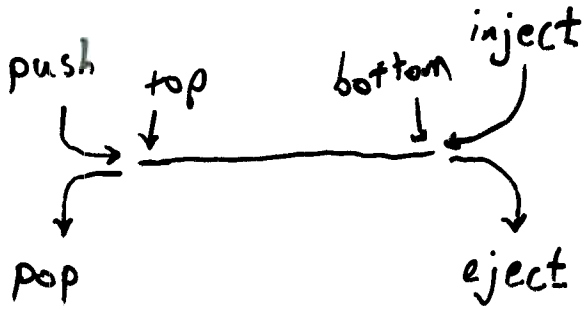
Lists



top, push, pop = stack

top, inject, pop = queue

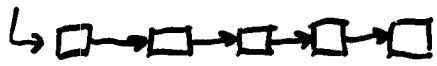
top, push, inject, pop = stack-ended queue = steque

all six = double-ended queue = deque

catenation

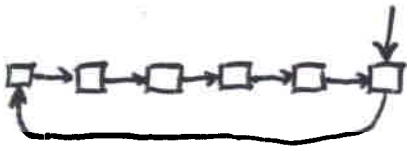
List Representations with O(1)-time operations

Singly-linked gives stacks



Singly-linked circular (or ptrs to front, rear)

gives steques with catenation



Doubly-linked circular (or ptrs to front, rear)

gives deques with catenation

Array with wraparound gives deque (no catenation)

Persistence

Preserve old versions as well as new

(non-destructive updates)

Just for access: partial persistence

For access + update: full persistence

For access + update + combining: confluent persistence

Problem: obtain confluently persistent

catenable deques

with $O(1)$-time operations

General result for making linked structures

persistent (Driscoll, Sarnak, Sleator, Tarjan '89)

gives fully persistent deques

but

O(1) time bound is amortized

no catenation

(Driscoll et al. result has been made

worst-case)

(Pure) LISP

Atoms, lists

cons (x, y) yields [x, y]

car ([x, y]) yields x

cdr ([x, y]) yields y

(Impure)

replaca, replacd

What data structures can be built in pure LISP? (or your favorite functional programming language)

Pure LISP gives (confluent) persistence

automatically (no rewriting)

Can build stacks, trees

Queues?

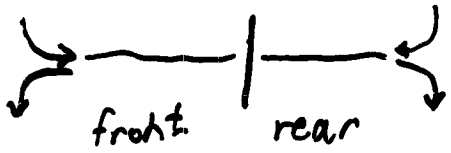
Queues with catenation?
(= steques with catenation)

Deques with catenation?

(LISP cat: reverse front list, copy onto

rear list - time = length of front list)

Deque as two stacks:



When one part is empty, create new front, rear,

each with half

$O(1)$ amortized time

Incrementally recopying gives $O(1)$

worst-case time

Catenation??

Note: self-catenation allows building

of exponentially-sized structures

$x_1 = [a]$

$x_2 = cat(x_1, x_1)$

$x_3 = cat(x_2, x_2)$

.
.
.

$x_k = cat(x_{k-1}, x_{k-1})$

x_k is a list of 2^k a's

History

Kosaraju: real-time simulation of a fixed number of catenable deques by non-catenable deques (but no self-catenation) 1979

real-time simulation of a variable number of catenable min-deques on a RAM 1994 (not confluently persistent)

Balanced trees: $O(\log n)$

Driscoll et al. $O(\log \log k)$ for k^{th} op. 1994

Buchsbaum & Tarjan $O(\log^* k)$ 1995

→ Kaplan & Tarjan $O(1)$ 1999

Amortized, with memoization

Okasaki 1995

Kaplan, Okasaki, Tarjan 2000

Our goal: simplify!

Redundant Binary Numbers

Goal: add or subtract 1 without carries

Solution:

Allow 2 as a digit (as well as 0, 1)

Require regularity: 0's and 2's alternate, ignoring 1's.

