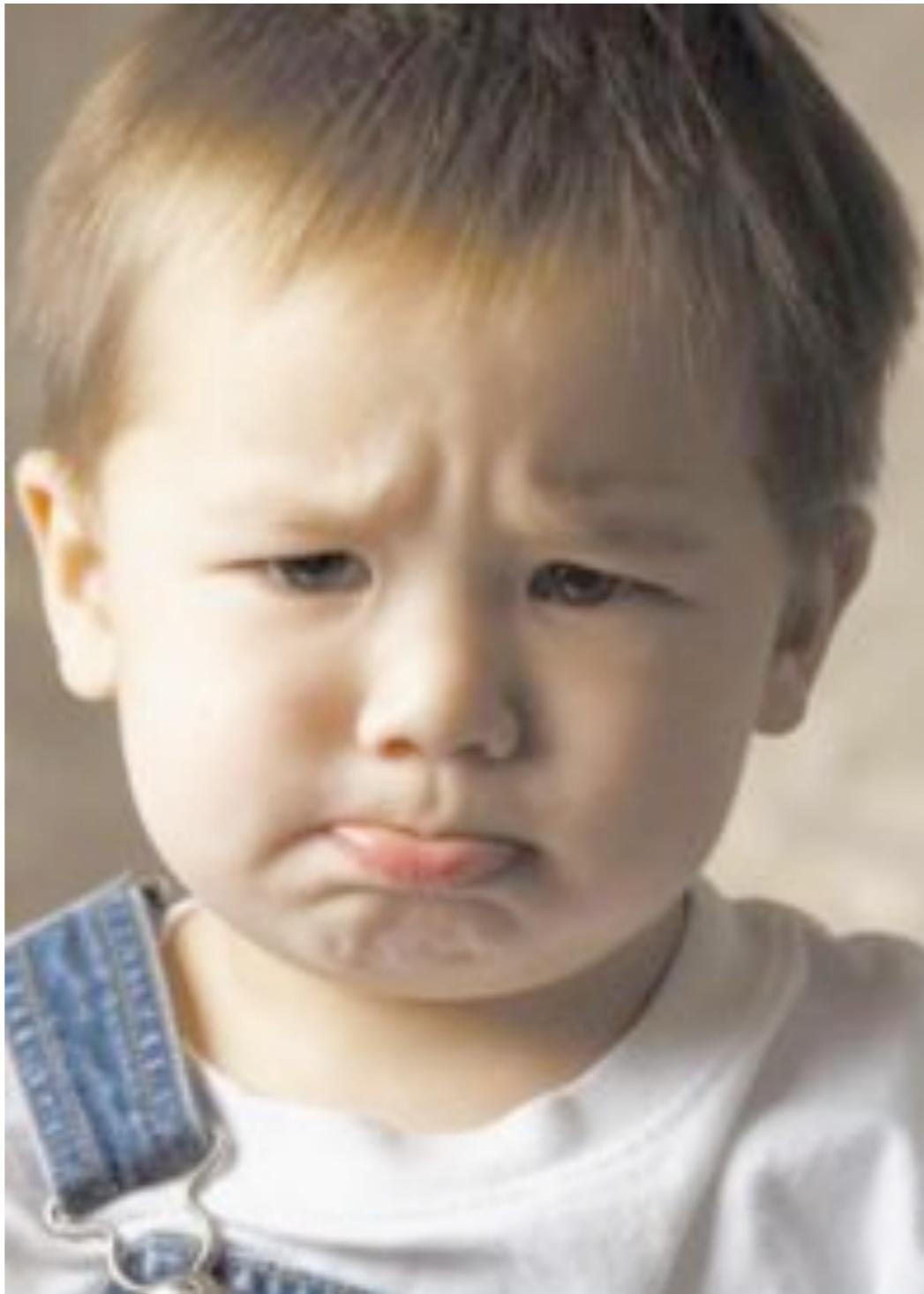


How you could have won the VPW 2011 contest with Haskell



Tom Schrijvers
Programming Languages Group
UGent



... but didn't ???



