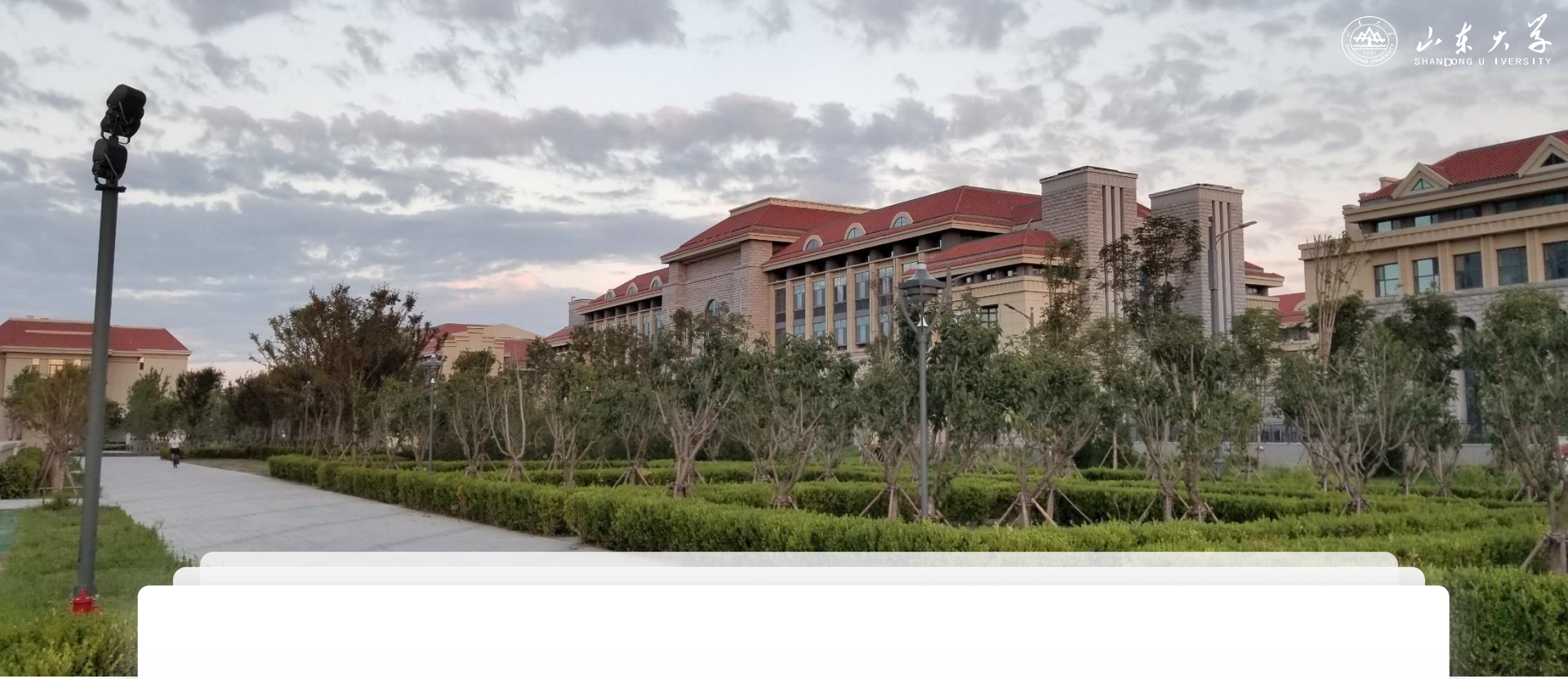
程序设计思维与实践

Thinking and Practice in Programming

复杂模拟题的普适性方法 | 内容负责：师浩晏

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

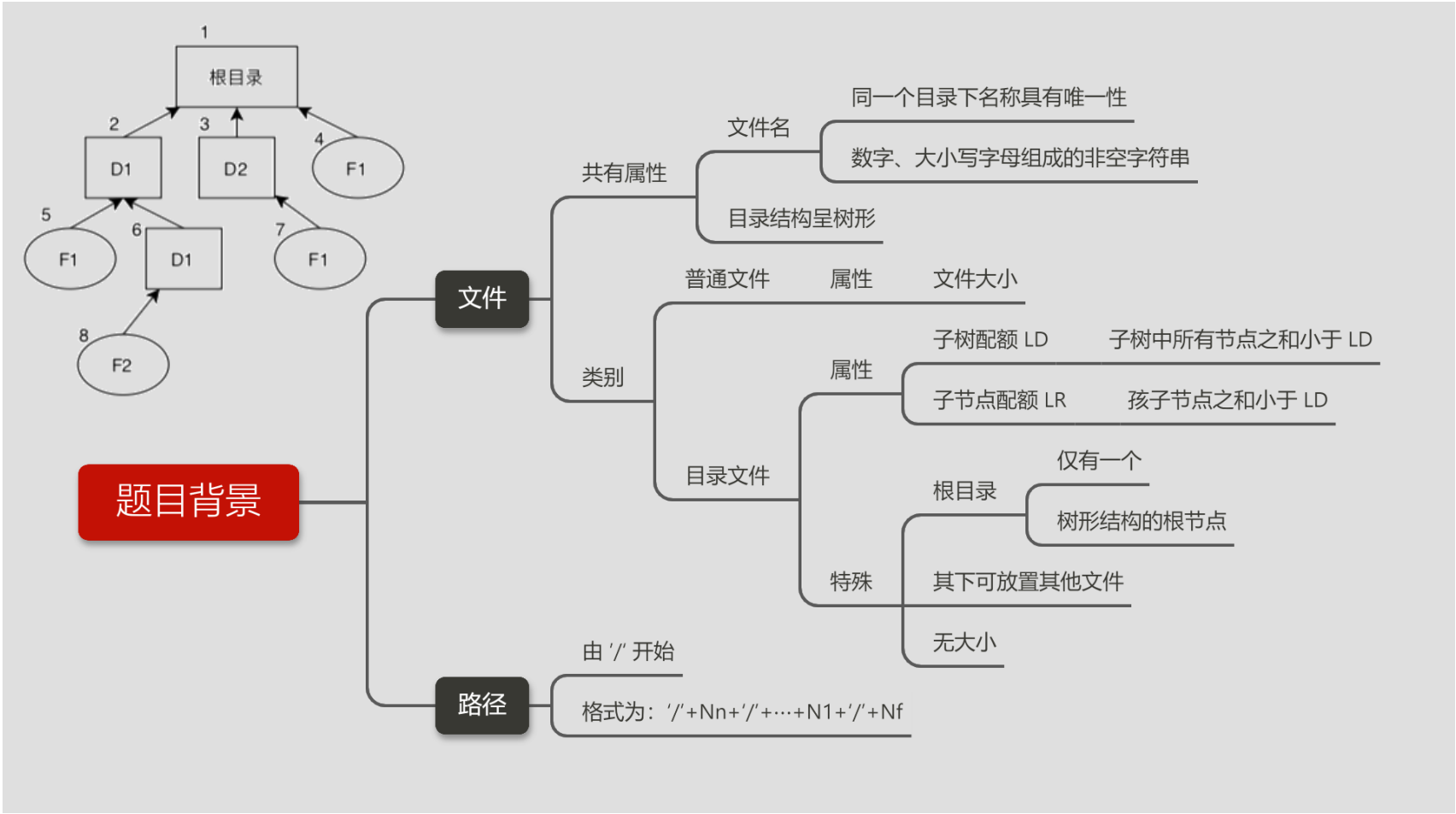
- 模拟题的定义
  - “模拟”意指题目对于输入和输出间的关系有了十分明确的规定，对于给定输入，人脑给出正确输出的难度不大，但是却是要用程序实现；
  - 题目特点是题面冗长、约束条件多、关系复杂。
- 模拟题的级别
  - Lv.1：题意较为简单，考察题意阅读和字符串处理。
  - Lv.2：题面较长（或难以理解），但要求较为简单，题目不同部分相对独立，较少的函数调用。读题与实现需要十分耐心。
  - Lv.3：并非传统的模拟题，往往是对一个系统（文件系统，语法解析）。会对实现算法的复杂度有一定的要求，程序的各个部分联系紧密。经验、STL、面向对象的运用会起到关键作用。

