

代码模板

zdq

SCUT

August 16, 2021

Contents

开始	2
宏定义	2
快读	2
对拍	2
数学	3
模乘模幂	3
GCD	3
CRT	3
高次同余方程	4
BSGS	4
EXBSGS	4
二次剩余	5
线性筛	6
ϕ 单点欧拉函数	6
Miller-Rabin 素性测试	6
Pollard-Rho 分解质因数	7
组合数	7
exLucas	7
类欧几里得算法	8
多项式乘法 FFT	8
数值积分	9
日期操作	10
用于跳转的常量	10
辅助函数	10
日期和整数的一一对应	10
二维计算几何	10
点向量基本运算	10
位置关系	11
多边形	11
求多边形面积	11
判断点是否在多边形内	11
凸包	11
凸包直径·平面最远点对	12
平面最近点对	12
圆	13
三点垂心	13
最小覆盖圆	13
图论	13
存图	13
最短路	13
Dijkstra	13
LCA	14
连通性	14
有向图强联通分量	14
二分图匹配	15
匈牙利算法求二分图无权最大匹配	15
KM 算法求二分图带权最大匹配	15
数据结构	17
手写整数哈希	17
堆式线段树	17
小根堆	18
Treap	18

开始

宏定义

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4  typedef __int128 LLL;
5  typedef unsigned u32;
6  typedef unsigned long long u64;
7  typedef long double LD;
8  typedef pair<int,int> pii;
9  #define ff first
10 #define ss second
11 #define il inline
12 #define pln putchar('\n')
13 #define For(i,a,b) for(int i=(a),(i##i)=(b);i<=(i##i);++i)
14 #define Rep(i,n) for(int i=0,(i##i)=(n);i<(i##i);++i)
15 #define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)
16 const int M=1000000007,INF=0x3f3f3f3f;
17 const long long INFLL=0x3f3f3f3f3f3f3fLL;
18 const int N=1000007;
```

快读

```
1  ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);

1  template <typename T>
2  inline bool read(T &x) {
3      x = 0; char c = getchar(); int f = 1;
4      while (!isdigit(c) && (c != '-') && (c != EOF)) c = getchar();
5      if (c == EOF) return 0;
6      if (c == '-') f = -1, c = getchar();
7      while (isdigit(c)) { x = x * 10 + (c & 15); c = getchar();}
8      x *= f; return 1;
9  }

10
11 template <typename T, typename... Args>
12 inline bool read(T &x, Args &...args) {
13     bool res = 1;
14     res &= read(x);
15     res &= read(args...);
16     return res;
17 }
```

对拍

```
1  //in.txt
2  //AC.exe std.txt
3  //MY.exe my.txt
4
5  void init(){
6      FILE*F=fopen("int.txt","w");
7
8      //srand(time(0));
9      //int a=(long long)rand()*rand()%1001;
10     //fscanf(F,"%d",&a);fprintf(F,"%d\n",a);
11
12     fclose(F);
13 }
14
15 int main(){
16     init();
17     while(1){
18         system("AC.exe < in.txt > std.txt");
19
20         system("MY.exe < in.txt > my.txt");
21
22         if(system("fc std.txt my.txt")){
23             puts("WA");
24         }
```

```

24         return 0;
25     }else puts("AC\n\n");
26
27     init();
28 }
29 }

```

数学

模乘模幂

- longlong 范围用 fpl

```

1  inline LL mul(LL a, LL b, LL p) {
2      LL res = a * b - ((LL)((LD)a * b / p) * p);
3      return res < 0 ? res + p : (res < p ? res : res - p);
4  }
5
6  inline LL fp(LL a, LL b, LL Mod) {
7      LL res = (Mod != 1);
8      for (; b; b >>= 1, a = a * a % Mod)
9          if (b & 1)
10             res = res * a % Mod;
11     return res;
12 }
13
14 inline LL fpl(LL a, LL b, LL Mod) {
15     LL res = (Mod != 1);
16     for (; b; b >>= 1, a = mul(a, a, Mod))
17         if (b & 1)
18             res = mul(res, a, Mod);
19     return res;
20 }

```

GCD

