```
Haskell test run started Tue Sep  5 16:40:57 AEST 2017
Proj1 testing
                          Test   1 ... PASSED 4.0
                          Test   2 ... PASSED 3.0
                          Test   3 ... PASSED 5.0
                          Test   4 ... PASSED 2.0
                          Test   5 ... PASSED 4.0
                          Test   6 ... PASSED 5.0
                          Test   7 ... PASSED 3.0
                          Test   8 ... PASSED 4.0
                          Test   9 ... PASSED 6.0
                          Test  10 ... PASSED 6.0
                          Test  11 ... PASSED 5.0
                          Test  12 ... PASSED 4.0
                          Test  13 ... PASSED 4.0
                          Test  14 ... PASSED 5.0
                          Test  15 ... PASSED 4.0
                          Test  16 ... PASSED 5.0
                          Test  17 ... PASSED 5.0
                          Test  18 ... PASSED 4.0
                          Test  19 ... PASSED 3.0
                          Test  20 ... PASSED 4.0
                          Test  21 ... PASSED 4.0
                          Test  22 ... PASSED 2.0
                          Test  23 ... PASSED 5.0
                          Test  24 ... PASSED 4.0
                          Test  25 ... PASSED 4.0
                          Test  26 ... PASSED 4.0
                          Test  27 ... PASSED 4.0
                          Test  28 ... PASSED 3.0
                          Test  29 ... PASSED 3.0
                          Test  30 ... PASSED 4.0
                          Test  31 ... PASSED 3.0
                          Test  32 ... PASSED 5.0
                          Test  33 ... PASSED 3.0
                          Test  34 ... PASSED 5.0
                          Test  35 ... PASSED 3.0
                          Test  36 ... PASSED 3.0
                          Test  37 ... PASSED 5.0
                          Test  38 ... PASSED 4.0
                          Test  39 ... PASSED 4.0
                          Test  40 ... PASSED 4.0
                          Test  41 ... PASSED 3.0
                          Test  42 ... PASSED 4.0
                          Test  43 ... PASSED 4.0
                          Test  44 ... PASSED 4.0
                          Test  45 ... PASSED 5.0
                          Test  46 ... PASSED 4.0
                          Test  47 ... PASSED 3.0
                          Test  48 ... PASSED 4.0
                          Test  49 ... PASSED 4.0
                          Test  50 ... PASSED 4.0
                          Test  51 ... PASSED 4.0
                          Test  52 ... PASSED 7.0
                          Test  53 ... PASSED 6.0
                          Test  54 ... PASSED 4.0
                          Test  55 ... PASSED 4.0
                          Test  56 ... PASSED 5.0
                          Test  57 ... PASSED 5.0
Test  58 ... PASSED 4.0
Test  59 ... PASSED 4.0
Test  60 ... PASSED 4.0
Test  61 ... PASSED 3.0
Test  62 ... PASSED 4.0
Test  63 ... PASSED 5.0
Test  64 ... PASSED 5.0
Test  65 ... PASSED 6.0
Test  66 ... PASSED 3.0
Test  67 ... PASSED 5.0
Test  68 ... PASSED 4.0
Test  69 ... PASSED 4.0
Test  70 ... PASSED 5.0
Test  71 ... PASSED 5.0
Test  72 ... PASSED 5.0
Test  73 ... PASSED 5.0
Test  74 ... PASSED 4.0
Test  75 ... PASSED 5.0
Test  76 ... PASSED 3.0
Test  77 ... PASSED 5.0
Test  78 ... PASSED 5.0
Test  79 ... PASSED 4.0
Test  80 ... PASSED 4.0
Test  81 ... PASSED 4.0
Test  82 ... PASSED 3.0
Test  83 ... PASSED 3.0
Test  84 ... PASSED 5.0
Test  85 ... PASSED 4.0
Test  86 ... PASSED 3.0
Test  87 ... PASSED 5.0
Test  88 ... PASSED 3.0
Test  89 ... PASSED 5.0
Test  90 ... PASSED 4.0
Test  91 ... PASSED 3.0
Test  92 ... PASSED 5.0
Test  93 ... PASSED 5.0
Test  94 ... PASSED 5.0
Test  95 ... PASSED 5.0
Test  96 ... PASSED 5.0
Test  97 ... PASSED 5.0
Test  98 ... PASSED 5.0
Test  99 ... PASSED 5.0
Test 100 ... PASSED 6.0
Test 101 ... PASSED 5.0
Test 102 ... PASSED 4.0
Test 103 ... PASSED 2.0
Test 104 ... PASSED 3.0
Test 105 ... PASSED 4.0
Test 106 ... PASSED 4.0
Test 107 ... PASSED 4.0
Test 108 ... PASSED 3.0
Test 109 ... PASSED 4.0
Test 110 ... PASSED 4.0
Test 111 ... PASSED 3.0
Test 112 ... PASSED 4.0
Test 113 ... PASSED 4.0
Test 114 ... PASSED 4.0
Test 115 ... PASSED 5.0
Test 116 ... PASSED 5.0
```

```
                          Test 117 ... PASSED 4.0
                          Test 118 ... PASSED 5.0
                          Test 119 ... PASSED 3.0
                          Test 120 ... PASSED 5.0
Total tests: 120.0
Tests successfully guessed: 120.0
Total guesses for successful tests: 503.0
Average guesses: 4.191666666666666
Points available: 70.0 * 120.0 / 120.0 = 70.0
Points: 70.0 / 70.0
Haskell test run ended Tue Sep  5 16:40:59 AEST 2017
Total CPU time used = 2117 milliseconds
```

```
-- COMP30020 Declarative Programming Project 1 ChordProbe code, written by
-- Hongfei Yang 783661 <hongfeiy1@student.unimelb.edu.au>.
