

lesson2

高精度

- 存储方式: 个位存到数组第0位, 最高位存到最后
- 高精度加法
 - $A_i + B_i + t$ (进位0 or 1)

```
1 static List<Integer> add(List<Integer> A, List<Integer> B) {
2     List<Integer> res = new ArrayList<Integer>();
3
4     for (int i=0, t=0; i < A.size() || i < B.size() || t>0; i++) {
5         if (i < A.size()) t += A.get(i);
6         if (i < B.size()) t += B.get(i);
7         res.add(t % 10);
8         t /= 10;
9     }
10
11     return res;
12 }
13
14 public static void main(String[] args) throws Exception {
15     String a = inb.readLine(), b = inb.readLine();
16     List<Integer> A = new ArrayList<Integer>();
17     List<Integer> B = new ArrayList<Integer>();
18     List<Integer> C = new ArrayList<Integer>();
19
20     for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
21     for (int i=b.length()-1; i>=0; i--) B.add(b.charAt(i)-'0');
22
23     C = add(A, B);
24
25     for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));
26
27     out.flush();
28 }
```

- 高精度减法
- $A_i - B_i - t$
 - $/ \geq 0$ $A_i - B_i - t$
 - $/ < 0$ $A_i - B_i - t + 10$
- 保证 $A \geq B$
 - $A \geq B$ $A - B$
 - $A < B$ $-(B - A)$

```
1 static boolean cmp(List<Integer> A, List<Integer> B) {
2     if (A.size() != B.size()) return A.size() > B.size();
3     else {
```

```

4         for (int i=A.size()-1; i>=0; i--)
5             if (A.get(i) != B.get(i)) return A.get(i) > B.get(i);
6
7     }
8     return true;
9 }
10
11 static List<Integer> sub(List<Integer> A, List<Integer> B) {
12     List<Integer> res = new ArrayList<Integer>();
13
14     for (int i=0, t=0; i < A.size() || t > 0; i++) {
15         t = A.get(i) - t;
16         if (i < B.size()) t -= B.get(i);
17         res.add((t+10) % 10);
18         t = t < 0 ? 1 : 0;
19     }
20
21     while (res.size() > 1 && res.get(res.size()-1) == 0)
22         res.remove(res.size()-1);
23
24     return res;
25 }
26
27 public static void main(String[] args) throws Exception {
28     String a = inb.readLine(), b = inb.readLine();
29     List<Integer> A = new ArrayList<Integer>();
30     List<Integer> B = new ArrayList<Integer>();
31     List<Integer> C = new ArrayList<Integer>();
32
33     for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
34     for (int i=b.length()-1; i>=0; i--) B.add(b.charAt(i)-'0');
35
36     if (cmp(A, B)) C = sub(A, B);
37     else {
38         out.print("-");
39         C = sub(B, A);
40     }
41
42     for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));
43
44     out.flush();
45 }

```

- 高精度乘法

- ```

1 static List<Integer> mul(List<Integer> A, int b) {
2 List<Integer> res = new ArrayList<Integer>();
3
4 for (int i=0, t=0; i < A.size() || t > 0; i++) {
5 if (i < A.size()) t += A.get(i)*b;
6 res.add(t % 10);
7 t /= 10;
8 }
9

```

```

10 while (res.size() > 1 && res.get(res.size()-1) == 0)
11 res.remove(res.size()-1);
12 return res;
13 }
14
15 public static void main(String[] args) throws Exception {
16 String a = inb.readLine(); int b =
17 Integer.parseInt(inb.readLine());
18
19 List<Integer> A = new ArrayList<Integer>();
20 List<Integer> C = new ArrayList<Integer>();
21
22 for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
23
24 C = mul(A, b);
25
26 for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));
27
28 out.flush();
29 }

```

- 高精度除法

- 从最高位开始运算

```

1 static List<Integer> div(List<Integer> A, int b, int[] rr) {
2 List<Integer> res = new ArrayList<Integer>(); int r = rr[0];
3
4 for (int i=A.size()-1; i>=0; i--) {
5 r = r*10+A.get(i);
6 res.add(r / b);
7 r = r % b;
8 }
9
10 Collections.reverse(res); rr[0] = r;
11 while (res.size() > 1 && res.get(res.size()-1) == 0)
12 res.remove(res.size()-1);
13
14 return res;
15 }
16
17 public static void main(String[] args) throws Exception {
18 String a = inb.readLine(); int b =
19 Integer.parseInt(inb.readLine()); int[] rr = {0};
20
21 List<Integer> A = new ArrayList<Integer>();
22 List<Integer> C = new ArrayList<Integer>();
23
24 for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
25
26 C = div(A, b, rr);
27
28 for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));
29
30 out.print("\n"+rr[0]);
31 }

```

```

28
29 out.flush();
30 }

```

### 前缀和&差分（互为逆运算，下标从1开始）

前缀和：（下标需要从1开始，方便定义 $s_0$ ，处理边界问题，统一公式形式）

- 前缀和数组： $S_i = a_1 + a_2 + \dots + a_i$
- 如何求 $S_i$ ? 递推一遍

```

1 s[0] = 0
2 for i = 1; i <= n; i++
3 s[i] = s[i-1] + a[i]

