UNITECH 2023 GABROVO

INTERNATIONAL SCIENTIFIC CONFERENCE
17-18 November 2023, GABROVO

ЕДНО ПРИЛОЖЕНИЕ НА BITSET В ТЕОРИЯ НА КОДИРАНЕТО

Стоян Капралов, Валентина Кукенска

Технически универитет - Габрово

AN APPLICATION OF BITSET IN CODING THERY

Stoyan Kapralov, Valentina Kukenska

Technical University of Gabrovo

Abstract

In modern programming languages such as C/C++, Java, Python, JavaScript, Julia, Go, the ability to perform bitwise operations is provided. In these languages, bitwise operations are performed on the binary representations of integers. Only in the C++ and Java languages there are special tools that offer additional facilities for implementing bitwise operations. In the article we consider the basics of working with C++ bitset. An application for computing the spectrum of binary liner codes is presented.

Keywords: bitwise operations; C++ bitset, binary linear code.

ВЪВЕДЕНИЕ

В съвременните езици за програмиране като C/C++, Java, Python, JavaScript, Julia, Go е осигурена възможност за извършване на побитови операции [1]. В тези езици се извършват побитови операции върху двоичните представяния на цели числа. Само в езиците C++ и Java има специални инструменти, които предлагат допълнителни удобства за реализиране на побитови операции [2], [3], [4].

В статията представяме основните моменти при работа със C++ bitset и като приложение разглеждаме алгоритъм за изчисляване спектър на двоичен линеен код.

C++ BITSET

За начално запознаване с bitset, освен документацията [2], [3], е много полезна публикацията [5].

Класът bitset емулира масив от елементи от булев тип, но е оптимизиран по отношение на паметта, като всеки елемент заема само един бит. Броят на битовете се задава по време на компилация. Битовете в bitset са номерирани отдясно наляво – най-десният (последният) бит има номер 0, този вляво от него – номер 1 и т.н.

Всяка позиция може да бъде достъпвана индивидуално, както при обикновените масиви. Така, ако `b` е обект от клас bitset, `b[i]` осигурява достъп до *i*-тия бит.

Ако ще използваме bitset трябва да включим `#include <bitset>`
в началото на програмата.

Инициализация

`bitset<10> b;` `b=0000000000`
`bitset<10> b(53);` `b=0000110101`, защото $53_{(10)}=110101_{(2)}$
`bitset<10> b("110101");` `b=0000110101`

Преобразуващи функции

`b.to_string()` връща като стринг съдържанието на `b`
`b.to_ulong()` връща като число от тип unsigned long съдържанието на `b`
`b.to_ullong()` връща като число от тип unsigned long long съдържанието на `b`

Функции с отделни битове

`b.set(i)` дава на *i*-тия бит стойност 1
`b.reset(i)` дава на *i*-тия бит стойност 0
`b.flip(i)` променя *i*-тия бит (0 става 1 и обратно)
`b.test(i)` връща съдържанието на *i*-тия бит

Функции за всички битове

`b.size()` връща броя битове в `b` (дължината на `b`)
`b.count()` връща броя на битовете със стойност 1
`b.any()` връща true, ако някой бит има стойност 1
`b.none()` връща true, ако никой бит няма стойност 1
`b.all()` връща true, ако всички битове имат стойност 1

Операции

`b1 & b2` побитова конюнкция (and)
`b1 | b2` побитова дизюнкция (or)
`b1 ^ b2` побитова сума по модул 2 (xor)
`~b` обръща всички битове на `b`
`b << k` побитово изместване наляво на *k* бита
`b >> k` побитово изместване надясно на *k* бита

ПРИЛОЖЕНИЕ В ТЕОРИЯ НА КОДИРАНЕТО

В теория на кодирането линеен двоичен код C с дължина n е множество от двоични низове $x_1x_2x_3 \dots x_n$ с дължина n със следното свойство: ако $x \in C$ и $y \in C$, то $x + y \in C$. Събирането е покоординатно, като $x_i + y_i = (x_i + y_i)\%2,\ i = 1,2, \dots, n$.

Например, ако $x = 0100111$, $y = 0001011$, то $x + y = 0101100$.

Елементите на кода се наричат кодови думи. Тегло на кодова дума $\mathrm{wt}(x)$ е броят на единиците в x. Например $\mathrm{wt}(0100111) = 4$, $\mathrm{wt}(0001011) = 3$.

За всеки код C означаваме с $A_i, i = 0,1,2, \dots, n$ броя на кодовите думи с тегло i.

