Mahjong Analyzer

Progress 4

Muhammad Jilan Wicaksono

November 20, 2025

Universitas Indonesia

Target dan Pencapaian i

Membuat fungsi pencari partisi 90%

- Memodifikasi fungsi validasi kemenangan menjadi pencari partisi yang bertipe bisa mengeluarkan semua partisi yang menang.
- Belum sempat mengoptimasi lagi seperti fungsi shanten.

Mengimplementasikan fungsi untuk mengecek yaku: 70%

- Mengimplementasikan fungsi yang menerima partisi dan mengeluarkan list semua yaku yang terpenuhi.
- Baru mengimplementasikan 17 dari sekitar 26 pola yaku yang ada. Kebanyakan adalah pola edge case yang langka.

Target dan Pencapaian ii

Membuat fungsi untuk menghitung skor: 80%

- Mengimplementasikan fungsi yang menerima partisi dan mengeluarkan nilai Fu dari tangan, yaitu skor minor.
- Menghitung total skor yang diperoleh berdasarkan pola yang dibuat dan nilai dari Fu, serta informasi eksternal seperti apakah menjadi dealer.

Target dan Pencapaian iii

Yaku Triple Mixed Sequence: Memuat 3 sequence dengan nilai sama tapi menggunakan 3 suit berbeda


```
ghci> testYaku (ctxRon (Tile Honor 1)) sanshoku s1
Right [2]
```

```haskell
-- Mixed Triple Sequence
sanshoku :: HandContext -> AgariHand -> Int
sanshoku _ (Standard kmelds _) =
    if anyStartSanshoku kmelds
      then if (not . any isOpen) kmelds then 2 else 1
      else 0
sanshoku _ _ = 0


anyStartSanshoku :: [KMeld] -> Bool
anyStartSanshoku kmelds =
    let starts =
          [ (suit,n)
          | KMeld (Sequence (Tile suit n)) _ <- kmelds
          , suit /= Honor
          ]
    in any (hasThreeSuits starts) [1..7]


hasThreeSuits :: [(Suit,Int)] -> Int -> Bool
hasThreeSuits starts n =
    all (`elem` starts)
      [ (Manzu, n), (Pinzu, n), (Souzu, n)]
```

Target dan Pencapaian iv

Yaku Pure Straight: Memuat sequence (1,2,3), (4,5,6), (7,8,9) dengan suit yang sama.


```
ghci> let s2 = "123456789m234p55s"
ghci> testYaku (ctxRon (Tile Pinzu 2)) ittsu s2
Right [2]
```

```haskell
-- pure straight

ittsu :: HandContext -> AgariHand -> Int
ittsu ctx (Standard kmelds _) =
    let startsBySuit =
          [ (suit, n)
          | KMeld (Sequence (Tile suit n)) _ <- kmelds
          , suit /= Honor ]

        grouped =
          [ [ n | (s',n) <- startsBySuit, s' == s ]
          | s <- [Manzu, Pinzu, Souzu] ]

        ittsuFound =
          any (\ns -> all (`elem` ns) [1,4,7]) grouped

    in fromEnum ittsuFound * (fromEnum (isMenzen ctx) + 1)

ittsu _ _ = 0
```

Link Commit

Link Commit

- https://github.com/WLan1707/mahjong-analyzer/commit/e948dcd73627e5b8a2a4945314042c5f6dc07741

Lesson Learned: Memoization Using Tree i

Secara konsep mirip seperti Fibonacci, jadi akan digunakan contoh ini saja:

$$F(0) = 1, F(1) = 1, F(n) = F(n-1) + F(n-2)$$

Tanpa memoization, dapat memanggil

```
fix fib 30 == 1346269
```

```haskell
import Data.Function (fix)

fib :: (Integer -> Integer) -> (Integer -> Integer)
fib f 0 = 1
fib f 1 = 1
fib f n = f (n - 1) + f (n - 2)
```

Lesson Learned: Memoization Using Tree ii

Dengan list, dapat dipanggil

```
faster_f 3000 == 90018235110926312 49
```

```haskell
f_list :: [Integer]
f_list = map (fib faster_f) [0..]

faster_f :: Integer -> Integer
faster_f n = f_list !! fromInteger n
```

Seperti yang dipelajari di kelas

```haskell
fib' = 1 : 1 : zipWith (+)fib' (tail fib')
```

Jika diperhatikan, `faster_f` memanfaatkan `fmap` dari Functor List dengan fungsi lookup `(!!)` sebesar $O(n)$.

Lesson Learned: Memoization Using Tree iii

```
Visualisasi f_tree

- nats:
        0
      /   \
     1     2
    / \   / \
   3  4  5   6

- f_tree:
          g 0
        /   \
      g 1     g 2
     /   \   /   \
   g 3  g 4  g 5  g 6
dengan g = fib fastest_f
sehingga
g n = fib fastest_f
    = fastest_f (n - 1) + ( fastest_f (n-2) )
    = index f_tree (n - 1) + ( index f_tree (n - 2) )
dan
fastest_f n = index f_tree n
            = g n
```

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

index :: Tree a -> Integer -> 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 Integer
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

f_tree :: Tree Integer
f_tree = fmap (fib fastest_f) nats

fastest_f :: Integer -> Integer
fastest_f = index f_tree
```

`f_tree` dibuat dulu secara lazy, ketika `fastest_f n` dipanggil, hanya perlu melihat node ke $(n-1)$ dan $(n-2)$ dari `f_tree` menggunakan index, dan keduanya juga hanya melihat kode sebelumnya saja juga, dan seterusnya.

Lesson Learned: Memoization Using Tree iv

Dengan ini, dapat dipanggil bilangan fibonacci ke 70000 secara cepat.

