99 questions/Solutions/23

From HaskellWiki

< 99 questions | Solutions

Extract a given number of randomly selected elements from a list.

```
import System.Random
import Control.Monad (replicateM)

rnd_select :: [a] -> Int -> IO [a]
rnd_select [] _ = return []
rnd_select l  n
    | n<0 = error "N must be greater than zero."
    | otherwise = do pos <- replicateM n $
                                getStdRandom $ randomR (0, (length l)-1)
                     return [l!!p | p <- pos]
```

In order to use getStdRandom and randomR here, we need import module System.Random.

A more elegant solution using randomRs:

```
rnd_select xs n = do
    gen <- getStdGen
    return $ take n [ xs !! x | x <- randomRs (0, (length xs) - 1) gen]
```

or using sequence all the way:

```
import Control.Monad (replicateM)
rnd_select xs n
    | n < 0     = error "N must be greater than zero."
    | otherwise = replicateM n rand
        where rand = do r <- randomRIO (0, (length xs) - 1)
                        return (xs!!r)
```

Alternative solution:

The original Lisp problem suggested we use our solution from problem 20. I believe that each item from the list should only appear once, whereas the above solution can reuse items.

Therefore here is an alternative which uses the "removeAt" function from problem 20:

```
rnd_select :: RandomGen g => [a] -> Int -> g -> ([a], g)
rnd_select _ 0 gen = ([], gen)
rnd_select [] _ gen = ([], gen)
rnd_select l count gen
   | count == (length l) = (l, gen)
```

```
   | otherwise           =  rnd_select (removeAt l (k+1)) count gen'
                          where (k, gen') =
                                  randomR (0, (length l) - 1) gen

rnd_selectIO :: [a] -> Int -> IO [a]
rnd_selectIO l count = getStdRandom $ rnd_select l count
```

If the number of items we want is the same as the number of items in the list, then we just return the list. Otherwise we remove a random item from the list and then recurse.

Another Alternative Solution:

Since the above Alternative Solution works by removing things to create the target list, it's most efficient when the target list length is > (orig list / 2). Here's another solution that's efficient in the other way (target < (orig list / 2)) by constructing an accumulator list of selected random elements. (This one also uses removeAt from problem 20)

```
rnd_select :: RandomGen g => [a] -> Int -> g -> ([a], g)
rnd_select ol ocount ogen = rnd_select' ol [] ocount ogen
   where
      rnd_select' l acc count gen
         | count == 0 = (acc, gen)
         | otherwise   = rnd_select' (removeAt l (k+1)) ((l !! k) : acc)
                           (count - 1) gen'
                         where (k, gen') =
                                 randomR (0, (length l) - 1) gen

rnd_selectIO :: [a] -> Int -> IO [a]
rnd_selectIO l count = getStdRandom $ rnd_select l count
```

An O(N) algorithm:

```
import System.Random (randomRIO)
rnd_select :: [a] -> Int -> IO [a]
rnd_select _ 0 = return []
rnd_select (x:xs) n =
    do r <- randomRIO (0, (length xs))
       if r < n
           then do
               rest <- rnd_select xs (n-1)
               return (x : rest)
           else rnd_select xs n
```

A solution returns random results even when the number of items we want is the same as the number of items in the list:

```
import System.Random(randomRIO)

rnd_select :: [a] -> Int -> IO [a]
rnd_select _  0 = return []
rnd_select [] _ = return []
rnd_select xs count = do r <- randomRIO (0, (length xs)-1)
                         rest <- rnd_select (removeAt xs (r+1)) (count-1)
                         return ((xs!!r) : rest)
```

A very simple and elegant solution which uses "nub" function from Data.List.

```
rnd_select :: Int -> [a] -> [a]
rnd_select n x = map (x!!) is
 where is = take n . nub $ randomRs (0, length x - 1) (mkStdGen 100)
```

Note, all of these return IO [a]. Some recent version of GHCi added the ability to display

```
IO a
```

values at the top level. Hugs (and older GHCi) behaves differently, so an action to actually do IO (putStrLn in the example) is required.

The same solution using the global number generator and returning IO [a] (thanks to applicatives, $ is simply replaced by <$>):

```
rnd_select :: [a] -> Int -> IO [a]
rnd_select lst n = map (lst !!) <$> indices
    where indices = take n . nub . randomRs (0, length lst - 1) <$> getStdGen
```


Category:

- Programming exercise spoilers
