

# 左偏树

## 算法简介

一种可并堆，支持  $O(\log n)$  的  $\text{push}$ 、 $\text{pop}$ 、 $\text{merge}$  操作。

## 算法思想

定义外节点为左儿子或右儿子为空的节点。  $\text{dist}_i$  表示节点  $i$  到其子树中最近外节点的距离，规定空节点的  $\text{dist}$  为  $-1$ 。

左偏树的定义为对每个节点  $u$  均满足  $\text{dist}_{\text{lson}} \geq \text{dist}_{\text{rson}}$  的堆。

左偏树有如下性质

- $\text{dist}_u = \text{dist}_{\text{rson}} + 1$
- $\text{dist}_u \sim O(\log \text{sz}_u)$

关于性质一，有

$\text{dist}_u = \max(\text{dist}_{\text{lson}}, \text{dist}_{\text{rson}}) + 1 = \text{dist}_{\text{rson}} + 1$  证毕。

关于性质二，有  $\text{dist}_u$  表示节点  $u$  到其子树中最近外节点的距离，所以有节点  $u$  的子树的前  $\text{dist}_u$  层均为非外节点。

所以可将前  $\text{dist}_u$  层视为满二叉树，第  $\text{dist}_u + 1$  层至少有一个外节点，所以有  $\text{sz}_u \geq 2^{\text{dist}_u}$  证毕。

考虑左偏树的合并操作，类似  $\text{fhq treap}$  的  $\text{merge}$  操作，根据优先级(这里是点权)不断合并两棵树，遇到空结点返回。

不同之处在于合并两棵左偏树时对跳左节点还是右节点没有限制，所以强制每次跳右节点，使得  $\text{dist}_u$  单调递减。

这样，时间复杂度为  $O(\text{dist}_u) = O(\log n)$

左偏树的另一个核心操作为寻根，事实上寻根操作可以利用路径压缩的并查集优化时间复杂度到  $O(\log n)$  但需要注意一些细节，详细见代码。

## 代码模板

### 洛谷p3377

左偏树的  $\text{pop}$  操作类似  $\text{fhq treap}$  直接合并被删除节点的左右儿子，但是删除祖先节点会导致并查集出错。

具体解决方案为  $\text{root}[p] = \text{root}[\text{node}[p].\text{ch}[0]] = \text{root}[\text{node}[p].\text{ch}[1]] = \text{rt}$ ，即把原堆顶元素及其儿子节点的祖先节点指向新根节点。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n' ||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5;
struct Left_Heap{
```

```

int root[MAXN];
bool del[MAXN];
struct Node{
    int id,val,dis,ch[2];
    bool operator < (const Node &b)const{return
val<b.val||(val==b.val&&id<b.id);}
}node[MAXN];
void build(int *a,int n){
    node[0].dis=-1;
    _rep(i,1,n)
    node[i].id=root[i]=i,node[i].val=a[i];
}
int find_root(int x){return x==root[x]?x:root[x]=find_root(root[x]);}
void merge(int &k,int k1,int k2){
    if(!k1||!k2)return k=k1|k2,void();
    if(node[k2]<node[k1])swap(k1,k2);
    merge(node[k=k1].ch[1],node[k1].ch[1],k2);
    if(node[node[k].ch[0]].dis<node[node[k].ch[1]].dis)
    swap(node[k].ch[0],node[k].ch[1]);
    node[k].dis=node[node[k].ch[1]].dis+1;
}
void merge(int x,int y){
    if(del[x]||del[y])return;
    int rt,p1=find_root(x),p2=find_root(y);
    if(p1!=p2){
        merge(rt,p1,p2);
        root[p1]=root[p2]=rt;
    }
}
int top(int x){
    if(del[x])return -1;
    return node[find_root(x)].val;
}
void pop(int x){
    if(del[x])return;
    int rt,p=find_root(x);
    merge(rt,node[p].ch[0],node[p].ch[1]);
    root[p]=root[node[p].ch[0]]=root[node[p].ch[1]]=rt;
    del[p]=true;
}
}heap;
int a[MAXN];
int main()
{
    int n=read_int(),m=read_int(),opt,x;
    _rep(i,1,n)
    a[i]=read_int();
    heap.build(a,n);
    while(m--){
        opt=read_int(),x=read_int();
        switch(opt){

```

```
        case 1:
            heap.merge(x, read_int());
            break;
        case 2:
            enter(heap.top(x));
            heap.pop(x);
    }
}
return 0;
}
```

## 算法习题

### 习题一

[洛谷p1456](#)

#### 题意

维护  $n$  个堆，每个堆一开始只有一个元素  $m$  个操作。

每次选取两个堆，将堆顶元素减半，然后合并。

#### 题解

修改堆顶元素方法为先合并左右儿子，然后修改堆顶，再次加入。同样要注意并查集与维护。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
```

```

#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n' ||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5;
struct Left_Heap{
    int root[MAXN];
    struct Node{
        int id,val,dis,ch[2];
        bool operator < (const Node &b)const{return
val>b.val||(val==b.val&&id<b.id);}
    }node[MAXN];
    void build(int *a,int n){
        node[0].dis=-1;
        _rep(i,1,n)
node[i].id=root[i]=i,node[i].val=a[i],node[i].ch[0]=node[i].ch[1]=node[i].d
is=0;
    }
    int find_root(int x){return x==root[x]?x:root[x]=find_root(root[x]);}
    void merge(int &k,int k1,int k2){
        if(!k1||!k2)return k=k1|k2,void();
        if(node[k2]<node[k1])swap(k1,k2);
        merge(node[k=k1].ch[1],node[k1].ch[1],k2);
        if(node[node[k].ch[0]].dis<node[node[k].ch[1]].dis)
swap(node[k].ch[0],node[k].ch[1]);
        node[k].dis=node[node[k].ch[1]].dis+1;
    }
};

```

```
}
int merge(int x,int y){
    int rt,p1=find_root(x),p2=find_root(y);
    if(p1==p2)return -1;
    update(p1);update(p2);
    p1=find_root(x),p2=find_root(y);
    merge(rt,p1,p2);
    root[p1]=root[p2]=rt;
    return node[rt].val;
}
void update(int rt){
    int temp,Root;
    merge(temp,node[rt].ch[0],node[rt].ch[1]);
    node[rt].val>>=1;node[rt].ch[0]=node[rt].ch[1]=node[rt].dis=0;
    merge(Root,temp,rt);
    root[Root]=root[rt]=Root;
}
}heap;
int a[MAXN];
int main()
{
    int n,m,x,y;
    while(~scanf("%d",&n)){
        _rep(i,1,n)
            a[i]=read_int();
        heap.build(a,n);
        m=read_int();
        _rep(i,1,m){
            x=read_int(),y=read_int();
            enter(heap.merge(x,y));
        }
    }
    return 0;
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: [https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E5%B7%A6%E5%81%8F%E6%A0%91&rev=1594281156](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%B7%A6%E5%81%8F%E6%A0%91&rev=1594281156)

Last update: 2020/07/09 15:52