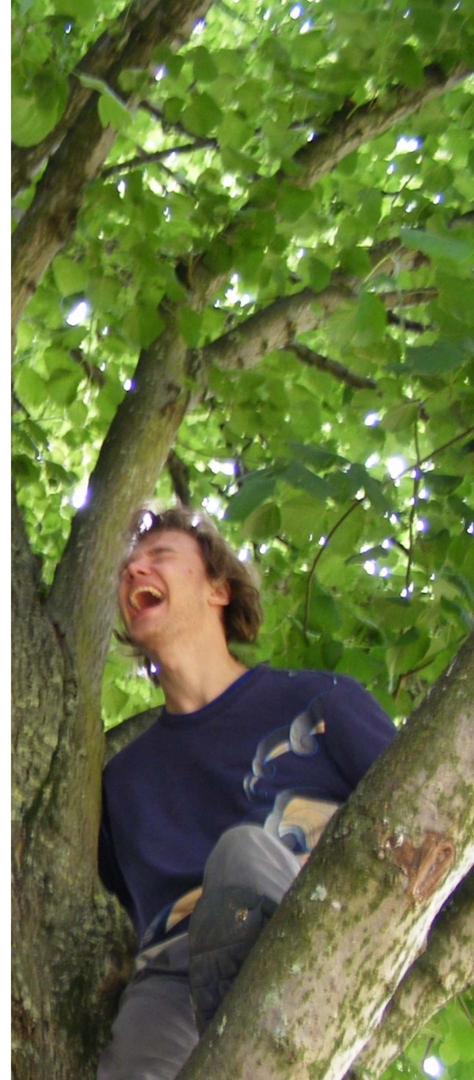
Climbing Trees & Catching Bugs

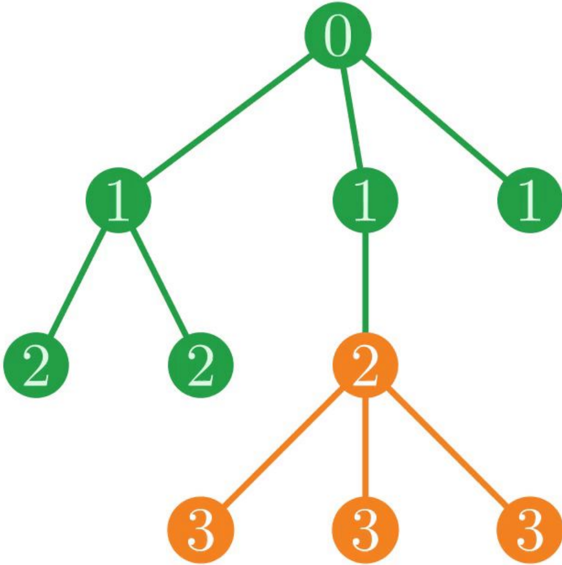
Asher Harvey-Smith

(senior) Intern at Dyalog this summer

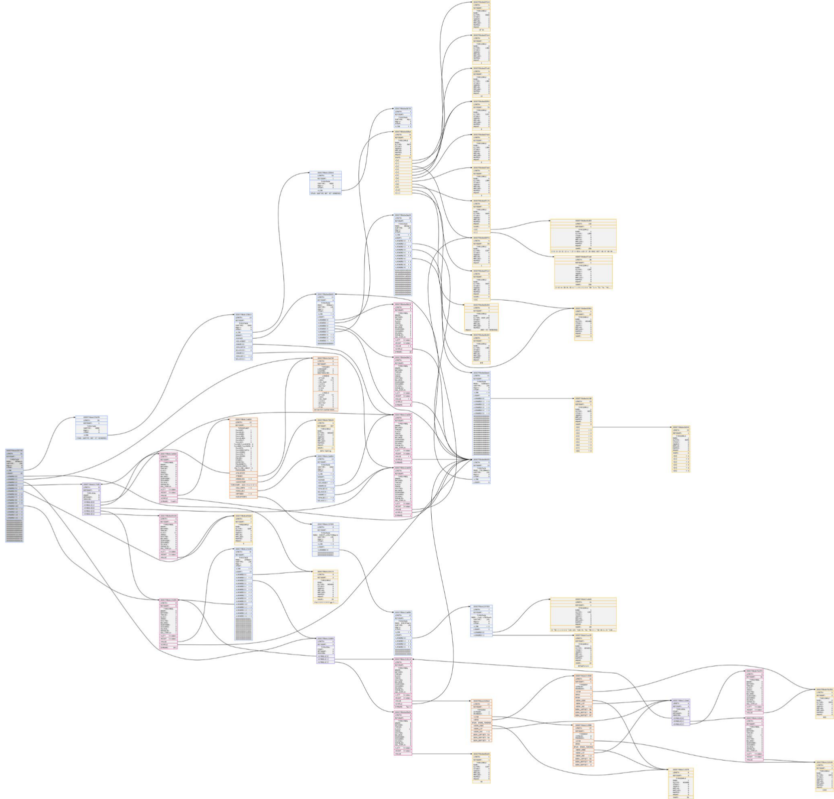
Computer Science student at the University of Warwick



