

## Lesson2

### Trie树(字典树)

- 基本作用：高效地**存储和查找字符串集合**的数据结构

#### 存储

- 以字典形式存储，且字母类型较少
- 每个单词结尾处打上一个标记，便于检索树

#### 查找

- e1: 字符串存储和查找

```
1  static int N = 100010;
2
3  static int n;
4  static int[][] son = new int[N][26];
5  static int[] cnt = new int[N];
6  static int idx; //编号为0的点既是根节点，也是空结点
7
8  static void insert(String s) {
9      int p = 0;
10     for (int i=0; i<s.length(); i++) {
11         int u = s.charAt(i)-'a';
12         if (son[p][u] == 0) son[p][u] = ++idx;
13         p = son[p][u];
14     }
15
16     cnt[p]++;
17 }
18
19 static int query(String s) {
20     int p = 0;
21     for (int i=0; i<s.length(); i++) {
22         int u = s.charAt(i)-'a';
23         if (son[p][u] == 0) return 0;
24         p = son[p][u];
25     }
26
27     return cnt[p];
28 }
29
30 public static void main(String[] args) throws Exception {
31     n = Integer.parseInt(inb.readLine());
32
33     while (n-- > 0) {
34         String[] ss = inb.readLine().split(" ");
35
36         if (ss[0].equals("I")) insert(ss[1]);
37         else out.println(query(ss[1]));
38     }
39
40     out.flush();
```

41 }

- e2: 最大异或(^)对

```
1  static int N = 100010, M = 31*N;
2
3  static int n;
4  static int[] a = new int[N];
5  static int[][] son = new int[M][2];
6  static int idx;
7  //由于每个整数固定用31位表示，故不需要结束标记数组
8
9  static void insert(int x) {
10     int p = 0;
11     for (int i=30; i>=0; i--) {
12         int u = x>>i&1;
13         if (son[p][u] == 0) son[p][u] = ++idx;
14         p = son[p][u];
15     }
16 }
17
18 static int query(int x) { //找出x的异或最大值
19     int p = 0, res = 0;
20     for (int i=30; i>=0; i--) {
21         int u = x>>i&1;
22         if (son[p][1-u] != 0) {
23             res += 1<<i;
24             p = son[p][1-u]; //注意使用son[p][1-u]更新p
25         }
26         else p = son[p][u];
27     }
28
29     return res;
30 }
31
32 public static void main(String[] args) throws Exception {
33     ins.nextToken(); n = (int)ins.nval;
34
35     for (int i=0; i<n; i++) {
36         ins.nextToken(); a[i] = (int)ins.nval;
37         insert(a[i]);
38     }
39
40     int res = 0;
41     for (int i=0; i<n; i++) res = Math.max(res, query(a[i])); //循环求
    出全局最大
42
43     out.print(res);
44
45     out.flush();
46 }
```

## 并查集

基本作用：

近乎  $O(1)$  时间内完成以下两种操作

- 将两个集合合并
- 询问两个元素是否在一个集合中

基本原理

- 每个集合用一棵树表示（不一定是二叉树），以树根结点的编号作为集合的编号。每个结点存储其父结点信息， $p[x]$ 表示 $x$ 的父结点。
  - q1: 如何判断树根?  $\text{if } (p[x] == x)$
  - q2: 如何求 $x$ 的集合编号?  $\text{while } (p[x] != x) p[x] = \text{find}(p[x]);$  (**路径压缩优化**  $\rightarrow O(1)$ )
  - q3: 如何合并两个集合?  $p_x$ 是 $x$ 的集合编号,  $p_y$ 是 $y$ 的集合编号, 则 $p[x] = p_y$  或者  $p[y] = p_x$  (按秩合并优化, 效果不明显)
- e1: 合并集合

```
1 static int N = 100010;
2
3 static int n, m;
4 static int[] p = new int[N];
5
6 static int find(int x) {
7     if (x != p[x]) p[x] = find(p[x]);
8     return p[x];
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++) p[i] = i;
16
17     while (m-- > 0) {
18         ins.nextToken(); String op = (String)ins.sval;
19         ins.nextToken(); int a = (int)ins.nval;
20         ins.nextToken(); int b = (int)ins.nval;
21
22         if (op.equals("M")) p[find(a)] = find(b);
23         else {
24             if (find(a) == find(b)) out.println("Yes");
25             else out.println("No");
26         }
27     }
28
29     out.flush();
30 }
```

