常见编程技巧

内存池

当我们需要动态分配内存的时候，频繁使用 **new/malloc** 会占用大量的时间和空间，甚至生成大量的内存碎片从而降低程序的性能，可能会使原本正确的程序 **TLE/MLE**。

这时候我们就需要使用到「内存池」这种技巧：在真正使用内存之前，先申请分配一定大小的内存作为备用，当需要动态分配时则直接从备用内存中分配一块即可。

当然在大多数 **OI** 题当中，我们可以预先算出需要使用到的最大内存并一次性申请分配。

如申请动态分配**32**位有符号整数数组的代码：

```
inline int* newarr(int sz) {
  static int pool[maxn], *allocp = pool;
  return allocp += sz, allocp - sz;
}
```

线段树动态开点的代码：

```
inline Node* newnode() {
  static Node pool[maxn << 1], *allocp = pool - 1;
  return ++allocp;
}
```

对拍

有的时候我们写了一份代码，但是不知道它是不是正确的。这时候就可以用对拍的方法来进行检验或调试。

什么是对拍呢？具体而言，就是通过对比两个程序的输出来检验程序的正确性。你可以将自己程序的输出与其他程序（打的暴力或者其他 dalao 的标程）的输出进行对比，从而判断自己的程序是否正确。

当然，对拍过程要多次进行，我们需要通过批处理的方法来实现对拍的自动化。

具体而言，我们需要一个数据生成器，两个要进行对拍的程序。

每次运行一次 数据生成器，将生成的数据写入输入文件，通过重定向的方法使两个程序读入数据，并将输出写入指定文件，利用 **Windows** 下的 **fc** 命令比对文件（**Linux** 下为 **diff** 命令），从而检验程序的正确性。

如果发现程序出错，可以直接利用刚刚生成的数据进行调试啦。

对拍程序的大致框架如下：

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  // For Windows
  //对拍时不开文件输入输出
  //当然，这段程序也可以改写成批处理的形式
```

```
  while (1) {
    system("gen > test.in");  //数据生成器将生成数据写入输入文件
    system("test1.exe < test.in > a.out");  //获取程序1输出
    system("test2.exe < test.in > b.out");  //获取程序2输出
    if (system("fc a.out b.out")) {
      //该行语句比对输入输出
      // fc返回0时表示输出一致，否则表示有不同处
      system("pause");  //方便查看不同处
      return 0;
      //该输入数据已经存放在test.in文件中，可以直接利用进行调试
    }
  }
}
```