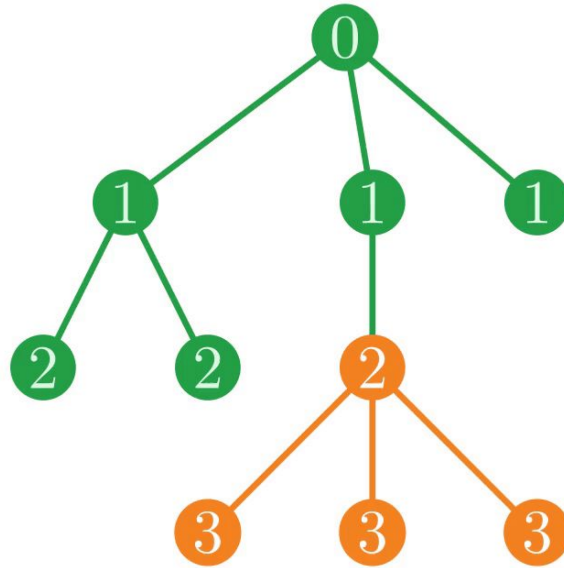
Climbing Trees



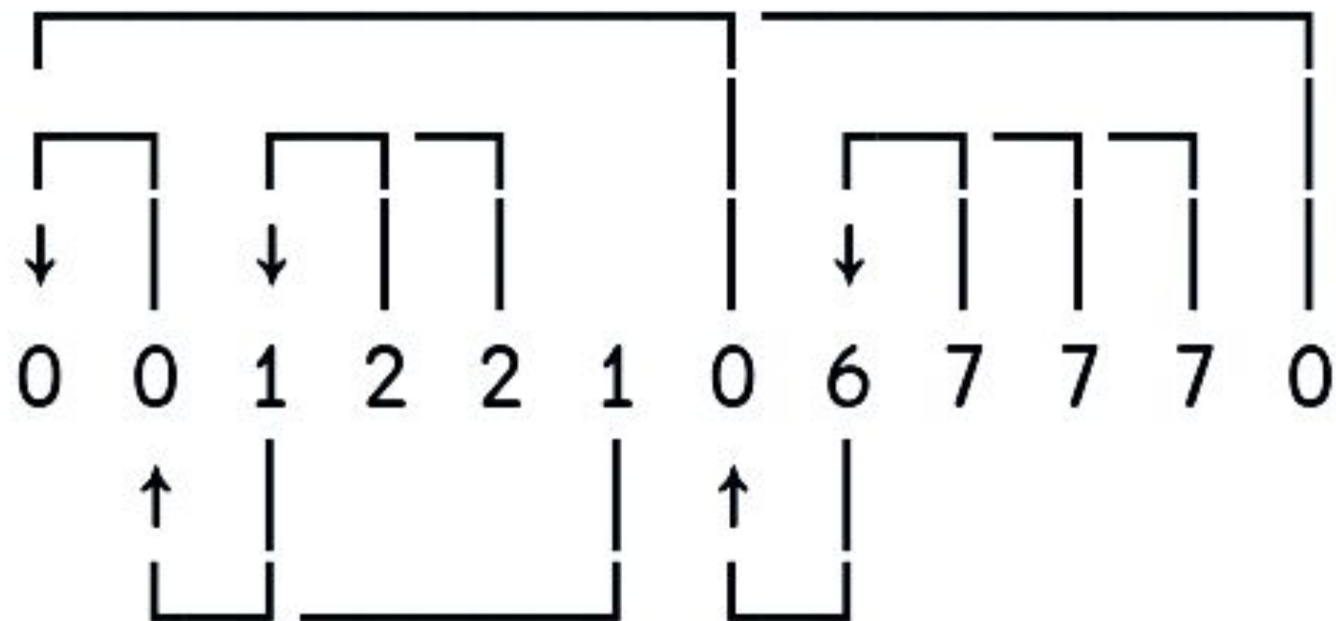
Catching Bugs

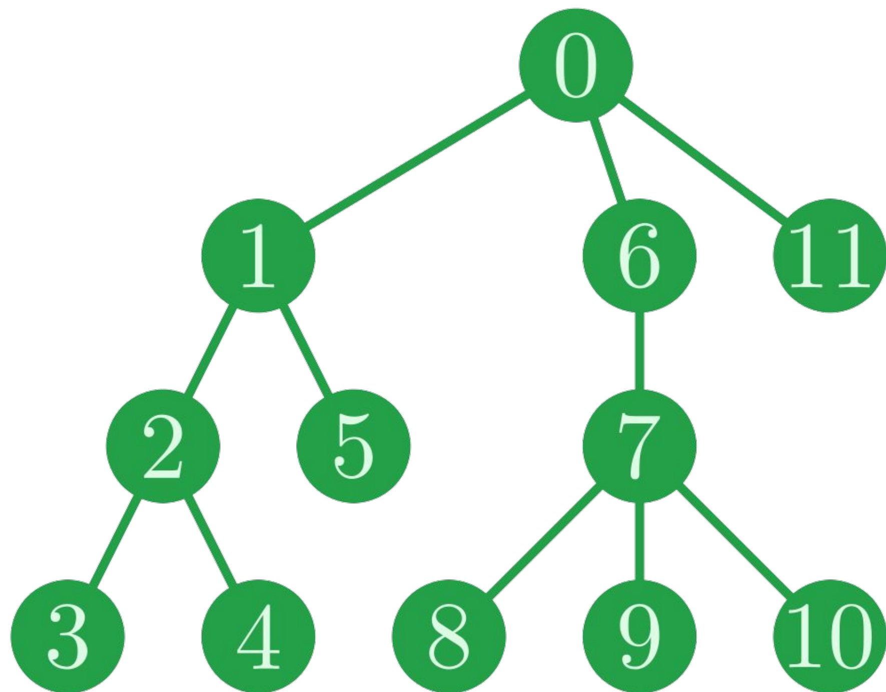


Climbing Trees



0 0 1 2 2 1 0 6 7 7 7 0


0 0 1 2 2 1 0 6 7 7 7 0


0
1
6
11
2
5
7
3
4
8
9
10

the journal of the British APL Association

NO STINKING LOOPS

Treetable: a case-study in q

Stevan Apter

This article is the first in an occasional column, *No Stinking Loops*. Stevan Apter is one of the programmers Jeffry Borror referred to as "the q gods" in his textbook *q for Mortals*. The world of q programming has so far been largely hidden behind corporate non-disclosure contracts. *Vector* is glad to see it opening and proud to be publishing this. *Ed.*

0. Introduction

A treetable is a table with four additional properties.

Firstly, the records of the table are related hierarchically. Thus, a record may have one or more child-records, which may in turn have children. If a record has a parent, it has exactly one. A record without a parent is called a root record. A record without any children is called a leaf record. A record with children is called a node record.

Secondly, it is possible to *drill down* into a treetable. If a record is a parent, then some of its columns may be rollups of its child-records. By drilling down into a parent-record, it is possible to inspect the elements which are aggregated in the parent. All rollups are performed on the leaves of the tree rather than on the immediate children. This means that tree-construction can be 'lazy': not all

© 1984-2024
British APL Association
All rights reserved.

Archive articles posted online on request: ask the archivist.

Page | Discussion

Essays/Tree Display

< Essays

Contents [hide]

1 Tree Display
2 Examples
3 Program Logic
3.1 subtree
3.2 graft
3.3 connect
3.4 root
3.5 extend

Getting Started with J
J code search
Main Page
NuVoc
Playground
Wiki Hints
Guides
System
Showcase
Library
Community
Recent changes
New Pages

Tools

What links here
Related changes
Special pages
Printable version
Permanent link
Page information
Cite this page

Tree Display

The verb `tree` pro

```
BOXC=: 9!:6 ''
EW  =: {: BOXC

tree=: 3 : 0
 assert. ($y) -
 y=. ":&.>^:(32
 assert. ((2 =
 j=.  ~. , y
 t=. (<EW,' ')
 c=. |: j i. y
 while. +./ b=.
```

User page | Discussion

User:Devon McCormick/Trees

< User:Devon McCormick

Getting Started with J
J code search
Main Page
NuVoc
Playground
Wiki Hints
Guides
System
Showcase
Library
Community
Recent changes
New Pages

Tools

What links here
Related changes
User contributions
Logs
View user groups
Special pages
Printable version
Permanent link
Page information

Here's an exposition of one method for representing trees ir

Contents [hide]

1 Parent Index Vector
1.1 Basic Navigation
1.1.1 Grafting
1.2 Validity Checking
1.3 Grafting, Pruning, and Displaying
2 Using "Tree Display"
3 Code

Parent Index Vector

The tree is a vector of integers where each element is the in

Say we have a directory tree like this:

```
C:
|__n0
|    |_n00
|    |_n01
|
|__n1
     |__n10
     |    |__n100
     |__n11
     |    |__n110
     |    |__n111
     |    |__n112
     |__n12
```

We typically separate the tree structure from the corresponding vector of nodes which, in this case, are

User page | Discussion

User:Doug Mennella/Trees

< User:Doug Mennella

Getting Started with J
J code search
Main Page
NuVoc
Playground
Wiki Hints
Guides
System
Showcase
Library
Community
Recent changes
New Pages

Tools

What links here
Related changes
User contributions
Logs
View user groups
Special pages
Printable version
Permanent link
Page information

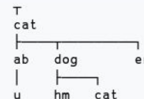
Contents [hide]

1 Representing trees with vectors
2 Paths
3 Tree display
3.1 The code in full
4 Centering display
4.1 Centering labels
4.2 the code
5 Toolset

Representing trees with vectors

Trees are familiar data structures in programming and there are a number of ways to represent them ir
recursively descend the structure of the tree from some root node. But a tree is also a graph and as suc

For this graph we have the following list of edges.



cat	ab
ab	u
cat	dog
dog	hm

```
:Namespace ⋄ f←{(2=+⌿0=X∘.|X)⌿X←⍳ω} ⋄ :EndNamespace
```

1. After lexing:

2. After parsing brackets:

1. After function specialization:

2. Complete AST:

```
0 0 0 0 0 1 3 0 0 0 4 0 1 0 2 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0
      1 0 1 0 1 0 0 0 0
      (NΔ[t],¨(⍕¨k))dwv pp3 p
```

```
_←(gz 1⌽←)¨(t[i]=Z)⊂i←⍸d=0
'Non-Z top-level node'assert t[⍸p=⍳≢p]=Z:

⍝ Wrap all function expression bodies as Z nodes
_←p[i]{end[α]←end[⊃⌽ω] ⋄ gz¨ω⊂⍨1,¯1↓t[ω]=Z}⌸i←⍸(t[p]∊T F)∧~t=L
'Non-Z/L dfns body node'assert t[⍸t[p]=F]∊Z L:

⍝ Parse the first line of a trad-fn as an H node
t[⍸(≠p)∧t[p]=T]←H
∨⌿msk←(n=-sym⍳⊂,'←')∧(≠p)∧t[p]=H:'EMPTY RETURN HEADER'SIGNAL SELECT ⍸msk
∨⌿msk←(n=-sym⍳⊂,';')∧(≠p)∧t[p]=H:'MISSING SIGNATURE'SIGNAL SELECT ⍸msk
msysv←'⎕IO' '⎕ML' '⎕CT' '⎕PP' '⎕PW' '⎕RTL' '⎕FR' '⎕PATH' '⎕RL' '⎕DIV' '⎕TRAP' '⎕USING'
msk←(t[p]=H)∧~(t=V)∨(n∊-sym⍳,¨'←(){};')∨(t=S)∧n∊-sym⍳⎕C¨msysv
```

UNCHAIN YOUR MIN

https://asherbhs.github.io/apl-site/trees/intro.html

Search Ctrl + K

Exploring Things with Dyalog APL

Trees

Introduction

Representing Trees

Parent Vectors

Relating the Depth and Parent Vectors

Forests

Deleting Nodes

Bottom-Up Accumulation

Working with ⎕JSON

Combinatorics

Enumerative Combinatorics

Introduction

APL is fantastic for working with linear data. If you can organise your data in an array, you have dozens of primitives and years of collective wisdom to lead you to success. Sadly, in the wild, there are many problems which have a fundamentally non-linear structure. These present an issue for the brave APL programmer, whose array primitives struggle to cope.

One of the most common cases of this is dealing with hierarchical data, where pieces of data are variously 'contained in' or 'belonging to' others. Data like this are examples of a general structure called a *tree*.

Formally, trees are made up of *nodes*. Each node may have some number of *child nodes*, and usually one *parent node*. There is exactly one node in a tree which has no parent, this is the tree's *root node*. Nodes which share a parent are *sibling nodes*, and nodes with no children are *leaf nodes*.

For many kinds of hierarchical data, we can model its structure as a tree:

Data	Nodes	Root node	x is a child node of y
A file system	Files and folders	The root or home directory	x is in folder y
Jobs in a company	Employees	CEO	x reports to y
Evolutionary tree of life	Species	Some unknown early micro-organism	x evolved from y
JSON	Data	The outermost object or array	x is a member of array or object y

Interestingly, a family tree is not an example of this kind of tree, as each child typically does not have just one unique parent.

We will generally draw trees with the root node at the top, and all child nodes arranged below, with a line connecting each child to its parent. For instance, in the following tree, the node labelled a is the root node, and is a parent of b, c and d - its children.

```
↑(⊂¨'index:' 'depths:' 'data:'),¨(⍳≢depths) depths data
```

index:	0	1	2	3	4	5	6	7	8	9
depths:	0	1	2	2	1	2	3	3	3	1
data:	a	b	e	f	c	g	h	i	j	d

At this point it's helpful to make a small mental shift. We are drawing a one-to-one correspondence between node and indices into these vectors, using the indices as unique identifiers. We will use this so frequently that making the distinction explicit becomes tiring, so from now on, we will often refer to 'the node associated with index `i`' simply as 'node `i`'.

