# 数据结构其他模板



## 树状数组

### 区间修改+区间查询

```
#include<bits/stdc++.h>
#define lowbit(x) (x&-x)
#define int long long
#define range_ask(l,r) query(r)-query(l-1)
#define range_add(l,r,x) add(l,x),add(r+1,-x)
using namespace std;
const int maxn=1e6+7;
int n,q,op,l,r,tmp,sum1[maxn],sum2[maxn],a[maxn];

void add(int x,int y){
    for(int i=x;i<=n;i+=lowbit(i)){
        sum1[i]+=y;
        sum2[i]+=x*y;
    }
}

int query(int x){
    int res=0;
    for(int i=x;i;i-=lowbit(i)){
        res+=(x+1)*sum1[i]-sum2[i];
    }
    return res;
}

signed main(){
```

```
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>q;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        tmp=a[i]-a[i-1];
        add(i,tmp);
    }
    for(int j=1;j<=q;j++){
        cin>>op;
        if(op==1){
            cin>>l>>r>>tmp;
            range_add(l,r,tmp);
        }
        if(op==2){
            cin>>l>>r;
            cout<<range_ask(l,r)<<endl;
        }
    }
    return 0;
}
```

**二维单点修改+区间查询**

```
#include<bits/stdc++.h>
#define lowbit(x) (x&-x)
#define int long long

using namespace std;
const int maxn=5000;
int n,m,op,a,b,c,d,tmp,tr[maxn][maxn];

void add(int x,int y,int z){
    for(int i=x;i<=n;i+=lowbit(i)){
        for(int j=y;j<=m;j+=lowbit(j)){
            tr[i][j]+=z;
        }
    }
}

int ask(int x,int y){
    int res=0;
    for(int i=x;i;i-=lowbit(i)){
        for(int j=y;j;j-=lowbit(j)){
            res+=tr[i][j];
        }
    }
    return res;
}

int range_ask(int xa,int ya,int xb,int yb){
    return ask(xb,yb)-ask(xb,ya-1)-ask(xa-1,yb)+ask(xa-1,ya-1);
}

signed main(){
    scanf("%lld%lld",&n,&m);
    while(scanf("%lld",&op)!=EOF){
        if(op==1){
```

```
            scanf("%lld%lld%lld",&a,&b,&c);
            add(a,b,c);
        }
        if(op==2){
            scanf("%lld%lld%lld%lld",&a,&b,&c,&d);
            printf("%lld\n",range_ask(a,b,c,d));
        }
    }
    return 0;
}
```

**二维区间修改+单点查询**

```
#include<bits/stdc++.h>
#define lowbit(x) (x&-x)
#define int long long

using namespace std;
const int maxn=5000;
int n,m,op,a,b,c,d,tmp,tr[maxn][maxn];

void add(int x,int y,int z){
    for(int i=x;i<=n;i+=lowbit(i)){
        for(int j=y;j<=m;j+=lowbit(j)){
            tr[i][j]+=z;
        }
    }
}

int ask(int x,int y){
    int res=0;
    for(int i=x;i;i-=lowbit(i)){
        for(int j=y;j;j-=lowbit(j)){
            res+=tr[i][j];
        }
    }
    return res;
}

void range_add(int xa,int ya,int xb,int yb,int k){
    add(xa,ya,k);
    add(xa,yb+1,-k);
    add(xb+1,ya,-k);
    add(xb+1,yb+1,k);
}

signed main(){
    scanf("%lld%lld",&n,&m);
    while(scanf("%lld",&op)!=EOF){
        if(op==1){
            scanf("%lld%lld%lld%lld%lld",&a,&b,&c,&d,&tmp);
            range_add(a,b,c,d,tmp);
        }
        if(op==2){
            scanf("%lld%lld",&a,&b);
            printf("%lld\n",ask(a,b));
        }
```

```
    }
    return 0;
}
```

