程序设计思维与实践

Thinking and Practice in Programming

算法复杂度、C++ 与 STL（上） | 内容负责：张腾/杨家齐

1 课程概述、大纲

The outline of course

课程概述

- 课程概述
  - 课程名：程序设计思维与实践 (Thinking and Practice in Programming)
  - 前置课程：计算导论与程序设计、数据结构与算法
  - 后续课程：算法设计与分析、创新创业教育实践（CCF CSP 认证）
  - 意义：
    - 是前置课程应用于解决实际问题的最佳实践，真正落实到代码实现层**针对具体问题**去思考**数据结构的应用**和**算法的设计**；
    - 是后续课程学习强有力的助攻，比如图论算法的实践与后续算法设计课中涉及的原理证明，课程的实验课与 CCF CSP 认证课 (1 学分)；
    - 培养计算机思维；
    - …

课程概述

- 上课方式
  - 每周 2 小时理论课，2 小时上机实验
  - 每周理论课后布置课堂内容相关的作业题
  - 实验课以 4 周为一轮，每 4 周一道模拟题，一次限时模拟题比赛，一次 CSP 赛制的考试。具体形式如下：
    - 第 i 周：检查前一轮即第 i-2、i-1 周留的作业，并布置这一轮的模拟题作业。
    - 第 i+1 周：ACM 赛制的模拟题比赛。（面向 CSP-T3，赛后补题同等计入平时分）
    - 第 i+2 周：检查当轮即第 i、i+1 周作业。
    - 每 i+3 周：一次 CSP 模式的模拟考试。（面向 CSP-T1、T2、T4）
- 编程语言：C / C++ / Java / Python，即 CCF CSP 认证要求语言
- 对应 IDE：Dev-C++ / Eclipse / IDLE
- PS：实验课所有代码查重，一旦发现雷同，则取消该次实验分数。

课程概述

- 作业/模拟题 验收形式：
  - 形式 1： 向 TA 简要讲述自己对于该具体问题的程序设计思路，回答助教提出的问题，包括但不限于算法复杂度、程序实现框架的细节等。
  - 形式 2： 在 CSDN/博客园/简书 等博客平台上发布“原创”题解博客，要求包含题目题号、简要题意、解题思路、通过代码、解题总结（样例博客见下图二维码）；将博客链接发送给 TA，回答相关问题；
  - 每次实验课两位 TA 验收实验，预计 120 位同学，平均 2min。特殊情况（如请假）可课下验收。
  - 实验课无特殊情况不提供答疑，实验课提前验收完或课余时间可答疑。

https://blog.csdn.net/qq_41552508/article/details/100886708

课程概述

- 课程团队及上课分配
  - 18 级 1 班、2 班、智能班、菁英班：
    - 周三第 5 大节：黄瑞哲、杨家齐（1-16 周 讲课）
    - 周五第 3 大节：俞旭铮、黄/杨（1-16 周 实验）
  - 18 级 3 班、4 班、数据班：
    - 周三第 3 大节：张腾、刘建东（1-16 周 讲课）
    - 周五第 1 大节：张腾、刘建东（1-5 周 实验）

      + 周金鹏、俞/张/刘（6-16 周 实验）
- 预授课：团队内部对每节课划分内容负责人和内容校验人，在正式授课前会在团队内部进行预授课，以确保：知识点的时间分配与把握、语言的简洁易懂等。
- 课程实时反馈：对于知识点、作业等难度的反馈，根据反馈考虑调整授课大纲。
- TA 博客：
  - 刘建东：https://me.csdn.net/qq_41552508
  - 张腾：https://zhangt.top/
  - 杨家齐：https://me.csdn.net/moon_sky1999
  - 黄瑞哲：https://me.csdn.net/weixin_43135318

