```
Beginning tests for haonanl5 Sat Apr 14 07:28:36 AEST 2018

Compiling sources
ghc -O2 --make Project2Test -o Project2Test
[1 of 2] Compiling Project2         ( Project2.hs, Project2.o )
[2 of 2] Compiling Main             ( Project2Test.hs, Project2Test.o )
Linking Project2Test ...

Running a few simple tests
Your guess 1:  ["BP","BP","BP","BP"]
My answer:  (0,0,2)
Your guess 2:  ["BN","BN","WN","WN"]
My answer:  (1,0,2)
Your guess 3:  ["BB","BN","WB"]
My answer:  (1,0,2)
Your guess 4:  ["BR","BN","WR"]
My answer:  (1,0,2)
Your guess 5:  ["BQ","BN","WQ"]
My answer:  (3,0,0)
You got it in 5 guesses!

Your guess 1:  ["BP","BP","BP"]
My answer:  (0,3,0)
Your guess 2:  ["WP","WP","WP"]
My answer:  (3,0,0)
You got it in 2 guesses!

Your guess 1:  ["BP","BP","BP","BP"]
My answer:  (0,0,1)
Your guess 2:  ["BN","WB","WN","WN"]
My answer:  (0,0,2)
Your guess 3:  ["BR","WR"]
My answer:  (1,0,1)
Your guess 4:  ["BR","WQ"]
My answer:  (1,0,1)
Your guess 5:  ["BR","WK"]
My answer:  (2,0,0)
You got it in 5 guesses!

Your guess 1:  ["BP","BP","BP","BP","BP"]
My answer:  (0,0,3)
Your guess 2:  ["BB","BN","BN","WN","WN"]
My answer:  (1,0,2)
Your guess 3:  ["BR","BR","BN"]
My answer:  (2,0,1)
Your guess 4:  ["BQ","BR","BN"]
My answer:  (1,0,2)
Your guess 5:  ["BR","BR","BB"]
My answer:  (3,0,0)
You got it in 5 guesses!

Your guess 1:  ["BP","BP","BP"]
My answer:  (0,0,3)
Your guess 2:  ["BB","BN","BN"]
My answer:  (1,0,2)
Your guess 3:  ["BR","BR","BN"]
My answer:  (2,0,1)
Your guess 4:  ["BQ","BR","BN"]
My answer:  (1,0,2)
```

```
Your guess 5:  ["BR","BR","BB"]
My answer:  (3,0,0)
You got it in 5 guesses!


Running formal tests with hidden results
Completed tests Sat Apr 14 07:28:38 AEST 2018
```

```haskell
-- Author   : Haonan Li <haonanl5@student.unimelb.edu.au>
-- Purpose  : Implement the guessing part of a logical guessing game.


-- Introduction of ChessGuess: ChessGuess is a two-player logical guessing
-- game. one player is hider and the other is guesser. The hider begins by
-- selecting the size of the game from 0 to 32. And then selects up to size
-- chess pieces out of a chess set and hides them. Once the hider has selected
-- the target set of pieces, the guesser repeatedly chooses a subset of chess
-- pieces and tells it to the hider, who responds by giving the guesser three
-- numbers indicate the number of correct guess pieces, right kind but wrong
-- colour pieces, and right colour but wrong kind pieces separatly. This
-- program complete the guesser part.

-- Introduction of Code: We first guess all "BP" with game size. from the
-- feedback we know the number of "BP" and "WP" and black pieces except "BP".
-- Then we build a candidate set with all possible target with exact number of
-- "BP", "WP" and black pieces. The we always choose a set with maximum number
-- of pieces as new guess. And compare the feedback with our own judgement,
-- remove the candidate does not match.


module Project2 (initialGuess, nextGuess, GameState) where
import Data.List

-- Function : Save times of guesses and the candidate target set.
type GameState = (Int, [[String]])

-- Function : Initial guess, guess all "BP" with the size of the game.
-- Input    : One number.
