大数据算法基础 实验1

张飞

2015210887

1. 问题描述

输入：1亿个自然数（数之间可能有重复），并且无序排列。
输出：第一个没有在这些数中出现的自然数。
实例：输入`{2,4,5,1,10,8,9,3,4,0,11,6,13,14}`，输出`7`

2. 实现思路

输入规模：假设1亿个数全部为`int`类型，则总占用空间约为：100,000,000*4 B 约为381.47 MB，对内存压力较大。
基本思路：考虑采用位图标记为1的方式记录是否出现该数。然后再从0开始遍历自然数，直至找到位图中对应位为0的数。此时占用内存为：100,000,000b 约为 11.92MB，节省空间约为96%。
优化方案：

- 当1亿位数中有包含大于1亿的自然数，直接舍去。因为最终结果不会存在大于1亿的数，最差情况这1亿个数分解存前1亿个数各一次。
- 利用一个min值记录连续出现自然数的最大值，在查找过程中，直接从这个下界开始。例如在一个序列中`{0,..,1,..,2,..,3,..}`，则我们直接从4开始查找。
- 在进行查找过程中，不采用一位一位查找，而是先异或多字节。例如先异或0xff，如果结果为1，再在该字节内查找。

3. 程序实现

3.1 配置变量

程序实现两个变量控制随机数生成，如下：

```
#ifndef _CONFIG_H
#define _CONFIG_H
/*
How many numbers to generate.
*/
#define NUMBERS_COUNT 100000000

/*
The maximum of random numbers.
Since the unsigned int range at 0~4,294,967,295,
the max config vlaue is 4,294,967,295
*/
#define MAX_NUMBER 100000000

#endif
```

NUMBERS_COUNT代表生成多少随机数，MAX_NUMBER代表生成随机数最大值

3.2 随机数生成算法

采用了C++中random库中uniform_int_distribution的模板类进行生成，并写入二进制文件”data.in”，如下：

```
#include<iostream>
#include<fstream>
#include<random>
#include<ctime>

#include "config.h"

using namespace std;

int main(){

    //open the file
    ofstream fout("data.in",ios::out|ios::binary);

    //generate the engine by time
    default_random_engine dre(time(NULL));

    uniform_int_distribution<unsigned int> uid(0,MAX_NUMBER);

    for(int i=0;i<NUMBERS_COUNT;i++){
        fout<<uid(dre)<<endl;
    }

    fout.close();

    return 0;
}
```

3.3 寻找最小自然数算法

算法实现步骤如下：

1. 判断NUMBERS_COUNT和MAX_NUMBER的大小关系。选择较小的一个作为bitmap中位的长度，并初始化bitmap所有位为0。

```
unsigned int MAX_RESULT;
if(NUMBERS_COUNT < MAX_NUMBER){
    MAX_RESULT = NUMBERS_COUNT;
}else{
    MAX_RESULT = MAX_NUMBER;
}
unsigned char* bitmap;
unsigned int bytes_length = MAX_RESULT/BYTE_LENGTH + 1;
bitmap =(unsigned char*) malloc(bytes_length);
memset(bitmap,0,bytes_length);
```

2. 读取输入文件中的每一个数，然后将bitmap中对应位置为1。

```
ifstream fin("data.in",ios::in|ios::binary);
unsigned int t;
unsigned int i;
unsigned int min = 0;
for(i=0;i<NUMBERS_COUNT;i++){
    fin>>t;
    if(t>=MAX_RESULT) continue;
    bitmap[t/BYTE_LENGTH] = bitmap[t/BYTE_LENGTH] |
(1<<(t%BYTE_LENGTH));
    if(t==min) min++;
}
```

3. 从第min个自然数开始判断是否出现在bitmap中，如果没有出现，则返回该值。

```
for(i=min/BYTE_LENGTH;i<bytes_length;i++){
    if(bitmap[i] ^ 0xff){
        int j;
        for(j=0;j<BYTE_LENGTH;j++){
            if(!(bitmap[i] & (1<<j))) break;
        }
        min = i*BYTE_LENGTH+j;
        break;
    }else{
        continue;
    }
}
cout<<min<<endl;
delete(bitmap);
fin.close();
```

4 算法测试

Test1: 当数据规模NUMBERS_COUNT=10,000，随机数中最大自然数MAX_NUMBER=10,000时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 0
The tiem of this program is 11ms
```

Test2：当NUMBERS_COUNT=100,000，MAX_NUMBER=100,000时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 3
The tiem of this program is 35ms
```

Test3：当NUMBERS_COUNT=1,000,000，MAX_NUMBER=1,000,000时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 5
The tiem of this program is 376ms
```

Test4：当NUMBERS_COUNT=1,000,000，MAX_NUMBER=10,000,000时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 0
The tiem of this program is 359ms
```

Test5：当NUMBERS_COUNT=10,000,000，MAX_NUMBER=10,000,000时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 0
The tiem of this program is 4058ms
```

Test5：当NUMBERS_COUNT=10,000,000，MAX_NUMBER=10,000,000时，且前一个10,000,000个自然数每个出现一次时，测试结果为：

```
C:\zf\code\homework\FindMinNumber>a.exe
The min number is 10000000
The tiem of this program is 3634ms
```

5 算法复杂度分析

假设输入文件随机数的个数为n，则
算法将顺序读取一遍输入文件，并建立bitmap，时间复杂度为：$O(n)$
并从min开始顺序验证是否在bitmap中，至多到n，时间复杂度为：$O(n)$
因此总的时间复杂度为：$O(n)$

参考链接：
C++ chrono
C++ uniform_int_distribution