hw_1 作业完成思路分享

前言

在完成能够顺利通过第一次作业中测的代码以及搭评测机的过程中，我受到了许多助教学长或者同学的热情帮助。这些帮助对我作业的顺利完成是十分重要的，所以我也想将自己的经验分享给各位同学，如果有不对的地方或者更好的思路也欢迎大家找我交流。

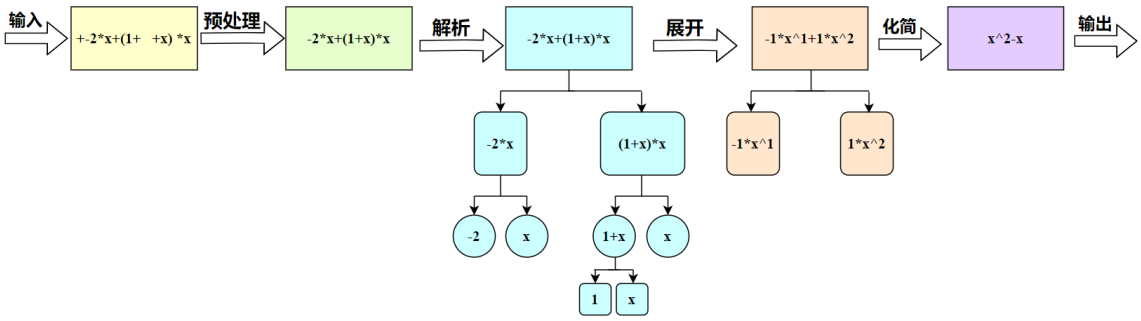
架构设计分析

本次作业是让我们展开一个最多嵌套一层括号的单变量多项式，多项式中项的因子只能是常数因子、幂函数因子、表达式因子，最后在正确展开的前提下再比较表达式的长度。

为了完成本次作业，我想我们至少需要考虑完成如下几个任务：

- 分析题干数据结构并**建类**
- **预处理**输入的字符串，得到方便后续操作的字符串
- **解析**预处理后的字符串
- **展开**有括号的表达式，得到一个多项式
- **优化**多项式并输出

为了能够让同学们更好地理解这几个任务，在这里给出一个样例：



建类

首先分析题目的输入，可以很明显地发现题目的输入有**表达式**→**项**→**因子**的结构，所以我们可以很自然地建立三个类：`Expr`,`Term`,`Factor`。而`Factor`又有三种情况，因此又可以很自然地建立它的三个子类：`ExprFactor`,`PowFactor`,`ConFactor`。并且我建议将`Factor`定义为抽象类。

其次我们分析题目的输出，题目让我们输出一个**去除括号**的表达式。那实际上就是要输出多个形如`a*x^b`的项的和或者差。所以很自然地会想到建立一个`Mono`类以及`Poly`类，分别用来存储符合上述形式的项以及项的集合。

预处理

在预处理阶段我主要做了两件事情：

- 将输入字符串中的所有空白字符替换为空串。
- 将题目中连续的`+`与`-`合并起来（如`+--`替换成`+`）。

解析

由于各位助教以及其他的一些学长在之前就向我大力推荐了**递归下降**的方法，本着拿来主义的思想，我并没有多少犹豫地选择了这个方法进行解析（不过确实是有人用正则表达式做的，但是感觉可扩展性太低了并且不容易debug，而且更重要的是使用递归下降是可以很容易地实现未来几乎一定会出现的**多层括号嵌套**）。具体的方法大家相信大家应该在`oolens`的推送以及训练题中学到了，在这里我提一下我的一些特殊的处理：

- 在识别到数字时将前导0去除。
- 对于加减号来说，如果加减号前是`^`或者`*`，那么可以发现其后必然是一个数字，所以不妨将其识别成后面数字的一部分；否则便将其作为正常的加减号，值得一提的是对于表达式或表达式因子中第一项前可能出现的正负号也会认为是正常的加减号。
- 项之间可能是由`+`或者`-`号连接的，我们当然可以在`Expr`类中存储这种排列的信息，但是我感觉这样做的话不太美观，在处理时也会变得更加复杂。所以我们不妨在这里认为项之间全都是相加的关系，而`-`的存储则由`Term`类负责（其实就是项这一个整体的符号`sign`）。