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。

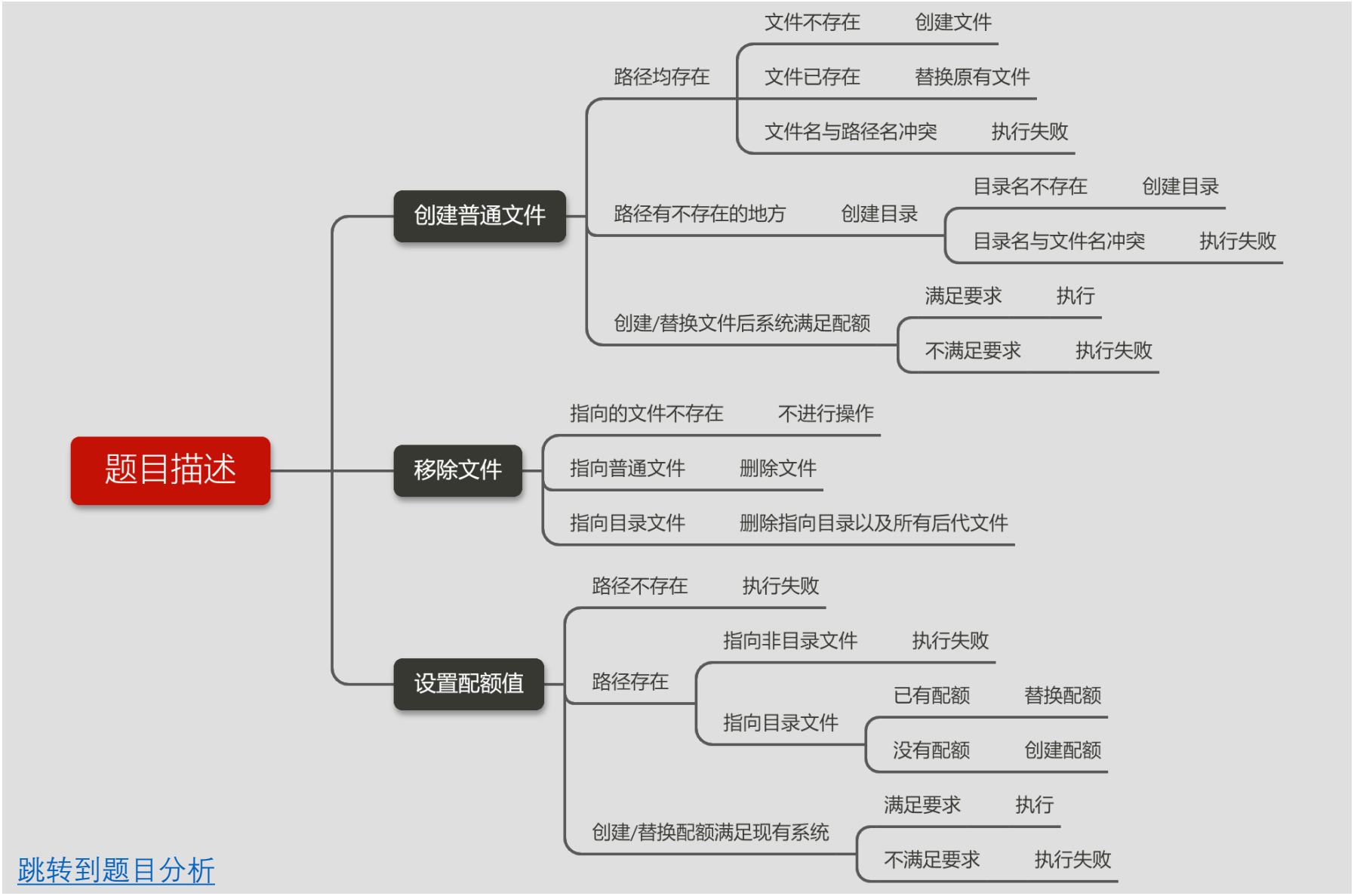
题意分析

- 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

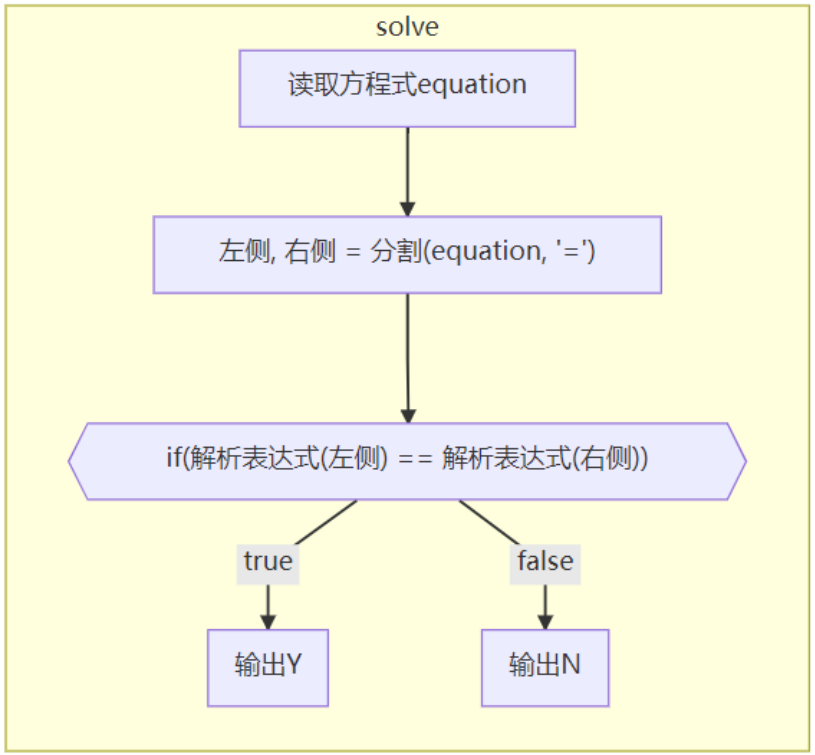
- 拿 CSP201912-T3 化学方程式 为例
- 本题的题意是：输入化学方程式，输出是否配平（等号左右的元素数量相等）
  - 例如：4Au+8NaCN+2H20+02=4Na(Au(CN)2)+4NaOH
  - 步骤 1：
    - 对于一个化学方程式，由 '=' 将其分为左右两个部分
    - 相当于求等号左右两边的式子的 <元素, 数量> 集合
      - 例如上式左边的集合为：
        - {<Au, 4>,<Na, 8>,<C, 8>, <N, 8>, <H, 4>, <O, 4>}
    - 通过判断等式两边的 <元素, 数量> 集合是否相同来判断是否配平
    - 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

