数据与算法小班辅导

2020年10月11日 饶淙元

0. 约定

参加本节小班的前提条件是你已经阅读了 OJ 题目《军训排队问题》与《numpy二维数组计算》，并进行了相应思考。如果你没有遵从这一约定，那么本小班为你《数据与算法》课程学习过程中带来的负面影响由你本人担责。

1. OJ 技巧

1.1 OJ 基本认识

1. AC、RE、WA、MLE、TLE、CE
2. 输入与输出是两个互不干扰的流
3. cin 可能 TLE，除非开了 `std::ios::sync_with_stdio(false);`
4. 如果真的开了 3 中的东西，不能再用 `<stdio.h>` 的函数，包括 `getchar`
5. 优先考虑 `scanf`， 因为 `<iostream>` 占内存更多
6. 要想比 `scanf` 更快，考虑用 `fgets`
7. 如非必要，勿用 `new`
8. 内存做最坏情况的计算，结构体估算时不要忘了内存对齐
9. 知道复杂度后，还需要代入数据算算具体循环量，并熟知 CPU 主频量级
10. OJ 读取采用了重定向，会读到 EOF，有时候可以用来做输入结束的判定，自己尝试时可以同样重定向或者新起一行输入 ctrl+Z（windows）
11. 自己验证时灵活运用重定向可以方便debug，包括命令行法或者 freopen（自学）
12. 尽量使用 STL 而不是自己手写一些基本结构
13. 某些STL cover 不了的需求确实要手写（如 `std::list` 很少在 OJ 中被使用，后面会解释原因）

1.2 神秘技巧

1.2.1 快读

逐行单整数快读（`fgets` 版本）：

```
int quickRead()
{
    static char s[16];
    fgets(s, 16, stdin);
    int result = 0;
    char* p = s;
    while ('0' <= *p && *p <= '9')
    {
        result *= 10;
        result += *p - '0';
```

```
        p++;
    }
    return result;
}
```

手动解析字符串以获取更快的读取速度。（去年第一题《寻找丢失的筷子》用它可以用 $O(nlogn)$ 的复杂度完成，否则只能 $O(n)$）

1.2.2 assert

正常 OJ 都会给出数据范围，但是电子系数算题不给范围是传统艺能。这一现象从八字班开始得以改善，但也只是逐步改善，总有没给范围的（如《军训排队问题》），为此我们有类似以下的办法：

```
#include<stdio.h>
#include <assert.h>
int main()
{
    int n, m, k, l;
    scanf("%d%d%d%d", &n, &m, &k, &l);
    assert(n + m + k + l <= 100000);
    return 0;
}
```

可以得知实际数据范围。

assert: 如果传进去的参数为 true 则什么都不会发生，为 false 则会 Bad Syscall。

2. 题解

2.1 军训排队问题

维护一个结点有编号的链表，支持插入、删除、全链逆序、两结点交换操作。

结点至多 200000 个，操作总数未给范围（数算传统艺能）。

2.1.1 数据结构

首先考虑数据结构的使用，这无疑是一道链表题，且是一道双向链表题。但是由于要支持逆序，因此普通的双向链表还不够，考虑如下两种结构：

1. 带头尾哨兵结点的双向链表（关于哨兵见 https://www.xuetangx.com/learn/THU08091000384/THU08091000384/4231547/video/6326487）
2. 使用带头哨兵结点的循环链表

2.1.2 算法

容易看出必须要各个操作都 $O(1)$ 完成，否则 TLE 的结局几乎是必然的。

插入删除交换都很容易做到 $O(1)$（注：交换有几种情况需要考虑？），逆序如何实现 $O(1)$？

对双向链表，交换头尾结点语义，交换左右语义。对循环链表，直接交换左右即可。

2.1.3 实现

最原始的想法，实现一个传统链表结点，带有数据域存储编号。为了 $O(1)$ 访问到显然需要用数组下标访存。（以双向链表为例，需要 200002 个结点，循环链表只需要 200001）

```
struct Node {
    Node * last;
    Node * next;
    int id;
}nodes[200002];
```

其中 `nodes[i].id = i`。但这样会 MLE（自行验证）。

压缩内存，将 8 字节指针改为 4 字节数组下标，使用 Array of Struct：

```
struct Node {
    int last;
    int next;
    int id;
}nodes[200002];
```

进一步优化内存，可以用 Struct of Array：

```
struct Nodes {
    int last[200002];
    int next[200002];
    int id[200002];
}nodes;
```

进一步的， Struct 都省了，直接有三个数组足够了

```
int last[200002], next[200002], id[200002];
```

截止这一步，我们真的需要 id 吗？其实 `id[i] = i`， 已经没必要存在了。

```
int last[200002], next[200002];
```

两个数组足以完成任务。

反过来，我们能否省略掉 id 而使用传统链表完成任务呢？估算内存发现没问题，实际上自然也是可以操作的：

```
struct Node {
    Node * last;
    Node * next;
}nodes[200002];
```

这种情况下对应一个指针 `Node * p`，它的编号是 `p - nodes` （黑魔法？C语言基础）

2.2 numpy二维数组计算

2.2.1 算法

问题可以转化为中缀表达式求值~~或者编写编译器~~。

1. 遍历输入字符串，准备操作符栈与操作数栈，初始操作符栈准备标识符，初始操作数栈为空

2. 对于操作数压栈，对于操作符判断新操作符与栈顶操作符优先级来决定运算、抵消还是压栈
3. 待到操作符栈空后结束循环，此时操作数栈应该只剩下一个操作数，即运算结果

2.2.2 实现

各种 numpy 相关函数可以当做一个操作符，参数当做操作数。简化起见可以对输入串先做如下预处理：

1. 补全缺省参数，包括 axis 以及 np.ones 的维度（如 `np.ones(3)` 改为 `np.ones((3,1))`）
2. 将函数与左括号改成一个字符，压栈时先压左括号，再压函数字符（相当于新定义一个操作符，如 `np.ones(` 改为 `o`，`np.concatenate(` 改为 `c`）
3. 将多余的信息去掉，参数改为逗号隔开的操作数，可以简化操作数格式（如 `o(3,1))` 改为 `o3,1)`，`c[o2,2)+1,o2,1)],axis=1)` 改为 `co2,2)+1,o2,1),1))`）

预处理后可以得到比较规整的形式，之后完成简单的中缀表达式操作。

2.2.3 简化操作

上述操作是一个较优的中缀表达式实现，但实际上由于该题数据量较小，也可以直接用递归替代栈简化完成（我没试过，确实可行）

3. 课程反馈