

bitset

1.介绍

bitset 在 `bitset` 头文件中

bitset是一个0-1串，每一位占一个bit，可以进行单点0/1修改，左移右移以及按位运算操作。

一个非常好用的用法是统计某个数是否出现过，类似一个桶。

另外由于一个bitset中每一位是一个bit，bitset的空间复杂度也比使用桶存储信息优秀。

```
1 //头文件
2 #include<bitset>
```

2.初始化定义

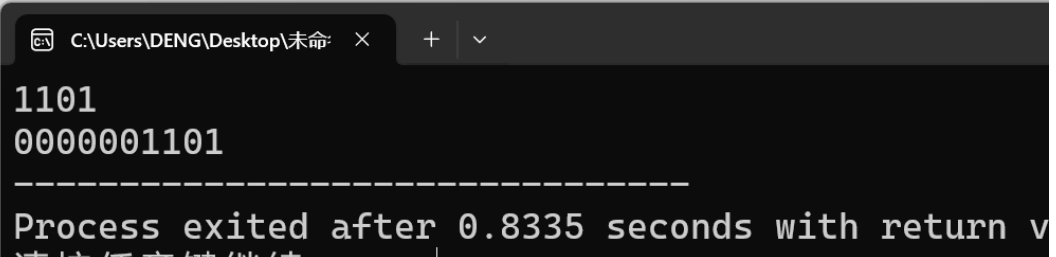
初始化方法

1	<code>bitset<n> a</code>	a有n位，每位都为0
2	<code>bitset<n> a(b)</code>	a是整数类型b的二进制副本
3	<code>bitset<n> a(s)</code>	a是string对象s的二进制副本

注意：n必须为常量表达式

默认是一个0-1字符串，而且bitset重载了<<和>>输入输出流，可以使用cin和cout来读入和输出一个bitset的所有元素。

```
未命名1.cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3  bitset<10> a;
4  int main() {
5
6      cin >> a;
7      cout << a;
8
9      return 0;
10 }
```



演示代码

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int main() {
4      // 1 无参构造
5      bitset<4> bitset1; //无参构造，长度为4，默认每一位为0
6
7      // 2 有参构造：数字作为参数
8      bitset<9> bitset2(12); //长度为9，将数字12转化为二进制保
存，前面用0补充
9
10     // 3 有参构造：string字符串作为参数
11     string s = "100101";
12     bitset<10> bitset3(s); //长度为10，将该字符串转化为二进制保
存，前面用0补充
```

```

13
14 // 4 有参构造: char字符数组 (字符串) 作为参数
15 char s2[] = "10101";
16 bitset<13> bitset4(s2); //长度为13, 前面用 0 补充
17
18 cout << bitset1 << endl; //0000
19 cout << bitset2 << endl; //000001100
20 cout << bitset3 << endl; //0000100101
21 cout << bitset4 << endl; //0000000010101
22
23 return 0;
24 }

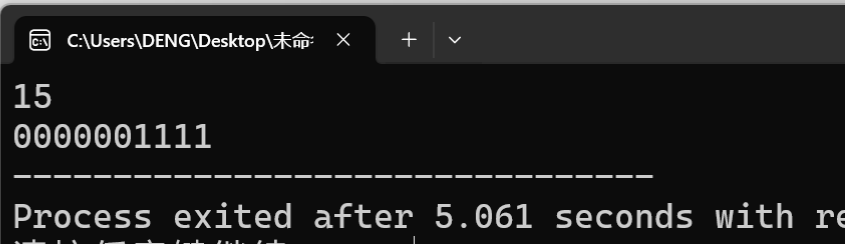
```

可以支持快速得到整数的二进制

```

using namespace std;
int b;
int main() {
    cin >> b;
    bitset<10> a(b);
    cout << a;
}

```



15
0000001111

Process exited after 5.061 seconds with re

PS: n必须为常量表达式 -----> 这个是bitset的一个麻烦

如果数字随机, 想**通过bitset直接转为二进制**麻烦就是必须知道数字转化为二进制的长度

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 int main(){
4
5     int t = 10;
6     bitset<t> f(100);
7     cout << f;
8
9     return 0;
0 }

