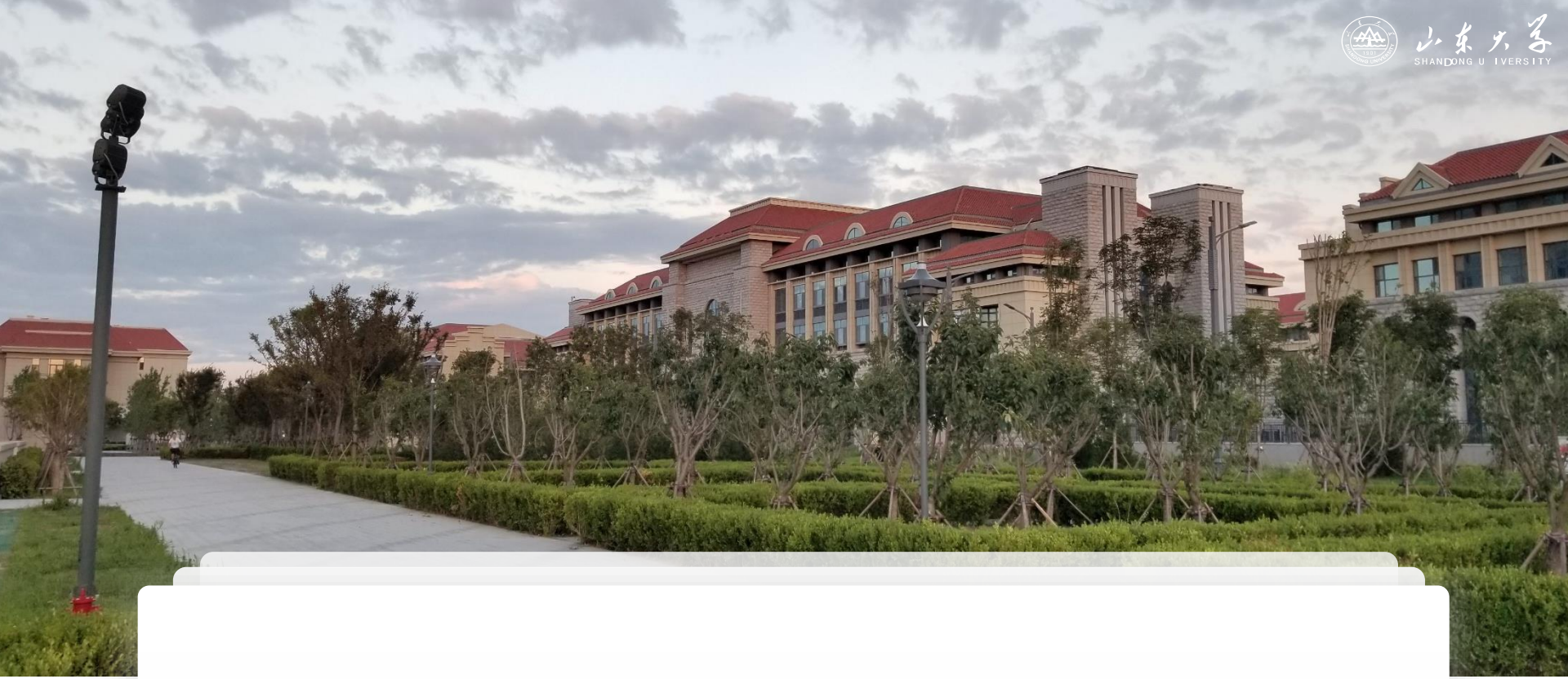
程序设计思维与实践

Thinking and Practice in Programming

复杂模拟题的普适性方法 | 内容负责：尹浩飞/韩宵玥

绪论

- 本节课的内容面向 CSP-T3
- 复杂模拟题是 CSP 中的一道坎，对于解决某道模拟题，一般来说没有定式的套路，熟能生巧尔（刷题）！
- 为此，本节课讲述的是一些经验性的总结，主要包括
  - 题意分析：高效全面的获取题目信息
  - 解题框架设计：自上而下的设计，先完成轮廓，再填充细节
  - 面向对象：适当封装，各类各司其职，提高代码的可读性与复用率
  - 课上实战：看看怎么运用上述方法解决一道复杂模拟题



