[ACM] 经验总结（一） - CSDN博客

1> I/O效率问题

一篇文章中指出——getchar()/putchar()快于scanf/printf快于cin/cout

有些题目数据可能非常多，但是对数据的操作可能却相对简单，这时候不要因为I/O导致TLE，我想，几乎所有人都遇到过cin/cout换scanf/printf才能过的情形。

另外就是对于整型，可能手动解析是最快的（我还没试验，不能下定论）。因为scanf/sscanf/fscanf都是支持多种格式，所以实现上不可避免要各种判断。

最近做的一道题就是这样，用cin/cout超时，用scanf/printf大约0.7s，用自己手动解析的方式0.14s。

题目大致是先输入n（n<10^6），后面跟n行数据，每行都是整数。然后输出行数不定（和输入有关），也可以认为是和n同一个数量级。我最后的写法是：

[cpp]view plaincopy

```
1. // 这是手动解析int和输出int
2. #define  PARSE_INT( x ) do{ \
3.                 x = 0; \
4. int  sgn = 1; \
5. while( *ptr < '0' || *ptr > '9' ) { if( *ptr == '-' ) sgn = -1; ++ptr; } \
6. while( *ptr >= '0' && *ptr <= '9' ) { x = x*10 + *ptr - '0'; ++ptr; }  \
7.                 x = x * sgn; \
8.         }while(0)
9. #define  OUTPUT( x ) do { \
10. int  x0 = x; \
11. char  num[32], *digit_pos = num + 31; \
12.                 num[31] = '\n'; \
13. while( x0 > 0 ) { \
14.                         *--digit_pos = x0 % 10 + '0'; \
15.                         x0 /= 10; \
16.                 } \
17. int  sz = num + 32 - digit_pos; \
18.                 memcpy( optr, digit_pos, sz ); \
19.                 optr += sz; \
20.         }while(0)
21. // 这是输入输出缓冲区，确保足够大
22. char  buff[MAXN*12];
23. char  obuff[MAXN*12];
24. ……
25. // 一口气将所有输入全部读入缓冲区
26. n = fread( buff, 1, sizeof( buff ), stdin );
27. PARSE_INT( n );
28. while( n-- ) {
29.     PARSE_INT( x );
30.     .....
31.     OUTPUT( y );
32. }
33. // 一口气将所有输出
34. fwrite( obuff, 1, optr - obuff, stdout );
```

2> two pointers的实现

何谓two pointers？很多题目都是这个解题模式——先对数据排序，然后用两个指针遍历一遍数组，两个指针随着一定的限制条件向后或向前跳跃。

例如最近做的一道题，一个升序数组，当两个数之间的差恰好小于某个阈值，则进行一些计算，这时应该是两个指针依次向后走，他们之间的间距时大时小。

我的一个实现是：

[cpp]view plaincopy

```
1. for( int i = 0, j = 1; i < n; ++i ) {
2. while( j < n && a[j] - a[i] <= threshold ) ++j;
3.     do_sth( i, j - 1 );
4. }
```

然后看到另一个人的实现是：

[cpp]view plaincopy

```cpp
for( int i = 0, j = 1; j < n; ++j ) {
while( a[j] - a[i] > threshold ) ++i;
    do_sth( i, j );
}
```

这两种实现对比一下，不难发现，我之前的实现是从前往后推，很有可能会推出界，所以要多个判断，而后面的实现是从后往前拉，有后面的指针作为天然的哨兵，不用担心越界。