```

1  template <typename T>
2  inline T gcd(T a, T b) {
3      while (b){
4          T t = b;
5          b = a % b;
6          a = t;
7      }
8      return a;
9  }
10
11 template <typename T>
12 inline T lcm(T a, T b) { return a / gcd(a, b) * b; }
13
14 template<typename T>
15 inline T exgcd(T a,T b,T&x,T&y){
16     x=1,y=0;T m=0,n=1;
17     while(b){
18         const T q=a/b;
19         tie(x,m)=make_tuple(m,x-q*m);
20         tie(y,n)=make_tuple(n,y-q*n);
21         tie(a,b)=make_tuple(b,a-q*b);
22     }
23     return a;
24 }

```

CRT

- 需要 GCD 64 位模乘
- 用来合并同余方程
- 返回最小正数解或最小非负解无解返回-1

```

1 inline LL Crt(LL a1, LL a2, LL mod1, LL mod2) {
2     LL u, v;
3     LL g = exgcd(mod1, mod2, u, v);
4     if ((a2 - a1) % g) return -1;
5     LL m12 = abs(lcm(mod1, mod2));
6     LL res = (mul(mod1, mul(u, ((a2 - a1) / g), m12), m12) + a1) % m12;
7     return res <= 0 ? res + m12 : res; /* 求最小正数解还是非负解 */
8 }

```

高次同余方程

- 以下 p 均为正整数

BSGS

- 求最小非负整数 x 满足 $a^x \equiv b \pmod{p}$
- 要求 $(a, p) = 1$ p 不必是质数无解返回 -1
- 需要 `fp` 和 `map/unorded_map` 如果 p 超过 $1e9$ 用 LL 版本
- 期望复杂度 $O(\sqrt{p})$
- 自带哈希可以被卡 TLE (luogu P4192)
- 用 `map` 复杂度多个 \log 慢
- 手写整数哈希 `unordered_map<int, int, custom_hash>h;`
- 若多组询问 a, p 不变且始终满足 $(b, p) = 1$ 可以复用 `h` 降低复杂度

```

1 inline int log_p(int a, int b, int p){
2     if ((b - 1) % p == 0) return 0;
3     if ((b - a) % p == 0) return 1;
4     //在%p 前特判 此后满足 p>1 且 result>1
5     a %= p; (a < 0) && (a += p);
6     b %= p; (b < 0) && (b += p);
7     //assert(abs(gcd(a,p))==1);//调用时保证 a&p 则此行可省
8     unordered_map<int, int> h; //仔细判断出题人卡不卡
9     int t = (int)sqrt(p) + 1;
10    for (int j = 0; j < t; ++j, b = (LL)b * a % p) h[b] = j;
11    int at = (a = fp(a, t, p));
12    for (int i = 1, j = t; i <= t; ++i, a = (LL)a * at % p, j += t){
13        auto bat = h.find(a);
14        if (bat != h.end()) return j - (*bat).second;
15    } return -1;
16 }

```

EXBSGS

- 求最小非负整数 x 满足 $a^x \equiv b \pmod{p}$
- 不要求正整数 a, p 互质无解返回 -1
- 注意乘法溢出

```

1 inline int getinv(int a, int p){
2     int m = 0, n = 1, x = 1, y = 0, b = p;
3     while (b){
4         int q = a / b;
5         tie(x, m) = make_tuple(m, x - q * m);
6         tie(y, n) = make_tuple(n, y - q * n);
7         tie(a, b) = make_tuple(b, a - q * b);
8     }
9     x %= p; (x < 0) && (x += p);
10    return x;
11 }
12
13 inline int log(int a, int b, int p){
14     if ((b - 1) % p == 0) return 0;
15     if ((b - a) % p == 0) return 1;

```

