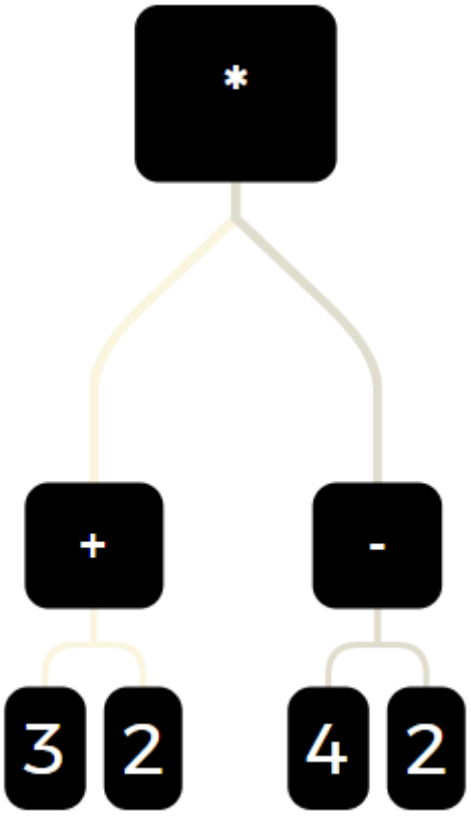
后缀（逆波兰）表示法 Reverse Polish notation (abbr. RPN)

先修要求

1. 树的遍历
2. 栈

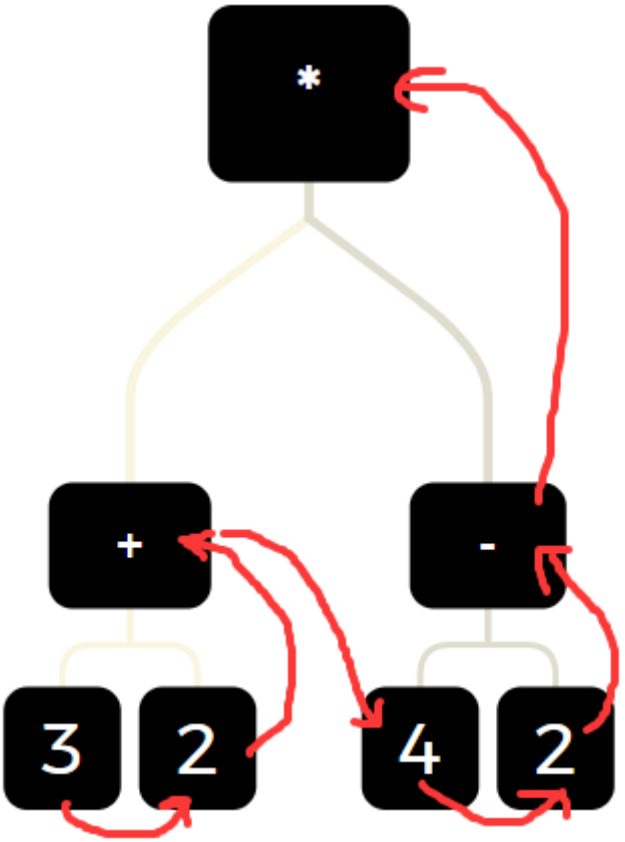
中缀表达式

平时生活中，我们最常见的表达式是“中缀表达式”，简单理解就是运算符（operator）放在两个算子（operand）之间，例如简单的 (3 + 2) * (4 - 2) 这符合我们平时的阅读习惯，这里我们如果引入树结构来存储中缀表达式，就是如下形式：



那么实际的运算顺序是什么样的呢？这里细心的朋友们会发现，可以用树的遍历来进行计算。那究竟是前后中序遍历的哪一种呢？

1. 如果是前序遍历，依次获得的算子和运算符为+ 3 2 - 4 2*，如果以“先获得运算符，再连着计算后两个算子”的原则，那么乘号*就不好办了，所以前序是不可以的。
2. 如果是中序遍历，依次获得的算子和运算符为3 + 2 4 - 2 *，似乎看起来像样了，我们可以以“获得运算符后计算相邻左右两个算子”的原则运算，但是*的问题仍然不好解决。
3. 如果是后序遍历，依次获得的算子和运算符为 3 2 + 4 2 - *，发现“要运算的算子永远在运算符之前”！



也就是：

1. 获得算子3,2
2. 选择运算符+，运算结果为5
3. 接下来获得了5,4,2三个算子，那运算哪两个算子呢？答案是：遇到运算符时，计算运算符的前两个算子
4. 获得算子4,2
5. 获得运算符-，运算结果为2
6. 获得算子5,2
7. 获得运算符*，结果为10

于是乎，后缀表达式就诞生了。

后缀表达式

后缀表示法，又名逆波兰表示法，用于实现含不同运算优先级的表达式的运算。最大的优点就是不带括号，方便计算机直接运算。

中缀表达式到后缀表达式

如何实现中缀表达式到后缀表达式的转换呢？答案就是使用栈stack，遵循以下规则：

注意：栈结构与表达式为两个不同的东西，栈pop出来的元素push_back到表达式后

1. 算子直接放入表达式
2. 左括号直接入栈
3. 如果运算符优先级小于等于栈顶运算符，那么pop直到栈顶元素大于该运算符，将运算符push入栈
4. 如果运算符为右括号，则一直pop到上一个左括号

文字模拟一下(3+2)*(4-2)

1. (

栈底 (栈顶

表达式

中缀表达式3+2)*(4-2)

---

2. 直接push_back 3

栈底 (+ 栈顶

表达式3

中缀表达式3+2)*(4-2)

---

3. +

栈底 (+ 栈顶

表达式 3

中缀表达式2)*(4-2)

---

4. 直接push_back 2

栈底 (+ 栈顶

表达式3 2

中缀表达式)*(4-2)

---

5. 遇到)，pop到前一个左括号

栈底 栈顶

表达式3 2 +

中缀表达式*(4-2)

---

6. *

栈底 * 栈顶

表达式3 2 +

中缀表达式(4-2)

---

7. (

栈底 * (栈顶

表达式3 2 +

中缀表达式4-2)

---

8. 直接push_back 4

栈底 * (栈顶

表达式3 2 + 4

中缀表达式-2)

---

9. -

栈底 * (-栈顶

表达式3 2 + 4

中缀表达式2)

---

10. 直接push_back 2

栈底 * (- 栈顶

表达式3 2 + 4 2

中缀表达式)

---

11. 遇到)，pop到前一个左括号

栈底 * 栈顶

表达式3 2 + 4 2 -

中缀表达式

---

12. 中缀表达式遍历完成，pop直到栈空

栈底 栈顶

表达式3 2 + 4 2 - *

中缀表达式

中缀表达式构建完毕！接下来只要依照 *两算子一运算符* 的规则求表达式即可！