Lecture 22. Lists and Recursion

Recursive Definition of a List

- A **list** is a set of data elements in some sequential order

$$x_1, x_2, \ldots, x_n$$

where all of the x_i's are of the same type (e.g., integers, strings, etc.).

Recursive Definition of a List

- A **list** is a set of data elements in some sequential order

$$x_1, x_2, \ldots, x_n$$

  where all of the x_i's are of the same type (e.g., integers, strings, etc.).
- Recursive Definition of a List Let X be a set. A list of elements of X is
  B. x where $x \in X$.
  R. L, x where $x \in X$ and L is a list of elements of X.

Recursive Definition of a List

- A **list** is a set of data elements in some sequential order

$$x_1, x_2, \ldots, x_n$$

  where all of the x_i's are of the same type (e.g., integers, strings, etc.).
- Recursive Definition of a List Let X be a set. A list of elements of X is
  B. x where $x \in X$.
  R. L, x where $x \in X$ and L is a list of elements of X.
- **Remark.** A list may repeat the same element several times, and the order of the elements matters. Every symbol in this definition is important; the commas between the list elements are part of the structure of a list.

Recursive Definition of a List

- A **list** is a set of data elements in some sequential order

$$x_1, x_2, \ldots, x_n$$

  where all of the x_i's are of the same type (e.g., integers, strings, etc.).

- Recursive Definition of a List Let X be a set. A list of elements of X is
  - **B.** x where $x \in X$.
  - **R.** L, x where $x \in X$ and L is a list of elements of X.

- **Remark.** A list may repeat the same element several times, and the order of the elements matters. Every symbol in this definition is important; the commas between the list elements are part of the structure of a list.

- **Example 22.1**. Let $X = \{\text{cubs}, \text{bears}, \text{bulls}\}$. Use the recursive definition to build up the following list of strings

$$\text{cubs}, \text{bears}, \text{bulls}, \text{cubs}$$

Solution to Example 22.1

$$\begin{aligned}
L_1 &= \text{cubs} && \text{by part } \mathbf{B}\\
L_2 &= L_1, \text{bears} = \text{cubs}, \text{bears} && \text{by part } \mathbf{R}\\
L_3 &= L_2, \text{bulls} = \text{cubs}, \text{bears}, \text{bulls} && \text{by part } \mathbf{R}\\
L_4 &= L_3, \text{cubs} = \text{cubs}, \text{bears}, \text{bulls}, \text{cubs} && \text{by part } \mathbf{R}
\end{aligned}$$

SLists [1]

The next recursive definition is for educational purposes only; the object it defines is simple and somewhat limited. The point is to help us study ways to search for an element in a list.

- **Definition** An SList is
  - **B.** x where $x \in \mathbb{R}$.
  - **R.** (X, Y) where X and Y are SLists having the same number of elements, and the last number in X is less than the first number in Y.

[1] The elements of an SList must be sorted in order from left to right.

SLists [1]

The next recursive definition is for educational purposes only; the object it defines is simple and somewhat limited. The point is to help us study ways to search for an element in a list.

- **Definition** An SList is
  - **B.** x where $x \in \mathbb{R}$.
  - **R.** (X, Y) where X and Y are SLists having the same number of elements, and the last number in X is less than the first number in Y.
- **Example 22.2**. Show an example of an SList using numbers 1, 3, 8, 9, 12, 16, 25, and 30, which are in $\mathbb{R}$.

[1] The elements of an SList must be sorted in order from left to right.

SLists [1]

The next recursive definition is for educational purposes only; the object it defines is simple and somewhat limited. The point is to help us study ways to search for an element in a list.

- **Definition** An SList is
  - **B.** x where $x \in \mathbb{R}$.
  - **R.** (X, Y) where X and Y are SLists having the same number of elements, and the last number in X is less than the first number in Y.
- **Example 22.2**. Show an example of an SList using numbers 1, 3, 8, 9, 12, 16, 25, and 30, which are in $\mathbb{R}$.
  - $(((1, 3), (8, 9)), ((12, 16), (25, 30)))$ is an SList.

[1] The elements of an SList must be sorted in order from left to right.

SLists [1]

The next recursive definition is for educational purposes only; the object it defines is simple and somewhat limited. The point is to help us study ways to search for an element in a list.

- **Definition** An SList is

  B. x where $x \in \mathbb{R}$.

  R. (X, Y) where X and Y are SLists having the same number of elements, and the last number in X is less than the first number in Y.

- **Example 22.2**. Show an example of an SList using numbers 1, 3, 8, 9, 12, 16, 25, and 30, which are in $\mathbb{R}$.
  - $(((1, 3), (8, 9)), ((12, 16), (25, 30)))$ is an SList.

- SLists always have 2^p elements, for some $p \geq 0$. The number p counts the **depth** of parentheses of the SList (or the depth of the Slist). So, the example SList above has depth 3 and contains 2^3 elements.

[1] The elements of an SList must be sorted in order from left to right.

SLists [1]

The next recursive definition is for educational purposes only; the object it defines is simple and somewhat limited. The point is to help us study ways to search for an element in a list.