2110111201	regular
102112011112	not

0-exposed: rightmost non-1 is a 0

2-exposed: rightmost non-1 is a 2

0-fix: eliminate 0-exposure

2-fix: eliminate 2-exposure

0-fix: rightmost 0: $10 \to 02$
$20 \to 12$

2-fix: rightmost 2: $02 \to 10$
$12 \to 20$

Add 1: if 2-exposed, 2-fix, then add 1 (no carry)

Subtract 1: if 0-exposed, 0-fix, then subtract 1 (no carry)

1121011 21	+1
1121012 01	2-fix
1121012 02	add 1

Data Structure:

Stack of stacks of digits,

each stack a maximal block of

all 1's except rightmost digit

1112|110|12|0|11112|1111

Each block, except possibly the rightmost,

ends in a 0 or 2

Fix must access the right end of the rightmost

or second-rightmost block

Deques (without catenation)

Recursive representation:

empty ϕ or singleton $[x]$

or $[p, c, s]$

where p and s are buffers of

0, 1, or 2 elements

and c is a child deque of

pairs of elements

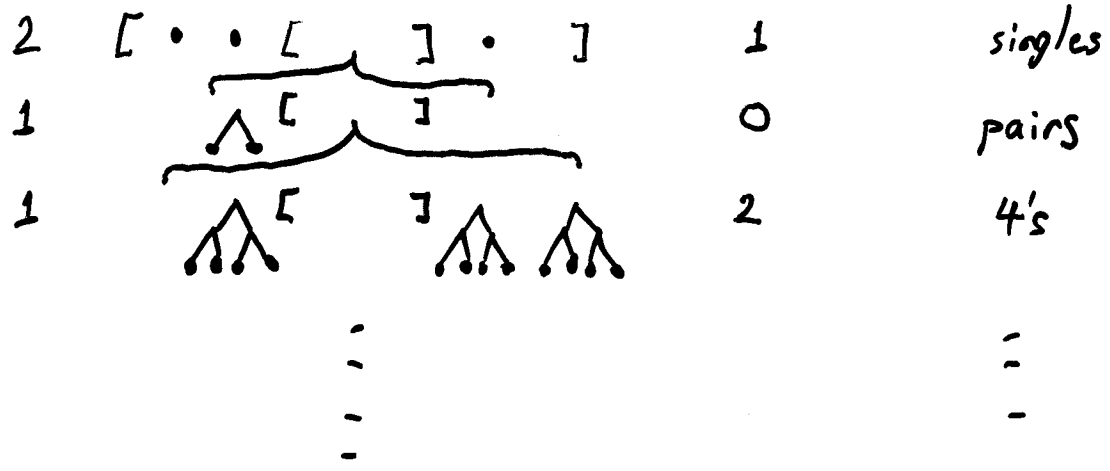
Regularity: ignoring the bottom deque (empty or singleton)

prefix sizes form a redundant binary number,
as do suffix sizes

(topmost size = rightmost digit)

Forbidden configurations: $[0,0,1]$, $[1,0,0]$,
$[0,0,2]$, $[2,0,0]$,
$[0,1,0]$

($[x]$ is the unique 1-element deque, $[1,0,1]$ the unique
2-element deque)



push, inject like +1

pop, eject like -1

need 0-fix, 2-fix on both sides

must deal with two sides, interaction

at bottom deque

Prefix operations
(suffix side symmetric)

n-push (x, d) =
  if $d = \emptyset$ then $[x]$
  else if $d = [y]$ then $[[x], \emptyset, [y]]$
  else if $d = [\emptyset, c, s]$ then $[[x], c, s]$
  else if $d = [[y], c, s]$ then $[[x, y], c, s]$

2-fix (d):
  let $d' = [p', c', s']$ be the topmost descendant deque in d with $|p'| = 2$. Replace d' in d by $[\emptyset, n\text{-push}(p', c'), s']$

push (x, d): if d is 2-exposed then
    n-push $(x,$ 2-fix $(d))$
  else n-push (x, d)

n-pop(d):
  if d = ∅ then error
  else if d = [x] then ∅
  else if d = [[x], ∅, [y]] then [y]
  else if d = [[x], [[y,z]], ∅] then [[y], ∅, [z]]
  else if d = [[x], c, s] then [∅, c, s]
  else if d = [[x,y], c, s] then [[y], c, s]

d = or[[x], ∅, [y,z]]

0-fix(d):
  let d' = [∅, c', s'] be the topmost descendant
  deque in d with 0-prefix. Replace d' in d
  by [top(c'), n-pop(c'), s']

pop(d):
  if d is 0-exposed then n-pop(0-fix(d))
  else n-pop(d)