1

题 意 分 析

Problem Analysis

题意分析

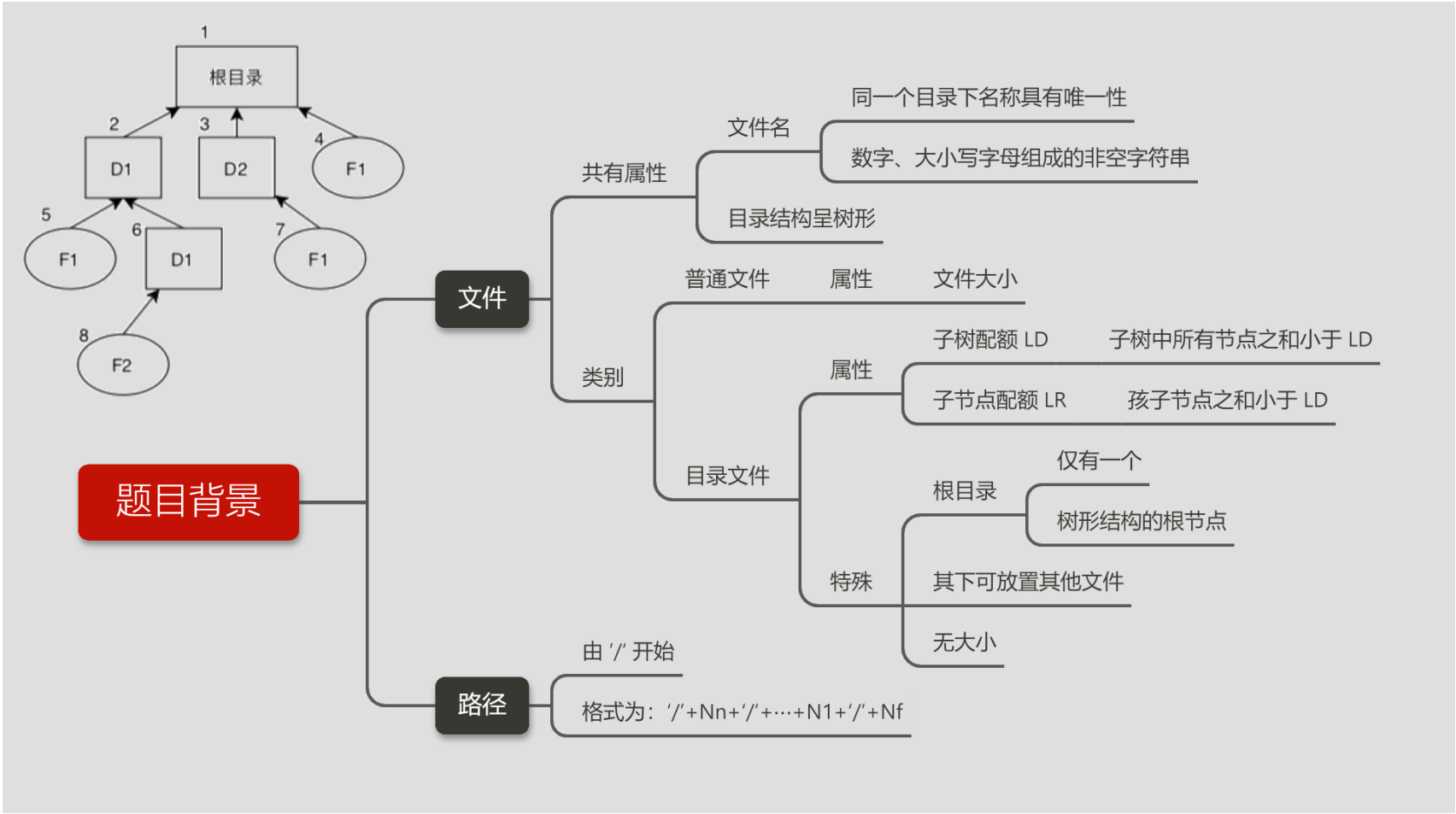
- 模拟题的定义
  - “模拟”意指题目对于输入和输出间的关系有了十分明确的规定，对于给定输入，人脑给出正确输出的难度不大，但是却是要用程序实现；
  - 题目特点是题面冗长、约束条件多、关系复杂。
- 模拟题的级别
  - Lv.1：题意较为简单，考察题意阅读和字符串处理。
  - Lv.2：题面较长（或难以理解），但要求较为简单，题目不同部分相对独立，较少的函数调用。读题与实现需要十分耐心。
  - Lv.3：并非传统的模拟题，往往是对一个系统（文件系统，语法解析）。会对实现算法的复杂度有一定的要求，程序的各个部分联系紧密。经验、STL、面向对象的运用会起到关键作用。

