Balanced Partition in Haskell

Jacob O'Keeffe (13356691), Dublin City University 13/12/2015

Plagiarism Declaration

I hereby declare that the following work is my own. I am aware of DCU's regulations in regards to plagiarism and I acknowledge the consequences should I breach those regulations.

Design of Solution

Given a list of numbers A, where the sum of A (lets call this s) is even, a perfectly balanced partition would be ($S1$, $S2$) where the sums of both $S1$ and $S2$ are equal to $s \div 2$. If s is odd, then we're looking for a partition of A whose sum is equal to $(s - 1) \div 2$. In Haskell, both of these cases can be covered using the quot function which carries out "integer division truncated towards 0". Lets call this this value $target$.

If an element of A is equal to $target$ then it is also equal to the sum of all the other elements of A combined and should therefore form $S1$ by itself. If I first sort and then reverse A so that its elements are in descending order, then only the first element could satisfy this clause.

With that special case out of the way, I want to follow an algorithm until I have created a balanced partition of A. Each iteration, if the head of A is less than $target$, I'll add it to $S1$ and subtract it from $target$ to get my new $target$. I'll then call the function on the tail of A.

This approach avoids all the unnecessary computation required to generate all possible combinations of A.

Implementation

```
1 -- takes a list xs and returns a tuple of balanced partitions of xs
2 balancedPartition :: [Int] -> ([Int],[Int])
3 balancedPartition [] = ([],[])
4 balancedPartition xs = (xxs, (otherPartition xs xxs))
5   where
6     xxs = bPartition xs
```

The balancedPartition function itself, takes a list of integers and produces a tuple containing two balanced partitions. It calls bPartition on xs which returns one balanced partition as xxs. It then calls otherPartition on xs and xss which returns xs less all the elements in xxs. These are our two balanced partitions.

```haskell
-- takes a list xs and a list ys
-- returns xs less the elements from ys
otherPartition :: [Int] -> [Int] -> [Int]
otherPartition xs []     = xs
otherPartition xs (y:ys) = otherPartition (delete y xs) ys
```

bPartition is used to pass arguments to bPartition'. It sorts and then reverses the list it receives from balancedPartition and also passes a target value which is calculated using the quotient function.

Given a list of integers in descending order along with a target, bPartition will create a partition of the list whose sum is as close to target as possible. It does this by finding the next largest value in the list which is less than target and adding it to the head of a new list to be returned. The tail of this new list is created using bPartition' on the tail of the earlier list with its new target.

```haskell
-- sorts the elements in reverse order for use with bPartition'
-- this new list is passed, along with a target value, to bPartition'
bPartition :: [Int] -> [Int]
bPartition [] = []
bPartition xs = bPartition' (reverse $ sort xs) target
  where
    target = (sum xs) `quot` 2

-- calculates one of the partitions
bPartition' :: [Int] -> Int -> [Int]
bPartition' [] _           = []
bPartition' _ 0            = []
bPartition' (x:xs) target
  | x == target            = [x]
  | x < target             = x : bPartition' xs (target - x)
  | otherwise              = bPartition' xs target
```