



Подсчитайте общее количество битов в первых N натуральных числах (все числа от 1 до N)



Последнее обновление: 13 мая 2024 г.

Учитывая положительное целое число N , задача состоит в том, чтобы подсчитать общее количество установленных битов в **двоичном представлении** всех чисел от 1 до N .

Примеры:

Ввод: $N = 3$

Выход: 4

Пояснение: Числа от 1 до 3: {1, 2, 3}

Двоичное представление 1: 01 -> Установить биты = 1

Двоичное представление 2: 10 -> Установить биты = 1

Двоичное представление 3: 11 -> Установленные биты = 2

Всего установленных бит от 1 до 3 = $1 + 1 + 2 = 4$

Вход: $N = 6$

Выход: 9

Вход: $N = 7$

Выход: 12

Рекомендуемая Проблема

Подсчитать общее количество Setbits

Математический Битовая магия +2 еще

Решить Проблему

Количество отправленных:
7,8 тыс.

Источник: Вопрос для интервью Amazon.

Подсчитайте общее количество битов, преобразуя каждое число в его двоичное представление:

Идея состоит в том, чтобы преобразовать каждое число от 1 до N в двоичное и посчитать установленные биты в каждом числе отдельно.

Выполните следующие действия, чтобы понять, как:

- Пройти цикл от **1** до **N**
- Для каждого целого числа от **1** до **N** :
 - Преобразуйте число в его двоичное представление
 - Добавьте к ответу количество **единиц** в двоичном представлении.
- Возвращает общее количество установленных битов.

Ниже приведена реализация описанного выше подхода:

C++

C

Джава

Питон

C#

JavaScript

PHP

```
// A simple program to count set bits
// in all numbers from 1 to n.
#include <iostream>
using namespace std;

// A utility function to count set bits
// in a number x
unsigned int countSetBitsUtil(unsigned int x);

// Returns count of set bits present in x
```

```

        for (int i = 1; i <= n; i++)

            bitCount += countSetBitsUtil(i);

    return bitCount;
}

// A utility function to count set bits
// in a number x
unsigned int countSetBitsUtil(unsigned int x)
{
    if (x <= 0)
        return 0;
    return (x % 2 == 0 ? 0 : 1) + countSetBitsUtil(x / 2);
}

// Driver program to test above functions
int main()
{
    int n = 4;
    cout << "Total set bit count is " << countSetBits(n);
    return 0;
}

// This code is contributed by shivanisinghss2110.

```

Выход

Общее количество установленных битов равно 5.

Временная сложность: $O(N \cdot \log(N))$, где N — заданное целое число, а время $\log(N)$ используется для двоичного преобразования числа.

Вспомогательное пространство: $O(1)$.

Метод 2 (простой и эффективный, чем метод 1). Если мы наблюдаем биты с крайней правой стороны на расстоянии i , то биты инвертируются после позиции 2^i в вертикальной последовательности. например $n = 5$; $0 = 0000$ $1 = 0001$ $2 = 0010$ $3 = 0011$ $4 = 0100$ $5 = 0101$ Обратите внимание на самый правый бит ($i = 0$), биты переворачиваются после ($2^0 = 1$) Обратите внимание на третий крайний правый бит ($i = 2$) биты переворачиваются после ($2^2 = 4$). Итак, мы можем считать биты по вертикали, так что в i -й правой позиции большинство битов позиции будут перевернуты после 2^i итерации;

```
#include <bits/stdc++.h>
using namespace std;

// Function which counts set bits from 0 to n
int countSetBits(int n)
{
    int i = 0;

    // ans store sum of set bits from 0 to n
    int ans = 0;

    // while n greater than equal to 2^i
    while ((1 << i) <= n) {

        // This k will get flipped after
        // 2^i iterations
        bool k = 0;

        // change is iterator from 2^i to 1
        int change = 1 << i;

        // This will loop from 0 to n for
        // every bit position
        for (int j = 0; j <= n; j++) {

            ans += k;

            if (change == 1) {
                k = !k; // When change = 1 flip the bit
                change = 1 << i; // again set change to 2^i
            }
            else {
                change--;
            }
        }

        // increment the position
        i++;
    }

    return ans;
}

// Main Function
int main()
{
    int n = 17;
    cout << countSetBits(n) << endl;
    return 0;
}
```

Выход

Временная сложность: $O(k \cdot n)$,

где k = количество битов для представления числа n

$k \leq 64$.

