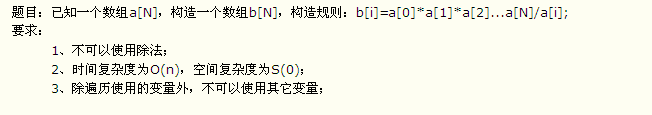
刚进博客园就在《[一道面试附加题的另类求解](http://www.cnblogs.com/grenet/archive/2012/04/07/2436384.html)》看到一道有趣的题，正好，偶对这题也有一些想法，因此写来分享下。题目如下：



先来看第一个条件，不可用除法。要满足这个条件倒是很简单：

令forward[i] = a[0] \* a[1] \*... \* a[i];

   backward[i] = a[i] \* a[i+1] \*...\* a[N];

那么b[i] = forward[i-1] \* backward[i+1];

在此基础上，我们再来看一下第2个条件，时间复杂度为0(N)，空间复杂度为O(1)。

先将forward和backward数组的值算出，然后才算b数组的值虽然可以满足时间的要求，但空间要求却满足不了。因此有必要减少空间的使用量。仔细地观察forward数组，就会发现这个数组可以用一个变量代替。而backward数组的每个元素只使用一次，并且是按顺序使用的。那么如果将backward数组的值放到b组中，不就可以解决这个问题了。具体代码如下：

复制代码

int Zero(int a[], int b[], int N) {  
 int i;  
  
 b[N - 1] = a [N - 1];  
 for(i = N - 2; i >= 0; --i) { //construct backward array in b array  
 b[i] = b[i + 1] \* a[i];  
 }  
  
 int forward = 1;  
 for(i = 0; i < N - 1; ++i) {   
 b[i] = forward \* b[i + 1];  
 forward \*= a[i];  
 }  
 b[N - 1] = forward;  
}

复制代码

现在只剩下第三个条件——只能使用遍历变量的限制了。上面的方法只要去掉forward这个局部变量就满足全部条件。在思考怎样去掉之前，不防回顾前面是怎样去掉backward数组的。没错，利用了b数组暂时没用到的性质。那么去掉forward的方法也可以如法炮制。

复制代码

void Zero(int a[], int b[], int N) {  
 int i;  
  
 b[N - 1] = a [N - 1];  
 for(i = N - 2; i >= 0; --i) { //construct backward array in b array  
 b[i] = b[i + 1] \* a[i];  
 }  
  
 b[N - 1] = 1; //use as forward  
 for(i = 0; i < N - 2; ++i) {   
 b[i] = b[N - 1] \* b[i + 1];  
 b[N - 1] \*= a[i];  
 }  
 b[N - 2] = b[N - 1] \* a[N - 1];  
 b[N - 1] = b[N - 1] \* a[N - 2];  
}

复制代码

一点思考：

      就通常来说，编程是主张一物一用，一个函数最好只完成一个功能，即一个物体不要同时用于多个用途。因为这样的代码易于阅读和维护。但是却有不能这样做的时候，例如本题，例如在一台普通PC上处理大数据。在这样的时候，只能打破这些约束了。因为如果功能都完成不了，还谈什么阅读与维护。