Daily Coding Problem #238

Problem

This problem was asked by MIT.

Blackjack is a two player card game whose rules are as follows:

- The player and then the dealer are each given two cards.
- The player can then "hit", or ask for arbitrarily many additional cards, so long as their total does not exceed 21.
- The dealer must then hit if their total is 16 or lower, otherwise pass.
- Finally, the two compare totals, and the one with the greatest sum not exceeding 21 is the winner.

For this problem, cards values are counted as follows: each card between 2 and 10 counts as their face value, face cards count as 10, and aces count as 1.

Given perfect knowledge of the sequence of cards in the deck, implement a blackjack solver that maximizes the player's score (that is, wins minus losses).

Solution

Dynamic programming provides an elegant solution to this problem, but it is not trivial to implement.

We can think about the problem like this: suppose we start with a fresh deck of cards, and we deal out two cards each to the player and the dealer. At the end of the hand, the player may have hit twice, and the dealer may have hit once, so that in total 7 cards have been dealt. If we already know the best score the player can obtain starting with the 8th card of the deck, then the overall score for this play can be expressed as `value(play) + best_scores[8]`.

In fact, this hand may have been played differently- neither player may have hit, or both may have hit five times, so that the number of cards dealt may have been anywhere between 4 and 52. Each of these outcomes can be considered subproblems of our original problem that begin with an alternate starting index. If all these subproblems have been solved (that is, `best_scores[N]` is known for all `N > start`), we can express the best score of a play beginning at `start` as:

```
max([value(play) + best_scores[start + cards_used(play)] for play in plays])
```

So our dynamic programming approach will be as follows. For each starting index of the deck, beginning with the fourth-to-last card and going back to the first card:

- Simulate all the possible ways that a round of blackjack starting with that card can be played.
- For each play, compute its value (`1` for wins, `0` for ties, and `-1` for losses), and track the number of cards used.
- Set the value of `best_scores[start]` to be the result of the equation above.

Once we have our algorithm in place, we just need to add boilerplate code to represent the deck and the players, and to flesh out the logic for how the game is played.

In particular, `Deck` will be a class that starts with `52` randomly shuffled cards, and can deal out `n` cards beginning at a specified start index. The `Player` class will be initialized with two starting cards, and will hold cards in a `hand` that can be appended to and summed up.

The full implementation can be found below.

```
import random

class Deck:
    def __init__(self, seed=None):
        self.cards = [i for i in range(1, 10)] * 4 + [10] * 16
        random.seed(seed)
        random.shuffle(self.cards)

    def deal(self, start, n):
        return self.cards[start:start + n]

class Player:
    def __init__(self, hand):
        self.hand = hand
        self.total = 0

    def deal(self, cards):
        self.hand.extend(cards)
        self.total = sum(self.hand)

def cmp(x, y):
    return (x > y) - (x < y)

def play(deck, start, scores):
    player = Player(deck.deal(start, 2))
    dealer = Player(deck.deal(start + 2, 2))

    results = []

    # The player can hit as many times as there are cards left.
```

```
    for i in range(49 - start):
        count = start + 4
        player.deal(deck.deal(count, i))
        count += i

        # Once a player busts, there is no point in exploring further hits, so
we skip to evaluation.
        if player.total > 21:
            results.append((-1, count))
            break

        # The dealer responds deterministically, only hitting if below 17.
        while dealer.total < 17 and count < 52:
            dealer.deal(deck.deal(count, 1))
            count += 1

        # If the dealer busts, the player wins. Else, compare their totals.
        if dealer.total > 21:
            results.append((1, count))
        else:
            results.append((cmp(player.total, dealer.total), count))

    options = []
    for score, next_start in results:
        options.append(score + scores[next_start] if next_start <= 48 else
score)

    scores[start] = max(options)

def blackjack(seed=None):
    deck = Deck(seed)
    scores = [0 for _ in range(52)]
    for start in range(48, -1, -1):
        play(deck, start, scores)
    return scores[0]
```

We have `N` subproblems corresponding to each starting index, and for each of these the player can hit `O(N)` times, in response to which the dealer can hit `O(N)` times. This leads to a complexity of $O(N^3)$.

In reality, neither the player nor the dealer will actually able to hit `N` times- after holding a hand of all four aces, twos, threes, and fours, the next card would put them over `21`. Further, the distribution of low cards in the deck would make it impossible for a player to hit `10` times near both the start and the end of the deck. But for the purposes of an interview question, this should suffice.
