C++常用库在 OI 中的运用

福州一中 陈俊锟

0 Abstract（导读）

STL（标准模板库）和 pb_ds（基于政策的数据结构）库并称 C++两大常用库。这些库提供了一些实现良好的函数和数据结构，能极大的节省我们在 OI 比赛中的编码和运行时间。如果我们能够了解一些库的知识，便可在比赛中事半功倍。


回复：您好，请问NOIP时C++是否可以使用ext/pb_ds目录下的库函数？

发件人：noieval<noieval@ccf.org.cn>

收件人：郭耀星<codewayskygo@126.com>

时 间：2012年10月12日 21:59 (星期五)

可以使用。

---------- 原始邮件 ----------

发件人："郭耀星"<codewayskygo@126.com>;
发送时间：2012年10月12日(星期五) 晚上7:35
收件人："noieval"<noieval@ccf.org.cn>;
主题：Re:回复：您好，请问NOIP时C++是否可以使用ext/pb_ds目录下的库函数？

正确执行没有问题，我主要不清楚的问题是 using namespace __gnu_pbds; 会不会导致不能通过代码审核？

目前，NOI 系列比赛已经对 STL 和 pb_ds 库解禁。

本文将介绍 STL 部分实用函数及数据结构，以及 pb_ds 中的数据结构，并给出部分参考程序。

1 STL

1.1 常用算法函数

以下函数包含在头文件 algorithm 中，且需要 using namespace std。我们比较常用的是 sort 函数。

1.1.1 排序

```
std::sort(begin, end, cmp = std::less<T>());
```

C++库中提供的快排，内部实现的是快速排序和基数排序的组合。begin 表示需要排序的第一个元素，end 表示需要排序的最后一个元素的下一个。cmp 参数为比较函数，默认为 less<T>，即小于号。

如果我们要对整数数组 a 进行从小到大排序，下标为 1～n，则使用：

```
std::sort(a + 1, a + 1 + n);
```

如果我们在运行 kruskal 算法，需要把所有边按照边权从大到小排序，则我们需要定义一个比较函数。我们既可以在 struct Edge 中定义小于号，也可以定义比较函数 smaller。

```
//使用自定义小于号
struct Edge
{
    int a, b, l;

    //默认构造函数
    Edge(aa = 0, bb = 0, ll = 0) : a(aa), b(bb), l(ll) {}

    //小于号的定义
    bool operator<(const Edge& e)const
    {
        return l < e.l;
    }
}e[1000010];
//调用方式：std::sort(e + 1, e + n + 1);

//使用自定义比较函数
bool smaller(const Edge& a, const Edge& b){return a.l < b.l;}
//调用方式：std::sort(e + 1,  e + n + 1, smaller);
```

第二种方式常常更加灵活，因为我们可能需要对内置类型进行其他规则的排序。有时我们只需要获得排过序的编码，不能修改原来元素的位置。这种情况下，我们这么定义：

```
//令 b 数组为 a 数组的从小到大的编号
bool smaller(int x, int y){return a[x] < a[y];}
//先令 b[i] = i，接着调用 std::sort(b + 1, b + n + 1, smaller)。
```

OJ1837：凸包，利用 sort 获得排序编码，节省大量拷贝结构体的时间

```
//FZYZ OJ P1837
#include<cstdio>
#include<cstring>
#include<algorithm>
const int MAXN=10010;
struct Point{
    int x,y;
}P[MAXN];
int N,St[MAXN],top,topp,Ans=0;
bool cmp(Point A,Point B)
{
    if(A.x==B.x) return A.y<B.y;
    return A.x<B.x;
}
int Cross(Point A,Point B,Point C)
{
    return (B.x-A.x)*(C.y-B.y)-(B.y-A.y)*(C.x-B.x);
}
int Cro(Point A,Point B,Point C)
{
    return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
}
int main()
{
    scanf("%d",&N);
    for(int i=1;i<=N;i++)
        scanf("%d%d",&P[i].x,&P[i].y);
    std::sort(P+1,P+N+1,cmp);
    top=1;
    St[1]=1;
    for(int i=2;i<=N;i++)
    {
        while(top>1 && Cross(P[St[top-1]],P[St[top]],P[i])<=0) top--;
        St[++top]=i;
```

```
    }
    topp=top;
    for(int i=N-1;i>=1;i--)
    {
        while(topp>top && Cross(P[St[topp-1]],P[St[topp]],P[i])<=0) topp--;
        St[++topp]=i;
    }
    for(int i=2;i<topp;i++)
        Ans+=Cro(P[St[1]],P[St[i]],P[St[i+1]]);
    printf("%d\n",Ans/100);
    return 0;
}
```

STL 还提供了 qsort 和 stable_sort 函数。前者适用于 C，需要 cmp 函数完成对 void*指针的转换，使用不多；后者牺牲了一些时间，保留了相等元素之间的次序关系，可用于基数排序。

1.1.2 元素查找

C++标准库提供了多种和元素查找有关的函数。这里只写出适用于有序数组的函数。这些函数的时间复杂度均为 O(logn)。

binary_search(l, r, v)	若在[l, r)区间内存在 v，则返回 true，否则 false
lower_bound(l, r, v)	若在[l, r)区间内存在 v，则返回出现的第一个位置的指针，否则返回 r
upper_bound(l, r, v)	若在[l, r)区间内存在 v，则返回出现的最后一个位置的指针，否则返回 r
equal_range(l, r, v)	返回一个 pair，表示等于 v 的区间

我们可以利用这些函数更快的完成离散化等操作。

OJ1718：扫描线，离散化横坐标，利用 lower_bound。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <set>
#include <map>
#define For(a,b,c) for(int a=b,d=c;a<=d;++a)
#define ForDown(a,b,c) for(int a=c;a>=b;--a)
#define checkmin(a,b) ((a>b)?(a=b):a)
#define checkmax(a,b) ((a<b)?(a=b):a)
using namespace std;
int max(int a,int b){return a>b?a:b;}
template<class T>void read(T&aa)
{
     int bb;
          char ch;
          while(ch=getchar(),(ch<'0'||ch>'9')&&ch!='-');
     ch=='-'?(bb=1,aa=0):(aa=ch-'0',bb=0);
     while(ch=getchar(),ch>='0'&&ch<='9')aa=aa*10+ch-'0';if(bb)aa=-aa;
}
int nextInt()
{
     int aa;
     int bb;
     char ch;
          while(ch=getchar(),(ch<'0'||ch>'9')&&ch!='-');
     ch=='-'?(bb=1,aa=0):(aa=ch-'0',bb=0);
     while(ch=getchar(),ch>='0'&&ch<='9')aa=aa*10+ch-'0';if(bb)aa=-aa;
     return aa;
```

```
}
bool covered[500010];
long long sum[500010];
long long Hash[1000040],top,htop,n;
struct Segment
{
    long long l,r,h;
    Segment(){}
    bool operator<(const Segment &a)const{return h<a.h;}
}s[500010];

#define LS at<<1
#define RS at<<1|1
void pushup(int at,int l,int r)
{

(covered[at])?sum[at]=Hash[r+1]-Hash[l]:(l==r)?sum[at]=covered[at]:sum[at]=sum[LS]+sum[RS];
}

void cover(int L,int R,int at=1,int l=1,int r=htop)
{
    if(L<=l&&r<=R)
    {
        covered[at]=1;
        pushup(at,l,r);
        return;
    }
    int mid=(l+r)>>1;
    if(L<=mid)cover(L,R,LS,l,mid);
    if(R>mid)cover(L,R,RS,mid+1,r);
    pushup(at,l,r);
}
int binarySearch(long long key)
{
    //For(i,1,htop)if(Hash[i]==key)return i;return -1;
    return std::lower_bound(Hash+1,Hash+htop+1,key) - Hash;
}
int main()
{
     read(n);For(i,1,n)read(s[i].l),read(s[i].r),read(s[i].h),Hash[++top]=s[i].l,Hash[++top]
=s[i].r;
    s[0].h=0;sort(Hash+1,Hash+top+1);sort(s+1,s+n+1);
    Hash[htop=1]=Hash[1];For(i,2,top)(Hash[i]==Hash[i-1])?1:Hash[++htop]=Hash[i];
//cout<<htop<<endl;
    long long ans=0;

