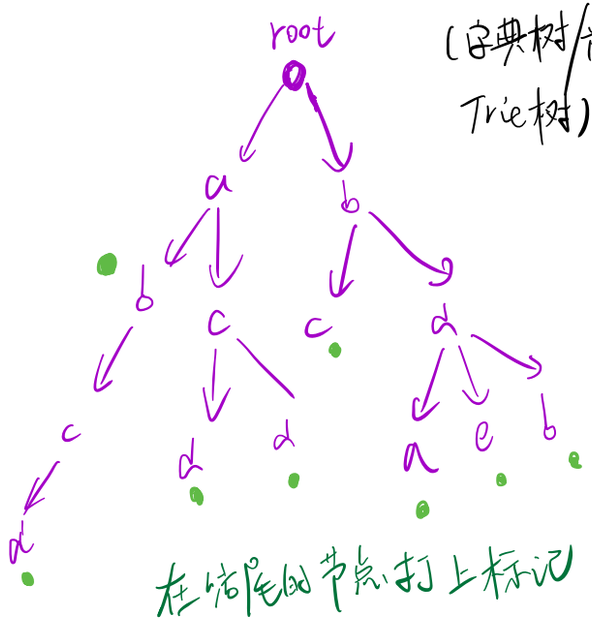
数据结构2

Trie 树

快速存储和查找字符串集合



(字典树/前缀树 Trie树)

在结尾的节点打上标记

一个 son[N][26]，一个 idx → 最多26个分支

cnt[] → 以该节点结尾的个数

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

using namespace std;

const int N = 1e5 + 10;

int son[N][26],cnt[N];
int idx;
char st[N];

void insert(char st[])
{
    int p = 0;
    for (int i = 0; st[i]; i ++ )
    {
        int u = st[i] - 'a';
        if(!son[p][u]) son[p][u] =
++idx;
        p = son[p][u];
    }
    cnt[p]++;
}
```

```
void query(char st[])
{
    int p = 0;
    for (int i = 0; st[i]; i ++ )
    {
        int u = st[i] - 'a';
        if(!son[p][u])
          {
              printf("%d\n",0);
              return;
          }
        p = son[p][u];
    }
   printf("%d\n",cnt[p]) ;

}
int main()
{
    int n;
    scanf("%d",&n);
    while (n -- )
    {
        char op;
        cin>>op;
        if(op == 'I')
        {
            cin>>st;
            insert(st);
        }
        else
        {
            cin>>st;
            query(st);
        }
    }
    return 0;
}
```

拓展
动态记录元素个数
维护一个记录元素个数的数组
遍历时，判断是否连通 { 是：不变 / 不是：相加
size[find(a)] + size[find(b)]

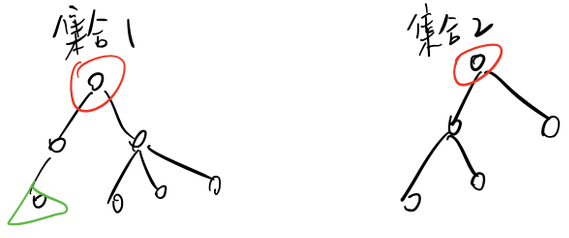
并查集（定义为不重复）

1. 合并两个集合
2. 询问两个元素是否在一集合中

基本原理：每个集合用一颗树表示，树根的编号就是整个集合的编号，每个节点存储父节点 p[x]

两个优化：
路径压缩
按秩合并

p[x] = find(p[x]);



每个点存储父节点

询问在哪个集合 ⇒ 找树根

① 如何判断树根：p[x] == x ?

② 集合编号：while[x != p[x]] ?

③ 如何合并：

把右边/左边的树 插入到某个节点

p[x] = y

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

using namespace std;

const int N = 1e5 + 10;

int p[N];

int find(int a)
{
    if(p[a] != a) p[a] = find(p[a]);
    return p[a];
}
int main()
{
    int n,m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i ++ ) p[i] = i;
    while (m -- ){
        char op;
        int a,b;
        cin>>op>>a>>b;
        if(op == 'M')
        {
            p[find(a)] = p[find(b)];
        }
        else
        {
            if(p[find(a)] == p[find(b)])
printf("Yes\n");
            else printf("No\n");
        }
    }
    return 0;
}
```

