

高精度加减

高精度的主要思想是用数组模拟加减运算, 数组的每一位代表一位数字.

因为要实现减法, 所以就要实现正负数, 还有数字比较.

两个大整数进行比较, 可以先比较符号位, 再比较长度, 最后进行每一位的比较.

```
//a>b 返回1
int cmp(BigNumber a, BigNumber b)
{
    if(a.flag == -1 && b.flag == 1)
        return -1;
    if(a.flag == 1 && b.flag == -1)
        return 1;
    if(a.len != b.len)
    {
        if (a.len > b.len)
            return a.flag;
        return -1 * a.flag;
    }
    for(int i = a.len-1; i >= 0; i--)
    {
        if(a.d[i] > b.d[i])
        {
            return a.flag;
        }else if(a.d[i] < b.d[i])
        {
            return -1 * a.flag;
        }
    }
    return 1;
}
```

因为有负数的存在, 所以加减法需要考虑不同的情况.

```
BigNumber operator + (BigNumber b)
{
    if(this->flag == -1 && b.flag == -1)
    {
        return (this->qf() + b.qf()).qf();
    }
    if(this->flag == -1 && b.flag == 1)
    {
        return (b - this->qf());
    }
}
```

```

    }
    if(this->flag == 1 && b.flag == -1)
    {
        return *this - b.qf();
    }
    BigNumber c = *this;int i;
    for(i = 0;i < b.len;i++)
    {
        c.d[i] += b.d[i];
        if(c.d[i] >= 10)c.d[i]%=10,c.d[i+1]++; //进位
    }
    while(c.d[i] >= 10)c.d[i]%=10,c.d[i+1]++,i++; //进位
    c.len = max(len,b.len)+1;
    c.clean();
    return c;
}

BigNumber operator - (BigNumber b)
{
    if(this->flag == -1 && b.flag == -1)
    {
        return b.qf() - (*this).qf();
    }
    if(this->flag == -1 && b.flag == 1)
    {
        return (this->qf() + b).qf();
    }
    if(this->flag == 1 && b.flag == -1)
    {
        return *this + b.qf();
    }
    if(cmp(*this,b) == -1)
    {
        return (b-*this).qf();
    }
    BigNumber c = *this;int i;
    for(i = 0;i < b.len;i++)
    {
        c.d[i] -= b.d[i];
        if(c.d[i] < 0)c.d[i]+=10,c.d[i+1]--; //借位
    }
    while(c.d[i] < 0)c.d[i]+=10,c.d[i+1]--,i++; //借位
    c.len = max(len,b.len)+1;
    c.clean();
    return c;
}

```



-2

5 3

8

2

$$\begin{pmatrix} -1 & 2 \end{pmatrix}$$

1

-3

$-1 \ 5$

4

-6

 $3 - 5$

-2

8

 3^{-2}

1

5

5

[illegible][illegible][illegible][illegible]

111

999

1110

- 888

-111 -999

-1110

388

小结

在这个问题里使用了很多的技巧来简化代码编写, 这种思想是很实用的. 例如将数位倒着存入数组中, 将进位与计算分开处理, 还有把负数计算问题转化为两个正数的运算. 这个代码还可以很容易的改成其他进制的计算, 还可以很方便的添加其他运算.