课程大纲

周次	主题	课程内容	内容负责人	内容校验人
1	算法复杂度、C++与STL（上）	1. 课程概述/大纲（10min） 2. 程序设计竞赛相关名词，如 VJ、OJ、AC、RE 等，以及评测系统的使用（10min） 3. 按题目分析复杂度：快速幂、快速乘（30min） 1. 数据范围、时限以及复杂度三者关系（10min） 2. Input / Ouput（10min） 3. vector / list、algorithm (只讲 sort，结构体 sort，lambda sort) (30min) 作业: 快速幂习题一道、结构体sort习题一道	张腾 杨家齐	刘建东 黄瑞哲
2	C++与STL（下）、搜索（上）	1. stack (5min) 2. queue (5min) 3. deque (5min) 4. priority_queue (5min) 5. Lambda (5min) 6. 拓展：红黑树/哈希拉链法实现的 map/unordered_map、set/unordered_set，要求会实现红黑树STL的多关键字排序，重写哈希方法、哈希因子调参 (25min) 1. BFS (10min) 2. BFS 例题及其剪枝 (40min) 资料：附加文档（常用技巧、IDE、白盒测试总结），作业：STL 练习题、BFS例题	张腾 杨家齐	刘建东 黄瑞哲
3	搜索（下）、贪心（上）	1. DFS (20min) 2. DFS例题：CSP-再卖菜 (80分做法) (30min) 3. 贪心算法概念、贪心指标 (15min) 4. 端点排序贪心 + (35min) 作业：STL 练习题、BFS例题	张腾 黄瑞哲	刘建东 杨家齐
4	贪心（下）、二分	区间选点问题证明 +典型例题 （25min）+（25min） 二分算法框架（比较不同写法产生的边界问题）（20min）浮点数二分 二分答案（25min） 作业：课上例题	刘建东 黄瑞哲	张腾 杨家齐

课程大纲

周次	主题	课程内容	内容负责人	内容校验人
5	复杂模拟题的普适性方法	1. 题意分析与解题框架设计 （30min） 2. 面向对象与模拟题 （20min） 3. 例题演示：CSP某道题 （50min） 作业：课上模拟题一道	张腾 杨家齐	刘建东 黄瑞哲
6	线性数据结构的应用	1. 单调栈，及其例题 （25min） 2. 单调队列，及其例题 （25min） 3. 尺取 例题×2（30min） 4. 一维前缀和、二维前缀和，及其例题 (20min) 作业：滑动窗口 最大立方图 尺取	刘建东 黄瑞哲	张腾 杨家齐
7	图和树的性质与应用（上）	树的存储（邻接表、前向星）（15min） 树的遍历（DFS、BFS）（35min） 并查集与 Kruskal（prim）（50min） 作业：树的直径题、最小生成树题	张腾 黄瑞哲	刘建东 杨家齐
8	图和树的性质与应用（中）	1. 多元最短路floyd（20min） 2. 最短路 Dijkstra（20min） 3. Bellman-ford及队列优化（40min） 作业：最短路练习2题	刘建东 黄瑞哲	张腾 杨家齐
9	图和树的性质与应用（下）	差分约束系统（30min） 图的拓扑排序（20min） 图的强连通分量分解（30min） 作业：差分约束一道，拓扑排序一道	刘建东 黄瑞哲	张腾 杨家齐
10	动态规划（一）	1. 递推（20min） 2. 数字三角形（10min） 3. 记忆化（30min） 4. LIS（25min） 5. LCS（15min） 作业：递推 LIS	张腾 杨家齐	刘建东 黄瑞哲

课程大纲

周次	主题	课程内容	内容负责人	内容校验人
11	动态规划（二）	1. 背包 DP 0-1型、滚动数组 （40min） 2. 多重背包及二进制拆分（30min） 3. 完全背包（20min） 4. 输出方案 作业：0-1背包 多重背包	刘建东 杨家齐	张腾 黄瑞哲
12	动态规划（三）	区间dp（40min） 状压dp（50min） 作业：区间dp例题、状压dp例题	刘建东 杨家齐	张腾 黄瑞哲
13	动态规划 （四）	1. 树形DP（60min） 2. 单调队列优化DP（45min） 作业：树形dp例题、单调队列优化dp	刘建东 杨家齐	张腾 黄瑞哲
14	矩阵的应用	1. 矩阵结构体（15min） 2. 矩阵快速幂（40min） 3. 矩阵快速幂优化DP（30min） 作业：矩阵快速幂优化DP	刘建东 杨家齐	张腾 黄瑞哲
15	字符串	1. KMP（50min）+ 例题 2. Trie（50min）+例题 作业：KMP、Trie	刘建东 周金鹏	张腾 杨家齐
16	待定	增补内容		
17	待定	增补内容		
18	期末考试			

