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 (newS) $\equiv T$

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

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

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

l = (1 3 2)

isSorted ((1 3 2))

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

A_3^* ~~$3 \leq 2$~~ and ——

F

F

isSorted ((1 2 3))

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

T

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

A_2 T

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

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

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

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

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

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

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

A_4 $cons(1, cons(2, cons(3, (4)))) \equiv cons(1, cons(2, (3\ 4)))$

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

reversing a list: (A B C) → (C B A)

define an extension to ADT List with the 3 operations:

* last(l) last(A B C D) ≡ D
* allButLast(l) allButLast((A B C D)) ≡ (A B C)
* reverse (l)

ADT extension List

Operations

last: List → Element

allButLast: List → List

reverse: List → List

Preconditions

last(l) iff allButLast(l) ——— l !≡ new (or !isEmpty(l))

A_1 last(cons(e, newr)) ≡ e

A_2 last(cons(i, l)) ≡ last(l)

A_2^* last(l) ≡ last(rest(l))

last((A B C))

A_2^* last((B C))

A_2^* last((C))

A_1^* C

A_3 allButLast(cons(e, newr)) ≡ newr

A_4 allButLast(cons(e, l)) ≡ cons(e, allButLast(l))

A_5^* allButLast(l) ≡ 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)) ≡ cons(A, (B)) ≡ (A B)

A_5 reverse(newr) ≡ newr

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_6 ≡ cons(C, cons(B, reverse((A))))

A_6 ≡ cons(C, cons(B, cons(A, reverse(newr))))

A_5 ≡ cons(C, cons(B, cons(A, newr)))

≡ (C B A)

(A B C)

(C B A)

D

Complexity of reverse?
unit op : cons
param : N, size of list

⇒ Complexity of last and allButLast?

last:

$C(1) = 0$

$C(N) = C(N-1)$

$= 0$

allButLast:

$C(1) = 0$

$C(N) = C(N-1) + 1$

$= ((N-2)+1) + 1$

$= C(N-2) + 2$

$\vdots$

$= C(N-i) + i$

$i = N-1 = \cancel{C(1)} + N - 1$

$= N - 1$

$= \Theta(N)$

reverse:

$C(0) = 0$

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

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

$= ((n-2) + n - 2 + n - 1$

$= ((n-3) + n - 3 + n - 2 + n - 1$

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

$= \cancel{(0)} + \sum_{j=0}^{n-1} j = \frac{n(n-1)}{2}$

$= \Theta(n^2)$

indexOf: indexOf(B, (A B C)) ≡ 2
indexOf(D, (A B C)) ≡ 0

ADT extension List

Operations

indexOf : Element × List ⟶ Natural

Axioms

A_1 indexOf(e, new) ≡ 0

A_2^* indexOf(first(l), l) ≡ 1

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

indexOf(B, (A B C))

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

A_2 1 + 1 ≡ 2

---

indexOf(B, (A C))

A_3 indexOf(B, (C)) + 1

A_3 indexOf(B, new) + 2

A_1 0 + 2 ≡ 2

Operations:

indexOf : Element × List → Natural

indexAux : Element × List × Natural → Natural

A_1 indexOf(e, l) ≡ indexAux(e, l, 1)

A_2 indexAux(e, new, n) ≡ 0

A_3 indexAux(first(l), l, n) ≡ n

A_4 indexAux(e, l, n) ≡ indexAux(e, rest(l), n+1)

indexOf(B, (A B C))

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

A_4 indexAux(B, (B C), 2)

A_3 2

---

indexOf(B, (A C))

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

A_4 indexAux(B, (C), 2)

A_4 indexAux(B, new, 3)