```

- 有什么用?  
快速求出 $[l, r]$ 的和  $S_r - S_{l-1}$
- 一维前缀和（求出某段区间和）

```

1 static int N = 100010;
2
3 static int n, m;
4 static int[] a = new int[N], s = new int[N];
5
6 public static void main(String[] args) throws Exception {
7 ins.nextToken(); n = (int)ins.nval;
8 ins.nextToken(); m = (int)ins.nval;
9
10 for (int i=1; i<=n; i++) { ins.nextToken(); a[i] =
 (int)ins.nval; }
11
12 for (int i=1; i<=n; i++) s[i] = s[i-1]+a[i];
13
14 while (m-- > 0) {
15 ins.nextToken(); int l = (int)ins.nval;
16 ins.nextToken(); int r = (int)ins.nval;
17 out.println(s[r]-s[l-1]);
18 }
19
20 out.flush();
21 }

```

- 二维前缀和
  - 快速求出某个子矩阵之和
  - 左上角  $(x_1, y_1)$ ，右下角  $(x_2, y_2)$
  - 初始化前缀和

```

1 for (i: 1 - n)
2 for (j: 1-m)
3 s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + a[i][j];

```

- S子矩阵 =  $S[x_2, y_2] - S[x_2, y_1 - 1] - S[x_1 - 1, y_2] + S[x_1 - 1, y_1 - 1]$

```

1 static int N = 1010;
2
3 static int n, m, q;
4 static int[][] a = new int[N][N], s = new int[N][N];
5
6 public static void main(String[] args) throws Exception {
7 ins.nextToken(); n = (int)ins.nval;
8 ins.nextToken(); m = (int)ins.nval;
9 ins.nextToken(); q = (int)ins.nval;
10
11 for (int i=1; i<=n; i++)
12 for (int j=1; j<=m; j++) { ins.nextToken(); a[i][j] =
13 (int)ins.nval; }
14 //初始化前缀和
15 for (int i=1; i<=n; i++)
16 for (int j=1; j<=m; j++) s[i][j] = s[i-1][j]+s[i][j-1]-s[i-1]
17 [j-1]+a[i][j];
18 while (q-- > 0) {
19 ins.nextToken(); int x1 = (int)ins.nval;
20 ins.nextToken(); int y1 = (int)ins.nval;
21 ins.nextToken(); int x2 = (int)ins.nval;
22 ins.nextToken(); int y2 = (int)ins.nval;
23 //求子矩阵和
24 out.println(s[x2][y2]-s[x2][y1-1]-s[x1-1][y2]+s[x1-1][y1-1]);
25 }
26
27 out.flush();
28 }

```

差分（前缀和的逆运算）：

- 一维差分
  - 构造原数组
  - $b[n] = a[n] - a[n-1]$
  - 初始假定前缀和数组a所有元素为0，通过n次插入操作进行初始化b数组
  - 作用：
    - $O(n)$  B→A
  - 快速对原数组给定[l,r]区间内全部数进行同一种操作（例如加减运算）
    - $b[l] + c, b[r+1] - c$  ( $O(1)$ 复杂度)

```

1 static int N = 100010;
2
3 static int n, m;
4 static int[] a = new int[N], b = new int[N];
5
6 static void insert(int l, int r, int c) {
7 b[l] += c;
8 b[r+1] -= c;
9 }
10
11 public static void main(String[] args) throws Exception {
12 ins.nextToken(); n = (int)ins.nval;
13 ins.nextToken(); m = (int)ins.nval;
14
15 for (int i=1; i<=n; i++) { ins.nextToken(); a[i] =
(int)ins.nval; }
16
17 for (int i=1; i<=n; i++) insert(i, i, a[i]);
18
19 while (m-- > 0) {
20 ins.nextToken(); int l = (int)ins.nval;
21 ins.nextToken(); int r = (int)ins.nval;
22 ins.nextToken(); int c = (int)ins.nval;
23 insert(l, r, c);
24 }
25
26 for (int i=1; i<=n; i++) { b[i] += b[i-1]; out.print(b[i]+"
"); }
27
28 out.flush();
29 }

```

- 二维差分

- 通过原矩阵 $a[i][j]$ 构造差分矩阵 $b[i][j]$ , 使得 $a[i][j]$ 是 $b[i][j]$ 的前缀和
- $b[x, y] += c$ ,  $b[x+1, y] -= c$ ,  $b[x, y+1] -= c$ ,  $b[x+1, y+1] += c$

```

1 static int N = 1010;
2
3 static int n, m, k;
4 static int[][] a = new int[N][N], b = new int[N][N];
5
6 static void insert(int x1, int y1, int x2, int y2, int c) {
7 b[x1][y1] += c;
8 b[x2+1][y1] -= c;
9 b[x1][y2+1] -= c;
10 b[x2+1][y2+1] += c;
11 }
12
13 public static void main(String[] args) throws Exception {
14 ins.nextToken(); n = (int)ins.nval;
15 ins.nextToken(); m = (int)ins.nval;
16 ins.nextToken(); k = (int)ins.nval;
17
18 for (int i=1; i<=n; i++)
19 for (int j=1; j<=m; j++) {
20 ins.nextToken(); a[i][j] = (int)ins.nval;

```

```
21 insert(i, j, i, j, a[i][j]);
22 }
23
24 while (k-- > 0) {
25 ins.nextToken(); int x1 = (int)ins.nval;
26 ins.nextToken(); int y1 = (int)ins.nval;
27 ins.nextToken(); int x2 = (int)ins.nval;
28 ins.nextToken(); int y2 = (int)ins.nval;
29 ins.nextToken(); int c = (int)ins.nval;
30 insert(x1, y1, x2, y2, c);
31 }
32
33 for (int i=1; i<=n; i++) {
34 for (int j=1; j<=m; j++) {
35 b[i][j] += b[i-1][j]+b[i][j-1]-b[i-1][j-1];
36 }
37 out.print(b[i][j]+" ");
38 }
39
40 out.flush();
41 }
```