Assignment: 10
Examples Due: Friday, December 1, 2023 8:00 am
Remainder Due: Tuesday, December 5, 2023 9:00 pm
Coverage: End of Module 18
Language level: Intermediate Student with Lambda
Allowed recursion: Any
Files to submit: examples-a10.rkt, edges.rkt, goldbach.rkt, engine.rkt, bonus.rkt

• Make sure you read the A10 Official Post and FAQ on Piazza for the answers to frequently asked questions.

• You are permitted to use any function or language feature listed on slides used in class.

• You are strongly encouraged to use local and higher-order-functions. In many cases these will make your solutions simpler and more efficient.

• Examples are due Friday for questions 1 and 2. None required for question 3.

Here are the assignment questions that you need to solve and submit:

1. In Module 18 we represented a directed graph with an adjacency list.

    ;; A Node is a Sym
    ;; A Graph is a (listof (list Node (listof Node)))

    A simpler implementation, which is convenient for many graph operations, is to represent the graph as an edge list.

    ;; A Node is a Sym
    ;; An EdgeList is a (listof (list Node Node))

    For example, given the following graph in adjacency-list representation:

    '(((A (C D E)) (B (E J)) (C ()) (D (F J))
       (E (K)) (F (K H)) (H ()) (J (H)) (K ())))

    the corresponding graph in edge-list representation is:
The pair 'A C' indicates that the graph contains nodes 'A' and 'C', with an edge from node 'A' to node 'C'. Note that nodes in an adjacency-list representation that have neither out-neighbours nor in-neighbours can’t be represented in an edge list.

Place your solutions to the following questions in edges.rkt.

(a) (10%) Write a function (adj->edge adj-g) that converts a graph in adjacency-list representation to a graph in edge-list representation. You can assume that every node in the adjacency-list representation has at least one out- or in-neighbour. Edges can appear in any order in the edge list.

(b) (10%) Write a function (neighbours v edge-g) that consumes a node v and a graph edge-g in edge-list representation, and produces the list of out-neighbours of v in edge-g. Your function must work directly on the edge representation. Assume that v is a node in edge-g.

(c) (20%) Write a function (edge->adj edge-g) that converts a graph in edge-list representation to a graph in adjacency-list representation. Remember that edges can appear in any order in the edge list representation.

2. Place your solutions to the following questions in goldbach.rkt.

(a) (20%) The Sieve of Eratosthenes is an algorithm for finding all the prime numbers up to a natural number n > 1. It starts with an empty list of primes and a list of natural numbers from 2 up to n, which we will call the candidate primes, or just candidates. As an example, we will use a list of the natural numbers up to n = 20:

'(2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20)

It adds the 2 to the list of primes and then removes all multiples of 2 from the candidates:

'(3 5 7 9 11 13 15 17 19)

Then adds 3 to the list of primes, and removes all multiples of 3 from the candidates list:

'(5 7 11 13 17 19)

And so on, until the candidate list is empty and we have our list of primes. Please see the linked Wikipedia page for a full discussion and an animated illustration. Write a function (primes n) that consumes a natural number and returns a list of primes that are less than or equal to that number. The primes should be sorted from smallest to
largest. While you are not required to use the Sieve of Eratosthenes, we will be testing your functions with values of n up to 100,000 – so efficiency counts.

(check-expect (primes 20) '(2 3 5 7 11 13 17 19))

(b) (20%) **Goldbach’s conjecture** states that every even natural number greater than 2 is the sum of two prime numbers. For example, the number 14 can be written as 7 + 7 or 3 + 11. The *Goldbach partition function* is the function that associates to each even integer the number of different ways it can be written as the sum of two primes. For example, since 14 can be written in 2 different ways, the value of the partition function for 14 is 2.

Write a function *(goldbach n)* that consumes an even natural number greater than 2 and produces a natural number indicating the number of ways it can be written as the sum of two primes. While nobody knows if Goldbach’s Conjecture holds for all even natural numbers greater than 2, we do know it holds for \( n \leq 10,000 \), and we won’t be testing any larger numbers.

(check-expect (goldbach 14) 2)
(check-expect (goldbach 10000) 127)

3. (20%) This question continues from related questions on assignments #4 and #5. In this question, you will write a game engine that allows three players to play a game of Dou Dizhu. For this question, we will supply the players for testing purposes. In the bonus you will have an opportunity to write your own player and try to win bonus marks.

We use the definitions of *Card* and *Hand* from assignments #4 and #5. Recall from those assignments that a *Hand* is a sorted list of *Card*, where a Card is one of 3, 4, 5, 6, 7, 8, 9, 10, 'Jack, 'Queen, 'King, 'Ace, 2, 'Black, and 'Red. Playable hands are limited to those given in assignment #5, specifically rockets, bombs, solos, pairs, trios, straights, straight pairs, and airplanes, plus empty, which will be used to indicate a pass. If two players in a row pass, the third player may not pass, but can play any of the cards they are holding that form a playable hand.