Метод 3 (сложный).

Если входное число имеет форму $2^b - 1$, например, 1, 3, 7, 15... и т. д., количество установленных бит равно $b \cdot 2^{b-1}$. Это связано с тем, что для всех чисел от 0 до $(2^b) - 1$, если вы дополните и перевернете список, вы получите один и тот же список (половина битов включена, половина выключена).

Если число не имеет всех установленных битов, то некоторая позиция m является позицией крайнего левого установленного бита. Количество установленных битов в этой позиции равно $n - (1 \ll m) + 1$. Остальные установленные биты состоят из двух частей:

- 1) Биты в $(m-1)$ позициях до точки, где самый левый бит становится 0 и
- 2) числа $2^{(m-1)}$ ниже этой точки, что является закрытой формой выше.

Самый простой способ взглянуть на это — рассмотреть число 6:

```
0|0
0 0
|0 1 0
|1 0 0|1 1
-|--
1|0 0
1|0 1
1|1 0
```

Крайний левый установленный бит находится в позиции 2 (позиции считаются начиная с 0). Если мы замаскируем это, то останется 2 («1 0» в правой части последней строки). Таким образом, количество битов во 2-й позиции (нижний левый квадрат) равно 3 (то есть $2 + 1$). Установленные биты от 0 до 3 (верхнее правое поле выше) равны $2 \cdot 2^{(2-1)} = 4$. Поле в правом нижнем углу — это оставшиеся биты, которые мы еще не подсчитали, и это количество установленных битов от 0 до 3 (верхнее правое поле выше). биты для всех чисел до 2 (значение последней записи в правом нижнем поле), которые можно вычислить рекурсивно.

C++

C

Джава

Питон

C#

JavaScript

```
#include <bits/stdc++.h>
```

```

    /* Returns position of leftmost set bit.
    The rightmost position is considered
    as 0 */
    unsigned int getLeftmostBit(int n)
    {
        int m = 0;
        while (n > 1) {
            n = n >> 1;
            m++;
        }
        return m;
    }

    /* Given the position of previous leftmost
    set bit in n (or an upper bound on
    leftmost position) returns the new
    position of leftmost set bit in n */
    unsigned int getNextLeftmostBit(int n, int m)
    {
        unsigned int temp = 1 << m;
        while (n < temp) {
            temp = temp >> 1;
            m--;
        }
        return m;
    }

    // The main recursive function used by countSetBits()
    unsigned int _countSetBits(unsigned int n, int m);

    // Returns count of set bits present in
    // all numbers from 1 to n
    unsigned int countSetBits(unsigned int n)
    {
        // Get the position of leftmost set
        // bit in n. This will be used as an
        // upper bound for next set bit function
        int m = getLeftmostBit(n);

        // Use the position
        return _countSetBits(n, m);
    }

    unsigned int _countSetBits(unsigned int n, int m)
    {
        // Base Case: if n is 0, then set bit
        // count is 0
        if (n == 0)
            return 0;

        /* get position of next leftmost set bit */
        m = getNextLeftmostBit(n, m);

        // If n is of the form 2^x-1, i.e., if n
        // is like 1, 3, 7, 15, 31, .. etc,
        // then we are done.
        // Since positions are considered starting
        // from 0, 1 is added to m
        if (n == ((unsigned int)1 << (m + 1)) - 1)
            return (unsigned int)(m + 1) * (1 << m);

        // update n for next recursive call
        n = n - (1 << m);
        return (n + 1) + countSetBits(n) + m * (1 << (m - 1));
    }

```

```

    int n = 17;
    cout << "Total set bit count is " << countSetBits(n);
    return 0;
}

```

// This code is contributed by rathbhupendra

Выход

Общее количество установленных бит составляет 35.

