算法复杂度与程序调试-郑得贤

算法复杂度与程序调试

课程概述

程序设计思维与实践注重于实际问题的实践，需要对具体问题去思考数据结构的应用和算法的设计，并要求实现完整的代码。

后续相关课程有图论算法的实践、算法设计与分析。特别相关的是下学期需要参加CCF CSP认证（1学分）

CCF CSP认证

CSP认证是CCF面向全国开展的一个大学生软件职业能力认证项目。主要考察的就是如何对具体问题进行程序设计。每年有多次认证机会。每次时间4小时，共有5道题。

T1：较为简单的程序设计题目。需要掌握具体的语法。

T2：难度增加，只掌握语法可能有30~70分。需要掌握一些优化方法。

T3：较为复杂的模拟题。考察对题目的理解以及程序设计中的细节

T4：考察对数据结构、算法的具体应用。

T5：考察对数据结构、算法的具体应用。难度较大

T5相关的知识点会在课堂中讲述，但课程主要专注与前4题的有效得分。

课程由四位讲师混合讲课，周二的第五大节安排了实验课，周三的第五大节安排的理论课。每次上课后可以对知识点、作业难度进行反馈，会根据反馈调整授课大纲。

上课方式

每周2小时理论课，2小时上机实验。每节理论课后会布置与课堂内容相关的作业题。

实验课每4周为一轮。

第i周：一场限时的CSP-T3题型比赛，同时布置一道T3练习（练习在4周内完成即可）

第$i+1$周：完成作业与CSP-T3练习

第$i+2$周：一场CSP赛制的模拟考试（题型为CSP-T1、T2、T4）

第$i+3$周：完成作业与CSP-T3练习

编程语言

C / C++ / Java / Python，课程讲述C++的应用

提供IDE：VS-CODE / Eclipse / IDLE

验收形式

根据每周作业和实验的内容在实验报告中给出题目（考察算法复杂度，程序设计思路和实现细节）。完成题目即可。报告需要在下次上课前提交。

成绩组成

$40%$ 期末考试$+20%$CSP模测$+40%$平时分（考勤+实验）

T3练习、作业、实验报告按时完成即可获得全部分数。超时则只能获得部分分数。

CSP模测与T3测试场上分数即为最终分数。

相关名词

OJ：在线评测系统，编译并运行用户程序代码，用于比较程序输出与正确答案，并返回结果。

VJ：虚拟评测系统。

评测结果

AC (Accepted)，程序输出了正确的结果，通过了测试。

WA(Wrong Answer),程序输出的结果是错误的。

PE(Presentation Error)，输出格式有误，较少出现。

TLE(Time Limit Exceeded)，运行超时。可能出现了死循环或者时间复杂度不对。

MLE(Memory)，内存超限。数组开的过大。

RE(Runtime Error)，运行时错误，可能是除零，数组访问越界、访问空指针或栈溢出等

OLE(Output Limit Exceeded)，输出超限

CE(Compile Error)，编译错误，本地编译就没有通过或者提交OJ时选择了错误的语言

OI赛制：以选手最后一次提交的程序为准，在赛后同一评测。

IOI赛制：可以多次提交，实时评测，可以看到自己与别人的成绩以及排行榜。

CSP赛制：与IOI赛制类似，但只能看到自己的成绩。

复杂度分析

好的算法需要好的效率，算法效率通常使用时间复杂度和空间复杂度来度量。

在算法竞赛中，代码运行的时间不能超过题目给出的时间限制，代码使用的空间不能超过题目的空间限制。

时间复杂度

对于规模为n的问题，代码中语句执行次数T(n)是关于问题规模n的函数。通过分析T(n)随n的变化确定T(n)的数量级。则时间复杂度可以记作T(n) = O(f(n))

具体求解步骤

找出执行次数最多的语句

计算执行次数的数量级，只保留最高次幂，并忽略系数

放入O()来表示

常见的时间复杂度有O(1)、O(logN)、O(N)、O(NlogN)、O(N^2)