Labelling each node of our tree with its corresponding index reveals an interesting pattern.

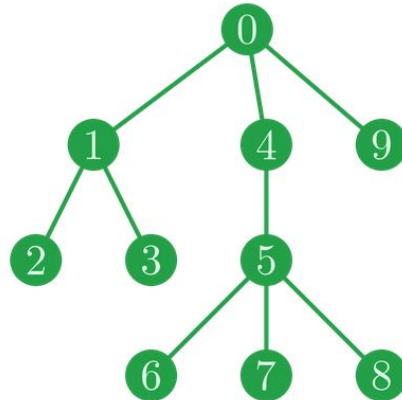


Fig. 4 The same tree, with nodes labelled with their index into the `depth` and `data` vectors.

Search Ctrl + K
Exploring Things with Dyalog APL
Trees
Introduction
Representing Trees
Parent Vectors
Relating the Depth and Parent Vectors
Forests
Deleting Nodes
Bottom-Up Accumulation
Working with ⎕JSON
Combinatorics
Enumerative Combinatorics

Contents

Preliminary Definitions
The Nested Representation
The Depth Vector Representation
The Path Matrix Representation
The Parent Vector Representation
Challenge

Search Ctrl + K

Exploring Things with Dyalog APL

Trees

Introduction
Representing Trees
Parent Vectors
Relating the Depth and Parent Vectors
Forests
Deleting Nodes
Bottom-Up Accumulation
Working with ⎕JSON

Combinatorics

Enumerative Combinatorics

Contents

Basic Operations
Favourite Children (Ordering Siblings)
Inverting
Pretty Printing

Inverting

Because it doesn't really fit anywhere else in the tutorial, let's look at a neat way to reverse the order of all siblings in a tree - in other words, mirroring the tree.

We will again reset our tree.

```
⊢p←parent
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```

On our example tree, inverting looks like this:

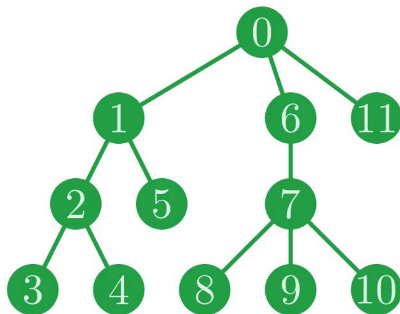


Fig. 15 Mirroring the tree.

Our first step is to invert the parent vector is simply reverse it.

Search Ctrl + K

Exploring Things with Dyalog APL

Trees

Introduction

Representing Trees

Parent Vectors

Relating the Depth and Parent Vectors

Forests

Deleting Nodes

Bottom-Up Accumulation

Working with `⎕JSON`

Combinatorics

Enumerative Combinatorics

Working with `⎕JSON`

We've laboriously gone through many different ways to work with parent vectors, but we're yet to do any 'real work' with them. This page will cover an interesting application of the techniques we've learned so far. We're going to look at manipulating JSON-formatted data, making use of Dyalog APL's built-in `⎕JSON`.

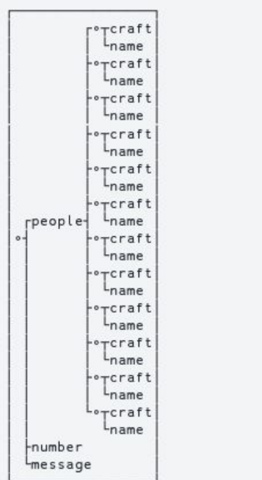
There's a wonderful little API which returns some JSON describing every person currently in space.

```
]load HttpCommand
json←(HttpCommand.Get 'http://api.open-notify.org/astros.json').Data
⎕JSON⎕OPT'Compact' 0⎕JSON json     ⍝ prettify it
```

```
#.HttpCommand
```


```
{
  "message": "success",
  "number": 12,
  "people": [
    {
      "craft": "ISS",
      "name": "Oleg Kononenko"
    },
    {
      "craft": "ISS",
      "name": "Nikolai Chub"
    },
    {
      "craft": "ISS",
      "name": "Tracy Caldwell Dyson"
    },
    {
      "craft": "ISS",
      "name": "Matthew Dominick"
    },
    {
      "craft": "ISS",
      "name": "Michael Barratt"
    },
    {
      "craft": "ISS",
      "name": "Jeanette Epps"
    },
    {
      "craft": "ISS",
      "name": "Alexander Grebenkin"
```

```json
{
  "message": "success",
  "number": 12,
  "people": [
    {
      "craft": "ISS",
      "name": "Oleg Kononenko"
    },
    {
      "craft": "ISS",
      "name": "Nikolai Chub"
    },
    {
      "craft": "ISS",
      "name": "Tracy Caldwell Dyson"
    },
    {
      "craft": "ISS",
      "name": "Matthew Dominick"
    },
    {
      "craft": "ISS",
      "name": "Michael Barratt"
    },
    {
      "craft": "ISS",
      "name": "Jeanette Epps"
    },
    {
      "craft": "ISS",
      "name": "Alexander Grebenkin"
    },
    {
      "craft": "ISS",
      "name": "Butch Wilmore"
    },
    {
      "craft": "ISS",
      "name": "Sunita Williams"
    },
    {
      "craft": "Tiangong",
      "name": "Li Guangsu"
    },
    {
      "craft": "Tiangong",
      "name": "Li Cong"
    },
    {
      "craft": "Tiangong",
      "name": "Ye Guangfu"
    }
  ]
}
```




```
   ┌∘┬ISS
   │ └Oleg Kononenko
   ├∘┬ISS
   │ └Nikolai Chub
   ├∘┬ISS
   │ └Tracy Caldwell Dyson
   ├∘┬ISS
   │ └Matthew Dominick
   ├∘┬ISS
   │ └Michael Barratt
   ├∘┬ISS
   │ └Jeanette Epps
 ┌∘┤
∘┤ ├∘┬ISS
 │ │ └Alexander Grebenkin
 │ ├∘┬ISS
 │ │ └Butch Wilmore
 │ ├∘┬ISS
 │ │ └Sunita Williams
 │ ├∘┬Tiangong
 │ │ └Li Guangsu
 │ ├∘┬Tiangong
 │ │ └Li Cong
 │ └∘┬Tiangong
 │   └Ye Guangfu
 ├12
 └success
```

```json
{
  "ISS": [
    "Oleg Kononenko",
    "Nikolai Chub",
    "Tracy Caldwell Dyson",
    "Matthew Dominick",
    "Michael Barratt",
    "Jeanette Epps",
    "Alexander Grebenkin",
    "Butch Wilmore",
    "Sunita Williams"
  ],
  "Tiangong": [
    "Li Guangsu",
    "Li Cong",
    "Ye Guangfu"
  ]
}
```

Finding Leaves

We can create a vector of all the node IDs in a tree, as it is just every index in `p`.

```
⍳≢p
```

```
0 1 2 3 4 5 6 7 8 9 10 11
```

The leaf nodes are those nodes which do not have any children, i.e. those nodes which are not pointed to in the parent vector. We can think of the parent vector as a list of all nodes which are not leaves, and remove them from a list of all nodes to obtain only the leaf nodes:

```
(⍳≢p)~p
```

```
3 4 5 8 9 10 11
```

Alternatively, if we want a mask of leaf nodes, we just mask those nodes which are not in the parent vector:

```
~(⍳≢p)∊p
```

```
0 0 0 1 1 1 0 0 1 1 1 1
```

- Tutorials
- Documentation
- ‘Literate Programming’

````
# Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of $n$ nodes, we associate each node with an index in `ιn`, and create an $n$-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

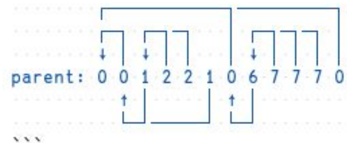
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
````




````
```

In this section, we're going to look at some of the basic operations you can do on trees represented in this way.
````

jupyter {book}

~~~
# Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of $n$ nodes, we associate each node with an index
↪ in `⍳n`, and create an $n$-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

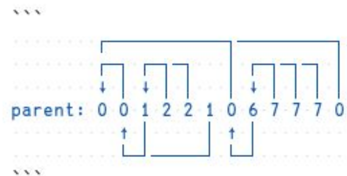
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
~~~



~~~
```

In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.
~~~

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

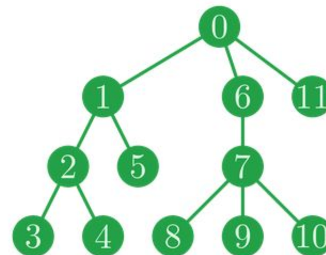


Fig. 9 The tree we're going to work with in this section.

and the parent vector representing this tree:

```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```

In this section, we're going to look at some of the basic operations you can do on trees represented in this way.

jupyter {book}

````
# Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of $n$ nodes, we associate each node with an index
↪ in `ιn`, and create an $n$-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

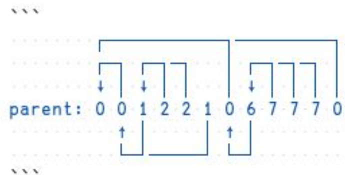
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
````



````
```