ForDown(i,1,n)cover(binarySearch(s[i].l),binarySearch(s[i].r)-1),ans+=sum[1]*(s[i].h-s[i-1].
h);
    printf("%lld\n",ans);
}
```

1.1.4 数据结构相关函数

STL 提供了一些函数，使得我们可以利用它们把一个数组当做一个堆使用。使用 STL 函数的堆中，i 节点的左右儿子分别为 2i 和 2i+1。

make_heap(l, r)	制作一个新的堆。将排序。
pop_heap(l, r)	将堆头移动到 r
push_heap(l, r, v)	在堆中插入元素
sort_heap(l, r)	将对堆进行重新排序。

这些函数的效率并不高。

1.1.5 其他函数

for_each(l, r, f)：对于 l 到 r 的每个元素*i 执行 f(*i)。

generate(l, r, g)：对 l 到 r 的每个函数*i，执行*i = g()。

next_permutation(l, r)：把该数组改为下一个排列

nth_element(l, r, n)：对[l, r)进行重排，保证 a[n]恰好是第 n 大，且小于该数的数均在它前面

其他函数由于效率不太高，不出现在这里。

1.2 序列容器

1.2.1 vector<T>

vector<T>是一个可变大小的数组。它的内部实现就是数组，可以实现 O(1)的随机访问和修改（用法同数组，可用 a[i]）。如果当前的数组最大支持空间过小，它会自动把容量翻倍。其速度接近于数组。

用法：

```
vector<int> a;
vector<int> b(大小);
a.push_back(1);
b[1] = 1;
```

一个值得注意的陷阱是，对于上面创建的 a，如果执行 a[10]，将会导致运行时错误！目前 a[10]并不存在，所以对它的任何使用包括赋值都是不合法的。一般我们使用 push_back 函数在末尾加上一个元素。

vector 所有类似数组的功能（包括 push_back 和 size）都是 O(1)，但在中部插入删除元素都是 O(n)的。

在竞赛中，我们可以用 vector 来实现邻接表。在刘汝佳的所有书中，都是用 vector 实现邻接表的：

```
vector<int> edges[MAXN];
void addEdge(int a, int b){edges[a].push_back(b);}
```

1.2.2 list<T>

List<T>实现的是一个链表。它支持 O(1)的在任意位置插入数据和删除数据，以及合并分离，但一切随机访问、遍历、甚至 size()都是 O(n)的。

在 list 的遍历中，我们一般使用 list<T>::iterator 来代替遍历指针。我们使用这样的循环：

```
for(list<int>::iterator i = a.begin(); i != a.end(); ++i)
    cout<<*i<<endl;
```

来输出这个 list 中的每一个元素。

迭代器还可以作为下标使用。比如，我们开启 list<int>::iterator pos[10]，然后每次 pos[i] = list.push_back(i)，这样我们就得到了每个数的对应节点的“指针”。无论其中如何插入删除元素，这些指针的意义和指向的元素都不会改变，也就是说，随时可以修改、删除这些点。这是一个很好的思想，在后面讲堆优化 dijkstra 修改距离时也会提到。

注意，迭代器只是一个指针，方便你访问当前节点，而不能访问“距离头为

i 的节点”。

OJ1604：链表，使用 list 实现，使用迭代器数组。（速度较慢，0.499s）

```
#include <iostream>
#include <list>
using namespace std;

list<int> students;
list<int>::iterator points[100010], insert, iter;

int n, m, i, k;
bool p;

int main()
{
    cin >> n;
    students.push_back(1);
    points[1] = students.begin();

    //prevent "insert(students.end(), t)"
    students.push_front(-1);
    students.push_back(-1);

    for(i = 2; i <= n; ++i)
    {
        cin >> k >> p;
        insert = points[k];
        if(p)++insert;
        points[i] = students.insert(insert, i);
    }

    cin >> m;
    while(m--)
    {
        cin >> k;
        if(points[k] != students.end())
        {
            students.erase(points[k]);
            points[k] = students.end();
        }
    }

    for(iter = students.begin(); iter != students.end(); ++iter)
        if(*iter != -1)cout << *iter << ' ';
}
```

OJ1604 附手写链表实现（0.082s）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <set>
#include <map>
#define For(a,b,c) for(int a=b,d=c;a<=d;++a)
#define ForDown(a,b,c) for(int a=c;a>=b;--a)
#define checkmin(a,b) ((a>b)?(a=b):a)
#define checkmax(a,b) ((a<b)?(a=b):a)
using namespace std;
int max(int a,int b){return a>b?a:b;}
char B[1<<15],*S=B,*T=B;char getc(){
    return S==T&&(T=(S=B)+fread(B,1,1<<15,stdin),S==T)?0:*S++;
}
template<class T>T read(T&aa)
{
    int bb;
        char ch;
        while(ch=getc(),(ch<'0'||ch>'9')&&ch!='-');
    ch=='-'?(bb=1,aa=0):(aa=ch-'0',bb=0);
    while(ch=getc(),ch>='0'&&ch<='9')aa=aa*10+ch-'0';if(bb)aa=-aa;
    return aa;
}
int nextInt()
```

```
{
    int t;
    return read(t);
}
int L[100023],R[100023],head=1;
bool del[100023];
int t;
int n,m;
int main()
{
    read(n);
    For(i,2,n)
    {
        int x,y;
        read(x),read(y);
        if(y)
        {
            R[i]=R[x];
            L[i]=x;
            (R[x])?L[R[x]]=i:0;
            R[x]=i;
        }
        else
        {
            (x==head)?head=i:0;
            L[i]=L[x];
            R[i]=x;
            (L[x])?R[L[x]]=i:0;
            L[x]=i;
        }
//cout<<"AT "<<n<<' '<<i<<' '<<L[i]<<' '<<R[i]<<endl;
    }
    read(m);
    For(i,1,m)del[t=nextInt()]=1;
    for(int i=head;i;i=R[i])(del[i])?1:printf("%d ",i);
    puts(" ");
}
```

List 虽然可以减少代码量，但常数较大，在信息学竞赛中，应谨慎使用。

1.2.3 其他序列容器

queue<T>：实现一个 FIFO 的队列，内部使用类似 list，不可遍历。慢。

deque<T>：可在双端添加的队列，内部使用类似 list。慢。

slist<T>：单向链表，其迭代器只支持++，不是 STL 的一部分。比较快。

stack<T>：实现一个 FILO 的栈。慢。

priority_queue<T>：使用堆函数封装的一个 vector 实现的堆。很慢。

string 或 vector<char>：C 风格字符串的替代品。慢。

bitset 或 vector<bool>：使用二进制位存放的布尔值。慢。

1.3 关联容器

1.3.1 set 及 multiset

set 是一个关联容器“集合”，保存着一个值“是否存在”的属性。内部实现是红黑树。set 和 multiset 的区别在于可否允许重复元素。

两种 set 都提供 insert 函数和 count 函数，前者为在集合中插入，后者为输出集合中该元素的个数（显然，set<T>::count 的返回值只会有 0 和 1 两种）。

两个函数的时间复杂度均为 O(logn)。注意：set 虽然可以代替 std::priority_queue，但之后介绍了 pb_ds 中的优先队列（堆）后会有一个更好的解决方案。

