**其他**

2020.09.05

目录

[其他算法 2](#_Toc50208295)

[从pbds、rope到stl数据结构的奇技淫巧 2](#_Toc50208296)

[pbds 2](#_Toc50208297)

[rope 5](#_Toc50208298)

[stl 5](#_Toc50208299)

# 其他算法

## 从pbds、rope到stl数据结构的奇技淫巧

### pbds

pbds常用的有**哈希表**,**平衡树**,以及各种**堆**等数据结构。

首先使用pbds都要加上**using namespace \_\_gnu\_pbds;**

#### 哈希表

#include<ext/pb\_ds/assoc\_container.hpp>  
#include<ext/pb\_ds/hash\_policy.hpp>

cc\_hash\_table<int, bool> hash1; //拉链法处理冲突  
gp\_hash\_table<int, bool> hash2; //探测法处理冲突（较快）

##### 使用

直接hash1[a]=x，类似map。

使用hash1.find(a)，查找一个键值。

#### 平衡树

#include <ext/pb\_ds/tree\_policy.hpp>  
#include <ext/pb\_ds/assoc\_container.hpp>

tree<nod, null\_type, less<nod>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> t;//第二维有可能是null\_mapped\_type(老版本)

rb*tree*tag一维可选其他的，但都太慢了。

**会去重！**& **迭代器可以--**

##### 使用

.insert(一个元素)

.erase(一个元素)

.order\_of\_key(一个元素) //求k在树中是第几大

.find\_by\_order(一个数)//查第几大是谁

.lower\_bound() .upper\_bound()

.join(另一个树)//要求元素值区间不相交，全大或全小

.split(v,b)key小于等于v的元素属于a，其余的属于b

##### 高能使用

自定义类来实现求前缀和之类的操作。

tree*order*statistics*node*update这一维是可以自定义结构体的。

首先必须的结构是

template<class Node\_CItr,class Node\_Itr,class Cmp\_Fn,class \_alloc>   
struct my\_node\_update  
{  
 virtual Node\_CItr node\_begin() const =0;  
 virtual Node\_CItr node\_end() const =0;  
 typedef int metadata\_type;  
};

在这个struct里面要做的就是我们想做的事情啦，添加一些和结点相关的数据，即插入的时候有一个含有数据的第二维，相当于插入的数据是一个结构体。

首先肯定要定义一个合并左结点和右节点信息的函数，以下为一个例子。

void operator()(Node\_Itr it, Node\_CItr end\_it)  
{  
 Node\_Itr l=it.get\_l\_child();  
 Node\_Itr r=it.get\_r\_child();  
 int left=0,right=0;  
 if(l!=end\_it) left=l.get\_metadata();  
 if(r!=end\_it) right=r.get\_metadata();  
 const\_cast<int&> (it.get\_metadata())=left+right+(\*it)->val;  
}

可以在函数内部自定义一些方法，如下，相当于在平衡树上二分。

int prefix\_sum(int x)  
{  
 int ans=0;  
 Node\_CItr it=node\_begin();  
 while(it!=node\_end())  
 {  
 Node\_CItr l=it.get\_l\_child(),r=it.get\_r\_child();  
 if(Cmp\_Fn()(x,(\*it)->first)) it=l;  
 else  
 {  
 ans+=(\*it)->second;  
 if(l!=node\_end())  
 ans+=l.get\_metadata();  
 it=r;  
 }  
 }  
 return ans;  
}

在内部主要使用的函数就是.get*l*child() .get*r*child() .get\_metadata()

自己定义的时候就可以改为 tree<nod, null\_type, less<nod>, rb\_tree\_tag, my\_node\_update> t;

#### 堆

#include<ext/pb\_ds/priority\_queue.hpp>

\_\_gnu\_pbds ::priority\_queue<nod, less<nod>, pairing\_heap\_tag> a;//要加命名空间防冲突

第一维定义数据类型，第二维greater是大根堆，less是小根堆，第三维选择堆的类型。

##### 堆类型的选择

使用合并的时候使用 pairing*heap*tag

做dijisktra的时候使用 thin*heap*tag

##### 用法

.push(一个元素) .pop() .top() .empty() .clear()

.modify(一个迭代器，一个值)

.erase(一个迭代器)

.join(另外一个堆)//最nb的功能！

.split(Pred prd,priority\_queue &other)//分成两个堆，在树里可能比较好用

### rope

首先要带头文件 #include<ext/rope>

使用rope还需要带命名空间 using namespace \_\_gnu\_cxx;

定义方法 rope<变量类型>变量名称

复杂度基本都为级别，据说的字符串只需要不到的空间！

#### 用法

在字符串和整数类型的存储应用较广，在此只讲这两种。

插入s的前n位：insert(int 起始位置, string &s/int \*a, int 长度)

向末尾插入s的pos开始的n位： append(string &s/int \*a,int 串s的起始位置,int 长度)

取子串：substr(int 起始, int 长度)

取字符：at(int x)

删除子串：erase(int 起始, int 长度)

将rope从pos开始的len个字符用s替代copy(int pos, int len, string &s/int \*a)

将rope从pos开始的字符用s替代 replace(int pos, string &s/int \*a);

### stl

#### set

一个元素只能出现一次

删除元素之后结构可能发生改变，因此不要边删边跑迭代器。迭代完之后统一删除。

#### multiset

可以出现多个元素

删除一个值，删除所有等于该值的结点

要只删除一个值，就删除一个等于该值的指针

#### biset

foo.size() 返回大小（位数）  
f11oo.count() 返回1的个数  
foo.any() 返回是否有1  
foo.none() 返回是否没有1  
foo.set() 全都变成1  
foo.set(p) 将第p + 1位变成1  
foo.set(p, x) 将第p + 1位变成x  
foo.reset() 全都变成0  
foo.reset(p) 将第p + 1位变成0  
foo.flip() 全都取反  
foo.flip(p) 将第p + 1位取反  
foo.to\_ulong() 返回它转换为unsigned long的结果，如果超出范围则报错  
foo.to\_ullong() 返回它转换为unsigned long long的结果，如果超出范围则报错  
foo.to\_string() 返回它转换为string的结果