In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.
````

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `ιn`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

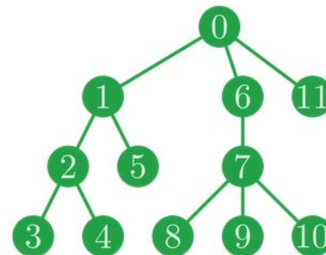
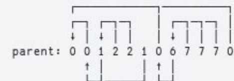


Fig. 9 The tree we're going to work with in this section.

and the parent vector representing this tree:

```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```



In this section, we're going to look at some of the basic operations you can do on trees represented in this way.

jupyter{book}

```
# Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of $n$ nodes, we associate each node with an index
↪ in `ιn`, and create an $n$-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

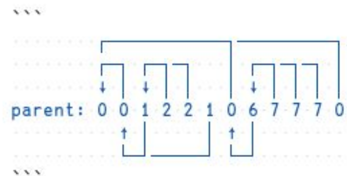
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
```



```
parent: 0 0 1 2 2 1 0 6 7 7 7 0
```

```
In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.
```

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `ιn`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

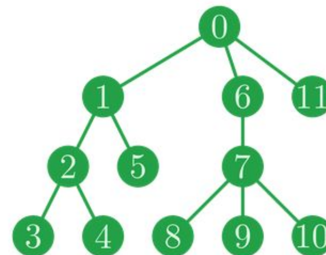
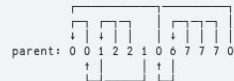


Fig. 9 The tree we're going to work with in this section.

and the parent vector representing this tree:

```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```



```
parent: 0 0 1 2 2 1 0 6 7 7 7 0
```

In this section, we're going to look at some of the basic operations you can do on trees represented in this way.
```

jupyter{book}

````
# Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of $n$ nodes, we associate each node with an index
↪ in `⍳n`, and create an $n$-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

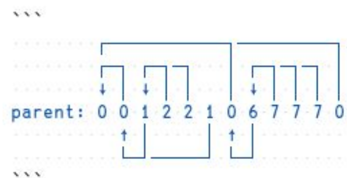
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
````



````
```

In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.
````

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

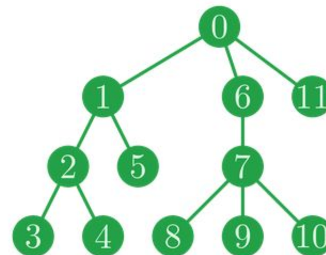
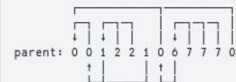


Fig. 9 The tree we're going to work with in this section.

and the parent vector representing this tree:

```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```



In this section, we're going to look at some of the basic operations you can do on trees represented in this way.

jupyter{book}

Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of n nodes, we associate each node with an index
↪ in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

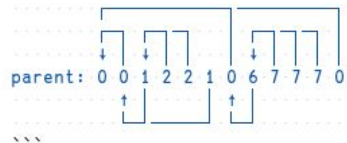
~~~
```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```
~~~

and the parent vector representing this tree:

~~~
```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
~~~




~~~
```
~~~

In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

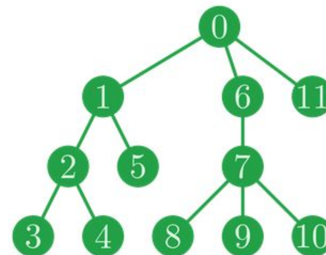
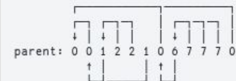


Fig. 9 The tree we're going to work with in this section.

and the parent vector representing this tree:

```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

```
0 0 1 2 2 1 0 6 7 7 7 0
```



In this section, we're going to look at some of the basic operations you can do on trees represented in this way.

https://dyalog.github.io/dyalog-jupyter-kernel/

Parent Vectors

At the end of the previous section we settled on the parent vector representation
↪ as the main representation of trees we will be using in this tutorial. To
↪ reiterate, to represent a tree of n nodes, we associate each node with an index
↪ in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a
↪ child of a node `j`, then `parent[i]=j`.

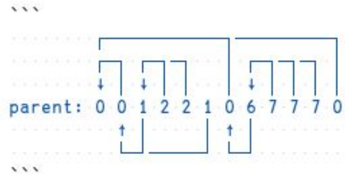
We're going to use a slightly larger tree for the examples in this section:

```{figure} media/PV_ManimCE_v0.18.1.png
:alt: A diagram of a new tree.

The tree we're going to work with in this section.
```

and the parent vector representing this tree:

```{code-cell}
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```



In this section, we're going to look at some of the basic operations you can do on
↪ trees represented in this way.

Parent Vectors

At the end of the previous section we settled on the parent vector representation as the main representation of trees we will be using in this tutorial. To reiterate, to represent a tree of n nodes, we associate each node with an index in `⍳n`, and create an n-element vector `parent` such that if a node `i` is a child of a node `j`, then `parent[i]=j`.

We're going to use a slightly larger tree for the examples in this section:

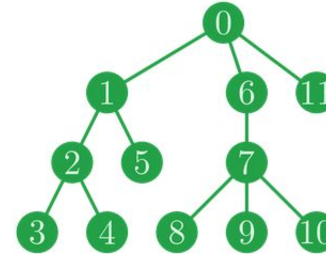
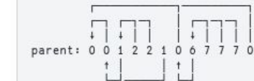


Fig. 9 The tree we're going to work with in this section.

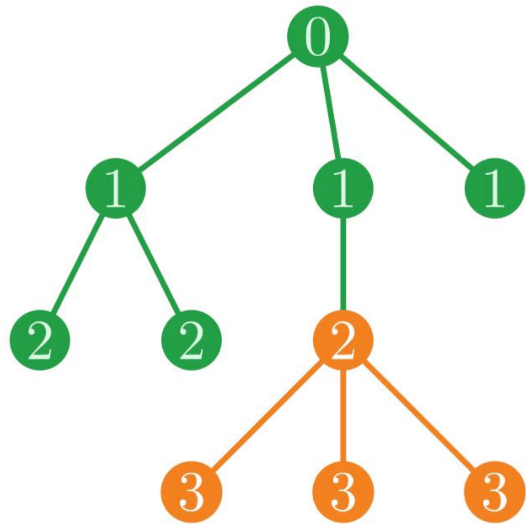
and the parent vector representing this tree:

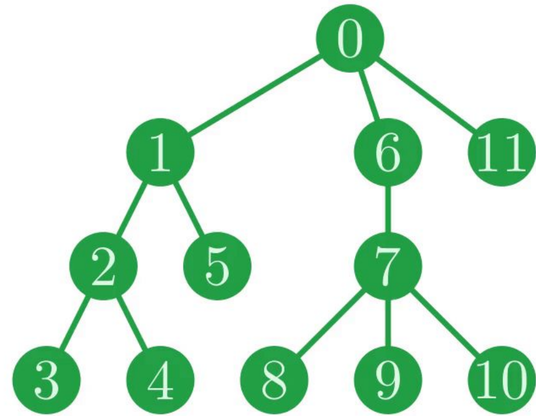
```
⊢parent←0 0 1 2 2 1 0 6 7 7 7 0
```

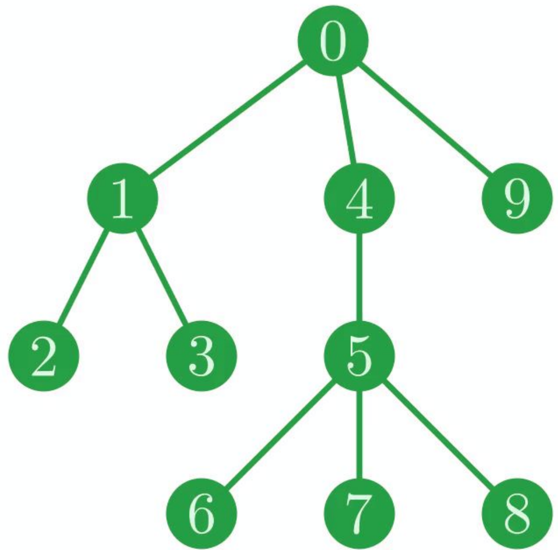
```
0 0 1 2 2 1 0 6 7 7 7 0
```

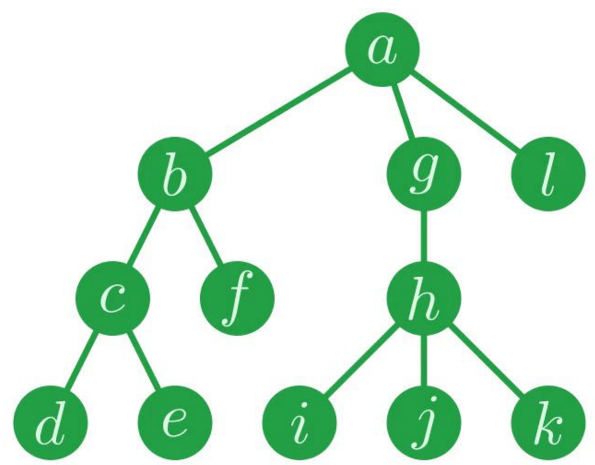


In this section, we're going to look at some of the basic operations you can do on trees represented in this way.


0
1
1
1
2
2
2
3
3
3


0
1
6
11
2
5
7
3
4
8
9
10

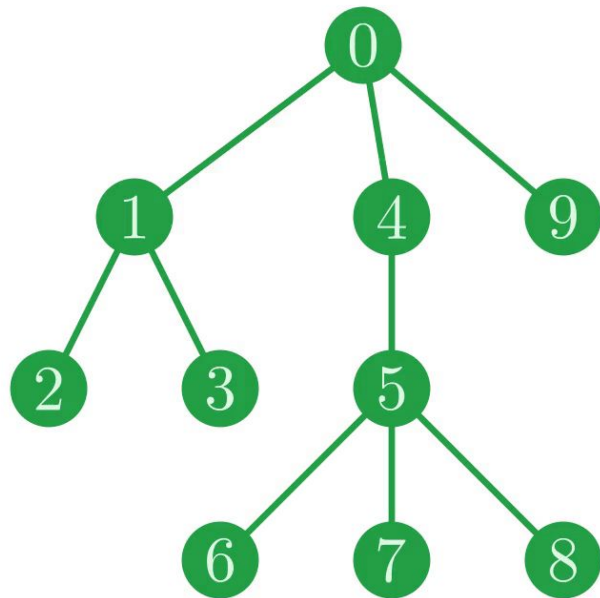

0
1
4
9
2
3
5
6
7
8


a
b
g
l
c
f
h
d
e
i
j
k