Временная сложность: $O(\log n)$. С первого взгляда на реализацию временная сложность выглядит больше. Но если мы присмотримся поближе, операторы внутри цикла while функции getNextLeftmostBit() выполняются для всех нулевых битов числа n. И количество раз, когда выполняется рекурсия, меньше или равно заданным битам в n. Другими словами, если элемент управления входит в цикл while getNextLeftmostBit(), то он пропускает эти многочисленные биты в рекурсии.

Спасибо Agatsu и IC за предложение этого решения.

Вот еще одно решение, предложенное **Пиюшем Капуром**.

Простое решение, использующее тот факт, что для i-го младшего бита ответ будет

$$(N/2^i) * 2^{(i-1)} + X$$

где

$$X = N\%(2^i) - (2^{(i-1)} - 1)$$

если

$$N\%(2^i) > (2^{(i-1)} - 1)$$

C++

Джава

Питон

C#

JavaScript

```

int getSetBitsFromOneToN(int N)
{
    int two = 2, ans = 0;
    int n = N;

```

```

    n >>= 1;
}
return ans;
}

```

Метод 4 (рекурсивный)

Подход:

Для каждого числа «n» будет число a, где $a \leq n$ и a — это полная степень двойки, например 1,2,4,8.....

Пусть $n = 11$, теперь мы видим, что

Числа до n:

```

0 -> 0000000
1 -> 0000001
2 -> 0000010
3 -> 0000011
4 -> 0000100
5 -> 0000101
6 -> 0000110
7 -> 0000111
8 -> 0001000
9 -> 0001001
10 -> 0 001010
11 -> 0001011

```

Теперь мы видим, что от 0 до $\text{row}(2,1)-1 = 1$ мы можем объединить самые верхние элементы с самыми нижними,

а количество установленных битов в паре равно 1.

Аналогично для $\text{row}(2,2)-1 = 4$, пары:

00 и 11

01 и 10,

здесь количество установленных битов в паре равно 2, поэтому в обеих парах равно 4.

Аналогично мы можем видеть для 7, 15 и вскоре.... .

поэтому мы можем обобщить это,

$\text{count}(x) = (x * \text{row}(2, (x-1)))$,

здесь x — позиция установленного бита наибольшей степени от 2 до n

для $n = 8$, $x = 3$.

для $n = 4$, $x = 2$

для $n = 5$, $x = 2$,

поэтому теперь для $n = 11$

мы добавили заданное количество бит от 0 до 7, используя **$\text{count}(x) = (x * \text{row}(2, (x-1)))$**

которое можно вычислить, используя $11 - 8 + 1 = (n - \text{pow}(2, x) + 1)$

Теперь обратите внимание, что после удаления передних битов из остальных чисел мы снова получаем число от 0 до некоторого m, поэтому мы можем рекурсивно вызывать нашу ту же функцию для следующего набора чисел,

вызывая `countSetBits(n - pow(2), x)`

8 -> 1000 -> 000 -> 0
9 -> 1001 -> 001 -> 1
10 -> 1010 -> 010 -> 2
11 -> 1011 -> 011 -> 3

Код:

C++

Джава

Питон

C#

JavaScript

```
#include <bits/stdc++.h>
using namespace std;

int findLargestPower(int n)
{
    int x = 0;
    while ((1 << x) <= n)
        x++;
    return x - 1;
}

int countSetBits(int n)
{
    if (n <= 1)
        return n;
    int x = findLargestPower(n);
    return (x * pow(2, (x - 1))) + (n - pow(2, x) + 1)
        + countSetBits(n - pow(2, x));
}

int main()
{
    int N = 17;
    cout << countSetBits(N) << endl;
    return 0;
}
```