题意分析

- 读题
  - 题目背景：介绍一些题目中的概念，一般会严谨的给出定义。注意仔细阅读，题目所定义的概念可能与通常情况不同。
  - 题目描述：本题需要完成的操作。可能是一段编撰的情景，但是重要的信息可能就包含在故事中，一定不要跳过某些段。
  - 输入输出格式：具体的 IO 格式。
  - 子任务：
    - 部分分的特性，是否有较好拿的分数。
    - 全部分的数据，根据数据范围选定合适的算法与数据结构。

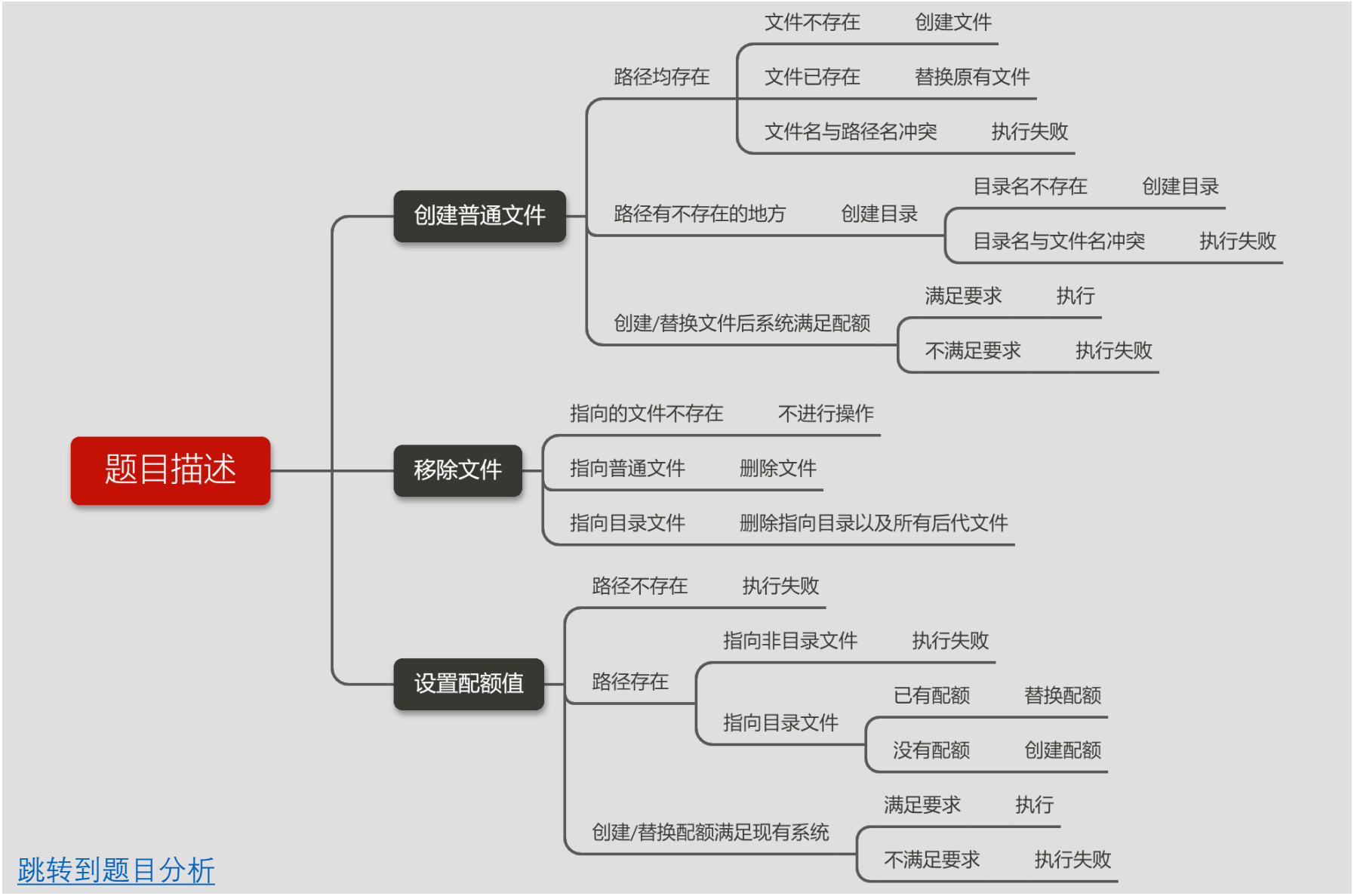
题意分析

- 拿 CSP202012-T3 带配额的文件系统 为例



题意分析

- 拿 CSP202012-T3 带配额的文件系统 为例



跳转到题目分析

2 解题框架设计

Frame Design

解题框架设计

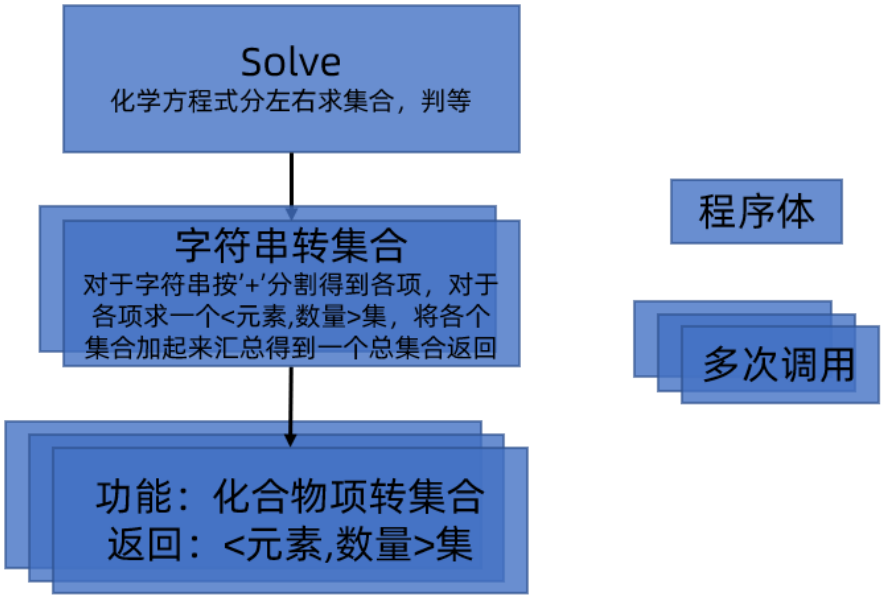
- 拿 CSP201912-T3 化学方程式 为例
- 本题的题意是：输入化学方程式，输出是否配平（等号左右的元素数量相等）
  - 例如：4Au+8NaCN+2H20+02=4Na(Au(CN)2)+4NaOH
  - 步骤 1：
    - 对于一个化学方程式，由 '=' 将其分为左右两个部分
    - 相当于求等号左右两边的式子的 <元素, 数量> 集合
      - 例如上式左边的集合为：
        - {<Au, 4>,<Na, 8>,<C, 8>, <N, 8>, <H, 4>, <O, 4>}
    - 通过判断等式两边的 <元素, 数量> 集合是否相同来判断是否配平
    - 注意到等式左右两边形式完全相同。所以只要关注怎么算等号左边的 <元素, 数量> 集合即可 。
    - 问题转换为：计算一个化学式的和式的 <元素, 数量> 集合

