《算法竞赛专题解析》附 A

测试数据的构造与对拍

华东理工大学 罗勇军 2020.6

文章链接：https://blog.csdn.net/weixin_43914593/article/details/106863166
本系列文章将于 2021 年整理出版，书名《算法竞赛专题解析》。
如有建议，请联系 QQ 群 567554289

目录

队员在 OJ 上做题时，提交到 OJ 的代码，OJ 是如何判断是否正确的？

OJ 并不看代码，而是使用“黑盒测试”，用测试数据来验证。每一题，OJ 都事先设定好很多组输入数据 datain 和输出数据 dataout，参赛队员提交代码后，OJ 执行代码，输入 datain，产生输出，看是否和 dataout 完全一致。

OJ 所设定的测试数据，可能非常多而且复杂，队员不可能知道，更不能手工一个个进行验证，这时要自己构造测试数据，并且用两种代码进行对比测试。包括以下两个任务：

（1）构造数据。包括输入和输出数据。如果输入情况复杂，不能方便手工构造，而要编一个小程序，随机生成。输出数据，则要通过下面的低效率代码来生成。

（2）两种代码。用两种算法写代码进行对比：低效率代码、高效率代码。被称为“对拍”。

低效率代码。一般用暴力法实现，代码简单、逻辑正确，它因为效率低无法通过 OJ 的时间限制，它的用途是做为基准（benchmark）代码，产生正确的输出数据。

高效率代码，准备提交到 OJ。把它产生的输出数据，与低效率代码产生的输出数据进行对比，如果一致，就认为是正确的，可以提交了；如果不一致，可以找到那些不一致的数据，用来修改程序。测试数据非常重要，往往能帮助编程者找到代码的错误所在。

A.0 随机数基础

构造输入数据，需要用到随机数。c 语言中与随机数相关的函数和变量有：

srand(seed)：产生以 seed 为种子的随机数列。对一个 seed 种子，产生的随机数列是固定的，所以是“伪随机”的。要产生“真”随机数列，就需要一个“随机”的 seed，一般用当前时间做 seed。

time(0)：返回当前时间，即从 1970.1.1 日零时到现在的秒数。

rand()：返回[0, RAND_MAX]内的一个随机正整数。

RAND_MAX：一般有 RAND_MAX = 32767。

A.1 构造负数和实数

需要[-n, n]内的随机数时，先产生一个[0, 2n]的正随机数，然后再减去 n。

需要实数时，先产生一个大的随机正整数，然后除以 10 的幂。

A.2 构造极大范围内的随机数

1. 例：生成[0, 1000000]内的随机数

rand() 只能生成大小在[0, RAND_MAX]内的随机数，RAND_MAX = 32767。如何用 rand()生成大随机数？有 2 个方法：

（1） rand()*rand()

这方法很简单，但是缺点是生成的几乎全是合数，很少有素数。

（2）用组合的方法，把 3 个随机数组合起来，得到一个 24 位的随机数。

```
unsigned long ulrand(){      //生成一个大随机数
    return (
             (((unsigned long)rand()<<24)& 0xFF000000ul)
            |(((unsigned long)rand()<<12)& 0x00FFF000ul)
            |(((unsigned long)rand())    & 0x00000FFFul));
}
```

2. 生成 [MIN, MAX]之间的一个随机数

```
const int MAX = 1000000;     //自己定义 MAX 和 MIN
const int MIN = 0;
int big_rand;                //大随机数
big_rand = ulrand() % (MAX - MIN + 1) + MIN;
```

A.3 去掉重复的随机数

上面生成的随机数，很多是重复的。如何去掉 100 万个数中的重复数字？而且要求去重后不改变剩下数据的顺序。

因为数太多，所以不可能用暴力法一个个对比。用 unique()函数去重则会改变顺序。

简单又快的方法是用 hash 去重：

```
int myhash[MAX];        //hash 表
int mynum[MAX];         //记录去重后的随机数
```

下面处理随机数 big_rand，把重复的去掉。

```
if(myhash[big_rand] == 0) {  //第 big_rand 位置还没有
     myhash[big_rand]=1;     //随机数 big_rand ，登记在 myhash[]的第 big_rand 个位置
     mynum[num]= big_rand;   //记录随机数
     num++;                  //记录随机数的数量
}
```

A.4 例题

用下面的例题演示如何构造大量的不重复的测试数据。

hdu 1425 sort

给你 n 个整数，请按从大到小的顺序输出其中前 m 大的数。

输入：每组测试数据有两行，第一行有两个数 n, m(0 < n, m < 1000000)，第二行包含 n 个各不相同，且都处于区间[-500000, 500000]的整数。