OJ2038：优先级优先拓扑搜索。使用 set 模拟堆。

```
//FUS File Prefix 20150107 +8
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <queue>
#include <set>
#include <map>
#include <cstdarg>
#define For(a,b,c) for(int a=b,dd=c;a<=dd;++a)
#define ForDown(a,b,c) for(int a=c;a>=b;--a)
#define CheckMin(a,b) (((a)>(b))?((a)=(b)):(a))
#define CheckMax(a,b) (((a)<(b))?((a)=(b)):(a))
#define IF (
#define DO_NOTHING (0)
#define THEN )?(
#define ELSE 1):(
#define DONE 1),1
#define ELSE_DONE ELSE DO_NOTHING, DONE
#define THEN_ELSE THEN DO_NOTHING, ELSE
#define O2_OPT
#define FUSDEBUG0
using namespace std;
//input
int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,__ReadS[1<<22],*__RSI=__ReadS,
__PrintCache[1<<22],*__PrintPt=__PrintCache,__IPS[100];va_list P;
inline void ScanLine(int Num, ...)
{
     for(va_start(P, Num),gets(__ReadS); Num;)
     {for(__RSI = __ReadS; *__RSI && (Num); --Num){int &aa = *va_arg(P, int*);
     while(__ReadCh=*__RSI++,(__ReadCh<'0'||__ReadCh>'9')&&__ReadCh!='-')DO_NOTHING;
     __ReadCh=='-'?(__ReadBool=1,aa=0):(aa=__ReadCh-'0',__ReadBool=0);

while(__ReadCh=*__RSI++,__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';(__ReadBool)?aa=-
aa:1;}
     (Num) ? gets(__ReadS), DO_NOTHING : DO_NOTHING;}
}
inline void Read(int &aa)
{
    while(__ReadCh=getchar(),(__ReadCh<'0'||__ReadCh>'9')&&__ReadCh!='-')DO_NOTHING;
    __ReadCh=='-'?(__ReadBool=1,aa=0):(aa=__ReadCh-'0',__ReadBool=0);

while(__ReadCh=getchar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';(__ReadBool)?aa=
-aa:1;
}
inline int NextInt(){Read(__ReadT);return __ReadT;}
//output
#define PutChar(c)  (*__PrintPt++=c)
#define Output()
(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache,1)
inline void Print(int x)
{
    if(!x)*__PrintPt++=48;else{(x<0)?x=-x,PutChar('-'):1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;PutChar(48+__IPS[__Prin
tIter--]));}
}
inline void Print(const char s[]){for(;*s;++s)PutChar(*s);}
#define ToString(x) #x
#define Fill(A,c) memset(A, c, sizeof(A))
//end

#define Next(at) _Next[at]
#define First(at) _First[at]
#define To(at) _To[at]
```

```
#define Size(at) _Size[at]
#define PATHSIZE 400010
#define MAXN 100010
int _To[PATHSIZE],_Size[PATHSIZE],_Next[PATHSIZE],_First[MAXN],_Top,_Tmp,_Ta,_Tb;
#define _AddNxt(a) (++_Top, Next(_Top) = _First[a],_First[a] = _Top)
#define AddPath(_a,_b,l) (_Ta=_a,_Tb=_b,_Tmp=_AddNxt(_Ta),To(_Tmp)=_Tb,Size(_Tmp)=l)

int N, M;
int InDegree[100010];
int Ans[100010];
set <int> Queue;
int main ()
{
    ScanLine(2, &N, &M);

    For (i, 1, M)
    {
        int A;
        int B;
        ScanLine(2, &A, &B);
//cout<<A<<' '<<B<<endl;
        AddPath (A, B, 1);
        ++ InDegree [B];
    }

    For (i, 1, N) IF InDegree [i] THEN_ELSE Queue. insert (i), DONE;

    for (int i = 1; (i <= N) && (!Queue. empty()); ++i)
    {
        int At     = *(Queue. begin ());
//cout<<"DONE "<<At<<endl;
        Ans [i] = At;
        Queue. erase (Queue. begin ());
        for (int i = First (At); i; i = Next (i))
            IF !(-- InDegree [To(i)]) THEN Queue. insert (To (i)), ELSE_DONE;
    }

    if (! Ans [N]) puts ("OMG.");
    else For (i, 1, N) Print (Ans[i]), PutChar(' ');
   PutChar('\n');
    Output();
}
```

这题是为数不多用 set 代替堆而不会超时的题目。set 有 hash 版本，但近期被废弃了。在 C++ 11 中，hashset 和 hashmap 被重命名为 unordered 系列，但目前竞赛中并没有提供 C++ 11 环境，所以这里并不介绍 hash 版容器。

1.3.2 map

map<V, E, C = std::less<V> >是一个常用的关联容器。map 也有 multi 版本，但很少使用。map 相当于一个任意下标的数组，它内部使用红黑树实现，要求类型 V 必须定义小于号，否则则必须提供比较类。一般我们将它作为名称和编号的对应，比如 num["John"] = 1。map 的效率均在 log 级别。

OJ1759：字符串为点名称的最短路，用 map 管理编号。（后面有用 hash 管理标号的程序）

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cmath>
#include<string>
#include<cstring>
#include<map>
using namespace std;
int n,dist[200002],next[200002],pre[200002],queue[200002],value[200002];
bool vis[200002]={0};
int F()
{ char ch=getchar(); int data=0; while (ch<'0'||ch>'9')
```

```
   ch = getchar(); do {
    data=data*10 + ch-'0';
    ch=getchar();
    } while (ch>='0' & ch <= '9');
    return data;
}
map<string,int>name;
int main()
{
    int i,j,c,tail=1,head=0,t,p,w[200002],m;
    string  q,s;
    n=F(),m=F();
    for(i=1;i<=n;i++)
    cin>>q,name[q]=i;
    memset(dist,63,sizeof(dist));
    for(i=1;i<=m;i++)
    {        string x,y;
             // int z;
             cin>>x>>y,value[i*2-1]=F();
                              //value[i*2-1]=z;
               int t=name[x];
int d=name[y];w[i*2-1]=d;

             next[i*2-1]=pre[t];
             pre[t]=i*2-1;
               w[i*2]=t;
               value[i*2]=value[i*2-1];
               next[i*2]=pre[d];
               pre[d]=i*2;
             /*std::cout<<next[i]<<" "<<w[i]<<" "<<pre[p];
             system("pause");*/
    }
    dist[1]=0;
    vis[1]=1;
    queue[0]=1;
    while(head<tail)
    {        int x,y;
               x=queue[head];
               y=pre[x];
               while(y)
               {
                 if (dist[x]<dist[w[y]]-value[y])
                 {
                     dist[w[y]]=dist[x]+value[y];
                     if (!vis[w[y]])
                     {
                     queue[tail++]=w[y];
                     vis[w[y]]=1;
                     }
                 }
                 y=next[y];
               }
               vis[x]=0;
              head++;
    }
    printf("%d",dist[2]);
    return 0;
}
```

本题如果使用后面提到的 pb_ds 中的 hash 系列，速度会快上 3 倍。

1.4 __gnu_cxx::rope<T>（头文件：ext/rope）

rope 是一种非常高级和神奇的数据结构，值得我用一整个大目录去介绍。

rope 的设计初衷是支持超大的字符串，因此它在红黑树中实现和很多接口和功能，使它变得非常快速和强大。它支持 log 复杂度的：随机访问、删除、插入、插入串、删除串、切割串、拷贝、替换，几乎支持了所有红黑树可以做的操作。其使用类似数组，用[]进行随机访问。

IOI2012 scrivener：要求维护一个可持久化序列，支持 log 及以下级别的添加、

撤消和随机访问（撤消可以撤消撤消操作），强制在线。使用 rope 可以轻松实现可持久化红黑树。

```
#include<cstdio>
#include<cstring>
#include<cctype>
#include<iostream>
#include<algorithm>
#include<ext/rope>
using namespace std;
using namespace __gnu_cxx;
const int maxn=1e5+10;
rope<char> *his[maxn];
int n;
int d[maxn];
inline int lowbit(int x){
     return x&-x;
}
inline void updata(int x){
     while(x<=n){
          d[x]++;
          x+=lowbit(x);
     }
}
inline int get(int x){
     int res=0;
     while(x){
          res+=d[x];
          x-=lowbit(x);
     }return res;
}
inline char getC(){
     char ch=getchar();
     while(!isalpha(ch))ch=getchar();
     return ch;
}
inline int getint(){
     int res=0;
     char ch,ok=0;
     while(ch=getchar()){
          if(isdigit(ch)){
               res*=10;res+=ch-'0';ok=1;
          }else if(ok)break;
     }return res;
}
void deb(rope<char> s){
     for(int i=0;i<s.length();i++)
     cout<<s[i];puts("");
}
int main(){
     freopen("type.in","r",stdin);
     freopen("type.out","w",stdout);
     n=getint();
     his[0]=new rope<char>();
     for(int i=1;i<=n;i++){
          //可持久化就在这里：O(1)的时间拷贝一个根作为副本
          his[i]=new rope<char>(*his[i-1]);
//        deb(*his[i]);
          char opt=getC();
          if(opt=='T'){
               char x=getC();
               his[i]->push_back(x);
               updata(i);
          }else
          if(opt=='U'){
               updata(i);
               int x=getint();
               int l=1,r=i,mid,now=get(i);
               while(l<r){
                    mid=(l+r)>>1;
                    if(now-get(mid)>x)
                         l=mid+1;
                    else
                         r=mid;
               }
               his[i]=his[l-1];
```

