競技プログラミングの鉄則*

やさしい理系お兄ちゃん

January 31, 2024

Contents

0 はじめに **2**

1 アルゴリズムと計算量 **2**
1.1 導入問題 2
1.2 全探索 1 3
1.3 全探索 2 4
1.4 2 進法 5
1.5 チャレンジ問題 6

2 累積和 **7**
2.1 一次元の累積和 (1) 7
2.2 一次元の累積和 (2) 9
2.3 二次元の累積和 (1) 9
2.4 二次元の累積和 (2) 9
2.5 チャレンジ問題 9

*https://atcoder.jp/contests/tessoku-book

0　はじめに

この PDF は関数型言語 Haskell の習得のために、「競技プログラミングの鉄則」の問題に対して Haskell で記述したコード例をまとめた資料です。私のプログラミング力の向上を主目的とし、Haskell を学びたい人への手助けとなると良いと考えています。Haskell の I/O[1] は一筋縄でいかないので、アルゴリズム以前の問題で苦戦してしまうかもしれないでしょう。ですが、入出力に時間を取られるのは勿体無いため、そのような面でも Haskell 習得の支えになれば良いと願います。

1　アルゴリズムと計算量

1.1　導入問題

問題. 整数 N が与えられるので、一辺の長さが N であるような正方形の面積を出力するプログラムを作成してください。

Listing 1: A01.hs

```
1 main :: IO ()
2 main = do
3     n <- readLn :: IO Int
4     putStrLn $ show (n * n)
```

解説. `readLn` の型は `Read a => IO a` です。ここで、`:: IO Int` と後ろに記述することで型変数 `a` は `Int` 型であると宣言することができます。もちろん、書かなくても `(*)` という演算が後ろで行われていることから、型クラス `Num` に属することを推論してくれます。（書いた方が見やすいのではないでしょうか？）`n <- readLn :: IO Int` を実行すると、`n` は通常の整数値になります。`show` の型は `Show a => a -> String` ですので、`n * n` の結果を `show` に渡すことで、整数値を文字列に変換することができます。最後に `String -> IO ()` という型をもつ `putStrLn` に渡せば完了です。

問題. 整数 A と B が与えられるので、$A + B$ の値を出力するプログラムを作成してください。ただし、制約は $1 \leqq A \leqq 100$、$1 \leqq B \leqq 100$ であるとします。

Listing 2: B01.hs

```
1 main :: IO ()
2 main = do
3     [a, b] <- map read . words <$> getLine
4     putStrLn $ show (a + b)
```

解説. まずは 2 つの空白区切りの整数の入力から説明します。<$>について、定義は以下の通りです。

```
(<$>) :: (Functor f) => (a -> b) -> f a -> f b
f <$> x = fmap f x
```

<$>の型に注目して下さい。`IO` は `Functor` のインスタンスですので、`f` を `IO` に置き換えると、`<$> :: (a -> b) -> IO a -> IO b` となります。また、

[1] I/O：Input(標準入力)， Output(標準出力)

- `getLine :: IO String`
- `map read . words :: Read b => String -> [b]`

と定義されています。以上を合わせると

```
map read . words <$> getLine :: (Read b) => IO [b]
```

となることが分かります。`IO [b]` については、コンパイラが勝手に `IO [Int]` としてくれます。[2] 当然、`:: IO Int` と型を指定するのもいいことでしょう。最終的に、結果を `Int` 型のリストとして `[a, b]` に束縛することができます。（➡注 入力が 2 つであると分かっているので、`[a, b]` としています。）あとは計算結果を A01 と同様に出力すれば完了です。

1.2　全探索 1

問題. N 個の整数 A_1, A_2, ..., A_N の中に、整数 X が含まれるかどうかを判定するプログラムを作成してください。

Listing 3: A02.hs

```
1 main :: IO ()
2 main = do
3     [n, x] <- map read . words <$> getLine :: IO [Int]
4     as <- map read . words <$> getLine :: IO [Int]
5     putStrLn $ if x `elem` as then "Yes" else "No"
```

解説. 3, 4 行目に関しては Listing 2 の解説で紹介しました。そろそろ気付いたかもしれませんが、Haskell では `for` を使った繰り返し処理を行わないので、リストの長さを示す `n` を受け取る必要がないことが多いです。そのため、3 行目は

```
[_, x] <- map read . words <$> getLine
```

としても良いです。そして、リスト `as` の中に `x` が含まれているかは、関数 `elem` で判定することができます。関数 `elem` は中置記法を用いることが多いです。

