j1k21

24 Almost absolutly Perfect but Still Brilliant! welldone

Tests (1/1)

Section A
Username: jlk21
Compilation: 1 / 1

HIT_2022 Page 1

```
COMPAGGACHIANG LAZA
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           82: | otherwise = []
83: encode' xs (Node _ tl t2) path
84: | null pathl = path2
85: otherwise = path1
86: where
87: pathl = encode' xs tl (path ++ [0]) (an just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just (oncat Sine path)2 = encode' xs tl (path ++ [1]) (on just encode code tl xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode encode' xs tl (path ++ [1]) (on just encode e
    1: module Compression where
2:
3: Impact Data.List
4: Impact Data.Char
5:
6: data HTree a = Leaf Int a | Node Int (HTree a) (HTree a)
7: deciving (Show)
           d: instance Eq (HTree a) where
d: t1 == t2 = freqCount t1 == freqCount t2
  10:
11:
12: instance Ord (HTree a) where
13: t1 <- t2' = freqCount t1 <- freqCount t2'
                   type Code - [Int]
    16:
17: freqCount :: HTree a -> Int
18: freqCount (Leaf n a)
19: = n
20: freqCount (Node n t1 t2)
21: = n
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                _ - wecome bs t2

compressTree :: BTree Char -> [Int]
compressTree (Leaf _ x)
= 1 : intToBitstring (ord x)
where
                   testString :: String
testString = "mississippi is missing"
  23: two sississippl as 25: "Mississippl as 26: 27: - 28: - Example Huffman coding tree from the spec.
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 -- Assume the character does not have ordinal 0, i.e. it is not \NUL intToBitstring 0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                count :: Eq a \Rightarrow a \rightarrow [a] \rightarrow Int count x xs = length (filter \langle x == \rangle xs \rangle
                 = length (filter (x ==) xs) countAll :: Eq a => [a] \rightarrow [a, Int)] put rulo in the next one 2 countAll xs ys = nub (map \( \)x \rightarrow (x, count x ys)) xs\) buildfable :: Eq a => [a] \rightarrow [(a, Int)] buildfable xs = countAll xs xs
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              - Returns a pair where the first element is the R(Sub)Tree
- and the second element is the remaining bits after the N(Sub)Tree
rebuildTree' (b: bs)
| b == 1 = (Lesf 0 (chr (bitstringToInt hd)), tl)
| b == 0 = (Kode 0 lt rt, rest')
where
(hd, tl) = splitAt 7 bs
| lit, rest| = rebuildTree' bs
| lit, rest| = rebuildTree' rest
| bitstringToInt
| = foldi (\x y -> x * 2 + y) 0
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   = COURTAIN AS AS A PRITE A -> HITEE A -> HITEE A PRITE A PRITE
 58: n2 = freqCount t2 U
59:
60: reduce :: [NTeea a] → NTree a
61: -- Pre: The argument list non-empty and sorted based on the ordering function
62: -- In the Ord instance above.
63: reduce (t)
65: reduce (t)
65: reduce (insert (merge t) t2) ts)
67:
                    buildTree :: Eq a => [a] -> HTree a -- Pre: The list is non-empty
os: bolldiree :: Rq a > (a) - Firse 6

60: - First The list is non-empty

11 be reduce (sort (map (xx, n) = leaf n xl (buildTable xs)))

71: - First The tree can encode each of the (tens the list

75: encode xxs t

75: encode xxs t

77: where

77: where

78: encode (I = 1

80: encode' (x : xs) (leaf _ x') path
```