秦九韶算法

Catalog

New Words

- **polynomial [ˌpɒlɪ'nəʊmɪəl] --adj.多项式的. --n.多项式**
  - a polynomial expression 多项式
  - quadratic polynomial 二次多项式

Content

1. 秦九韶算法

- 怎么求多项式 $f(x)=x^5+x^4+x^3+x^2+x+1$ 当 $x=5$ 时的值呢? 一个自然的做法是把 5 代入多项式 $f(x)$, 计算各项的值, 然后把它们加起来. 这时, 我们一共做了 1 + 2 + 3 + 4 = 10 次乘法运算(1), 5 次加法运算.
  - (1):

$$\begin{cases} x^2=5\times 5, & 1\text{ 次乘法} \\ x^3=5\times 5\times 5 & 2\text{ 次乘法} \\ x^4=5\times 5\times 5\times 5 & 3\text{ 次乘法} \\ x^5=5\times 5\times 5\times 5\times 5 & 4\text{ 次乘法} \end{cases}$$

  另一种做法是先计算 x^2 的值, 然后依次计算 $x^2\times x,((x^2\times x)\times x)\times x$ 的值, 这样每次都可以利用上一次计算的结果. 这时, 我们一共做了 4 次乘法运算, 5 次加法运算.

  第二种做法与第一种做法相比, 乘法的运算次数减少了, 因而能够提高运算效率. 对于计算机来说, 做一次乘法运算所用的时间比做一次加法运算要长得多, 所以采用第二种做法, 计算机能更快地得到结果.

  有没有更有效的算法呢?

  我国南宋时期的数学家秦九韶(约 1202-1261) 在他的著作《数书九章》中提出了下面的算法.

  把一个 n 次多项式 $f(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$ 改写成如下形式:

$$
\begin{aligned}
f(x) &= a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \\
&= (a_n x^{n-1} + a_{n-1}x^{n-2} + \ldots + a_1)x + a_0 \\
&= ((a_n x^{n-2} + a_{n-1}x^{n-3} + \ldots + a_2)x + a_1)x + a_0 \\
&= \ldots \\
&= (\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0
\end{aligned}
$$

求多项式的值时, 首先计算最内层括号内一次多项式的值, 即

$$
\begin{aligned}
v_0 &= a_n \\
v_1 &= v_0 x + a_{n-1}
\end{aligned}
$$

然后由内向外逐层计算一次多项式的值, 即

$$
\begin{aligned}
v_2 &= v_1 x + a_{n-2}, \\
v_3 &= v_2 x + a_{n-3}, \\
&\ldots \\
v_n &= v_{n-1} x + a_0,
\end{aligned}
$$

这样, 求 n 次多项式 $f(x)$ 的值就转化成求 n 个一次多项式的值.

上述方法称为 秦九韶算法. 直到今天, 这种算法仍是多项式求值比较先进的算法.

- **例(1)**: 已知一个 5 次多项式为

  $$f(x) = 4x^5 + 2x^4 + 3.5x^3 - 2.6x^2 + 1.7x - 0.8$$

  用秦九韶算法求这个多项式当 $x = 5$ 时的值.

  Additional Info: $f(x) = a_5 x^5 + a_4 x^4 + a_3 x^3 - a_2 x^2 + a_1 x - 0.8$

  解: 根据秦九韶算法, 把多项式改写成如下形式:

  $$
  \begin{aligned}
  f(x) &= (4x^4 + 2x^3 + 3.5x^2 - 2.6x + 1.7)x - 0.8 \\
  &= ((4x^3 + 2x^2 + 3.5x - 2.6)x + 1.7)x - 0.8 \\
  &= (((4x^2 + 2x + 3.5)x - 2.6)x + 1.7)x - 0.8 \\
  &= ((((4x + 2)x + 3.5)x - 2.6)x + 1.7)x - 0.8
  \end{aligned}
  $$

