

__gnu_pbds食用教程

引入

某P党：“你们C++的STL库真强（e）大（xin），好多数据结构和算法都不用手打。”

C党1：“STL能省下的代码量又不多，平衡树多难调啊。”

C党2：“欸？__gnu_pbds库就可以做到啊，它封装了hash,tree,trie,priority_queue这四种数据结构。”

正文

介绍

什么是__gnu_pbds？Policy based data structures！简称平板电脑pbds。在使用pbds前，你需要

```
#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;
```

woc，真jb烦，有没有什么简单的方法？当然有：

```
#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
```

但是在dev c++里如果这样写，会提示少一个文件，出各种莫名其妙的锅，其它的IDE请自行尝试，我的linux是deepin的，装了NOI Linux的dalao帮忙测一下。

hash

该引用的头文件和命名空间都讲过了，直接进入正题。

hash_table的用法与map类似，它是这么定义的：

```
cc_hash_table<int,bool> h;
gp_hash_table<int,bool> h;
```

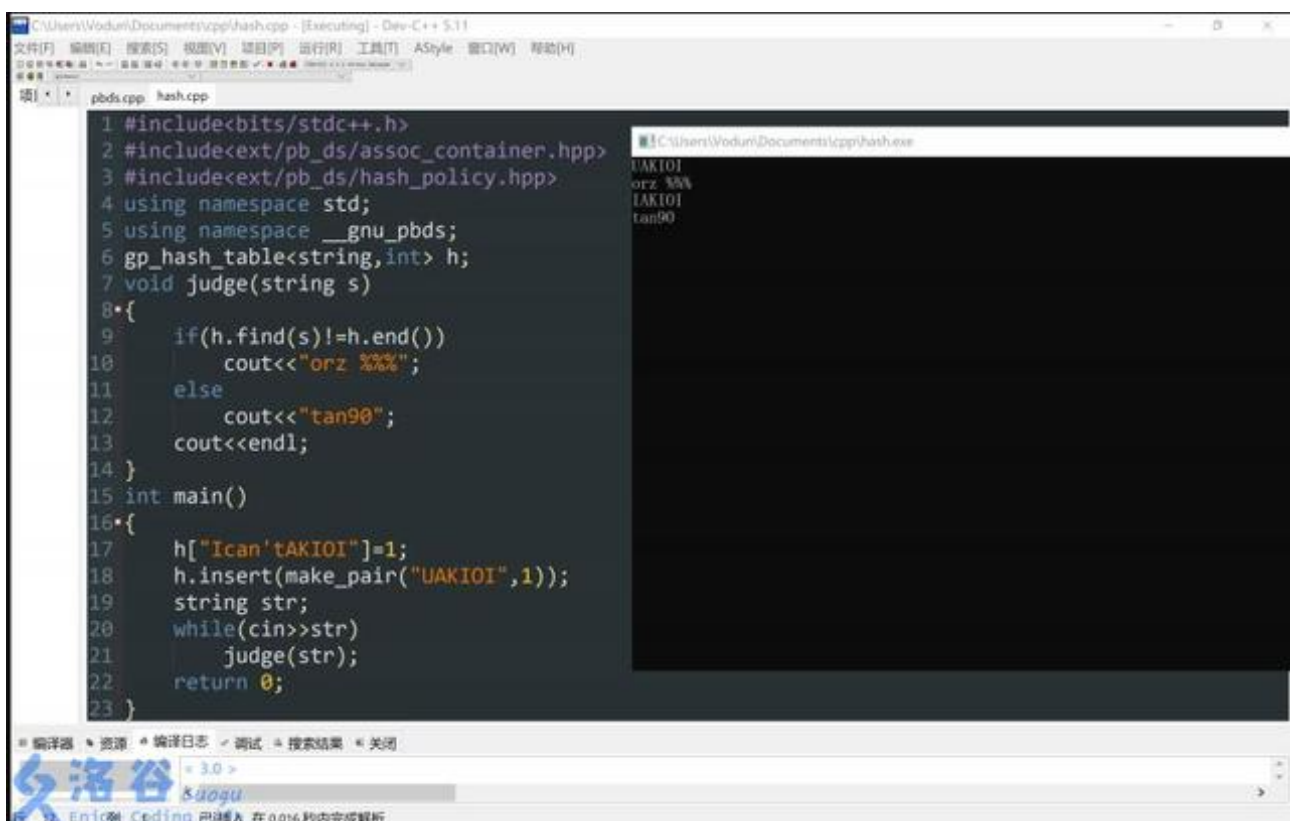
其中cc开头为拉链法，gp开头为探测法，个人实测探测法稍微快一些。

啥？操作？其实就和map差不多，支持[]和find。

```

#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
gp_hash_table<string,int> h;
void judge(string s)
{
    if(h.find(s)!=h.end())
        cout<<"orz %%%";
    else
        cout<<"tan90";
    cout<<endl;
}
int main()
{
    h["Ican'tAKIOI"]=1;
    h.insert(make_pair("UAKIOI",1));
    string str;
    while(cin>>str)
        judge(str);
    return 0;
}

