CS 135 Fall 2017

Tutorial 10: Generative recursion and Graphs
Reminders

- There is no assignment due this coming Tuesday, November 28
- Assignment 09 is due **Monday, December 4 at 9:00pm**
- Start Assignment 09 **early**!
Assignment Review: Binary Numbers

On Assignment 09 Question 1, you will work with binary numbers. Here’s an example:

The binary number 101101 represents the decimal number 45.

<table>
<thead>
<tr>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The positions of digits in a decimal number represent powers of 10:

1067 = 1 \cdot 10^3 + 0 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0

Similarly, the positions of bits in a binary number represent powers of 2:

101101 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0
Group Problem: Converting from Binary to Decimal

For this question, we will represent binary numbers as `(listof (anyof 0 1))`. For example:

1011 $\Rightarrow$ `(1 0 1 1)

Write a function `bin-to-dec` that consumes a binary number as a list and produces the decimal number it represents.

What kind of recursion might be most suitable here?

Examples:

`(bin-to-dec '()) $\Rightarrow$ 0

`(bin-to-dec '(1 0 0 0)) $\Rightarrow$ 8
Group Problem - partition

Write a function `partition` that consumes a list called `lst`, and a list of positive integers called `sizes`. The numbers in `sizes` must add up to the length of `lst`. `partition` splits `lst` into sublists of lengths specified by `sizes`.

Hint: Think back to how `string->strlines` was implemented in Module 11

```
(partition '() '(0 0 0)) ⇒ '()
(partition '(a b c) '(3)) ⇒ '((a b c))
(partition '(a b c d e f g) '(3 1 2 1)) ⇒ '((a b c) (d) (e f) (g))
```
Review: Graph Terminology

In a directed graph, an **edge** is an ordered pair of **nodes** (also called **vertices**), which we can represent by an arrow from one node to another.

A sequence of nodes $v_1, v_2, \ldots, v_k$ is a **path** of length $k - 1$ if $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ are all edges.

If $v_1 = v_k$, this is called a **cycle**.

Directed graphs without cycles are called **DAGs** (directed acyclic graphs).
Review: Adjacency List Representation

We can represent a graph as a list of pairs, each pair consisting of a symbol (the node’s name) and a list of symbols (the names of the node’s out-neighbours).

This is called the **adjacency list** representation.

```scheme
;; A Node is a Sym

;; A Graph is a (listof (list Node (listof Node)))
```
'((A (B C D))
(B (E F))
(C (E))
(D (F H G))
(E ())
(F (H))
(H ())
(G ()())
Clicker Question - count-vertices

How many vertices are there in the following graph?

'((A (C F))
  (B ()))
  (C (B))
  (D (A E))
  (E ())
  (F (B C G))
  (G (D))))

A  4
B  6
C  7
D  8
Clicker Question - Cycles in Explicit Graphs

Is there a cycle in this adjacency list representation of an explicit graph?

Note: the easiest way to see cycles is by drawing the graph

'(A (C F))
  (B ())
  (C (B))
  (D (A B E))
  (E ())
  (F (B C G))
  (G (D)))

A  Yes
B  No
Group Problem (optional) - ones-on-diagonal

In Tutorial 05, we wrote a function called \texttt{ones-on-diagonal} that produced a table as a \texttt{(listof (listof Nat))} that contains 1s on the main diagonal and 0s everywhere else. We originally wrote it is using pure structural recursion, with multiple helper functions. This time we will write the same function but without explicit recursion and without any helper functions.

\[
\begin{align*}
(\text{ones-on-diagonal } 0) & \Rightarrow \text{empty} \\
(\text{ones-on-diagonal } 1) & \Rightarrow (\text{list (list 1)}) \\
(\text{ones-on-diagonal } 3) & \Rightarrow \\
& (\text{list (list 1 0 0) (list 0 1 0)})
\end{align*}
\]
(list 0 0 1)))