AtCoder Beginners Selection を Haskell で

書いた日： 2019 年 12 月 15 日、書いた人： @unnohideyuki[*1]

はじめに

これは、Haskell Advent Calendar 2019[*2] 15 日目の記事です。

AtCoder[*3] は、オンラインで参加できるプログラミングコンテスト（競技プログラミング）のひとつです。わたしも、この AtCoder に Haskell で参戦してみたいと思い、そのために必要そうなことがらを調べてみました。

まずは、AtCoder の入門用に選出された 10 個の過去問からなる AtCoder Beginners Selection[*4] (通称 ABS）を Haskell で書いていき、その後、そこで用いた技術要素を紹介していくことにしたいと思います。

また、Haskell で競技プログラミングに参戦するにあたって心配していたのは、実行速度の問題と、DP (Dynamic Programming) のように配列を更新しながら計算していく方法についてだったので、それらについても少し触れます。

AtCoder Beginners Selection の各問題に対する回答例

いかに、ABS の 10 問それぞれについて、問題文へのリンクと Haskell で書いた回答例を示します。

1. ABC086A - Product

ABC086A の問題文[*5]

```
import           Control.Monad
import           Data.Maybe
import qualified Data.ByteString.Char8 as BS

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  [a, b] <- getIntList
  putStrLn $ ["Even", "Odd"] !! (a * b `mod` 2)
```

[*1] https://twitter.com/unnohideyuki

[*2] https://qiita.com/advent-calendar/2019/haskell

[*3] https://atcoder.jp/

[*4] https://atcoder.jp/contests/abs

[*5] https://atcoder.jp/contests/abs/tasks/abc086_a

2. ABC081A - Placing Marbles

ABC081A の問題文[*6]

```
import qualified Data.ByteString.Char8 as BS

main = do
  s <- BS.getLine
  print . BS.length . BS.filter (== '1') $ s
```

3. ABC081B - Shift only

ABC081B の問題文[*7]

```
import           Control.Monad
import           Data.Bits
import qualified Data.ByteString.Char8 as BS
import           Data.List
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  n <- getInt
  as <- getIntList
  print . minimum . map f $ as
  where
    f x | x .&. 1 == 1 = 0
        | otherwise    = 1 + f (x `shiftR` 1)
```

[*6] https://atcoder.jp/contests/abs/tasks/abc081_a

[*7] https://atcoder.jp/contests/abs/tasks/abc081_b

4. ABC087B - Coins

ABC087B の問題文[*8]

```
import           Control.Monad
import qualified Data.ByteString.Char8 as BS
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
getInt = readInt <$> BS.getLine

main = do
  [a, b, c, x] <- replicateM 4 getInt
  print . length $
    [1 | i<-[0..a], j<-[0..b], k<-[0..c], 500*i + 100*j + 50*k == x]
```

5. ABC083B - Some Sums

ABC083B の問題文[*9]

```
import           Control.Monad
import           Data.Maybe
import qualified Data.ByteString.Char8 as BS

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  [n, a, b] <- getIntList
  print $ sum [x | x <- [1..n], f x >= a, f x <= b]
  where f 0 = 0
        f x = x `mod` 10 + f (x `div` 10)
```

[*8] https://atcoder.jp/contests/abs/tasks/abc087_b
[*9] https://atcoder.jp/contests/abs/tasks/abc083_b

6. ABC088B - Card Game for Two

ABC088B の問題文[*10]

```
import           Control.Monad
import           Data.List
import           Data.Maybe
import qualified Data.ByteString.Char8 as BS

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  _ <- getInt
  as <- getIntList
  print $ sum $ zipWith (*) (cycle [1, -1]) (sortBy (flip compare) as)
```

`sortBy (flip compare) as` の部分は、はじめ `reverse $ sort as` と書いていたのですが、すると hlint が、「reverse は避けよ」と書き換えを促してくれました。

7. ABC085B - Kagami Mochi

ABC085B の問題文[*11]

```
import           Control.Monad
import           Data.Bits
import qualified Data.ByteString.Char8 as BS
import           Data.List
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
getInt = readInt <$> BS.getLine

main = do
  n <- getInt
```