```

16 a %= p; (a < 0) && (a += p);
17 b %= p; (b < 0) && (b += p);
18 int _a = a, _b = b, _p = p;
19 int d = gcd(a, p), ax = 1, t = 0;
20 while (d > 1){
21     if (b % d != 0)break; //若 b 不能再提出 d 说明 result<=t
22     b /= d; p /= d; ax = (LL)ax * (a / d) % p;
23     ++t;
24     d = gcd(a, p);
25 }
26 for (int i = 2, at = (LL)_a * _a % _p; i <= t; ++i, at = (LL)at * _a % _p)
27     if (at == _b) return i;
28 if (d > 1) return -1;
29 int res = log_p(a, (LL)b * getinv(ax, p) % p, p);
30 return res < 0 ? -1 : res + t;
31 }

```

二次剩余

判定定理

- p 是奇素数且 n 和 p 互质
- $n^{\frac{p-1}{2}} \bmod p = \begin{cases} 1 & n \\ -1 & n \end{cases}$

Cipolla 算法

- 寻找非负整数 x 满足 $x^2 \equiv n \pmod{p}$ 其中 p 为质数
- 期望复杂度 $O(\log p)$
- 需要用到快速幂 $\text{fp}()$ 若 p 大于 $1e9$ 要换 $\text{mul}()$ 和 $\text{fpl}()$
- 注意乘法溢出
- 先找到 a 使 $a^2 - n$ 是非二次剩余
- 定义 $w = i^2 = a^2 - n$ 则 $x = (a + i)^{\frac{p+1}{2}}$
- 无解返回 -1

```

1 namespace Cipolla{
2     LL w,p,d;
3
4     struct dat{
5         LL x,y;//复数 x+yi
6         dat operator*(const dat&a)const{
7             return (dat){(x*a.x+y*a.y%p*w)%p,(x*a.y+y*a.x)%p};
8         }
9     };
10
11     inline LL ask(LL n,LL P){//n 为非二次剩余返回-1
12         p=P;n%=p;if(n==0||p==2)return n;
13         d=(p-1)/2;
14         if(fp(n,d,p)!=1)return -1;//检验是否有解
15         LL a;
16         while(1){
17             a=(LL)rand()*rand()%p;
18             w=(a*a-n)%p;(w<0)&&(w+=p);
19             if(fp(w,d,p)==(p-1))break;//找到了非二次剩余 w
20         }
21         dat res={1,0};++d;
22         for(dat u={a,1};d>=1,u=u*u;if(d&1)res=res*u;
23         return res.x;
24     }
25 }
26
27 signed main(){//luogu P5491
28     int t;scanf("%d",&t);

```

```

29     while(t--){
30         LL n,p;scanf("%lld%lld",&n,&p);
31         LL ans=Cipolla::ask(n,p);
32         if(ans==1)puts("HoLa!");
33         else if(ans==0)puts("0");
34         else printf("%lld %lld\n",min(ans,p-ans),max(ans,p-ans));
35     }
36     return 0;
37 }

```

线性筛

```

1 struct primenumberlist{
2     #define MAXN (100000000)
3     int cnt, pri[100000000];
4     bool np[MAXN + 10];
5     primenumberlist(){
6         np[1] = 1; cnt = 0;
7         for (int i = 2; i <= MAXN; ++i) {
8             if (!np[i]) pri[++cnt] = i;
9             for (int j = 1; j <= cnt; ++j) {
10                 LL t = pri[j] * i;
11                 if (t > MAXN) break;
12                 np[t] = 1;
13                 if (!(i % pri[j])) break;
14             }
15         }
16     }
17 } prime;

```

ϕ 单点欧拉函数

```

1 template <typename T>
2 inline T phi(T x) {
3     T res = x;
4     for (T i = 2; i * i <= x; ++i)
5         if ((x % i) == 0) {
6             res = res / i * (i - 1);
7             while ((x % i) == 0) x /= i;
8         }
9     if (x > 1) res = res / x * (x - 1);
10    return res;
11 }

```

Miller-Rabin 素性测试

- $n \leq 10^{18}$
- 需要 64 位模乘 64 位模幂