维护额外信息

- e2: 连通块中点的数量 (额外维护每个连通块中点的数量)

```

1  static int N = 100010;
2
3  static int n, m;
4  static int[] p = new int[N], cnt = new int[N];
5
6  static int find(int x) {
7      if (x != p[x]) p[x] = find(p[x]);
8      return p[x];
9  }
10
11 public static void main(String[] args) throws Exception {
12     String[] ss = inb.readLine().split(" ");
13     n = Integer.parseInt(ss[0]); m = Integer.parseInt(ss[1]);
14
15     for (int i=1; i<=n; i++) { p[i] = i; cnt[i] = 1; }
16
17     while (m-- > 0) {
18         ss = inb.readLine().split(" ");
19
20         if (ss[0].equals("c")) {
21             int a = Integer.parseInt(ss[1]), b =
Integer.parseInt(ss[2]);
22
23             a = find(a); b = find(b);
24             p[a] = b;
25             if (a != b) cnt[b] += cnt[a];
26         }
27         else if (ss[0].equals("q1")) {
28             int a = Integer.parseInt(ss[1]), b =
Integer.parseInt(ss[2]);
29
30             if (find(a) == find(b)) out.println("Yes");
31             else out.println("No");
32         }
33         else {
34             int a = Integer.parseInt(ss[1]);
35
36             out.println(cnt[find(a)]);
37         }
38     }
39
40     out.flush();
41 }

```

- e3: 食物链

```

1  static int N = 50010;
2
3  static int n, k;
4  static int[] p = new int[N], d = new int[N];    //d维护每个结点到根结点的距
离信息
5
6  static int find(int x) {
7      if (x != p[x]) {

```

```

8         int t = find(p[x]);
9         d[x] += d[p[x]];
10        p[x] = t;
11    }
12
13    return p[x];
14 }
15
16 public static void main(String[] args) throws Exception {
17     ins.nextToken(); n = (int)ins.nval;
18     ins.nextToken(); k = (int)ins.nval;
19
20     for (int i=1; i<=n; i++) p[i] = i;
21
22     int res = 0;
23     while (k-- > 0) {
24         ins.nextToken(); int t = (int)ins.nval;
25         ins.nextToken(); int x = (int)ins.nval;
26         ins.nextToken(); int y = (int)ins.nval;
27
28         if (x > n || y > n) res++;
29         else {
30             if (t == 1) {
31                 int px = find(x), py = find(y);
32                 if (px == py && (d[x]-d[y]) % 3 != 0) res++;
33                 else if (px != py) {
34                     p[px] = py;
35                     d[px] = d[y]-d[x];
36                 }
37             }
38             else if (t == 2){
39                 int px = find(x), py = find(y);
40                 if (px == py && (d[x]-d[y]+1) % 3 != 0) res++;
41                 else if (px != py) {
42                     p[px] = py;
43                     d[px] = d[y]-d[x]-1;
44                 }
45             }
46         }
47     }
48
49     out.println(res);
50
51     out.flush();
52 }

```

## 堆 (手写堆)

堆是一颗**完全二叉树**

支持操作

- 插入一个数  
heap[++cnt] = x; up(cnt);