解题框架设计

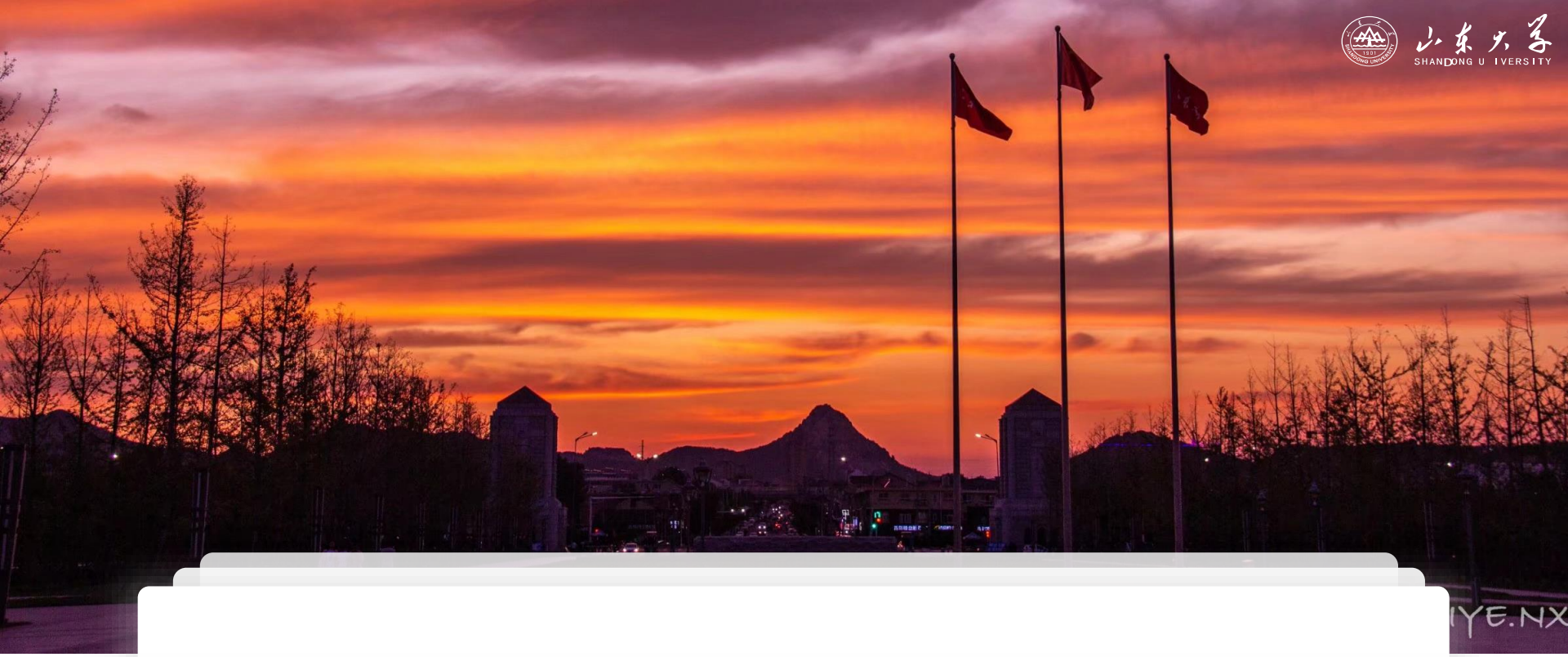
- 拿 CSP201912-T3 化学方程式 为例
  - 当前的问题为：计算一个化学式的和式的 <元素, 数量> 集合
  - 例如：4Na(Au(CN)2)+4NaOH
  - 步骤 2：
    - 可以发现和式是由互通的化学式通过 '+' 连接生成的
    - 求上述和式的 <元素, 数量> 集合 即分别求每个化学式的 <元素, 数量> 集合 后将其合并即可。
    - 问题转换为：计算一个化学式的 <元素, 数量> 集合
    - 该问题的处理在稍后的实战演示中详细讲解。

解题框架设计

- 拿 CSP201912-T3 化学方程式 为例
  - 代码框架总结如下：




```
solve(原式):
    左侧公式, 右侧公式 = split(原式, '=')

    左侧化学式集合 = split(左侧公式, '+')
    右侧化学式集合 = split(右侧公式, '+')

    define 左侧集合, 右侧集合 in <元素,数量>集合

    for 化学式 in 左侧化学式集合:
        左侧集合 += str2set(化学式)

    for 化学式 in 右侧化学式集合:
        右侧集合 += str2set(化学式)

    if 左侧集合 == 右侧集合:
        合法
    else:
        不合法
```



3

面向对象与模拟题

Object-oriented and Simulation

面向对象与模拟题

- 面向对象 (Object Oriented) 是相对于面向过程来讲的。

  面向对象方法，把相关的数据和方法组织为一个整体来看待，从更高的层次来进行系统建模，更贴近事物的自然运行模式。

  是一种对现实世界理解和抽象的方法。
- 面向对象的三大特性是：**封装**、继承、多态
  - 主要关注封装这一特性，因为我们不会在 CSP、ICPC 等各种程序设计竞赛上开始设计各种接口或者是纯虚类，再写个类继承它实现它，完成一道题 —— 而是直接针对具体问题设计实现类
- 为什么有面向对象？
  - 计算机科学 -> 客观真理
  - 软件工程 -> 人的需求

面向对象与模拟题

- 封装的取舍
  - 封装本要解决的问题
    - 重复代码：对相似代码段进行抽象，总结，封装成函数体
    - 简化用户接口 / 隐藏实现细节：只管调用，不管其内部的实现
    - 参数和返回值的设计：难点
  - 封装带来的问题
    - 程序耦合：即程序间的依赖关系，比如相互调用 / 乃至多层调用，可能牵一发而动全身 (比如改需求)
    - 代码冗余：有些代码只会封装后之后使用一次，此时再进行封装将造成代码量增加。
  - 对我们来说，取其助于解决问题的精华即可，不要过于拘束严格的面向对象！接下来实战，然后再总结。

4 实 战 演 示

Actual Problem Designing

实战演示1

- 题目：CSP201912-T3 化学方程式
  - 继续上面提到的，现在要解决的问题变为：
    - 计算一个化学式的 <元素, 数量> 集合
- 思路分享：
  - 维护当前读取到的位置，并不断向后枚举，有以下几种可能的情况：
    - 在化学式的最开始，读取数字作为整个化学式的系数；
    - 读取一个元素的元素符号，并读取其后的数字（没有视为1），表示该元素的数量。
    - 读取括号内的化学式，并读取其后的数字（没有视为1），作为括号内化学式的系数。
  - 不难发现，上述操作都基于两种原子操作：
    - 读取一个元素符号 read_symbol
    - 读取一个整数 read_int