```
        }else
        if(opt=='Q'){
            int x=getint()-1;
            putchar(his[i]->at(x));
            putchar('\n');
        }
//      deb(*his[i]);
    }
    return 0;
}
```

NOI2003 editor：维护字符串，要求可以插入串、删除串、翻转区间、输出区间。使用 rope 维护一正一反两个串，每次翻转的时候交换相应区间。

```
/**************************************************************
    Problem: 1507
    User: immortalCO
    Language: C++
    Result: Accepted
    Time:712 ms
    Memory:42184 kb
****************************************************************/

#include <bits/stdc++.h>
#define For(i, a, b) for(int i = a; i <= b; ++i)
#define ForDown(i, a, b) for(int i = b; i >= a; --i)

namespace tools
{
    inline bool checkMax(int &a, int b)
    {
        return (a < b ? a = b, 1 : 0);
    }

    inline bool checkMin(int &a, int b)
    {
        return (a > b ? a = b, 1 : 0);
    }

    inline int min(int a, int b){return (a < b ? a : b);}
    inline int max(int a, int b){return (a > b ? a : b);}
    inline int abs(int a, int b){return a > 0 ? a : -a;}

    template <class T> class Object
    {
    private:
        T* instance;
    public:
        operator T () {return *instance;}
        Object(T *a = NULL) : instance(a) {}
        Object(const T a)
        {
            instance = new int;
            *instance = a;
        }
        Object(const Object<T>& o) {instance = o.instance;}
        Object<T>& operator = (const Object<T>& o) {instance = o.instance; return *this;}
        Object<T>& operator = (T *a) {instance = a; return *this;}
        Object<T>& operator = (const T a) {instance = new int; *instance = a; return *this;}

        ~Object() {delete instance;}
    };

}

namespace io
{
    //input
    int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,
    __PrintCache[1<<25],*__PrintPt=__PrintCache,__IPS[100];
    inline int getChar(){static char buf[1<<15],*S=buf,*T=buf;return
(S==T)&&(T=(S=buf)+fread(buf,1,1<<15,stdin),S==T)?0:*S++;}
    //int (*getChar)() = getchar;
    template<class T>inline void read(T &aa)
    {
        while(__ReadCh=getChar(),(__ReadCh<'0'||__ReadCh>'9'));
```

```
        (aa=__ReadCh-'0',__ReadBool=0);
        while(__ReadCh=getChar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';
    }
    inline int nextInt(){read(__ReadT);return __ReadT;}
    inline int gets(char *t)
    {
        static char *s;
        s=t;while(__ReadCh=getChar(),(__ReadCh==' '||__ReadCh=='\n'));
        for(;(__ReadCh!=' '&&__ReadCh!='\n');__ReadCh=getChar())*s++=__ReadCh;
        *s=0;return (s-t);
    }
    //output
    //int (*putChar)(int) = putchar;
    inline void putChar(char c)  {(*__PrintPt++=c);}
    inline void reverse() {--__PrintPt;}
    //void pushDown() srd((tim("\55\117\62")));
    inline void output()
    {(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache);}
    template<class T>inline void print(T x)
    {
        if(!x)*__PrintPt++=48;else{(x<0)?x=-x,putChar('-'),1:1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;putChar(48+__IPS[__Prin
tIter--]));}
    }
    template<class T>inline void print(T x, char c){print(x);putChar(c);}
    inline void puts(const char s[]){for(;*s;++s)putChar(*s);}
    //end
}

#include <ext/rope>

namespace solve
{

    using namespace __gnu_cxx;
    crope string, reverse, tmp;
    int m, at = 0;

    inline void move(){io::read(at);}
    inline void prev(){--at;}
    inline void next(){++at;}
    inline void get()
    {
        static int size;
        io::read(size);
        tmp = string.substr(at, size);
        for(crope::const_iterator iter = tmp.begin(); iter != tmp.end(); ++iter)
            io::putChar(*iter);
        io::putChar('\n');
    }
    inline void insert()
    {
        static int size, length;
        static char ins[2000010], rev[2000010];
        io::read(size);
        length = string.length();
        For(i, 0, size - 1)
        {
            do
                ins[i] = io::getChar();
            while(ins[i] == '\n');
            rev[size - i - 1] = ins[i];
        }
        rev[size] = ins[size] = 0;
        string.insert(at, ins);
        reverse.insert(length - at, rev);
    }
    inline void del()
    {
        static int size, length;
        io::read(size);
        length = string.length();
        string.erase(at, size);
        reverse.erase(length - at - size, size);
    }
    inline void rev()
```

```
    {
        static int size, length;
        io::read(size);
        length = string.length();
        tmp = string.substr(at, size);
        string.replace(at, size, reverse.substr(length - at - size, size));
        reverse.replace(length - at - size, size, tmp);
    }

    void (*work[256])();

    void begin()
    {
        //puts("1");
        io::read(m);
        work['M'] = move;
        work['N'] = next;
        work['P'] = prev;
        work['G'] = get;
        work['I'] = insert;
        work['D'] = del;
        work['R'] = rev;
    }

    void solve()
    {
        while(m--)
        {
            static char command[100];
            io::gets(command);
            //puts(command);
            work[command[0]]();
        }
    }

    void end()
    {
        io::output();
    }
}



int main()
{
    solve::begin();
    solve::solve();
    solve::end();
}
```

BZOJ3674：可持久化并查集，用 rope 实现

```
#include<bits/stdc++.h>
#include<ext/rope>
using namespace std;
using namespace __gnu_cxx;
const int maxn=2e5+10;
rope<int> *fa[maxn];
rope<int>::iterator it;
int n,m;
int find(int i,int x){
    if(fa[i]->at(x)!=x){
        int f=find(i,fa[i]->at(x));
        if(f==fa[i]->at(x))return f;
        fa[i]->replace(x,f);
        return fa[i]->at(x);
    }
    return x;
}
void Union(int i,int x,int y){
    int fx=find(i,x),fy=find(i,y);
    fa[i]->replace(fy,fx);
}
int a[maxn];
int lastans=0;
int main(){
    scanf("%d%d",&n,&m);
    for(int i=0;i<=n;i++)a[i]=i;
```

```
    fa[0]=new rope<int>(a,a+n+1);
    for(int i=1;i<=m;i++){
        fa[i]=new rope<int>(*fa[i-1]);
        int op;scanf("%d",&op);
        if(op==1){
            int a,b;scanf("%d%d",&a,&b);
            Union(i,a^lastans,b^lastans);
        }else
        if(op==2){
            int k;scanf("%d",&k);
            fa[i]=fa[k^lastans];
        }else
        if(op==3){
            int a,b;scanf("%d%d",&a,&b);
            printf("%d\n",(lastans=find(i,a^lastans)==find(i,b^lastans)));
        }
    }
    return 0;
}
```

一个很大的好消息是，NOI 比赛中允许使用 rope。我们应当本着“不用重新发明轮子”的原则适当选用这些容器，给自己留出更多思考丧题的时间。

2 pb_ds

2.1 引子

pb_ds 的全称是“policy-based data structures”，是 gnu 发布的一套数据结构库，支持堆（优先队列）、树形（set 和 map）、trie 树、hash 表和链式数据结构，不仅允许选择实现方式（如支持配对堆、斐波那契堆等），而且所有运行速度都大大超过 stl，且添加了大量的实用功能（如树型支持查找第 K 大，堆支持合并），速度直逼手写版，是我们竞赛中极为有用的好帮手。

数据结构	需要的#include
priority_queue	ext/pb_ds/priority_queue.hpp
tree	ext/pb_ds/assoc_container.hpp; tree_policy.hpp
trie	ext/pb_ds/assoc_container.hpp; trie_policy.hpp
hash_tables	ext/pb_ds/assoc_container.hpp; hash_policy.hpp

pb_ds 还有一个 list_update 类，在竞赛中不经常使用，可以用 list 代替。

2.2 priority_queue

2.2.1 原型