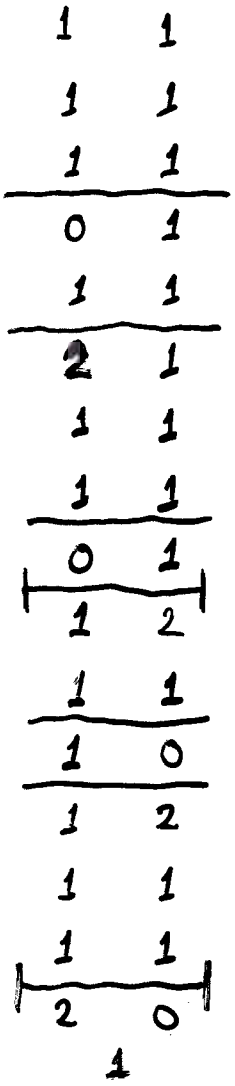
Need to access topmost non-1 prefix and suffix

Data structure:

stack of stacks of stacks (S^3)

inner stack: maximum block with only topmost not [1, -, 1] (or [1])

stack of stacks: maximum block of inner stacks without both a 0 or 2-prefix and a 0 or 2-suffix (in the same or different stacks) except on top


1 1
1 1
1 1
0 1
1 1
2 1
1 1
1 1
0 1
1 2
1 1
1 0
1 2
1 1
1 1
2 0
1

Yet another deque representation

but - $O(\log k)$-time access

to k^{th} from either end

Like finger search trees

No catenation yet!

Idea: convert catenate to push/inject(s)

Child deque elements more complicated:
contain stored deques

pop calls cat calls inject

no circularity!

Steqnes with Catenation

steqne = 0 to 3 elements
or [p, c, d]

with p, d of 1-4 elements
c a steqne of entries

entry = pair or triple or
[pair or triple, steqne]

[··· [[··], [···], [[··], s], [[···], t]] ·]

2-buffers are "mid", 1-buffers "low",
3 and 4-buffers "high"

Need low-fix and high-fix

Catenation:

$[P_1, c_1, s_1]$ & $[P_2, c_2, d_2]$

Combine s_1 and s_2. Inject pairs/triples from s_1 & s_2 into c_1. Last inject is of $[P, c_2]$.

Result is $[P_1, c', d_2]$, where c' is formed from c_1 by the injections.

Low-fix (new case)

$[[x], c, s]$:

if top(c) = [b, d]:

result is $[[x] \& b], d \& c', s]$

where $c' = pop(c)$

Details, details...

Deques with Catenation

deque = 0-7 elements
or [p, c, s]
or [p, c_1, m, c_2, s]

where p, s have 3-6 elements
m has 2 (or 3?) elements

c, c_1, c_2 are child deques of entries

entry = pair or triple or
[pair or triple, deque, pair or triple]

(deques in entries are non-empty)

Note: the skeleton of the structure is now

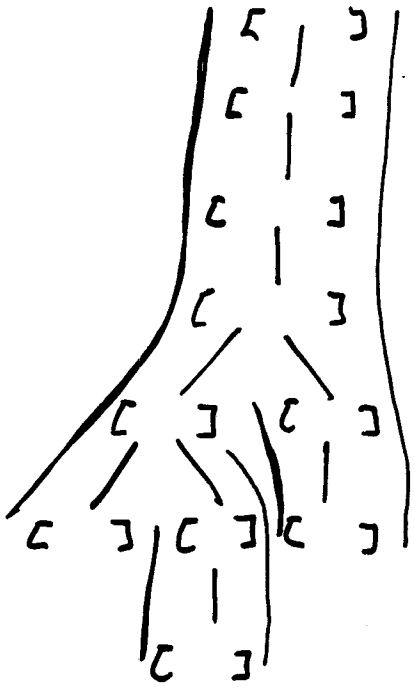
not unary (linear) but binary (tree)

pop/eject call cat calls push/inject

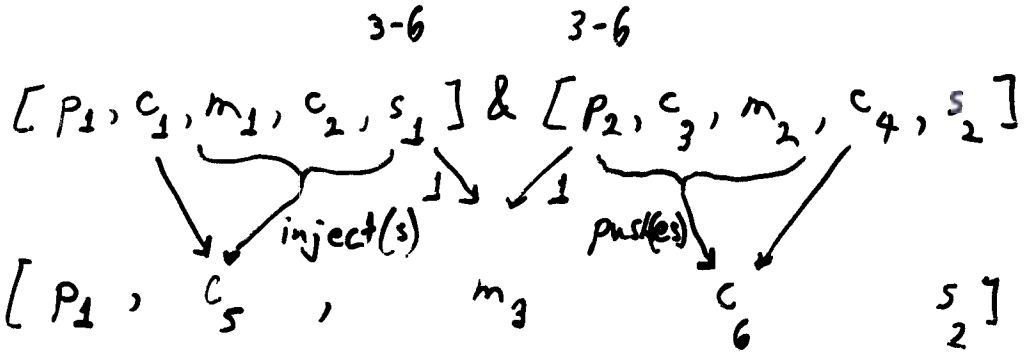
3 - buffers are "low"