[*10] https://atcoder.jp/contests/abs/tasks/abc088_b
[*11] https://atcoder.jp/contests/abs/tasks/abc085_b

```
  ds <- replicateM n getInt
  print . length . group . sort $ ds
```

8. ABC085C - Otoshidama

ABC085C の問題文[*12]

```
import           Control.Monad
import qualified Data.ByteString.Char8 as BS
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  [n, y] <- getIntList
  let a = [map show [a, b, c] | a <- [0..n], b <- [0..(n-a)],
           let c = n-a-b, 10000*a + 5000*b + 1000*c == y]
  if null a
    then putStrLn "-1 -1 -1"
    else putStrLn . unwords . head $ a
```

9. ABC049C - 白昼夢 / Daydream

ABC049C の問題文[*13]

```
{-# LANGUAGE OverloadedStrings #-}
import qualified Data.ByteString.Char8 as BS

main = do
  s <- BS.getLine
  putStrLn . solve $ BS.reverse s
  where
```

[*12] https://atcoder.jp/contests/abs/tasks/abc085_c
[*13] https://atcoder.jp/contests/abs/tasks/arc065_a

```
    solve s | s == "" = "YES"
            | BS.reverse "dream"   `BS.isPrefixOf` s = solve $ BS.drop 5 s
            | BS.reverse "dreamer" `BS.isPrefixOf` s = solve $ BS.drop 7 s
            | BS.reverse "erase"   `BS.isPrefixOf` s = solve $ BS.drop 5 s
            | BS.reverse "eraser"  `BS.isPrefixOf` s = solve $ BS.drop 6 s
            | otherwise = "NO"
```

10. ARC089A - Traveling

ARC089A の問題文[*14]

```
import           Control.Monad
import qualified Data.ByteString.Char8 as BS
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

main = do
  n <- getInt
  xs <- replicateM n getIntList
  putStrLn $ if solve ([0, 0, 0]:xs) then "Yes" else "No"
  where
    solve [a, b]   = reachable a b
    solve (a:b:xs) = reachable a b && solve (b:xs)
    reachable [t0, x0, y0] [t1, x1, y1] =
      t >= dist && (t - dist) `mod` 2 == 0
      where dist = abs (x1 - x0) + abs (y1- y0)
            t = t1 - t0
```

解説

10 個の回答例は、いずれも短いプログラムなので、それぞれについて長々と説明する必要はないかなと思います。いずれも `main` 関数はかなり簡潔で、Haskell の記述力の高さを再認識しました。

[*14] https://atcoder.jp/contests/abs/tasks/arc089_a

あまり普通じゃない、というか、競技プログラミング向けかなと思うのが、入力に String ([Char]) を一切使わなかったところでしょうか。ABS では、時間制限にひっかかるような大きな入力はないので、普通に String を用いても問題にならないのですが、後のことを考えると、最初から String を使わない入力関数群を用意しておいた方がいいです。

```
{-# LANGUAGE OverloadedStrings #-}
import           Control.Monad
import           Data.Maybe
import qualified Data.ByteString.Char8 as BS

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine
```

使い方は回答例のなかにありますが、以下にまとめておきます。

整数をひとつ読む

整数をひとつ読み込むには、getInt を用います。

```
n <- getInt
```

空白で区切られた複数の整数を読む

一行に空白でくぎられた複数の整数がならんでいる場合には、getIntList を使います。以下のようにパターンマッチと組み合わせるのが便利ですね。

```
[a, b] <- getIntList
```

複数行から読み込む

複数行からなる入力を読み込む際には、replicateM を使います。

```
xs <- replicateM n getInt
```

10. ARC089A - Traveling[*15] の回答例にあるように、`replicateM m getIntList` のようにすれば、リストのリストとなります。

もう少し進んだ例１： Vector を使おう

これまでの例では、`String` が遅いので使うのをやめましょうということを述べてきたわけですが、`String` が遅いというのは、要するに `List` が遅いからなので、大量に読みこむ場合には、`Vector` を使うべきです。