-- Output   : One guess set and a gamestate tuple.
initialGuess :: Int -> ([String],GameState)
initialGuess size = (gs, (1,[])) where
    gs = replicate size "BP"


-- Function : Return a element with longest length in a list
-- Input    : A empty list and a list (L) of list of String.
-- Output   : A list of String whoes length is the largest of the given list L.
longest :: [[String]] -> [String]
longest [a] = a
longest (x:y:z)
    | length x > length y = longest (x:z)
    | otherwise           = longest (y:z)


-- Function : Decide the next guess. We process the second guess seperatly.
--            For the nth guess (n>2). We always judge every candidate if the
--            result is the same with the feedback received from hinder, keep
--            it in candidate target set. Then, we randomly choose one set with
--            maximum number of pieces from the candidate set as new guess.
-- Input    : A tuple of [String] and GameState, and a tuple of three integers.
-- Output   : A tuple of [String] and GameState, They are new guess and updated
--            game state seperately.
nextGuess :: ([String],GameState) -> (Int,Int,Int) -> ([String],GameState)
nextGuess (bgs,(nth, bcand)) guess_res
    | nth == 1  = secondGuess (bgs,(nth, bcand)) guess_res
    | otherwise = (gs,((nth+1),cand)) where
        cand = filter (sameRes bgs guess_res) bcand
```

```haskell
        gs   = longest cand


-- Function : The second guess, From initial guess,  We initialize
--            candidate target set with these informations. And find one with
--            largest length as the second guess.
-- Input    : A tuple of [String] and GameState, and a tuple of three integers.
-- Output   : A tuple of [String] and GameState, They are new guess and updated
--            game state seperately.
secondGuess :: ([String],GameState) -> (Int,Int,Int) -> ([String],GameState)
secondGuess (bgs,(nth, bcand)) (t1,t2,t3) = (gs,((nth+1),cand)) where
    pieces1 = ["BK","BQ","BR","BR","BB","BB","BN","BN"]
    pieces2 = ["WK","WQ","WR","WR","WB","WB","WN","WN"]
    cand    = map ((++) ((replicate t1 "BP") ++ (replicate t2 "WP")))
        [ x ++ y |
            x <- (filter (prune t3) (subsequences pieces1)),
            y <- (filter (prune max_n_white) (subsequences pieces2))] where
                max_n_white = ((length bgs)-t1-t2-t3)
    gs      = longest cand


-- Function : A filter, tell if the list length no larger than the given number.
-- Input    : A number N and a list L.
-- Output   : If the length of L no larger than N
prune :: Int -> [String] -> Bool
prune n a = n >= length a


-- Function : A filter, compare the result of my own judgement system with the
--            tuple received from hinder. If they are the same, retain the set
--            as candidate, if not, delete it from candidate set.
-- Input    : A guess list and tuple of three integers, and a target list.
-- Output   : A Bool value, indicate whether given guess and target can compute
--            the given answer.
sameRes :: [String] -> (Int,Int,Int) -> [String] -> Bool
sameRes bgs (t1,t2,t3) x
    | myJudge x bgs == (t1,t2,t3) = True
    | otherwise                   = False


-- Function : Compute the size of common element, right kind of guess and right
--            color of guess.
-- Input    : A guess list and a target list.
-- Output   : A tuple with three integers, indicate the number of same elements,
--            same kind but different color elements and same color but
--            edifferent kind lements.
myJudge :: [String] -> [String] -> (Int,Int,Int)
myJudge cand gs = (n_same, n_same_k, n_same_c) where
    same      = myIntersect gs cand
    n_same    = length same
    n_same_c  = length (myIntersect (map (!!0) gs) (map (!!0) cand)) - n_same
    n_same_k  = length (myIntersect (map (!!1) gs) (map (!!1) cand)) - n_same


-- Function : Computer the insetsection of two lists.
-- Input    : Two list L1, L2.
-- Output   : One list of the same elements in L1 and L2.
myIntersect :: Eq a => [a] -> [a] -> [a]
myIntersect [] _ = []
```

```
myIntersect (x:xs) a
    | x ‘elem‘ a = x : myIntersect xs (delete x a)
    | otherwise  = myIntersect xs a
```