```
template
<
    typename Value_Type,                          //【1】
    typename Cmp_Fn = std::less<Value_Type>,      //【2】
    typename Tag = pairing_heap_tag,              //【3】
    typename Allocator = std::allocator<char>        //分配器，不用管
```

它
>
class priority_queue;
【1】在优先队列中储存的元素类型
【2】比较类，默认为小于号
【3】实现方式，默认为配对堆（参见 2.2.3 节）

2.2.2 函数及迭代器

top()：获取堆的头
pop()：删除堆的头
push(x)：把 x 插入堆中
erase(x)：删除 x（x 可以为迭代器）
modify(iter, x)：把迭代器 iter 对应的键改为 x，并移动到合法的未知
join(&other)：把 other 合并到自己，并清空 other
point_iterator：迭代器，是 push 的返回值

2.2.3 实现方式及效率

模板参数的 Tag 允许我们选择实现方式。
pb_ds 中的堆提供了 5 种实现方式：

实现名称	中文名称	备注
pairing_heap_tag	配对堆（默认）	一般最快
binary_heap_tag	二叉堆	可能需要 swap 函数
binomial_heap_tag	二项堆	
rc_binomial_heap_tag	重计数二项堆	
thin_heap_tag	斐波那契堆	常数很大

其中，如果我们要自定义实现方式，就必须在模板参数中制定比较方式，并选择合适的迭代器。

时间复杂度：

名称	push	pop	modify	erase	join
std	最坏Θ(n) 均摊Θ(logn)	最坏 Θ(logn)	最坏Θ(nlogn)	Θ(nlogn)	
pairing	O(1)	最坏Θ(n) 均摊Θ(logn)			O(1)
binary	最坏Θ(n) 均摊Θ(logn)		Θ(n)		
binomial	最坏Θ(n)	Θ(logn)			

	均摊Θ(logn)				
rc	O(1)	Θ(logn)			
thin	O(1)	最坏Θ(n)均摊Θ(logn)	降低键值均摊O(1)，最坏Θ(logn)。其他均摊Θ(logn)。	最坏Θ(n)均摊Θ(logn)	Θ(n)

经过实际测试，thin 的效果并没有复杂度预计的那么好。在没有修改操作的时候，binary 工作的比较好；在堆优化的 dijkstra 中，pairing 脱颖而出。

2.2.4 例题

OJ1605：双堆维护求中位数。使用 binary。

```
//FUS File Prefix 201502192
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <queue>
#include <set>
#include <bitset>
#include <map>
#include <cstdarg>
#define For(a,b,c) for(int a=b,dd=c;a<=dd;++a)
#define ForDown(a,b,c) for(int a=c;a>=b;--a)
#define CheckMin(a,b) (((a)>(b))?((a)=(b)):(a))
#define CheckMax(a,b) (((a)<(b))?((a)=(b)):(a))
#define IF (
#define DO_NOTHING (0)
#define THEN )?(
#define ELSE 1):(
#define DONE 1),1
#define ELSE_DONE ELSE DO_NOTHING, DONE
#define THEN_ELSE THEN DO_NOTHING, ELSE
#define O2_OPT
#define FUSDEBUG0
//input
int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,
__PrintCache[1<<22],*__PrintPt=__PrintCache,__IPS[100];va_list __P;
#ifdef ONLINE_JUDGE
inline int GetChar(){static char buf[1<<15],*S=buf,*T=buf;
     return (S==T)&&(T=(S=buf)+fread(buf,1,1<<15,stdin),S==T)?0:*S++;}
#else
#define GetChar() getchar()
#endif
template<class T>inline void Read(T &aa)
{
    while(__ReadCh=GetChar(),(__ReadCh<'0'||__ReadCh>'9')&&__ReadCh!='-')DO_NOTHING;
    __ReadCh=='-'?(__ReadBool=1,aa=0):(aa=__ReadCh-'0',__ReadBool=0);

while(__ReadCh=GetChar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';(__ReadBool)?aa=
-aa:1;
}
inline void ScanInt(int Num, ...){va_start(__P, Num);while(Num--)Read(*va_arg(__P, int*));}
inline int NextInt(){Read(__ReadT);return __ReadT;}
inline void Gets(char *s)
{
     while(__ReadCh=GetChar(),(__ReadCh==' '||__ReadCh=='\n'))DO_NOTHING;
     for(;(__ReadCh!=' '&&__ReadCh!='\n');__ReadCh=GetChar())*s++=__ReadCh;
     *s++=0;
}
inline char GetAlpha()
```

```
{
    for(__ReadCh=' ';!isalpha(__ReadCh);__ReadCh=GetChar());return __ReadCh;
}
//output
#define PutChar(c)  (*__PrintPt++=c)
#ifdef ONLINE_JUDGE
#define Output()
(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache,1)
#else
#define Output() PutChar(0),puts(__PrintCache)
#endif
template<class T>inline void Print(T x)
{
    if(!x)*__PrintPt++=48;else{(x<0)?x=-x,PutChar('-'):1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;PutChar(48+__IPS[__Prin
tIter--]));}
}
inline void Print(const char s[]){for(;*s;++s)PutChar(*s);}
#define ToString(x) #x
#define Fill(A,c) memset(A, c, sizeof(A))
//end

#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;

#define TAG binary_heap_tag
priority_queue<int, std::less<int>, TAG> Min;
priority_queue<int, std::greater<int>, TAG> Max;

int N, A;

int main()
{
    Max.push(1 << 30);
    Min.push(-(1 << 30));
    ScanInt(2, &N, &A);
    Max.push(A);
    Print(A), PutChar('\n');
    for(int i = 2; i <= N; ++i)
    {
        Read(A);
        (A > Max.top()) ? Max.push(A), 1 : (Min.push(A), 1);
       while(Min.size() + 1 < Max.size())
             Min.push(Max.top()), Max.pop();
       while(Min.size() > Max.size())
             Max.push(Min.top()), Min.pop();
        (i & 1)? Print(Max.top()), PutChar('\n'), 1: 1;
    }
    Output();
}
```

OJ1127：最短路裸题，使用 pairing 优化 dijkstra。

```
#include <bits/stdc++.h>
#define For(i, a, b) for(int i = a; i <= b; ++i)
#define ForDown(i, a, b) for(int i = b; i >= a; --i)

namespace tools
{
    inline bool checkMax(int &a, int b)
    {
        return (a < b ? a = b, 1 : 0);
    }

    inline bool checkMin(int &a, int b)
    {
        return (a > b ? a = b, 1 : 0);
    }

    inline int min(int a, int b){return (a < b ? a : b);}
    inline int max(int a, int b){return (a > b ? a : b);}
    inline int abs(int a, int b){return a > 0 ? a : -a;}

    template <class T> class Object
    {
    private:
        T* instance;
```

```
    public:
        operator T () {return *instance;}
        Object(T *a = NULL) : instance(a) {}
        Object(const T a)
        {
            instance = new int;
            *instance = a;
        }
        Object(const Object<T>& o) {instance = o.instance;}
        Object<T>& operator = (const Object<T>& o) {instance = o.instance; return *this;}
        Object<T>& operator = (T *a) {instance = a; return *this;}
        Object<T>& operator = (const T a) {instance = new int; *instance = a; return *this;}

        ~Object() {delete instance;}
    };

}

namespace io
{
    //input
    int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,
    __PrintCache[1<<25],*__PrintPt=__PrintCache,__IPS[100];
    inline int GetChar(){static char buf[1<<15],*S=buf,*T=buf;
        return (S==T)&&(T=(S=buf)+fread(buf,1,1<<15,stdin),S==T)?0:*S++;}
    template<class T>inline void read(T &aa)
    {
        while(__ReadCh=GetChar(),(__ReadCh<'0'||__ReadCh>'9'));
        (aa=__ReadCh-'0',__ReadBool=0);
        while(__ReadCh=GetChar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';
    }
    inline int nextInt(){read(__ReadT);return __ReadT;}
    inline int nextString(char *t)
    {
        static char *s;
        s=t;while(__ReadCh=GetChar(),(__ReadCh==' '||__ReadCh=='\n'));
        for(;(__ReadCh!=' '&&__ReadCh!='\n');__ReadCh=GetChar())*s++=__ReadCh;
        *s=0;return (s-t);
    }
    //output
    inline void putChar(char c)  {(*__PrintPt++=c);}
    inline void reverse() {--__PrintPt;}
    //void pushDown() srd((tim("\55\117\62")));
    inline void output()
    {(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache,1);}
    template<class T>inline void print(T x)
    {
        if(!x)*__PrintPt++=48;else{(x<0)?x=-x,putChar('-'),1:1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;putChar(48+__IPS[__Prin
tIter--]));}
    }
    template<class T>inline void print(T x, char c){print(x);putChar(c);}
    inline void puts(const char s[]){for(;*s;++s)putChar(*s);}
    //end
}


namespace graph
{
    const int MAXN = 100010;
    const int MAXM = 400010;

    int first[MAXN];
    int length[MAXM], to[MAXM], next[MAXM];
    int edgeTop;

    int S, T, numPoints;