  按照从内向外的顺序, 依次计算一次多项式当 $x = 5$ 时的值:

  $$
  \begin{aligned}
  v_0 &= 4; \\
  v_1 &= 4 \times 5 + 2 = 22; \\
  v_2 &= 22 \times 5 + 3.5 = 113.5; \\
  v_3 &= 113.5 \times 5 - 2.6 = 564.9; \\
  v_4 &= 565.9 \times 5 + 1.7 = 2826.2; \\
  v_5 &= 2826.2 \times 5 - 0.8 = 14130.2.
  \end{aligned}
  $$

  所以, 当 $x = 5$ 时, 多项式的值等于 14130.2.

2. 一维多项式求值算法实现 (秦九韶算法)

- 算法分析:

观察上述秦九韶算法中的 n 个一次式, 可见 v_i 的计算要用到 v_{i-1} 的值 (tip: 即 v_1的值要用到 v_0 的值, v_2的值要用到v_1, v_3的值要用到v_2, ...), 若令 $v_0 = a_n$, 我们可以得到下面的公式:

$$\begin{cases} v_0 = a_n \quad (1) \\ v_i = v_{i-1}x + a_{n-i} \quad (i = 1, 2, 3, \ldots, n) \quad (2). \end{cases}$$

Tip: 如果 `(2)` 式不太明白, 可以把 $i = 1, 2, , 3 \ldots, n$ 分别代入:

$$\begin{aligned} v_0 &= a_n \\ v_1 &= v_0 x + a_{n-1} \\ v_2 &= v_1 x + a_{n-2}, \\ v_3 &= v_2 x + a_{n-3}, \\ &\ldots \\ v_n &= v_{n-1} x + a_0, \end{aligned}$$

可以看到, 和上面的 `秦九韶算法` 中的红色执行步骤是一样的.

这是一个在秦九韶算法中反复执行的步骤, 因此可用循环结构来实现.

算法步骤如下:

- (1) 第 1 步, 输入多项式次数 n, 最高次项的系数 a_n 和 x 的值.
- (2) 第 2 步, 将 v 的值初始化为 a_n, 将 i 的值初始化为 $n - 1$.
- (3) 第 3 步, 输入 i 次项的系数 a_i.
- (4) 第 4 步, $v = vx + a_i, i = i - 1$.
- (5) 第 5 步, 判断 i 是否大于或等于 0, 若是, 则返回到第 3 步; 否则, 输出多项式的值 v.

- JavaScript 实现的源码如下:

```
// - 一维多项式算法
(function() {
    function onePolynomial (obj) {
        const {n, x, ary} = obj;

        // - f(x) = a_{5}x^5 + a_{4}x^4 + a_{3}x^3 - a_{2}x^2 + a_{1}x - 0.8
        // - 为什么 `i = n - 2` 请参考上面 `秦九韶算法` 的红色部分, 配合
        //   `例(1)` 便可明白, 此处的 i = 4, 对应 上面的 `v_1`.
        let i = n - 2;

        // - result 即算法讲解中的 `v_0`
        let result = ary[n - 1];
        console.log('result: ', result);
        for (; i >= 0; i--) {
            result = result * x + ary[i]
        }
        return result;
    }

    // - E.g: 以 5 次多项式 f(x) = 4x^5 + 2x^4 + 3.5x^3 - 2.6x^2 + 1.7x - 0.8 为
例.
    // - obj 对象有 3 个属性:
    //     + (1) n: 一维多项式的项数(即: 多项式由多少项组成, 上面的 f(x) 有 6 项)
    //     + (2) x: 指定的自变量的值(即: 一般为 x 的值)
```

```
    //     + (3) ary: 存放 n - 1 次多项式的 n 个系数的数组
    const obj = {
        n: 6,
        x: 5,
        ary: [-0.8, 1.7, -2.6, 3.5, 2, 4]
    };
    console.log(onePolynomial(obj));
})();
```