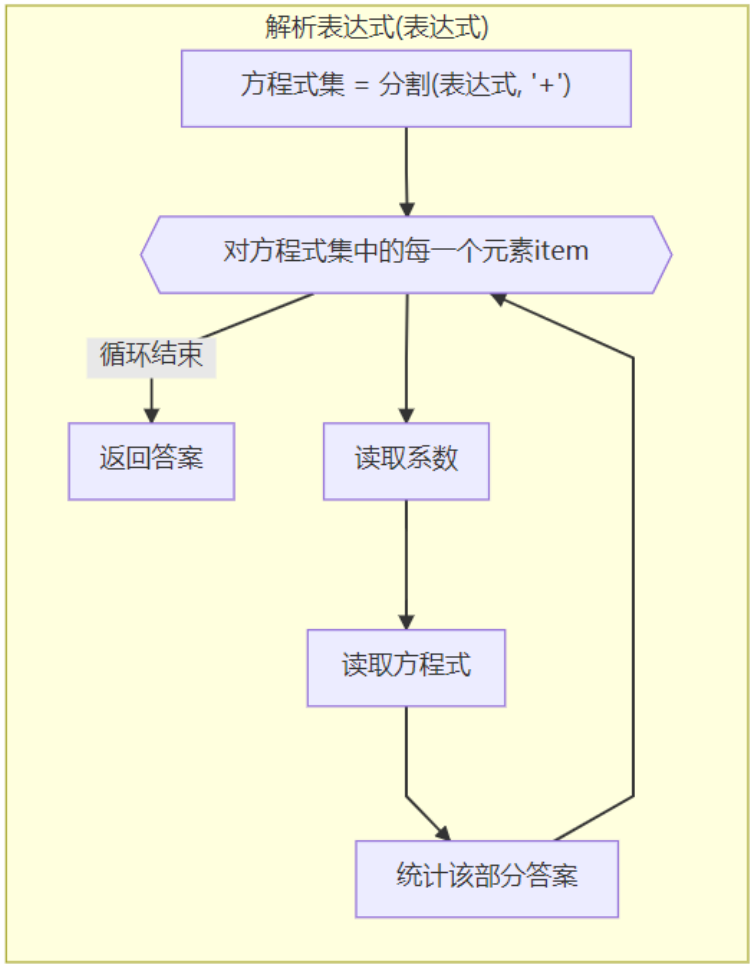
解题框架设计

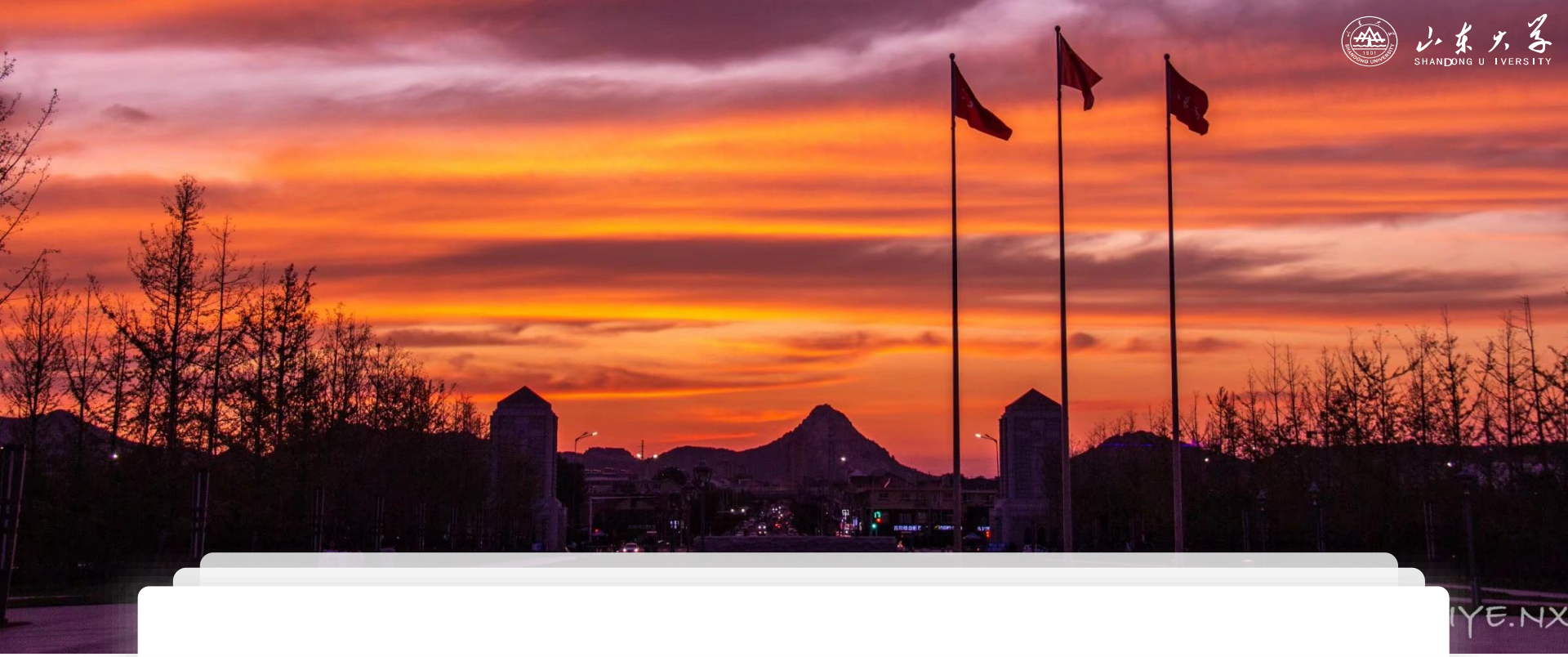
- 拿 CSP201912-T3 化学方程式 为例
  - 当前的问题为：计算一个化学式的和式的 <元素, 数量> 集合
  - 例如：4Na(Au(CN)2)+4NaOH
  - 步骤 2：
    - 可以发现和式是由互通的化学式通过 '+' 连接生成的
    - 求上述和式的 <元素, 数量> 集合 即分别求每个化学式的 <元素, 数量> 集合 后将其合并即可。
    - 问题转换为：计算一个化学式的 <元素, 数量> 集合
    - 该问题的处理在稍后的实战演示中详细讲解。

