数据结构: 位图法

一、定义

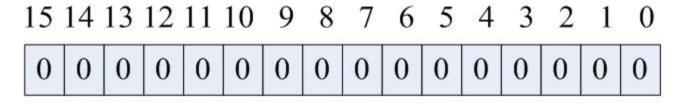
位图法就是 bitmap 的缩写。所谓 bitmap,就是用每一位来存放某种状态,适用于大规模数据,但数据状态又不是很多的情况。通常是用来判断某个数据存不存在的。在 STL 中有一个 bitset 容器,其实就是位图法,引用 bitset 介绍:

A bitset is a special container class that is designed to store bits (elements with only two possible values: 0 or 1,true or false, ...). The class is very similar to a regular array, but optimizing for space allocation: each element occupies only one bit (which is eight times less than the smallest elemental type in C++: char). Each element (each bit) can be accessed individually: for example, for a given bitset named mybitset, the expression mybitset[3] accesses its fourth bit, just like a regular array accesses its elements.

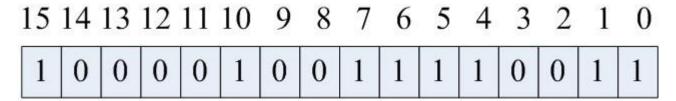
二、数据结构

unsigned int bit[N];

在这个数组里面,可以存储 N* sizeof(int) * 8 个数据,但是最大的数只能是 N* sizeof(int) * 8 – 1。假如,我们要存储的数据范围为 0-15,则我们只需要使得 N=1,这样就可以把数据存进去。如下图:



数据为【5, 1, 7, 15, 0, 4, 6, 10】,则存入这个结构中的情况为



三、相关操作

1,写入数据

定义一个数组: unsigned char bit[8 * 1024];这样做,能存 8K*8=64K 个 unsigned short 数据。bit 存放的字节位置和位位置(字节 0~8191,位 0~7)比如写 1234,字节序: 1234/8 = 154; 位序: 1234 & 0b111 = 2,那么 1234 放在 bit 的下标 154字节处,把该字节的 2 号位(0~7)置为 1

字节位置: int nBytePos =1234/8 = 154;

位位置: int nBitPos = 1234 & 7 = 2;

- 1 // 把数组的 154 字节的 2 位置为 1
- 2 unsigned short val = 1<<nBitPos;</pre>
- 3 bit[nBytePos] = bit[nBytePos] |val; // 写入 1234 得到 arrBit[154]=0b00000100 再比如写入 1236 ,

字节位置: int nBytePos =1236/8 = 154;

位位置: int nBitPos = 1236 & 7 = 4

- 1 // / 把数组的 154 字节的 4 位置为 1
- 2 val = 1<<nBitPos; arrBit[nBytePos] = arrBit[nBytePos] |val;</pre>
- 3 // 再写入 1236 得到 arrBit[154]=0b00010100 函数实现:
- 1 #define SHIFT 5
- 2 #define MAXLINE 32
- 3 #define MASK 0x1F
- 4 void setbit(int *bitmap, int i){
- 5 bitmap[i >> SHIFT] |= (1 << (i & MASK));</pre>

6 }

2,读指定位

- 1 bool getbit(int *bitmap1, int i){
- 2 return bitmap1[i >> SHIFT] & (1 << (i & MASK));</pre>

3 }

四、位图法的缺点

- 1. 可读性差
- 2. 位图存储的元素个数虽然比一般做法多,但是存储的元素大小受限于存储空间的大小。位图存储性质:存储的元素个数等于元素的最大值。比如, 1K 字节内存,能存储 8K 个值大小上限为 8K 的元素。(元素值上限为 8K ,这个局限性很大!)比如,要存储值为 65535 的数,就必须要 65535/8=8K 字节的内存。要就导致了位图法根本不适合存 unsigned int 类型的数(大约需要 2^32/8=5 亿字节的内存)。
- 3. 位图对有符号类型数据的存储,需要 2 位来表示一个有符号元素。这会让位图能存储的元素个数,元素值大小上限减半。比如 8K 字节内存空间存储 short 类型数据只能存 8K*4=32K 个,元素值大小范围为 -32K~32K。

五、位图法的应用

1、给 40 亿个不重复的 unsigned int 的整数,没排过序的,然后再给一个数,如何快速判断这个数是否在那 40 亿个数当中

首先,将这 40 亿个数字存储到 bitmap 中,然后对于给出的数,判断是否在 bitmap 中即可。

2、使用位图法判断整形数组是否存在重复

遍历数组,一个一个放入 bitmap, 并且检查其是否在 bitmap 中出现过,如果没出现放入,否则即为重复的元素。

3、使用位图法进行整形数组排序

首先遍历数组,得到数组的最大最小值,然后根据这个最大最小值来缩小 bitmap 的范围。这里需要注意对于 int 的负数,都要转化为 unsigned int 来处理,而且取位的时候,数字要减去最小值。