Например, ако C = {0000000, 1111111, 1000101, 0111010, 1100010, 0011101, 0100111, 1011000, 0110001, 1001110, 1110100, 0001011, 1010011, 0101100, 0010110, 1101001}, то $A_0 = 1$, $A_1 = 0$, $A_2 = 0$, $A_3 = 7$, $A_4 = 7$, $A_5 = 0$, $A_6 = 0$, $A_7 = 1$.

Редицата $A_0, A_1, A_2, \dots, A_n$ се нарича спектър на кода.

Един от начините за задаване на линеен код е посредством пораждаща матрица.

Пораждащата матрица G се състои от k кодови думи $c_1, c_2, \dots c_k$, а всичките кодови думи се получават като линейни комбинации на $c_1, c_2, \dots c_k$, т.е.

$$x = a_1c_1 + a_2c_2 + \cdots + a_kc_k,$$

където коефициентите $a_i,\ i = 1,2, \dots, k$ са 1 или 0. Ако $a_i = 1$, кодовата дума c_i участва в сумата, в противен случай – не участва.

Например, кодът C може да бъде зададен чрез следната пораждаща матрица:

$$G = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 1111111 \\ 1000101 \\ 1100010 \\ 0110001 \end{pmatrix}$$

Тогава

$x = c_2 + c_3 = 1000101 + 1100010 = 0100111$

и

$y = c_1 + c_2 + c_4 = 1111111 + 1000101 + 0110001 = 0001011.$

Тъй като за коефициентите $a_1, a_2, \dots, a_k$ има общо 2^k различни възможности, то код с пораждаща матрица от k реда има 2^k кодови думи. Числото k се нарича размерност на кода.

Една от основните задачи в теория на кодирането е по дадена пораждаща матрица да се намери спектърът на кода.

Следва един възможен подход за решаване на тази задача с използване на bitset.

За представяне на отделен ред от пораждащата матрица използваме `bitset<n>`. Тогава пораждащата матрица е вектор от `bitset<n>` с k елемента. Операцията събиране на кодови думи е всъщност XOR, която се означава с `^` в езика C++.

За получаване на всички комбинации от коефициентите използваме двоичното представяне на числата от 0 = 000...0 до 2^k–1 = 111...1. Позициите на единиците посочват кои редове на пораждащата матрица трябва да участват в сумата за получаване на поредната кодова дума.

Пресмятането на теглото на кодова дума е елементарно, защото bitset има функция `count`, която връща броя битове, равни на 1, т.е. точно интересуващото ни тегло.

```
// Програма на C++ за пресмятане
// спектър на линеен двоичен код

#include <iostream>
#include <vector>
#include <bitset>
using namespace std;

const int n = ..., k = ...;

int main()
{ vector<bitset<n>> G(k);
  for(int i=0; i<k; i++)
  { string s; cin >> s;
    G[i] = bitset<n>(s);
  }

  vector<int> A(n+1);
  for(int x=0; x<(1<<k); x++)
  { bitset<n> comb;
    for(int i=0; i<k; i++)
      if(x&(1<<i))comb      ^=
G[i];
    int wt = comb.count();
    A[wt]++;
  }

  for(int i=0; i<=n; i++)
    cout << A[i] << " ";
  cout << endl;

  return 0;
}
```

ЗАКЛЮЧЕНИЕ

В статията са представени основните моменти при работа със C++ bitset. Като правило в началните университетски курсове по програмиране с използване на езика за програмиране C++ въобще не се разглеждат възможностите на този инструмент. Аналогична е и ситуацията с Java и класа BitSet. Като илюстрация на удобствата на класа bitset е представен алгоритъм за решаване на класическа задача от теория на кодирането.

БЛАГОДАРНОСТИ

Изследванията, представени в статията, са частично финансирани по Проект 2209E/2022 на Технически университет – Габрово.

REFERENCES

[1] Learn to code, https://www.w3schools.com, Accessed 28/09/2023

[2] C++ Reference-1,

https://cplusplus.com/reference/bitset/bitset/, Accessed 28/09/2023

[3] C++ Reference-2, https://en.cppreference.com/w/cpp/utility/bitset, Accessed 28/09/2023

[4] Java™ Platform, Standard Edition 8, API Specification, https://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html, Accessed 28/09/2023

[5] Geeks for geeks, https://www.geeksforgeeks.org/cpp-bitset-and-its-application/, Accessed 28/09/2023