```

访问

PS: 注意集合内部的下标是**从右向左**单调递增的，所以访问的顺序和实际输入的顺序相反，直接输出无所谓

```

1 //可以通过下标访问元素(类似数组)，注意最低位下标为0，类似于数的二进制表示，如下：
2 string s = "1011";
3 bitset<4> f(s);
4 for (int i = 0; i < 4; ++i) {
5     cout << f[i]; // 输出1101
6 }

```

注意：bitset访问时候是从右边（最低位）开始访问的，但是左右移位的时候还是按照原来的形式左右移位的

修改与运算

bitset中每一个元素可以通过下标的方式访问。一个长度为 N 的bitset下标编号为 $[0, N)$ 。

进行单点修改时，直接访问位置然后赋值即可：

```
1 s[pos] = x;
```

单点修改的复杂度为 $O(1)$

3.特性

bitset可以进行位操作

```
1 string x = "1001";
2 string y = "0011";
3 bitset<4> s1(x);
4 bitset<4> s2(y);
5
6 cout << (s1 & s2) << endl; // 0001 (按位与)
7
8 cout << (s1 | s2) << endl; // 1011 (按位或)
9
10 cout << (s1 ^ s2) << endl; // 1010 (按位异或)
11
12 cout << (s2 << 1) << endl; // 0110 (左移，不赋值)
13
14 cout << (s2 >> 1) << endl; // 0001 (右移，不赋值)
15
16 cout << (~s2) << endl; // 1100 (按位取反)
17
18 cout << (s1 == s2) << endl; // false (1001==0011为false)
19
20 cout << (s1 != s2) << endl; // true (1001!=0011为true)
```

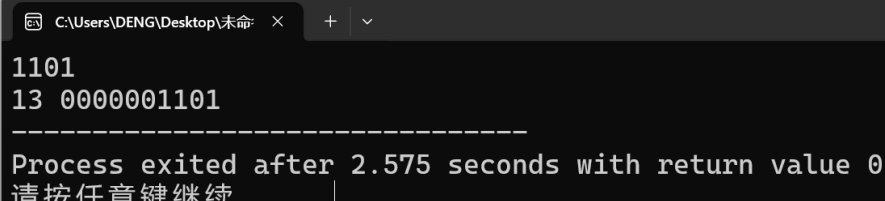
4.方法函数

1	b.any()	b中是否存在置为1的二进制位，有-返回true
2	b.none()	b中是否没有1，没有-返回true
3	b.count()	b中为1的个数
4	b.size()	b中二进制位的个数
5	b.test(pos)	测试b在pos位置是否为1，是 返回true
6	b[pos]	返回b在pos处的二进制位
7	b.set()	把b中所有位都置为1
8	b.set(pos)	把b中pos位置置为1
9	b.reset()	把b中所有位都置为0
10	b.reset(pos)	把b中pos位置置为0
11	b.flip()	把b中所有二进制位取反
12	b.flip(pos)	把b中pos位置取反
13	b.to_ulong()	用b中同样的二进制位返回一个unsigned long值
14	b.to_string()	用b中同样的二进制位返回一个string值

```
1 bitset<10> f(11);
2 cout << f.any() << "\n";
3 cout << f.none() << "\n";
4 cout << f.count() << "\n";
5 cout << f.test(0) << "\n";
6 cout << f[0] << f[1] << f[2] << f[3] << "\n";
```

这里面最常用的是b.to_ulong() 和 b.to_string()

```
1 using namespace std;
2 bitset<10> a;
3
4 int main() {
5
6     cin >> a;
7     cout << a.to_ulong() << " " << a.to_string();
8
9 }
```



```
C:\Users\DENG\Desktop\未命名 x + v
1101
13 0000001101
-----
Process exited after 2.575 seconds with return value 0
请按任意键继续
```

5.bitset优化

一般会使用bitset来优化时间复杂度，大概时间复杂度会除64或32，例如没有优化的时间复杂度为 $O(NM)$ ，使用bitset优化后复杂度可能就为 $O(\frac{NM}{64})$

bitset还有开动态空间的技巧，bitset常用在01背包优化等算法中