```python
class PVInvert(Scene):
    def construct(self):
        p = [ 0,   0,   1,   2,   2,   1,   0,   6,   7,   7,   7,    0]
        v = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11']
        t = tree(
            p,
            v,
            scale = 1.5,
        )
        pp = [11, 4, 4, 4, 5, 11, 10, 9, 9, 10, 11, 11]
        vv = ['11', '10', '9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
        tt = tree(
            pp,
            vv,
            scale = 1.5,
        )
        self.add(t)
        self.play(Wait(2))
        self.play(*(
            t[i].animate.move_to(tt[len(p) - i - 1]) for i in range(len(p))
        ))
        self.play(Wait(2))
```

Manim

```python
class PVInvert(Scene):
    def construct(self):
        p = [ 0,   0,   1,   2,   2,   1,   0,   6,   7,   7,   7,    0]
        v = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11']
        t = tree(
            p,
            v,
            scale = 1.5,
        )
        pp = [11, 4, 4, 4, 5, 11, 10, 9, 9, 10, 11, 11]
        vv = ['11', '10', '9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
        tt = tree(
            pp,
            vv,
            scale = 1.5,
        )
        self.add(t)
        self.play(Wait(2))
        self.play(*(
            t[i].animate.move_to(tt[len(p) - i - 1]) for i in range(len(p))
        ))
        self.play(Wait(2))
```



```
               siblings of i
                    ┌─┴─┐
depth: 0 1 2 2 1 2 3 3 3 1
                 ↑   ↑   ↑
    parent of i ─┘   i   └─ start of next sub-tree
```

```
index:     0 1 2 3 4 5 6 7 8 9
p:         1 7 4 1 7 6 3 7 6 1
mask:      0 0 0 1 0 1 1 0 1 0
                 ↑   ↑ ↑ ↑
  to-be-deleted ─┴───┴─┘ │
                     ┌───┘
                     ↓
(~mask)/p: 1 7 4 7 7 1
```

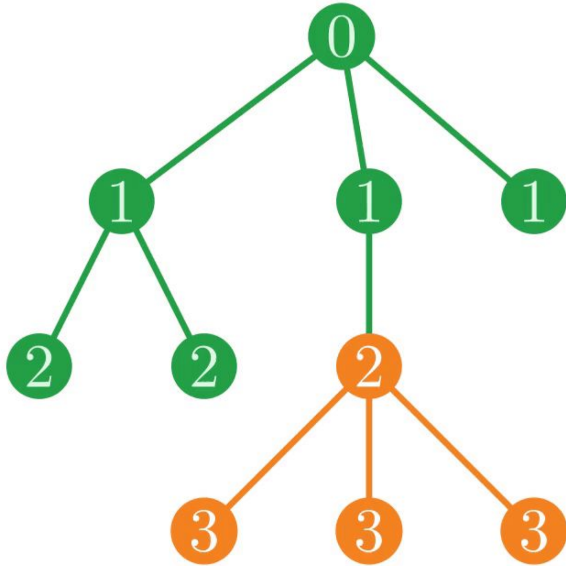
```
        ┌─┐ ┌───┐ ┌─┬─┬─┐
        ↓ │     │ ↓ │ │ │
parent: 0 0 1 1 0 4 5 5 5 0
        ↑ ↑ │ │ ↑ │       │
        │ └─┘─┘ └─┘       │
        └─────────────────┘
```

```
┌──────────────────┐
│        0         │
│   ┌────┼────┐    │
│   1    6    11   │
│ ┌─┴─┐  │         │
│ 2   5  7         │
│┌┴┐   ┌─┼─┐       │
│3 4   8 9 10      │
└──────────────────┘
```

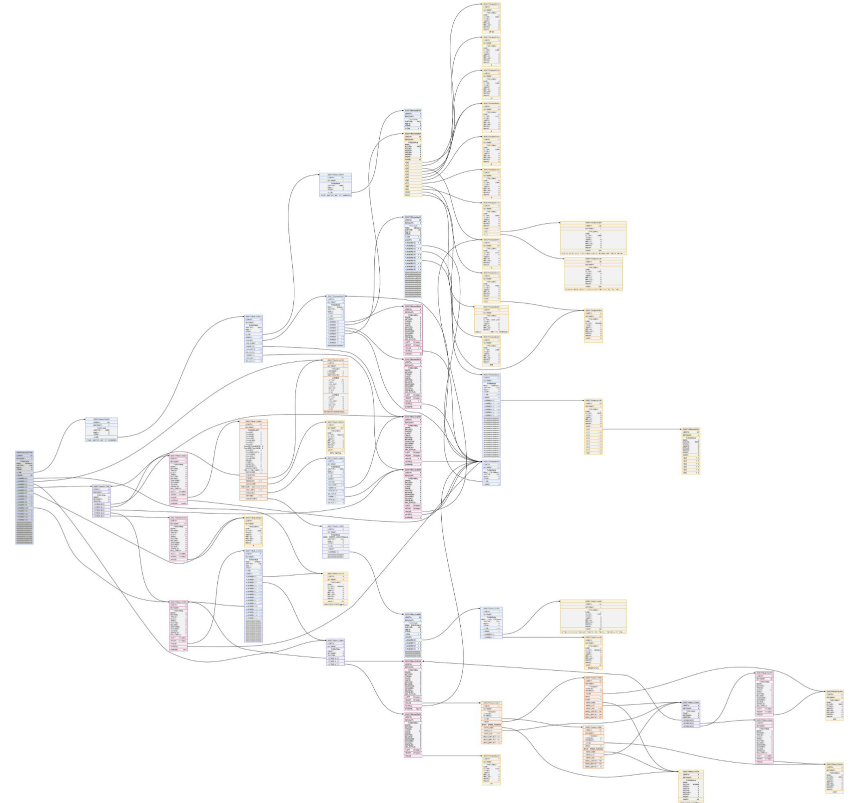
```
4
4 → 2
4 → 2 → 0
4 → 2 → 7
4 → 8
4 → 8 → 1
...
```

https://asherbhs.github.io/apl-site/trees/intro.html

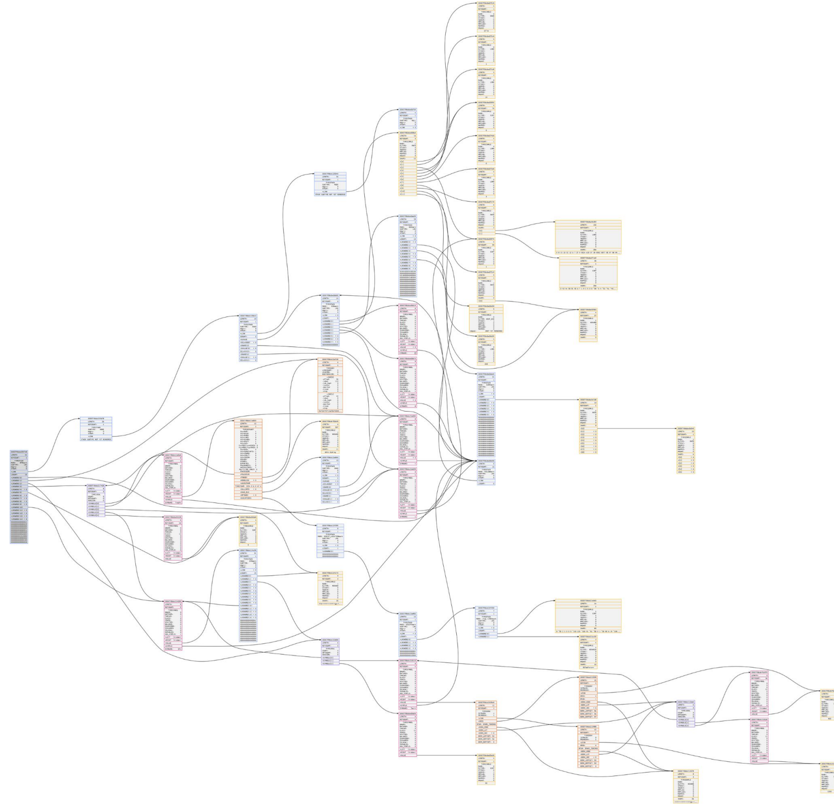
~~Climbing Trees~~



Catching Bugs



Catching Bugs



1. Broke the QA tests

1. Broke the QA tests
2. Broke the QA tests again

1. Broke the QA tests
2. Broke the QA tests again
3. Broke the QA tests AGAIN

TRANSLATION ERROR: Unicode character
□UCS 9060 (U+2364) not in □AVU

