Package 'dequer'

November 16, 2017

Type Package

Title Stacks, Queues, and 'Deques' for R

Version 2.0-1

Description Queues, stacks, and 'deques' are list-like, abstract data types. These are meant to be very cheap to ``grow'', or insert new objects into. A typical use case involves storing data in a list in a streaming fashion, when you do not necessarily know how may elements need to be stored. Unlike R's lists, the new data structures provided here are not necessarily stored contiguously, making insertions and deletions at the front/end of the structure much faster. The underlying implementation is new and uses a head/tail doubly linked list; thus, we do not rely on R's environments or hashing. To avoid unnecessary data copying, most operations on these data structures are performed via side-effects.

License BSD 2-clause License + file LICENSE

Depends R (>= 3.1.0)

NeedsCompilation yes

ByteCompile yes

Author Drew Schmidt [aut, cre]

URL https://github.com/wrathematics/dequer

BugReports https://github.com/wrathematics/dequer/issues

Maintainer Drew Schmidt <wrathematics@gmail.com>

RoxygenNote 5.0.1

Repository CRAN

Date/Publication 2017-11-16 21:15:20 UTC

R topics documented:

`dequer-package` *dequer*

Description

Queues, stacks, and 'deques' are list-like, abstract data types. These are meant to be very cheap to "grow", or insert new objects into. A typical use case involves storing data in a list in a streaming fashion, when you do not necessarily know how may elements need to be stored. Unlike R's lists, the new data structures provided here are not necessarily stored contiguously, making insertions and deletions at the front/end of the structure much faster. The underlying implementation is new and uses a head/tail doubly linked list; thus, we do not rely on R's environments or hashing. To avoid unnecessary data copying, most operations on these data structures are performed via side-effects.

Author(s)

Drew Schmidt `<wrathematics AT gmail.com>`

`as.deque` *Convert to Deque*

Description

Convert to Deque

Usage

```
as.deque(x)

## S3 method for class 'list'
as.deque(x)

## Default S3 method:
```

```
as.deque(x)

## S3 method for class 'queue'
as.deque(x)

## S3 method for class 'stack'
as.deque(x)
```

Arguments

x	An object either to be converted to the first element of a deque (default), or the elements of a list (or columns of a dataframe) to be set as elements of a deque.

Value

A queue, stack, or deque.

Examples

```
## Not run:
library(dequer)
d <- as.deque(lapply(1:5, identity))
d

## End(Not run)
```

`as.queue` *Convert to Queue*

Description

Convert to Queue

Usage

```
as.queue(x)

## S3 method for class 'list'
as.queue(x)

## Default S3 method:
as.queue(x)

## S3 method for class 'deque'
as.queue(x)

## S3 method for class 'stack'
as.queue(x)
```

Arguments

x	An object either to be converted to the first element of a queue (default), or the elements of a list (or columns of a dataframe) to be set as elements of a queue.

Value

A queue object.

Examples

```
## Not run:
library(dequer)
q <- as.queue(lapply(1:5, identity))
q

## End(Not run)
```

---

as.stack *Convert to Stack*

Description

Convert to Stack

Usage

```
as.stack(x)

## S3 method for class 'list'
as.stack(x)

## Default S3 method:
as.stack(x)

## S3 method for class 'deque'
as.stack(x)

## S3 method for class 'queue'
as.stack(x)
```

Arguments

x	An object either to be converted to the first element of a stack (default), or the elements of a list (or columns of a dataframe) to be set as elements of a stack.

Value

A stack object.

Examples

```
## Not run:
library(dequer)
s <- as.stack(lapply(1:5, identity))
s

## End(Not run)
```

`combine` *combine*

Description

Combine two objects (queue/stack/deque) into one of the same type.

Usage

```
combine(x1, x2)
```

Arguments

`x1, x2`	Two different deques, stacks, or queues. Arguments must be of the same type.

Details

Operates via side-effects; see examples for clarification on usage.

Value

Returns `NULL`. After combining, object `x2` is a 0-length (empty) object.

Examples

```
## Not run:
library(dequer)
s1 <- stack()
for (i in 1:5) push(s1, i)
s2 <- stack()
for (i in 10:8) push(s2, i)

combine(s1, s2)
s1 # now holds all 8 elements
s2 # holds 0 elements

## End(Not run)
```

deque *deque*

Description

A constructor for a deque.

Usage

```
deque()
```

Details

A deque is a double-ended queue. Insertion and deletion of objects can happen at either end.

The implementation is a head/tail doubly linked list.

Examples

```
## Not run:
library(dequer)
d <- deque()
d

## End(Not run)
```

peeking *peek/peekback*

Description

These methods are side-effect free. Note that unlike R's `head()` and `tail()`, the sub-objects are not actually created. They are merely printed to the terminal.

Usage

```
peek(x, n = 1L)

## S3 method for class 'deque'
peek(x, n = 1L)

## S3 method for class 'queue'
peek(x, n = 1L)

## S3 method for class 'stack'
peek(x, n = 1L)
```

```
peekback(x, n = 1L)

## S3 method for class 'deque'
peekback(x, n = 1L)

## S3 method for class 'queue'
peekback(x, n = 1L)

## S3 method for class 'stack'
peekback(x, n = 1L)
```

Arguments

x	A queue, stack, or deque.
n	The number of items to view.