解题框架设计

- 拿 CSP201912-T3 化学方程式 为例
  - 框架总结如下：







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

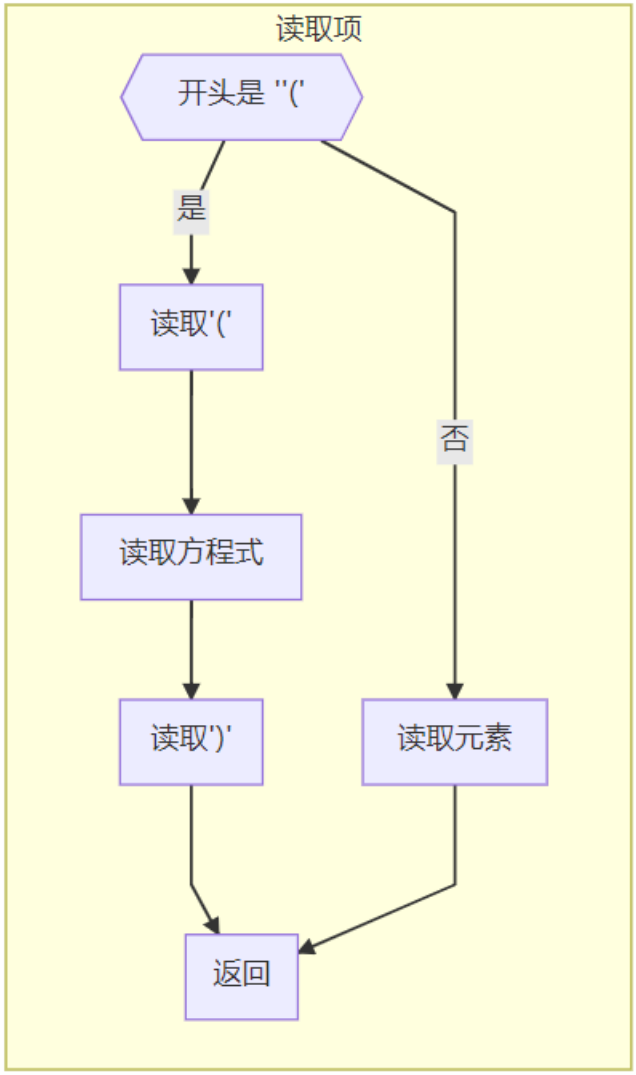
实战演示1

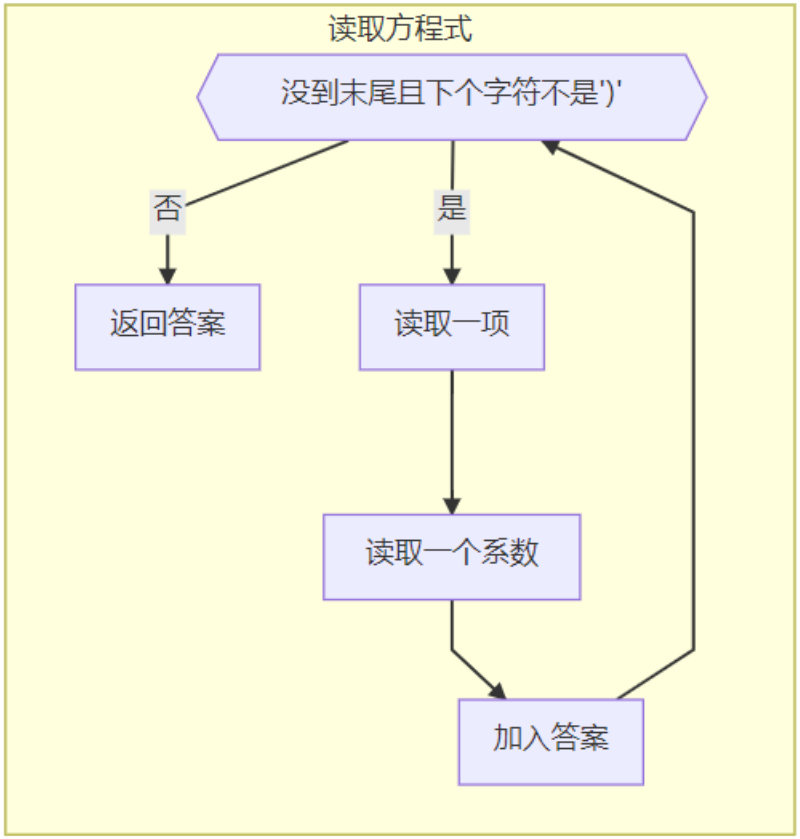
- 题目：CSP201912-T3 化学方程式
  - 继续上面提到的，现在要解决的问题变为：
    - 计算一个化学式的 <元素, 数量> 集合
- 思路分享：
  - 维护当前读取到的位置，并不断向后枚举，有以下几种可能的情况：
    - 在化学式的最开始，读取数字作为整个化学式的系数；
    - 接下来，循环读取一项，以及该项可能存在的系数
    - 对于一项，可能是一个元素符号，可能是括号内一个新的化学式
  - 不难发现，上述操作都基于两种原子操作：
    - 读取一个元素符号 readElement
    - 读取一个整数 readInt

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：
    - 不断循环，读取一项和一个系数，直到遇到末尾或右括号
    - 由于一项可能由（方程式）构成，在读取项的过程中，如果读取到左括号，则递归调用读取方程式的函数，否则调用读取元素操作

实战演示1


读取项
开头是 "('
是
否
读取'('
读取方程式
读取')'
读取元素
返回


读取方程式
没到末尾且下个字符不是')'
否
是
返回答案
读取一项
读取一个系数
加入答案

实战演示1

- 题目：

  CSP202012-T3 带配额的文件系统

- 部分代码提示：

```
// 分割字符串，以c为分界，把s分成多个字符串
vector<string> split(string &s, char c) {
    stringstream ss(s);
    vector<string> res;
    string t;
    while (getline(ss, t, c)) {
        res.push_back(t);
    }
    return res;
}
```

```
// 把cur中的元组，值乘上coef再加入res中
void merge(map<string, int> &res, map<string, int> const &cur, int coef) {
    for (auto &i : cur) {
        res[i.first] += i.second * coef;
    }
}
```

```
// 读取一个整数
int readInt(string &s, int &p) {
    int res = 0;
    while (p != s.size() and isdigit(s[p])) {
        res = res * 10 + s[p] - '0';
        p++;
    }
    if (res == 0) {
        return 1;
    } else {
        return res;
    }
}
```

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：
  - 数据结构
    - 应该是树形结构
    - 普通文件和目录文件一视同仁，使用一个bool变量区分类型
    - 普通文件记录大小，目录文件记录配额和孩子文件
    - 初步设计：

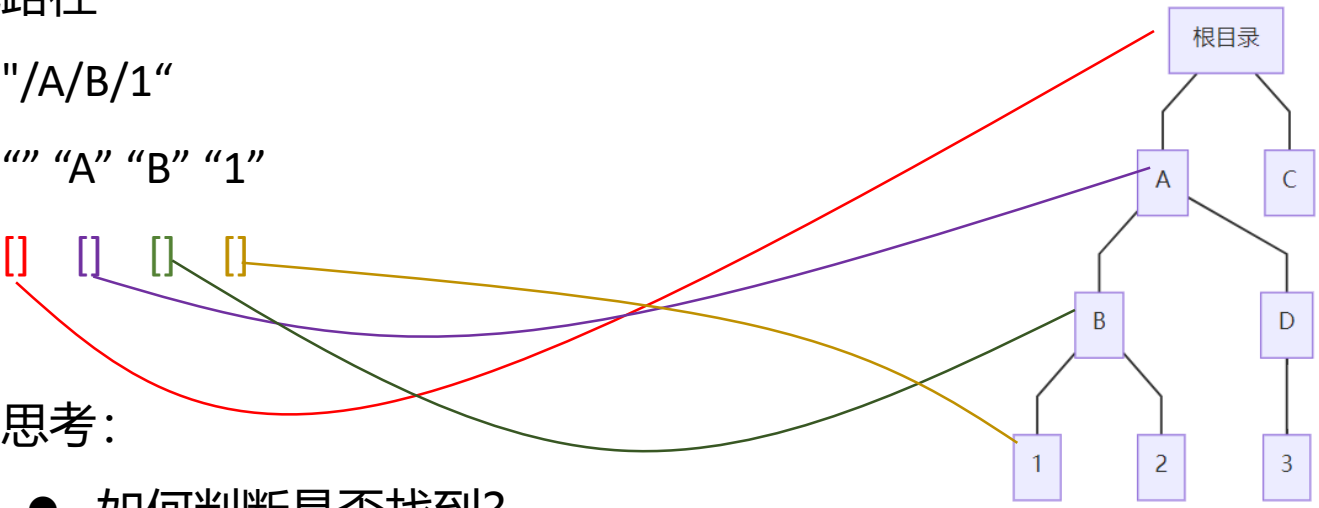
```
struct file {
    bool isDir;
    long long LD, LR;
    map<string, file> children;

    long long fileSize;
};
```