- 求集合中的最小值

```
heap[1];
```

- 删除最小值

```
heap[1] = heap[cnt]; cnt--; down(1);
```

————— STL容器（优先队列）不支持的操作 —————

- 删除任意一个元素

```
heap[k] = heap[cnt]; cnt--; down(k); up(k); (down与up只会执行一个)
```

- 修改任意一个元素

```
heap[k] = x; down(k); up(k);
```

基本性质（以小根堆为例）

- 每一个点小于等于左右儿子（递归定义）
- 根结点是最小值（小根堆）

堆的存储（使用一维数组进行存储（堆，完全二叉树），下标从1开始（kmp，前缀差分，堆存储））

**O(n)建堆方式：从n/2处 down 到 1**

基本操作

- down(x): x增大，往下调整x的位置（小根堆）
- up(x): x减小，往上调整x的位置（小根堆）
- e1: 堆排序

```
1 static int N = 100010;
2
3 static int n, m;
4 static int cnt;
5 static int[] h = new int[N];
6
7 static void down(int u) {
8     int t = u;
9     if (2*u <= cnt && h[2*u] < h[t]) t = 2*u;
10    if (2*u+1 <= cnt && h[2*u+1] < h[t]) t = 2*u+1;
11    if (t != u) {
12        int tp = h[u]; h[u] = h[t]; h[t] = tp;
13        down(t);
14    }
15 }
16
17 static void up(int u) {
18     while (u/2 > 0 && h[u/2] > h[u]) {
19         int tp = h[u]; h[u] = h[u/2]; h[u/2] = tp;
20         u >>= 1;
21     }
```

```

22 }
23
24 public static void main(String[] args) throws Exception {
25     ins.nextToken(); n = (int)ins.nval;
26     ins.nextToken(); m = (int)ins.nval;
27
28     for (int i=1; i<=n; i++) { ins.nextToken(); h[i] = (int)ins.nval; }
29     cnt = n;
30
31     for (int i=n/2; i>0; i--) down(i); //O(n)方式建堆
32
33     while (m-- > 0) {
34         out.print(h[1]+" ");
35         h[1] = h[cnt--];
36         down(1);
37     }
38
39     out.flush();
40 }

```

- e2: 模拟堆
  - 修改和删除任意一个元素需要额外维护两个数组。
    - ph[k]: 第k个插入的数在堆中的下标
- hp[i]: 下标是i的点是第几个插入的数

```

1  static int N = 100010;
2
3  static int n;
4  static int cnt;
5  static int[] h = new int[N], ph = new int[N], hp = new int[N];
6
7  static void swap(int[] a, int x, int y) {
8      int tp = a[x]; a[x] = a[y]; a[y] = tp;
9  }
10
11 static void heap_swap(int a, int b) {
12     // swap(ph, hp[a], hp[b]); //注意函数参数是下标!
13     // swap(hp, a, b);
14     // swap(h, a, b);
15     int tp = ph[hp[a]]; ph[hp[a]] = ph[hp[b]]; ph[hp[b]] = tp;
16     tp = hp[a]; hp[a] = hp[b]; hp[b] = tp;
17     tp = h[a]; h[a] = h[b]; h[b] = tp;
18 }
19
20 static void down(int u) {
21     int t = u;
22     if (2*u <= cnt && h[2*u] < h[t]) t = 2*u;
23     if (2*u+1 <= cnt && h[2*u+1] < h[t]) t = 2*u+1;
24     if (t != u) {
25         heap_swap(t, u);
26         down(t);
27     }
28 }

```

```

29
30 static void up(int u) {
31     while (u/2 > 0 && h[u/2] > h[u]) {
32         heap_swap(u/2, u);
33         u >>= 1;
34     }
35 }
36
37
38 public static void main(String[] args) throws Exception {
39     ins.nextToken(); n = (int)ins.nval;
40
41     int kk = 0;
42     while (n-- > 0) {
43         ins.nextToken(); String op = (String)ins.sval;
44
45         if (op.equals("I")) {
46             ins.nextToken(); int x = (int)ins.nval;
47             cnt++; kk++;
48             ph[kk] = cnt; hp[cnt] = kk;
49             h[cnt] = x; up(cnt);
50         }
51         else if (op.equals("PM")) out.println(h[1]);
52         else if (op.equals("DM")) {
53             heap_swap(1, cnt--); down(1);
54         }
55         else if (op.equals("D")) {
56             ins.nextToken(); int k = (int)ins.nval;
57             k = ph[k];
58             heap_swap(k, cnt--); down(k); up(k);
59         }
60         else {
61             ins.nextToken(); int k = (int)ins.nval;
62             ins.nextToken(); int x = (int)ins.nval;
63             k = ph[k];
64             h[k] = x; down(k); up(k);
65         }
66     }
67
68     out.flush();
69 }

```