ADT extension List

Use Element($\leq$)

Operations

isSorted : List $\rightarrow$ Boolean

merge : List $\times$ List $\rightarrow$ List

Precondition

merge(l_1, l_2) iff isSorted(l_1) and isSorted(l_2)

Axioms

A_1 isSorted(newr) $\equiv T$

A_2 isSorted(cons(e, newr)) $\equiv T$

A_2^* rest(l) $\equiv$ newr $\Rightarrow$ isSorted(l) $\equiv T$

A_3 isSorted(cons(e, l)) $\equiv$ e $\leq$ first(l) and isSorted(l)

A_3^* isSorted(l) $\equiv$ first(l) $\leq$ first(rest(l)) and isSorted(rest(l))

ex: isSorted((1 2 3))

A_3^* ~~1 $\leq$ 2~~ and isSorted((2 3))

isSorted((1 3 2))

A_3^* ~~1 $\leq$ 3 and~~ isSorted((3 2))

A_3^* ~~2 $\leq$ 3~~ and isSorted((3))

A_2^* T

A_3^* 3 $\leq$ 2 and ——

F

A_4 $merge(l, nil) \equiv l$

A_5 $merge(nil, l) \equiv l$

A_6^* $first(l_1) \leq first(l_2) \Rightarrow merge(l_1, l_2) \equiv cons(first(l_1), merge(rest(l_1), l_2))$

A_7^* $merge(l_1, l_2) \equiv cons(first(l_1), merge(l_1, rest(l_2)))$

$merge((1\ 3), (2\ 4))$

A_6^* $cons(1, merge((3), (2\ 4)))$

A_7^* $cons(1, cons(2, merge((3), (4))))$

A_6^* $cons(1, cons(2, cons(3, merge(nil, (4)))))$

A_5 $cons(1, cons(2, cons(3, (4))))$

$\equiv (1\ 2\ 3\ 4)$

ADT extension List

Operations

last : List → Element

allButLast : List → List

reverse : List → List

Preconditions

last(l) iff $l \not\equiv$ nenr

allButLast(l)

last((A B C)) ≡ C

allButLast((A B C)) ≡ (A B)

Axioms

A_1 last(cons(e, nenr)) ≡ e

A_1^* rest(l) ≡ nenr ⇒ last(l) ≡ first(l)

A_2 last(cons(e, l)) ≡ last(l)

A_2^* last(l) ≡ last(rest(l))

last((B C))

A_2^* last((C))

A_1^* C

A_3 $allButLast(cons(e, newr)) \equiv newr$

A_3^* $rest(l) \equiv newr \Rightarrow allButLast(l) \equiv newr$

A_4 $allButLast(cons(e, l)) \equiv cons(e, allButLast(l))$

A_4^* $allButLast(l) \equiv cons(first(l), allButLast(rest(l)))$

$allButLast((A\ B\ C))$

A_4^* $cons(A, allButLast((B\ C)))$

A_4^* $cons(A, cons(B, allButLast((C)))))$

A_3^* $cons(A, cons(B, newr)) \equiv (A\ B)$

A_5 reverse(nil) ≡ nil

A_6^* reverse(l) ≡ cons(last(l), reverse(allButLast(l)))

reverse((A B C))

A_6^* cons(C, reverse(allButLast(A B C)))

cons(C, reverse((A B)))

A_5^* cons(C, cons(B, reverse(allButLast((A B)))))

cons(C, cons(B, reverse((A))))

A_5^* cons(C, cons(B, cons(A, reverse(allButLast((A)))))))

cons(C, cons(B, cons(A, reverse(nil))))

A_5 cons(C, cons(B, cons(A, nil))) ≡ (C B A)

complexity of reverse?

unit op : cons

Param : N, length of list

last:

$$C(1) = 0$$
$$C(n) = C(n-1) = 0$$

reverse: $C(0) = 0$

$$C(n) = C(n-1) + 1 + 0$$
$$= C(n-1) + n$$
$$= (C(n-2) + n-1) + n$$
$$= (C(n-3) + n-2 + n-1 + n$$
$$\vdots$$
$$= C(n-i) + \sum_{j=n-i+1}^{n} j$$

allButLast:

$$C(1) = 0 \quad (A3)$$
$$C(n) = C(n-1) + 1 \quad (A4)$$
$$= (C(n-2)+1) + 1$$
$$= C(n-2) + 2$$
$$\vdots$$
$$= C(n-i) + i$$
$$= C(1) + n - 1 = \Theta(n)$$

