Problem 2166: Design Bitset

Problem Information

Difficulty: Medium
Acceptance Rate: 32.34%
Paid Only: No
Tags: Array, Hash Table, String, Design

Problem Description

A **Bitset** is a data structure that compactly stores bits.

Implement the `Bitset` class:

* `Bitset(int size)` Initializes the Bitset with `size` bits, all of which are `0`. * `void fix(int idx)` Updates the value of the bit at the index `idx` to `1`. If the value was already `1`, no change occurs. * `void unfix(int idx)` Updates the value of the bit at the index `idx` to `0`. If the value was already `0`, no change occurs. * `void flip()` Flips the values of each bit in the Bitset. In other words, all bits with value `0` will now have value `1` and vice versa. * `boolean all()` Checks if the value of **each** bit in the Bitset is `1`. Returns `true` if it satisfies the condition, `false` otherwise. * `boolean one()` Checks if there is **at least one** bit in the Bitset with value `1`. Returns `true` if it satisfies the condition, `false` otherwise. * `int count()` Returns the **total number** of bits in the Bitset which have value `1`. * `String toString()` Returns the current composition of the Bitset. Note that in the resultant string, the character at the `ith` index should coincide with the value at the `ith` bit of the Bitset.

Example 1:

Input ["Bitset", "fix", "fix", "flip", "all", "unfix", "flip", "one", "unfix", "count", "toString"] [[5], [3], [1], [], [], [0], [], [], [0], [], []] **Output** [null, null, null, null, false, null, null, true, null, 2, "01010"] **Explanation** Bitset bs = new Bitset(5); // bitset = "00000". bs.fix(3); // the value at idx = 3 is updated to 1, so bitset = "00010". bs.fix(1); // the value at idx = 1 is updated to 1, so bitset = "01010". bs.flip(); // the value of each bit is flipped, so bitset = "10101". bs.all(); // return False, as not all values of the bitset are 1. bs.unfix(0); // the value at idx = 0 is updated to 0, so bitset = "00101". bs.flip(); // the value of each bit is flipped, so bitset = "11010". bs.one(); // return True, as there is at least 1 index with value 1. bs.unfix(0); // the value at idx = 0 is updated to 0, so bitset = "01010". bs.count(); // return 2, as there are 2 bits with value 1. bs.toString(); //

return "01010", which is the composition of bitset.

Constraints:

* `1 <= size <= 105` * `0 <= idx <= size - 1` * At most `105` calls will be made **in total** to `fix`, `unfix`, `flip`, `all`, `one`, `count`, and `toString`. * At least one call will be made to `all`, `one`, `count`, or `toString`. * At most `5` calls will be made to `toString`.

Code Snippets

C++:

```
class Bitset {
public:
Bitset(int size) {

}

void fix(int idx) {

}

void unfix(int idx) {

}

void flip() {

}

bool all() {

}

bool one() {

}

int count() {

}
```

```
string toString() {

}
};

/**
* Your Bitset object will be instantiated and called as such:
* Bitset* obj = new Bitset(size);
* obj->fix(idx);
* obj->unfix(idx);
* obj->flip();
* bool param_4 = obj->all();
* bool param_5 = obj->one();
* int param_6 = obj->count();
* string param_7 = obj->toString();
*/
```

Java:

```
class Bitset {

public Bitset(int size) {

}

public void fix(int idx) {

}

public void unfix(int idx) {

}

public void flip() {

}

public boolean all() {

}
```

```
public boolean one() {

}

public int count() {

}

public String toString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = new Bitset(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* boolean param_4 = obj.all();
* boolean param_5 = obj.one();
* int param_6 = obj.count();
* String param_7 = obj.toString();
*/
```

Python3:

```
class Bitset:

def __init__(self, size: int):


def fix(self, idx: int) -> None:


def unfix(self, idx: int) -> None:


def flip(self) -> None:


def all(self) -> bool:
```

```
def one(self) -> bool:


def count(self) -> int:


def toString(self) -> str:



# Your Bitset object will be instantiated and called as such:
# obj = Bitset(size)
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.toString()
```