    #define goThrough(i,a) for(int i=first[a];i;i=next[i])

    inline void addEdge(int a, int b, int l)
    {
        ++edgeTop;
        next[edgeTop] = first[a];
        first[a] = edgeTop;
        length[edgeTop] = l;
```

```
          to[edgeTop] = b;
     }
}

#include <ext/pb_ds/priority_queue.hpp>

namespace dijkstra_with_heap
{
     using namespace __gnu_pbds;
     using namespace graph;

     struct DistData
     {
          int at, dist;
          bool operator < (const DistData& b) const
          {
               return dist > b.dist;
          }
          DistData(int a = 0, int d = 0) : at(a), dist(d) {}
     };

     priority_queue<DistData> heap;
     priority_queue<DistData>::point_iterator address[MAXN];
     bool done[MAXN];

     int shortestPath()
     {
          For(i, 1, numPoints)
               address[i] = heap.push(DistData(i, (i == S ? 0 : 1000000000)));

          while(!done[T])
          {
               static DistData dd;
               static int at, dist;
               dd = heap.top();
               at = dd.at;
               dist = dd.dist;
               done[at] = 1;
               if(at == T)return dist;
               heap.pop();

               goThrough(i, at)
                    if((!done[to[i]]) && (address[to[i]]->dist > dist + length[i]))
                         heap.modify(address[to[i]], DistData(to[i], dist + length[i]));
          }

          return -1;
     }

}

namespace solve
{
     int m;

     void begin()
     {
          using namespace graph;
          using namespace io;
          read(numPoints);
          read(m);
          read(S); ++S;
          read(T); ++T;
          while(m--)
          {
               static int a, b, l;
               read(a), read(b), read(l);
               ++a, ++b;
               addEdge(a, b, l);
          }
     }

     void solve()
     {
          io::print(dijkstra_with_heap::shortestPath());
     }
```

```
    void end()
    {
        io::output();
    }

}

int main()
{
    solve::begin();
    solve::solve();
    solve::end();
}
```

2.3 tree

2.3.1 原型

```
template
<
    typename Key,                                   //【1】
    typename Mapped,                                //【2】
    typename Cmp_Fn = std::less<Key>,               //【3】
    typename Tag = rb_tree_tag,                     //【4】
    template
    <
        typename Const_Node_Iterator,
        typename Node_Iterator,
        typename Cmp_Fn_,
        typename Allocator_
    >
    class Node_Update = null_tree_node_update, //【5】
    typename Allocator = std::allocator<char>
>
class tree;
```

【1】映射的 key

【2】映射的 value。注意，如果 value 为 null_type（在低版本上为 null_mapped_type）时，tree 相当于 set，否则相当于 map。

【3】比较类，默认为<

【4】实现方式 tag

【5】附加信息，可支持求 k 大值

2.3.2 函数

包含 map 或 set 的所有功能。

join(&other)：清空 other 并合并至本身（要求值域不得相交）

split(v, &other)：把大于 v 的一半子树存入 other

如果 Node_Update 定义为 tree_order_statistics_node_update：

find_by_order(k)：查询第 k 大元素

order_of_key(m)：查询元素 m 的排名

2.3.3 实现方式

Tree 支持 rb_tree_tag（红黑树）、splay_tree_tag（伸展树）、ov_tree_tag（排序向量树）。其中，ov 树的效率是 n 平方级别的，我们不予考虑。对于红黑树和伸展树的选择，我们根据实际测试情况考虑。如果需要支持拆分合并，则优先考虑 splay，否则，splay 将远比红黑树慢。

2.3.4 例题

我们伟大的 tree 支持非常多的平衡树功能，比如第 k 大，比如区间删除和添加（删除 = 两次拆分 + 一次合并，添加 = 一次拆分 + 两次合并）。通过自定义比较类的方法，我们可以轻松实现一些很复杂的平衡树功能，而且运行速度也很快，比如这一题。

OJ2119：经典的相离圆形处理问题，用平衡树维护括号序列，并进行前驱查询，最后简单的树形 DP。利用 splay 维护，自定义比较类维护括号序列的比较，利用迭代器数组进行删除。

```
#include <bits/stdc++.h>
#define For(i, a, b) for(int i = a; i <= b; ++i)
#define ForDown(i, a, b) for(int i = b; i >= a; --i)

namespace tools
{
    inline bool checkMax(int &a, int b)
    {
        return (a < b ? a = b, 1 : 0);
    }

    inline bool checkMin(int &a, int b)
    {
        return (a > b ? a = b, 1 : 0);
    }

    inline int min(int a, int b){return (a < b ? a : b);}
    inline int max(int a, int b){return (a > b ? a : b);}
    inline int abs(int a, int b){return a > 0 ? a : -a;}

    template <class T> class Object
    {
    private:
        T* instance;
    public:
        operator T () {return *instance;}
        Object(T *a = NULL) : instance(a) {}
        Object(const T a)
        {
            instance = new int;
            *instance = a;
        }
        Object(const Object<T>& o) {instance = o.instance;}
        Object<T>& operator = (const Object<T>& o) {instance = o.instance; return *this;}
```

```
          Object<T>& operator = (T *a) {instance = a; return *this;}
          Object<T>& operator = (const T a) {instance = new int; *instance = a; return *this;}

          ~Object() {delete instance;}
     };

}

namespace io
{
     //input
     int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,
     __PrintCache[1<<25],*__PrintPt=__PrintCache,__IPS[100];
     //inline int getChar(){static char buf[1<<15],*S=buf,*T=buf;\
     //   return (S==T)&&(T=(S=buf)+fread(buf,1,1<<15,stdin),S==T)?0:*S++;}
     int (*getChar)() = getchar;
     template<class T>inline void read(T &aa)
     {
         while(__ReadCh=getChar(),(__ReadCh<'0'||__ReadCh>'9')&&__ReadCh!='-');
         __ReadCh=='-'?(__ReadBool=1,aa=0):(aa=__ReadCh-'0',__ReadBool=0);

while(__ReadCh=getChar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';(__ReadBool)?aa=
-aa:1;
     }
     inline int nextInt(){read(__ReadT);return __ReadT;}
     inline int nextString(char *t)
     {
          static char *s;
          s=t;while(__ReadCh=getChar(),(__ReadCh==' '||__ReadCh=='\n'));
          for(;(__ReadCh!=' '&&__ReadCh!='\n');__ReadCh=getChar())*s++=__ReadCh;
          *s=0;return (s-t);
     }
     //output
     int (*putChar)(int) = putchar;
     //inline void putChar(char c)  {(*__PrintPt++=c);}
     inline void reverse() {--__PrintPt;}
     //void pushDown() srd((tim("\55\117\62")));
     inline void output()
     {(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache);}
     template<class T>inline void print(T x)
     {
         if(!x)putChar('0');else{(x<0)?x=-x,putChar('-'),1:1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;putChar(48+__IPS[__Prin
tIter--]));}
     }
     template<class T>inline void print(T x, char c){print(x);putChar(c);}
     inline void puts(const char s[]){for(;*s;++s)putChar(*s);}
     //end
}

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

int n, x[100010], y[100010], r[100010] = {210000000}, w[100010], father[100010], currentLine;

namespace scan_line
{
     int count;
     int line[200010], circle[200010], order[200010];
     bool edit[200010];

     bool cmp(int a, int b)
     {
          return (line[a] < line[b]) || ((line[a] == line[b]) && (y[circle[a]] < y[circle[b]]))
               || ((line[a] == line[b]) && (y[circle[a]] == y[circle[b]]) && (edit[a] > edit[b]));
     }
}

#ifdef ONLINE_JUDGE
#define null_type null_mapped_type
#endif

#define Tree __gnu_pbds::tree<int, __gnu_pbds::null_type, cut_point::Compare,
__gnu_pbds::splay_tree_tag>
#define TreePointer Tree::iterator
```

