大数运算

1 实验目的

由于不同语言中类型的限制，如C语言中的`int`，`long`和`long long`等能够表示的数值有限，因此需要自行实现对于大数的四则运算。

2 算法思路

2.1 大数存储

相较于通过`char*`字符串类型来存储数据的低效方式，整型数组的一位能够表示$2^{32}-1$个不同值。在实验中为了可读性，数组的每一位数值处于$[0, 999999999]$之间，即10^9进制。

同时，为了数据处理的方便，我采用了小端存储。

处理输入大数的代码如下：

```
int num1[1000];
char n1[1024] = {0};
for (int i = 0; i < 1000; i++) num1[i] = 0;
scanf("%s", n1);
for (int i = strlen(n1) - 1, d = 0; i >= 0; i--, d++) {
  num1[d / 9] += pow(10, d % 9) * (n1[i] - '0');
}
```

例如，输入`1234567890`后，`num1`值为`[234567890, 1, 0, ..., 0]`

2.2 大数加法

由于采用了小端存储的方式，因此数组从左至右遍历时恰好个位对齐，可以直接依次相加。而对于和大于10^9的位，需要进行进位处理。

实现函数如下：

```
void add(int* num1, int* num2, int* result) {
  for (int i = 0; i < 1000; i++) result[i] = 0;
  int len1 = len(num1), len2 = len(num2);
  int maxlen = len1 > len2 ? len1 : len2;
  for (int i = 0, flag = 0; i < maxlen + 1; i++) {
    int tmp = num1[i] + num2[i] + (flag ? 1 : 0);
    if (tmp >= BASE) {// BASE = 1000000000
      flag = 1;
      tmp -= BASE;
    } else
      flag = 0;
    result[i] = tmp;
```

```
    }
  }
```

2.3 大数减法

减法的思路与加法类似，为了操作方便，我们只做“大数”-“小数”的操作，对应当得到负值的结果取负。如果遇到需要借位的情况，则利用标识`flag`记录。

实现函数如下：

```
void minus(int* num1, int* num2, int f, int* result) {
  int len1 = len(num1);
  for (int i = 0; i < 1000; i++) result[i] = 0;
  if (isNoSmallerThan(num1, num2) && isNoSmallerThan(num2, num1)) return;
  if (isNoSmallerThan(num2, num1)) {
    minus(num2, num1, -1, result);
    return;
  }
  for (int i = 0, flag = 0; i < len1; i++) {
    int tmp = num1[i] - flag - num2[i];
    if (tmp < 0) {
      flag = 1;
      tmp += BASE;
    } else
      flag = 0;
    result[i] = tmp * f;
  }
}
```

2.4 大数乘法

思路参考了竖式计算，按照顺序依次将每一位的数字相乘，而后相加。此处由于涉及了两个较大整数的相乘，因此需要额外声明一临时数组`long temp[1000]`辅助计算。

实现函数如下：

```
void multi(int* num1, int* num2, int* result) {
  long temp[1000];
  long t1[1000], t2[1000];
  for (int i = 0; i < 1000; i++) {
    temp[i] = 0;
    t1[i] = num1[i];
    t2[i] = num2[i];
  }
  for (int i = 0; i < len(num1); i++) {
    for (int j = 0; j < len(num2); j++) {
```

```
      temp[i + j] += t1[i] * t2[j];
    }
  }
  for (int i = 0; i < llen(temp); i++) {
    if (temp[i] >= BASE) {
      temp[i + 1] += temp[i] / BASE;
      temp[i] = temp[i] % BASE;
    }
  }
  for (int i = 0; i < llen(temp); i++) {
    result[i] = (int)temp[i];
  }
}
```

2.5 大数除法

最简单的求解方式是对输入的被除数与除数进行相减操作，直到余数小于除数位置，此时被减去的次数即为商，所剩的差值为余数。

该算法可以通过循环的方式暴力得出，但仍有很大的优化空间。例如，可以将被除数与除数左对齐相减，接下来用余数与剩余部分进行组合来循环这个过程，直到最终的余数小于除数，最后将各部分的商值拼接得到最终的商。

具体实现函数如下：

```
void divide(int* num1, int* num2, int* result, int* mod) {  // num1 / num2
  int len1 = len(num1), len2 = len(num2);
  if (len1 == 0) return;
  if (isNoSmallerThan(num2, num1)) return;

  int res[1000];
  for (int i = 0; i < 1000; i++) res[i] = 0;

  int tmp[1000], r1[1000], r2[1000];
  for (int i = 0; i < 1000; i++) {
    r1[i] = num1[i];
    r2[i] = num2[i];
  }
  while (1) {
    for (int i = 0; i < 1000; i++) {
      tmp[i] = num2[i];
      r2[i] = 0;
    }
    int l1 = len(r1), l2 = len(tmp);

    int delta = 0;
```

```
    if (r1[l1 - 1] >= tmp[l2 - 1])
      delta = 1;
    else
      delta = 2;

    for (int i = l1 - delta; l1 > l2 && i > l1 - l2 - delta; i--) {
      tmp[i] = tmp[i - (l1 - l2 - delta + 1)];
    }
    for (int i = l1 - l2 - delta; i >= 0; i--) {
      tmp[i] = 0;
    }

    minus(r1, tmp, 1, r2);
    result[l1 - l2 + 1 - delta]++;
    for (int i = 0; i < 1000; i++) r1[i] = r2[i];

    if (!isNoSmallerThan(r2, num2)) break;
  }
  for (int i = 0; i < len(r1); i++) mod[i] = r1[i];
  return;
}
```

3 样例测试

```
6666666666666666666666666666666666666
3333333333333333
6666666666666666666666666666666666666 + 3333333333333333 = 6666666666666666666669999999999999999
6666666666666666666666666666666666666 - 3333333333333333 = 6666666666666666666663333333333333333
6666666666666666666666666666666666666 * 3333333333333333 = 22222222222222221999999999999999999997777777777777778
6666666666666666666666666666666666666 / 3333333333333333 = 2000000000000000200000
6666666666666666666666666666666666666 mod 3333333333333333 = 66666
```

DH算法

1 算法原理

迪菲-赫尔曼密钥交换作为一种安全协议，可以让双方在完全没有对方任何预先信息的条件下通过不安全信道创建起一个密钥。

2 算法思路

对于Alice，令其选取的生成元为`g`，私钥为`a`，所选质数为`m`。则计算得到的公钥为$g^a \mod m$。

由于g^a的结果通常远大于一般变量类型，因此可以通过快速幂，在计算的过程中直接取模。具体函数实现如下：

```
long long modpow(long long a, long long b, long long m) {
  a %= m;
  long long res = 1;
  while (b > 0) {
    if (b & 1) res = res * a % m;
    a = a * a % m;
    b >>= 1;
  }
  return res;
}
```

计算结果也对应了公钥值。

3 样例测试

```
p: 353
g: 3
A's private key: 97
B's private key: 233
40
248
160
```