

U06SGCEHJ : thanks! I'll let <@U07BWACEA> review those changes, he's the primary on the GSoC stuff

U46LFMYTD : I am constructing a nested datastructure, similar to a binary tree, recursively using``

```
(defn make-tree [{:in :in out :out left :left right :right :as tree} indices]
  (cond
    (empty? indices) tree
    :else (assoc
      tree
      :left (make-tree {:in (conj in (first indices)) :out out} (rest indices))
      :right (make-tree {:in in :out (conj out (first indices))} (rest indices))))))
...

```

U46LFMYTD : however the recursion will run quite deep and cause stack overflow. Is there a lazy way I can define this tree? I will not actually visit all of the tree

U46LFMYTD : In short it sets up a deeply nested map, but I want to do this in a lazy way so that the values mapped to by keys are not evaluated until accessed

U0EJ065V2 : You can do tail recursion for the stack overflow but that won't help with the lazy part.

U3L6TFEJF : michaellindon: There's <<https://github.com/Malabarba/lazy-map-closure>>

U3L6TFEJF : I haven't tried it myself but it seems to do what you want

U3L6TFEJF : otherwise, you'll have to implement something like that yourself with `delay` and `force`

U46LFMYTD : Hi <@U3L6TFEJF>

U46LFMYTD : I have come up with this, i think it is inelegant

```
(defn make-lazy-tree [{:in :in out :out left :left right :right :as tree} indices]
  (cond
    (empty? indices) tree
    :else (assoc
      tree
      :left (fn [] (make-lazy-tree {:in (conj in (first indices)) :out out} (rest indices)))
      :right (fn [] (make-lazy-tree {:in in :out (conj out (first indices))} (rest indices))))))
...

```

U46LFMYTD : so instead of returning the next subtree it returns a function to the next subtree, which can be called like this

U46LFMYTD : `` (defn my-left [t] ((:left t)))

(defn my-right [t] ((:right t)))

```
(-> foo
  my-right
  my-right
  my-left)
...

```

U46LFMYTD : giving:``

```
=> {:in #{2},
     :out #{0 1},
     :left #function[bab.core/make-lazy-tree/fn--20560],
     :right #function[bab.core/make-lazy-tree/fn--20562]}
...

```

U46LFMYTD : I guess delay is kind of doing the same thing?

U46LFMYTD : just tried it, delay works nicely too

U051SS2EU : you can't tail call that code though

U051SS2EU : not without a radical design shift to move stack data into the heap

U46LFMYTD : agreed. As per <@U3L6TFEJF> s suggestion i wrapped the calls with delay

```
(defn make-delay-tree [{:in :in out :out left :left right :right :as tree} indices]
  (cond
    (empty? indices) tree
    :else (assoc
      tree
      :left (delay (make-delay-tree {:in (conj in (first indices)) :out out} (rest indices))))
...

```

```
... :right (delay (make-delay-tree {:in in :out (conj out (first indices))} (rest indices))))))
```

```
U46LFMYTD : writing ```  
(defn my-left [t] @(:left t))  
(defn my-right [t] @(:right t))  
```
```

U46LFMYTD : I can traverse the tree to the leaf nodes that I want, without creating the others