

Lesson1 (数组模拟数据结构)

链表与邻接表 (数组模拟)

单链表&邻接表 (n个链表) : 邻接表主要用于存储图与树

- e[N]:值 ne[N]:next
- e与ne数组通过下标关联, 空结点下标用-1表示, 链表最后一个有效结点ne为-1

```
1 static int N = 100010;
2
3 static int n;
4 static int head, idx; //head表示头节点的下标, idx表示当前已经使用到了哪个点
5 static int[] e = new int[N], ne = new int[N]; // e[i]表示结点i的值,
   ne[i]表示节点i的next指针
6
7 static void init() {
8     head = -1; idx = 0;
9 }
10
11 static void addToHead(int x) {
12     e[idx] = x; ne[idx] = head; head = idx++;
13 }
14
15 static void add(int k, int x) {
16     e[idx] = x; ne[idx] = ne[k]; ne[k] = idx++;
17 }
18
19 static void remove(int k) {
20     ne[k] = ne[ne[k]];
21 }
22
23 public static void main(String[] args) throws Exception {
24     ins.nextToken(); n = (int)ins.nval;
25
26     init(); //记得初始化
27
28     while (n-- > 0) {
29         ins.nextToken(); String op = (String)ins.sval;
30
31         if (op.equals("H")) {
32             ins.nextToken(); int x = (int)ins.nval;
33             addToHead(x);
34         }
35         else if (op.equals("D")) {
36             ins.nextToken(); int k = (int)ins.nval;
37             if (k == 0) head = ne[head];
38             else remove(k-1);
39         }
40         else if (op.equals("I")) {
41             ins.nextToken(); int k = (int)ins.nval;
42             ins.nextToken(); int x = (int)ins.nval;
43             add(k-1, x);
44         }
45     }
```

```

45     }
46
47     for (int i=head; i!=-1; i=ne[i]) out.print(e[i]+" ");
48
49     out.flush();
50 }
51 }

```

双链表：优化某些特定问题

- 在单链表基础上增加指向前一个结点的指针
 - l[N], r[N]
- 不定义头结点与尾结点, 0: head 1: tail

```

1  static int N = 100010;
2
3  static int n;
4  static int idx;
5  static int[] l = new int[N], r = new int[N], e = new int[N];
6
7  static void init() {      //0表示头节点, 1表示尾节点, 头尾节点只有指示作用, idx
    从2开始
8      r[0] = 1; l[1] = 0;
9      idx = 2;
10 }
11
12 static void add(int k, int x) {      //在第k个插入的数右边插入一个结点
13     e[idx] = x;
14     l[idx] = k; r[idx] = r[k];
15     l[r[k]] = idx; r[k] = idx++;
16 }
17
18 static void remove(int k) {
19     r[l[k]] = r[k];
20     l[r[k]] = l[k];
21 }
22
23 public static void main(String[] args) throws Exception {
24     ins.nextTok(); n = (int)ins.nval;
25
26     init();
27
28     while (n-- > 0) {
29         ins.nextTok(); String op = (String)ins.sval;
30
31         if (op.equals("L")) {
32             ins.nextTok(); int x = (int)ins.nval;
33             add(0, x);
34         }
35         else if (op.equals("R")) {
36             ins.nextTok(); int x = (int)ins.nval;
37             add(l[1], x);
38         }
39         else if (op.equals("D")) {

```

```

40         ins.nextToken(); int k = (int)ins.nval;
41         remove(k+1);
42     }
43     else if (op.equals("IL")) {
44         ins.nextToken(); int k = (int)ins.nval;
45         ins.nextToken(); int x = (int)ins.nval;
46         add(l[k+1], x);
47     }
48     else if (op.equals("IR")) {
49         ins.nextToken(); int k = (int)ins.nval;
50         ins.nextToken(); int x = (int)ins.nval;
51         add(k+1, x);
52     }
53 }
54
55 for (int i=r[0]; i!=1; i=r[i]) out.print(e[i]+" ");
56
57 out.flush();
58 }

```

邻接表 (多个单链表)

- head[1]->...
- head[2]->...
- ...
- head[3]->...