```



The screenshot shows a Dev-C++ IDE window titled "C:\Users\Vodum\Documents\cpp\hash.cpp - [Executing] - Dev-C++ 5.11". The code editor on the left contains the same C++ code as shown in the previous block. The output window on the right, titled "C:\Users\Vodum\Documents\cpp\hash.exe", displays the program's output:

```

UAKIOI
orz %%%
AKIOI
tan90

```

At the bottom of the IDE, there is a status bar with the text "EnjC 例 Ceding 已插入 在 0.016 秒内完成解析".

等一等？和map一样，那不如直接用map了。不不不，map的总时间复杂度是 $O(n \log n)$ 的，而hash_table的总时间复杂度仅为 $O(n)$ ！所以我们可以用这个特性来做洛谷P1333 瑞瑞的木棍 (<https://www.luogu.org/problemnew/show/P1333>)。前置知识：并查集

(<https://www.luogu.org/blog/41785/jian-yi-bing-zha-ji>) 欧拉路
(<https://www.luogu.org/blog/lzhbigbird/zong-qi-qiao-wen-ti-dao-ou-la-lu>) 。

感谢Great_Influence的代码：

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/hash_policy.hpp> //pb_ds库
#include<ccctype>
#define For(i,a,b) for(i=(a);i<=(b);++i)
#define Forward(i,a,b) for(i=(a);i>=(b);--i)
using namespace std;
using namespace __gnu_pbds; //命名空间
template<typename T> inline void read(T &x){
    T s=0,f=1;char k=getchar();
    while(!isdigit(k)&&k!='-')k=getchar();
    if(!isdigit(k)){f=-1;k=getchar();}
    while(isdigit(k)){s=s*10+(k^48);k=getchar();}
    x=s*f;
}
void file(void){
    #ifndef ONLINE_JUDGE
    freopen("P1333.in","r",stdin);
    freopen("P1333.out","w",stdout);
    #endif
}
const int MAXN=250050;
char l[15],r[15];
int e,f[MAXN];
bool in[MAXN];
gp_hash_table<string,int>G; //hash_table定义
int find(int x)
{
    int t=f[x],d;while(t!=f[t])t=f[t];
    while(f[x]!=t){d=f[x];f[x]=t;x=d;}
    return t;
}
void work()
{
    while(scanf("%s%s",l,r)!=EOF)
    {
        if(!G[l])G[l]=++e,f[e]=e;
        if(!G[r])G[r]=++e,f[e]=e;
        if(e>250010)//如果点数超过n+1的话，一定不存在通路，直接返回。
        {

```

```

        printf("Impossible\n");
        return;
    }
    in[G[l]] ^= 1; //修改奇偶情况
    in[G[r]] ^= 1;
    f[find(G[l])] = find(G[r]); //合并并查集
}
int flag = 0;
int i;
For(i, 1, e) if(in[i]) //判断奇点个数是否超过2
{
    if(flag == 2)
    {
        printf("Impossible\n");
        return;
    }
    else ++flag;
}
int father = find(1);
For(i, 2, e) if(find(i) != father) //判断是否连通
{
    printf("Impossible\n");
    return;
}
printf("Possible\n");
}
int main(void){
    file();
    work();
    return 0;
}

```

tree

pbds里面的tree都是平衡树，其中有rb_tree, splay_tree, avl_tree（后两种都容易超时，所以请不要用它们）。需要的头文件与命名空间也讲了，下面我们来看它的食用方法：

```

#define pii pair<int,int>
#define mp(x,y) make_pair(x,y)
tree<pii, null_type, less<pii>, rb_tree_tag, tree_order_statistics_node_update> tr;
pii //存储的类型
null_type //无映射(低版本g++为null_mapped_type)
less<pii> //从小到大排序
rb_tree_tag //红黑树
tree_order_statistics_node_update //更新方式
tr.insert(mp(x,y)); //插入;
tr.erase(mp(x,y)); //删除;
tr.order_of_key(pii(x,y)); //求排名
tr.find_by_order(x); //找k小值, 返回迭代器
tr.join(b); //将b并入tr, 前提是两棵树类型一样且没有重复元素
tr.split(v,b); //分裂, key小于等于v的元素属于tr, 其余的属于b
tr.lower_bound(x); //返回第一个大于等于x的元素的迭代器
tr.upper_bound(x); //返回第一个大于x的元素的迭代器
//以上所有操作的时间复杂度均为O(logn)

```

下面我们来试一试洛谷P3369 普通平衡树 (<https://www.luogu.org/problemnew/show/P3369>)（感谢shenben的代码）：