You

You

You

VPW 2012

The VPW Formula

The VPW Formula

1 team = 1 laptop + 3 people

solve many problems quickly

The VPW Formula *for success*

3. Solve Easy Problems First

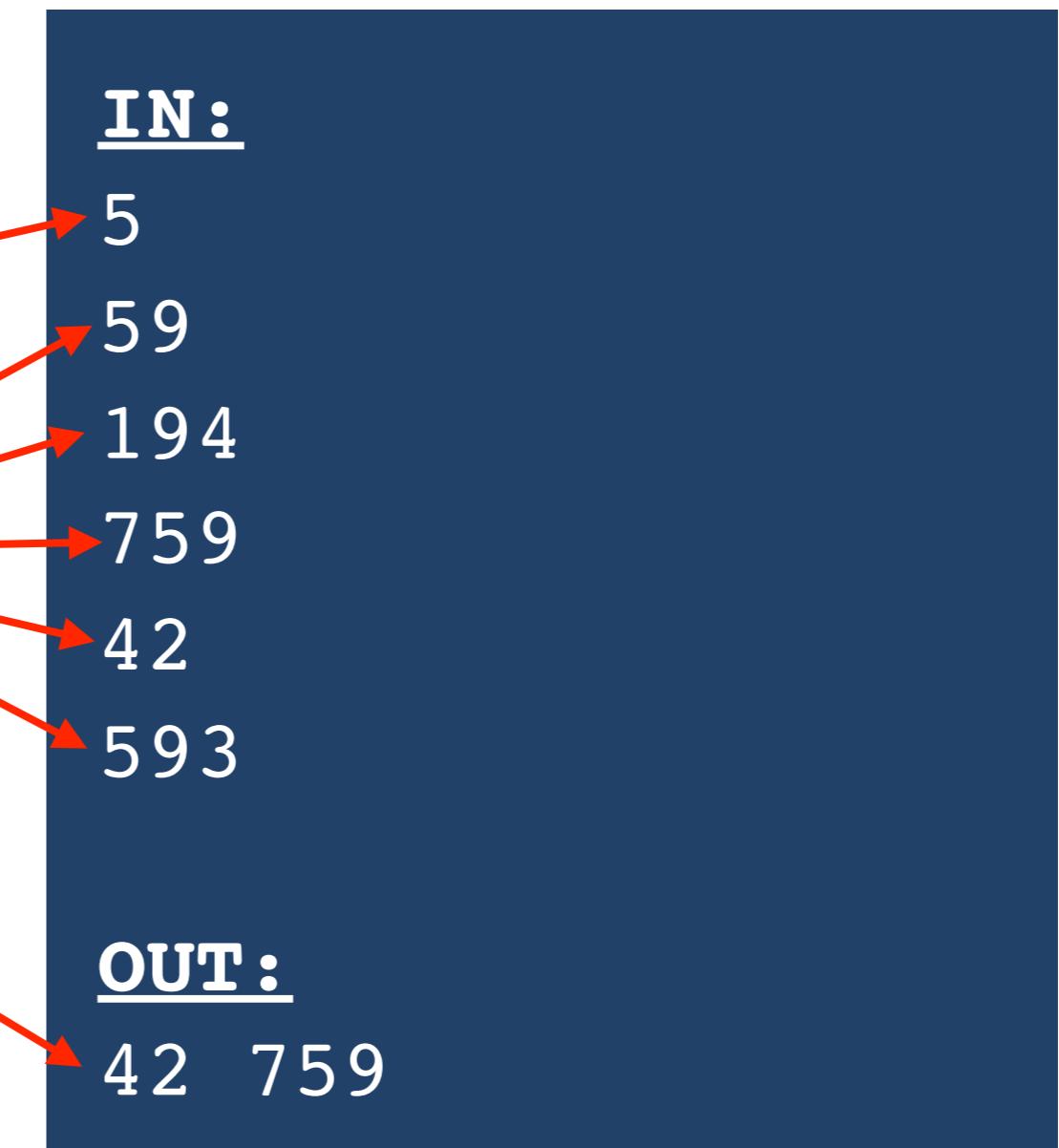
2. Program in Pairs

I. Program in Haskell

Problem 0

Warming Up

- Read n
- Read n numbers
- Write out the minimum and maximum.



