PB_DS 库
1）头文件
#include<ext/rope>
2)调用命名空间
using namespace __gnu_cxx;
先介绍几个可能使用到的函数
1）append()
string &append(const string &s,int pos,int n);//把字符串 s 中从 pos 开始的 n 个字符连接到当前字符串的结尾 或
a.append(b); 2）substr()
s.substr(0,5);//获得字符串 s 中从第零位开始长度为 5 的字符串（默认时长度为刚好开始位置到结尾）
定义/声明
rope<char> str;
<crope>r="abcdefg"
具体内容
总的来说，
1）运算符：rope 支持 operator += -= + - < ==
2）输入输出：可以用<<运算符由输入输出流读入或输出。
3）长度/大小：调用 length()，size()都可以哦
4）插入/添加等：
push_back(x);//在末尾添加 x
insert(pos,x);//在 pos 插入 x，自然支持整个 char 数组的一次插入
erase(pos,x);//从 pos 开始删除 x 个
copy(pos,len,x);//从 pos 开始到 pos+len 为止用 x 代替
replace(pos,x);//从 pos 开始换成 x
substr(pos,x);//提取 pos 开始 x 个
at(x)/[x];//访问第 x 个元素
访问
1）迭代器：不说，在竞赛是超时大忌
2）单点访问，直接用数组形式调用下标即可
应用
一、bzoj1269 文本编辑器
如果想看正常版本的看我的 splay 平衡树代码
实现操作： 1.（已知）move k：移动光标到目标，初始为 0 2.（已知）prev：光标前移一个字符 3.（已知）next：光标后移一个字符 4.insert n s：在光标后插入长度为 n 的字符串 s 光标位置不变 5.delete n 删除光标后的 n 个字符，光标位置不变 6.rotate n 反转光标后的 n 个字符，光标位置不变 7.get 输出光标后一个字符，光标位置不变
solution
为实现反转操作且保证不超时，我们不调用 rope 自带的可怕函数，暴力构建两个 rope，插入时一个正序插入一个倒序插入，区间即为子串赋值
```
#include<cstdio>
#include<ext/rope>
#include<iostream>
using namespace std;
using namespace __gnu_cxx;
inline int Rin(){
  int x=0,c=getchar(),f=1;
  for(;c<48||c>57;c=getchar())
    if(!(c^45))f=-1;
  for(;c>47&&c<58;c=getchar())
    x=(x<<1)+(x<<3)+c-48;
  return x*f;
}
int n,pos,x,l;
rope<char>a,b,tmp;
char sign[10],ch[1<<22],rch[1<<22];
int main(){
  n=Rin();
  while(n--){
    scanf("%s",sign);
    switch(sign[0]){
    case'M':pos=Rin();break;
    case'P':pos--;break;
    case'N':pos++;break;
    case'G':putchar(a[pos]);putchar('\n');break;
    case'I':
      x=Rin();
      l=a.length();
      for(int i=0;i<x;i++){
    do{ch[i]=getchar();}
    while(ch[i]=='\n');
    rch[x-i-1]=ch[i];
      }
      ch[x]=rch[x]='\0';
      a.insert(pos,ch);
      b.insert(l-pos,rch);
      break;
    case'D':
      x=Rin();
      l=a.length();
      a.erase(pos,x);
      b.erase(l-pos-x,x);
      break;
    case'R':
      x=Rin();
      l=a.length();
      tmp=a.substr(pos,x);
      a=a.substr(0,pos)+b.substr(l-pos-x,x)+a.substr(pos+x,l-pos-x);
      b=b.substr(0,l-pos-x)+tmp+b.substr(l-pos,pos);
      break;
    }
  }
  return 0;
}
```
头文件：
#include<ext/pb_ds/priority_queue.hpp>

用法：
主要用 pairing_heap_tag，配对堆，比 thin_heap_tag 快
和::priority queue 用法大致相同，但多了可并堆的 join(), modify() , erase() 等
例如： __gnu_pbds::priority_queue<node,less<node>,pairing_heap_tag> pq;
less 是 stl 里的比较器，需要 using namespace std, 也可以换成 greater。
例题:N 个点，M 条边的有向图，求点 1 到点 N 的最短路（保证存在）。
1<=N<=1000000，1<=M<=10000000
第一行两个整数 N、M，表示点数和边数。
第二行六个整数 T、rxa、rxc、rya、ryc、rp。
前 T 条边采用如下方式生成：
1.初始化 x=y=z=0。
2.重复以下过程 T 次：
x=(x*rxa+rxc)%rp;y=(y*rya+ryc)%rp;
a=min(x%n+1,y%n+1);b=max(y%n+1,y%n+1);
则有一条从 a 到 b 的，长度为 1e8-100*a 的有向边。后 M-T 条边采用读入方式：
接下来 M-T 行每行三个整数 x,y,z，表示一条从 x 到 y 长度为 z 的有向边。
1<=x,y<=N，0<z,rxa,rxc,rya,ryc,rp<2^31

```
#include<iostream>
#include<algorithm>
#include<climits>
#include<cstdio>
#include<ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
const int M=1000005;
inline int read(){
    int r=0;char c=getchar();
    while(c<'0'||c>'9') c=getchar();
    while(c>='0'&&c<='9') {r=r*10+c-'0';c=getchar();}
    return r;
}
struct node{
    int i;
    ll v;
    node(int a=0,ll c=0){i=a;v=c;}
    bool operator < (node b) const{
        return v>b.v;
    }
};
typedef __gnu_pbds::priority_queue<node,less<node>,pairing_heap_tag> heap;//!!!
heap::point_iterator hit[M];
heap pq;
int n;
ll d[M];
int last[1000005],cnt;
struct data{int to,next,v;}e[10000005];
void insert(int u,int v,int w){
    e[++cnt].to=v;e[cnt].next=last[u];last[u]=cnt;e[cnt].v=w;
}
inline void setpath(){
    int T,rxa,rxc,rya,ryc,rp,x,y,z,m;
    x=y=z=0;
    n=read();m=read();
    T=read();rxa=read();rxc=read();rya=read();ryc=read();rp=read();
    int a,b,q=min(n,T);
    for(int i=0;i<q;i++){
        x=((ll)x*rxa+rxc)%rp;
        y=((ll)y*rya+ryc)%rp;
        a=min(x%n+1,y%n+1);
        b=y%n+1;
        insert(a,b,100000000-100*a);
    }
    for(int i=0;i<m-T;i++)
    {
        x=read(),y=read(),z=read();
        insert(x,y,z);
    }
}
ll dijkstra()
{
    for(int i=1;i<=n;i++) d[i]=LLONG_MAX;
    hit[1]=pq.push(node(1,0));
    d[1]=0;
    int o;
    while(!pq.empty()){
        o=pq.top().i;
        pq.pop();
        if(o==n) return d[o];
        for(int i=last[o];i;i=e[i].next){
            if(d[e[i].to]>d[o]+e[i].v){
                d[e[i].to]=d[o]+e[i].v;
                if(hit[e[i].to]==0) hit[e[i].to]=pq.push(node(e[i].to,d[e[i].to]));
                else pq.modify(hit[e[i].to],node(e[i].to,d[e[i].to]));
            }
        }
    }
}
int main(){
    setpath();
    cout<<dijkstra();
    return 0;
}
```