Выход

35

Временная сложность: $O(\log N)$

0 -> 0000000 8 -> 001000 16 -> 010000 24 -> 011000
1 -> 0000001 9 -> 001001 17 -> 010001 25 -> 011001
2 -> 0000010 10 -> 001010 18 -> 010010 26 -> 011010
3 -> 0000011 11 -> 001011 19 -> 010010 27 -> 011011
4 -> 0000100 12 -> 001100 20 -> 010100 28 -> 011100
5 -> 0000101 13 -> 001101 21 -> 010101 29 -> 011101
6 -> 0000110 14 -> 001110 22 -> 010110 30 -> 011110
7 -> 0000111 15 -> 001111 23 -> 010111 31 -> 011111

Ввод: N = 4

Выход: 5

Ввод: N = 17

Выход: 35

Подход: Распознавание образов

Пусть «N» — любое произвольное число и рассмотрим индексацию справа налево (самое правое — 1); тогда ближайший $Pow = \text{pow}(2, i)$.

Теперь, когда вы запишете все числа от 1 до N, вы увидите закономерность, указанную ниже:

Для каждого индекса i есть ровно неуставленные непрерывные элементы ближайшего $Pow/2$, за которыми следуют установленные элементы ближайшего $Pow/2$.

На протяжении всего решения я буду использовать эту концепцию.

Вы можете ясно увидеть изложенную выше концепцию в приведенной выше таблице.

Общая формула, которую я придумал:

a. $\text{addRemaining} = \text{mod} - (\text{ближайшийPos}/2) + 1$, если $\text{mod} \geq \text{NearestPow}/2$;

b. $\text{totalSetBitCount} = \text{totalRep} * (\text{ближайшийPow}/2) + \text{addRemaining}$

где totalRep -> общее количество повторений шаблона с индексом i

addRemaining -> общее количество установленных бит, оставшихся для добавления после исчерпания шаблона

я = 1 => ближайшееPos = pow(2,1) = 2; totalRep = (17+1)/2 = 9 (добавьте 1 только для i=1)

$$\text{мод} = 17\%2 = 1$$

addRemaining = 0 (только для базового случая)

$$\text{totalSetBitCount} = \text{totalRep} * (\text{ближайшийPos}/2) + \text{addRemaining} = 9 * (2/2) + 0 = 9 * 1 + 0 = 9$$

я = 2 => ближайшееPos = pow(2, 2)=4; всего повторов = 17/4 = 4

$$\text{мод} = 17\%4 = 1$$

$$\text{mod}(1) < (4/2) \Rightarrow 1 < 2 \Rightarrow \text{addRemaining} = 0$$

$$\text{totalSetBitCount} = 9 + 4 * (2) + 0 = 9 + 8 = 17$$

я = 3 => ближайшийPow = pow(2,3) = 8; всего повторов = 17/8 = 2

$$\text{мод} = 17\%8 = 1$$

$$\text{мод} < 4 \Rightarrow \text{addRemaining} = 0$$

$$\text{totalSetBitCount} = 17 + 2 * (4) + 0 = 17 + 8 + 0 = 25$$

я = 4 => ближайшийPow = pow(2, 4) = 16; всего повторов = 17/16 = 1

$$\text{мод} = 17\%16 = 1$$

$$\text{мод} < 8 \Rightarrow \text{addRemaining} = 0$$

$$\text{totalSetBitCount} = 25 + 1 * (8) + 0 = 25 + 8 + 0 = 33$$

Мы не можем просто оперировать следующей степенью (32) как $32 > 17$. Кроме того, поскольку первые полубиты будут только 0, нам нужно найти расстояние данного числа (17) от последней степени, чтобы напрямую получить количество добавляемых единиц.

я = 5 => ближайшийPow = (2, 5) = 32 (базовый случай 2)

$$\text{последнийPow} = \text{pow}(2, 4) = 16$$

$$\text{мод} = 17\%16 = 1$$

$$\text{totalSetBit} = 33 + (\text{mod}+1) = 33 + 1 + 1 = 35$$

Следовательно, общее количество установленных бит от 1 до 17 равно 35.

Попробуйте выполнить итерацию с N = 30, чтобы лучше понять решение.