展开

将有括号的表达式展开的思路我主要是参考了一位学长的博客（在此大力推荐`hyggge`学长的博客），他是在`Expr`,`Term`,`ConFactor`,`PowFactor`,`ExprFactor`中都实现了`toPoly()`方法，并在`Poly`类中实现了`addPoly()`,`mulPoly()`,`mulPoly()`方法。这里的`toPoly()`理解起来比较困难，所以我在这里简述一下`hyggge`学长的展开思路：

- 对于`ConFactor`与`PowFactor`，转化为仅含有一个`Mono`对象的`Poly`
- 对于`ExprFactor`,我们假设此时已经实现了`expr.toPoly()`，那么可以先后使用`expr.toPoly()与powPoly``
- 对于`Term`,仅需使用`mulPoly`将`Factor`调用`topoly()`的结果乘起来即可
- 对于`Expr`,仅需使用`addPoly`将`Term`调用`topoly()`的结果乘起来即可

这种方法为什么使用起来体验极好呢？我想是因为`hyggge`学长抓住了题目的核心需求（让我们输出由若干最基本的幂函数单项式组成的多项式），这使得代码的可扩展性变得十分高，即使是以后新增其他东西（比如三角函数、多变量、自定义函数）也是完全适用的。我想这种获取客户的需求的能力是我们应当在这学期的面向对象课程以及今后的软件工程课程中着重培养的。

在这里我还想说一下几点我与该学长展开思路不一样或者学长没有详细谈到的地方：

- 我并没有实现学长所提到`Poly`类中的`negate()`方法（将`Poly`中所有的单项式的系数取反），而是构建一个仅存有`-1*x^0`的`Poly`对象，然后利用已经实现的`mulPoly`将其与要取反的`Poly`相乘。

> 实际上我的`Poly`类和`Mono`类都有不止一种构造方法,这样可以有效地简化代码，增大代码的可读性：
>
> - `Poly`类中有3种构造方法：
>   - 无传入参数： 则属性`ArrayList<Mono> monos`为空
>   - 参数为一个`int`类型变量`a`：`monos`之中仅有一项`a*x^0`
>
> ```
> // 构造一个 常数poly, 用于得到1和-1
> public Poly(int a) {
>     Mono mono = new Mono(String.valueOf(a), expStr: "0");
>     this.monos = new ArrayList<>();
>     this.monos.add(mono);
> }
> ```

    - 参数为一个`ArrayList<Mono>`类型的变量：则将其送给属性`monos`
  - `Mono`类中有2种构造方法，由于`Mono`类有两个属性`coef`(单项式系数)和`exp`（单项式指数），而单项式系数和单项式指数在代码中有时会以两个字符串的形式出现，有时又会以两个数的形式出现，所以不如直接写两个构造方法，一个接受的是字符串类型的两个参数，另一个是接受以`BigInteger`类型的两个参数。

- 另外，我在代码中实现了`Poly`的深拷贝方法，是因为才开始担心会有共享数据的问题，后来发现似乎没有这个问题，但是又考虑到后续迭代作业会有相关的问题所以还是保留下来了。

> 通过上学期先导课程的学习我们知道浅拷贝只是复制对象本身，对象的属性和包含的对象不做复制。深拷贝则对对象本身复制，同时对**对象的属性**也进行复制。实际上，深浅拷贝的本质区别是对象或者说对象属性的**内存地址**是否一样，一样的话为浅拷贝，不一样的话为深拷贝。
>
> 深拷贝有许多种方法，比如qs学长就是使用的序列化和反序列化实现的深拷贝，而我是使用`for`循环实现的深拷贝，即新建一个`ArrayList<Mono>`变量，遍历`Poly`对象中的`monos`，将遍历到的每一个`Mono`对象的属性取出来，然后新建一个`Mono`对象并将其存入`ArrayList<Mono>`变量中。