問題. A 以上 B 以下の整数のうち、100 の約数であるものは存在しますか。答えを Yes か No で出力するプログラムを作成してください。

Listing 4: B02.hs

```
1 main :: IO ()
2 main = do
3     [a, b] <- map read . words <$> getLine :: IO [Int]
4     let x = [a..b]
5     putStrLn $ if 0 `elem` map (100 `mod`) x then "Yes" else "No"
```

解説. 4 行目の `[a..b]` は、`[a, a+1, a+2, ..., b-1, b]` を表します。このリストのすべての要素について、100 から割った余りを考えます。その余りのリストの中に `0` が含まれてるか否かを判定します。

[2] Type Defaulting というものが使われています。詳しくはいつか書きます。（January 31, 2024）

1.3　全探索 2

問題. 赤いカードが N 枚あり、それぞれ整数 P_1, P_2, ..., P_N が書かれています。また、青いカードが N 枚あり、それぞれ整数 Q_1, Q_2, ..., Q_N が書かれています。太郎くんは、赤いカードの中から 1 枚、青いカードの中から 1 枚、合計 2 枚のカードを選びます。選んだ 2 枚のカードに書かれた整数の合計が K となるようにする方法は存在しますか。答えを出力するプログラムを書いてください。

Listing 5: A03.hs

```
1 main :: IO ()
2 main = do
3     [_, k] <- map read . words <$> getLine :: IO [Int]
4     ps <- map read . words <$> getLine :: IO [Int]
5     qs <- map read . words <$> getLine :: IO [Int]
6     putStrLn $ if k `elem` [p + q | p <- ps, q <- qs] then "Yes"
          else "No"
```

解説. 6 行目の

```
[p + q | p <- ps, q <- qs]
```

では、リスト `ps` とリスト `qs` から一つずつ要素を取り出し、それらを足し合わせて可能性のある答えをすべて計算して提示しています。これを**非決定性計算**とみなすことができます。このとき、

```
(+) <$> ps <*> qs
```

と書くことができます。これについては**アプリカティブ**という用語を聞くまでは保留で構いません。非決定性計算という言葉も同様です。（いずれ、後者の記法の方がきれいだと感じる日が来るはずです！私は内包表記も好きです。）

問題. N 個の商品があり、商品 i ($1 \leqq i \leqq N$) の価格は A_i 円です。異なる 3 つの商品を選び、合計価格をピッタリ 1000 円にする方法は存在しますか。答えを Yes か No で出力するプログラムを作成してください。ただし、制約は $3 \leqq N \leqq 1000$ であるとします。

Listing 6: B03.hs

```
1  just :: Int -> Int -> [Int] -> Bool
2  just _ _ [] = False
3  just 1 num xs = num `elem` xs
4  just 2 num (x : xs) = any (\m -> m + x == num) xs || just 2 num
       xs
5  just k num as@(x : xs) = length as >= k && k >= 0
6      && (just (k - 1) (num - x) xs || just k num xs)
7
8  main :: IO ()
9  main = do
10     _ <- read <$> getLine :: IO Int
11     as <- map read . words <$> getLine :: IO [Int]
12     putStrLn $ if just 3 1000 as then "Yes" else "No"
```

解説. 本問も非決定性計算だから、Listing 5 と同様に

```
1000 `elem` [x + y + z | x <- as, y <- as, z <- as]
```

で可能性のある答えを列挙しよう！としては...方針は正しいですが、その手法に誤りが生じています。これでは、同じ商品を選ぶ場合も含まれてしまうからです。

それでは、関数 just について説明します。`just k num xs` とは、リスト `xs` から、異なる `k` 個の要素を取り出して足し合わせたとき、その答えが `num` になることがあるか否かを示す式です。したがって、本文では入力から得たリストに `just 3 1000` を適用します。

① `just _ _ []`
リストが空なら、問答無用で `False` を返します。

② `just 1 num xs`
リストから 1 個だけ取り出して、`num` に等しいかを判定します。おなじみの関数 `elem` を使えば解決です。

③ `just 2 num xs`
まず、`any (\m -> m + x == num) xs` で、先頭要素とその他の要素を足して `num` になるかを判定しています。そこで、`True` が得られなかった場合、`just 2 num xs` で先頭要素を捨てて残りのリストで同じことを試します。以降、再帰的に判定します。

④ `just k num xs`
`k` 個取り出すので、`k` が負でないこと、リストの長さが `k` 以上であることを判定しておきます。(➡**注** このような判定を書かずとも、AtCoder では正しい入力が与えられることが保証されているので問題ありません。しかし、一般的にはそうではないので、意図せぬ入力にも備えておく癖をつけましょう！とは言っても、AtCoder ではこれを書くのは手間ですので、やっぱり以降はサボります。)