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：
  - 解析路径
    - 对于输入的路径，应该解析出路径上各个目录的名字
  - 查找路径
    - 对于输入的路径，应该找到路径上各个文件的位置
    - 返回一个指针数组保存路径上各级目录指针
    - 如果从某一步骤找不到了，则直接返回
    - 思考：
      - 如何判断是否找到?
      - 如何判断找到的是目录文件还是普通文件

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：
  - 解析路径
    - 对于输入的路径，应该解析出路径上各个目录的名字
  - 查找路径
    - "/A/B/1“
    - “” “A” “B” “1”
    - [] [] [] []
    - 思考：
      - 如何判断是否找到?
      - 如何判断找到的是目录文件还是普通文件



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

创建普通文件指令的格式如下：

```
C <file path> <file size>
```

创建普通文件的指令有两个参数，是空格分隔的字符串和一个正整数，分别表示需要创建的普通文件的路径和文件的大小。

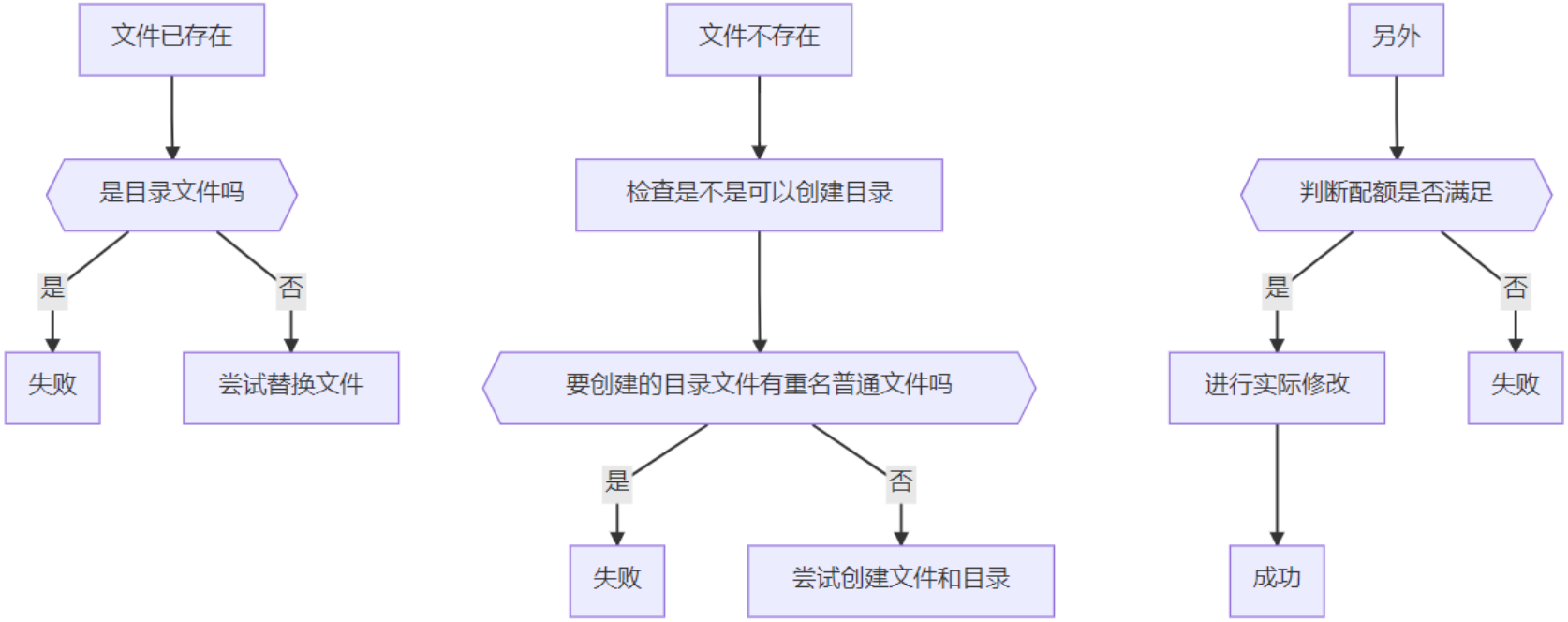
对于该指令，若路径所指的文件已经存在，且也是普通文件的，则替换这个文件；若路径所指文件已经存在，但是目录文件的，则该指令不能执行成功。

当路径中的任何目录不存在时，应当尝试创建这些目录；若要创建的目录文件与已有的同一双亲目录下的孩子文件中的普通文件名称重复，则该指令不能执行成功。

另外，还需要确定在该指令的执行是否会使该文件系统的配额变为不满足，如果会发生这样的情况，则认为该指令不能执行成功，反之则认为该指令能执行成功。

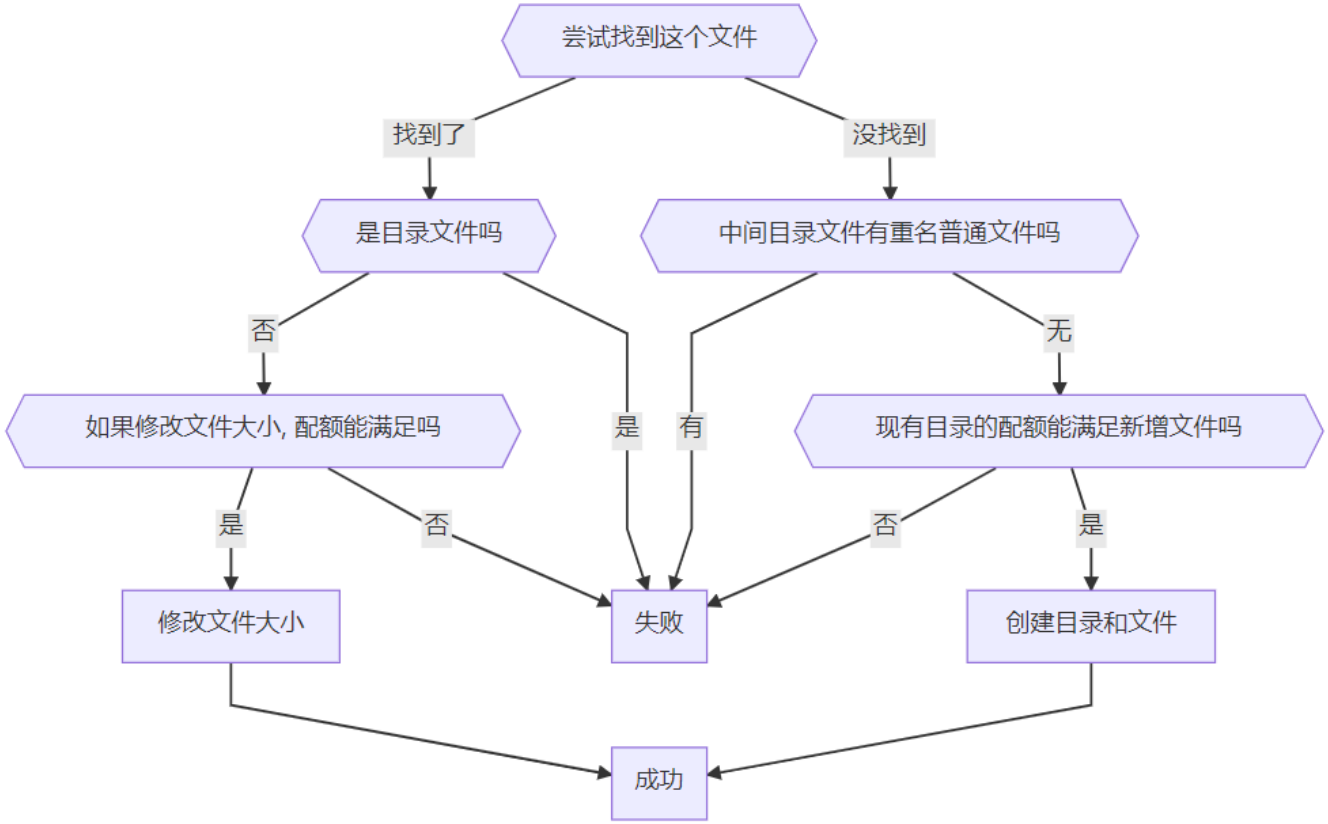
实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 创建普通文件



