Mahjong Analyzer

Progress 3

Muhammad Jilan Wicaksono

November 10, 2025

Universitas Indonesia

Target dan Pencapaian

Mengimplementasikan fungsi penghitung shanten: 100%

- Mengidentifikasi apakah suatu tile merupakan bagian dari meld, pair, ataupun pmeld.
- Dapat menghitung meskipun ada meld yang sudah terbuka.
- Dapat menghitung untuk hand yang tidak standar.
- Optimisasi menggunakan memoization.

Link Commit

- https://github.com/WLan1707/mahjong-analyzer/commit/e691aec4f02b99c42f568873b3faacf42b7e9a44
- https://github.com/WLan1707/mahjong-analyzer/commit/9754714ae62ab10469f3a9ef9a12bba3aafe9848

Lesson Learned: Lazy Evaluation & Memoization i

- Kelompokkan Hand berdasarkan suit, lalu ubah menjadi bilangan asli.

```haskell
extractSuit ::  Suit -> HandCount -> Int
extractSuit suit handCount = encodeSuit [Map.findWithDefault 0 (Tile
suit n) handCount | n <- [1..tileCount suit]]
  where
    tileCount Honor = 7
    tileCount _     = 9
    encodeSuit counts = sum [c * (5 ^ i) | (i,c) <- zip [0..] counts]
```

Perhitungan shanten kini dilakukan per suit, dan hasil per-suit dapat dimemoisasi menggunakan struktur lazy agar evaluasi setiap state hanya terjadi sekali.

Lesson Learned: Lazy Evaluation & Memoization ii

Masalah: jika menggunakan list, indexing dilakukan secara linier.

- Buat infinite tree bilangan asli
- Karena fungsi bertipe `Int -> Int`, gunakan functor dari tree untuk mengubah fungsi shanten menjadi infinite tree juga, dengan node ke *i* bernilai `shanten i`.
- Karena disimpan dalam tree, mencari node *n* cukup dalam waktu $O(\log n)$.

```haskell
data Tree a = Tree (Tree a) a (Tree a)
instance Functor Tree where
    fmap :: (a -> b) -> Tree a -> Tree b
    fmap f (Tree l m r) = Tree (fmap f l) (f m) (fmap f r)

index :: Tree a -> Int -> a
index (Tree _ m _) 0 = m
index (Tree l _ r) n = case (n - 1) `divMod` 2 of
    (q,0) -> index l q
    (q,1) -> index r q

nats :: Tree Int
nats = go 0 1
    where
        go !n !s = Tree (go l s') n (go r s')
            where
                l = n + s
                r = l + s
                s' = s * 2
```

```haskell
memoTree :: ((Int -> a) -> Int -> a) -> Int -> a
memoTree f = memoTree_f
  where memoTree_f = index memo
        memo = fmap (f memoTree_f) nats
```