```

1 inline bool MR(LL x, LL n, int t) {
2     LL las = x;
3     for (int i = 1; i <= t; ++i) {
4         x = mul(x, x, n);
5         if (x == 1 && las != 1 && las != (n - 1)) return 0;
6         las = x;
7     }
8     return x == 1;
9 }
10
11 inline bool isPrime(LL n) {
12     if (n == 4685624825598111 || n < 2) return 0;
13     if (n == 2 || n == 3 || n == 7 || n == 61 || n == 24251) return 1;
14     LL d = n - 1;
15     int t = 0;
16     while ((d & 1) == 0) d >>= 1, ++t;
17     return MR(fpl(2, d, n), n, t) && MR(fpl(61, d, n), n, t);
18 }

```

Pollard-Rho 分解质因数

- 需要 64 位模乘 gcd
- 求 n 的一个大于 1 的因子可能返回 n 本身
- 调用 PR() 前务必判断 n 的素性检查 $n > 1$

```
1 mt19937 mt(time(0)); //随机化
2 inline LL PR(LL n) {
3     LL x = uniform_int_distribution<LL>(0, n - 1)(mt), s, t, c = uniform_int_distribution<LL>(1, n - 1)(mt); //随机化
4     for (int gol = 1; 1; gol <= 1, s = t, x = 1) {
5         for (int stp = 1; stp <= gol; ++stp) {
6             t = (mul(t, t, n) + c) % n;
7             x = mul(x, abs(s - t), n);
8             if ((stp & 127) == 0) {
9                 LL d = gcd(x, n);
10                if (d > 1) return d;
11            }
12        }
13        LL d = gcd(x, n);
14        if (d > 1) return d;
15    }
16 }
```

组合数

- 数较小模数为较大质数求逆元
- - 如果模数固定可以 $O(n)$ 预处理阶乘的逆元
- 数较大模数为较小质数用 Lucas 定理
- -

$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n \bmod p}^{m \bmod p} (\bmod p)$$

- 数较大模数较小用 exLucas 定理求 $C_n^m \bmod P$

exLucas

- 需要模乘 CRT
- $O(P \log P)$
- 不要求 P 为质数

```
1 namespace EXLUCAS {
2     inline LL idxp(LL n, LL p) {
3         LL nn = n;
4         while (n > 0) nn -= (n % p), n /= p;
5         return nn / (p - 1);
6     }
7
8     LL facp(LL n, LL p, LL pk) {
9         if (n == 0) return 1;
10        LL res = 1;
11        if (n >= pk) {
12            LL t = n / pk, k = 1, els = n - t * pk;
13            for (LL i = 1; i <= els; ++i) if (i % p) k = k * i % pk;
14            res = k;
15            for (LL i = els + 1; i < pk; ++i) if (i % p) k = k * i % pk;
16            res = res * fp(k, n / pk, pk) % pk;
17        }
18        else for (LL i = 1; i <= n; ++i) if (i % p) res = res * i % pk;
19        return res * facp(n / p, p, pk) % pk;
20    }
21
22    inline LL exlucas(LL n, LL m, LL p, LL pk, LL k) {
23        LL a = facp(n, p, pk) * fp(facp(n - m, p, pk) * facp(m, p, pk) % pk, pk / p * (p - 1) - 1, pk) % pk;
24        LL b = idxp(n, p) - idxp(m, p) - idxp(n - m, p);
```

```

25     if (b >= k) return 0;
26     while (b--) a *= p;
27     return a % pk;
28 }
29
30 /* 接口 */ inline LL exlucas(LL n, LL m, LL p) {
31     LL a = 0, b = 1;
32     for (LL i = 2; i * i <= p; ++i) {
33         if (p % i) continue;
34         LL t = 0, pk = 1;
35         while (p % i == 0) ++t, p /= i, pk *= i;
36         a = Crt(a, exlucas(n, m, i, pk, t), b, pk);
37         b *= pk;
38     }
39     return (p > 1) ? Crt(a, exlucas(n, m, p, p, 1), b, p) : a;
40 }
41 }

```

类欧几里得算法

- 计算直线下整点数
- $f = \sum [(ai+b)/c]$ $g = \sum [(ai+b)/c]^2$ $i=0..n$ $a, b, n \in \mathbb{N}$ $c \in \mathbb{N}^*$
- 复杂度 $\log(\max\{a, c\})$