```
ghci> fastest_f 70000
98700678144192191533879964601875439104642052643022303492799984683869674657420320806903396491025367416116741711939696149813658285363870339312585789009409389443966311558131599113563078948482498257066465314598370326114892881006750982647528643744630372732993790497233641159651072992703200330630701154712890212431460585214311795014368842930234106739167043520670218209432507219476218350134789956937385638775542104727649273961331058421531681678906751304015662329326274895366260915346289761924931556766822355340373545112325253610629207157245394243955521304127657964294645915975022275314249662715224852963227478821181371492881750681936479939281203417476717418711719655404564237889866439422077698364525063306467072534258645848473197360632723234610661240785884086060444888670407174651351987639596048347207324822844676843058354979875050216042974927730700007975116183504316034256871296577366914923259303724029814928911212418407684689906072269525955319357582045908531951043007944296792436933209142632839219814456199110840613414205313982952206222283593929662590848181092585599223029999826169171333933188592104754227378086189210465991083610880268741931473020482014011198139398465599203028705242970325589969986504859210390241013268130129562968394481047259747450657579346541928438581430556589841859562484749985634307391088385087803751419268755256238895845111786906702018556891009346572479246906881396198716014232397286256276727611801200460695078858883778054771802849071594644362938528048988081262396733963562795944417840963094213058083259478947496083269315781431370913808360697347992190382863946211582636958392163752843665268364876286714319177083712799808655149108220046251739361900188993957187506559854095089169146987534092946500156674553316610055914010556075028375674641312757056873984237881334235484550015181872339517629970193343311352826792948112108693234019246409244536027676611998765794942491281695988367500509983534339301325439295477841247730800791495291812827472515170385978538341063854512604987009493198544951633401352845805320856902427522205183683911196532489372031091720911059263907379199621054751867280766797468368512821782561758525394078035876500443066001070646866462587264269482338961735178027588253504701539480758859715551251221405524020858244118090467591293829213876408535273397624105821819393917384348559494624791949371970498192532018172588752875555222731479461829445691757071462483524043356928361923607157448917714895079328298738904806373418527567161328560694322632638975959985142024331103181779435662184753595881756807496696774708883997310554270596264556891533581540650374382250325667779877907107550512411607665678201347465942015405251394856980829977364723203032013118481760074163924878317917422845652979617934222476844922083946397508734295917198450310835515608711701819856644290808280136641776599591784838441446630200663430513419067580145163202770298921899537544811842798804415097995872352191062528589710425456046574972587430446348327106917782975868548015005327664579332667591258109796463072637009884126623394068515022028925282659630606347042583962515007057638041285829785791558051377647023172999719694519409502745223145754587369621524546235440621171287451649235051358877283669064175774395061406765469986069629831216653303072357579010320011607132499330494301587391716963233322154330450833888377605800942147885861947010194902079759321382905855087972806055926468257899097296290369988005448300608807805242236251651615676976031235923720123455271689751409440176224644168132555673089729505841379888593467634506618208514076904526277044445620203989498291415082983005619659228859905039194978847900319487513976931665841487068899424504749244804597101394216173918960647253865138567936397417733030198099737005333562267812095954850882852480189336823808517153577987313084542095630631794153200970079663770920822332996888584320348105531480663801926573930959887926657163944112137722416053508909900575201657700298599463295975823961224487225988543471519649238383769642909570466229971046305892802082950374034419740066761780651326494318388539526058950823844146561551397427571920472535382532332019188499831952970136835623156482645210915163813941504314704059351824406368027732528215491558989160365577511479516006689134977341354680792890295489836083866776698517704403417726157015713106477110490304110552193389919905510631765007046210785240115277204418678353914232129984164445019729462819976156921814233322022364219780795060880991170409589006917718347310086573767930263903874960208530252276561508999439632985439051209900594865313063243008143087310201842306739341474250117726063677853024436783816321062034238666614941067334573381164678026707273947742538490087390234756052425142921563978708205876932777675048113828844188366892108822282232845939969106381309302071810061812383640751358151714958385541011242904790040382708080781860370631363994130569128456077083924064593720664731193340758573747548814538136259423580942655245157830793097393259565060099014482365033424734951735287822794481645423889885756841872019479014557232531725051403556005211225219302456970564754775941768374933566028302977756111432867009742696176201074361107529500049061366992040869275474145022284472454360413736380847007733873108169133251184514676050781035989154128087261427587466551942534540195723265372969079251449320635208856512643295711240765314442674327312540843344270840973462640896253713925586112807575784498378198375450950660070397118240631769515254928330459521117853715614076050139583658018714200491817142880835458376202468268539225200381474647464456405915101231593774107973709341085878663042391060340564111086359257843974599191406126260230470286090562894013324644924156062805529781946011971746794709356939963265649395558257770264973305968541294885492772914983140708831463178580116073420595507502242015715725547820870111265650714554078963180525881851033875012908547638441698734372641087316664142852266225322829081048315200357001410727447602959064552321820346825996095049066324032457386169757711385367535050856153872457538888104402087717954829585884640153764774159277764858392590753155134391922676028424652459046852175237367049504452509472808195536819773792959004358297353161326086823872955253194047582020469699093554970725258345905037678645506745860588437690185000902021159953270814187120734303387595579989691820940143556136547853007110007722502232714207526892032790058253130640040772081748561261854833585905502207175860662186441280459733697467847802873932400197678927080368437035023519191380031856999887744875417098924478196767354100939700165904987223996004720313660423917578697006465620331044919678330674537322477330230164820476733847059766575014577394305654971615524008600
```