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
  - Ai+Bi+t(进位0 or 1)

-
  ```java
  static List<Integer> add(List<Integer> A, List<Integer> B) {
      List<Integer> res = new ArrayList<Integer>();

      for (int i=0, t=0; i < A.size() || i < B.size() || t>0; i++) {
          if (i < A.size()) t += A.get(i);
          if (i < B.size()) t += B.get(i);
          res.add(t % 10);
          t /= 10;
      }

      return res;
  }

  public static void main(String[] args) throws Exception {
      String a = inb.readLine(), b = inb.readLine();
      List<Integer> A = new ArrayList<Integer>();
      List<Integer> B = new ArrayList<Integer>();
      List<Integer> C = new ArrayList<Integer>();

      for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
      for (int i=b.length()-1; i>=0; i--) B.add(b.charAt(i)-'0');

      C = add(A, B);

      for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));

      out.flush();
  }
  ```

- 高精度减法
- Ai-Bi-t
  - / >=0 Ai-Bi-t
  - / <0 Ai-Bi-t+10
- 保证A >= B
  - A>=B A-B
  - A<B -(B-A)

```java
static boolean cmp(List<Integer> A, List<Integer> B) {
    if (A.size() != B.size()) return A.size() > B.size();
    else {
        for (int i=A.size()-1; i>=0; i--)
            if (A.get(i) != B.get(i)) return A.get(i) > B.get(i);

    }
    return true;
}

static List<Integer> sub(List<Integer> A, List<Integer> B) {
    List<Integer> res = new ArrayList<Integer>();

    for (int i=0, t=0; i < A.size() || t > 0; i++) {
        t = A.get(i) - t;
        if (i < B.size()) t -= B.get(i);
        res.add((t+10) % 10);
        t = t < 0 ? 1 : 0;
    }

    while (res.size() > 1 && res.get(res.size()-1) == 0) res.remove(res.size()-1);

    return res;
}


public static void main(String[] args) throws Exception {
    String a = inb.readLine(), b = inb.readLine();
    List<Integer> A = new ArrayList<Integer>();
```

```
    List<Integer> B = new ArrayList<Integer>();
    List<Integer> C = new ArrayList<Integer>();

    for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');
    for (int i=b.length()-1; i>=0; i--) B.add(b.charAt(i)-'0');

    if (cmp(A, B)) C = sub(A, B);
    else {
        out.print("-");
        C = sub(B, A);
    }

    for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));

    out.flush();
}
```

- 高精度乘法

-
  ```
  static List<Integer> mul(List<Integer> A, int b) {
      List<Integer> res = new ArrayList<Integer>();

      for (int i=0, t=0; i < A.size() || t > 0; i++) {
          if (i < A.size()) t += A.get(i)*b;
          res.add(t % 10);
          t /= 10;
      }

      while (res.size() > 1 && res.get(res.size()-1) == 0) res.remove(res.size()-1);

      return res;
  }

  public static void main(String[] args) throws Exception {
      String a = inb.readLine(); int b = Integer.parseInt(inb.readLine());

      List<Integer> A = new ArrayList<Integer>();
      List<Integer> C = new ArrayList<Integer>();

      for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');

      C = mul(A, b);

      for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));

      out.flush();
  }
  ```

- 高精度除法
  - 从最高位开始运算

```
static List<Integer> div(List<Integer> A, int b, int[] rr) {
    List<Integer> res = new ArrayList<Integer>(); int r = rr[0];

    for (int i=A.size()-1; i>=0; i--) {
        r = r*10+A.get(i);
        res.add(r / b);
        r = r % b;
    }

    Collections.reverse(res); rr[0] = r;
    while (res.size() > 1 && res.get(res.size()-1) == 0) res.remove(res.size()-1);

    return res;
}

public static void main(String[] args) throws Exception {
    String a = inb.readLine(); int b = Integer.parseInt(inb.readLine()); int[] rr = {0};

    List<Integer> A = new ArrayList<Integer>();
    List<Integer> C = new ArrayList<Integer>();

    for (int i=a.length()-1; i>=0; i--) A.add(a.charAt(i)-'0');

    C = div(A, b, rr);

```

```
    for (int i=C.size()-1; i>=0; i--) out.print(C.get(i));
    out.print("\n"+rr[0]);

    out.flush();
}
```