2 程序设计竞赛的相关名词

The related terms of programming competition

相关名词

- OJ，Online Judge，在线评测系统。如数据结构实验课使用的评测系统。
  它可以将用户的程序代码编译运行，将输出结果与正确答案进行比对，返回比对结论。
- VJ，Virtual Judge，虚拟评测系统，并非真实具有评测能力的系统，只是利用爬虫技术将用户的程序代码提交到各大 OJ 上，并爬取评测结果展现。
- 评测结果
  - AC，Accepted，该程序产生了判断系统或检查程序（通常称为 Special Judge）认为正确的输出。
  - WA，Wrong Answer，该程序未产生被系统期待的输出
  - PE，Presentation Error，该程序产生的结果是正确的但是输出格式有误
  - TLE，Time Limit Exceeded，该程序超过限定的运行时间，程序超时
  - MLE，Memory Limit Exceeded，该程序使用的内存超限
  - RE，Runtime Error，该程序在运行期间产生了无法处理的异常
  - OLE，Output Limit Exceeded，该程序产生了过多的输出
  - CE，Compile Error，该程序代码未能成功通过编译

(OJ  : ”本地编译通过了再交，行吗？”)

相关名词

- ACM赛制、OI赛制
  - 相关人员的口头术语
  - ACM赛制意指：可以实时评测、看排行榜，多次提交取最优的竞赛模式
  - OI赛制（CSP赛制）意指：不实时评测，无排行榜的，多次提交取最后一次提交，且为赛后评测的竞赛模式

3 复 杂 度 分 析

The analysis of program complexity

复杂度分析

- 快速幂引入
  - 考虑：**计算 2^{61} 需要进行多少次运算。**
  - 朴素方法：
    - 累乘 60 次，60 次乘法运算。
    - 时间复杂度为 O(n)。
  - 分治策略：
    - $2^{61} = (2^{30})^2 \times 2 = ((2^{15})^2)^2 \times 2 = (((2^7)^2 \times 2)^2)^2 \times 2 = \ldots$

      $= ((((2^2 \times 2)^2 \times 2)^2 \times 2)^2)^2 \times 2$
    - 一共需要 9 次乘法运算。
    - 时间复杂度为 $O(\log_2 n)$。

复杂度分析

- 快速幂实现
  - 把求 2^{61} 的问题一般化，对于求 a^n 的问题，如何设计？
  - 如果分治到某一步，需要求解 a^x
    - 当 x 是偶数时，直接返回 $a^{x/2} \times a^{x/2}$
    - 当 x 是奇数时，返回 $a^{x/2} \times a^{x/2} \times a$
  - 递归边界是 x = 0 时，返回1。
  - 这就是快速幂

```
long long quick_pow(long long a, long long x) {
    if (x == 0) return 1;
    long long tmp = quick_pow(a, x/2);
    if (x % 2 == 0) // 偶数
        return tmp * tmp;
    else
        return tmp * tmp * a;
}
```

复杂度分析

- 快速幂实现
  - 我不喜欢用递归，可以换成迭代吗？ —— 当然
  - 继续考虑刚才 2^{61} 的计算方法：
    - $61 = (111101)_2 = 32 + 16 + 8 + 4 + 1$
    - $2^{61} = 2^{32} \times 2^{16} \times 2^8 \times 2^4 \times 2$
    - $2^{61} = (2^{32})^1 \times (2^{16})^1 \times (2^8)^1 \times (2^4)^1 \times (2^2)^0 \times (2^1)^1$
    - 可以按位枚举指数，根据每一位的情况进行处理

```
long long quick_pow(long long a, long long x) {
    long long ans = 1;
    while (x) { // 当 x 不为 0 时继续循环
        if (x & 1) ans = ans * a;
        a = a * a;
        x >>= 1;
    }
    return ans;
}
```