Решение



```

#include <bits/stdc++.h>
#include <iostream>

using namespace std;

int GetLeftMostSetBit(int n)
{
    int pos = 0;

    while (n > 0) {
        pos++;
        n >>= 1;
    }

    return pos;
}

int TotalSetBitsFrom1ToN(int n)
{
    int leftMostSetBitInd = GetLeftMostSetBit(n);

    int totalRep, mod;
    int nearestPow;
    int totalSetBitCount = 0;
    int addRemaining = 0;

    int curr
        = 0; // denotes the number of set bits at index i

    // cout<<"leftMostSetBitInd: "<<leftMostSetBitInd<<endl;

    for (int i = 1; i <= leftMostSetBitInd; ++i) {
        nearestPow = pow(2, i);
        if (nearestPow > n) {
            int lastPow = pow(2, i - 1);
            mod = n % lastPow;
            totalSetBitCount += mod + 1;
        }
        else {
            if (i == 1 && n % 2 == 1) {
                totalRep = (n + 1) / nearestPow;
                mod = nearestPow % 2;
                addRemaining = 0;
            }
            else {
                totalRep = n / nearestPow;
                mod = n % nearestPow;

                if (mod >= (nearestPow / 2)) {
                    addRemaining
                        = mod - (nearestPow / 2) + 1;
                }
                else {
                    addRemaining = 0;
                }
            }

            curr = totalRep * (nearestPow / 2)
                + addRemaining;
            totalSetBitCount += curr;
        }
    }
}

```

```

        return totalSetBitCount;
    }

    int main()
    {
        std::cout << TotalSetBitsFrom1ToN(4) << endl;
        std::cout << TotalSetBitsFrom1ToN(17) << endl;
        std::cout << TotalSetBitsFrom1ToN(30) << endl;
        return 0;
    }
    // This code is contributed by rjkrnsgh

```

Выход

ДПО Практикуйте побитовые алгоритмы MCQ на побитовых алгоритмах Учебник по бивайзным алгоритмам Двоич

35

75

Временная сложность: $O(\log(n))$

Метод 6 (использование встроенных функций)

__builtin_popcount() : подсчитывает количество установленных битов в числе.

C++

Джава

Питон

C#

JavaScript

```

#include <bits/stdc++.h>
#include <iostream>

int TotalSetBitsFrom1ToN(int n)
{
    int totalSetBitCount = 0;
    for(int number = 1 ; number <= n ; number++)
    {
        totalSetBitCount += __builtin_popcount(number);
    }
    return totalSetBitCount;
}

int main()
{
    std::cout << TotalSetBitsFrom1ToN(4) << std::endl;
    std::cout << TotalSetBitsFrom1ToN(17) << std::endl;
    std::cout << TotalSetBitsFrom1ToN(30) << std::endl;
    return 0;
}
// This code is contributed by yash_cse_21

```

5
35
75

Временная сложность: $O(n \cdot k)$, где n — общее количество чисел, а k — общее количество бит в числе.

Космическая сложность: $O(1)$

Вы хотите преодолеть разрыв между **структурами данных и алгоритмами (DSA)** и **разработкой программного обеспечения**? Погрузитесь в наш [курс DSA to Development — от начинающего до продвинутого уровня](#) на GeeksforGeeks, созданный как для начинающих разработчиков, так и для опытных программистов. Изучите основные навыки кодирования, принципы разработки программного обеспечения и методы практического применения с помощью практических **проектов** и примеров из реальной жизни. Независимо от того, начинаете ли вы свой путь или стремитесь усовершенствовать свои навыки, этот курс даст вам возможность создавать надежные программные решения. Готовы усовершенствовать свои навыки программирования? Зарегистрируйтесь сейчас и измените свои способности в программировании!



126

Предыдущая статья

Подсчитать множество битов в целом числе

Следующая статья

Проверьте, установлены ли в номере только первый и последний биты.

Похожие чтения

Подсчет пар {X, Y} из массива, в котором сумма количества установленных...

