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

```cpp
#include <iostream>

using namespace std;

const int N = 100010;

char str[N];
int son[N][26], cnt[N], idx;     //下标是0的点，既是空结点，也是根结点

//插入
void insert(char *str) {
    int p = 0;
    for (int i=0; str[i]; i++) {
        int u = str[i]-'a';   //映射到0-25
        if (!son[p][u]) son[p][u] = ++idx;
        p = son[p][u];
    }

    cnt[p]++;
}

//查找
int query(char *str) {
    int p = 0;
    for (int i=0; str[i]; i++) {
        int u = str[i]-'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }

    return cnt[p];
}

int main(void) {
    int n;
    scanf("%d", &n);

    while (n--) {
        char op[2];
        scanf("%s%s",op, str);

        if (op[0] == 'I') insert(str);
        else printf("%d\n", query(str));
    }


    return 0;
}
```

- e2: 最大异或(^)对

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010, M = 3000010;

int n;
int a[N];
int son[M][2], idx;

void insert(int x) {
    int p = 0;
```

```
    for (int i=30; ~i; i--) {
        int s = x>>i&1;     //位运算

        if (!son[p][s]) son[p][s] = ++idx;
        p = son[p][s];
    }
}

int query(int x) {
    int p = 0, res = 0;
    for (int i=30; ~i; i--) {
        int s = x>>i&1;

        //异或性质
        if (son[p][!s]) {
            res += 1<<i;
            p = son[p][!s];
        }
        else p = son[p][s];
    }

    return res;
}

int main(void) {
    scanf("%d", &n);

    for (int i=0; i<n; i++) {
        scanf("%d", &a[i]);
        insert(a[i]);
    }

    int res = 0;
    for (int i=0; i<n; i++) res = max(res, query(a[i]));

    printf("%d", res);

    return 0;
}
```

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

```
#include <iostream>

using namespace std;

const int N = 100010;

int n, m;
int p[N];

//返回x的祖宗结点 + 路径压缩
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int main(void) {
    scanf("%d%d", &n, &m);

    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
```

```
    if (p[x] != x) {
        int t = find(p[x]);
        d[x] += d[p[x]];
        p[x] = t;
    }

    return p[x];
}

int main(void) {
    scanf("%d%d", &n, &m);

    for (int i=1; i<=n; i++) p[i] = i;

    int res = 0;
    while (m--) {
        int t, x, y;
        scanf("%d%d%d", &t, &x, &y);

        if (x>n || y>n) res++;
        else {
            if (t == 1) {
                int px = find(x), py = find(y);
                if (px==py && (d[x]-d[y])%3) res++;
                else if (px != py) {
                    p[px] = py;
                    d[px] = d[y]-d[x];
                }
            }
            else if  (t == 2) {
                int px = find(x), py = find(y);
                if (px==py && (d[x]-d[y]-1)%3) res++;
                else if (px != py) {
                    p[px] = py;
                    d[px] = d[y]+1-d[x];
                }
            }
        }
    }

    printf("%d", res);

    return 0;
}
```

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

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 100010;

int n, m;
int h[N], cnt;

void down(int u) {
    int t = u;
    if (2*u<=cnt && h[2*u]<h[t]) t = 2*u;
    if (2*u+1<=cnt && h[2*u+1]<h[t]) t= 2*u+1;
    if (u != t) {
        swap(h[u], h[t]);
        down(t);
    }
}

void up(int u) {
    while (u/2 && h[u/2]>h[u]) {
        swap(h[u/2], h[u]);
        u >>= 1;
    }
}

int main(void) {
    scanf("%d%d", &n, &m);

    for (int i=1; i<=n; i++) scanf("%d", &h[i]);
    cnt = n;

    //O(n)方式建堆
    for (int i= n/2; i; i--) down(i);

    while (m--) {
        printf("%d ", h[1]);
        h[1] = h[cnt], cnt--;
        down(1);
    }

    return 0;
}
```

- 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

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010;

int h[N], ph[N], hp[N], cnt;

void heap_swap(int a, int b) {
    swap(ph[hp[a]], ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}

void down(int u) {
    int t = u;
    if (2*u<=cnt && h[2*u]<h[t]) t = 2*u;
```

```c
    if (2*u+1<=cnt && h[2*u+1]<h[t]) t = 2*u+1;
    if (t != u) {
        heap_swap(t, u);
        down(t);
    }
}


void up(int u) {
    while (u/2 && h[u/2]>h[u]) {
        heap_swap(u/2, u);
        u >>= 1;
    }
}

int main(void) {
    int n, m = 0;
    scanf("%d", &n);

    while (n--) {
        int k, x;
        char op[10];
        scanf("%s", op);

        if (!strcmp(op, "I")) {
            scanf("%d", &x);
            cnt++, m++;
            ph[m] = cnt, hp[cnt] = m;
            h[cnt] = x;
            up(cnt);
        }
        else if (!strcmp(op, "PM")) printf("%d\n", h[1]);
        else if (!strcmp(op, "DM")) {
            heap_swap(1, cnt);
            cnt--;
            down(1);
        }
        else if (!strcmp(op, "D")) {
            scanf("%d", &k);
            k = ph[k];
            heap_swap(k, cnt);
            cnt--;
            down(k), up(k);
        }
        else {
            scanf("%d%d", &k, &x);
            k = ph[k];
            h[k] = x;
            down(k), up(k);
        }
    }

    return 0;
}
```