$i = n-1$

$i = n$ $C(n) = C(0)$

$$+ \sum_{j=1}^{n} j$$
$$= \frac{n(n+1)}{2}$$
$$= \Theta(n^2)$$

indexOf((A B C), B) ⟶ 2

indexOf((A B C), D) ⟶ 0

ADT extension List

Operation

indexOf : List × Element ⟶ Natural

Axioms

A_1 indexOf(new, e) ≡ 0

A_2 indexOf(add(e, l), e) ≡ 1

A_2^* indexOf(l, first(l)) ≡ 1

A_3^* indexOf(l, e) ≡ 1 + indexOf(rest(l), e)

indexOf((A B C), B)

A_3^* 1 + indexOf((B C), B)

A_2^* 1 + 1 ≡ 2

indexOf((A B), C)

A_3^* 1 + indexOf((B), C)

A_3^* 2 + indexOf(new, C)

A_1 2 + 0 ≡ 2

$indexOf : List \times Element \rightarrow Natural$

$indexAux : List \times Element \times Natural \longrightarrow Natural$

A_1 $indexOf(l, e) \equiv indexAux(l, e, 1)$

A_2 $indexAux(nil, e, n) \equiv 0$

A_3^* $indexAux(l, first(l), n) \equiv n$

A_4^* $indexAux(l, e, n) \equiv indexAux(rest(l), e, n+1)$