实战演示1

- 题目：CSP201912-T3 化学方程式
  - 要解决的问题变为：计算一个化学式的 <元素, 数量> 集合
  - 思路分享：
    - 读取括号内的表达式是一个与当前问题完全相同的问题。
    - 递归调用当前函数即可解决。
    - 于是实现的伪代码如下：

```
<元素,数量>集合 str2set(str:string&, pos:int&):
    define 答案 in <元素,数量>集合
    化学式总系数 = read_int();
    while pos < str.length():
        if str[pos] == '(':              // 递归解决
            答案 += str2set(str, pos + 1)
        else if str[pos] == ')':         // 当前化学式已处理完毕
            化学式总系数 = read_int(str, pos + 1)
            break
        else :                           // 读取元素
            元素 = read_symbol(str, pos)
            数量 = read_int(str, pos)
            res += {元素, 数量}
    return res * 化学式总系数
```

实战演示1

- 题目：

  CSP201912-T3 化学方程式

  - 部分代码提示：

```cpp
// 字符串分割函数，把 str 按照 c 进行分割
vector<string> split(const string &str, char c) {
    vector<string> res;
    string tmp;
    for (char i : str)
        if (i == c) res.push_back(tmp), tmp = "";
        else tmp += i;
    if (!tmp.empty()) res.push_back(tmp);
    return res;
}
```

```cpp
// <元素, 数量> 集合类
struct ele_set {
    map<string, int> mp;
    void add(const string& str, int val){ mp[str] += val; }
    ele_set operator+=(const ele_set &t) {
        for (const auto & it : t.mp) mp[it.first] += it.second;
        return *this;
    }
    ele_set operator*=(int v) {
        for (auto & it : mp) it.second *= v;
        return *this;
    }
    bool operator==(ele_set &t) const { return mp == t.mp; }
};
```

```cpp
// 读取一个整数
int r_int(const string &str, int &p) {
    int res = 0;
    while (p < str.length() && isdigit(str[p]))
        res = res * 10 + str[p] - '0', ++p;
    return res == 0 ? 1 : res;
}
```

p 作为参数要传递引用，

保证其持续向后枚举。

实战演示2

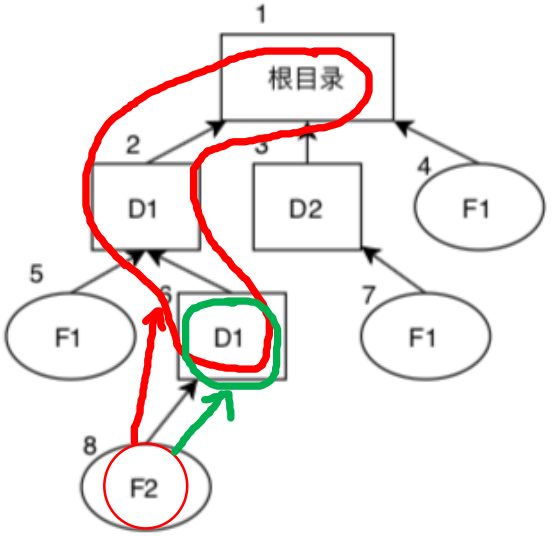
- 题目：CSP202012-T3 带配额的文件系统 跳转到题目描述
- 题目分析：
  - 路径测试
    - 对于输入的路径，我们应当首先检查其合法性，其状态有：
      - 路径错误：路径把普通文件当做目录文件进行访问
      - 路径不完整：执行到某一目录，发现下一级目录不存在
      - 成功找到指定的文件。
    - 路径测试需要向之后的其他操作提供输入路径的状态，从而进行不同的处理。

实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 改变配额（路径测试状态要求：找到对应文件且是目录文件）
    - 统计目录中的文件是否满足输入的配额：满足则更改，否则操作失败。
      - 由于子树大小可能是 O(操作数) 的，若统计需要枚举子树中的节点，那统计操作的时间复杂度即为 O(操作数)
    - 若使用上述方法，总时间复杂度为 O(操作数^2)，这是无法接受的。
    - 如果子树中节点的大小之和是在插入删除时进行维护的（即作为目录节点的一个属性，保存在目录节点中），那么每次只需 O(1) 进行判断即可。
    - 如此，就需要在插入或删除文件是维护节点的：
      - 当前子树节中所有文件大小之和
      - 当前子节点的文件大小之和