# pb_ds

### 声明

```
 #include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>//用tree
#include<ext/pb_ds/hash_policy.hpp>//用hash
#include<ext/pb_ds/trie_policy.hpp>//用trie
#include<ext/pb_ds/priority_queue.hpp>//用priority_queue
//#include<bits/extc++.h> 万能头好像dev c++用不了
using namespace __gnu_pbds;
```

### 哈希表

因为$map$的操作是$O(logn)$的，有时候可以换成hash来加速一下。

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace __gnu_pbds;

cc_hash_table<string,int>mp1;//拉链法
gp_hash_table<string,int>mp2;//查探法（快一点）
```

### 堆

```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
__gnu_pbds::priority_queue<int>q;//带命名空间防止和std重复
__gnu_pbds::priority_queue<int,greater<int>,binary_heap_tag>pq;//二叉堆
__gnu_pbds::priority_queue<int,greater<int>,binomial_heap_tag>pq;//二项堆
__gnu_pbds::priority_queue<int,greater<int>,rc_binomial_heap_tag>pq;
__gnu_pbds::priority_queue<int,greater<int>,thin_heap_tag>pq;
__gnu_pbds::priority_queue<int,greater<int> >pq;

//常用操作
push() //返回一个迭代器
top() //返回堆顶
size() //返回大小
empty() //判断是否为空
clear() //清空堆
pop() //弹出堆顶元素
join(priority_queue,&other) //合并两个堆.other会被清空
split(Pred prd,priority_queue &other) //分离除两个堆
modify(point_iterator it,const key //修改一个节点的值)
```

### Trie

目前还没有开发出什么有趣的用法。

```
typedef
trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,trie_prefix_searc
h_node_update> tr;
//第一个参数必须为字符串类型，tag也有别的tag，但pat最快，与tree相同，node_update支持自定义
tr.insert(s); //插入s
tr.erase(s); //删除s
tr.join(b); //将b并入tr
pair//pair的使用如下：
pair<tr::iterator,tr::iterator> range=base.prefix_range(x);
for(tr::iterator it=range.first;it!=range.second;it++)
    cout<<*it<<' '<<endl;
//pair中第一个是起始迭代器，第二个是终止迭代器，遍历过去就可以找到所有字符串了。
```

### 平衡树

rb_tree用起来应该会比splay_tree和ov_tree更快。

```
#include<ext/pb_ds/tree_policy.hpp>
#include<ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>t;
/*定义一颗红黑树，
int为关键字类型，
null_type无映射，
less从小到大排序，
rb_tree_tag红黑树(splay_tree_tag)，
tree_order_statistics_node_update节点更新
*/
// 常用方法
t.insert()//插入
t.erase()//删除
t.order_of_key(k);//求k在树中是第几大
t.find_by_order(k); //求树中的第k大
t.lower_bound();//前驱
t.upper_bound();//后继
a.join(b)//b并入a，前提是两棵树的key的取值范围不相交
a.split(v,b)//key小于等于v的元素属于a，其余的属于b
t.lower_bound(x)//>=x的min的迭代器
t.upper_bound(x)//>x的min的迭代器
tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update>::ite
rator it;//迭代器也可以使用auto来判断
```

### 普通平衡树模板

```
#include<bits/stdc++.h>
#include<ext/pb_ds/tree_policy.hpp>
#include<ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
tree<ll,null_type,std::less<ll>,rb_tree_tag,tree_order_statistics_node_update>bt
;
ll T,ans,op,x;
```

```
signed main(){
    scanf("%lld",&T);
    for(int i=1;i<=T;i++){
        scanf("%lld%lld",&op,&x);
        if(op==1) bt.insert((x<<20)+i); //插入
        else if(op==2) bt.erase(bt.lower_bound(x<<20));  //删除
        else if(op==3) printf("%lld\n",bt.order_of_key(x<<20)+1);//查询x的排名
        else{
            if(op==4) ans=*bt.find_by_order(x-1); //查询排名为x的数
            if(op==5) ans=*--bt.lower_bound(x<<20);  //查询前驱
            if(op==6) ans=*bt.lower_bound((x+1)<<20);  //查询后继
            printf("%lld\n",ans>>20);
        }
    }
    return 0;
}
```

**堆优化dijkstral**

不知道用pbds写这玩意会有什么奇效。

