретните операции не са важни; ще се съсредоточим върху анализа на йерархията на подпакетите.

Един пакет-оператор съдържа един или повече пакети. За да се посочи кои следващи двоични последователности представляват неговите подпакети, се използва един от два режима, посочени от бита непосредствено след заглавието на пакета; това ще наричаме **тип на дължината**:

- Ако типът на дължината е `0`, то следващите `15` бита са число, което представлява общата дължина в битове на подпакетите, съдържащи се в този пакет.
- Ако типът на дължината е `1`, то следващите `11` бита са число, което представлява броя на подпакетите, непосредствено съдържащи се в този пакет.

Накрая, след бита, задаващ типа на дължината и `15`-битовото или `11`-битовото поле, се появяват подпакетите.

Например, пакетът-оператор с шестнадесетичен запис `38006F45291200` има тип на дължината със стойност `0` и съдържа два подпакета:

```
001110000000000001101111010001010010100100100000000
VVVTTTILLLLLLLLLLLLLLLAAAAAAAAAAABBBBBBBBBBBBBBB___
```

- Трите бита, означени с `V` (`001`), са версията на пакета, `1`.
- Трите бита, означени с `T` (`110`), са `ID` на типа пакет, `6`, което означава, че пакетът е оператор.
- Битът с етикет `I` (`0`) е `ID` на типа дължина, което показва, че дължината е `15`-битово число, представляващо броя на битовете в подпакетите.
- `15`-те бита, означени с `L` (`000000000011011`), съдържат дължината на подпакетите в битове, `27`.
- `11`-те бита, означени с `A`, съдържат първия подпакет, буквена стойност, представляваща числото `10`.
- `16`-те бита, означени с `B`, съдържат втория подпакет, буквена стойност, представляваща числото `20`.

След прочитане на `11` и `16` бита от данни на подпакета се достига общата дължина, посочена в `L` (`27`), и така анализирането на този пакет приключва.

Пакетът-оператор с шестнадесетичен запис `EE00D40C823060` има тип на дължината със стойност 1 и съдържа три подпакета:

```
111011100000000001101010000001100100000100011000001100000
VVVTTTILLLLLLLLLLLAAAAAAAAAAABBBBBBBBBCCCCCCCCCCCCC_____
```

- Трите бита, означени с `V` (`111`), са версията на пакета, `7`.
- Трите бита, означени с `T` (`011`), са `ID` на типа пакет, `3`, което означава, че пакетът е оператор.
- Битът с етикет `I` (`1`) е `ID` на типа дължина, който показва, че дължината е `11`-битово число, представляващо броя на подпакетите.
- `11`-те бита, означени с `L` (`00000000011`), съдържат броя на подпакетите, `3`.
- `11`-те бита, означени с `A`, съдържат първия подпакет, литерала `1`.
- `11`-те бита, означени с `B`, съдържат втория подпакет, литерала `2`.
- `11`-те бита, обозначени с `C`, съдържат третия подпакет, литерала `3`.

След прочитане на трите подпакета се достига броят на подпакетите, посочен в `L` (`3`), и така анализирането на този пакет приключва.

Целта на тази задача е да се анализира йерархията на пакетите по време на предаването и да се намери **сумата от техните стойности за версиите**.

Ето още няколко примера за шестнадесетично кодирани предавания:

- `8A004A801A8002F478` представлява операторен пакет (с версия `4`), който съдържа операторен пакет (с версия `1`), който съдържа операторен пакет (с версия `5`), който съдържа пакет-литерал (с версия `6`); този пакет има сума от версиите **`16`**.

- `620080001611562C8802118E34` представлява операторен пакет (с версия `3`), който съдържа два подпакета; всеки подпакет е операторен пакет, който съдържа два пакета-литерали. Сборът на версиите е **`12`**.
- `C0015000016115A2E0802F182340` има същата структура като предишния пример, но най-външният пакет използва идентификатор на тип с различна дължина. Сборът на версиите е **`23`**.
- `A0016C880162017C3686B18A3D4780` е операторен пакет, който съдържа операторен пакет, който съдържа операторен пакет, който съдържа пет пакета-литерали; Сборът на версиите е **`31`**.

Да се дефинира функция **`versionSum :: String -> Int`**, която приема съобщение от предаването (в шестнадесетичен вид) и намира сумата от версиите на всички пакети, съдържащи се в него.

Примери:

```
versionSum "8A004A801A8002F478"             → 16
versionSum "620080001611562C8802118E34"     → 12
versionSum "C0015000016115A2E0802F182340"   → 23
versionSum "A0016C880162017C3686B18A3D4780" → 31
```