

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  #include <iostream>
2
3  using namespace std;
4
5  const int N = 100010;
6
7  char str[N];
8  int son[N][26], cnt[N], idx;    //下标是0的点，既是空结点，也是根结点
9
10 //插入
11 void insert(char *str) {
12     int p = 0;
13     for (int i=0; str[i]; i++) {
14         int u = str[i]-'a';    //映射到0-25
15         if (!son[p][u]) son[p][u] = ++idx;
16         p = son[p][u];
17     }
18
19     cnt[p]++;
20 }
21
22 //查找
23 int query(char *str) {
24     int p = 0;
25     for (int i=0; str[i]; i++) {
26         int u = str[i]-'a';
27         if (!son[p][u]) return 0;
28         p = son[p][u];
29     }
30
31     return cnt[p];
32 }
33
34 int main(void) {
35     int n;
36     scanf("%d", &n);
37
38     while (n--) {
39         char op[2];
40         scanf("%s%s", op, str);
41
42         if (op[0] == 'I') insert(str);
43         else printf("%d\n", query(str));
44     }
45
46
47     return 0;
48 }
```

- e2: 最大异或(^)对

```
1  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 100010, M = 3000010;
7
8  int n;
9  int a[N];
10 int son[M][2], idx;
11
12 void insert(int x) {
13     int p = 0;
```

```
14     for (int i=30; ~i; i--) {
15         int s = x>>i&1;      //位运算
16
17         if (!son[p][s]) son[p][s] = ++idx;
18         p = son[p][s];
19     }
20 }
21
22 int query(int x) {
23     int p = 0, res = 0;
24     for (int i=30; ~i; i--) {
25         int s = x>>i&1;
26
27         //异或性质
28         if (son[p][!s]) {
29             res += 1<<i;
30             p = son[p][!s];
31         }
32         else p = son[p][s];
33     }
34
35     return res;
36 }
37
38 int main(void) {
39     scanf("%d", &n);
40
41     for (int i=0; i<n; i++) {
42         scanf("%d", &a[i]);
43         insert(a[i]);
44     }
45
46     int res = 0;
47     for (int i=0; i<n; i++) res = max(res, query(a[i]));
48
49     printf("%d", res);
50
51     return 0;
52 }
```

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？ if (p[x] == x)
 - q2：如何求x的集合编号？ while (p[x] != x) p[x] = find(p[x]);（**路径压缩优化** -> O(1)）
 - q3：如何合并两个集合？ px是x的集合编号，py是y的集合编号，则p[x] = py 或者 p[y] = px （按秩合并优化，效果不明显）
- e1: 合并集合

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 100010;
6
7  int n, m;
8  int p[N];
9
10 //返回x的祖宗结点 + 路径压缩
11 int find(int x) {
12     if (p[x] != x) p[x] = find(p[x]);
13     return p[x];
14 }
15
16 int main(void) {
17     scanf("%d%d", &n, &m);
18
19     for (int i=1; i<=n; i++) p[i] = i;
```

```
20
21     while (m--) {
22         int a, b;
23         char op[2];
24         scanf("%s%d%d", op, &a, &b);
25
26         if (op[0] == 'M') p[find(a)] = find(b);
27         else {
28             if (find(a) == find(b)) puts("Yes");
29             else puts("No");
30         }
31     }
32
33     return 0;
34 }
```