Solution

```
main =  
  do n <- readLn  
      xs <- replicateM n readLn :: IO [Int]  
      putStrLn $ unwords  
        $ map show  
        $ [minimum xs, maximum xs]
```

Problem I

HASKELL ROCKS

++++++

FPGFPGFPGFPGF

=====

NQZQUSRPYUSRY

NQZQUSRPYUSRY



FPGFPGFPGFPGF



HASKELL ROCKS



Vignere Code

- Read n
- Encode n messages
- Read m
- Decode m messages

IN:

1

FPG HASKELL ROCKS

1

VPW KGKBGXHBEIWWDHWAJJ

OUT:

NQZQUSRPYUSRY

PROGRAMMING IS FUN

Solution Core

```
encoder, decoder :: String -> String -> String
encoder = coder $ \x y -> (x + y) `mod` 27
decoder = coder $ \x y -> (y - x) `mod` 27

coder :: (Int -> Int -> Int)
       -> (String -> String -> String)
coder (<*>) =
  zipWith $ \x y -> chr $ ord x <*> ord y

ord ' ' = 0
ord c = Char.ord c - 64

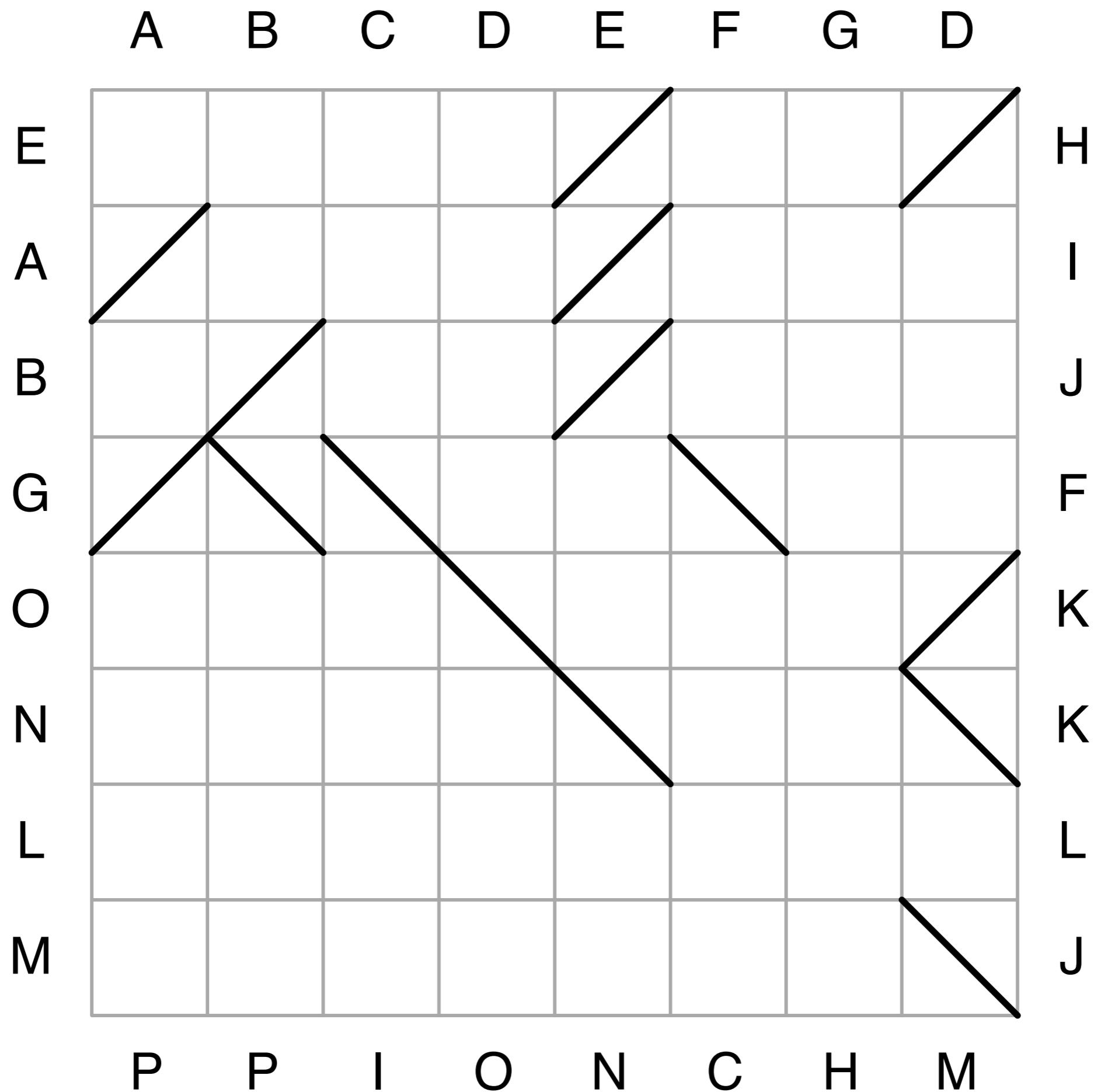
chr 0 = ' '
chr n = Char.chr (n + 64)
```