```
namespace cut_point
{
     int top;
     int cir[200010], up[200010];

     inline double sqr(double a){return a * a;}

     struct Compare
     {
          //本程序的绝妙之处：如何定义比较函数使得平衡树能维护括号序列
          //但也是性能瓶颈，因为手写平衡树可以直接插入到合适的位置
          //虽然如此，这也只比手写 splay 的程序 25 个点共慢 1.1s 罢了
          inline bool operator()(int a, int b)
          {
               static double aa, bb;
               if(!cir[a])return 1;
               if(!cir[b])return 0;
               if(cir[a] == cir[b])return up[a] < up[b];
               aa = sqrt(sqr(r[cir[a]]) - sqr(currentLine - x[cir[a]]));
               bb = sqrt(sqr(r[cir[b]]) - sqr(currentLine - x[cir[b]]));
               aa = y[cir[a]] + (up[a] ? aa : -aa);
               bb = y[cir[b]] + (up[b] ? bb : -bb);
               return (aa < bb) || ((aa == bb) && (a < b));
          }
     };
     int upLine[100010], downLine[100010];
     TreePointer address[200010];
}

namespace tree
{
     Tree tree;
     inline void add(int c)
     {
          using namespace cut_point;
          //printf("add cir %d\n", c);
          ++top, cir[top] = c, up[top] = 0, downLine[c] = top; address[top] =
tree.insert(top).first;
          ++top, cir[top] = c, up[top] = 1, upLine[c] = top; address[top] =
tree.insert(top).first;
          //printf("added top = %d %d\n", top-1,top);
          //printf("verify %d %d %d\n", *address[top-1], *address[top], tree.size());
     }
     inline void del(int c)
     {
          using namespace cut_point;
          //printf("del cir %d %d %d\n", c, *address[upLine[c]], *address[downLine[c]]);
          tree.erase(address[upLine[c]]);
          tree.erase(address[downLine[c]]);
     }
}

namespace tree_graph
{
     const int MAXN = 100010;
     int first[MAXN], next[MAXN], son[MAXN], numSon[MAXN];
     int sonTop;
     #define forEachSon(i,at) for(int i = first[at]; i; i = next[i])
     inline void addSon(int f, int s)
     {
          ++sonTop;
          next[sonTop] = first[f];
          first[f] = sonTop;
          son[sonTop] = s;
          ++numSon[f];
     }

}

namespace solve
{

     void begin()
     {
          io::read(n);
          For(i, 1, n)
```

```
        {
            using namespace io;
            using namespace scan_line;
            read(x[i]), read(y[i]), read(r[i]), read(w[i]);
            ++count, order[count] = count, line[count] = x[i] - r[i], circle[count] = i,
edit[count] = 1;
            ++count, order[count] = count, line[count] = x[i] + r[i], circle[count] = i,
edit[count] = 0;
            father[i] = i;
        }
    }

    int dp(int at)
    {
        using namespace tree_graph;
        int ans = 0;
        forEachSon(i, at)
            ans += dp(son[i]);
        return tools::max(ans, w[at]);

    }

    void solve()
    {
        using namespace scan_line;
        using namespace tree;
        using namespace cut_point;
        add(0);
        std::sort(order + 1, order + count + 1, cmp);
        For(t, 1, count)
        {
            static int i, c;
            //printf("%d ", t);
            i = order[t];
            currentLine = line[i];
            c = circle[i];
            if(edit[i])
            {
                static TreePointer iter;
                add(c);
                iter = cut_point::address[upLine[c]];
                ++iter;
                //printf("    next = %d\n", *iter);
                father[c] = up[*iter] ? cir[*iter] : father[cir[*iter]];
                //printf("fa[%d] = %d\n", c, father[c]);
            }
            else del(c);
            //for(TreePointer i=tree::tree.begin();i!=tree::tree.end();++i)
                //printf("At = %d Cir = %d Up = %d Y = %d\n",*i,cir[*i],up[*i],y[cir[*i]]);
        }
        For(i, 1, n)
            tree_graph::addSon(father[i], i);

        io::print(dp(0));
    }

    void end()
    {
        io::output();
    }
}



int main()
{
    solve::begin();
    solve::solve();
    solve::end();
}
```

使用 pb_ds 实现良好的“$p14y”神树，不仅编码简单，根本没有涉及到直接操作平衡树，而且真正实现的地方也容易实现和 debug（比如程序未处理半径为 0 的圆，导致还没添加就删除），是竞赛中不会、懒得、不想、没时间、

一时忘了、不太会实现平衡树的选手的首要选择，更是对拍的绝妙选择。

OJ2119 附标准程序：by Trinkle

```
#include<cstdio>
#include<cmath>
#include<cstring>
#include<algorithm>
#define ld double
#define N 200010
char ch,B[1<<15],*S=B,*T=B;char getc(){
     return S==T&&(T=(S=B)+fread(B,1,1<<15,stdin),S==T)?0:*S++;
}
int aa;int F(){
     while(ch=getc(),ch<'0'||ch>'9');aa=ch-'0';
     while(ch=getc(),ch>='0'&&ch<='9')aa=aa*10+ch-'0';return aa;
}
int n,m,i,j,x,y,f[N],far[N],l,r,q[N],la[N],et;
struct E{int to,nxt;}e[N];
#define add(x,y) (e[++et]=(E){y,la[x]},la[x]=et)
#define abs(x) (x>0?x:-(x))
#define sig(x) (x>0?1:-1)
#define sqr(x) ((x)*(x))
struct P{ld x,y,r;int w;}a[N];
bool operator<(const P&a,const P&b){return a.x<b.x||a.x==b.x&&a.y<b.y;}
struct L{ld x;int n;}b[N];
bool operator<(const L&a,const L&b){return a.x<b.x||a.x==b.x&&a.n<b.n;}
int fa[N],son[N][2],siz[N],key[N],tot,tag[N][2];
void pu(int t){
     if(t)siz[t]=siz[son[t][0]]+siz[son[t][1]]+1;
}
void rtt(int t){
     int f=fa[t],p=son[f][1]==t;
     son[fa[t]=fa[f]][son[fa[f]][1]==f]=t;
     (son[f][p]=son[t][!p])?fa[son[f][p]]=f:1;
     pu(son[fa[f]=t][!p]=f);
}
void splay(int t,int rt=0){
     for(int f;(f=fa[t])!=rt;rtt(t))
     if(fa[f]!=rt)rtt(son[f][1]==t^son[fa[f]][1]==f?t:f);pu(t);
}
bool cmp(ld tim,int i,int j){
     int x=sig(i),y=sig(j);i=abs(i),j=abs(j);
     return
a[i].y+x*sqrt(sqr(a[i].r)-sqr(a[i].x-tim))<a[j].y+y*sqrt(sqr(a[j].r)-sqr(a[j].x-tim));
}
int findl(int t){int lasid;
     for(splay(t),lasid=t=son[t][0];t;lasid=t,t=son[t][1]);
     return lasid;
}
int findr(int t){int lasid;
     for(splay(t),lasid=t=son[t][1];t;lasid=t,t=son[t][0]);
     return lasid;
}
void ins(ld tim,int k){
     int t=son[0][1],lasid=0;
     while(t)lasid=t,t=son[t][cmp(tim,key[t],k)];
     key[tag[abs(k)][k>0]=++tot]=k;
     fa[tot]=lasid,son[lasid][cmp(tim,key[lasid],key[tot])]=tot;
     splay(tot);
}
void del(int i){
     int t,l,r;
     t=tag[i][0],l=findl(t),r=findr(t);splay(l),splay(r,l),son[r][0]=0;
     t=tag[i][1],l=findl(t),r=findr(t);splay(l),splay(r,l),son[r][0]=0;
}
int main(){
     for(n=F(),i=1;i<=n;i++)a[i]=(P){F(),F(),F(),F()};
     a[n+1].r=1<<30;std::sort(a+1,a+1+n);
     for(i=1;i<=n;i++)
     b[++m]=(L){a[i].x-a[i].r,i},
     b[++m]=(L){a[i].x+a[i].r,-i};
     std::sort(b+1,b+1+m);
     //renew
     tot=2;fa[2]=1,key[1]=-n-1,key[2]=n+1,siz[1]=2,siz[2]=1,son[1][1]=2,son[0][1]=1;
     for(i=1;i<=m;i++)
     if(b[i].n>0){//ins
```