复杂度分析

- 快速乘实现？
  - 把快速幂的‘×’号 换成‘+’

```
long long quick_mul(long long a, long long b) { // a*b
    long long ans = 0;
    while (b) { // 当 b 不为 0 时继续循环
        if (b & 1) ans = ans + a;
        a = a + a;
        b >>= 1;
    }
    return ans;
}
```

  -  这有什么用呢?
    - 在没有乘法器的时候用加法器实现乘法...
    - 实际上，更多的用在 a × b % MOD，其中 a × b 会爆精度（超出 32/64/128 位数的数据范围）。所以一般上述函数一般不这样用。而是 ->

复杂度分析

- 带模快速幂，带模快速乘实现

```
long long quick_pow(long long a, long long x, long long MOD) {
    long long ans = 1;
    for(; x ; x>>=1, a=a*a%MOD)
        if (x & 1)
            ans = ans * a % MOD;
    return ans;
}

long long quick_mul(long long a, long long b, long long MOD) { // a*b%MOD
    long long ans = 0;
    while (b) { // 当 b 不为 0 时继续循环
        if (b & 1) ans = (ans + a) % MOD;
        a = (a + a) % MOD;
        b >>= 1;
    }
    return ans;
}
```

4 时 限 与 复 杂 度

Time limit and Program complexity

时限与复杂度

- 在 ACM、CSP 竞赛中，题目是会给出时间限制和空间限制的，比如

时间限制：	1.0s
内存限制：	256.0MB

- 也会给出最大问题规模的范围，比如

输入格式

输入只有一行，包括恰好一个正整数n（4 ≤ n ≤ 1000）。

输出格式

输出只有一行，包括恰好n 位的整数中有趣的数的个数除以1000000007的余数。

- 有时也会以分段的形式给出

子任务
共有 10 个测试点，各测试点特点如下：
测试点1：n=10，m=10。
测试点2：n=50，m=50。
测试点3：n=300，m=300。
测试点4：n=2000，m=2000。
测试点5：n=2000，m=100000。
测试点6：n=100000，m=100000。保证 $f_i=i$。
测试点7：n=100000，m=100000。保证 $f_i=\lfloor(i+1)/2\rfloor$，
测试点8：n=100000，m=100000。保证对于同一组数据
测试点9：n=100000，m=100000。
测试点10：n=100000，m=100000。

时限与复杂度

- 计算机每秒的计算量有限，只有效率更优的算法，才能解决更大的问题规模。
- 不同的复杂度对应的计算量不同
- 假设机器速度是每秒 10^8 次基本运算，在计算机 1s 之内能解决的最大问题规模 n 如表所示：

运算量	$n!$	2^n	n^3	n^2	$n\log_2 n$	n
最大规模	11	26	464	10000	$4.5*10^6$	100000000
速度扩大两倍后	11	27	584	14142	$8.6*10^6$	200000000

- 如果我们知道一个题目的时限是 1s，数据范围是 1e6
  - 显然地，我们无法接受 $O(n^2)$ 及以上的算法
  - O(n)、O(nlogn) 是可以接受的

时限与复杂度

- 回到以分段形式给出的测试点规模
  当我们设计出较优解，无法一次性
  解决全部测试点时，思考：
  如何“骗分”？

子任务
共有 10 个测试点，各测试点特点如下：
测试点1：n=10，m=10。
测试点2：n=50，m=50。
测试点3：n=300，m=300。
测试点4：n=2000，m=2000。
测试点5：n=2000，m=100000。
测试点6：n=100000，m=100000。保证 $f_i=i$。
测试点7：n=100000，m=100000。保证 $f_i=\lfloor(i+1)/2\rfloor$，
测试点8：n=100000，m=100000。保证对于同一组数据
测试点9：n=100000，m=100000。
测试点10：n=100000，m=100000。