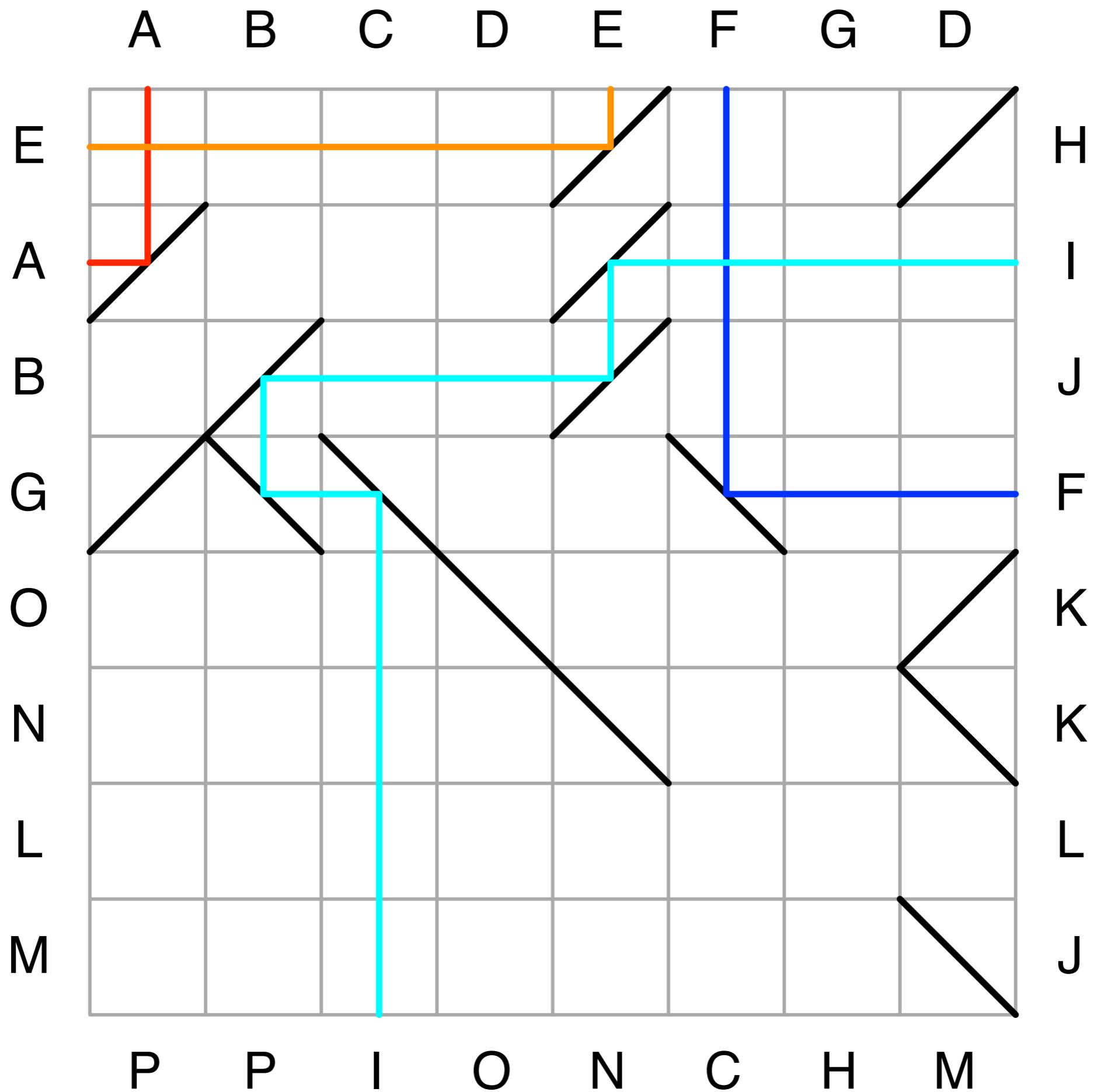
Solution I/O

```
main = do process decoder  
          process encoder
```

```
process :: (String -> String -> String) -> IO ()  
process coder =  
  do n <- readLn  
      replicateM_ n $  
        getLine >>=  
          putStrLn  
          . uncurry coder  
          . (cycle . head &&& unwords . tail)  
          . words
```

Problem 2





Reflections

- Read n
- Read grid
- Write well-formedness

IN:

```
1
8 8
    ABCDEFGD
    E      /   /
    A/     /   I
    B /    /   J
    G/\ \  \   F
    O   \   /K
    N   \   \K
    L       L
    M       \J
    PPIONCHM
```