```

1 struct dat{LL f, g, h;};
2 const LL i2 = 499122177, i3 = 332748118, M = 998244353; //预处理出模 M 意义下 2 和 3 的逆元
3 dat f(LL a, LL b, LL c, LL n){
4     LL ac = a / c, bc = b / c;
5     LL n2 = (n * (n + 1) % M) * i2 % M, n3 = n2 * (2ll * n + 1) % M * i3 % M;
6     dat res = {
7         (n2 * ac % M + (n + 1) * bc % M) % M,
8         (ac * n3 % M + bc * n2 % M) % M,
9         (ac * ac % M * n3 % M +
10          bc * bc % M * (n + 1) % M + ac * bc % M * n2 % M * 2ll) % M};
11     a %= c; b %= c; if (a == 0) return res;
12     LL m = (a * n + b) / c;
13     dat p = f(c, c - b - 1, a, m - 1);
14     LL fc = (n * m % M - p.f + M) % M, gc = (n2 * m % M - i2 * (p.f + p.h) % M + M) % M;
15     return{(res.f + fc) % M, (res.g + gc) % M,
16            (res.h + 2ll * (bc * fc % M + ac * gc % M) % M +
17             n * m % M * m % M - 2ll * p.g - p.f + 3ll * M) % M};}

```

多项式乘法 FFT

- fft 计算卷积复杂度 $O(n \log n)$ luogu P3803

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=2100009;
4
5 //fft 计算卷积 复杂度 O(nlogn) luogu P3803
6 namespace FFT{
7     const double PI=acos(-1.0);
8     struct comp{
9         double x,y;
10        comp(double _x=0,double _y=0):x(_x),y(_y){}
11        comp operator+(const comp&a)const{return comp(x+a.x,y+a.y);}
12        comp operator-(const comp&a)const{return comp(x-a.x,y-a.y);}
13        comp operator*(const comp&a)const{return comp(x*a.x-y*a.y,x*a.y+y*a.x);}
14    };
15    int rev[N]; //位逆序置换
16    inline void getrev(int len){ //每次 len 变化应重新计算 rev
17        const int d=len>>1;
18        for(int i=1;i<len;++i) rev[i]=(rev[i>>1]>>1)|((i&1)*d);
19    }
20
21    inline void dft(comp y[],int len,int on){ //on==1 DFT on== -1 IDFT
22        for(int i=1;i<len;++i) if(i<rev[i]) swap(y[i],y[rev[i]]);

```

```

23     for(int mid=1;mid<len;mid<=1){
24         comp wn(cos(PI/mid),on*sin(PI/mid));
25         for(int st=0,t=(mid<=1);st<len;st+=t){
26             comp w(1,0);
27             for(int i=st,j=st+mid;j<st+t;++i,++j,w=w*wn){
28                 comp u=y[i],v=w*y[j];
29                 y[i]=u+v;y[j]=u-v;
30             }
31         }
32     }
33     if(on== -1)for(int i=0;i<len;++i)y[i].x/=len,y[i].y/=len;//保证结果在 R 上可只除实部
34 }
35 }
36
37 int n,m;
38 FFT::comp f1[N],f2[N];
39 signed main(){
40     scanf("%d%d",&n,&m);
41     for(int i=0;i<=n;++i)scanf("%lf",&f1[i].x);
42     for(int i=0;i<=m;++i)scanf("%lf",&f2[i].x);
43     int len=1;while(len<=m+n)len<=1;FFT::getrev(len);
44     FFT::dft(f1,len,1);FFT::dft(f2,len,1);
45     for(int i=0;i<len;++i)f1[i]=f1[i]*f2[i];
46     FFT::dft(f1,len,-1);
47     for(int i=0;i<=n+m;++i)printf("%d ",(int)(f1[i].x+0.5));
48     return 0;
49 }

