1 Problem Statement

In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

- **High Card**: Highest value card.
- **One Pair**: Two cards of the same value.
- **Two Pairs**: Two different pairs.
- **Three of a Kind**: Three cards of the same value.
- **Straight**: All cards are consecutive values.
- **Flush**: All cards of the same suit.
- **Full House**: Three of a kind and a pair.
- **Four of a Kind**: Four cards of the same value.
- **Straight Flush**: All cards are consecutive values of same suit.
- **Royal Flush**: Ten, Jack, Queen, King, Ace, in same suit.

The cards are valued in the order:
2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

Hand	Player 1	Player 2	Winner
1	5H 5C 6S 7S KD Pair of Fives	2C 3S 8S 8D TD Pair of Eights	Player 2
2	5D 8C 9S JS AC Highest card Ace	2C 5C 7D 8S QH Highest card Queen	Player 1
3	2D 9C AS AH AC Three Aces	3D 6D 7D TD QD Flush with Diamonds	Player 2
4	4D 6S 9H QH QC Pair of Queens Highest card Nine	3D 6D 7H QD QS Pair of Queens Highest card Seven	Player 1
5	2H 2D 4C 4D 4S Full House With Three Fours	3C 3D 3S 9S 9D Full House with Three Threes	Player 1

Figure 1: Example

The file, poker.txt, contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

2 Code

```
# Project Euler #54 | Sidharth Baskaran | 01/13/2022

import time, re
import numpy as np
from collections import Counter

RANKS = dict(zip(['hc','1p','2p','3k','s','f','fh','4k','sf','rf'],[*range(1,11)]))
VALUES = dict(zip(['2','3','4','5','6','7','8','9','T','J','Q','K','A'], [*range(2,15)]))

def score_hand(raw_data, db=False):
    # separate values and suit string lists
    values = [s[0] for s in raw_data]
    suits = [s[1] for s in raw_data]
    # get frequency of suits and values
    freq_s = Counter(suits)
    freq_v = Counter(values)
    card_scores = [VALUES[x] for x in values]
    # zip suits, values, evaluation metric (score)
    hand_uns = list(zip(suits ,values, card_scores))
    # conditions to check
    conditions = [
        lambda: RANKS['hc'],
        lambda: RANKS['1p'] if 2 in list(freq_v.values()) else 0,
        lambda: RANKS['2p'] if list(freq_v.values()).count(2) >= 2 else 0,
        lambda: RANKS['3k'] if 3 in list(freq_v.values()) else 0,
        lambda: RANKS['s'] if sum(np.diff(sorted([c for _,_,c in hand_uns])) == [1] * 4) == 4 else 0,
        lambda: RANKS['f'] if len(freq_s.keys()) == 1 else 0,
        lambda: RANKS['fh'] if 2 in list(freq_v.values()) and 3 in list(freq_v.values()) else 0,
        lambda: RANKS['4k'] if 4 in list(freq_v.values()) else 0,
        lambda: RANKS['sf'] if sum(np.diff(sorted([c for _,_,c in hand_uns])) == [1] * 4) == 4 and len(freq_s.
            keys()) == 1 else 0,
        lambda: RANKS['rf'] if sorted(['T','J','K','Q','A']) == sorted(values) else 0
    ]
    # populate the gained scores
    score_conditions = [c() for c in conditions]
    # set the highest rank attained as the score
    score = (score_conditions[i] for i in range(len(conditions) - 1, -1, -1) if score_conditions[i] > 0)
    ret = (next(score), score_conditions, card_scores, sorted(values),  freq_v)
    if db:
        print(ret)
    return ret

def solve():
    raw = open('project-euler/p054_poker.txt','r').readlines()
    score = 0
    for line in raw:
        split = re.split(r'\s', line)[:-1]
        sc1, r1, scores1, vals1, f1 = score_hand(split[:5])
        sc2, r2, scores2, vals2, f2 = score_hand(split[5:])
        if sc1 == sc2:
            p1 = [r1[7], r1[6], r1[3], r1[2], r1[1]]
            p2 = [r2[7], r2[6], r2[3], r2[2], r2[1]]
            if sum(p1) == sum(p2) and sum(p1) > 0: # paired case, get highest value of paired numbers
                for k1,k2 in zip(f1.keys(), f2.keys()):
                    # remove single-count keys to leave pairs
                    if f1[k1] == 1:
                        vals1.remove(k1)
                    if f2[k2] == 1:
                        vals2.remove(k2)
                # get highest of paired values
                highest_pair_val1 = max([VALUES[v] for v in set(vals1)])
                highest_pair_val2 = max([VALUES[v] for v in set(vals2)])
                if highest_pair_val1 == highest_pair_val2:
                    score += max(scores1) > max(scores2)
                else:
                    score += highest_pair_val1 > highest_pair_val2
            else:
                # check for max value card
                score += max(scores1) > max(scores2)
        elif sc1 > sc2:
            score += 1
    print(score)

if __name__ == "__main__":
    s = time.time()
    solve()
    e = time.time()
    print('%.3fms' % ((e-s)*1000))
```

3 Explanation

This problem requires a brute-force approach to evaluate each game of poker presented in the file. We are given the priorities of each card and the possible ranks. The code begins by initializing dictionaries called `VALUES` and `RANKS` in order to assign an integer priority value to each rank and card value.

The `score_hand` method takes in a hand of 5 cards determines its rank score by iterating through the possible ranks, defined by succinct anonymous functions. The highest rank score is recorded and returned, along with other useful data: all rank scores for the hand, the card values, and the card frequency.

To solve the problem, we iterate through each of the thousand poker games given and compare the scores of the players, incrementing Player 1's count if there is no tie. If there is a tie, we either consider the case of a tie where pairs of cards exist, or a tie without pairs of cards. The rules of the game dictate that if a tie occurs where both players have the same "paired" rank (one pair, two pairs, three of a kind, full house, four of a kind), the player with pairs of the highest value card takes priority. So the code takes the scores from each player pertaining to these 5 "paired" ranks, determines if there is a tie, and removes all single occurences of cards from the hand (thus resulting in only pairs remaining). Then, we compare the maximum value from each player's pairs and increment the count if Player 1 wins. If there is a simple tie, we check which player has the highest value card and increment Player 1's count if they do.