- 答案是：解决一部分测试点
- 框架如下：

```
int main() {
    scanf("%d%d", &n, &m);
    if ( n<=10 && m<=10 )         test1();
    else if ( n<=50 && m<=50 )    test2();
    else if ( n<=300 && m<=300 ) test3();
    // ... 以此类推
}
void test1() {
    // 解决小规模问题的暴力算法
}
void test2() {
    // 尝试解决较小规模问题的暴力算法
}
void test3() {
    // ... 瞎写
}
```

5

C++与STL（上）

C++ & STL

Input/Output

- 输入与输出
  - 在 ACM 比赛中，一般使用标准输入输出，读入题目输入，将程序输出与答案进行对比，来判定程序的正确性。
  - cstdio (C Standard Input and Output Library) 是 C 语言原生库，比起 C++ 新定义的流对象操作库 iostream 在效率方面自然更优，故建议是**只用 cstdio，不用 iostream**，因为前者熟练使用起来几乎可以替代后者，且在时间效率上无懈可击。
  - cstdio 这个库较为重要的知识在于五个方面：
    - 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）
    - 输入输出重定向 （掌握）
    - 快速读入、缓冲区冲刷 （了解）

Input/Output

- 格式化输入输出到标准 IO
  - 主要关注两个函数
    int scanf (const char * format, ...);
    int printf (const char * format, ...);

```
/* 变量定义 */
int i;  double d;  char c;  int is[15];  unsigned int ui;  long long ll;

/* 读入 */
scanf("%d %lf %c %d %u %lld", &i, &d, &c, &is[1], &ui ,&ll); // 其中&aa[1]可换为`aa + 1`
scanf("%1d", &i); // 只读1位到a

/* 输出 */
printf("%d %.2f %c %s %d %u %lld", i, d, c, s+1, is[1], ui, ll);  // 注意duoble输出
是%f,%.2f表示四舍五入到百分位

/* 有关进制的输入输出 */
int a, b; scanf("%o %x", &a, &b); // 表示将输入看作8进制、16进制读入，比如输入"0777
0x3FFFFFF"(可以没有前导0、0x)。输出同理。

/* 变量地址的输出 */
int a; printf("%p %#p", a); // 输出类似00000053 0x53
```

Input/Output

- 格式化输入输出到标准 IO
  - 主要关注两个函数
    int scanf (const char * format, ...);
    int printf (const char * format, ...);

```
/* 关于char[]的输入输出 */
char s[105];
scanf("%s", s); // 读入一个字符串，且首指针放到s+0
scanf("%s%s%s", s+1, s+21, s+41); // 读入三个字符串，且首指针分贝放到s+1、s+21、s+41，注意本样例若前两个字符串长度超过19，就会在内存有重叠而造成使用出错

/* 读入一行的处理 */ //课上做实验
scanf("%[^\n]%*c\n", s); // 读入一行(第一种形式)，首指针为s+0。%*c表示忽略后一个字符
scanf("%[^\n]", s);getchar(); // 读入一行(第二种形式)，首指针为s+0。getchar()表示把\n取掉
```

  - 其他例如“关于 string 的输入输出”等知识点可自行拓展，见二维码

Input/Output

- 格式化输入输出到字符串
  - 主要关注 sscanf、sprintf。和 scanf、printf 的使用一致，只是输入、输出的对象从标准输入输出变成了 char[]。

```
char s[] = "123 52.2 adas";
int a; double b; char ss[105];
sscanf(s, "%d%lf%s", &a, &b, ss); // a=123, b=52.2, ss="adas"
a++; ss[0]='b';
sprintf(s, "%d%lf%s", a, b, ss); // s="12452.2bdas"  后面还追有 '\0' 's' '\0'
```

- 单字符输入输出
  - 大多数情况下 scanf、printf 配合 %c 可以适应单字符输入输出的要求。但是对于优化效率，getchar()、putchar()十分有用。

```
char c = getchar();  // 从标准输入读入一个字符
putchar(c);          // 输出一个字符到标准输出
```

- 输入输出重定向 （课上做实验）
  - freopen 可以将标准输入输出导向文件，在一些规定了文件输入输出的题目，或者本地调试时极为有用。

```
freopen("in.txt", "r", stdin);  // 重定向输入流
freopen("out.txt", "w", stdout);// 重定向输出流
```