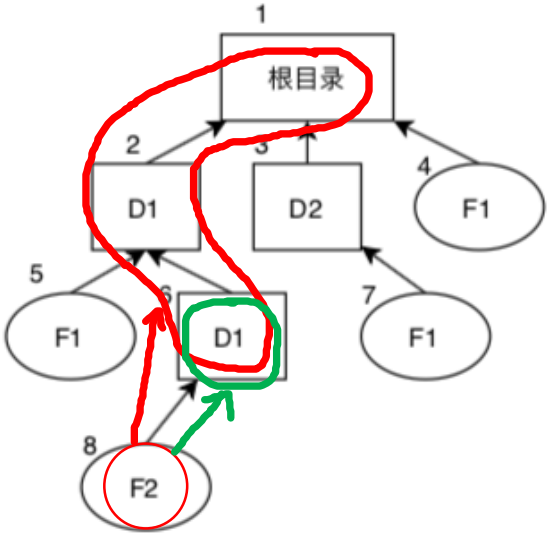
实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 创建普通文件



实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 检查配额是否满足
  - 需要检查包含指定文件的所有路径
  - 是否可以通过遍历子树统计文件大小？

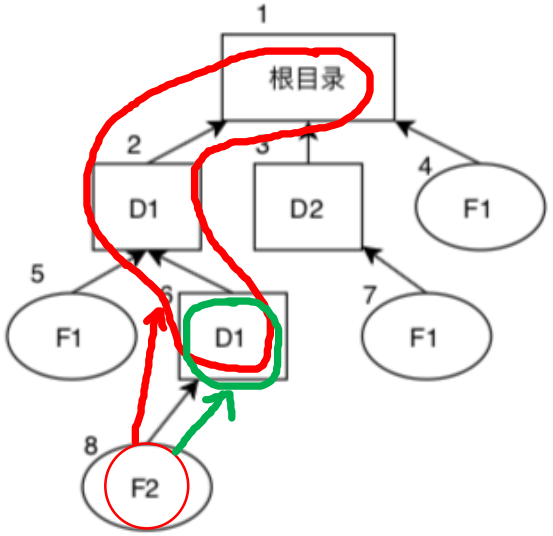


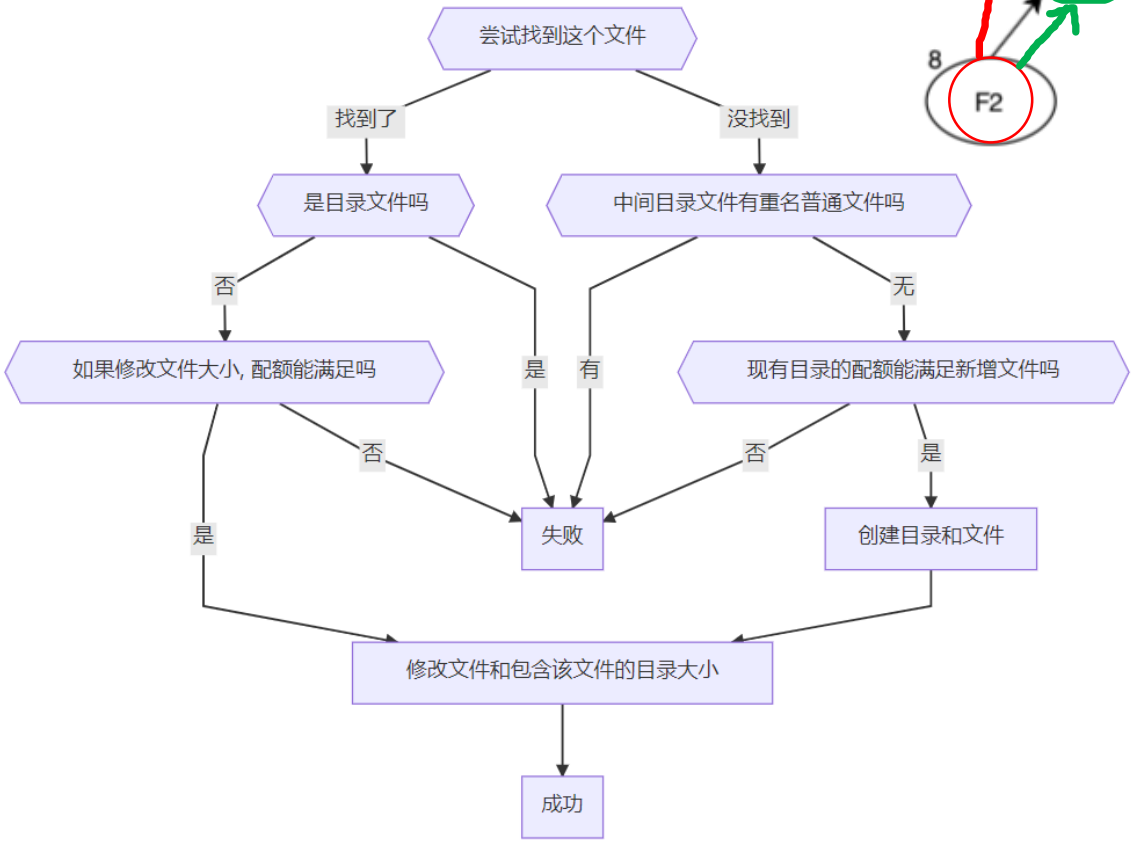
  - 直接做可能超时
    - 需要在结构体里开变量，保存后代总尺寸和目录下文件的总尺寸
    - 值发生变化时时刻根据变化量修改

```
struct file {
    bool isDir;
    long long LD, LR;
    long long SD, SR; // 实际尺寸
    map<string, file> children;

    long long fileSize;
};
```