```
00007f855e3a6e00 fffffffffffffffb 0000000000000001 000000000000271f
simple                          5 00000000001a7248  Vc
            6e18 0000000000000007 006e666461727420
                                    t r a d f n
00007f855e3a6e28 fffffffffffffffa 0000000000000002 000000000000c001
symbol                          6 00000000001a7270         C
            6e40 0000000000000000 0000000000000000 00007f855e3a6e00
                 *<=              *=>              *value
00007f855e3a6e58 ffffffffffffffd9 0000000000000001 000000000009009e
body                           39 00000000001a72a0  <#>
            6e70 006a0000004b7000 00c8000000a20000 00f2000000e30000
                 [     1][     0] [     3][     2] [     5][     4]
            6e88 012c000001150000 3a70670a01330000 0033016f1b4a0000
                 [     7][     6]         [     8]
            6ea0 5601670e6f235800 6900004c00324a2c 084c01724c036f1c
            6eb8 000033010d03005e 201f6b670e6f2378 a100004c001b3b02
            6ed0 4c006021742c6f1c 6021614c01474c02 071e57064a405705
            6ee8 027821614c013357 000033010d09005e 560a60670e6f23b9
            6f00 324a2c61560b6061 00004c011f214c00 005e024c016f1cc7
            6f18 23d4000033010d0c 011f15570d670e6f 570e6f1ce200004c
            6f30 003301570f4a3d67 3a80670e6f23f100 010033014c014f81
            6f48 155712670e6f2307 022a331b0b022a33 1c1401004c011b0b
            6f60 010d13005e07806f 670e6f232b010033 1621852180215714
            6f78 33010d17005e0457 00670a6f23320100 000000006f1f4300
00007f855e3a6f90 ffffffffffffffe1 0000000000000001 0000000030005145
stack                          31 00000000001a73d8 DFN
            6fa8 0000000000000000 000000000000000f 00007f855e3a6e58
                 *LINK            count            *DefCo
            6fc0 0000000000000000 00007f855e3a5db8 0000000000000000
                 *RefCo           *DefLs           *RefLs
            6fd8 0000000000000000 0000000000000000 0000000000000000
                 *Alpha           *Omega           *Oland
            6ff0 0000000000000000 0000000000000000 0000000000000000
                 *Orand           *Static          *Ptokn/Utokn
            7008 00007f855e3a3428 00007f855e352b30 0000000000000000
                 *Name            *User            *Monitor
            7020 0000000000000000 00007f855e3a7088 000000000000004a
                 *OptCo           *ScriptInfo      DefOs
            7038 0000000000000000 0000000000000000 0000000000000000
```

```
00007f5a7c8842f0 fffffffffffffff2 0000000000000001 000000000000281f
simple                         14 0000000000084738  Vc
            4308 0000000000000026 0020002000200020 0041239500200020
                                                            □   A
            4320 007e2368002f0056 2374237323680027 220a003022182373
                   V   /   ̈   ~    '   ̈   ι   ρ    ι   ∘   0   ϵ
            4338 2218003000282364 00a800292355002c 0030003000382373
                   ̈   (   0   ∘    ,   ∓   )   ¨    ι   8   0   0
            4350 2395220a00270032 0000000000560041
                   2   '   ϵ   □    A   V
```

```
00007f5a7c8842f0 fffffffffffffff2 0000000000000001 000000000000281f
simple                         14 0000000000084738  Vc
            4308 0000000000000026 0020002000200020 0041239500200020
                                                          ⎕     A
            4320 007e2368002f0056 2374237323680027 220a003022182373
                    V     /     ⍨     ~     '     ⍨     ⍳     ⍴     ⍳     ∘     0     ∊
            4338 2218003000282364 00a800292355002c 0030003000382373
                    ⍤     (     0     ∘     ,     ⍕     )     ¨     ⍳     8     0     0
            4350 2395220a00270032 0000000000560041
                    2     '     ∊     ⎕     A     V
```

```
00007f5a7c8842f0
LENGTH:        14
REFCOUNT:       1
   TYPESIMPLE
RANK:           1
ELTYPE:   WCHAR16
STICKY:         0
SQUOZE:         1
MMFLAG:         0
MMFLAG2:        0
MAPPED:         0
mmpad:          0
SHAPE:         38
⎕AV/⍨~'⍨⍳⍴⍳∘0∊...
```

00007f5a7c884288	
LENGTH:	7
REFCOUNT:	1
TYPEGEN	
RANK:	1
ELTYPE:	PNTR
STICKY:	0
SQUOZE:	0
MMFLAG:	0
MAPPED:	0
mmpad:	0
SHAPE:	3
*[0]	(hidden)
*[1]	
*[2]	(hidden)

00007f5a7c8842f0	
LENGTH:	14
REFCOUNT:	1
TYPESIMPLE	
RANK:	1
ELTYPE:	WCHAR16
STICKY:	0
SQUOZE:	1
MMFLAG:	0
MMFLAG2:	0
MAPPED:	0
mmpad:	0
SHAPE:	38
⎕AV/⍨~'⍨⍳⍴⍳∘0∊...	

00007f5a7c884120
LENGTH: 6
REFCOUNT: 1
TYPEFPTR
FSUB: SYSPROP
version: 1
*FVALUE
id: 7
*inf (hidden)

00007f5a7c884288
LENGTH: 7
REFCOUNT: 1
TYPEGEN
RANK: 1
ELTYPE: PNTR
STICKY: 0
SQUOZE: 0
MMFLAG: 0
MAPPED: 0
mmpad: 0
SHAPE: 3
*[0] (hidden)
*[1]
*[2] (hidden)

00007f5a7c8842f0
LENGTH: 14
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR16
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 38
⎕AV/⍨~'⍨⍳⍴⍳∘0∊...

```
00007f5a7c884120
LENGTH:      6
REFCOUNT:    1
TYPEFPTR
FSUB:   SYSPROP
version:     1
*FVALUE
id:          7
*inf

00007f5a7c884288
LENGTH:      7
REFCOUNT:    1
TYPEGEN
RANK:        1
ELTYPE:   PNTR
STICKY:      0
SQUOZE:      0
MMFLAG:      0
MAPPED:      0
mmpad:       0
SHAPE:       3
*[0]
*[1]
*[2]

00007f5a7c8842c0
LENGTH:      6
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE:  WCHAR8
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      12
LENGTH ERROR

00007f5a7c8842f0
LENGTH:     14
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE: WCHAR16
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      38
⎕AV/⍨~'⍨⍳⍴⍳∘0∊...

00007f5a7c884360
LENGTH:      7
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE: WCHAR16
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      10
^

00007f5a7c8838a8
LENGTH:     19
REFCOUNT:   10
TYPESTACK
MARK:   DMXmark
SUBTYPE:    LNS
magic:       0
XTRAP:       0
version:     0
*LINK      = 0
LCOUNT:      7
*LNSWORD(0) = 0
*LNSWORD(1) = 0
*LNSWORD(2) = 0
*LNSWORD(3)
*LNSWORD(4)
*LNSWORD(5)
*LNSWORD(6)
0000000000000000
0000000000000005
0000000000000000
0000000000000000
0000000000000000
000000000000042d
000000000000000a

00007f5a7c883940
LENGTH:     68
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE: WCHAR32
STICKY:      1
SQUOZE:      0
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:     128
???????????????????...

00007f5a7c883b60
LENGTH:     26
REFCOUNT:    1
TYPESTACK
MARK:  NULLmark
SUBTYPE:   MODE
magic:       0
XTRAP:       0
version:     0
*LINK      = 0
*MNAME/HSTACK/MMAGSPL = 0
MLINE: 1.229782938E19
*MFUNC     = 0
*MCODE
MPC:        51
*MLST
_MFLAG: 1.229782938E19
MPTOK/MUTOK: SPARE_TOKEN00
*MSPACE    = 0
MCPU[0]: 1.229782938E19
MCPU[1]: 1.229782938E19
MELA[0]: 1.229782938E19
MELA[1]: 1.229782938E19
JWIN:        0
*FLST      = 0
*CNTRL     = 0
MFLAGS: 0000000000000000
        0000000000000000
        0000000000000000
*MCLASSDEF = 0
MCARETPOS: 1.229782938E19
MERRORTASK:  0

00007f5a7c8827a0
LENGTH:      8
REFCOUNT:    2
TYPESIMPLE
RANK:        1
ELTYPE:   NCHAR
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      27
NOT YET RENDERED

00007f5a7c882780
LENGTH:      4
REFCOUNT:    2
TYPELOCAL
COUNT:       1
OPTCOUNT:    0
VERSION:     0
*SYMBOLS[0]

00007f5a7c882828
LENGTH:      6
REFCOUNT:    2
TYPESYMBOL
QNAME:       0
WATCHED:     0
THREAD:      0
CLASS:      12
JVWID:       0
VISITED:     0
BALANCE:     0
ISUNICODE:   0
ISSHARED:    0
ISLOCAL:     0
ISPUBLIC:    0
HAS_SYMFLO:  0
*LEFT   (hidden)
*RIGHT  (hidden)
*VALUE