堆（修改/删除任意元素）

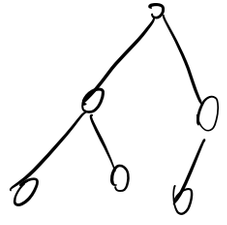
1. 插入一个数
2. 求集合中最小值
3. 删除最小值

STL可支持（优先队列）

---

4. 删除任意元素
5. 修改任意元素

堆：二叉树，完全二叉树



小根堆：每个点小于左右儿子

∴ root → 最小值

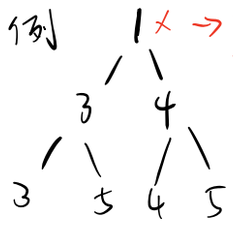
存储

↑ root

{ x的左节点：2x
  x的右节点：2x+1 } → 下标从1开始

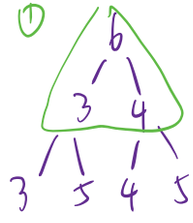
操作：down(x) … 向下调整 上浮 sink

up(x) … 向上调整 上浮 swim

例 

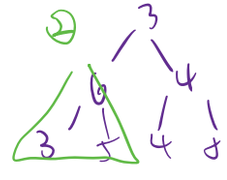
1 × → 6

变大后如何调整该结构？

① 

找最小值 → 3

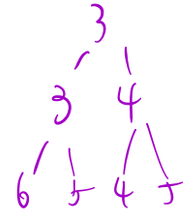
6 ⟷ 3 交换

② 

找最小值 → 3

6 ⟷ 3 交换

直到无法交换



小根堆：

down → sink　数变小 → 下移

up → swim　数变大 → 上移

1. 插入 → heap[++size] = x; up[size];
2. 最小值 → heap[1];
3. 删除最小值 → 用最后一个元素覆盖 root → 并 down(root);
4. 删除任意元素 → heap[k] = heap[size]; size--; down(k); up(k);
5. 修改一个元素 → heap[k] = x; down(k); up(k);

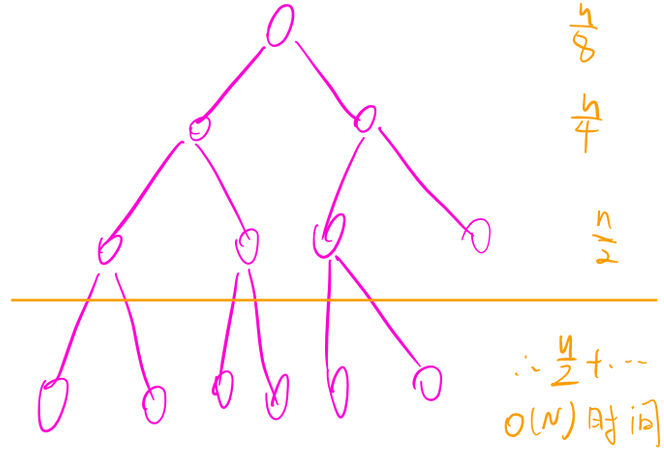
初始化 / down / up

递归操作

初始化

数组建堆 → 叶子不用 down

```
for(int i=n/2; i; i--) down(i);
```



$\therefore \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots < n$

O(N) 时间复杂

ph[k] 第k个插入的点在堆中下标

hp[k] 堆中的点是第几个插入的点（辅助，找到）

双向 map

heap_swap a, b

swap(ph[hp[a]], ph[hp[b]]);

swap(hp[a], hp[b]);

swap(h[a], h[b]);

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;


const int N = 1e5 + 10;

// hp 堆中点的插入次序。ph 次序插入点的下标
int heap[N],hp[N],ph[N];
int idx;

void heapswap(int a,int b)
{
    swap(heap[a],heap[b]);
    swap(ph[hp[a]],ph[hp[b]]);
    swap(hp[a],hp[b]);
}
void sink(int site)
{
    int t = site;

    if( site*2 <= idx &&
heap[t]>heap[site*2]) t = 2*site;
    if( site*2+1 <= idx &&
heap[t]>heap[site*2 + 1]) t = 2*site+1;
    if (t!=site)
    {
        heapswap(site,t);
        sink(t);
    }
}

void swim(int site)
{
    if(site / 2 && heap[site] < heap[site/
2])
    {
        heapswap(site,site/2);
        swim(site/2);
    }
}
```

```
int main()
{
    int n;
    int m = 0;
    scanf("%d", &n);
    while (n -- ){
        string op;
        cin>>op;
        if(op == "I")
        {
            m ++;
            int x;
            scanf("%d",&x);
            heap[++idx] = x;
            hp[idx] = m;
            ph[m] = idx;
            swim(idx);
        }
        else if(op == "PM")
        {
            printf("%d\n",heap[1]);
        }
        else if(op == "DM")
        {
            heapswap(1,idx--);
            sink(1);
        }
        else if(op == "D")
        {
            int x;
            scanf("%d",&x);
            int tmp = ph[x];
            heapswap(tmp,idx--);
            swim(tmp);
            sink(tmp);

        }
        else
        {
            int x,cx;
            scanf("%d%d",&x,&cx);
            int tmp = ph[x];
            heap[tmp ] = cx;
            swim(tmp);
            sink(tmp);
        }
    }
}
```