5, 6 - buffers are "high"

Regularity along left paths and right paths in the tree



Catenation



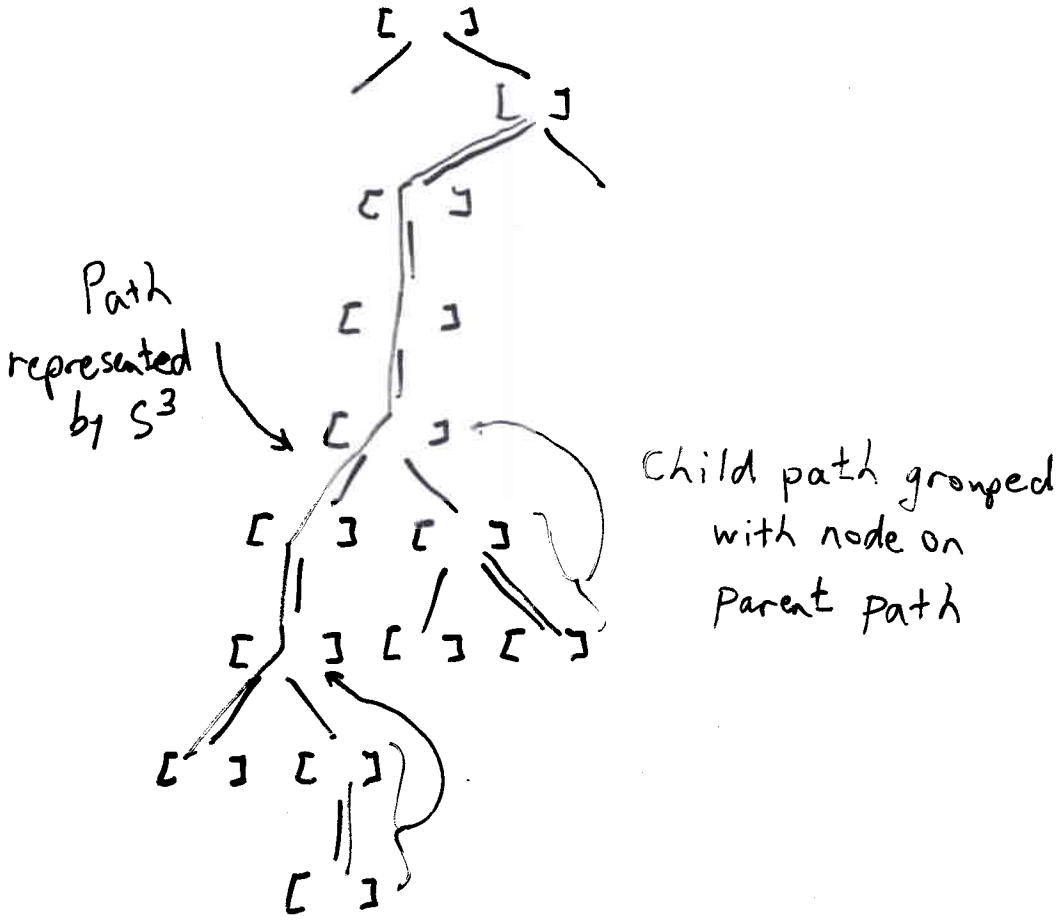
Low-fix like steqnes

Data Structure:

Split into paths at zig-zags
(left child of right child
or vice versa)

Represent paths using S^3

Each node is grouped with its
(unique) child path


Path represented by S^3
Child path grouped with node on parent path

Benefits

Smaller buffers: as small or smaller than for amortized structure

Previous deque structure (worst-case)

doesn't even have constant size buffers

(uses noncatenable deques as buffers)

and has a more complicated regularity condition.

New S^3 data structure

Deques with catenation via linear skeleton?

I don't think so.

Finger search trees with catenation in LISP?

Best to date is $O(\log \log n)$