Учитывая массив `arr[]`, состоящий из N неотрицательных целых чисел и целого числа M , задача состоит в том, чтобы найти количество неупорядоченных пар $\{X, \dots\}$

14 минут чтения

Функция карты Python | Подсчитайте общее количество бит во всех числах ...

Учитывая положительное целое число n , подсчитайте общее количество установленных битов в двоичном представлении всех чисел от 1 до n . Примеры:

Подсчитайте общее количество бит во всех числах от L до K.

Учитывая два положительных целых числа L и R, задача состоит в том, чтобы подсчитать общее количество установленных битов в двоичном представлении...

15+ минут чтения

Вывести числа, имеющие первый и последний биты в качестве единственн...

Учитывая целое положительное число n. Проблема состоит в том, чтобы напечатать числа в диапазоне от 1 до n, имеющие первый и последний биты в...

10 минут чтения

Сгенерируйте последовательность из первых X натуральных чисел, которая...

Учитывая два целых числа X и S, задача состоит в том, чтобы построить последовательность различных целых чисел из диапазона [1, X] такую, что сумм...

10 минут чтения

[Посмотреть больше статей](#)

Тэги статьи :

[Битовая магия](#)

[ДПО](#)

[Амазонка](#)

Теги практики:

[Амазонка](#)

[Битовая магия](#)



Sanchhaya Education Private Limited

A-143, 9-й этаж, Sovereign Corporate
Tower, сектор 136, Нойда, Уттар-Прадеш
- 201305

Компания

О нас
Юридический
В СМИ
Связаться с нами
Рекламируйте с нами
Корпоративное решение GFG
Программа обучения по трудоустройству
Сообщество
GeeksforGeeks

Языки

Питон
Джава
C++
PHP
ГоЛанг
SQL
Язык Р
Учебное пособие по Android
Архив уроков

ДПО

Структуры данных
Алгоритмы
ДСА для начинающих
Основные проблемы DSA
Дорожная карта DSA
100 главных проблем на собеседовании DSA
Дорожная карта DSA, автор: Сандип Джейн
Все шпаргалки

Наука о данных и машинное обучение

Наука о данных с Python
Наука о данных для начинающих
Учебник по машинному обучению
ML Математика
Учебное пособие по визуализации данных
Учебное пособие по пандам
Учебное пособие по NumPy
Учебное пособие по НЛП
Учебное пособие по глубокому обучению

Веб-технологии

HTML
CSS
JavaScript
Машинопись
РеактJS
СледующийJS
Бутстрап
Веб-дизайн

Учебник по Python

Примеры программирования на Python
Python-проекты
Питон Ткинтер
Веб-скрапинг
Учебное пособие по OpenCV
Вопрос на собеседовании по Python
Джанго

Информатика

Операционные системы
Компьютерная сеть
Система управления базами данных
Программная инженерия
Проектирование цифровой логики
Инженерная математика
Разработка программного обеспечения
Тестирование программного обеспечения

DevOps

Гит
Линукс
ABC
Докер
Кубернетес
Лазурный
GCP
Дорожная карта DevOps

Системный дизайн

Дизайн высокого уровня
Низкоуровневый дизайн
UML-диаграммы
Руководство по собеседованию
Шаблоны проектирования
ООАД
Учебный курс по системному проектированию
Вопросы для собеседования

Подготовка к собеседованию

Соревновательное программирование
Лучший DS или алгоритм для CP
Процесс подбора персонала на уровне компании
Подготовка на уровне компании
Подготовка к профессиям
Загадки

Школьные предметы

Математика
Физика
Химия
Биология
Социальная наука
Английская грамматика
Коммерция
Мировой ГК

Видео GeeksforGeeks

ДПО
Питон
Джава
C++
Веб-разработка
Наука о данных
Предметы CS

@GeeksforGeeks, Sanchhaya Education Private Limited , Все права защищены.

Исходный текст

цените этот перевод
аш отзыв поможет нам улучшить Google
ереводчик



аксимально удобные возможности
тверждаете, что прочитали и поняли
[политику конфиденциальности.](#)