```

数值积分

- 自适应辛普森算法复杂度 $O(\sqrt{\epsilon})$ (看玄学调参)
- luogu p4542

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct integration{f(x)dx≈(r-l)*(f(l)+f(r)+4*f((l+r)/2))/6
5      typedef double T;
6      const T EPS=(1e-6)*15;
7
8      T aa,bb,cc,dd;
9      integration(T _a,T _b,T _c,T _d):aa(_a),bb(_b),cc(_c),dd(_d){}
10
11      T f(T x)const{//被积函数 f(x)
12          return (cc*x+dd)/(aa*x+bb);
13      }
14
15      T simpson(T len,T l,T mid,T r)const{//时间优化 传入算好的函数值
16          return len*(l+4*mid+r)/6;
17      }
18      T asr(T l,T r,T a,T c,T e,int stp)const{
19          const T len=r-l,mid=l+len/2,lmid=l+len/4,rmid=mid+len/4;
20          const T b=f(lmid),d=f(rmid);
21          const T L=simpson(len/2,a,b,c),R=simpson(len/2,c,d,e),o=L+R-simpson(len,a,c,e);
22          return (abs(o)<=EPS&&stp<0)?(L+o/15):asr(l,mid,a,b,c,stp-1)+asr(mid,r,c,d,e,stp-1);
23      }
24      T operator()(T l,T r)const{return asr(l,r,f(l),f(l+(r-l)/2),f(r),6);};//最小迭代次数
25 };
26
27 signed main(){
28     double a,b,c,d,l,r;
29     scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&l,&r);
30     integration F(a,b,c,d);
31     printf("%.6lf\n",F(l,r));
32     return 0;
33 }

```

日期操作

用于跳转的常量

```
1  const LL year_1[2]={365, 366};
2  const LL year_400=1460097;
3  const LL m_day[13]={ (LL)0x3f3f3f, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

辅助函数

```
1  inline bool isLeap(LL t){return (t % 400 == 0) || ((t % 4 == 0) && (t % 100 != 0));}
2  inline bool pick(LL a, LL b){return ((isLeap(a) && b <= 2) || (isLeap(a + 1) && b > 2));}
3  inline LL dayThisMonth(LL y, LL m){return m_day[m] + isLeap(y) * (m == 2);}
```

日期和整数的一一对应

- LL 可以改成 int

```
1  struct MY_DATE{
2      LL year, month, day;
3      MY_DATE(LL y = 2021, LL m = 1, LL d = 1) : year(y), month(m), day(d){};
4      LL p(MY_DATE op = {0, 0, 0}){//日期转换为整数
5          LL y = year - op.year, m = month - op.month, d = day - op.day;
6          if (m <= 2){ y--; m += 12;}
7          return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
8      }
9      MY_DATE run(LL k){//当前日期过 k 天
10         k += p();
11         LL x = k + 1789995, n = 4 * x / 146097, i, j, d;
12         x -= (146097 * n + 3) / 4;
13         i = (4000 * (x + 1)) / 1461001;
14         x -= 1461 * i / 4 - 31;
15         j = 80 * x / 2447;
16         d = x - 2447 * j / 80;
17         x = j / 11;
18         return MY_DATE(100 * (n - 49) + i + x, j + 2 - 12 * x, d);
19     }
20 };
```

二维计算几何

- Point 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 1..n

点向量基本运算

```
1  template <typename T>
2  struct Point {
3      T x, y;
4      Point() {}
5      Point(T u, T v) : x(u), y(v) {}
6      Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
7      Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
8      Point operator*(const T &a) const { return Point(x * a, y * a); }
9      T operator*(const Point &a) const { return x * a.x + y * a.y; }
10     T operator%(const Point &a) const { return x * a.y - y * a.x; }
11     double len() const { return hypot(x, y); }
12     double operator^(const Point &a) const { return (a - (*this)).len(); }
13     double angle() const { return atan2(y, x); }
14     bool id() const { return y < 0 || (y == 0 && x < 0); }
15     bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
16 };
17 typedef Point<double> point;
18
19 #define sqr(x) ((x) * (x))
20 const point O(0, 0);
21 const double PI(acos(-1.0)), EPS(1e-8);
```

```

22 inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
23 inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
24 inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }

```

位置关系

```

1 inline bool in_same_seg(point p, point a, point b) {
2     if (fabs(mul(p, a, b)) < EPS) {
3         if (a.x > b.x) swap(a, b);
4         return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
5     } else return 0;
6 }
7
8 inline bool is_right(point st, point ed, point a) {
9     return ((ed - st) % (a - st)) < 0;
10 }
11
12 inline point intersection(point s1, point t1, point s2, point t2) {
13     return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
14 }
15
16 inline bool parallel(point a, point b, point c, point d) {
17     return dcmp((b - a) % (d - c), 0);
18 }
19
20 inline double point2line(point p, point s, point t) {
21     return fabs(mul(p, s, t) / (t - s).len());
22 }
23
24 inline double point2seg(point p, point s, point t) {
25     return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
26 }

```

多边形

求多边形面积

```

1 inline double area(int n, point s[]) {
2     double res = 0;
3     s[n + 1] = s[1];
4     for (int i = 1; i <= n; ++i)
5         res += s[i] % s[i + 1];
6     return fabs(res / 2);
7 }

```

判断点是否在多边形内

- 特判边上的点
- 使用了 $a[1] \dots a[n+1]$ 的数组

```

1 inline bool in_the_area(point p, int n, point area[]) {
2     bool ans = 0; double x;
3     area[n + 1] = area[1];
4     for (int i = 1; i <= n; ++i) {
5         point p1 = area[i], p2 = area[i + 1];
6         if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
7         if (p1.y == p2.y) continue;
8         if (p.y < min(p1.y, p2.y)) continue;
9         if (p.y >= max(p1.y, p2.y)) continue;
10        ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
11    }
12    return ans;
13 }

```

凸包

- *Andrew* 算法
- $O(n \log n)$

- 可以应对凸包退化成直线/单点的情况但后续旋转卡壳时应注意特判
- 注意是否应该统计凸包边上的点

```

1 inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }
2
3 inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
4     sort(p + 1, p + 1 + n, pcmp1);
5     int m = 0;
6     for (int i = 1; i <= n; ++i) {
7         while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
8         ans[++m] = p[i];
9     }
10    int k = m;
11    for (int i = n - 1; i >= 1; --i) {
12        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
13        ans[++m] = p[i];
14    }
15    return m - (n > 1); //返回凸包有多少个点
16 }

```

凸包直径·平面最远点对

- 旋转卡壳算法
- $O(n)$
- 凸包的边上只能有端点，否则不满足严格单峰
- 凸包不能退化成直线，调用前务必检查 $n \geq 3$
- 使用了 $a[1] \dots a[n+1]$ 的数组

```

1 inline double Rotating-Caliper(int n, point a[]) {
2     a[n + 1] = a[1];
3     double ans = 0;
4     int j = 2;
5     for (int i = 1; i <= n; ++i) {
6         while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
7         ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
8     }
9     return ans;
10 }

```

平面最近点对

- 分治 + 归并
- $O(n \log n)$

```

1 namespace find_the_closest_pair_of_points {
2     const int N = 200010; //maxn
3     inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
4     inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }
5
6     point a[N], b[N];
7     double ans;
8     inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }
9
10    void find(int l, int r) {
11        if (l == r) return;
12        if (l + 1 == r) {
13            if (a[l] > a[r]) swap(a[l], a[r]);
14            upd(a[l], a[r]); return;
15        }
16        int mid = (l + r) >> 1;
17        double mx = (a[mid + 1].x + a[mid].x) / 2;
18        find(l, mid); find(mid + 1, r);
19        int i = l, j = mid + 1;
20        for (int k = l; k <= r; ++k) b[k] = a[(j > r) || (i <= mid && a[j] > a[i]) ? (i++) : (j++)];
21        for (int k = l; k <= r; ++k) a[k] = b[k];
22        int tot = 0;
23        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
24            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);