```
            ins(b[i].x+0.001,-b[i].n);
            ins(b[i].x+0.001,b[i].n);
            j=key[findl(tag[b[i].n][0])];
            if(abs(j)==n+1)far[b[i].n]=0;
            else if(j<0)far[b[i].n]=-j;
            else far[b[i].n]=far[j];
        }
        else del(-b[i].n);
        for(i=1;i<=n;i++)add(far[i],i);
        for(q[l=r=0]=0;l<=r;l++)
        for(i=la[q[l]];i;i=e[i].nxt)q[++r]=e[i].to;
        for(i=r;i;i--)f[far[q[i]]]+=std::max(f[q[i]],a[q[i]].w);
        printf("%d\n",f[0]);
}
```

Trinkle 学长的平衡树是一个良好的实现，虽然和主题无关，但仍推荐大家学习，作为一种技能掌握。

2.4 hash、trie

2.4.1 使用方式

hash 的定义：

cc_hash_table<K, V, Equals, Hash>：Equals 为相等函数类，Hash 为一个范围散列函数类

gp_hash_table<K, V, Equals, Hash>：Hash 为一个探针序列函数类

trie 的定义

trie<K, V>：要求 K 可以迭代

如果 K 是 string 之类的内置或标准库类型，可以直接使用<K, V>的方式定义，不需要附加任何函数类。

2.4.2 效率

实际测试中，hash 效率很快，trie 并未发挥出应有的效率。在大多数情况，我们可以用 hash 代替 map。

2.4.3 例题

OJ1759：字符串为点的名称的最短路，用 hash 管理标号（比 map 快 3 倍）

```
#include <bits/stdc++.h>
#define For(i, a, b) for(int i = a; i <= b; ++i)
#define ForDown(i, a, b) for(int i = b; i >= a; --i)

namespace tools
{
    inline bool checkMax(int &a, int b)
    {
        return (a < b ? a = b, 1 : 0);
    }

    inline bool checkMin(int &a, int b)
    {
        return (a > b ? a = b, 1 : 0);
```

```
    }

    inline int min(int a, int b){return (a < b ? a : b);}
    inline int max(int a, int b){return (a > b ? a : b);}
    inline int abs(int a, int b){return a > 0 ? a : -a;}

    template <class T> class Object
    {
    private:
        T* instance;
    public:
        operator T () {return *instance;}
        Object(T *a = NULL) : instance(a) {}
        Object(const T a)
        {
            instance = new int;
            *instance = a;
        }
        Object(const Object<T>& o) {instance = o.instance;}
        Object<T>& operator = (const Object<T>& o) {instance = o.instance; return *this;}
        Object<T>& operator = (T *a) {instance = a; return *this;}
        Object<T>& operator = (const T a) {instance = new int; *instance = a; return *this;}

        ~Object() {delete instance;}
    };

}

namespace io
{
    //input
    int __ReadBool,__RO,__PrintIter,__ReadT;char __ReadCh,
    __PrintCache[1<<25],*__PrintPt=__PrintCache,__IPS[100];
    inline int getChar(){static char buf[1<<15],*S=buf,*T=buf;
        return (S==T)&&(T=(S=buf)+fread(buf,1,1<<15,stdin),S==T)?0:*S++;}
    //int (*getChar)() = getchar;
    template<class T>inline void read(T &aa)
    {
        while(__ReadCh=getChar(),(__ReadCh<'0'||__ReadCh>'9'));
        (aa=__ReadCh-'0',__ReadBool=0);
        while(__ReadCh=getChar(),__ReadCh>='0'&&__ReadCh<='9')aa=aa*10+__ReadCh-'0';
    }
    inline int nextInt(){read(__ReadT);return __ReadT;}
    inline int nextString(char *t)
    {
        static char *s;
        s=t;while(__ReadCh=getChar(),(__ReadCh==' '||__ReadCh=='\n'));
        for(;(__ReadCh!=' '&&__ReadCh!='\n');__ReadCh=getChar())*s++=__ReadCh;
        *s=0;return (s-t);
    }
    //output
    //int (*putChar)(int) = putchar;
    inline void putChar(char c)  {(*__PrintPt++=c);}
    inline void reverse() {--__PrintPt;}
    //void pushDown() srd((tim("\55\117\62")));
    inline void output()
    {(fwrite(__PrintCache,1,__PrintPt-__PrintCache,stdout),__PrintPt=__PrintCache);}
    template<class T>inline void print(T x)
    {
        if(!x)*__PrintPt++=48;else{(x<0)?x=-x,putChar('-'),1:1;

for(__PrintIter=0;x;x/=10)__IPS[++__PrintIter]=x%10;for(;__PrintIter;putChar(48+__IPS[__Prin
tIter--]));}
    }
    template<class T>inline void print(T x, char c){print(x);putChar(c);}
    inline void puts(const char s[]){for(;*s;++s)putChar(*s);}
    //end
}

namespace graph
{
    const int MAXN = 100010;
    const int MAXM = 400010;

    int first[MAXN];
    int length[MAXM];
    int to[MAXM], next[MAXM];
```

```
     int edgeTop;

     int S, T, numPoints;

     #define goThrough(i,a) for(int i=first[a];i;i=next[i])

     inline void addEdge(int a, int b, int l)
     {
          ++edgeTop;
          next[edgeTop] = first[a];
          first[a] = edgeTop;
          length[edgeTop] = l;
          to[edgeTop] = b;
     }
}

namespace spfa
{
     using namespace graph;
     int dist[MAXN];
     int queue[1 << 23], l, r;
     bool inside[MAXN];

     int shortestPath()
     {
          memset(dist, 100, sizeof(dist));
          l = r = numPoints + 10;
          dist[S] = 0;
          inside[S] = 1;
          queue[r++] = S;
          while(l < r)
          {
               static int at;
               static int d;
               at = queue[l++];
               d = dist[at];
               //printf("at = %d d = %d\n", at, d);
               goThrough(i, at)
               if(dist[to[i]] > d + length[i])
               {
                    dist[to[i]] = d + length[i];
                    //from[to[i]] = at;
                    if(!inside[to[i]])
                    {
                         inside[to[i]] = 1;
                         queue[((dist[queue[l]] > dist[to[i]]) ? --l : r++)] = to[i];
                    }
               }
               inside[at] = 0;
          }

          return dist[T];
     }
}

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>

namespace solve
{
     using namespace graph;
     int m;
     int t;
     std::string a, b;
     __gnu_pbds::cc_hash_table<std::string, int> hash;

     void begin()
     {
          std::ios::sync_with_stdio(0);
          std::cin >> numPoints >> m;
          S = 1, T = 2;
          For(i, 1, numPoints)
          {
               std::cin >> a;
               hash[a] = i;
          }
          while(m--)
```

```
        {
            std::cin >> a >> b >> t;
            static int aa, bb;
            aa = hash[a], bb = hash[b];
            addEdge(aa, bb, t);
            addEdge(bb, aa, t);
        }
    }

    void solve()
    {
        std::cout << spfa::shortestPath() << std::endl;
    }

    void end()
    {

    }
}


int main()
{
    solve::begin();
    solve::solve();
    solve::end();
}
```

3 选择合适的库

在信息学竞赛中，有时候我们需要实现一些数据结构时，会面临选择是否手写，甚至纠结于使用 STL 还是 pb_ds 之间。这里给出一些参考。注意，没提到的数据结构说明不建议在竞赛中使用（一般效率太低）。

需要什么？	可以用什么？
基本数组操作，不担心内存（队列、栈）	数组
基本数组操作，担心内存，需要动态开	数组（使用 new int[n]） vector
需要大量任意位置（区间）插入删除、不需随机访问	list
需要以上所有功能，且需要大量随机访问，甚至可持久化	rope
非整数的数组下标	cc_hash_table
需要一个堆	priority_queue，使用默认配置
需要类似集合的功能	tree<T, null_type/*较低版本编译器为 null_mapped_type*/>
需要平衡二叉查找树	tree

即使我们有了如此丰富的工具，它们可能也因为过度封装无法被我们直接使用。如果时间充裕，我们还需要自己学会写一些数据结构，以便比赛时使用。