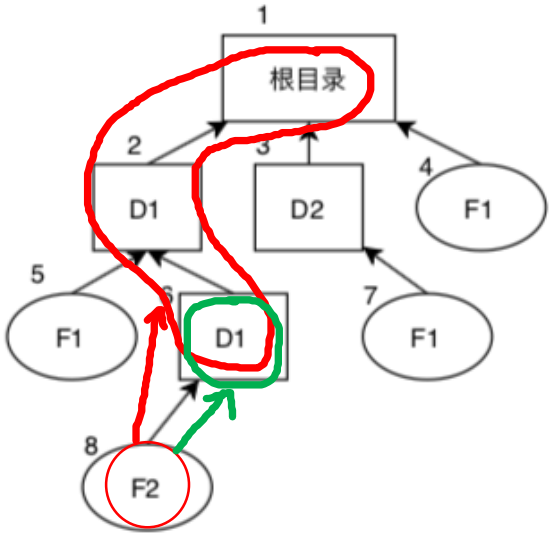
实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：



  - 移除文件（路径测试状态要求：找到对应文件）
    - 获取文件大小
      - 普通文件：直接使用文件大小
      - 目录文件：以当前目录文件为根的子树内所有文件大小之和（已维护）
    - 更新从根到当前文件的所有目录文件的当前配额使用信息
    - 删除对应文件

实战演示2



- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 创建普通文件
    - 新建文件：（路径测试状态要求：路径不完整）
      - 测试加入后是否满足配额的要求
        - 从根到当前文件路径上的所有目录文件都需要满足子树配额（如图）
        - 当前文件的父节点还需额外满足后代配额
        - 可以使用预分配方法（假设分配上去，判断是否合法）
      - 若满足配额要求，则更新从根到当前文件的所有目录文件的当前配额使用信息。从根向下依次访问时，若发现下一层目录不存在，则新建目录文件后再进行访问。
    - 替换文件：（路径测试状态要求：找到对应文件且是普通文件）
      - 为了简化关于配额的操作，先删除原有文件，然后判断当前文件能否成功插入：若可行则保留；若不可行则重新新建原有文件。

实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 程序设计：
  - 设计文件类

成员变量

```
struct file {
    // 文件类型 0 普通文件  1 目录
    int file_type;
    // 文件大小（普通）
    ll file_size;
    // 子树配额（目录），后代配额（目录）
    ll dir_all_mx, dir_son_mx;
    // 子树现有大小（目录），后代现有大小（目录）
    ll dir_all_used, dir_son_used;
    // 目录内信息
    map<string, file *> child_set;
```

成员函数

```
    // 构造函数
    file(int type, ll size = 0);
    // 析构函数
    ~file();
    // 文件大小分配 预测试
    bool pre_add_size(ll size, bool last = false);
    // 文件大小分配
    void add_size(ll size, bool last = false);
    // 返回当前文件以及下级目录的大小总和
    ll get_size() const ;
    // 设置配额大小
    bool set_size(ll all_mx, ll son_mx);
};
```

实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 程序设计：
  - 设计文件系统类

```
struct file_system {
    // 根目录
    file *root;
    // 当前操作路径
    vector<string> path_sp;
    // 当前操作文件
    string ls_dir;

    // 构造函数
    file_system();
    // 析构函数
    ~file_system();
    // 设置当前路径
    void set_path(const string &path);
    // 路径测试 返回节点指针与查找状态
    pair<file *, int> find() const;
    // 新建文件，函数判断是否满足配额，维护大小数据
    bool creat_file(ll size) const;
    // 删除文件，维护大小数据
    void remove_file(ll size) const;
};
```

实战演示2

- 题目：CSP202012-T3 带配额的文件系统

- 程序设计：
  - 文件系统的使用

```
输入 操作类类型 与 路径(path)
FileSystem.set_path(path);
ret_point, ret_status = FileSystem.find();
if 操作类型 == "C":
    输入 文件大小(file_size)
    if ret_status == 成功找到:
        if ret_point->file_type == 目录文件:
            输出失败
        else :
            old_flie_size = ret_point->get_size()
            FileSystem.remove_file(old_flie_size)
            if FileSystem.creat_file(file_size):
                输出成功
            else :
                FileSystem.creat_file(old_flie_size)
                输出失败
    else if ret_status == 路径不完整:
        if FileSystem.creat_file(file_size):
            输出成功
        else:
            输出失败
    else: // ret_status == 路径错误
        输出失败
else if 操作类型 == "R":
    if ret_status == 成功找到:
        FileSystem.remove_file(ret_point->get_size())
    输出成功
else if 操作类型 == "Q":
    输入 后代最大值(son_mx) 与 子树最大值(all_mx)
    if ret_status == 成功找到 且 ret_point->file_type == 目录文件:
        if ret_point->set_size(all_mx, son_mx):
            输出成功
        else:
            输出失败
    else:
        输出失败
```

