Skill Pill: Julia

Lecture 3: Data Structures and Algorithms

Valentin Churavy James Schloss

Okinawa Institute of Science and Technology
valentin.churavy@oist.jp
james.schloss@oist.jp

July 7, 2017

Disclaimer

When designing this Skill Pill, we assumed the following

- You have seen and are familiar with common data structures
- You know how to program and use programming as part of your daily work.

As such, we have designed today's lesson so that you may begin using Julia in your work as soon as possible.

Datastructures.jl

DataStructures.jl has the following data structures:

- Deque (based on block-list)
- Stacks and Queues
- Accumulators and Counters
- Disjoint Sets
- Binary Heap
- Mutable Binary Heap
- Ordered Dicts and Sets
- Dictionaries with Defaults
- Trie (Tree)
- Linked List
- Sorted Dict, Multi-Dict and Set
- Priority Queue

Datastructures.jl

DataStructures.jl has the following data structures:

- Deque (based on block-list)
- Stacks and Queues
- Accumulators and Counters
- Disjoint Sets
- Binary Heap
- Mutable Binary Heap
- Ordered Dicts and Sets
- Dictionaries with Defaults
- Trie (Tree)
- Linked List
- Sorted Dict, Multi-Dict and Set
- Priority Queue

All information regarding these Data Structures can be found Here: `http://datastructuresjl.readthedocs.io/en/latest/index.html/`

All of these algorithms can be viewed with `@edit`. We'll use two more before hopping into algorithms: `Binary Trees and Priority Queues`

Using simple trees

First, let's get used to using DataStructures.jl by Depth-First searching in binary trees. You might be used to binary tree nodes that look like this (C++):

```
struct node{
    double weight;

    node *left;
    node *right;
    node *parent;
};
```

Basically, each node has parents and children.

Using simple trees

In Julia, the tree nodes might look like:

```
typealias BT{T} Union{T,Empty}

type BTree
    weight::Float64
    left::BT{BTree}
    right::BT{BTree}
end
```

Depth-First-Search

To search through a tree, we need a **recursive** strategy. One of these strategies is known as **Depth-First Search** (or corresponding **Breadth-First Search**).

These searching algorithms always go from the root → nodes. In C++:

```
void depth_first_search(node* &node){
    if (root->right){
        depth_first_search(node->right);
    }
    if (root->left){
        depth_first_search(node->left);
    }
}
```

Depth-First Search

In Julia, this looks like:

```
function DFS(node::BTree)
    if (node.right != et)
        DFS(node.right)
    end

    if (node.left) != et
        DFS(node.left)
    end
end
```

Exercise

Write a DFS that acts on a binary tree and output a binary string that traverses to a leaf node.

PriorityQueue

`Priority Queues` are found in the `DataStructures.jl` Package:

```
julia> using DataStructures

julia> pq = PriorityQueue()

julia> pq["a"] = 15
julia> pq["b"] = 20

julia> pq
DataStructures.PriorityQueue{Any,Any,Base.Order.ForwardOrdering}
    with 2 entries:
  "b" => 20
  "a" => 15

julia> dequeue!(pq)
"a"
```

Other Data Structures

Obviously, all your favorite data structures can be implemented in Julia, but for now, we will move on to...

Algorithms in Julia

Implement your favorite algorithm in Julia

PS: We're here to help!

What is next?

Last Session Parallel computing, threading, GPUs? Up to grabs.

Join us for the exciting conclusion!