4、在 2.5 亿个整数中找出不重复的整数,注,内存不足以容纳这 2.5 亿个整数 参考的一个方法是:采用 2-Bitmap(每个数分配 2bit,00 表示不存在,01 表示出现一次,10 表示多次,11 无意义)。其实,这里可以使用两个普 通的 Bitmap,即第一个 Bitmap 存储的是整数是否出现,如果再次出现,则在第二个 Bitmap 中设置即可。这样的话,就可以使用简单的 1- Bitmap 了。

六、实现

001 #include <iostream>

002 #include <cstdlib>

```
003 #include <cstdio>
004 #include <cstring>
005 #include <fstream>
006 #include <string>
007 #include <vector>
008 #include <algorithm>
009 #include <iterator>
010
011 #define SHIFT 5
012 #define MAXLINE 32
013 #define MASK 0x1F
014
015 using namespace std;
016
017 // w397090770
018 // wyphao.2007@163.com
019 // 2012.11.29
020
021 void setbit(int *bitmap, int i){
      bitmap[i >> SHIFT] |= (1 << (i & MASK));
023 }
024
025 bool getbit(int *bitmap1, int i){
026
        return bitmap1[i >> SHIFT] & (1 << (i & MASK));
027 }
028
029 size_t getFileSize(ifstream &in, size_t &size){
030 in.seekg(0, ios::end);
031 size = in.tellg();
032 in.seekg(0, ios::beg);
033
     return size;
034 }
035
036 char * fillBuf(const char *filename){
```

```
037
      size t size = 0;
038
      ifstream in(filename);
039
      if(in.fail()){
         cerr<< "open " << filename << " failed!" << endl;
040
041
         exit(1);
042
      }
043
      getFileSize(in, size);
044
045
      char *buf = (char *)malloc(sizeof(char) * size + 1);
046
      if(buf == NULL){
047
         cerr << "malloc buf error!" << endl;
048
         exit(1);
      }
049
050
051
      in.read(buf, size);
052
     in.close();
053
      buf[size] = '\0';
054
      return buf;
055 }
056 void setBitMask(const char *filename, int *bit){
      char *buf, *temp;
058
      temp = buf = fillBuf(filename);
059
      char *p = new char[11];
060
      int len = 0;
061
      while(*temp){
062
         if(*temp == '\n'){}
063
           p[len] = '\0';
064
           len = 0;
065
           //cout<<p<<endl;
066
           setbit(bit, atoi(p));
067
         }else{
068
           p[len++] = *temp;
069
         }
070
         temp++;
```

```
071
      }
072
      delete buf;
073 }
074
075 void compareBit(const char *filename, int *bit, vector &result){
076
      char *buf, *temp;
077
      temp = buf = fillBuf(filename);
078
      char *p = new char[11];
079
      int len = 0;
080
      while(*temp){
081
         if(*temp == '\n'){
082
           p[len] = '\0';
083
           len = 0;
084
           if(getbit(bit, atoi(p))){
085
             result.push_back(atoi(p));
086
           }
087
         }else{
880
           p[len++] = *temp;
089
         }
090
         temp++;
091
      }
092
      delete buf;
093 }
094
095 int main(){
096
      vector result;
097
      unsigned int MAX = (unsigned int)(1 << 31);
098
        unsigned int size = MAX >> 5;
099
      int *bit1;
100
101
      bit1 = (int *)malloc(sizeof(int) * (size + 1));
102
      if(bit1 == NULL){
103
         cerr<<"Malloc bit1 error!"<<endl;
104
         exit(1);
```

```
105
      }
106
107
      memset(bit1, 0, size + 1);
      setBitMask("file1", bit1);
108
109
      compareBit("file2", bit1, result);
110
      delete bit1;
111
112
      cout<<result.size();
113
      sort(result.begin(), result.end());
      vector< int >::iterator it = unique(result.begin(), result.end());
114
115
116
      ofstream of("result");
117
      ostream iterator output(of, "\n");
118
      copy(result.begin(), it, output);
119
120
      return 0;
121 }
```

云帆教育大数据学院 www.cloudyhadoop.com

通过最新实战课程,系统学习 hadoop2.x 开发技能,在云帆教育,课程源于企业真实需求,最有实战价值,成为正式会员,可无限制在线学习全部教程;培训市场这么乱,云帆大数据值得你选择!! 详情请加入 QQ 群: 374152400,咨询课程顾问!



实时在线授课,一线研发技术 www.yfteach.com

关注云帆教育微信公众号 yfteach,第一时间获取公开课信息。