--
-- The code implements mini-max techique, trying to guess the correct chord
-- using as few guesses as possible. After each guess, the next guess is made
-- based on the feedback of the previous guess.

module Proj1 (initialGuess, nextGuess, GameState) where

import Data.List

_MAX = 1330 -- max number of guesses one can remove
_MIN = (-1) -- min score of mini-max, which will never happen :)

-- GameState consists a list of possible combinations, each combination is a
-- list of three distinct strings, with one letter from A to G and a number
-- from 1 to 3
type GameState = [[String]]

-- Generate all 1330 combinations
initialGuess:: ([String], GameState)
initialGuess = (firstGuess, allCombi)
    where
        firstGuess = ["A1", "B1", "C2"]
        allPitch = [a:b:[] | a <- [’A’,’B’..’G’], b <- [’1’,’2’..’3’]]
        allCombi = [[a,b,c] | a <- allPitch, b <- allPitch, c <- allPitch,
         a < b, b < c]

-- Calculate the score of two combinations
getScore :: [String] -> [String] -> (Int, Int, Int)
getScore guess target = (cPitchCount, cNoteCount, cOctaveCount)
    where
        -- get the list of correct pitches
        cPitchList = intersect guess target

        -- then get the number of the correct pitches
        cPitchCount = length cPitchList

        -- List of the remaining incorrect pitches
        remainTarget = target \\ cPitchList
        wrongGuess = guess \\ cPitchList

        -- Get the second and third digit in the score (x,x,x)

        remainTargetNote = getRemainNote remainTarget
        remainTargetOctave = getRemainOctave remainTarget
        remainGuessNote = getRemainNote wrongGuess
        remainGuessOctave = getRemainOctave wrongGuess

        -- Number of correct note in the incorrect pitches guessed
        cNoteCount = length remainTargetNote - (length (remainTargetNote
         \\ remainGuessNote))

        -- Number of correct octave in the incorrect pitches guessed
        cOctaveCount = length remainTargetOctave - (length (remainTargetOctave
         \\ remainGuessOctave))

        -- Extract the list of incorrect note guessed from the incorrect
        -- pitches
        getRemainNote :: [String] -> [Char]
        getRemainNote [] = []
        getRemainNote (onePitch:rest) = onePitch!!0 : getRemainNote rest

        -- Extract the list of incorrect octave guessed from the incorrect
        -- pitches
        getRemainOctave :: [String] -> [Char]
        getRemainOctave [] = []
        getRemainOctave (onePitch:rest) = onePitch!!1 : getRemainOctave rest

-- Make the next guess based on the feedback of the previous guess
nextGuess :: ([String], GameState) -> (Int, Int, Int) -> ([String], GameState)
nextGuess (lastGuess, prevReducedSet) score = (nGuess, nextReducedSet)
    where
        -- Reduce the size of the possible candidates based on the feedback,
        -- This is to remove all combinations that do not have the same score
        -- as if the guess is the actual target
        nextReducedSet = reduce lastGuess prevReducedSet score

        -- The next guess is made by choosing the maximum of the minimum
        -- number each guess can clear as if it is the guess. Initially
        -- the current minimum will be set to be a large number
        nGuess = miniMax nextReducedSet nextReducedSet [] _MAX

-- Reduce the number of possible candidates by only keeping the ones that has
-- the same score
reduce :: [String] -> [[String]] -> (Int, Int, Int) -> [[String]]
reduce target [] targetScore = []
reduce target (candidate:rest) targetScore
    | getScore candidate target == targetScore = candidate : reduce
     target rest targetScore
    | otherwise = reduce target rest targetScore

-- Group the scores and their occurence frequencies, by comparing a given
-- combination against a set of combinations
groupByScore :: [String] -> [[String]] -> [((Int, Int, Int), Int)]
groupByScore target [] = []
groupByScore target (oneGuess:rest) = updateScore (getScore oneGuess target)
 (groupByScore target rest)

-- Update the frequency of occurence of a score in a dictionary of scores
-- frequencies
updateScore :: (Int, Int, Int) -> [((Int, Int, Int), Int)] ->
    [((Int, Int, Int), Int)]
updateScore newScore [] = [(newScore, 1)]
updateScore newScore ((currScore, count):rest)
    | newScore == currScore = ((currScore, count+1):rest)
    | otherwise = (currScore, count):(updateScore newScore rest)

-- Get the maximum number of occurence of a given score group
getMaxCount :: [((Int, Int, Int), Int)] -> Int -> Int
getMaxCount [] currMax = currMax
getMaxCount ((score, count):rest) currMax
    | count > currMax = getMaxCount rest count
    | otherwise = getMaxCount rest currMax

-- Apply mini-max technique to get the maximum of the minimum number each guess
-- can clear as if it is the guess, then get the guess with that can clear the
-- largest number of the number of minimum guess it can clear in the given set.
```

```
miniMax :: [[String]] -> [[String]] -> [String] -> Int -> [String]
miniMax [] _ currMinGuess currMinScore = currMinGuess
miniMax (thisGuess:restGuess) reducedSet currMinGuess currMinScore
    | thisScore < currMinScore =
      miniMax restGuess reducedSet thisGuess thisScore
    | otherwise = miniMax restGuess reducedSet currMinGuess currMinScore
    where
      thisScore = getMaxCount (groupByScore thisGuess reducedSet) _MIN
```