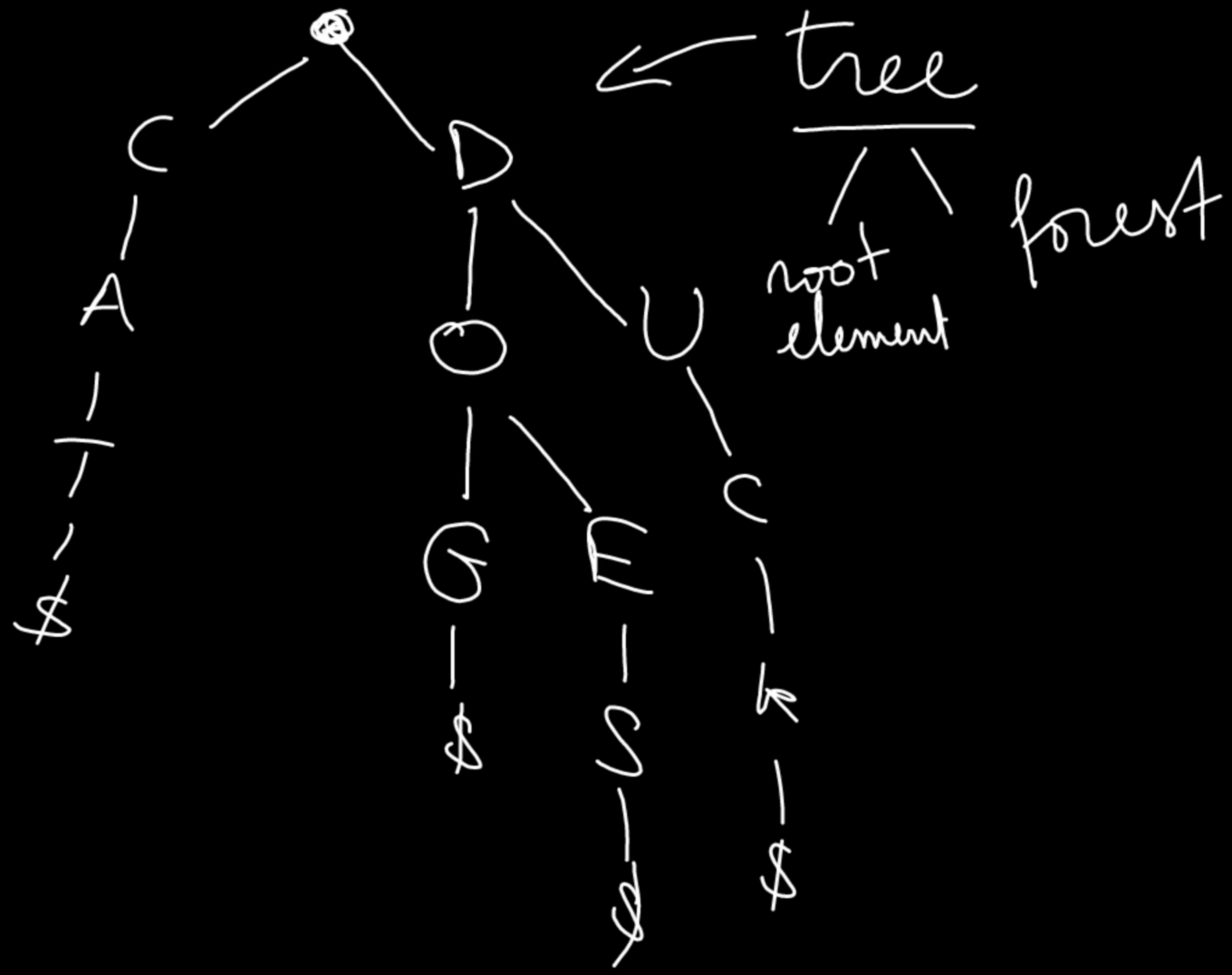
A_2 0

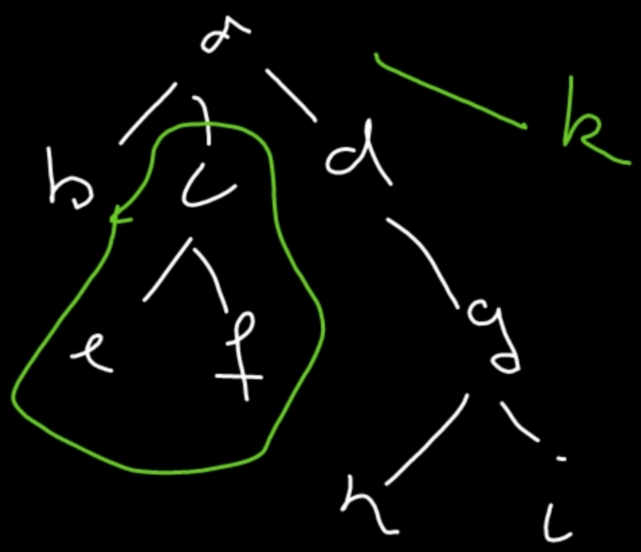
param: N, number of words

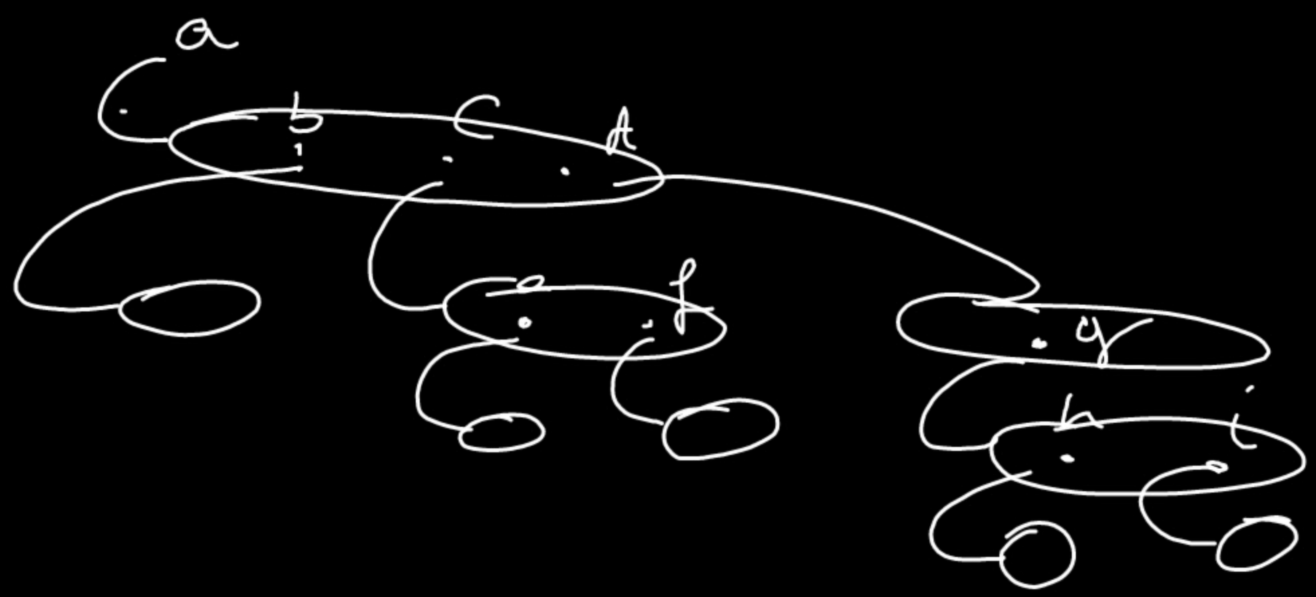
unit op: char comparison

C(check for word) ?

((N) = (4)(1))




a
b
c
d
e
f
g
h
i
k


a
b
c
d
e
f
g
h
i

ADT Tree, Forest

Use Element, Integer, Boolean

Operations

* newT : 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 $1 \leq n \leq nTrees(f)$

addTree(f, t, n) if $1 \leq n \leq nTrees(f)+1$

delTree(f, n) if $1 \leq n \leq nTrees(f)$

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