以下は、ABC147D[*16] の回答例：

```
import           Control.Monad
import           Data.Array.IO
import           Data.Bits
import qualified Data.ByteString.Char8       as BS
import           Data.Char
import           Data.Maybe
import qualified Data.Vector.Unboxed         as V
import qualified Data.Vector.Unboxed.Mutable as VM

readInt = fst . fromJust . BS.readInt
getInt = readInt <$> BS.getLine
getIntVec n = V.unfoldrN n (BS.readInt . BS.dropWhile isSpace) <$> BS.getLine

main = do
  n <- getInt
  as <- getIntVec n

  cs <- VM.new 61
  VM.set cs (0::Int)

  forM_ [0..n-1] $ \i -> do
    let a = as V.! i
    forM_ [0..60] $ \j ->
      when (testBit a j) $ do
        c <- VM.read cs j
        VM.write cs j (c+1)
  cs' <- V.freeze cs
```

[*15] https://uhideyuki.sakura.ne.jp/studs/index.cgi/ja/AbsInHaskell#p13
[*16] https://atcoder.jp/contests/abc147/tasks/abc147_d

```
  let calc i c a = let c1 = fromIntegral c :: Integer
                       c2 = fromIntegral (n-c) :: Integer
                   in
                     c1 * c2 * bit i + a

  print $ V.ifoldr calc 0 cs' `mod` 1000000007
```

この問題では、最大 3×10^5 個の整数が並ぶことになるので、そういった場合のために `getIntVec` を用意しました。

これさえあれば、`getIntList` は不要かというと、そうでもなくて、`[a, b] <- getIntList` のような使い方が便利なので、両方使い分けるのがいいと思います。

また、この例は、mutable な Vector の使用例にもなっています。

`ifoldr` というのも、私は今回初めてつかいました。fold 関数のなかで、その要素に対応する添え字 i が使えるという、便利なしろもの。

もう少し進んだ例２： Array も使おう

Mutable な Vector があれば、一次元 dp の類は OK なのですが、AtCoder には多次元 dp も頻出します。そこで、Array を使います。

以下は、少し古い問題ですが、ABC012D[*17] の回答例です。

```
import           Control.Exception     (assert)
import           Control.Monad
import qualified Control.Monad.ST      as ST
import qualified Data.Array.IO         as IO
import qualified Data.Array.ST         as ST
import qualified Data.Array.Unboxed    as A
import qualified Data.ByteString.Char8 as BS
import           Data.List
import           Data.Maybe

readInt = fst . fromJust . BS.readInt
readIntList = map readInt . BS.words
getInt = readInt <$> BS.getLine
getIntList = readIntList <$> BS.getLine

warshallFloyd :: A.UArray (Int, Int) Int -> A.UArray (Int, Int) Int
```

[*17] https://atcoder.jp/contests/abc012/tasks/abc012_4

```
warshallFloyd iarr = oarr
  where
    l  = (fst . fst . A.bounds) iarr
    l' = (snd . fst . A.bounds) iarr
    r  = (fst . snd . A.bounds) iarr
    r' = (snd . snd . A.bounds) iarr
    oarr = assert (l==l' && r==r') $ ST.runSTUArray $ do
      d <- ST.thaw iarr
      forM_ [l..r] $ \k ->
        forM_ [l..r] $ \i ->
          forM_ [l..r] $ \j -> do
            dik <- ST.readArray d (i,k)
            dkj <- ST.readArray d (k,j)
            dij <- ST.readArray d (i,j)
            when (dik + dkj < dij) $
              ST.writeArray d (i,j) (dik + dkj)
      return d

main = do
  let infty = 10^6 :: Int
  [n, m] <- getIntList

  arr <- IO.newArray ((1,1),(n,n)) infty :: IO (IO.IOUArray (Int, Int) Int)
  forM_ [1..n] $ \i -> IO.writeArray arr (i,i) 0
  replicateM_ m $ do
    [a, b, t] <- getIntList
    IO.writeArray arr (a, b) t
    IO.writeArray arr (b, a) t

  edges <- IO.freeze arr
  let d = warshallFloyd edges
  print $ minimum [maximum [d A.! (i, j) | j<-[1..n]] | i<-[1..n]]
```

Mutable なデータの参照・更新には、IO や ST （両者を含む共通クラスが PrimMonad) を用いるのですが、上の例には両方の使用例が含まれます。

あと、最後の行なんかみると、やっぱりリストは便利だなぁ（Vector の方が速いからと、完全のリストを禁じてにはできないなぁ）という気がしますね。