输出：对每组测试数据按从大到小的顺序输出前 m 大的数。

Hdu 1425 的测试数据构造代码：

```
#include <bits/stdc++.h>
using namespace std;
const int MAX  = 1000000;       //100 万个数
const int MIN  = 0;
int myhash[MAX]={0};            //用 hash 查重
int mynum[MAX];                 //记录去重后的随机数

unsigned long ulrand(){         //生成一个大随机数
    return (
      (((unsigned long)rand()<<24)& 0xFF000000ul)
     |(((unsigned long)rand()<<12)& 0x00FFF000ul)
     |(((unsigned long)rand())    & 0x00000FFFul));
}
int main(){
    int big_rand;                       //大随机数
//printf("system: RAND_MAX= %d\n", RAND_MAX);//查看系统随机数的最大值，一般是 32767
    srand(time(0));                     //用当前时间做随机数种子
//本题需要不同的随机数，用 hash 的方法来去重
     int num = 0;                       //去重后随机数的数量
     for(int i=0; i< MAX; i++){         //hash
        big_rand = ulrand() % (MAX-MIN + 1) + MIN;  //[MIN,MAX]之间的一个随机数
```

```
        if(myhash[big_rand] == 0){   //用 hash 查重
           myhash[big_rand]=1;       //数字 big_rand，登记在 big_rand 这个位置
           mynum[num]= big_rand;     //记录随机数
           num++;                    //最后产生 num-1 个随机数
        }
     }
     big_rand = ulrand() % (MAX-MIN + 1) + MIN;
     printf("%d %d\n",num, big_rand %(num +1));  //case 的第一行

     int RANGE= 500000;                //题目要求[-500000,500000]内的数字
     for(int i=0; i<= num-1; i++){
        if(i< num-1)   printf("%d ", mynum[i]-RANGE);  //有正有负
        if(i == num-1) printf("%d\n", mynum[i]-RANGE); //最后一个，末尾没有空格
     }
    return 0;
}
```

上面的代码在一次运行后，从生成的 100 万个随机数中，得到约 63 万个不重复的数。

设代码的文件名是 hdu1425，执行代码，输出到文件 data.in：

```
D:\>hdu4585 >data.in
```

A.5 对拍

把程序提交到 OJ 之前，要先检验程序的正确性。前面已经获得了输入测试数据，那么输出数据如何产生?此时需要尽快写一个代码，它的逻辑和代码很简单，但是保证结果是正确的。

1、对拍的多个代码

下面以 Hdu 1425 为例，给出 3 个实现：暴力的冒泡代码、sort 排序、hash 算法。

（1）冒泡排序

冒泡算法很容易写，但是它的时间复杂度是 O(n2)，如果排序 100 万个数，时间长达数小时，不能作为 benchmark。

不过，由于冒泡排序的代码非常简单，在赛场能快速编码，那么可以用小规模的输入测试数据，例如对 10 万个数排序，运行时间就能接受了。

（2）sort 排序

下面代码也能通过 OJ 测试，不过这里把它看成 benchmark，用来验证后面 hash 算法的正确性。

```
#include<bits/stdc++.h>
using namespace std;
int a[1000001];                          //记录数字
#define swap(a, b) {int temp = a; a = b; b = temp;}    //交换
int n, m;
int main(){
```

```
    while(~scanf("%d%d", &n, &m)){
        for(int i=1; i<=n; i++)  scanf("%d", &a[i]);   //scanf 比 cin 快
        sort(a + 1, a + n + 1);
        for(int i = n; i >= n-m+1; i--){               //打印前 m 大的数，反序打印
            if(i == n-m+1)  printf("%d\n", a[i]);
            else            printf("%d ",  a[i]);
        }
    }
    return 0;
}
```

设代码的文件名是 sort，执行代码，读取输入 data.in，输出到文件 sort.out

```
D:\>sort <data.in >sort.out
```

（3）hash 算法

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1000001;
int a[MAXN];
int main(){
    int n,m;
    while(~scanf("%d%d", &n, &m)){
        memset(a, 0, sizeof(a));
        for(int i=0; i<n; i++){
            int t;
            scanf("%d", &t);     //此题数据多，如果用很慢的 cin 输入，肯定 TLE
            a[500000+t]=1;       //数字 t，登记在 500000+t 这个位置
        }
        for(int i=MAXN; m>0; i--)
            if(a[i]){
                if(m>1)  printf("%d ", i-500000);
                else      printf("%d\n", i-500000);
                m--;
            }
    }
    return 0;
}
```

设代码的文件名是 hash，执行代码，读取输入 data.in，输出到文件 hash.out

```
D:\>hash <data.in >hash.out
```

2、比较 2 个代码的输出是否一样

Windows：fc 命令

```
D:\>fc sort.out hash.out /n
```

Linux：diff 命令

```
[root]#diff -c hash.out sort.out
```

A.6 统计执行时间

代码运行的时间是多少，能通过 OJ 的时间限制吗？一般情况下，通过分析算法的复杂度就足够估算运行时间了。如果确实想知道运行时间，可以用下面的代码计时（windows 环境）。不过，个人电脑和 OJ 的服务器速度并不同，而且还有输入输出的时间，所以下面的代码只能作为参考。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    clock_t start, end;
    start = clock();                                    //开始时间
    system("d:\\hash <d:\\data.in >d:\\data.out");      //运行 hash 代码，注意目录
    end = clock();                                      //结束时间
    cout << (double)(end - start) / CLOCKS_PER_SEC << endl;   //输出时间
}
```