00007f5a7c8827e0
LENGTH:      9
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE: WCHAR16
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      20
⍨⍳⍴⍳∘0∊?(0∘,⍵)¨⍳8002

00007f5a7c883c30
LENGTH:      5
REFCOUNT:    2
TYPESIMPLE
RANK:        1
ELTYPE:  WCHAR8
STICKY:      0
SQUOZE:      1
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:       6
Dyalog

00007f5a7c883c58
LENGTH:     12
REFCOUNT:    1
TYPESIMPLE
RANK:        1
ELTYPE:  WCHAR8
STICKY:      1
SQUOZE:      0
MMFLAG:      0
MMFLAG2:     0
MAPPED:      0
mmpad:       0
SHAPE:      64
replicate.cpp???????...
```

00007f5a7c884120
LENGTH: 6
REFCOUNT: 1
TYPEFPTR
FSUB: SYSPROP
version: 1
*FVALUE
id: 7
*inf

00007f5a7c884288
LENGTH: 7
REFCOUNT: 1
TYPEGEN
RANK: 1
ELTYPE: PNTR
STICKY: 0
SQUOZE: 0
MMFLAG: 0
MAPPED: 0
mmpad: 0
SHAPE: 3
*[0]
*[1]
*[2]

00007f5a7c8842c0
LENGTH: 6
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR8
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 12
LENGTH ERROR

00007f5a7c8842f0
LENGTH: 14
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR16
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 38
⎕AV/⍨~'⍨⍳⍴⍳∘0∊...

00007f5a7c884360
LENGTH: 7
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR16
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 10
^

00007f5a7c8838a8
LENGTH: 19
REFCOUNT: 10
TYPESTACK
MARK: DMXmark
SUBTYPE: LNS
magic: 0
XTRAP: 0
version: 0
*LINK = 0
[illegible]UNT: [illegible]
*LNSWORD(0) = 0
*LNSWORD(1) = 0
*LNSWORD(2) = 0
*LNSWORD(3)
*LNSWORD(4)
*LNSWORD(5)
*LNSWORD(6)
0000000000000000
0000000000000005
0000000000000000
0000000000000000
0000000000000000
000000000000042d
000000000000000a

00007f5a7c883940
LENGTH: 68
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR32
STICKY: 1
SQUOZE: 0
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 128
????????????????????...

00007f5a7c883b60
LENGTH: 26
REFCOUNT: 1
TYPESTACK
MARK: NULLmark
SUBTYPE: MODE
magic: 0
XTRAP: 0
version: 0
*LINK = 0
*MNAME/HSTACK/MMAGSPL = 0
MLINE: 1.229782938E19
*MFUNC = 0
*MCODE
MPC: 51
*MLST
_MFLAG: 1.229782938E19
MPTOK/MUTOK: SPARE_TOKEN00
*MSPACE = 0
MCPU[0]: 1.229782938E19
MCPU[1]: 1.229782938E19
MELA[0]: 1.229782938E19
MELA[1]: 1.229782938E19
JWIN: 0
*FLST = 0
*CNTRL = 0
MFLAGS: 0000000000000000
0000000000000000
0000000000000000
*MCLASSDEF = 0
MCARETPOS: 1.229782938E19
MERRORTASK: 0

00007f5a7c8827a0
LENGTH: 8
REFCOUNT: 2
TYPESIMPLE
RANK: 1
ELTYPE: NCHAR
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 27
NOT YET RENDERED

00007f5a7c882780
LENGTH: 4
REFCOUNT: 2
TYPELOCAL
COUNT: 1
OPTCOUNT: 0
VERSION: 0
*SYMBOLS[0]

00007f5a7c882828
LENGTH: 6
REFCOUNT: 2
TYPESYMBOL
QNAME: 0
WATCHED: 0
THREAD: 0
CLASS: 12
JVWID: 0
VISITED: 0
BALANCE: 0
ISUNICODE: 0
ISSHARED: 0
ISLOCAL: 0
ISPUBLIC: 0
HAS_SYMFLO: 0
*LEFT (hidden)
*RIGHT (hidden)
*VALUE

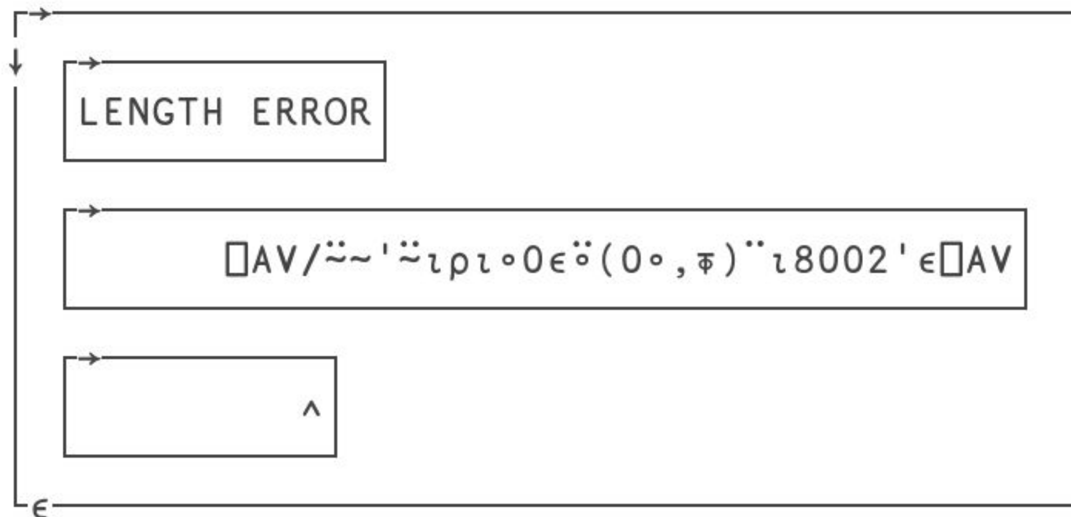
00007f5a7c8827e0
LENGTH: 9
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR16
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 20
⍨⍳⍴⍳∘0∊?(0∘,⍵)¨⍳8002

00007f5a7c883c30
LENGTH: 5
REFCOUNT: 2
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR8
STICKY: 0
SQUOZE: 1
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 6
Dyalog

00007f5a7c883c58
LENGTH: 12
REFCOUNT: 1
TYPESIMPLE
RANK: 1
ELTYPE: WCHAR8
STICKY: 1
SQUOZE: 0
MMFLAG: 0
MMFLAG2: 0
MAPPED: 0
mmpad: 0
SHAPE: 64
replicate.cpp???????...

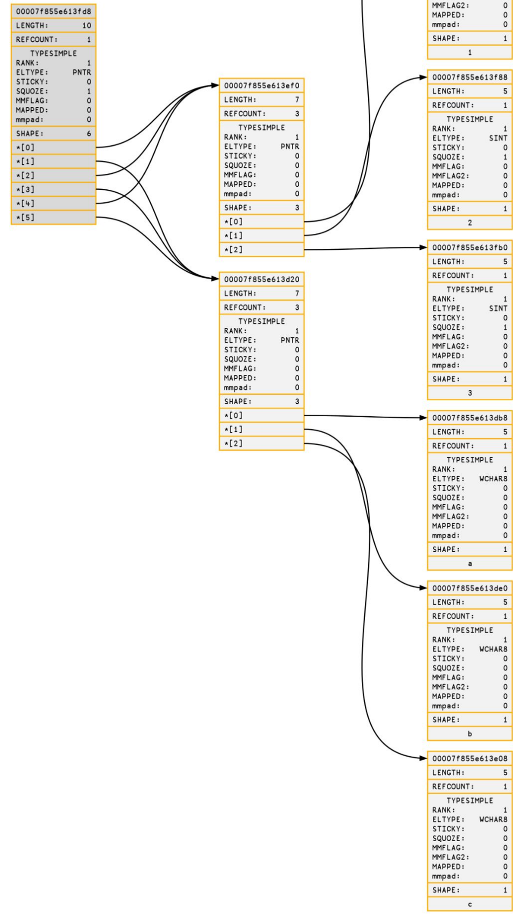
00007f5a7c8838a8	
LENGTH:	19
REFCOUNT:	10
TYPESTACK	
MARK:	DMXmark
SUBTYPE:	LNS
magic:	0
XTRAP:	0
version:	0
*LINK	= 0
LCOUNT:	7

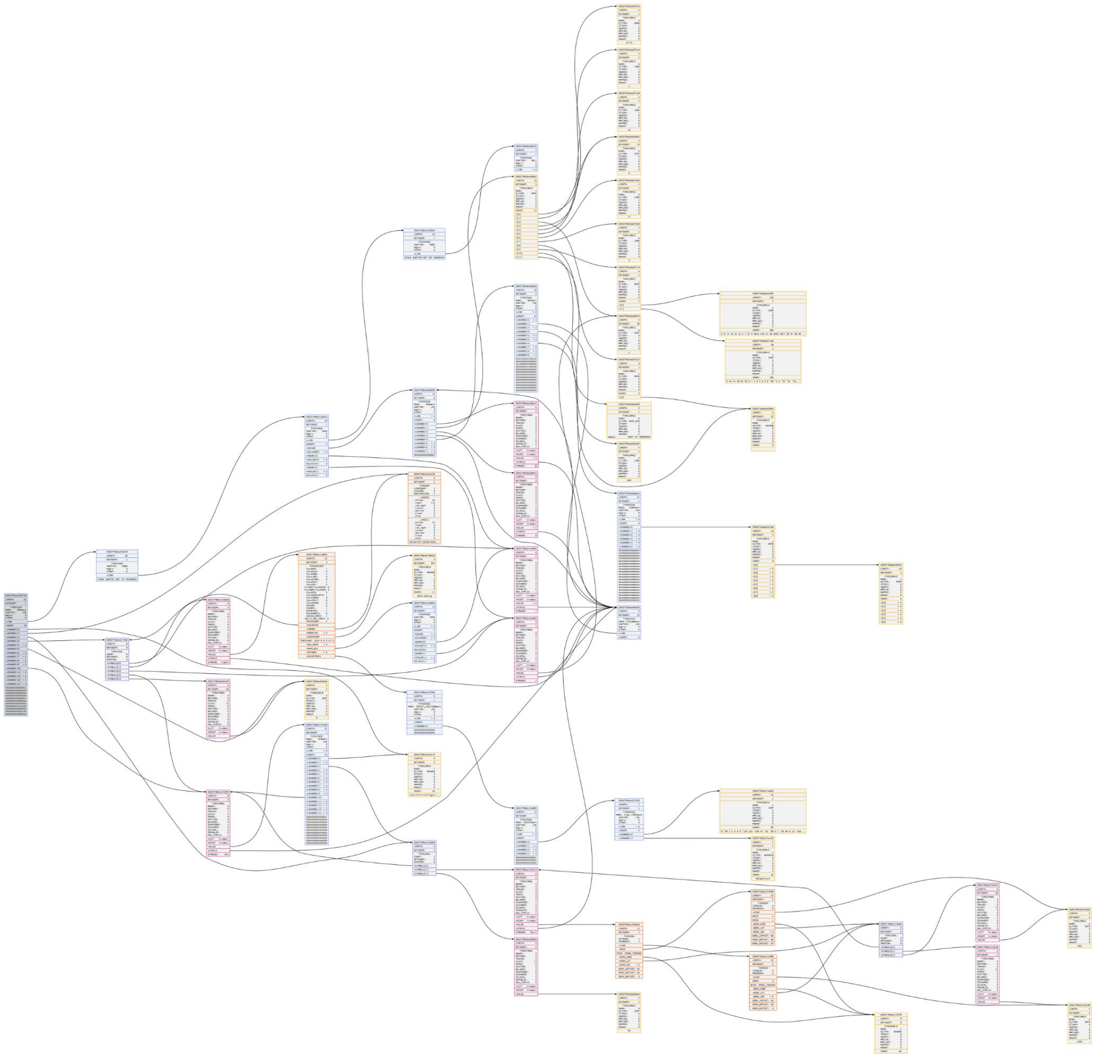
00007f5a7c8838a8	
LENGTH:	19
REFCOUNT:	10
TYPESTACK	
MARK:	DMXmark
SUBTYPE:	LNS
magic:	0
XTRAP:	0
version:	0
*LINK	= 0
LCOUNT:	7
*LNSWORD(0)	= 0



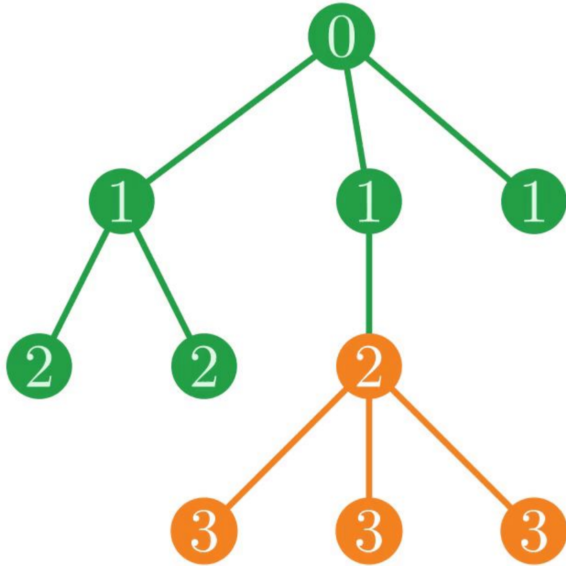
1. Broke the QA tests
2. Broke the QA tests again
3. Broke the QA tests AGAIN
4. Fixed the QA tests!
5. Made a tool
6. …