优化

如果上述过程没有bug的话，那么我们现在会得到一个符合如下格式的多项式$\sum_{i=1}^{n}$coef$_i$*x*^exp$_i$，并且其中的指数不会相同（因为在`addPoly`中就可以实现**同类项合并**）

优化时我主要考虑了如下几点：

- 若coef$_i$为1，则化为x*^exp$_i$
- 若coef$_i$为-1，则化为-x*^exp$_i$
- 若exp$_i$为0，则化为coef$_i$
- 若exp$_i$为1，则化为coef$_i$*x
- 删除coef$_i$为0的项，当`Poly`中`monos`被删空时（即表达式本身是0的情况），向其中加入一个0
- 由于会出现`-x+1`比`1-x`长的情况，因此这一步需要改变输出顺序使得第一项尽可能为正

其他值得注意的地方

- 注意去除前导0时不要将本身等于0的数字全部去掉
- 注意0^0的值应取
- 删除值为1的项时不要将表达式删空

数据生成器

我的数据生成器可以认为是直接翻译的题目中的形式化表达，同时为了控制嵌套层数，我将可嵌套层数`deep`作为参数传入了`genDate()`,每当调用一次`getExprFactor()`时传入`deep-1`。理论上可以生成所有可能的合法数据。

二、设定的形式化表述

- 表达式 → 空白项 [加减 空白项] 项 空白项 | 表达式 加减 空白项 项 空白项
- 项 → [加减 空白项] 因子 | 项 空白项 '*' 空白项 因子
- 因子 → 变量因子 | 常数因子 | 表达式因子
- 变量因子 → 幂函数
- 常数因子 → 带符号的整数
- 表达式因子 → '(' 表达式 ')' [空白项 指数]
- 幂函数 → 'x' [空白项 指数]
- 指数 → '^' 空白项 ['+'] 允许前导零的整数 **(注：指数一定不是负数)**
- 带符号的整数 → [加减] 允许前导零的整数
- 允许前导零的整数 → ('0'|'1'|'2'|...|'9'){'0'|'1'|'2'|...|'9'}
- 空白项 → {空白字符}
- 空白字符 → （空格） | `\t`
- 加减 → '+' | '-'

另外，在该数据生成器的可嵌套层数取大于3的情况时，有小概率生成极其复杂的数据，以至于仅是跑完这一轮测试就需要数分钟时间，主要原因是因为可能会出现一个项数很大的表达式因子的指数也很大的情况，所以我认为因当将指数与表达式因子的项数联系起来，使其呈现负相关的关系。

相关博客分享

在完成本次作业时，我发现许多朋友与我讨论的问题其实大多都可以在学长博客中找到很优秀的解决方案，而我认为这种向更优秀的人借鉴学习的行为是值得提倡并且是高效的，所以我在此列出我在过去一个学期收集到的面向对象有关的学长博客（排名不分先后），供大家参考：

> 钟鼓楼（我最喜欢的博客之一，作者也是个非常优秀非常有趣的学长，他的很多话总是能让我茅塞顿开）
>
> Hyggge's Blog （给了我好多思路，文章朴实易懂）
>
> 春日野草
>
> Musel's blog(一个很优秀的学姐，她做的图很值得学习)
>
> Coekjan
>
> volca's blog
>
> 菠菜白菜花菜
>
> Tan's Blog
>
> BUAA-Wander's Blog
>
> Frida_h
>
> Bluebean’s Blog(zh学长的作业架构我也好喜欢)
>
> Xlucidator
>
> 零落闲拾
>
> saltyfishyjk's Blog(写的非常详细)

> lethan
>
> 圆*
>
> MarkDay
>
> Stargazer

由于个人时间有限，必然还有很多优秀的学长博客没有被我发现，也不能对这些博客给出详细的分类介绍，所以欢迎大家补充。