pbds

头文件

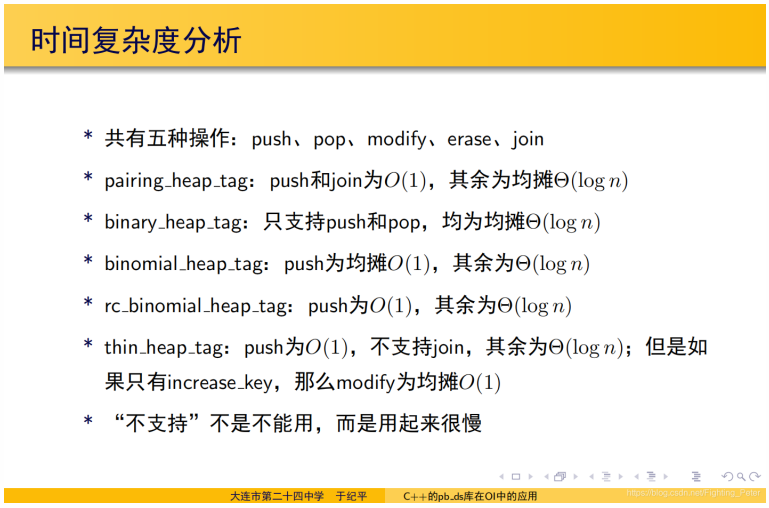
```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>    // 用tree
#include<ext/pb_ds/hash_policy.hpp>    // 用hash
#include<ext/pb_ds/trie_policy.hpp>    // 用trie
#include<ext/pb_ds/priority_queue.hpp>// 用priority_queue
using namespace __gnu_pbds;
---
#include<bits/extc++.h>
using namespace __gnu_pbds;
// bits/extc++.h与bits/stdc++.h类似，bits/extc++.h是所有拓展库，bits/stdc++.h是所有标准库
```

哈希表

```
#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(快一些)
```

可并堆




```
#include<ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
__gnu_pbds::priority_queue<int>q;//因为放置和std重复，故需要带上命名空间
__gnu_pbds::priority_queue<int,greater<int>,pairing_heap_tag> q;//最快
__gnu_pbds::priority_queue<int,greater<int>,binary_heap_tag> q;
__gnu_pbds::priority_queue<int,greater<int>,binomial_heap_tag> q;
__gnu_pbds::priority_queue<int,greater<int>,rc_binomial_heap_tag> q;
__gnu_pbds::priority_queue<int,greater<int>,thin_heap_tag> q;
```

```
push()  //会返回一个迭代器
top()    //同 STL
size()   //同 STL
empty()  //同 STL
clear()  //同 STL
pop()    //同 STL
join(priority_queue &other)            //合并两个堆,other会被清空
split(Pred prd,priority_queue &other)  //分离出两个堆
modify(point_iterator it,const key)    //修改一个节点的值
```

操作\数据结构	配对堆	二叉堆	左偏树	二项堆	斐波那契堆
插入 (insert)	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$
查询最小值 (find-min)	$O(1)$	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$
删除最小值 (delete-min)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
合并 (merge)	$O(1)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$
减小一个元素的值 (decrease-key)	$o(\log n)$（下界 $\Omega(\log\log n)$，上界 $O(2^{2\sqrt{\log\log n}})$）	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$

平衡树

```
template<typename T>
using ordered_set = tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>;
// rb_tree_tag 和 splay_tree_tag 选择树的类型(红黑树和伸展树)

null_type // 无映射（老版本g++为null_mapped_type）
less<T> // Node的排序方式从小到大排序
tree_order_statistics_node_update// 参数表示如何更新保存节点信息
tree_order_statistics_node_update会额外获得order_of_key()和find_by_order()两个功能。

less<node>
bool operator <(node a,node b) { return a.z<b.z; }
greater<node>
bool operator >(node a,node b) { return a.z>b.z; }

ordered_set<Node> Tree;  // Node 自定义struct 注意重载less
Tree.insert(Node);       // 插入
Tree.erase(Node);        // 删除
Tree.order_of_key(Node); // 求树中严格比x小的元素的个数，x不一定在树中
Tree.find_by_order(k);   // 0-index，返回迭代器
Tree.join(b);            // 前提是两棵树的key的取值范围不相交，合并后b被清空
Tree.split(v, b);        // 分裂，key小于等于v的元素属于Tree，其余属于b
Tree.lower_bound(Node);  // 返回第一个大于等于x的元素的迭代器
Tree.upper_bound(Node);  // 返回第一个大于x的元素的迭代器

// 以上的所有操作的时间复杂度均为O(logn)
```

```
// 注意，插入的元素会去重，如set
ordered_set<T>::point_iterator it=Tree.begin(); // 迭代器
// 显然迭代器可以++，--运算
```