实战演示2

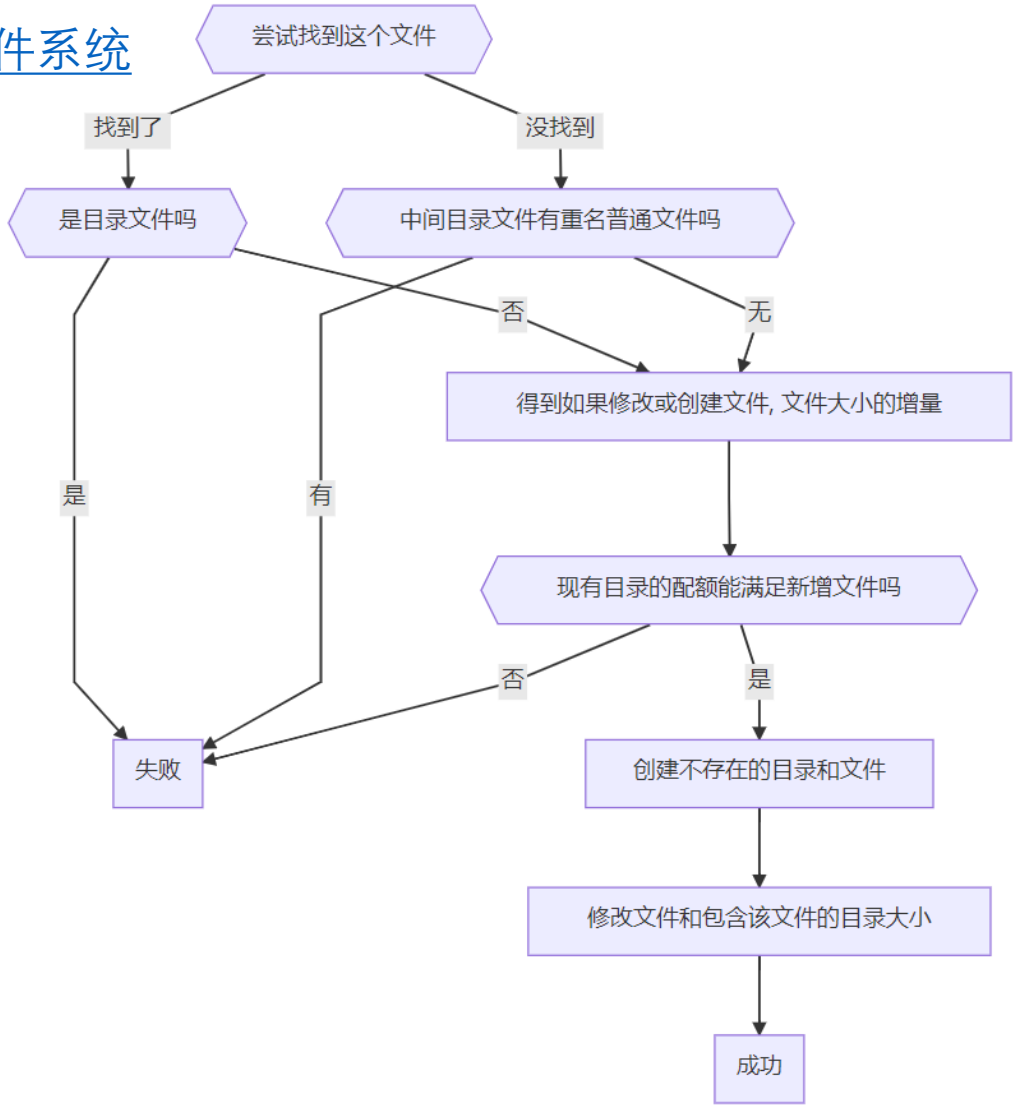
- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 创建普通文件（修改）





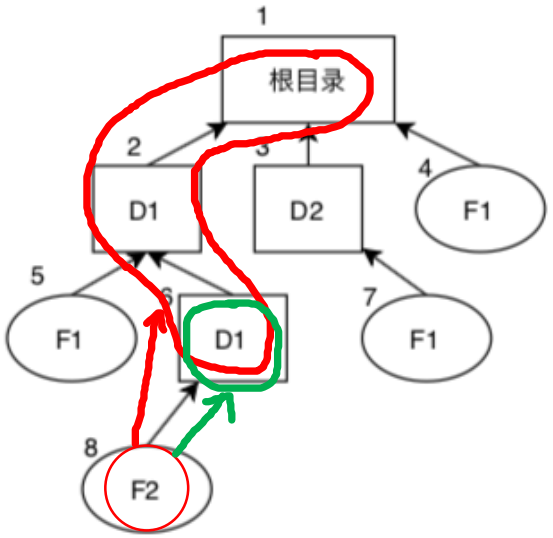
实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 创建普通文件
  - 消除一部分冗余



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



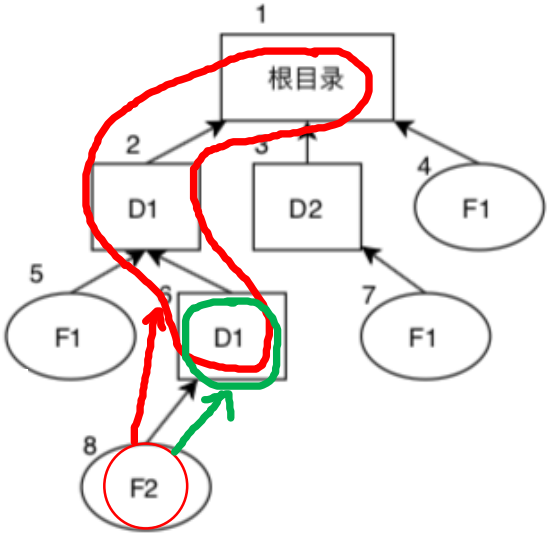
移除文件的指令有一个参数，是字符串，表示要移除的文件的路径。

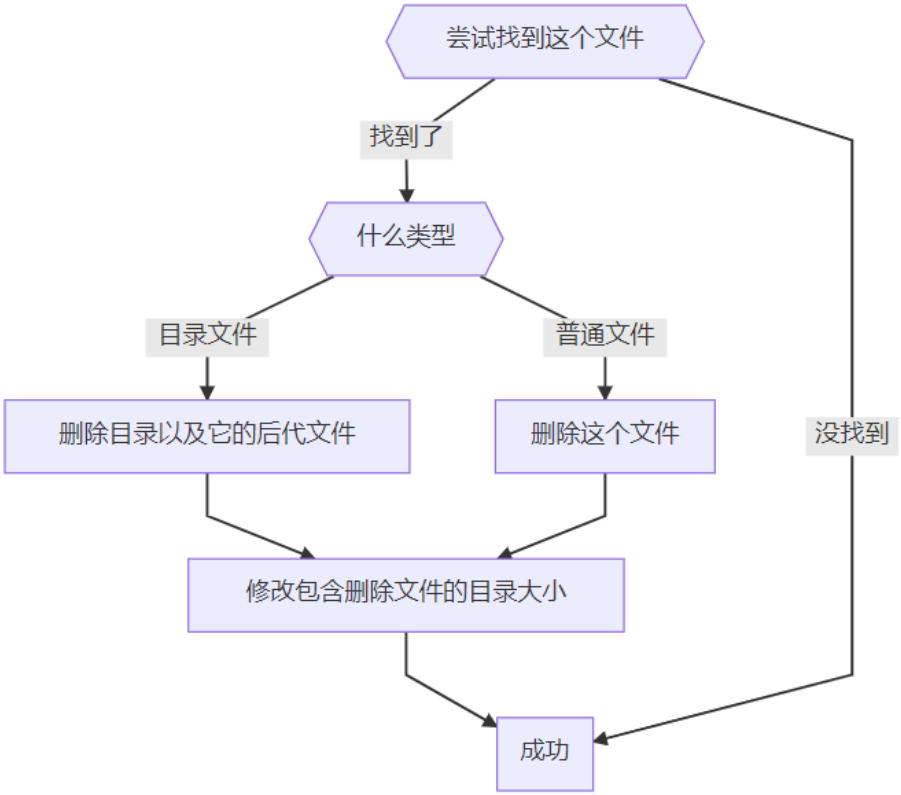
若该路径所指的文件不存在，则不进行任何操作。若该路径所指的文件是目录，则移除该目录及其所有后代文件。在上述过程中被移除的目录（如果有）上设置的配额值也被移除。

