

一、算法原理

1. 当输入的数很大时，可采用字符串方式接收。
2. 拆成一位一位的数字，把它们存在一个数组中，一个数组元素表示一位数字

数组中是这样存储的：

如：s="3275865"

拆： 3 2 7 5 8 6 5

保存： a[7] a[6] a[5] a[4] a[3] a[2] a[1]

倒序存储原因：

在平常，数字从左到右依次为从高位到低位....可这里却与日常的习惯相反。

这是因为加法可能会产生进位，而数组在最前面加上数字是不可能的，但在尾巴处加上数字是好做的，所以，倒着放。

$$\begin{array}{r} a[7] a[6] a[5] a[4] a[3] a[2] a[1] \\ + \quad 0 \quad 0 \quad b[5] b[4] b[3] b[2] b[1] \\ \hline c[7] c[6] c[5] c[4] c[3] c[2] c[1] \end{array}$$

二、AcWing_791.高精度加法

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 100010;
5
6  int a[N], b[N];
7  int al, bl;
8  void add(int a[], int &al, int b[], int &bl) {
9      int t = 0;
10     al = max(al, bl);
11     for (int i = 0; i < al; i++) {
12         t += a[i] + b[i];
13         a[i] = t % 10;
14         t /= 10;
15     }
16     if (t) a[al] = 1;
17     while (al > 0 && a[al] == 0) al--;
18 }
19
20 int main() {
21     string x, y;
22     cin >> x >> y;
23     for (int i = x.size() - 1; i >= 0; i--) a[al++] = x[i] - '0';
24     for (int i = y.size() - 1; i >= 0; i--) b[bl++] = y[i] - '0';
25     add(a, al, b, bl);
```

```

26     for (int i = a1; i >= 0; i--) printf("%d", a[i]);
27     return 0;
28 }

```

三、AcWing_792.高精度减法

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5  int a[N], al;
6  int b[N], bl;
7
8  void sub(int a[], int &a1, int b[], int &b1) {
9      int t = 0;
10     a1 = max(a1, b1);
11     for (int i = 0; i < a1; i++) {
12         t = a[i] - b[i] - t;
13         a[i] = (t + 10) % 10;
14         t < 0 ? t = 1 : t = 0;
15     }
16     while (a1 > 0 && a[a1] == 0) a1--;
17 }
18
19 int main() {
20     string x, y;
21     cin >> x >> y;
22
23     // 负号
24     if (x.size() < y.size() || (x.size() == y.size() && x < y)) {
25         printf("-");
26         swap(x, y);
27     }
28
29     for (int i = x.size() - 1; i >= 0; i--) a[al++] = x[i] - '0';
30     for (int i = y.size() - 1; i >= 0; i--) b[bl++] = y[i] - '0';
31
32     sub(a, al, b, bl);
33
34     for (int i = al; i >= 0; i--) printf("%d", a[i]);
35     return 0;
36 }

```

四、AcWing_793.高精度乘法

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  const int N = 1e5 + 10;
5  int a[N], al;
6  int b[N], bl;
7  void mul(int a[], int &a1, int b[], int &b1) {
8      int t = 0;
9      int c[N] = {0};

```

```

10     for (int i = 0; i < a1; i++)
11         for (int j = 0; j < b1; j++)
12             c[i + j] += a[i] * b[j];
13     a1 += b1;
14     for (int i = 0; i < a1; i++) {
15         t += c[i];
16         a[i] = t % 10;
17         t /= 10;
18     }
19     // 前导0
20     while (a1 > 0 && a[a1] == 0) a1--;
21 }
22
23 int main() {
24     string x, y;
25     cin >> x >> y;
26     for (int i = x.size() - 1; i >= 0; i--) a[a1++] = x[i] - '0';
27     for (int i = y.size() - 1; i >= 0; i--) b[b1++] = y[i] - '0';
28
29     mul(a, a1, b, b1);
30     for (int i = a1; i >= 0; i--) printf("%d", a[i]);
31     return 0;
32 }
33

```

注：网上对于高精度乘法，一般有高精乘低精 和 高精乘高精两种，学生们背起来太麻烦，容易出错，我把两个合并了一下，形成了一个模板，对于高精乘低精也可以兼容到，比如：

```

1  int main() {
2      string x;
3      int y;
4      cin >> x >> y;
5      for (int i = x.size() - 1; i >= 0; i--) a[a1++] = x[i] - '0';
6      b[0] = y, b1 = 1;
7      mul(a, a1, b, b1);
8      for (int i = a1; i >= 0; i--) printf("%d", a[i]);
9      return 0;
10 }

```

五、[AcWing_794.高精度除法](#)

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5  int a[N], a1, r;
6  void div(int a[], int &a1, int b, int &r) {
7      r = 0;
8      for (int i = a1 - 1; i >= 0; i--) {
9          r = r * 10 + a[i];
10         a[i] = r / b;
11         r %= b;
12     }
13     // 去前导0

```

```

14     while (a1 > 0 && !a[a1]) a1--;
15 }
16
17 int main() {
18     string x;
19     int y;
20
21     cin >> x >> y;
22
23     for (int i = x.size() - 1; i >= 0; i--) a[a1++] = x[i] - '0';
24     // 高精度除法
25     div(a, a1, y, r);
26
27     // 输出
28     for (int i = a1; i >= 0; i--) printf("%d", a[i]);
29     puts("");
30     printf("%d\n", r);
31     return 0;
32 }

```

六、保留结果 K 位

```

1  // 高精乘高精, 限长K(K<=100, 不是很大)
2  void mul(int a[], int b[]) {
3      int t = 0;
4      int c[N] = {0};
5      for (int i = 0; i < K; i++)
6          for (int j = 0; j < K; j++)
7              if (i + j < K) c[i + j] += a[i] * b[j]; // 防止越界, 需要加上i+j<K的
            限制
8
9      for (int i = 0; i < K; i++) {
10         t += c[i]; // 进位与当前位置的和
11         a[i] = t % 10; // 余数保留下来
12         t /= 10; // 进位
13     }
14 }
15
16 // 去前导0
17 //int al=k;
18 //while (a1 > 0 && a[a1]==0) a1--;

```