Number Of Different Bits (Medium)

Determine the number of bits that are different for two given integers.

Examples

- 5(“0101”) and 8(“1000”) has 3 different bits

```
int differentBits(int a, int b){
    int count = 0;
    for(unsigned int c = a ^ b; c != 0; c = c >> 1){
        count += (c & 1);
    }
    return count;
}
```

Time Complexity: O(k), k is the number of bits of the integer.

// 正确！Accepted！

// 复杂度也等于 O(logn)，n=max(a, b)，log 是以 2 为底的对数函数。

All Unique Characters II (Medium)

Determine if the characters of a given string are all unique.

Assumptions

- We are using ASCII charset, the value of valid characters are from 0 to 255
- The given string is not null

Examples

- all the characters in "abA+\8" are unique
- "abA+\a88" contains duplicate characters

```
bool isUnique(string s){
    vector<int> dic(8, 0);
    for(int i = 0; i < s.length(); i++){
        int row = s[i] / 32;
        int col = s[i] % 32;
        if( (dic[row] >> col) & 1)
            return false;
        else
            dic[row] = dic[row] | (1 << col);
    }
    return true;
}
```

Time Complexity: O(n), n is the length of the given string.

// 正确！Accepted！

Counting Bits (Medium)

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example:
For `num = 5` you should return `[0,1,1,2,1,2]`.

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

```
vector<int> countBits(int num){
    vector<int> result(num+1, 0);
    for(int i = 1; i <= num; i++){
        result[i] = result[i >> 1] + (i & 1);
    }
    return result;
}
```

Time Complexity: O(n)

// 正确！Accepted！

Missing Number I (Medium)

Given an integer array of size N - 1, containing all the numbers from 1 to N except one, find the missing number.

Assumptions

- The given array is not null, and N >= 1

Examples

- A = {2, 1, 4}, the missing number is 3
- A = {1, 2, 3}, the missing number is 4
- A = {}, the missing number is 1

```
int missingNum(vector<int> nums){
    if(nums.empty())
        return 1;
    int n = nums.size();
    int x1 = nums[0];
    int x2 = 1;
    for(int i = 1; i < n; i++){
        x1 = x1 ^ nums[i];
    }
    for(int i = 2; i <= n + 1; i++){
        x2 = x2 ^ i;
    }
    return int(x1 ^ x2);
}
```

Time Complexity: O(n)

// 正确！Accepted！

Bitwise AND of Numbers Range (Medium)

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

For example, given the range [5, 7], you should return 4.

```
int rangeAnd(int m, int n){
    int count = 0;
    while(m != n){
        m = m >> 1;
        n = n >> 1;
        count ++;
    }
    return int (m << count);
}
```

Time Complexity: O(k), k is the number of bits of integer.

// 正确！Accepted!

Compress String (Hard) (from HW10)

Given a string, replace adjacent, repeated characters with the character followed by the number of repeated occurrences. If the character does not has any adjacent, repeated occurrences, it is not changed.

Assumptions

- The string is not null
- The characters used in the original string are guaranteed to be 'a '- 'z'

Examples

- "abbcccdeee" →" ab2c3de3"

```
void compressString(string& s){
    int slow = 0;
    for(int i = 0; i < s.length(); i++){
        int count = 1;
        while(i < s.length() - 1 && s[i] == s[i+1]){
            i++;
            count++;
        }
        s[slow++] = s[i];
        if(count > 1) {
            string nums = to_string(count);
            for(int i = 0; i < nums.length(); i++){
                s[slow++] = nums[i];
            }
        }
    }
    s.resize(slow);
    return;
}
```

Time Complexity: O(n), since we traverse the string once.

// 正确！Accepted！