手写堆+配对堆+左偏树

手写堆

模板/常用操作

```
int tr[N<<2];
#define ls (p<<1)
#define rs (p<<1|1)
//上浮结点
void checkup(int p){ //与父节点比较向上更新
    if(p<=1) return;//已经达到堆顶
    if(tr[p]<tr[p>>1]){ //该节点比父节点要小
        swap(tr[p],tr[p>>1]); //交换两者
        checkup(p>>1); //继续上浮
    }
}

//下沉结点
void checkdown(int p){ //向下更新二叉堆
```

```
    if(p>num) return; //已经无法下沉
    if (tr[p]<tr[ls]&&tr[p]<tr[rs]) return;    //没有比子节点小则返回
    if (tr[ls]<tr[rs]){ //左儿子小于右儿子
        swap(tr[p],tr[ls]); //交换左儿子
        if(p*2<num) checkdown(ls);    //继续下沉
    }else{ //右儿子小于左儿子
        swap(tr[p],tr[rs]);
        if(p*2+1<num) checkdown(rs);
    }
}

//插入结点
void push(long long val){  //插入val
    tr[++num]=val; //先插入到最后的位置
    checkup(num); //对他进行上浮操作
}

//删除结点
void pop(){ //删除最小的数，即tr[1]
    tr[1]=tr[num]; //首先让最后的元素移动到堆顶
    tr[num]=inf; //原来的位置不再有任何元素
    num--;
    checkdown(1); //对堆顶进行下沉操作
}
```

配对堆

简介

配对堆(pairing heap)，支持插入，查询/删除最小值，合并，修改等操作，跑得比较快，但不能可持久化。

操作

合并

直接把根节点权值较大的那个配对堆设成另一个的儿子即可。

插入

新元素可视为一个配对堆，与原堆合并即可。

删除最小值(复杂)

若把根节点的儿子都一个个合并到一起，复杂度会退化到O(n)

Eileen保证均摊复杂度为O(logn)，我们需要把儿子从左往右两两配对。

```
Node* merges(Node* x) {
  if (x == nullptr || x->xd == nullptr)
    return x;  // 如果该树为空或他没有兄弟（即他的父亲的儿子数小于2），就直接return。
  Node *a = x->xd, *b = a->xd;  // a：x的一个兄弟，b：x的另一个兄弟
  x->xd = a->xd = nullptr;      // 拆散
  return merge(merge(x, a), merges(b));  // 核心部分
}
```

减少元素的值

左偏树

简介

左偏树和配对堆一样，是一种可并堆，可实现快速合并。

定义

外节点：当且仅当一个节点的左子树和右子树其中一个为空节点。

点的距离：节点子树中离它最近的外接点到这个点的距离+1。

我们规定空节点的距离为0，外接点距离为1。

性质

性质（1）左偏树中，任何一个节点父节点的权值都要小于等于这个节点的权值。（堆性质，若长成完全二叉树，就是一个堆）

性质（2）左偏树中任意一个节点的左儿子的距离一定大于等于右儿子的距离。（左偏性质）

推论（1）左偏树中任意一个节点的距离为其右儿子距离+1

推论（2）n个点的左偏树，最大距离为log(n+1)-1

操作

Merge(核心)

```
int merge(int x, int y) {
  if (!x || !y) return x | y;  // 若一个堆为空则返回另一个堆
  if (t[x].val > t[y].val) swap(x, y);  // 取值较小的作为根
  t[x].rs = merge(t[x].rs, y);          // 递归合并右儿子与另一个堆
  if (t[t[x].rs].d > t[t[x].ls].d)
    swap(t[x].ls, t[x].rs);   // 若不满足左偏性质则交换左右儿子
  t[x].d = t[t[x].rs].d + 1;  // 更新dist
  return x;
}
```

Push

单个节点可视为一个堆，合并即可。

Pop

删除根节点：合并根的左右儿子即可。

Pop(x)

删除任意节点：先将左右儿子合并，任何自底向上更新dist、不满足左偏性质时交换左右儿子，当dist无需更新时结束递归。

```
int& rs(int x) { return t[x].ch[t[t[x].ch[1]].d < t[t[x].ch[0]].d]; }

// 有了 pushup，直接 merge 左右儿子就实现了删除节点并保持左偏性质
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (t[x].val < t[y].val) swap(x, y);
  t[rs(x) = merge(rs(x), y)].fa = x;
  pushup(x);
  return x;
}

void pushup(int x) {
  if (!x) return;
  if (t[x].d != t[rs(x)].d + 1) {
    t[x].d = t[rs(x)].d + 1;
    pushup(t[x].fa);
  }
}

int pop(int x) {
  return merge(t[x].ls, t[x].rs);
}
```