OUT:

correct

Solution Core

```
check :: Point -> Dir -> Grid -> Bool  
check p dir grid = go dir p == grid ! p
```

where

```
go :: Dir -> Point -> Char  
go dir p = let p' = step dir p  
            in switch (grid ! p') dir p'
```

```
switch :: Char -> Dir -> Point -> Char  
switch '/' dir p = go (deflect (-1) dir) p  
switch '\\' dir p = go (deflect ( 1) dir) p  
switch ' ' dir p = go dir p  
switch sym dir p = sym
```

Points, Directions, ...

```
type Dir    = (Int,Int)
type Point = (Int,Int)
type Grid   = Map Point Char

down, up, left, right :: Dir
down  = (-1, 0) ; up    = ( 1, 0)
left   = ( 0,-1) ; right = ( 0, 1)

step :: Dir -> Point -> Point
step (dx,dy) (x,y) = (x+dx,y+dy)

deflect :: Int -> Dir -> Dir
deflect c (dx,dy)  = (dy * c, dx * c)
```

Check All

```
check_all m n grd =  
    and [check (0,j) up grd | j <- [1..n]]  
    && and [check (m+1,j) down grd | j <- [1..n]]  
    && and [check (i,0) right grd | i <- [1..m]]  
    && and [check (i,n+1) left grd | i <- [1..m]]
```

I/O

```
main = do n <- readLn
          replicateM_ n process
process =
  do [m,n] <- getLine >>=
      return . map read . words
ls <- sequence
  [getLine >>=
    return . zip [(i,j)
                  | j <- [0..n+1]]
    | i <- [0..m+1]]
let arr = fromList (concat ls)
if check_all m n arr
  then putStrLn "correct"
  else putStrLn "verkeerd"
```

Problem 3

How many ways to
change 6 cents?



Change

- Read n
- Read amount and coin denominations
- Write number of ways to change amount

IN:

2

6 1 2 5

100 1 2 5 20 50 100

OUT:

5

1442

Solution Core

```
count :: Int -> [Int] -> Int
count n [c]
| n `mod` c == 0      = 1
| otherwise             = 0
count n (c:cs)
=  sum [ count (n - i * c) cs
| i <- [0..n `div` c] ]
```

Solution I/O

```
main :: IO ()
main =
  do n <- readLn
     replicateM_ n $
       getLine >>=
         print
         . uncurry count
         . (head    &&& reverse . sort . tail)
         . map read
         . words
```

Bonus

l e t t e r

t1t

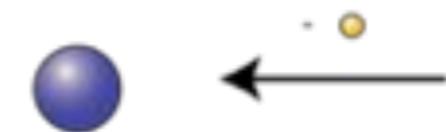
l e t-t e r

s t a t i o n

1t i o n

s t a - t i o n

c a t i o n



1t i o n

. c a²t

• • •

c a t - i o n

Hyphenate

- Read patterns
- Read words
- Write hyphenated words

IN:

.a4

...

z4z5w

babbelaar

...

