Demo of the bit package

Dr. Jens Oehlschlägel

2022-11-13

Contents

---

bit type

Create a huge boolean vector (no NAs allowed)

```
n <- 1e8
b1 <- bit(n)
b1
#> bit length=100000000 occupying only 3125000 int32
#>         1         2         3         4         5         6         7         8
#>     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#>            99999993  99999994  99999995  99999996  99999997  99999998  99999999
#>        ..     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#> 100000000
#>     FALSE
```

It costs only one bit per element

```
object.size(b1)/n
#> 0.1 bytes
```

A couple of standard methods work

```
b1[10:30] <- TRUE
summary(b1)
#>    FALSE     TRUE     Min.     Max.
#> 99999979       21       10       30
```

Create a another boolean vector with TRUE in some different positions

```
b2 <- bit(n)
b2[20:40] <- TRUE
b2
#> bit length=100000000 occupying only 3125000 int32
#>         1         2         3         4         5         6         7         8
#>     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#>            99999993  99999994  99999995  99999996  99999997  99999998  99999999
#>        ..     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
```

```
#> 100000000
#>     FALSE
```

fast boolean operations

```
b1 & b2
#> bit length=100000000 occupying only 3125000 int32
#>         1         2         3         4         5         6         7         8
#>     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#>            99999993  99999994  99999995  99999996  99999997  99999998  99999999
#>        ..     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#> 100000000
#>     FALSE
```

fast boolean operations

```
summary(b1 & b2)
#>    FALSE     TRUE      Min.      Max.
#> 99999989       11       20       30
```

bitwhich type

Since we have a very skewed distribution we may coerce to an even sparser representation

```
w1 <- as.bitwhich(b1)
w2 <- as.bitwhich(b2)
object.size(w1)/n
#> 0 bytes
```

and everything

```
w1 & w2
#> bitwhich: 11/100000000 occupying only 11 int32 in 1 representation
#>         1         2         3         4         5         6         7         8
#>     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#>            99999993  99999994  99999995  99999996  99999997  99999998  99999999
#>        ..     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
#> 100000000
#>     FALSE
```

works as expected

```
summary(w1 & w2)
#>    FALSE     TRUE      Min.      Max.
#> 99999989       11       20       30
```

even mixing

```
summary(b1 & w2)
#>    FALSE     TRUE      Min.      Max.
#> 99999989       11       20       30
```

processing chunks

Many bit functions support a range restriction,

```
summary(b1, range=c(1,1000))
#> FALSE  TRUE  Min.  Max.
#>   979    21    10    30
```

which is useful

```
as.which(b1, range=c(1, 1000))
#>  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
```

for filtered chunked looping

```
lapply(chunk(from=1, to=n, length=10), function(i)as.which(b1, range=i))
#> $`1:10000000`
#>  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`10000001:20000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`20000001:30000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`30000001:40000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`40000001:50000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`50000001:60000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`60000001:70000000`
```

```
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`70000001:80000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`80000001:90000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
#>
#> $`90000001:100000000`
#> integer(0)
#> attr(,"maxindex")
#> [1] 100000000
#> attr(,"class")
#> [1] "booltype" "which"
```

over large ff vectors

```
options(ffbatchbytes=1024^3)
x <- ff(vmode="single", length=n)
x[1:1000] <- runif(1000)
lapply(chunk(x, length.out = 10), function(i)sum(x[as.hi(b1, range=i)]))
#> $`1:10000000`
#> [1] 9.47269
#>
#> $`10000001:20000000`
#> [1] 0
#>
#> $`20000001:30000000`
#> [1] 0
#>
#> $`30000001:40000000`
#> [1] 0
#>
#> $`40000001:50000000`
#> [1] 0
#>
#> $`50000001:60000000`
#> [1] 0
#>
#> $`60000001:70000000`
#> [1] 0
#>
```

```
#> $`70000001:80000000`
#> [1] 0
#>
#> $`80000001:90000000`
#> [1] 0
#>
#> $`90000001:100000000`
#> [1] 0
```

and wrap-up

```
delete(x)
#> [1] TRUE
rm(x, b1, b2, w1, w2, n)
```

for more info check the usage vignette