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s0，处理边界问题，统一公式形式）

- 前缀和数组：Si = a1+a2+...+ai
- 如何求Si？ 递推一遍

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

- 有什么用？

  快速求出[l, r]的和 Sr - S(l-1)

- 一维前缀和（求出某段区间和）

```
static int N = 100010;

static int n, m;
static int[] a = new int[N], s = new int[N];

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=1; i<=n; i++) { ins.nextToken(); a[i] = (int)ins.nval; }

    for (int i=1; i<=n; i++) s[i] = s[i-1]+a[i];

    while (m-- > 0) {
        ins.nextToken(); int l = (int)ins.nval;
        ins.nextToken(); int r = (int)ins.nval;
        out.println(s[r]-s[l-1]);
    }

    out.flush();
}
```

- 二维前缀和
  - 快速求出某个子矩阵之和
  - 左上角（x1，y2），右下角（x2，y2）
  - 初始化前缀和

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

- S子矩阵 = S[x2,y2] - S[x2, y1-1] - S[x1-1, y2] + S[x1-1, y1-1]

```
static int N = 1010;

static int n, m, q;
static int[][] a = new int[N][N], s = new int[N][N];

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;
    ins.nextToken(); q = (int)ins.nval;

    for (int i=1; i<=n; i++)
        for (int j=1; j<=m; j++) { ins.nextToken(); a[i][j] = (int)ins.nval; }

    //初始化前缀和
    for (int i=1; i<=n; i++)
        for (int j=1; j<=m; j++) s[i][j] = s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];

    while (q-- > 0) {
        ins.nextToken(); int x1 = (int)ins.nval;
        ins.nextToken(); int y1 = (int)ins.nval;
        ins.nextToken(); int x2 = (int)ins.nval;
        ins.nextToken(); int y2 = (int)ins.nval;
        //求子矩阵和
        out.println(s[x2][y2]-s[x2][y1-1]-s[x1-1][y2]+s[x1-1][y1-1]);
    }

    out.flush();
}
```

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）
    -
```
static int N = 100010;

static int n, m;
static int[] a = new int[N], b = new int[N];

static void insert(int l, int r, int c) {
    b[l] += c;
    b[r+1] -= c;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=1; i<=n; i++) { ins.nextToken(); a[i] = (int)ins.nval; }

    for (int i=1; i<=n; i++) insert(i, i, a[i]);

    while (m-- > 0) {
        ins.nextToken(); int l = (int)ins.nval;
        ins.nextToken(); int r = (int)ins.nval;
        ins.nextToken(); int c = (int)ins.nval;
        insert(l, r, c);
    }

    for (int i=1; i<=n; i++) { b[i] += b[i-1]; out.print(b[i]+" "); }

    out.flush();
}
```

- 二维差分
    - 通过原矩阵a[ij]构造差分矩阵b[ij]，使得aij是bij的前缀和
    - b[x, y] += c, b[x2+1, y1] -= c, b[x1, y2+1] -=c, b[x2+1, y2+1] += c
-
```
static int N = 1010;

static int n, m, k;
static int[][] a = new int[N][N], b = new int[N][N];

static void insert(int x1, int y1, int x2, int y2, int c) {
    b[x1][y1] += c;
    b[x2+1][y1] -= c;
    b[x1][y2+1] -= c;
    b[x2+1][y2+1] += c;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;
    ins.nextToken(); k = (int)ins.nval;

    for (int i=1; i<=n; i++)
        for (int j=1; j<=m; j++) {
            ins.nextToken(); a[i][j] = (int)ins.nval;
```

```java
            insert(i, j, i, j, a[i][j]);
        }

    while (k-- > 0) {
        ins.nextToken(); int x1 = (int)ins.nval;
        ins.nextToken(); int y1 = (int)ins.nval;
        ins.nextToken(); int x2 = (int)ins.nval;
        ins.nextToken(); int y2 = (int)ins.nval;
        ins.nextToken(); int c = (int)ins.nval;
        insert(x1, y1, x2, y2, c);
    }

    for (int i=1; i<=n; i++) {
        for (int j=1; j<=m; j++) {
            b[i][j] += b[i-1][j]+b[i][j-1]-b[i-1][j-1]; out.print(b[i][j]+" ");
        }
        out.print("\n");
    }

    out.flush();
}
```