Input/Output

- 快速读入、缓冲区冲刷 (了解)
  - 读入和输出时耗同样计入程序运行时间对于一些大量数据读入的题目，格式化的输入，在底层对于识别格式串时已浪费不少时间。
  - 快速读入是一种声明了众多针对不同数据类型的读入函数，且预先将所有 bytes 当作 char 类型读到数组作为缓冲区，随后再从缓冲区中模拟读入的一种优化读入效率操作。
  - 粗略的原理代码如下，如有兴趣，可自行阅读约 50 行代码的快读模板

```
char s[]="-123456"; // 模拟的缓冲区
void read(int &x)
{
    x=0;
    bool sign=0;  int i=0;
    if(s[i] == '-') sign=1, i++;
    for(; s[i]!='\0'; i++) x = x*10 + (s[i]-'0');
    if(sign) x=-x;
}
```

- 缓冲区冲刷 (了解计算机底层并非 printf 后就立即输出，而是存在一个缓冲区)
  - fflush(stdout);
    可以将输出缓冲区中的内容当即输出，在一些交互题上是必须要使用的。

vector/list/algorithm

- vector
  - vector 是不定长数组容器。其数组长度的扩增策略是每次乘 2，所以倘若数组长度下界较为确定的话，声明 vector 时传入初始大小是比较明智的。
  - 对于容器类，较为关注的是它的声明、赋值、遍历、清空操作及相应复杂度

```
/* 声明 */
vector<int> v(10);    // 初始大小为10
vector<int> v(10, 0); // 初始大小为10，每个值都是0
vector<vector<int> > G(n+1,vector<int>(m+1));  // 二维动态数组声明
/* 赋值 */
v.push_back(20); // 末尾放入一个元素20
v[0] = 12; // 首元素改为12
v.insert(v.begin()+1, 1);  // 第2个位置插入1
v.emplace(v.begin()+1, 1); // 第2个位置插入1，不复制构造而直接插入，优化
/* 遍历 */
for(int i=0, n=v.size();i<n;i++) printf("%d\n", v[i]); // 原生
for(vector<int>::iterator it=v.begin();it!=v.end();++it) printf("%d\n", *it); // 迭代器
for(int i : v) printf("%d", i); // for range
for(auto i : v) printf("%d", i); // for range
for(auto &i : v) i++; // 每个数自增1
/* 清空 */
v.clear(); // 整个清空
v.pop_back(); // 删除最后元素
v.erase(v.begin()); // 删除首个元素
v.erase(v.begin(), v.begin()+3); // 删除前3个元素
```

vector/list/algorithm

- list
  - list 是链表容器。将数组换作链表使用，自然更在意的是其增删操作的时间复杂度和存储的空间复杂度。
  - 在链表容器中，除了对基本函数了解外，理应对迭代器有较好的理解和应用。若对时空复杂度有更高要求，单向链表 forward_list 也许更适合。

```
/* 声明 */
list<int> l1;
list<int> l2(l1.begin(), l1.end()-1); // 拷贝其他链表的一部分，复杂度是O(n)

/* 赋值 */
l1.push_back(1);  // 在链表尾加一个元素
l1.push_front(1); // 在链表头加一个元素

/* 遍历 */
l1.front(); // 首元素
l1.back();  // 尾元素
for(list<int>::iterator it = l1.begin(); it != l1.end(); it++) printf("%d\n", *it);
for(list<int>::reverse_iterator it = l1.rbegin(); it != l1.rend(); it++) printf("%d\n", *it);
for(auto &i : l) printf("%d ", i);
```

vector/list/algorithm

- list
  - list 是链表容器。将数组换作链表使用，自然更在意的是其增删操作的时间复杂度和存储的空间复杂度。
  - 在链表容器中，除了对基本函数了解外，理应对迭代器有较好的理解和应用。若对时空复杂度有更高要求，单向链表 forward_list 也许更适合。

```
/* 迭代器删减 */
list<int>::iterator it = l1.begin();
it++; // 迭代器移动
l1.insert(it, 1);  // 在迭代器后加一个元素
it = l1.erase(it); // 删除迭代器当前元素，必须重新赋值迭代器否则失效，赋值后迭代器等于被删元素下一个

/* 清空 */
l1.pop_front(); // 删除首元素
l1.pop_back();  // 删除尾元素
l1.clear();     // 清空链表

/* 排序，复杂度nlogn */
bool cmp(const int &o1, const int &o2) {return o1 > o2;}
l1.sort(cmp); // 空参数为递减
```

