Assignment: 8
Due: Mon, March 30th, 9:00 pm
Language level: Intermediate Student
Files to submit: debug-a08.rkt, wordtree.rkt, a08-bonus.rkt
Warmup exercises: HtDP Without using explicit recursion: 9.5.2, 9.5.4
Practice exercises: HtDP 20.2.4, 24.3.1, 24.3.2

- Make sure you read the OFFICIAL A08 post on Piazza for the answers to frequently asked questions.
- All helper functions must be encapsulated using local.
- Unless stated otherwise, all policies from Assignment 07 carry forward.
- This assignment covers material up to the end of module 12.
- You may use any list or string function covered in class, as well as any mathematical function or type-checking predicate.
- Remember that basic tests are meant as sanity checks only; by design, passing them should not be taken as any indication that your code is correct, only that it has the right form.
- Unless the question specifically says otherwise, you are always permitted to write helper functions (however, note that they must be local, as stated above). You may use any constants or functions from any constants or required functions from earlier question parts to help with later parts even though they are not defined locally.
- Solutions will be marked for both correctness [80%] and style [20%]. Follow the guidelines in the Style Guide, pages 1–15.

1. [20% Correctness] Question 1
   Consider the code in debug-a08.rkt, which may contain syntax, mathematical, and style errors. The code may also be difficult to read and/or maintain, lack helper functions and constants. Correct the code.

2. [60% Correctness] Question 2
   In this question you will implement various functions to produce an “autocomplete” for sentences without punctuation.
   For this question you may NOT have top-level helper functions. Use local instead.
   This question uses the following data definition:
A WordTree is one of:
* Str
* (cons Str (listof WordTree))

requires: Str is not "".
  (listof WordTree) is nonempty.
  sibling nodes cannot have the same Str:
    i.e., '("a" "b" "b") is not valid, neither is '("a" ("b" "c") ("b" "d"))

A WordTree is a general tree stored in a nested list.

Implement the following:

(a) Write a template for a WordTree. Call your template WordTree-template. As part
    of your template, be sure to include the necessary template for the children, call this
    template WT-children-template.

(b) Write a predicate wt-equal? that consumes two WordTrees and produces true if they
    are equal, and false otherwise. You may NOT use equal? for this question. Note that
    order DOES matter for this question. '("a" "b" "c") is not equal to '("a" "c" "b")

(c) Write a predicate exists? that consumes a path as a list of strings and a WordTree and
    produces true if that path exists in the tree.
    Note that the path, if it exists in the tree, may end at an internal node or a leaf. The path
    must also start at the root and cannot skip any levels (each element in the path is a child
    of the former one).
    Note that '() is a valid (empty) path that exists in every tree.

(d) Write a function get-leaves that consumes a WordTree and produces a list of strings
    that are the leaves of the consumed tree. Use a preorder traversal where the node is
    visited, followed by its left most subtree, then each of the other subtrees in turn going
    left-to-right.

(e) Write a function all-sentences that consumes a WordTree and produces a list of all
    paths to leaves in the tree.
    The list of paths may be in any order.

(f) Write a function suggest that consumes a sentence as a list of strings and a WordTree
    and produces all possible endings to the provided sentence. That is, it should produce
    all possible paths to leaves from the end of the provided sentence. Note that the last
    word of each consumed sentence should be the first word of each suggested completion.
    Note that sentences must contain at least one word. However, the order of suggestions
    does not matter.

Place your solution in wordtree.rkt. Note that this file already contains the data definition
and a test tree. Also note that we have provided a few tests for some of the above functions.
These tests are meant to show you the expected behaviour of the function. You may use our
provided tests in your examples and tests if you beleive they are appropriate.
Bonus Question [5%] Consider the following BST definition.

```
(define-struct node (key left right))
;; A BST is one of:
;; * empty
;; * (make-node Int BST BST)
;; requires: each key in left is smaller than key
;; each key in right is larger than key
```

Write a function `next-key` that consumes a BST, and a key, and produces the next larger key in the BST. You can assume the consumed key exists somewhere in the BST. If the consumed key is the largest key in the BST, then produce empty. For example, given: (define bst-example (make-node 4 (make-node 1 empty empty) (make-node 9 empty empty)))

then:

```
(next-key bst-example 1) → 4
(next-key bst-example 4) → 9
(next-key bst-example 9) → empty
```

You may NOT flatten the BST and then search through the flattened result. You may NOT have top-level helper functions. Any helper functions must be defined locally.

Put your solution in a08-bonus.rkt.

Enhancements: Reminder—enhancements are for your interest and are not to be handed in.

Consider the function `(euclid-gcd)` from slide 7-16. Let $f_n$ be the $n$th Fibonacci number. Show that if $u = f_{n+1}$ and $v = f_n$, then `(euclid-gcd u v)` has depth of recursion $n$. Conversely, show that if `(euclid-gcd u v)` has depth of recursion $n$, and $u > v$, then $u \geq f_{n+1}$ and $v \geq f_n$. This shows that in the worst case the Euclidean GCD algorithm has depth of recursion proportional to the logarithm of its smaller input, since $f_n$ is approximately $\phi^n$, where $\phi$ is about 1.618.

You can now write functions which implement the RSA encryption method (since Racket supports unbounded integers). In Math 135 you will see fast modular exponentiation (computing $m^e \mod t$). For primality testing, you can implement the little Fermat test, which rejects numbers for which $a^{n-1} \not\equiv 1 \pmod n$, but it lets through some composites. If you want to be sure, you can implement the Solovay–Strassen test. If $n - 1 = 2^d m$, where $m$ is odd, then we can compute $a^m \pmod n$, $a^{2m} \pmod n$, $\ldots$, $a^{n-1} \pmod n$. If this sequence does not contain 1, or if the number which precedes the first 1 in this sequence is not $-1$, then $n$ is not prime. If $n$ is not prime, this test is guaranteed to work for at least half the numbers $a \in \{1, \ldots, n - 1\}$.

Of course, both these tests are probabilistic; you need to choose random $a$. If you want to run them for a large modulus $n$, you will have to generate large random integers, and the built-in function `random` only takes arguments up to 4294967087. So there is a bit more work to be done here.
For a real challenge, use Google to find out about the recent deterministic polynomial-time algorithm for primality testing, and implement that.

Continuing with the math theme, you can implement the extended Euclidean algorithm: that is, compute integers $a, b$ such that $am + bn = \gcd(m, n)$, and the algorithm implicit in the proof of the Chinese Remainder Theorem: that is, given a list $(a_1, \ldots, a_n)$ of residues and a list $(m_1, \ldots, m_n)$ of relatively coprime moduli ($\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq n$), find the unique natural number $x < m_1 \cdots m_n$ (if it exists) such that $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$. 