CSP 复赛复习 - 基础算法(2)

4. 数值处理算法

高精度算法概述

适用场景：处理超过 2^{64} 的大整数运算

存储方式：用数组存储每一位数字，低位在前，高位在后

时间复杂度：$O(n)$，其中 n 为数字位数

高精度加法

算法思想：模拟竖式加法，逐位相加并处理进位

```
char a2[N], b2[N];
int a[N], b[N], c[N];
int main()
{
    cin >> a2 >> b2; /* 读入两个高精度加数 */
    int len1 = strlen(a2), len2 = strlen(b2);
    int lenMax = max(len1, len2);

    /* 把两个加数从字符数组转为整型数组，逆序存放 */
    for (int i = 0; i < len1; i++)
        a[len1 - i - 1] = a2[i] - '0';
    for (int i = 0; i < len2; i++)
        b[len2 - i - 1] = b2[i] - '0';

    // 逐位相加
    for (int i = 0; i < lenMax; i++)
        c[i] = a[i] + b[i];

    // 处理进位
    for (int i = 0; i < lenMax; i++)
        c[i + 1] += c[i] / 10, c[i] %= 10;

    // 处理最高位进位
    if (c[lenMax])
        lenMax++;

    // 输出结果（高位在前）
    for (int i = lenMax - 1; i >= 0; i--)
        cout << c[i];
}
```

高精度减法

算法思想：模拟竖式减法，逐位相减并处理借位

```
char a2[N], b2[N], c2[N];
int a[N], b[N], c[N];
int main()
{
    cin >> a2 >> b2;
    int len1 = strlen(a2), len2 = strlen(b2);
    /* 判断是否为负数 */
    if (len1 < len2 || (len1 == len2 && strcmp(a2, b2) < 0))
    {
        cout << '-';
        /* 交换被减数和减数 */
        strcpy(c2, a2),strcpy(a2, b2),strcpy(b2, c2);
    }
    // 重新获取长度
    len1 = strlen(a2), len2 = strlen(b2);

    // 逆序存放到整型数组
    for (int i = 0; i < len1; i++)
        a[len1 - i - 1] = a2[i] - '0';
    for (int i = 0; i < len2; i++)
        b[len2 - i - 1] = b2[i] - '0';

    // 逐位相减
    for (int i = 0; i < len1; i++)
        c[i] = a[i] - b[i];

    // 处理借位
    for (int i = 0; i < len1; i++)
    {
        if (c[i] < 0)
            c[i] += 10, c[i + 1]--;
        c[i] %= 10;
    }

    // 处理前导0
    while (len1 && c[len1] == 0)
        len1--;

    // 输出结果
    for (int i = len1; i >= 0; i--)
        cout << c[i];
}
```

高精度乘法

算法思想：模拟竖式乘法，逐位相乘并累加

时间复杂度：$O(n \times m)$，其中 n,m 为两个数字的位数

```cpp
char a1[N], b1[N];
int a[N], b[N], c[N];
int main()
{
    cin >> a1 >> b1;
    int len1 = strlen(a1), len2 = strlen(b1);

    /* 逆序存放到整型数组中 */
    for (int i = 0; i < len1; i++)
        a[len1 - i] = a1[i] - '0';
    for (int i = 0; i < len2; i++)
        b[len2 - i] = b1[i] - '0';

    /* 乘法运算 */
    for (int i = 1; i <= len2; i++)     // 遍历乘数2 b
        for (int j = 1; j <= len1; j++) // 遍历乘数1 a
            c[i + j - 1] += (b[i] * a[j]);

    /* 进位处理 */
    for (int i = 1; i <= len1 + len2; i++)
        c[i + 1] += c[i] / 10, c[i] %= 10;

    /* 找到最高位的有效位 */
    int len = len1 + len2;
    while (len > 1 && c[len] == 0)
        len--;

    /* 逆序输出 */
    for (int i = len; i >= 1; i--)
        cout << c[i];
}
```

高精度除法

算法思想：模拟竖式除法，从高位到低位逐位处理

```
char a1[N];
int a[N], c[N], b;
int main()
{
    cin >> a1 >> b;
    /* 正序存放到整型数组中 */
    int len1 = strlen(a1);
    for (int i = 0; i < len1; i++)
        a[i] = a1[i] - '0';

    long long g = 0; // 当前实际被除数
    for (int i = 0; i < len1; i++)
    {
        g = g * 10 + a[i];
        c[i] = g / b;
        g = g - c[i] * b;
    }

    // 去除前导0
    int len = 0;
    while (len < len1 - 1 && c[len] == 0)
        len++;

    // 输出结果
    for (int i = len; i < len1; i++)
        cout << c[i];
}
```

高精度算法关键要点

存储方式对比

运算类型	存储方式	处理顺序	输出方式
加、减、乘	逆序存储（低位在前）	从低位到高位	逆序输出（高位在前）
除法	正序存储（高位在前）	从高位到低位	正序输出

进位/借位处理

加法进位：

```
c[i + 1] += c[i] / 10;
c[i] %= 10;
```

减法借位：

```
if (c[i] < 0) {
    c[i] += 10;
    c[i + 1]--;
}
```

乘法进位：

```
c[i + 1] += c[i] / 10;
c[i] %= 10;
```

前导零处理

逆序存储：

```
while (len > 1 && c[len] == 0)
    len--;
```

正序存储：

```
while (len < total_len - 1 && c[len] == 0)
    len++;
```

高精度算法复杂度总结

运算类型	时间复杂度	空间复杂度	关键要点
加法	$O(n)$	$O(n)$	逐位相加，处理进位
减法	$O(n)$	$O(n)$	确保被减数更大，处理借位
乘法	$O(n \times m)$	$O(n + m)$	双重循环，注意进位处理
除法	$O(n)$	$O(n)$	从高位到低位，模拟竖式

复习要点

1. 熟练掌握各种高精度运算的实现方法
2. 注意不同运算的存储方式差异
3. 正确处理进位和借位
4. 及时去除前导零
5. 注意边界情况的处理

掌握高精度算法，轻松应对大数运算！