$indexOf((A\ B\ C), B)$

A_1 $indexAux((A\ B\ C), B, 1)$

A_4^* $indexAux((B\ C), B, 2)$

A_3^* 2

$indexOf((A\ B), C)$

A_1 $indexAux((A\ B), C, 1)$

A_4^* $indexAux((B), C, 2)$

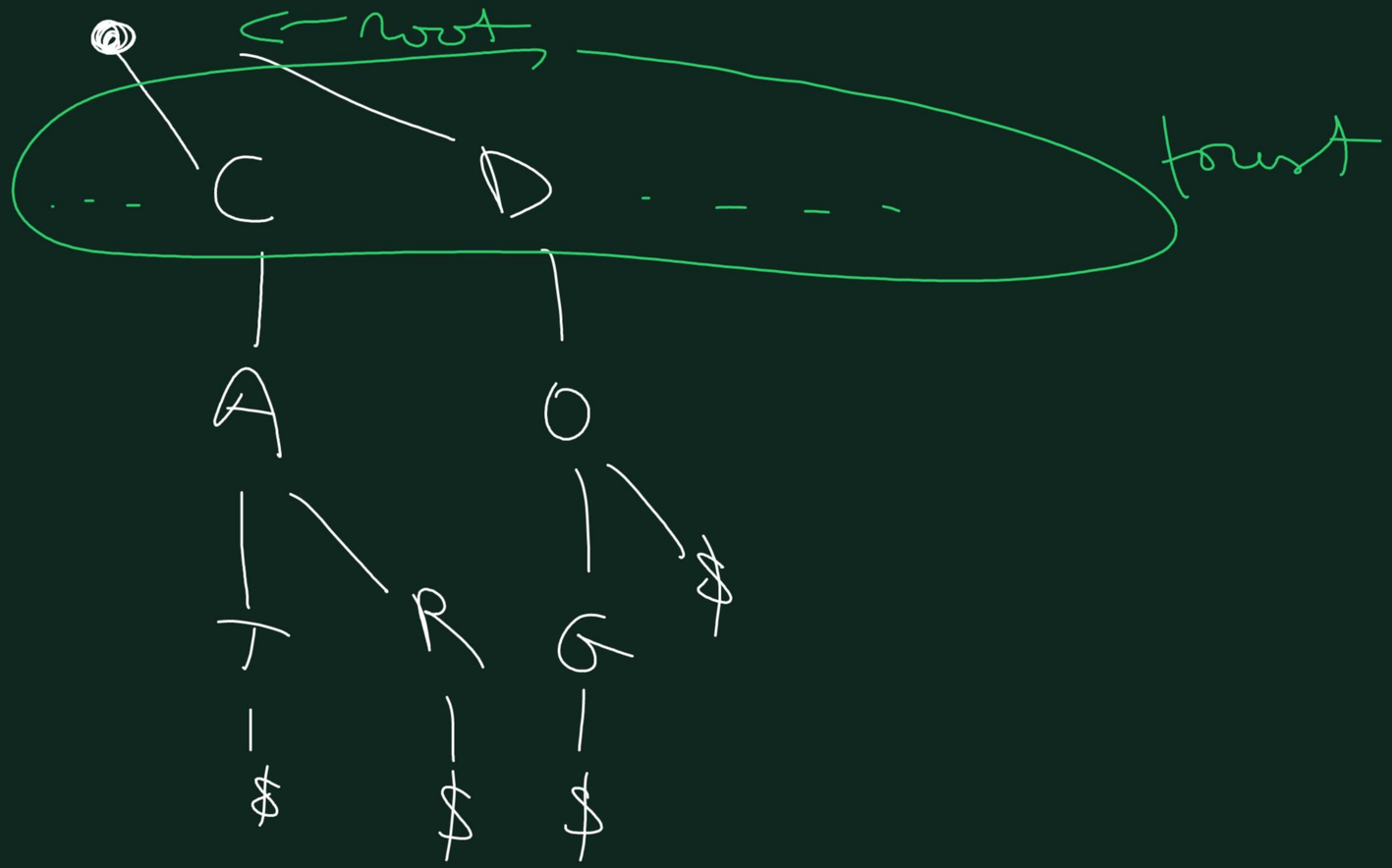
A_4^* $indexAux(nil, C, 3)$

A_2 0

param: N, size of dictionary (number of words)

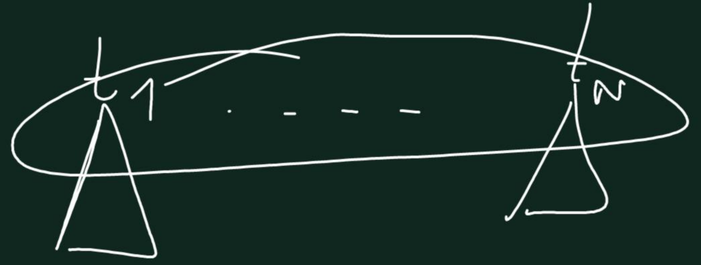
unit op: comparison between chars

$C(N)$ $\log N$ / $N \log N$ / N / 1



ADT Tree, Forest

Use Element, Integer, Boolean



Operations

* new : Element × Forest → Tree
* newf : → Forest
  root : Tree → Element
  trees : Tree → Forest
  nth : Forest × Integer → Tree
  nTrees : Forest → Integer
* addTree : Forest × Tree × Integer → Forest
  delTree : Forest × Integer → Forest

preconditions

nth(f, n) if 1 ≤ n ≤ nTrees(f)