vector/list/algorithm

- list
  - list 是链表容器。将数组换作链表使用，自然更在意的是其增删操作的时间复杂度和存储的空间复杂度。
  - 在链表容器中，除了对基本函数了解外，理应对迭代器有较好的理解和应用。若对时空复杂度有更高要求，单向链表 forward_list 也许更适合。

```
/* 链表反向 */
l1.reverse();

/* 链表拼接 */
l1.splice(l1.end(), l2); // 之后l2被清空
l1.splice(l1.end(), l2, l2.begin(), l2.end());
l1.merge(l2);            // 有序链表的归并

/* 删除链表中全部某值元素 */
l1.remove(89);                                          // 第一种
bool cmp(const int &o) {return o==89;}          l1.remove_if(cmp); // 第二种

/* 去重 */
l1.sort(); l1.unique();
```

vector/list/algorithm

- algorithm
  - 算法库，有 C++ 内置的各种常用算法，常用的有：二分、排序、去重、全排列等，这里关注排序 sort

```
int a[] = {5,3,1,4};
vector<int> b = {5,3,1,4};
sort(a, a+4);                    // 左闭右开区间 升序排序
sort(b.begin()+1, b.end());      // 只排序第2个元素及其后面的元素，升序排序
sort(a, a+4, greater<int>()); // 降序排序
```

- 传参比较方法

```
struct P { int a,b,c; }Ps[10];
bool cmp(const P &p1, const P &p2) { return a!=p.a ? a<p.a : (b!=p.b ? b<p.b : c<p.c); }
sort(Ps, Ps+10, cmp); // 传参比较方法，三关键字升序排序
```

- 结构体重写比较方法 (课上写代码)

```
struct P {
    int a,b,c;
    bool operator<(const P &p) const {// 第1关键字升序，第2关键字降序，第3关键字升序的多关键字排序
        if(a != p.a) return a < p.a;
        if(b != p.b) return b > p.b;
        return c < p.c;
    }
}Ps[10];
sort(Ps, Ps+10);      // 重载比较方法，第1关键字升序，第2关键字降序，第3关键字升序的多关键字排序
```