```
__gnu_pbds::priority_queue<pair<int,int>,greater<pair<int,int> > >G;
__gnu_pbds::priority_queue<pair<int,int>,greater<pair<int,int> >
>::point_iterator its[MAXN];
static int dis[MAXN];
const int INF=2e9+7;
inline void dijkst(int s){
    G.clear();
    its[s]=G.push(make_pair(0,s));dis[s]=0;
    Rep(i,2,n)dis[i]=INF,its[i]=G.push(make_pair(INF,i));
    static int u;
    while(!G.empty()){
        u=G.top().second;G.pop();
        for(register int v=head[u];v;v=P[v].nxt)
            if(dis[P[v].v]>dis[u]+P[v].w){
                dis[P[v].v]=dis[u]+P[v].w;
                G.modify(its[P[v].v],make_pair(dis[u]+P[v].w,P[v].v));
            }
    }
}
```

# 笛卡尔树

笛卡尔树是一种二叉树，每一个结点由一个键值二元组$(k,w)$构成。要求$k$满足二叉搜索树的性质，而$w$满足堆的性质。一个有趣的事实是，如果笛卡尔树的$k,w$键值确定，且$k$互不相同，$w$互不相同，那么这个笛卡尔树的结构是唯一的。

**P5854 【模板】笛卡尔树**

给定一个$1\sim n(\leq 10^7)$的排列$p$，构建其笛卡尔树。

即构建一棵二叉树，满足：

1. 每个节点的编号满足二叉搜索树的性质。

2. 节点 $i$ 的权值为$p_i$，每个节点的权值满足小根堆的性质。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e7+7;
typedef long long ll;

int n,a[N],stk[N],ls[N],rs[N];
ll L,R;

ll read(){ll 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;}
void write(ll x){if(x>9) write(x/10);putchar(x%10+'0');}

signed main(){
    n=read();
    //用单调栈维护一个权值单调递增的下标序列，
    //插入一个点到栈中第一个比他小的点做右儿子，最后一个比它大的数做做儿子。
    int pos=0,top=0;
    for(int i=1;i<=n;++i){
        a[i]=read();
        pos=top;
        while(pos&&a[stk[pos]]>a[i]) pos--;
        if(pos) rs[stk[pos]]=i;
        if(pos<top) ls[i]=stk[pos+1];
        stk[top=++pos]=i;
    }
    for(int i=1;i<=n;++i) L^=1ll*i*(ls[i]+1),R^=1ll*i*(rs[i]+1);
    write(L);putchar(' ');
    write(R);putchar('\n');
    return 0;
}
```

**P1377 [TJOI2011] 树的序**

给一个生成序列，建出一棵笛卡尔树，求字典序最小的可以得到相同笛卡尔树的生成序列。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+7;
typedef long long ll;

int n,a[N],stk[N],ls[N],rs[N];
ll L,R;

ll read(){ll 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;}
void write(ll x){if(x>9) write(x/10);putchar(x%10+'0');}

void dfs(int x){
    if(!x) return;
    printf("%d ",x);
    dfs(ls[x]),dfs(rs[x]);
```

```
}

signed main(){
    n=read();
    //与模板不同的是：把权值作为下标，下标作为权值，可得到字典序最小。
    int pos=0,top=0;
    for(int i=1,x;i<=n;++i) x=read(),a[x]=i;
    for(int i=1,x;i<=n;++i){
        pos=top;
        while(pos&&a[stk[pos]]>a[i]) pos--;
        if(pos) rs[stk[pos]]=i;
        if(pos<top) ls[i]=stk[pos+1];
        stk[top=++pos]=i;
    }
    dfs(stk[1]);
    return 0;
}
```

## 参考资料


OI -wiki

https://blog.csdn.net/riba2534/article/details/80454602

《C的pb_ds库在OI中的应用》于纪平

https://blog.csdn.net/TRiddle/article/details/71001764

https://www.cnblogs.com/zwfymqz/p/7264240.html