```
00007f855e3a6e00 fffffffffffffffb 0000000000000001 000000000000271f
simple                          5 00000000001a7248  Vc
            6e18 0000000000000007 006e666461727420
                                    t r a d f n
00007f855e3a6e28 fffffffffffffffa 0000000000000002 000000000000c001
symbol                          6 00000000001a7270           C
            6e40 0000000000000000 0000000000000000 00007f855e3a6e00
                 *<=              *=>              *value
00007f855e3a6e58 ffffffffffffffd9 0000000000000001 000000000009009e
body                           39 00000000001a72a0  <#>
            6e70 006a0000004b7000 00c8000000a20000 00f2000000e30000
                 [      1][     0] [      3][     2] [      5][     4]
            6e88 012c000001150000 3a70670a01330000 0033016f1b4a0000
                 [      7][     6]          [     8]
            6ea0 5601670e6f235800 6900004c00324a2c 084c01724c036f1c
            6eb8 000033010d03005e 201f6b670e6f2378 a100004c001b3b02
            6ed0 4c006021742c6f1c 6021614c01474c02 071e57064a405705
            6ee8 027821614c013357 000033010d09005e 560a60670e6f23b9
            6f00 324a2c61560b6061 00004c011f214c00 005e024c016f1cc7
            6f18 23d4000033010d0c 011f15570d670e6f 570e6f1ce200004c
            6f30 003301570f4a3d67 3a80670e6f23f100 010033014c014f81
            6f48 155712670e6f2307 022a331b0b022a33 1c1401004c011b0b
            6f60 010d13005e07806f 670e6f232b010033 1621852180215714
            6f78 33010d17005e0457 00670a6f23320100 000000006f1f4300
00007f855e3a6f90 ffffffffffffffe1 0000000000000001 0000000030005145
stack                          31 00000000001a73d8 DFN
            6fa8 0000000000000000 000000000000000f 00007f855e3a6e58
                 *LINK            count            *DefCo
            6fc0 0000000000000000 00007f855e3a5db8 0000000000000000
                 *RefCo           *DefLs           *RefLs
            6fd8 0000000000000000 0000000000000000 0000000000000000
                 *Alpha           *Omega           *Oland
            6ff0 0000000000000000 0000000000000000 0000000000000000
                 *Orand           *Static          *Ptokn/Utokn
            7008 00007f855e3a3428 00007f855e352b30 0000000000000000
                 *Name            *User            *Monitor
            7020 0000000000000000 00007f855e3a7088 000000000000004a
                 *OptCo           *ScriptInfo      DefOs
            7038 0000000000000000 0000000000000000 0000000000000000
```

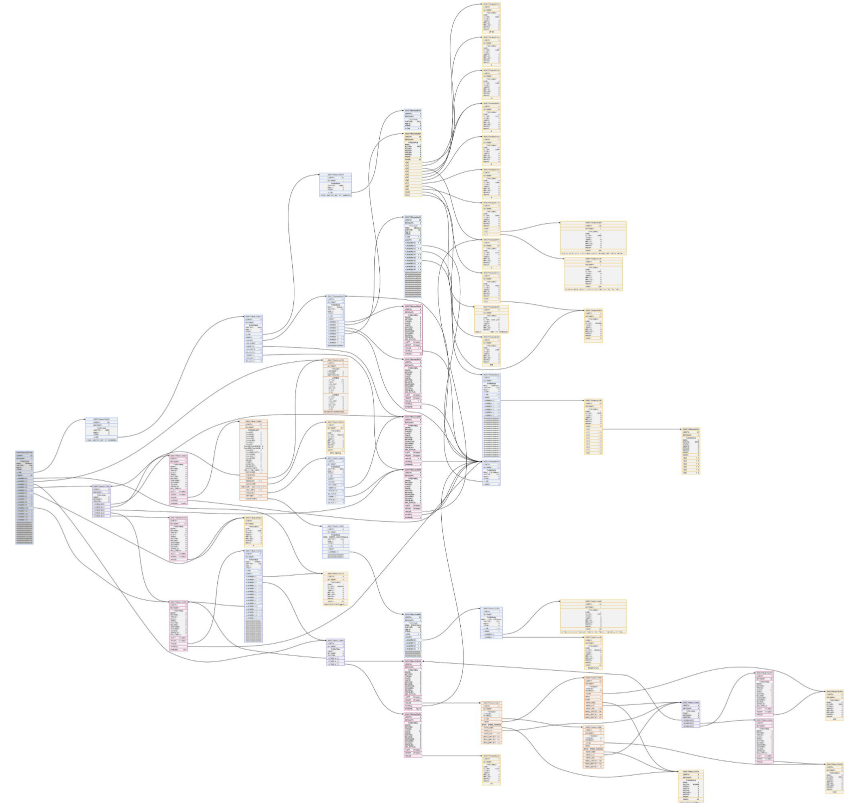




~~Climbing Trees~~



~~Catching Bugs~~



https://asherbhs.github.io/apl-site/trees/intro.html