

## Ruby - Feature #18809

## Add Numeric#ceildiv

05/27/2022 10:23 PM - kyanagi (Kouhei Yanagita)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

### Description

pull request: <https://github.com/ruby/ruby/pull/5965>

I have needed to implement "rounding up division" several times.

("rounding up division" means getting a quotient of division which is rounded up to the nearest integer.)

Typically, this is implemented as follows:

```
# notice that b > 0 is assumed
def rounding_up_division(a, b)
  (a + b - 1) / b
end
```

But for me, this is difficult to write without careful consideration.

Every time I implement this, I need to think for a few minutes on paper.

So I propose to add a new method `Numeric#ceildiv`.

Typical examples where this is necessary are counting groups and pagination.

e.g. There are 123 items. If you display 10 items on each page, how many pages are there?

```
123.ceildiv(10) # => 13
```

We can find several examples of this division also in the Ruby's source code. (Try `grep -r -E -e '([^\ ]+)*\ *1\)\ */ *\1'`.)

```
./internal.h:#define roomof(x, y) (((x) + (y) - 1) / (y))
./array.c:    len = (len + ustep - 1) / ustep;
./include/ruby/internal/memory.h:    const size_t cnt = (total_size + sizeof(VALUE) - 1) / sizeof(VALUE);
./ext/bigdecimal/missing/dtoa.c:#define PRIVATE_mem ((PRIVATE_MEM+sizeof(double)-1)/sizeof(double))
./ext/bigdecimal/bigdecimal.c:    nc += (nc + mc - 1) / mc + 1;
./ext/bigdecimal/bigdecimal.c:    mx = (mx + BASE_FIG - 1) / BASE_FIG;    /* Determine allocation unit. */
./ext/bigdecimal/bigdecimal.c:    mf = (mf + BASE_FIG - 1) / BASE_FIG + 2; /* Needs 1 more for div */
./ext/bigdecimal/bigdecimal.c:    nalloc = (ni + nf + BASE_FIG - 1) / BASE_FIG + 1;    /* set effective allocation */
./ext/bigdecimal/bigdecimal.c:    size_t const round_limit = (VpGetPrecLimit() + BASE_FIG - 1) / BASE_FIG;
./ext/bigdecimal/bigdecimal.c:    if ((ix + BASE_FIG - 1) / BASE_FIG > ixDigit + 1) return 0;
./ext/bigdecimal/bits.h:#define roomof(x, y) (((x) + (y) - 1) / (y))
./internal/numeric.h:    VALUE values[(SIZEOF_DOUBLE + SIZEOF_VALUE - 1) / SIZEOF_VALUE];
./regcomp.c:    OnigDistance str_len = (byte_len + mb_len - 1) / mb_len;
./bignum.c:    size_t num_bdigits = (num_bits + BITSPERDIG - 1) / BITSPERDIG;
./missing/dtoa.c:#define PRIVATE_mem ((PRIVATE_MEM+sizeof(double)-1)/sizeof(double))
./numeric.c:    char buf[float_dig + (decimal_mant + CHAR_BIT - 1) / CHAR_BIT + 10];
./gc.c:#define CEILDIV(i, mod) (((i) + (mod) - 1) / (mod))
```

### Naming:

I was not sure whether to name it `ceildiv` or `divceil` because there are both `divmod` and `fdiv`.

Since `divmod` is a method that returns two elements, the quotient and the remainder,

while fdiv is a method that performs Float division, I decided to follow fdiv.

## Associated revisions

Revision 0617cba197cdff626ee9c74cece480df31d384ef - 08/13/2022 02:23 AM - nobu (Nobuyoshi Nakada)

[DOC] Add the link to [Feature #18809]

Revision 0617cba197cdf626ee9c74cece480df31d384ef - 08/13/2022 02:23 AM - nobu (Nobuyoshi Nakada)

[DOC] Add the link to [Feature #18809]

Revision 0617cba1 - 08/13/2022 02:23 AM - nobu (Nobuyoshi Nakada)

[DOC] Add the link to [Feature #18809]

## History

#1 - 05/28/2022 01:50 AM - nobu (Nobuyoshi Nakada)

I'm positive.

It may be nice to alias div as floordiv too?

#2 - 05/28/2022 06:26 AM - mrkn (Kenta Murata)

Julia provides [cld](#) and [fld](#) for the ceiling and flooring division, respectively. They are implemented as aliases of `div` with rounding mode argument, like `cld(a, b) = div(a, b, RoundUp)`. It may be good to introduce an optional rounding mode argument in `div` in addition to adding these divisions.

#3 - 06/21/2022 12:45 PM - Dan0042 (Daniel DeLorme)

Why not simply use `a.fdiv(b).ceil` ?

It expresses the intent of the code clearly, and I doubt there would be a measurable difference in performance except in the tightest of tight loops.

#4 - 06/21/2022 06:21 PM - sawa (Tsuyoshi Sawada)

Dan0042 (Daniel DeLorme) wrote in [#note-3](#):

Why not simply use `a.fdiv(b).ceil` ?

It expresses the intent of the code clearly, and I doubt there would be a measurable difference in performance except in the tightest of tight loops.

```
a = 9999999999999999
```

```
(a + b - 1) / b # => 9999999999999999
```

```
a.fdiv(b).ceil # => 1000000000000000000
```

#5 - 07/21/2022 05:03 AM - matz (Yukihiro Matsumoto)

Let's add Integer#ceildiv.

#6 - 07/21/2022 11:58 AM - mame (Yusuke Endoh)

Additional information.

- We do introduce only Integer#ceildiv.
- We do not introduce Numeric#ceildiv until we see the need. There is already Numeric#div, but a consistency with it is not a sufficient reason to introduce it.
- We do not introduce Numeric#floordiv.
- 3.ceildiv(-2) should return -1, which is  $\text{ceil}(-3/2)$ . Note that the naive implementation of  $(a + b - 1) / b$ , which returns  $(3 + (-2) - 1) / (-2) = 0$ . (As far as we glanced at the PR, it is implemented correctly.)

#7 - 07/21/2022 10:52 PM - kyanagi (Kouhei Yanagita)

Thank you for accepting.

I updated the PR. The PR contains only Integer#ceildiv.

#8 - 08/11/2022 03:06 AM - kyanagi (Kouhei Yanagita)

Are there any blocking issues?

If exist, I will work to resolve them.

#9 - 08/13/2022 02:24 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset [git|0617cba197cdff626ee9c74cece480df31d384ef](#).

---

[DOC] Add the link to [Feature [#18809](#)]