```
//by shenben
#include<cstdio>
#include<iostream>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
tree<ll,null_type,less<ll>,rb_tree_tag,tree_order_statistics_node_update> bbt;
int n;ll k,ans;
inline int read(){
    int x=0,f=1;char ch=getchar();
    while(ch<'0' || ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0' && ch<='9'){x=x*10+ch-'0';ch=getchar();}
    return x*f;
}
int main(){
    n=read();
    for(int i=1,opt;i<=n;i++){
        opt=read();k=read();
        if(opt==1) bbt.insert((k<<20)+1);
        if(opt==2) bbt.erase(bbt.lower_bound(k<<20));
        if(opt==3) printf("%d\n",bbt.order_of_key(k<<20)+1);
        if(opt==4) ans=*bbt.find_by_order(k-1),printf("%lld\n",ans>>20);
        if(opt==5) ans=*--bbt.lower_bound(k<<20),printf("%lld\n",ans>>20);
        if(opt==6) ans=*bbt.upper_bound((k<<20)+n),printf("%lld\n",ans>>20);
    }
    return 0;
}
```

前方高能！ 前方高能！ 前方高能！

在看这里之前，你需要熟练地掌握c++的特性。如果看不懂我也没有办法，你可以跳过这一部分。

你以为pbds种的tree只能实现这些功能？不不不，你可以自定义它，我们需要写一个自己的node_update，它是长这样的：

```
template<class Node_Citr,class Node_Itr,class Cmp_Fn,class _Alloc>
struct my_node_update
{
    typedef my_type metadata_type;
    void operator()(Node_Itr it, Node_Citr end_it)
    {
        ...
    }
};
```

我们先解释一下这个类是如何工作的。节点更新的tree都会保存一个my_type类型的变量。当我们修改这棵树的时候，会从叶子节点开始修改，并且每次都会调用operator()，我们来看一下这个函数的两个参数：

Node_Itr it为调用该函数的元素的迭代器，Node_Citr end_it可以const到叶子节点的迭代器，Node_Itr有以下的操作：

1.get_l_child(), 返回其左孩子的迭代器，没有则返回node_end；

- 2.get_r_child(), 同get_l_child();
 - 3.get_metadata(), 返回其在树中维护的数据;
 - 4.it可以获取it的信息。
- 为了详细讲解，我们举一个更新子树大小的例子：

```
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+1;
}
```

现在我们学会了更新，那么我们该如何自己写操作呢？node_update所有public方法都会在树中公开。如果我们在node_update中将它们声明为virtual，则可以访问基类中的所有virtual。所以，我们在类里添加以下内容：

```
virtual Node_CItr node_begin() const=0;
virtual Node_CItr node_end() const=0;
```

这样我们就能直接访问树了，还有，node_begin指向树根，node_end指向最后一个叶子节点的后一个地址，下面这个就是查排名的操作：


```
int myrank(int x)
{
    int ans=0;
    Node_CIt r=node_begin();
    while(r!=node_end())
    {
        Node_CIt l=r.get_l_child();
        Node_CIt r=r.get_r_child();
        if(Cmp_Fn()(x,**r))
            r=l;
        else
        {
            ans++;
            if(l!=node_end()) ans+=l.get_metadata();
            r=r;
        }
    }
    return ans;
}
```

下面我们来看CF459D (<https://www.luogu.org/problemnew/show/CF459D>) :

```

#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
template<class Node_CItr,class Node_Itr,class Cmp_Fn,class _Alloc>
struct my_node_update
{
    typedef int metadata_type;
    int order_of_key(pair<int,int> x)
    {
        int ans=0;
        Node_CItr it=node_begin();
        while(it!=node_end())
        {
            Node_CItr l=it.get_l_child();
            Node_CItr r=it.get_r_child();
            if(Cmp_Fn()(x,**it))
                it=l;
            else
            {
                ans++;
                if(l!=node_end()) ans+=l.get_metadata();
                it=r;
            }
        }
        return ans;
    }
    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+1;
    }
    virtual Node_CItr node_begin() const = 0;
    virtual Node_CItr node_end() const = 0;
};
tree<pair<int,int>,null_type,less<pair<int,int> >,rb_tree_tag,my_node_update> me;
int main()

```