vlaskop

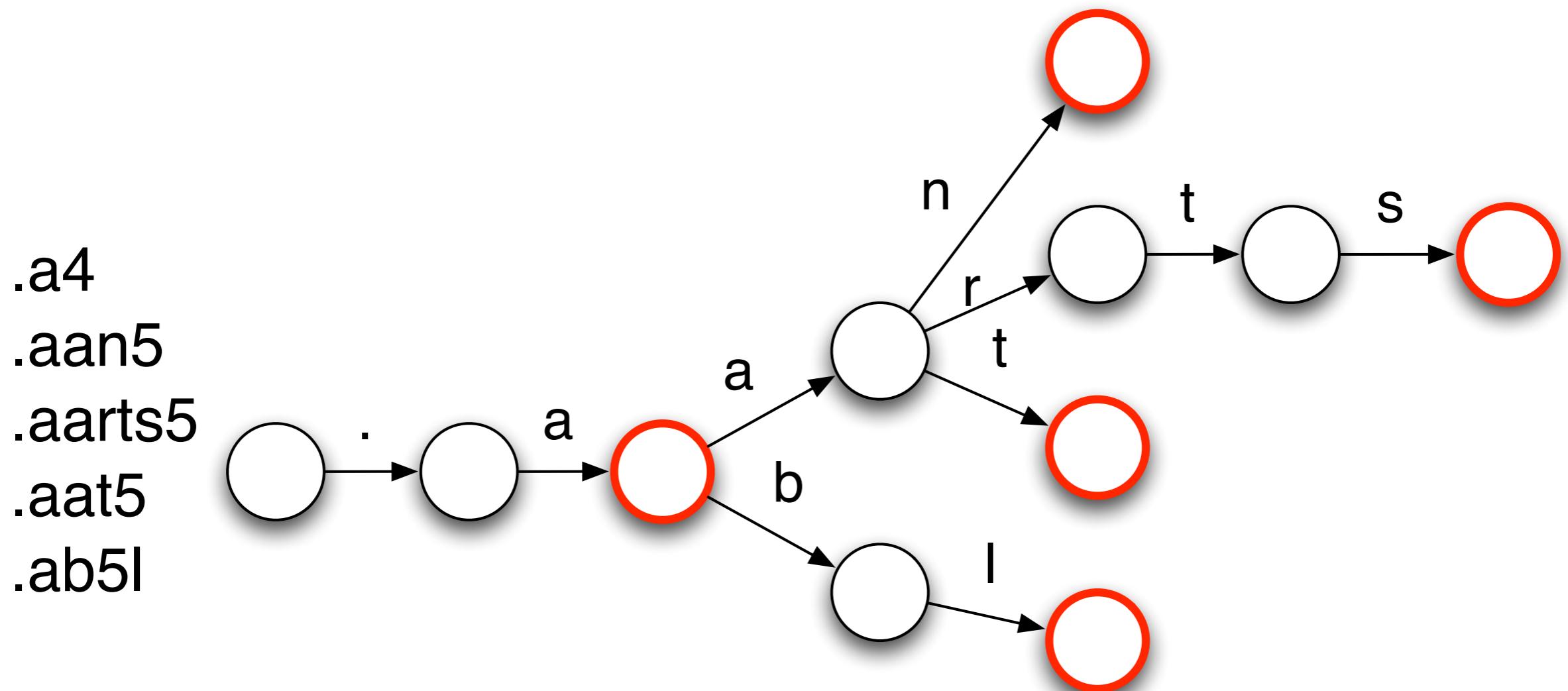
OUT:

bab-be-laar

...

vlas-kop

Trie



Trie

```
data Trie c v = T (Map c (Trie c v)) (Maybe v)
```

```
empty :: Trie c v
```

```
empty = T Map.empty Nothing
```

```
insert :: Ord c
```

```
    => [c] -> Trie c v -> v -> Trie c v
```

```
insert [] (T ts _) r = T ts (Just r)
```

```
insert (c:cs) (T ts mr) r = T ts' mr
```

where

```
ts' = alter ins c ts
```

```
ins (Just trie) = Just (insert trie cs r)
```

```
ins Nothing = Just (insert empty cs r)
```

Trie (ctd.)

```
data Trie c v = T (Map c (Trie c v)) (Maybe v)

lookup :: Ord c => Trie c v -> [c] -> [v]
lookup trie list = catMaybes $ go string list
where
  go []          (T _   mr)
    = mr : []
  go (c:cs) (T ts mr)
    = mr : go cs (findWithDefault empty c ts)
```

Matching

```
match :: Trie Char Score -> String -> Score
match trie =
  merge2 . map (merge1 . lookup trie) . tails
where
  merge1, merge2 :: [Score] -> Score

  merge1 []     = repeat 0
  merge1 XSS   = map maximum $ transpose XSS

  merge2 []           = []
  merge2 ((h:t):l) = h:zipWith max t (merge2 l)
```

Processing

```
process patterns words = map go words
```

where

```
  go w = score w $ match trie $ '.' : w ++ ". "
```

```
  trie = foldl ins empty patterns
```

```
  ins trie pattern =
```

```
    insert (toText pattern) trie
```

```
    (toScore pattern)
```

```
toText word = filter (not . isDigit) $ word
```

```
toScore []      = repeat 0
```

```
toScore (p:ps)
```

```
| isDigit p  = read [p] : toScore (drop 1 ps)
```

```
| otherwise  = 0           : toScore ps
```

I/O

```
main  =
  do ls <- getContents >>= return . lines
     let (patterns,_:words)
         = span (\w -> head w /= '-') ls
     putStrLn $ unlines $ process patterns words

score :: String -> Score -> String
score cs (_:_:ns) = go cs ns
  where
    go [c]      [_,_]          = [c]
    go (c:cs) (n:ns) =
      c : if odd n then '-' : go cs ns
           else                 go cs ns
```

Conclusion



Hasselt



Haskell



Thank You