Assignment 10

due at 23:59 on Fri Dec 8 (80 points)

Question: Suppose we draw **three** cards from a shuffled deck of playing cards. What is the probability of getting a **straight**, a **flush**, or both?

You should answer this using the probability distribution monad from class on 30 November, save your implementation as `a10.hs`, and submit it to the dropbox at `https://www.dropbox.com/request/xz9LTmkMl6fCcCCLwZKR`.

Definitions

In case you're not familiar, I'll define all the situations needed here. Playing cards have both a **suit** and a **rank**. There are four suits, known as Hearts (♡), Spades (♠), Clubs (♣), and Diamonds (♢). There are thirteen ranks, known as the integers 2 through 10, and then Jack, Queen, King, and Ace. Every combination of suit and rank appears in a deck, so that's $4 \times 13 = 52$ cards.

When you have a small selection of cards (called a **hand**), you can recognize patterns in it. A **flush** means that all the cards in the hand have the **same suit.** So with three cards, if you drew 3♡, 9♡, A♡, that would be a flush. A **straight** means that all the cards in the hand have **consecutive ranks**. So with three cards, if you drew 9♢, 10♣, J♠, that would be a straight.

When you draw cards from a deck, you are selecting **without replacement.** So instead of each draw being an independent probability (one in 52), each draw constrains subsequent ones.

Also, for the purpose of determining straights and flushes, the *order* in which cards are drawn *doesn't matter.* In other words, to get a straight, you don't need to draw a 3, 4, 5 in that order; you could have drawn 5, 3, 4. It's still a straight.

We can further illustrate these principles with some numbers. If you were selecting something with a one-in-52 chance, three times independently (with replacement), the probability of any particular outcome would be $(\frac{1}{52})^3 = \frac{1}{140,608}$. But since we are not replacing cards into the deck after drawing them, it's actually $\frac{1}{52} \times \frac{1}{51} \times \frac{1}{50} = \frac{1}{132,600}$.

Furthermore, since the ordering of the cards you draw isn't significant, we actually multiply by the number of possible permutations of those three cards. So the chance of any particular set of three cards $\frac{1}{132,600} \times 3! = \frac{1}{132,600} \times 6 = \frac{1}{22,100}$.

Implementation tips

You should use custom data types to represent `Suit`, `Rank`, and `Card`. Derive or implement the `Show` type class to print them. They also should implement `Eq` and `Ord`, so you can use them in `Set` and `Map` structures.

Create lists of these types, enumerating all of the suits, ranks, and cards:

```
allSuits :: [Suit]
allRanks :: [Rank]
allCards :: [Card]
```

Hint: You don't have to write out 52 different elements for `allCards`; use a list comprehension! Something like this, assuming your `Card` type has a `Card` constructor taking rank and then suit:

```
allCards = [Card r s | r <- allRanks, s <- allSuits]
```

So far we're using lists, but decks and hands are better represented as `Set` structures. I defined these type aliases – they're the same thing (a set of cards) but I can use them in signatures to represent which set is the hand vs. the whole deck.

```
type Deck = S.Set Card
type Hand = S.Set Card
```

(The `S.Set` assumes you did `import qualified Data.Set as S`.) Then you can define the deck as a set and its size will be 52.

```
deck :: Deck
deck = S.fromList allCards
```

You can also use `S.toList` with the `oneOf` probability operator we defined previously:

```
ghci> S.size deck
52
ghci> oneOf (S.toList deck)
Prob {toList = [(Ace of Hearts,1 % 52),(Ace of Clubs,1 % 52),(Ace of
Spades,1 % 52),(Ace of Diamonds,1 % 52),(2 of Hearts,1 % 52),(2 of
Clubs,1 % 52),(2 of Spades,1 % 52),(2 of Diamonds,1 % 52),...
...
(King of Spades,1 % 52),(King of Diamonds,1 % 52)]}
```

In the probability distribution, each card is equally likely, with probability $\frac{1}{52}$.

Central to your calculations will be a definition like this:

```
drawThreeCards :: Prob Hand
```

It can use `oneOf` to draw from the initial deck, and then again to draw from a deck from which the initial card has been removed (`S.delete`). And then once again. It accumulates the drawn set of cards into a `Hand`. Use `optimize` to remove duplicates from the distribution. The result should be triples of cards, each with the probability $\frac{1}{22{,}100}$.

```
ghci> drawThreeCards
Prob {toList = [(fromList [Ace of Hearts,Ace of Clubs,Ace of Spades],1
% 22100),
............lots omitted............
(fromList [King of Clubs,King of Spades,King of Diamonds],1 % 22100)]}
```

This is a significant amount of computation, and may require a lot of memory, but it was reasonably fast on my laptop. The slowest part is printing out all the results. But whatever you do, **don't try to draw four or more cards!** Doing so slowed my computer to a crawl as everything I was doing got swapped out of the main memory. And I killed it before I got a result. This probability monad is flexible, but computing intensive as numbers get large. (It's basically simulating *everything* that can happen.)

Now that you've got a distribution of all 22,100 three-card hands, you want to look for straights and flushes. I wrote functions with these signatures:

```
isFlush :: Hand -> Bool
isStraight :: Hand -> Bool
```

The flush is the easiest. If your `Card` data type has a selector to extract the suit field (I called mine `cardSuit`), then you can basically do `S.map cardSuit hand`. Since the result is a set, each distinct suit in the original hand appears only once. So if the final size of that set is 1, you have a flush! Here's an illustration in pseudo-interpreter set notation:

- `S.map cardSuit` $\{3\heartsuit, \mathrm{J}\diamondsuit, 8\clubsuit\} \Rightarrow \{\heartsuit, \diamondsuit, \clubsuit\}$ (not flush)
- `S.map cardSuit` $\{4\heartsuit, \mathrm{A}\spadesuit, 5\heartsuit\} \Rightarrow \{\heartsuit, \spadesuit\}$ (not flush)
- `S.map cardSuit` $\{2\clubsuit, \mathrm{Q}\clubsuit, 5\clubsuit\} \Rightarrow \{\clubsuit\}$ (flush!)

Detecting a straight is a little more difficult. You can use `S.map cardRank hand` to extract only the ranks, and then `S.toAscList` to put them in ascending order. Then, figure out how to ensure there are three ranks with no gaps.

- `S.toAscList $ S.map cardRank` $\{4\spadesuit, 8\spadesuit, 5\diamondsuit\} \Rightarrow$ `[4,5,8]` (not a straight)
- `S.toAscList $ S.map cardRank` $\{2\heartsuit, 3\clubsuit, 2\diamondsuit\} \Rightarrow$ `[2,3]` (not a 3-card straight)
- `S.toAscList $ S.map cardRank` $\{2\heartsuit, 4\clubsuit, 3\heartsuit\} \Rightarrow$ `[2,3,4]` (straight!)

Finally, I paired these two Boolean results into a custom data type like this:

```
data Result = Straight | Flush | Both | Neither
  deriving (Show, Eq, Ord)

scoreHand :: Hand -> Result
scoreHand h =
  case (isStraight h, isFlush h) of
    (True, True) -> Both
    (True, False) -> Straight
    (False, True) -> Flush
    (False, False) -> Neither
```

So you should be able to calculate:

```
ghci> optimize (scoreHand <$> drawThreeCards)
Prob {toList = [(Straight, XXXXXX),
                (Flush, XXXXXX),
                (Both, XXXXXX),
                (Neither, XXXX)]}
```

I'm censoring the actual probabilities! What do you get?