あとは、③ と同様に、まずは先頭要素とその他の要素について条件を満たす組合せが存在するかを判定します。このとき、`just (k - 1) (num - x) xs` に帰着されることを理解しましょう。

1.4　2進法

問題. 整数 N が 10 進法表記で与えられます。N を 2 進法に変換した値を出力するプログラムを作成してください。

Listing 7: A04.hs

```
1  toBinary :: Int -> String
2  toBinary 1 = "1"
3  toBinary n = toBinary (n `div` 2) ++ if even n then "0" else "1"
4
5  assignZero :: Int -> String -> String
6  assignZero 0 str = str
7  assignZero x str = "0" ++ assignZero (x - 1) str
8
9  main :: IO ()
10 main = do
11     n <- readLn :: IO Int
12     let binary = toBinary n
13     putStrLn $ assignZero (10 - length binary) binary
```

解説. 関数 `toBinary` の動きを見てみましょう。

```
toBinary 12 -> toBinary 6 ++ "0" -> toBinary 3 ++ "00"
            -> toBinary 1 ++ "100" -> "1100"
```

どうでしょう？2 進数を計算する方法は数 A で習いますね！

問題. 整数 N（8 桁以内）が 2 進法表記で与えられます。N を 10 進法に変換した値を出力するプログラムを作成してください。

Listing 8: B04.hs

```
bin2list :: Int -> [Int] -> [Int]
bin2list 0 ys = ys
bin2list x ys = bin2list (x `div` 10) (x `mod` 10 : ys)

list2dec :: [Int] -> Int
list2dec [] = 0
list2dec (x : xs) = x * (2 ^ length xs) + list2dec xs

main :: IO ()
main = do
    n <- readLn :: IO Int
    print $ list2dec $ bin2list n []
```

解説. 与えられた入力（例えば 101011）を

```
101011 -> [1,0,1,0,1,1] -> 2^5 + 2^3 + 2^1 + 2^0 -> 43
```

という流れで計算しています。入力を `Int` として受け取り、それをリスト構造に変換しているなら、最初からリスト構造である `String` で受け取るべきかもしれませんね。

おっと、関数 `print` が初登場しました。型は以下のようになっています。`putStrLn` との違いを見てみましょう。

```
print :: Show a => a -> IO ()
putStrLn :: String -> IO ()
```

1.5　チャレンジ問題

問題. 赤・青・白の 3 枚のカードがあります。太郎くんは、それぞれのカードに 1 以上 N 以下の整数を書かなければなりません。3 枚のカードの合計を K にするような書き方は何通りありますか。

Listing 9: A05.hs

```
main :: IO ()
main = do
    [n, k] <- map read . words <$> getLine :: IO [Int]
    print $ length [(a, b, c) | a <- [1..n], b <- [1..n], let c =
        k - a - b, 1 <= c, c <= n]
```

解説. リスト内包表記のおかげで解説いらず！1 以上 `n` 以下の範囲で、`a + b + c = k` となるように `c` を定めてあげます。出来上がったトリプルのリストの長さが答えになります。

2　累積和

2.1　一次元の累積和 (1)

問題. ある遊園地では N 日間にわたるイベントが開催され、i 日目には A_i 人が来場しました。以下の Q 個の質問に答えるプログラムを作成してください。

- 質問 1：L_1 日目から R_1 までの来場者数は？

  ⋮

- 質問 Q：L_Q 日目から R_Q までの来場者数は？

Listing 10: A06_00.hs

```
import Control.Monad ( replicateM_ )

cumulativeSum :: [Int] -> [Int]
cumulativeSum = scanl (+) 0

solve :: [Int] -> [Int] -> Int
solve s [l, r] = (s !! r) - (s !! (l - 1))

main :: IO ()
main = do
    [_, q] <- map read . words <$> getLine :: IO [Int]
    as <- map read . words <$> getLine :: IO [Int]
    let cumSum = cumulativeSum as
    replicateM_ q $ do
        lr <- map read . words <$> getLine :: IO [Int]
        print $ solve cumSum lr
```

解説. 新しい関数がたくさん登場しました。一つずつ紹介します。

関数 `scanl` の定義は以下です。

```
scanl :: (b -> a -> b) -> b -> [a] -> [b]
scanl = scanlGo
    where
        scanlGo :: (b -> a -> b) -> b -> [a] -> [b]
        scanlGo f q ls = q : (case ls of
                              []   -> []
                              x:xs -> scanlGo f (f q x) xs)
```

なぜ、`scanlGo` を挟んでいるのか !? (実力不足によりわかっておりません。) 実装自体は理解できそうですね！動作を見てみましょう。

```
scanl (+) 0 [1,2,3,4,5] -> [0,1,3,6,10,15]
```