Details

View items from the front (peek()) or back (peekback()) of a queue, stack, or deque.

Value

Returns NULL; sub-elements are only printed.

Examples

```
## Not run:
library(dequer)
s <- stack()
for (i in 1:3) push(s, i)

peek(s)
peekback(s)
peek(s, length(s))

## End(Not run)
```

---

popping *pop/popback*

Description

Remove items from the front of a stack, queue, or deque for pop(); or, remove items from the back of a deque for popback().

Usage

```
pop(x)

## S3 method for class 'deque'
pop(x)

## S3 method for class 'queue'
pop(x)

## S3 method for class 'stack'
pop(x)

popback(x)

## S3 method for class 'deque'
popback(x)
```

Arguments

x	A queue, stack, or deque.

Details

Operates via side-effects; see examples for clarification on usage.

Value

Returns NULL; deletion operates via side-effects.

Examples

```
## Not run:
library(dequer)

### A simple queue example
q <- queue()
for (i in 1:3) pushback(q, i)

pop(q)
str(q)

### A simple stack example
s <- stack()
for (i in 1:3) push(s, i)
pop(s)
str(s)

## End(Not run)
```

printer *Printing Deques, Stacks, and Queues*

Description

Printing Deques, Stacks, and Queues

Usage

```
## S3 method for class 'deque'
print(x, ..., output = "summary")

## S3 method for class 'stack'
print(x, ..., output = "summary")

## S3 method for class 'queue'
print(x, ..., output = "summary")
```

Arguments

x	A queue, stack, or deque.
...	Unused.
output	A character string; determines what exactly is printed. Options are "summary", "truncated", and "full".

Details

If `output=="summary"`, then just a simple representation is printed.

If `output=="truncated"`, then the first 5 items will be printed.

If `output=="full"` then the full data structure will be printed.

pushing *push/pushback*

Description

Add items to the front of a stack or deque via `pop()`. Add items to the back of a queue or deque via `popback()`.

Usage

```
push(x, data)

## S3 method for class 'deque'
push(x, data)

## S3 method for class 'stack'
push(x, data)

pushback(x, data)

## S3 method for class 'deque'
pushback(x, data)

## S3 method for class 'queue'
pushback(x, data)
```

Arguments

x	A queue, stack, or deque.
data	R object to insert at the front of the deque/stack.

Details

Operates via side-effects; see examples for clarification on usage.

Value

Returns NULL; insertion operates via side-effects.

Examples

```
## Not run:
library(dequer)

### A simple queue example
q <- queue()
for (i in 1:3) pushback(q, i)

str(q)

### A simple stack example
s <- stack()
for (i in 1:3) push(s, i)

str(s)

## End(Not run)
```

queue *queue*

Description

A queue is a "first in, first out" abstract data type. Like a checkout queue (line) at a store, the first item in the queue is the first one out. New items are added to the end of the queue via `pushback()`. Items are removed from the queue at the front via `pop()`.

The implementation is a head/tail doubly linked list.

Usage

```
queue()
```

Details

A constructor for a queue.

Examples

```
## Not run:
library(dequer)
q <- queue()
q

## End(Not run)
```

revver *rev*

Description

rev

Usage

```
## S3 method for class 'deque'
rev(x)

## S3 method for class 'stack'
rev(x)

## S3 method for class 'queue'
rev(x)
```

Arguments

x	A queue, stack, or deque.

Details

Operates via side-effects; see examples for clarification on usage.

Value

Returns NULL; insertion operates via side-effects.

Examples

```
## Not run:
library(dequer)
s <- stack()
for (i in 1:5) push(d, i)

str(s)
rev(s)
str(s)

## End(Not run)
```

---

sep *sep*

Description

Spliot one object (queue/stack/deque) into two of the same type. NOTE: this function operates via side-effects AND has a return.

Usage

```
sep(x, k)
```

Arguments

x	A deque.
k	Index to split the deque at.

Details

Operates via side-effects ALTHOUGH THERE IS A NON-NULL RETURN; see examples for clarification on usage.

The split occurs after index k. So if the input x has, say, elements 1 to n, then after running `sep(x, k)`, x will have elements 1 to k, and the return will have values k+1, k+2, ..., n.

Value

A deque, stack, or queue (depending on the input)

Examples

```
## Not run:
library(dequer)
s <- stack()
for (i in 1:5) push(s, i)

### Split s into 2 stacks holding: (s) the first 3, and (s_last_2) last 2 elements
s_last_2 <- sep(s, 3)

str(s)
str(s_last_5)

## End(Not run)
```

`stack` *stack*

Description

A stack is a "last in, first out" (LIFO) abstract data type. New items are added to the front of the stack via `push()`. Items are removed from the stack at the front via `pop()`.

The implementation is a head/tail doubly linked list.

Usage

```
stack()
```

Details

A constructor for a stack.

Examples

```
## Not run:
library(dequer)
s <- stack()
s

## End(Not run)
```

Index