实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 部分代码提示：
  - 文件系统类

```
79      // 设置当前路径
80      void set_path(const string &path) {
81          path_sp.clear();
82          string tmp;
83          for (auto c:path) {
84              if (c == '/') {
85                  if (!tmp.empty()) path_sp.push_back(tmp), tmp = "";
86              } else tmp += c;
87          }
88          ls_dir = tmp;
89      }
```

```
91      // 根据路径查找文件与查找状态
92      /**
93       * 状态:
94       *      0 成功找到
95       *      1 路径不完整
96       *      2 路径错误
97       * */
98      pair<file *, int> find() const {
99          file *now = root;
100         for (const auto &np : path_sp) {
101             if (!now->child_set.count(np)) return make_pair( x: nullptr, y: 1);
102             if (now->child_set[np]->file_type == 0) return make_pair( x: nullptr, y: 2);
103             now = now->child_set[np];
104         }
105         // ls_dir 为空时，是根目录
106         if (!ls_dir.empty()) {
107             if (!now->child_set.count(ls_dir)) return make_pair( x: nullptr, y: 1);
108             now = now->child_set[ls_dir];
109         }
110         return make_pair(now, y: 0);
111     }
```

实战演示2

- 题目：

  CSP202012-T3 带配额的文件系统

- 部分代码提示：
  - 文件系统类

```
113     // 创建文件
114     // 此时应保证路径上一定可以成功创建指定文件
115     // 当前函数判断是否满足配额
116     bool creat_file(ll size) const {
117         file *now = root;
118         // 先进行配额检查，顺便创建不存在的目录
119         for (const auto &np : path_sp) {
120             if (!now->pre_add_size(size)) return false;
121             if (!now->child_set.count(np))
122                 now->child_set[np] = new file( type: 1);
123             now = now->child_set[np];
124         }
125         if (!now->pre_add_size(size,  last: true)) return false;
126 
127         // 经过检查合法，进行分配并新建
128         now = root;
129         for (const auto &np : path_sp) {
130             now->add_size(size);
131             now = now->child_set[np];
132         }
133         now->add_size(size,  last: true);
134         now->child_set[ls_dir] = new file( type: 0, size);
135         return true;
136     }
```

```
138     // 删除文件
139     // 此时应保证路径上一定可以成功访问到指定文件
140     void remove_file(ll size) const {
141         file *now = root;
142         // 先进行配额检查，顺便创建不存在的目录
143         for (const auto &np : path_sp) {
144             now->add_size(-size);
145             now = now->child_set[np];
146         }
147         // 只有删除的是普通文件时，更新一级孩子大小
148         if (now->child_set[ls_dir]->file_type == 0) {
149             now->add_size(-size,  last: true);
150         } else now->add_size(-size);
151 
152         delete now->child_set[ls_dir];
153         now->child_set.erase(ls_dir);
154     }
```

总结

- 题意分析之利用时限计算复杂度设计数据结构：
  - 设计树形数据结构时比起 O(n) 用 map 可以 O(logn) 来找子目录；
  - 分析时限，发现设置配额时计算重新计算是否合法，单次计算的复杂度可能退化为 O(n)，所以在新增与删除时进行维护。
- 解题框架设计之从何入手：
  - 如果不知从何入手，不要盲目的开始写，建立框架最为重要；
  - 然后对于一个 “功能” 不要一上来就写细节，而是假装已经封装写好了，从而先避免细节，来关注整体设计。
- 面向对象中之封装不要过于拘束：
  - 类似的抉择还会有的，坚持高内聚的架构，还是肆意放飞，你写代码舒服就好的。毕竟现在只是为了解决一道题，以后工程架构时这些经历会帮助你的。

总结

- 今日学到


复杂模拟题的普适性方法
绪论
课程针对题目 CSP-T3
课堂本质：编程技巧与经验分享，没有固定的套路
题意分析
模拟题的特点
难度分类
读题时的注意事项
不要跳过一段不读
要时刻整理记录已经读到的题面
解题框架设计
自上而下的程序设计
面向对象
增加代码的复用性
无需严格面向对象，高效是第一目的
实战演示
字符串处理技巧
文件系统设计方案
思考的方法

感谢收听

Thank You For Your Listening