これがまさに累積和になります。引数のリストの長さを N とすると、関数 `cumulativeSum` は $O(N)$ となります。上記の例で考えると、2 日目から 4 日目までの来場者数は、`10 - 1 = 9`（人）となります。3 日目から 5 日目までの来場者数は、`15 - 3 = 12`（人）となります。

リストの `k` 番目の要素にアクセスするために関数 (`!!`) が使えます。したがって、質問に回答する関数 `solve` は

```
solve s [l, r] = (s !! r) - (s !! (l - 1))
```

と定義できます。(「0 日目は累計 0 人」という無意味とも思える情報をリストの先頭に置いておくことで、リストのインデックスと日にちの数字が同じ数字になるので扱いやすくなっています。) これを、関数 `replicateM_` を用いて `q` 個の質問に適用します。ここでは、使用例だけ紹介します。

```
ghci> replicateM_ 3 (putStrLn "a")
a
a
a
```

これで完成！いえ、実行時間超過です。なぜでしょう... オーダーに問題があります。プログラムとしては正しい出力が得られますが、このプログラムは $O(N^2)$ となっています。$0 \leqq N \leqq 10^5$ ですので、最悪計算量が 10^{10} となってしまうため制限時間超過となります。問題文では、$O(N+Q)$ であることが望ましいと書かれていますので、それを目標にプログラムを変更しましょう。

関数 `cumulativeSum` は $O(N)$ で、関数 `replicateM_` は $O(Q)$ です。したがって、関数 `(!!)` が $O(1)$ であれば、非常に嬉しい！ということになります。しかしながら、関数 `(!!)` は $O(N)$ であるようです。下記の定義からも確認できますね。

```
(!!) :: [a] -> Int -> a
xs     !! n | n < 0 =  errorWithoutStackTrace
                            "Prelude.!!: negative index"
[]     !! _         =  errorWithoutStackTrace
                            "Prelude.!!: index too large"
(x:_)  !! 0         =  x
(_:xs) !! n         =  xs !! (n-1)
```

では、$O(1)$ で配列の要素にアクセスするにはどうしたらよいのでしょうか？ `Data.Array` モジュールを使いましょう。関数 `listArray` はインデックスの範囲とリストを引数に取り、インデックスの付いたリストを返してくれます。

```
listArray (0, 5) [0,1,3,6,10,15]
-> array (0, 5) [(0,0), (1,1), (2,3), (3,6), (4,10), (5,15)]
```

リストにインデックスが付くことにより、i 番目の要素に $O(1)$ でアクセスできるようになります。リストにアクセスする関数は `(!)` です。

これらを用いて変更したプログラムは Listing 11 のようになります。関数 `cumulativeSum` では、累積和を取ったのちに関数 `listArray` を適用しています。関数 `solve` では、関数 `(!!)` が関数 `(!)` に変わっただけですね！

Listing 11: A06_01.hs

```
1 import Control.Monad ( replicateM_ )
2 import Data.Array ( Array, (!), listArray )
3
4 cumulativeSum :: Int -> Int -> [Int] -> Array Int Int
5 cumulativeSum l r as = listArray (l, r) $ scanl (+) 0 as
6
7 solve :: Array Int Int -> [Int] -> Int
8 solve s [l, r] = (s ! r) - (s ! (l - 1))
```

```
 9
10 main :: IO ()
11 main = do
12     [n, q] <- map read . words <$> getLine :: IO [Int]
13     as <- map read . words <$> getLine :: IO [Int]
14     let cumSum = cumulativeSum 0 n as
15     replicateM_ q $ do
16         lr <- map read . words <$> getLine :: IO [Int]
17         print $ solve cumSum lr
```

➡**注** ここで初めて、Figure 1 のように警告が表示されたのではないでしょうか？関数 `solve` の第二引数にパターン漏れがあるらしいです。その通りなのですが、AtCoder に絶大な信頼を置いているので、期待外れの入力が来ないことを前提に、この警告を解消せずに進めていきましょう。警告を解消するとしたら、

```
solve :: Array Int Int -> [Int] -> Int
solve s [l, r] = (s ! r) - (s ! (l - 1))
solve _ _ = -1
```

のように、来場者数としてあり得ない数値を返すようにしておきましょう。

コンパイルエラー

```
app/Main.hs:8:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for ‘solve’:
        Patterns of type ‘Array Int Int’, ‘[Int]’ not matched:
            _ []
            _ [_]
            _ (_:_:_:_)
  |
8 | solve s [l, r] = (s ! r) - (s ! (l - 1))
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Figure 1: 初めての警告

2.2　一次元の累積和 (2)

2.3　二次元の累積和 (1)

2.4　二次元の累積和 (2)

2.5　チャレンジ問題