栈 (先进后出)

```

1 //stk表示栈，tt表示栈顶
2 int stk[N], tt;
3
4 //插入x
5 stk[++tt] = x;
6
7 //弹出栈顶
8 tt--;
9
10 //判断是否为空
11 if (tt > 0) not empty
12 else empty
13
14 //栈顶
15 stk[tt];

```

- e1:

```

1 static int N = 100010;
2
3 static int n;
4 static int tt; //栈顶指针

```

```

5  static int[] stk = new int[N];
6
7  public static void main(String[] args) throws Exception {
8      ins.nextToken(); n = (int)ins.nval;
9
10     while (n-- > 0) {
11         ins.nextToken(); String op = (String)ins.sval;
12
13         if (op.equals("push")) {
14             ins.nextToken(); int x = (int)ins.nval;
15             stk[++tt] = x;
16         }
17         else if (op.equals("pop")) tt--;
18         else if (op.equals("empty")) {
19             if (tt > 0) out.println("NO");
20             else out.println("YES");
21         }
22         else out.println(stk[tt]);
23     }
24
25     out.flush();
26 }

```

- e2: 表达式求值

```

1  static int N = 100010;
2
3  //使用两个栈模拟表达式树的计算方式
4  //表达式树：叶结点为数，其余结点为运算符，且符号优先级随深度增加保持严格单调递增
5  static int tt1, tt2;
6  static int[] num = new int[N]; //数字栈
7  static char[] op = new char[N]; //表达式栈
8
9  static void eval() {
10     int b = num[tt1--];
11     int a = num[tt1--];
12     char c = op[tt2--];
13
14     int x = 0;
15     if (c == '+') x = a+b;
16     else if (c == '-') x = a-b;
17     else if (c == '*') x = a*b;
18     else if (c == '/') x = a/b;
19
20     num[++tt1] = x;
21 }
22
23 public static void main(String[] args) throws Exception {
24     String str = inb.readLine();
25
26     Map<Character, Integer> pr = new HashMap<Character, Integer>();
27     pr.put('+', 1); pr.put('-', 1); pr.put('*', 2); pr.put('/', 2);
28
29     for (int i=0; i<str.length(); i++) {

```

```

30         char c = str.charAt(i);
31
32         if (Character.isDigit(c)) {
33             int x = 0, j = i;
34             while (j < str.length() && Character.isDigit(str.charAt(j)))
35                 x = x*10 + str.charAt(j++)-'0';
36             i = j-1;
37             num[++tt1] = x;
38         }
39         else if (c == '(') op[++tt2] = c;
40         else if (c == ')') {
41             while (op[tt2] != '(') eval();
42             tt2--;
43         }
44         else {
45             //注意是>=, 而非>, 从而保持符号优先级严格递增
46             while (tt2 > 0 && op[tt2] != '(' && pr.get(op[tt2]) >=
pr.get(c)) eval();
47             op[++tt2] = c;
48         }
49     }
50
51     while (tt2 > 0) eval();
52
53     out.print(num[tt1]);
54
55     out.flush();
56 }

```

队列 (先进先出)

```

1 //在队尾插入元素, 在队头弹出元素
2 int q[N], hh, tt = -1;
3
4 //插入
5 q[++tt] = x;
6
7 //弹出
8 hh++;
9
10 //判断是否为空
11 if (hh <= tt) not empty
12 else empty
13
14 //取队头
15 q[hh]

```

- e1

```

1 static int N = 100010;
2

```

```

3  static int n;
4  static int hh, tt = -1;
5  static int[] q = new int[N];
6
7  public static void main(String[] args) throws Exception {
8      ins.nextToken(); n = (int)ins.nval;
9
10     while (n-- > 0) {
11         ins.nextToken(); String op = (String)ins.sval;
12
13         if (op.equals("push")) {
14             ins.nextToken(); int x = (int)ins.nval;
15             q[++tt] = x;
16         }
17         else if (op.equals("pop")) hh++;
18         else if (op.equals("empty")) {
19             if (hh <= tt) out.println("NO");
20             else out.println("YES");
21         }
22         else out.println(q[hh]);
23     }
24
25     out.flush();
26 }