```

{
    map<int,int> cnt[2];
    int n;
    cin>>n;
    vector<int> a(n);
    for(int i=0;i<n;i++)
        cin>>a[i];
    vector<int> pre(n),suf(n);
    for(int i=0;i<n;i++)
    {
        pre[i]=cnt[0][a[i]]++;
        suf[n-i-1]=cnt[1][a[n-i-1]]++;
    }
    long long ans=0;
    for(int i=1;i<n;i++)
    {
        me.insert({pre[i-1],i-1});
        ans+=i-me.order_of_key({suf[i],i});
    }
    cout<<ans<<endl;
}

```

trie

trie即为字典树，我们先看如何定义一个trie与它的操作：

```

typedef trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,trie_prefix_search_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。

```

现在我们来Astronomical Database (<http://acm.timus.ru/problem.aspx?space=1&num=1414>)：

```

#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/trie_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef trie<string,null_type,trie_string_access_traits<>,pat_trie_tag,trie_prefix_search_node_update>pref_trie;
int main()
{
    pref_trie base;
    base.insert("sun");
    string x;
    while(cin>>x)
    {
        if(x[0]!='?')
        {
            cout<<x.substr(1)<<endl;
            auto range=base.prefix_range(x.substr(1));
            int t=0;
            for(auto it=range.first;t<20 && it!=range.second;it++,t++)
                cout<<" "<<*it<<endl;
        }
        else
            base.insert(x.substr(1));
    }
}

```

priority_queue

priority_queue为优先队列，用堆实现，priority_queue的定义与操作：

```

priority_queue<int,greater<int>,TAG> Q;//小根堆，大根堆写less<int>
/*其中的TAG为类型，有以下几种：
pairing_heap_tag
thin_heap_tag
binomial_heap_tag
rc_binomial_heap_tag
binary_heap_tag
其中pairing_help_tag最快*/
Q.push(x);
Q.pop();
Q.top();
Q.join(b);
Q.empty();
Q.size();
Q.modify(it,6);
Q.erase(it);
//以上操作我都不讲了，pbds里的优先队列还可以用迭代器遍历

```

时间复杂度：

- * pairing_heap_tag: push和join为 $O(1)$ ，其余为均摊 $\Theta(\log n)$
- * binary_heap_tag: 只支持push和pop，均为均摊 $\Theta(\log n)$
- * binomial_heap_tag: push为均摊 $O(1)$ ，其余为 $\Theta(\log n)$
- * rc_binomial_heap_tag: push为 $O(1)$ ，其余为 $\Theta(\log n)$
- * thin_heap_tag: push为 $O(1)$ ，不支持join，其余为 $\Theta(\log n)$ ；但是如果只有increase_key，那么modify为均摊 $O(1)$



*洛谷“不支持”不是不能用，而是用起来很慢

堆优化dijkstra (<https://www.luogu.org/problemnew/show/P4779>)
(感谢Great_Influence的代码)：

```

#include<bits/stdc++.h>
#include<ext/pb_ds/priority_queue.hpp>
#define Rep(i,a,b) for(register int i=(a),i##end=(b);i<=i##end;++i)
#define Repe(i,a,b) for(register int i=(a),i##end=(b);i>=i##end;--i)
#define For(i,a,b) for(i=(a),i<=(b);++i)
#define Forward(i,a,b) for(i=(a),i>=(b);--i)
#define Chkmax(a,b) a=a>b?a:b
template<typename T>inline void read(T &x)
{
    T f=1;x=0;char c;
    for(c=getchar();!isdigit(c);c=getchar())if(c=='-')f=-1;
    for(;isdigit(c);c=getchar())x=x*10+(c^48);
    x*=f;
}

inline void write(int x)
{
    if(!x){putchar(48);putchar('\n');return;}
    static int sta[45],tp;
    for(tp=0;x;x/=10)sta[++tp]=x%10;
    for(;tp;putchar(sta[tp--]^48));
    putchar('\n');
}

using namespace std;
void file()
{
    #ifndef ONLINE_JUDGE
        freopen("water.in","r",stdin);
        freopen("water.out","w",stdout);
    #endif
}

const int MAXN=1e5+7,MAXM=4e5+7;

static int n,m;

static struct edg
{
    int u,v,w,h;
    friend bool operator<(edg a,edg b){return a.h>b.h;}
}EDG[MAXM];

static struct edge

```

```

{
    int v,w,nxt;
}P[MAXM<<1];

static int head[MAXN],e;

inline void add(int u,int v,int w)
{P[++e]=(edge){v,w,head[u]};head[u]=e;}

__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));
            }
    }
}

static int s;

inline void init()
{
    read(n);read(m);read(s);
    static int u,v,w;
    Rep(i,1,m)read(u),read(v),read(w),add(u,v,w);
}

```

```
}

inline void solve()
{
    dijkst(s);
    Rep(i,1,n)printf("%d ",dis[i]);
    puts("");
}

int main()
{
    file();
    init();
    solve();
    return 0;
}
```