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

A_3 $nTrees(newF) \equiv 0$

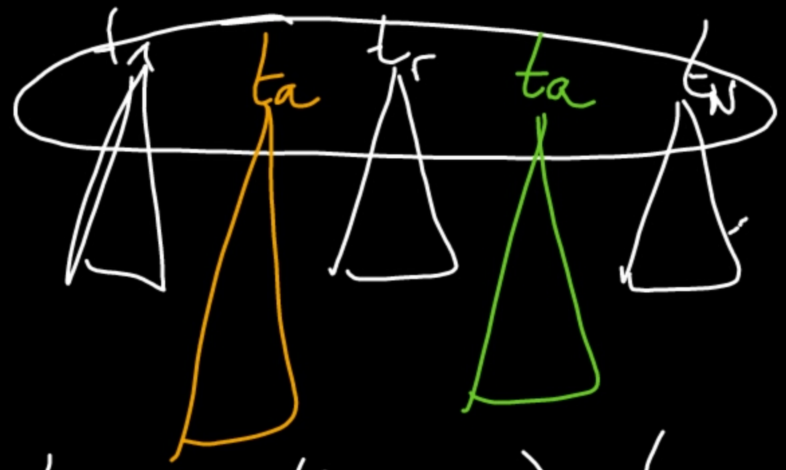
A_4 $nTrees(addTree(f,t,n)) \equiv nTrees(f) + 1$

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

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

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

A_8 $a > r \Rightarrow delTree(addTree(f, t, a), r)$
$\equiv addTree(delTree(f, r), t, a-1)$



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

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

https://tiny.one/algo34