Daily Coding Problem #137

Problem

This problem was asked by Amazon.

Implement a bit array.

A bit array is a space efficient array that holds a value of `1` or `0` at each index.

- `init(size)`: initialize the array with `size`
- `set(i, val)`: updates index at `i` with `val` where `val` is either `1` or `0`.
- `get(i)`: gets the value at index `i`.

Solution

Although Python has arbitrary precision, let's assume each Python `int` has 32 bits.

We can use a list of ints to store `size` bits. Since `size` may not be divisible by `32`, we find the ceiling of `size / 32` to figure out the number of integers we need in the list.

Using the above strategy, we would store the first 32 bits in `list[0]`, the next 32 bits in `list[1]`, and so forth.

For the `get(i)` method, we first find the index in the list that holds the i^th bit, which is at `i / BITS_PER_INT`. Then we find the offset in the integer which holds the bit, which is at `i % BITS_PER_INT`. After these steps, we can extract the bit like so: `(self._list[list_idx] >> int_idx) & 1`

```
(self._list[list_idx] >> int_idx) & 1
```

As for `set(i, val)`, we use the following formula:

```
self._list[list_idx] ^= (-val ^ self._list[list_idx]) & (1 << int_idx)
```

The above formula works since if `val` is `0`, then the i^{th} bit ends up getting xord by itself, resulting in `0`. If `val` is 1, then, the i^{th} bit gets xored by itself again, which results in 1.

```
import math


BITS_PER_INT = 32


class BitArray(object):
    def __init__(self, size):
        self._list = [0] * int(math.ceil(size / float(BITS_PER_INT)))
        self._size = size

    def get(self, i):

        if i < 0 or i >= self._size:
            raise IndexError('Index out of bounds')

        list_idx = i / BITS_PER_INT
        int_idx = i % BITS_PER_INT

        return (self._list[list_idx] >> int_idx) & 1

    def set(self, i, val):
        if i < 0 or i >= self._size:
            raise IndexError('Index out of bounds')

        list_idx = i / BITS_PER_INT
        int_idx = i % BITS_PER_INT

        self._list[list_idx] ^= (-val ^ self._list[list_idx]) & (1 << int_idx)
```

`get(i)` and `set(i)` takes O(1) and the data structure takes O(N) space.