```

单调栈

常见模型：在一个序列中，找出每一个数左边离它最近的且满足某种性质（如 最大最小）的数的位置

先想暴力算法，再利用单调性进行优化

```

1  static int N = 100010;
2
3  static int n;
4  static int tt;
5  static int[] stk = new int[N];
6
7  public static void main(String[] args) throws Exception {
8      ins.nextToken(); n = (int)ins.nval;
9
10     while (n-- > 0) {
11         ins.nextToken(); int x = (int)ins.nval;
12
13         while (tt > 0 && stk[tt] >= x) tt--;
14         if (tt > 0) out.print(stk[tt]+" ");
15         else out.print("-1 ");
16
17         stk[++tt] = x;
18     }
19
20     out.flush();
21 }

```

单调队列

经典应用：求滑动窗口中最大值or最小值

```
1  static int N = 1000010;
2
3  static int n, k;
4  static int[] a = new int[N];
5  static int hh, tt = -1;
6  static int[] q = new int[N];    //存储下标的单调队列(与目标值一一对应，所以合理)
7
8  public static void main(String[] args) throws Exception {
9      ins.nextToken(); n = (int)ins.nval;
10     ins.nextToken(); k = (int)ins.nval;
11
12     for (int i=0; i<n; i++) { ins.nextToken(); a[i] = (int)ins.nval; }
13
14     for (int i=0; i<n; i++) {
15         if (hh <= tt && i-k+1 > q[hh]) hh++;
16
17         while (hh <= tt && a[q[tt]] >= a[i]) tt--;
18         q[++tt] = i;
19
20         if (i-k+1 >= 0) out.print(a[q[hh]]+" ");
21     }
22
23     out.println();
24
25     hh = 0; tt = -1;
26     for (int i=0; i<n; i++) {
27         if (hh <= tt && i-k+1 > q[hh]) hh++;
28
29         while (hh <= tt && a[q[tt]] <= a[i]) tt--;
30         q[++tt] = i;
31
32         if (i-k+1 >= 0) out.print(a[q[hh]]+" ");
33     }
34
35     out.flush();
36 }
```

KMP

习惯下标从1开始

1. 暴力算法如何做

```
1  S[N], p[M];
```

```

2
3 //朴素做法
4 for (int i=1; i<=n; i++) {
5     bool flag = true;
6     for (int j=1; j<=m; j++) {
7         if (s[i+j-1] != p[j]) {
8             flag = false;
9             break;
10        }
11    }
12
13    if (flag) 匹配成功
14 }

```

2. 如何去优化

- next[i] 表示以i为终点，且后缀和前缀相等的子串的最大长度。

next[i] = j p[1, j] = p[i-j+1, i]

```

1 static int N = 100010, M = 1000010;
2
3 static int n, m;
4 static int[] ne = new int[N];
5 static char[] p = new char[N], s = new char[M];
6
7
8 public static void main(String[] args) throws Exception {
9     n = Integer.parseInt(inb.readLine()); //下标从1开始
10    String tmp = inb.readLine(); tmp = " "+tmp; p = tmp.toCharArray();
11    m = Integer.parseInt(inb.readLine());
12    tmp = inb.readLine(); tmp = " "+tmp; s = tmp.toCharArray();
13
14    for (int i=2, j=0; i<=n; i++) {
15        while (j > 0 && p[i] != p[j+1]) j = ne[j];
16        if (p[i] == p[j+1]) j++;
17        ne[i] = j;
18    }
19
20    for (int i=1, j=0; i<=m; i++) {
21        while (j > 0 && s[i] != p[j+1]) j = ne[j];
22        if (s[i] == p[j+1]) j++;
23        if (j == n) {
24            out.print(i-n+" ");
25            j = ne[j];
26        }
27    }
28
29    out.flush();
30 }
31 }

```

作用

- 可用于求循环节（字符串哈希无法使用）