随即合并

```
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (t[y].val < t[x].val) swap(x, y);
  if (rand() & 1)  //随机选择是否交换左右子节点
    swap(t[x].ls, t[x].rs);
  t[x].ls = merge(t[x].ls, t[y]);
  return x;
}
```

例题

luoguP3377 【模板】左偏树（可并堆）

```
#include<bits/stdc++.h>
#define inf 0x3f3f3f3f
#define int long long
#define ls tr[x].ch[0]
#define rs tr[x].ch[1]

using namespace std;
const int maxn=1e5+7;
const int mod=1e9+7;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}

struct X{
    int dis,val,ch[2],fa;
}tr[maxn];

int get(int x){
    return tr[x].fa==x?x:tr[x].fa=get(tr[x].fa);
}

int merge(int x,int y){
    if(!x||!y) return x+y;
    if((tr[x].val>tr[y].val)||(tr[x].val==tr[y].val&&x>y)) swap(x,y);
    rs=merge(rs,y);
    if(tr[ls].dis<tr[rs].dis) swap(ls,rs);
    tr[ls].fa=tr[rs].fa=tr[x].fa=x;
    tr[x].dis=tr[rs].dis+1;
    return x;
}

inline void pop(int x){
    tr[x].val=-1;
    tr[ls].fa=ls;
    tr[rs].fa=rs;
    tr[x].fa = merge(ls,rs);
}

int n,m;

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n>>m;
    tr[0].dis=0;
    int op,x,y;
    for(int i=1;i<=n;i++){
        cin>>tr[i].val;
        tr[i].fa=i;
    }
    for(int i=1;i<=m;i++){
        cin>>op;
```

```
        if(op==1){
            cin>>x>>y;
            if(tr[x].val==-1||tr[y].val==-1) continue;
            int f1=get(x),f2=get(y);
            if(f1!=f2){
                tr[f1].fa=tr[f2].fa=merge(f1,f2);
            }
        }else{
            cin>>x;
            if(tr[x].val==-1){
                cout<<"-1\n";
            }else{
                cout<<tr[get(x)].val<<"\n";
                pop(get(x));
            }
        }
    }
    return 0;
}
```

P2713 罗马游戏

给定一个序列，支持以下操作：

- M i j，将编号为i和j所在的两个团合并，如果i或j是死人，跳过
- K i,杀死i所在团分数最低的人，如果已经死了，跳过

```
#include<iostream>
using namespace std;
const int N=1e6+1;

int n,m,ls[N],rs[N],fa[N],d[N],val[N];
bool dead[N];
char op;

int find(int x){  //寻找父亲
    if(fa[x]==x) return x;
    return find(fa[x]);
}

int merge(int x,int y){
    if(!x||!y) return x+y;
    if(val[x]>val[y]) swap(x,y);
    rs[x]=merge(rs[x],y);
    if(d[ls[x]]<d[rs[x]]) swap(ls[x],rs[x]); //交换左右儿子
    d[x]=d[rs[x]]+1;
    return x;
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n;
    d[0]=-1;
    for(int i=1;i<=n;++i){
        cin>>val[i];
```

```
        fa[i]=i;
    }
    cin>>m;
    for(int i=1,l,r,x;i<=m;++i){
        cin>>op;
        if(op=='M'){
            cin>>l>>r;
            if(dead[l]||dead[r]) continue;
            int fx=find(l),fy=find(r);
            if(fx!=fy) fa[fx]=fa[fy]=merge(fx,fy);
        }else{  //删除
            cin>>x;
            if(dead[x]){
                cout<<0<<"\n";
                continue;
            }
            int f=find(x);
            cout<<val[f]<<"\n";
            dead[f]=1;
            fa[f]=fa[ls[f]]=fa[rs[f]]=merge(ls[f],rs[f]); //合并两个堆
            ls[f]=rs[f]=d[f]=0;
        }
    }
    return 0;
}
```

参考资料

OI-wiki

https://blog.csdn.net/wang3312362136/article/details/80615874

https://baike.baidu.com/item/%E9%85%8D%E5%AF%B9%E5%A0%86/22690562

https://www.cnblogs.com/pks-t/p/10326682.html

https://blog.csdn.net/qq_41900081/article/details/86670001

https://blog.csdn.net/qq_39445165/article/details/84932335