At the start of a Dou Dizhu game, each of the three players bid for the role of “landlord”. In this assignment, we assume that has already taken place. The landlord will play first and will have 20 cards, while the other players will have 17 cards. These hands together will form a standard deck of 54 cards. In the actual game, bids can be between 1 and 3, and can double when certain hands are played. In this assignment, we will ignore all aspects of the game related to bidding and scoring. Your job is to write a function that plays the game and decides the winner, where the winner is the first player to have an empty hand.

A **Player** is a function: Hand  Role (listof Hand) -> Hand

A **Player** consumes the Hand the player is holding, a **Role**, which is (anyof 'Landlord, 'Right, 'Left) and a (listof Hand) indicating the hands played so far. The 'Landlord
starts play, followed by the 'Right player, then the the 'Left player, then the 'Landlord again, and so on. The list of played hands records all plays in order, including empty for a pass, with the most recent hand first. The players we provide for testing will always produce playable hands from the cards they are holding. While the rules for which hand may follow another are given in the bonus question of assignment #5, you do not need to worry about them. Our test players will play fairly and not attempt to cheat.

To make things clearer, here is a helper function that consumes a list of played hands and determines if both previous players have passed:

```
(define (both-passed played)
    (and (cons? played) (empty? (first played))
         (cons? (rest played)) (empty? (second played))))
```

You are free to include this helper function in your solution.

The provided file players.rkt contains three examples of players: goldfish, cautious, and reckless. For example, the goldfish player passes unless the rules of the game require it to play, in which case it will play the lowest single card it holds:

```
(define (goldfish hand role played)
    (cond [(both-passed played) (list (first hand))]
          [else empty]))
```

Write a function (doudizhu players hands) that consumes a list of three players, as defined above and a list of the three hands they are holding, and plays the game. Both lists are ordered by role: 'Landlord first, 'Right second, and 'Left third. The function produces the role of the winning player. The critical step is removing the played cards from the hand the player is holding on the next recursive call.

For example, if it is the landlord’s turn to play, your engine should:

- Call the landlord’s Player, passing it the Hand the landlord is currently holding, the role 'Landlord, and the list of hands played so far. The landlord’s Player returns the Hand to play.
- Remove the played Hand from the Hand the landlord is holding.
- If the landlord now has an empty hand, the game ends. Produce 'Landlord.
- Otherwise, cons the played Hand on to the front of the list of hands played.
- Recursively call the game engine with 'Right as the next Player.

```
(define hand0
  '(3 3 3 3 4 5 6 7 7 7 9 9 Jack Jack Queen King 2 2 Black Red))
(define hand1
  '(4 4 5 5 6 6 7 8 9 10 Jack Queen King Ace 2 2))
```
\begin{verbatim}
(define hand2
  '(5 6 8 8 9 10 10 10 Jack Queen Queen King King Ace Ace Ace))
(check-expect
doudizhu (list goldfish goldfish goldfish) (list hand0 hand1 hand2))
 'Left)
(check-expect
doudizhu (list reckless goldfish goldfish) (list hand0 hand1 hand2))
 'Landlord)
(check-expect
doudizhu (list cautious reckless goldfish) (list hand0 hand1 hand2))
 'Landlord)

Place your solutions in engine.rkt

This concludes the list of questions for which you need to submit solutions, but you can also try for bonus marks with the question on the next page. Remember to always check your email for the basic test results after making a submission.
\end{verbatim}
4. **(Bonus 20% or more)**: Write a Player, as defined above. Your Player must be named student. Place it in the file bonus.rkt.

    (define (student holding role played) ...)

You may define any helper functions you wish. You should be careful that the player only produces playable hands based on the previously played hands and the cards they are holding. If at any point during the bonus testing, your player makes an invalid play, you will receive no bonus marks. You may use the follow function provided by the file players.rkt — or your own version from assignment #5 — to help test the validity of your play, but note that it does not check the validity of a pass.

The bonus will be conducted in two rounds, a qualifying round and a tournament round. In some games, your player will play as the ‘Landlord. In other games, your player will play as both ‘Left and ‘Right (the farmers) so that they can cooperate in defeating the landlord. You should expect that your player will play as the landlord in about half the games and as the farmers in about half the games. The landlord will generally have a “better” hand than the farmers (we will handle the bidding) as well as 20 cards instead of 17.

**Qualifying round:** Your player will play 20 games against different combinations of our cautious and reckless players. For this round, the hands and roles are predefined, so that all players play the same scenarios. If your player wins 15 or more of these games, you will receive 20 bonus marks and be entered into the tournament round.

**Tournament round:** In this round your player will play against players submitted by other students across all sections of CS135. Roles will be assigned randomly in each game. Cards will be shuffled and dealt anew for each game. The structure of the tournament will be determined after we see how many players qualify.

The final tournament round is solely intended to add additional fun and excitement to the course for interested students. We expect to pick several winners (1st place, 2nd place, etc.). The 1st place winner will receive 100 bonus marks, the 2nd place winner will receive 50 bonus marks, and the 3rd place winner will receive 25 bonus marks. Everyone else who finishes in the top quarter will receive 10 bonus marks and those in the top half will receive 5.

Your mark for the entire course is capped at 100.

We will ask your permission before releasing your name as a winner. In any case, we are happy to write the winners a letter or other reference.

We will make our best effort to run this tournament but if we run into unforeseen difficulties, we reserve the right to cancel it.