该指令始终认为能执行成功。

实战演示2

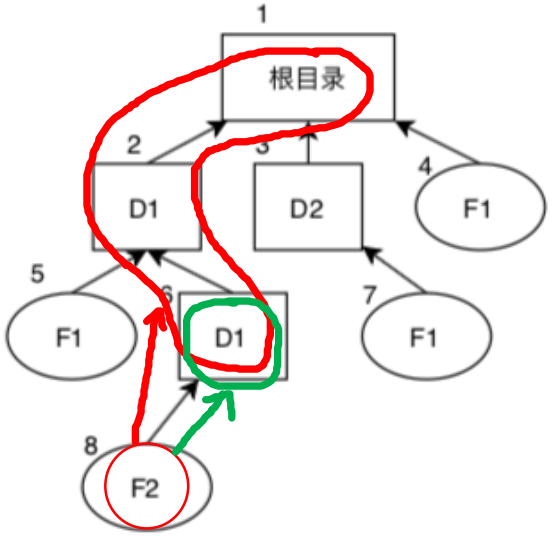
- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 移除文件





实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 设置配额




```
Q <file path> <LD> <LR>
```

设置配额值的指令有三个参数，是空格分隔的字符串和两个非负整数，分别表示需要设置配额值的目录的路径、目录配额和后代配额。

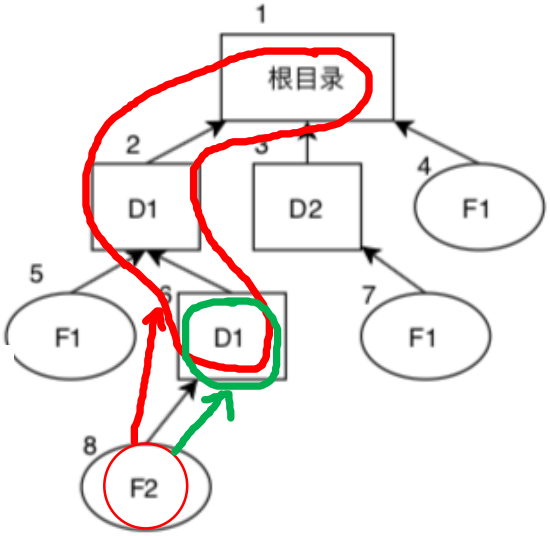
该指令表示对所指的目录文件，分别设置目录配额和后代配额。若路径所指的文件不存在，或者不是目录文件，则该指令执行不成功。

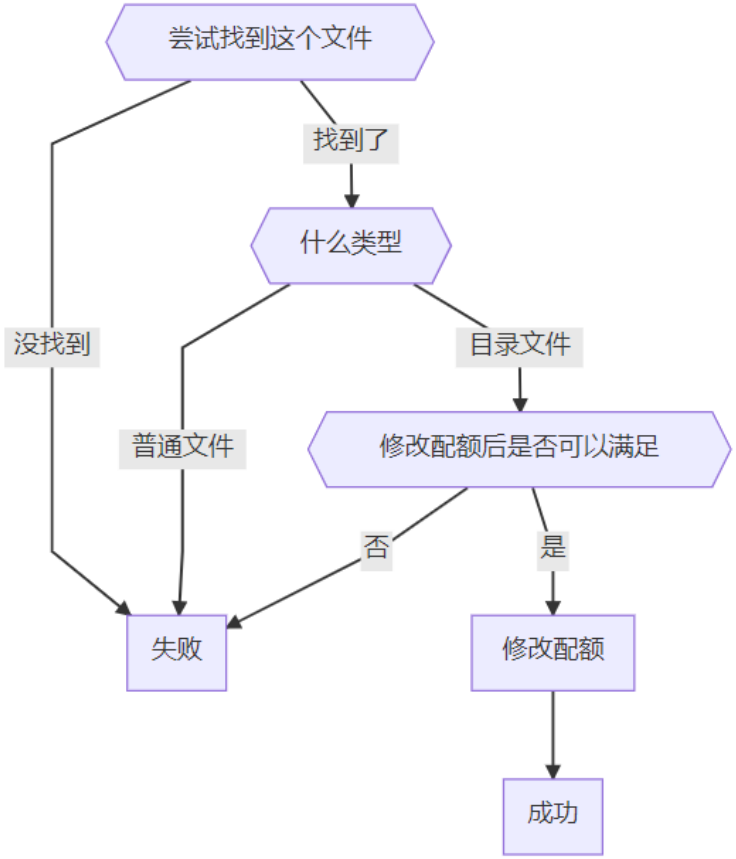
若在该目录上已经设置了配额，则将原配额值替换为指定的配额值。

特别地，若配额值为 0，则表示不对该项配额进行限制。若在应用新的配额值后，该文件系统配额变为不满足，那么该指令执行不成功。

实战演示2

- 题目：CSP202012-T3 带配额的文件系统
- 题目分析：
  - 设置配额





实战演示2

- 题目：

  CSP202012-T3 带配额的文件系统

- 部分代码提示：

```
// 将路径字符串分割成路径上各个目录，文件的名字
vector<string> parsePath(string &path) {
    stringstream ss(path);
    string fileName;
    vector<string> res;

    while (getline(ss, fileName, '/')) {
        res.push_back(fileName);
    }
    return res;
}
```

```
// 查找一个文件各级目录的指针
vector<file *> findFile(vector<string> &fileName) {
    vector<file *> filePos;
    file *cur = &root;
    filePos.push_back(cur);
    for (int i = 1; i < fileName.size(); i++) {
        if (cur->isDir == false or cur->files.count(fileName[i]) == 0) {
            break;
        }
        cur = &cur->files[fileName[i]];
        filePos.push_back(cur);
    }
    return filePos;
}
```

实战演示2

- 部分代码

提示：

```
bool create() {
    string path;
    long long fileSize;
    cin >> path >> fileSize;
    auto fileNames = parsePath(path);
    auto filePtrs = findPath(fileNames);
    long long inc;
    if (fileNames.size() == filePtrs.size()) { // 找到了文件
        if (filePtrs.back()->isDir) { // 是目录
            return false;
        } else { // 是普通文件
            inc = fileSize - filePtrs.back()->fileSize;
        }
    } else { // 没找到文件
        if (filePtrs.back()->isDir == false) { // 中间要建立的目录有重名普通文件
            return false;
        } else { // 没问题
            inc = fileSize;
        }
    }
    for (int i = 0; i < filePtrs.size(); i++) { // 检查配额
        if (filePtrs[i]->SR + inc > filePtrs[i]->LR) {
            return false;
        }
        if (i == fileNames.size() - 2) {
            if (filePtrs[i]->SD + inc > filePtrs[i]->LD) {
                return false;
            }
        }
    }
    // 后面略
```

实战演示2

- 题目：

  CSP202012-T3 带配额的文件系统

- 部分代码提示：

```
bool remove() {
    string path;
    cin >> path;
    auto fileNames = parsePath(path); // 各级目录或文件名称
    auto filePtrs = findPath(fileNames); // 各级目录或文件指针
    if (fileNames.size() != filePtrs.size()) { //检查文件是否存在
        return true;
    }
    if (filePtrs.back()->isDir) { // 待删除的是目录
        for (auto &i : filePtrs) {
            i->SR -= filePtrs.back()->SR;
        }
    } else { // 待删除的是普通文件
        for (auto &i : filePtrs) {
            i->SR -= filePtrs.back()->fileSize;
        }
        filePtrs.end()[-2]->SD -= filePtrs.back()->fileSize;
    }
    filePtrs.end()[-2]->children.erase(fileNames.back()); // 从倒数第二级移除倒数第一级这个孩子
    return true;
}
```

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