vector/list/algorithm

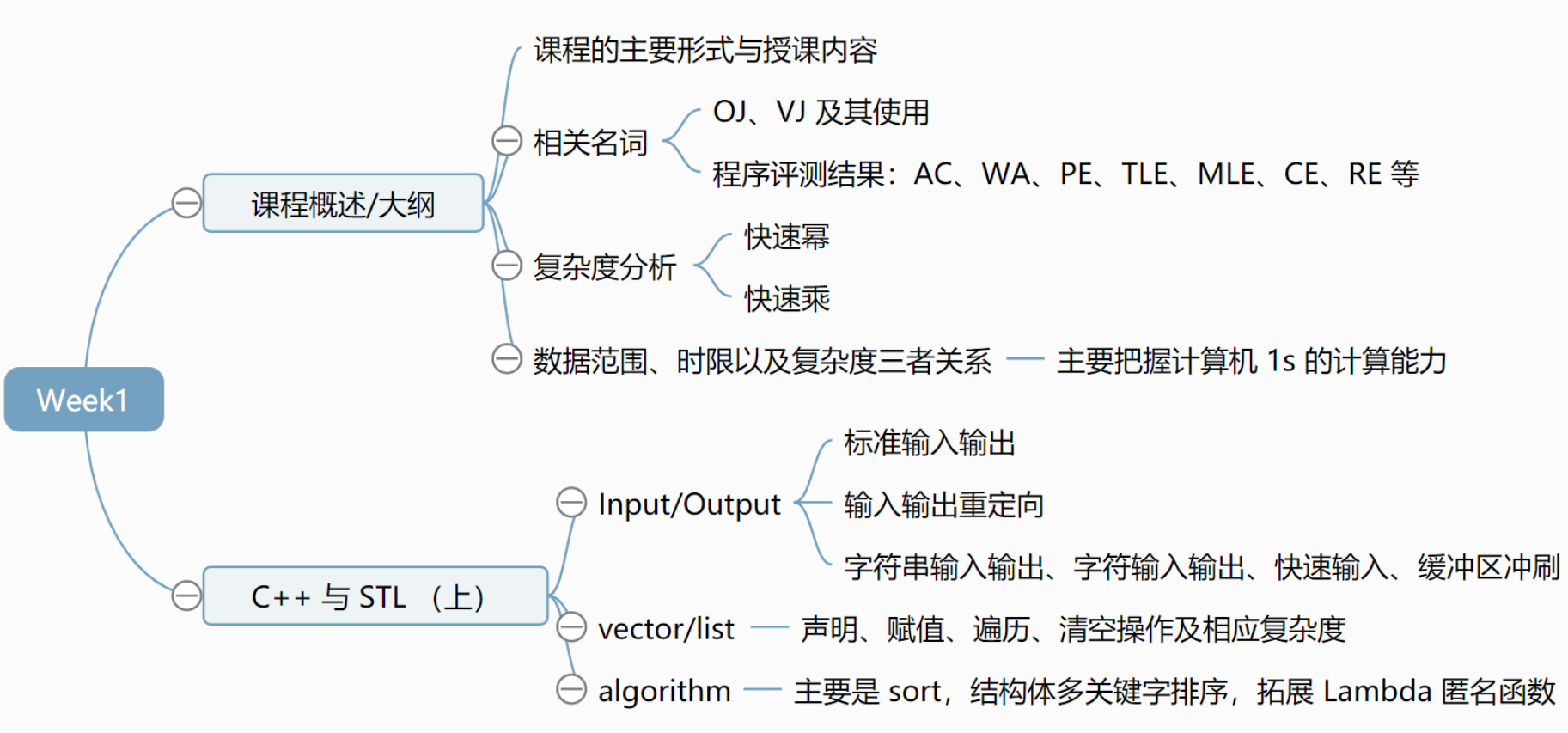
- Lambda 与 sort (了解)
  - Lambda 不是 C++ 中某个库，而是 C++11 新支持的一个特性，Lambda 是基于数学中的 λ 演算得名的，在 C++11 里表现为匿名函数的支持，可以替代掉一些一次性的谓词函数，起到简化逻辑、增加可读性的作用。
  - 是一种 “语法糖” 。
  - 这里属于拓展，主要关注怎样用在 sort 的比较函数上。

```
/* 传统的排序 */
bool cmp(int a, int b) {return a<b;}
int a[] = {1,5,3,2};
sort(a, a+4, cmp);

/* 引入Lambda，写匿名函数 */
sort(a, a+4, [](int a,int b)->bool{return a<b;} );

// Lambda 支持的结构体多关键字排序
struct P { int a,b,c; }Ps[10];
sort(Ps, Ps+10, [](P &p1, P &p2)->bool{
    if(p1.a != p2.a) return p1.a < p2.a;
    if(p1.b != p2.b) return p1.b > p2.b;
    return p1.c < p2.c;
});
```

总结


Week1
课程概述/大纲
课程的主要形式与授课内容
相关名词
OJ、VJ 及其使用
程序评测结果：AC、WA、PE、TLE、MLE、CE、RE 等
复杂度分析
快速幂
快速乘
数据范围、时限以及复杂度三者关系 — 主要把握计算机 1s 的计算能力
C++ 与 STL （上）
Input/Output
标准输入输出
输入输出重定向
字符串输入输出、字符输入输出、快速输入、缓冲区冲刷
vector/list — 声明、赋值、遍历、清空操作及相应复杂度
algorithm — 主要是 sort，结构体多关键字排序，拓展 Lambda 匿名函数

感谢收听

Thank You For Your Listening