delTree(f, n) if ———

addTree(f, t, n) if ——— + 1

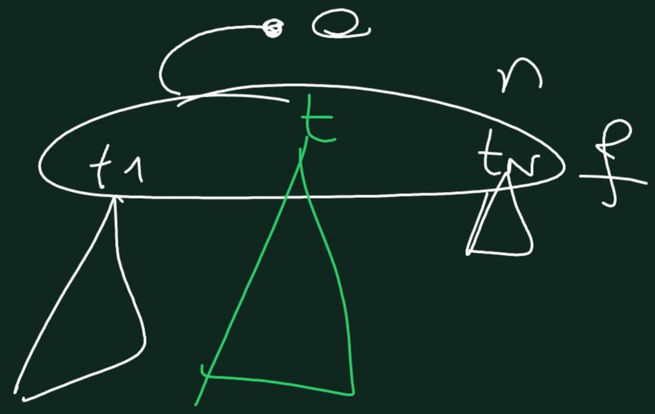
Axioms

A_1 $root(newf(e, f)) \equiv e$

A_2 $trees(newf(e, f)) \equiv f$

A_3 $\overline{nTrees}(newf) \equiv 0$

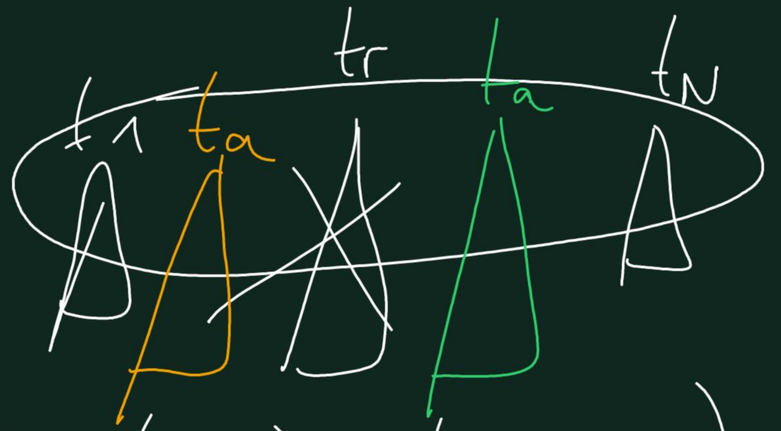
A_4 $\overline{nTrees}(\overline{addTree}(f, t, n)) \equiv \overline{nTrees}(f) + 1$



A_5 $a > l \Rightarrow nth(addTree(f, t, a), l) \equiv nth(f, l)$

A_6 $nth(addTree(f, t, n), n) \equiv t$

A_7 $nth(addTree(f, t, a), l) \equiv nth(f, l-1)$



A_8 $a > r$

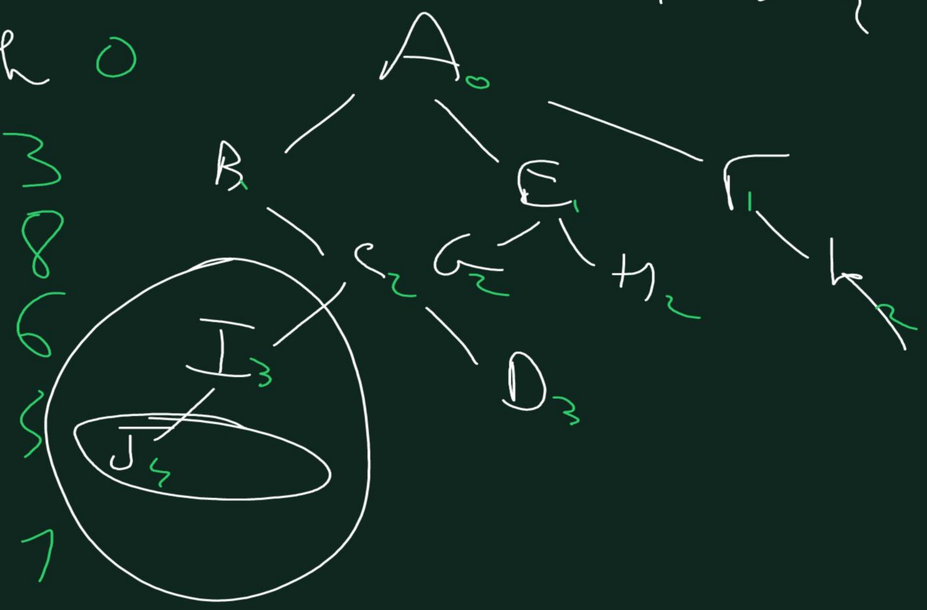
$delTree(addTree(f, t, a), r) \equiv addTree(delTree(f, r), t, a-1)$

A_9 $delTree(addTree(f, t, n), n) \equiv f$

A_{10} $delTree(addTree(f, t, a), r) \equiv addTree(delTree(f, r-1), t, a)$

measures on trees:

- size(t): number of elts 11
- length of a path from e_1 to e_2: number of links $e_1 \to e_2$
- height (depth) of a node: length of a path root → node 0
- height(tree) = $\max_{n \in t}(h(n))$ 4
- pathlength(t) = $\sum_{n \in t} h(n)$ 21



new(I, add(newf, new(J, newf), 1))

ADT extension Tree, Forest

Operations

size : Tree → Integer

sizeF : Forest → Integer

height : Tree → Integer

heightF : Forest → Integer

PL : Tree → Integer

PLF : forest → Integer

A_1 size(newt(e, f)) ≡ 1 + sizef(f)

A_2 sizef(newf) ≡ 0

A_3 sizef(addTree(f, t, n)) ≡ size(t) + sizef(f)

https://tiny.one/algo36