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  #include <iostream>
2
3  using namespace std;
4
5  const int N = 100010;
6
7  int n, m;
8  int p[N], cnt[N];    //cnt[i] i所在集合中结点的个数，只保证集合数中根结点的cnt正确。
9
10 int find(int x) {
11     if (p[x] != x) p[x] = find(p[x]);
12     return p[x];
13 }
14
15 int main(void) {
16     scanf("%d%d", &n, &m);
17
18     for (int i=1; i<=n; i++) p[i] = i, cnt[i] = 1;
19
20     while (m--) {
21         int a, b;
22         char op[3];
23         scanf("%s", op);
24
25         if (op[0] == 'C') {
26             scanf("%d%d", &a, &b);
27
28             a = find(a), b = find(b);
29             p[a] = b;
30             if (a != b) cnt[b] += cnt[a];
31         }
32         else if (op[1] == '1') {
33             scanf("%d%d", &a, &b);
34
35             if (find(a) == find(b)) puts("Yes");
36             else puts("No");
37         }
38         else {
39             scanf("%d", &a);
40             printf("%d\n", cnt[find(a)]);
41         }
42     }
43
44     return 0;
45 }
```

- e3: 食物链

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 50010;
6
7  int n, m;
8  int p[N], d[N];
9
10 //维护每个结点到根结点的距离信息
11 int find(int x) {
12
```

```
13     if (p[x] != x) {
14         int t = find(p[x]);
15         d[x] += d[p[x]];
16         p[x] = t;
17     }
18
19     return p[x];
20 }
21
22 int main(void) {
23     scanf("%d%d", &n, &m);
24
25     for (int i=1; i<=n; i++) p[i] = i;
26
27     int res = 0;
28     while (m--) {
29         int t, x, y;
30         scanf("%d%d%d", &t, &x, &y);
31
32         if (x>n || y>n) res++;
33         else {
34             if (t == 1) {
35                 int px = find(x), py = find(y);
36                 if (px==py && (d[x]-d[y])%3) res++;
37                 else if (px != py) {
38                     p[px] = py;
39                     d[px] = d[y]-d[x];
40                 }
41             }
42             else if (t == 2) {
43                 int px = find(x), py = find(y);
44                 if (px==py && (d[x]-d[y]-1)%3) res++;
45                 else if (px != py) {
46                     p[px] = py;
47                     d[px] = d[y]+1-d[x];
48                 }
49             }
50         }
51     }
52
53     printf("%d", res);
54
55     return 0;
56 }
```

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  #include <iostream>
2  #include <algorithm>
3
4  using namespace std;
5
6  const int N = 100010;
7
8  int n, m;
9  int h[N], cnt;
10
11 void down(int u) {
12     int t = u;
13     if (2*u<=cnt && h[2*u]<h[t]) t = 2*u;
14     if (2*u+1<=cnt && h[2*u+1]<h[t]) t = 2*u+1;
15     if (u != t) {
16         swap(h[u], h[t]);
17         down(t);
18     }
19 }
20
21 void up(int u) {
22     while (u/2 && h[u/2]>h[u]) {
23         swap(h[u/2], h[u]);
24         u >>= 1;
25     }
26 }
27
28 int main(void) {
29     scanf("%d%d", &n, &m);
30
31     for (int i=1; i<=n; i++) scanf("%d", &h[i]);
32     cnt = n;
33
34     //O(n)方式建堆
35     for (int i= n/2; i; i--) down(i);
36
37     while (m--) {
38         printf("%d ", h[1]);
39         h[1] = h[cnt], cnt--;
40         down(1);
41     }
42
43     return 0;
44 }
```

- 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  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 100010;
8
9  int h[N], ph[N], hp[N], cnt;
10
11 void heap_swap(int a, int b) {
12     swap(ph[hp[a]], ph[hp[b]]);
13     swap(hp[a], hp[b]);
14     swap(h[a], h[b]);
15 }
16
17 void down(int u) {
18     int t = u;
19     if (2*u<=cnt && h[2*u]<h[t]) t = 2*u;
```

```

20     if (2*u+1<=cnt && h[2*u+1]<h[t]) t = 2*u+1;
21     if (t != u) {
22         heap_swap(t, u);
23         down(t);
24     }
25 }
26
27
28 void up(int u) {
29     while (u/2 && h[u/2]>h[u]) {
30         heap_swap(u/2, u);
31         u >>= 1;
32     }
33 }
34
35 int main(void) {
36     int n, m = 0;
37     scanf("%d", &n);
38
39     while (n--) {
40         int k, x;
41         char op[10];
42         scanf("%s", op);
43
44         if (!strcmp(op, "I")) {
45             scanf("%d", &x);
46             cnt++, m++;
47             ph[m] = cnt, hp[cnt] = m;
48             h[cnt] = x;
49             up(cnt);
50         }
51         else if (!strcmp(op, "PM")) printf("%d\n", h[1]);
52         else if (!strcmp(op, "DM")) {
53             heap_swap(1, cnt);
54             cnt--;
55             down(1);
56         }
57         else if (!strcmp(op, "D")) {
58             scanf("%d", &k);
59             k = ph[k];
60             heap_swap(k, cnt);
61             cnt--;
62             down(k), up(k);
63         }
64         else {
65             scanf("%d%d", &k, &x);
66             k = ph[k];
67             h[k] = x;
68             down(k), up(k);
69         }
70     }
71
72     return 0;
73 }

```