算法竞赛中，log都是以2为底的对数

时限与时间复杂度的关系

可以假定机器速度是每秒10^8次基本运算

在设计算法时，需要考虑算法的时间复杂度是否能满足题目的时限

通过时限与数据范围来考虑使用哪一种算法

空间复杂度

程序空间复杂度指运行完一个程序所需要的空间大小的估计。

除了定义的变量、数组等，存储程序、程序运行需要的指令也会影响空间复杂度。

只能通过定义的变量、数组等对空间复杂度进行估计。

例：

对于int a[10000000]，数组的每个元素大小为4字节，所需空间为$4*10000000/1024/1024$约等于39MB

部分分

题目的时限是固定的，但通常会以分段的形式给出测试点的规模。

当无法实现通过规模最大的测试点的代码，或者考试快结束，来不及完成全部的代码。

可以退而求其次，考虑规模较小的数据点。

通常测试点的规模越小，实现的代码越简单，花费时间越少。

程序的调试

简单文件操作

C++

```cpp
freopen("a.in", "r", stdin);//打开文件a.in并从中读取数据
freopen("a.out", "w", stdout);//打开文件a.out并向其输出数据

close(stdin);//关闭读取的文件
close(stdout);//关闭输出的文件
//关闭文件可以不写
```

当题目的输入数据较大时，使用文件读写可以避免频繁手动输入数据，也能清晰地看到输出结果的格式

提交OJ时记得去掉文件操作

VS Code的调试

调试可以一步步执行代码，查看每一行代码发挥的作用，输出的答案是如何形成的。

实践中，调试代码是快速找到错误的好方法。

1

C++

```cpp
#include<iostream>
#include<cstdio>

using namespace std;

int a, b;

int main()  {
    int c, d;
    cin >> a >> b;
    c = a;
    d = b;
    b = a + b;
    cout << c + d;
    return 0;
}
```

调试时，变量栏只有当前函数内的局部变量，要查看全局变量需要在监视栏内添加

2

C++

```cpp
#include<iostream>
#include<cstdio>

using namespace std;

void add(int& d)  {
    d++;
}

int a, b;

int main()  {
    int c, d;
    cin >> a >> b;
    c = a;
    for (int i = 1;i <= 10; i++)  {
        d = c / 2;
        add(d);
        c += i;
    }
    cout << c << endl;
    return 0;
}
```

单步跳过：直接执行完该行代码，不进入子函数

单步进入：若代码中有子函数则进入

单步跳出：直接执行完该子函数，返回父函数

调试过程中可以随时改变断点位置来控制代码执行进度

C++输入与输出

在比赛中，使用标准输入输出，读入题目输出，将程序输出结果与答案对比，来判断程序的正确性

使用库cstdio，需要学习的有

格式化输入输出到标准IO（掌握）

格式化输入输出到字符串（了解）

单字符输入输出（了解）

快速读入、缓冲区冲刷（了解）

格式化输入输出到标准IO

输入函数 int scanf(const char * format, ...);

格式scanf(输入控制符，输入参数)

scanf将从键盘输入的字符转化为输入控制符所规定的格式，然后存入以输入参数的值为地址的变量中

输出函数 int printf(const char * format, ..);

格式printf(输出控制符与非输出控制符，输出参数)

printf将输出参数转化成输出控制符所规定的格式输出到屏幕上，将非输出控制符原样输出至屏幕

格式化输入输出到字符串（了解）

sscanf、sprintf和scanf、printf使用一致，但是将输入输出从标准输入输出变成了字符串

单字符输入输出（了解）

输入一个字符getchar()

输出一个字符putchar()

可以优化效率

快速输入输出、缓冲区刷新（了解）

读入和输出消耗的时间同样计入程序运行时间

遇到输入数据大的题目可能会导致时间超限

原理：将bytes当成char类型读到数组作为缓存区，再从缓冲区模拟读入

原理代码解释：一个存放数字的字符串，如果第一个字符是'-'，则标记该数字为负数。然后从左到右访问字符串的数字字符，如果还没到结尾，说明未读完整个数，将当前结果乘10，加上该字符对应的数。访问字符串结束后，如果有负数标记则取其相反数。

缓冲区刷新

printf后并非立刻输出，而是暂存在缓冲区。

刷新缓冲区会将缓冲区中的内容立刻输出，用于交互题

交互题并不会一次给完所有输入，而是根据程序每次的输出给出特定的输入。因此需要刷新缓冲区。

例如题目指定一个数让程序寻找，每次程序输出一个数。题目根据指定数与程序输出值的大小向程序输入过大、过小或相等。