- **Definition** An SList is
  - **B.** x where $x \in \mathbb{R}$.
  - **R.** (X, Y) where X and Y are SLists having the same number of elements, and the last number in X is less than the first number in Y.
- **Example 22.2**. Show an example of an SList using numbers 1, 3, 8, 9, 12, 16, 25, and 30, which are in $\mathbb{R}$.
  - $(((1,3),(8,9)),((12,16),(25,30)))$ is an SList.
- SLists always have 2^p elements, for some $p \geq 0$. The number p counts the **depth** of parentheses of the SList (or the depth of the Slist). So, the example SList above has depth 3 and contains 2^3 elements.
- Also, if $L = (X, Y)$ is an SList of depth p, then X and Y must have depth $p - 1$.

[1] The elements of an SList must be sorted in order from left to right.

Search Algorithm A

- Suppose we want to define a function that determines whether or not a given number is in the SList.
- **Algorithm A** Define a true or false function ASearch(t, L), where t is a number and L is an SList, as follows.

  B. Suppose $L = x$, a list of depth 0. Then

$$\text{ASearch}(t, L) = \begin{cases} \text{true} & \text{if} \quad t = x \\ \text{false} & \text{if} \quad t \neq x \end{cases}$$

  R. Suppose the depth of L is greater than 0, so $L = (X, Y)$. Then

$$\text{ASearch}(t, L) = \text{ASearch}(t, X) \vee \text{ASearch}(t, Y)$$

- **Example 22.3**. Let $L = (((1, 3), (8, 9)), ((12, 16), (25, 30)))$.

$$\text{ASearch}[8, L] =$$

Solution to Example 22.3

$$
\begin{aligned}
\mathrm{Search}[8, L] &= \mathrm{Search}[8, ((1,3),(8,9))] \vee \mathrm{Search}[8, ((12,16),(25,30))] \\
&= \mathrm{Search}[8, (1,3)] \vee \mathrm{Search}[8, (8,9)] \vee \mathrm{Search}[8, (12,16)] \\
&\quad \vee \mathrm{Search}[8, (25,30)] \\
&= \mathrm{Search}[8, 1] \vee \mathrm{Search}[8, 3] \vee \mathrm{Search}[8, 8] \vee \mathrm{Search}[8, 9] \\
&\quad \vee \mathrm{Search}[8, 12] \vee \mathrm{Search}[8, 16] \vee \mathrm{Search}[8, 25] \vee \mathrm{Search}[8, 30] \\
&= \mathrm{false} \vee \mathrm{false} \vee \mathrm{true} \vee \mathrm{false} \vee \mathrm{false} \vee \mathrm{false} \vee \mathrm{false} \vee \mathrm{false} \\
&= \mathrm{true}.
\end{aligned}
$$

Search Algorithm B

- **Algorithm B** Define a true or false function BSearch(t, L), where t is a number and L is an SList, as follows.

  B. Suppose $L = x$, a list of depth 0. Then

$$\text{BSearch}(t, L) = \begin{cases} \text{true} & \text{if} \quad t = x \\ \text{false} & \text{if} \quad t \neq x \end{cases}$$

  R. Suppose L has depth $p > 0$, so $L = (X, Y)$. Let r be the last element of X. Then

$$\text{BSearch}(t, L) = \begin{cases} \text{BSearch}(t, Y) & \text{if} \quad t > r \\ \text{BSearch}(t, X) & \text{if} \quad t \not> r \end{cases}$$

- **Example 22.4**. Let $L = (((1,3),(8,9)),((12,16),(25,30)))$.

$$\text{BSearch}[8, L] =$$

Solution to Example 22.4

$$\begin{aligned}
\text{BSearch}[8, L] &= \text{BSearch}[8, ((1,3),(8,9))] && \text{since } 8 \not> 9 \\
&= \text{BSearch}[8, (8,9)] && \text{since } 8 > 3 \\
&= \text{BSearch}[8, 8] && \text{since } 8 \not> 8 \\
&= \text{true} && \text{since } 8 = 8.
\end{aligned}$$

We will compare Algorithm A and B to determine which one is more efficient than the other using the Big-$\mathcal{O}$ estimation.

Exercises

1. Let L be a list. Define a numerical function f as follows.
   - **B.** If $L = x$, a single element, then $f(L) = 1$.
   - **R.** If $L = L', x$ for some list L', then $f(L) = f(L') + 1$.

   (a) Show the steps to find the value of $f(\text{veni}, \text{vidi}, \text{vici})$.
   (b) What does the value of $f(L)$ tell you about the list L, in general?
   (c) Prove your assertion in part (b), using induction.
2. Let $L = (((10, 20), (30, 40)), ((50, 60), (70, 80)))$ be an SList.
   (a) Compute ASearch(15, L), showing all steps.
   (b) Compute BSearch(15, L), showing all steps.
   (c) Write a python code of either ASearch(t,L) or BSearch(t,L). Verify your code by computing (a) or (b).
3. Let L be an SList. Define a recursive function Flip as follows.
   - **B.** Suppose $L = x$. Then Flip(L) $= x$.
   - **R.** Suppose $L = (X, Y)$. Then, Flip(L) = (Flip(Y), Flip(X)).

   Compute Flip[((2,3), (7, 9))], showing all steps.