```

```

25         b[++tot] = a[k];
26     }
27 }
28
29 //接口
30 inline double solve(int n, point ipt[]){
31     ans = 0x3f3f3f3f3f3f3f3f; //max distance
32     for (int i = 1; i <= n; ++i) a[i] = ipt[i];
33     sort(a + 1, a + 1 + n, cmp1);
34     find(1, n);
35     return ans;
36 }
37 }

```

圆

三点垂心

```

1 inline point geto(point p1, point p2, point p3) {
2     double a = p2.x - p1.x;
3     double b = p2.y - p1.y;
4     double c = p3.x - p2.x;
5     double d = p3.y - p2.y;
6     double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
7     double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
8     return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
9 }

```

最小覆盖圆

- 随机增量 $O(n)$

```

1 inline void min_circlefill(point &o, double &r, int n, point a[]) {
2     mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
3     o = a[1];
4     r = 0;
5     for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
6         o = a[i];
7         r = 0;
8         for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
9             o = (a[i] + a[j]) * 0.5;
10            r = (a[i] ^ a[j]) * 0.5;
11            for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
12                o = geto(a[i], a[j], a[k]);
13                r = (o ^ a[i]);
14            }
15        }
16    }
17 }

```

图论

存图

- 前向星
- 注意边数开够

```

1 int Head[N], Ver[N*2], Next[N*2], Ew[N*2], Gtot=1;
2 inline void graphinit(int n) {Gtot=1; for(int i=1; i<=n; ++i) Head[i]=0;}
3 inline void edge(int u, int v, int w=1) {Ver[++Gtot]=v; Next[Gtot]=Head[u]; Ew[Gtot]=w; Head[u]=Gtot;}
4 #define go(i,st,to) for (int i=Head[st], to=Ver[i]; i; i=Next[i], to=Ver[i])

```

最短路

Dijkstra

- 非负权图

```

1 namespace DIJK{//适用非负权图 满足当前 dist 最小的点一定不会再被松弛
2     typedef pair<long long,int> pii;
3     long long dist[N];//存最短路长度
4     bool vis[N];//记录每个点是否被从队列中取出 每个点只需第一次取出时扩展
5     priority_queue<pii,vector<pii>,greater<pii> >pq;//维护当前 dist[] 最小值及对应下标 小根堆
6
7     inline void dijk(int s,int n){//s 是源点 n 是点数
8         while(pq.size())pq.pop();for(int i=1;i<=n;++i)dist[i]=INFL,vis[i]=0;//所有变量初始化
9         dist[s]=0;pq.push(make_pair(0,s));
10        while(pq.size()){
11            int now=pq.top().second;pq.pop();
12            if(vis[now])continue;vis[now]=1;
13            go(i,now,to){
14                const long long relx(dist[now]+Ew[i]);
15                if(dist[to]>relx){dist[to]=relx;pq.push(make_pair(dist[to],to));};//松弛
16            }
17        }
18    }
19 }

```

LCA

- 倍增求 lca
- 数组开够

```

1 namespace LCA_Log{
2     int fa[N][22],dep[N];
3     int t,now;
4     void dfs(int x){
5         dep[x]=dep[fa[x][0]]+1;
6         go(i,x,to){
7             if(dep[to])continue;
8             fa[to][0]=x;for(int j=1;j<=t;++j)fa[to][j]=fa[fa[to][j-1]][j-1];
9             dfs(to);
10        }
11    }
12
13    //初始化接口
14    inline void lcainit(int n,int rt){//记得初始化全部变量
15        now=1;t=0;while(now<n)++t,now<=1;
16        for(int i=1;i<=n;++i)dep[i]=0,fa[i][0]=0;
17        for(int i=1;i<=t;++i)fa[rt][i]=0;
18        dfs(rt);
19    }
20
21    //求 lca 接口
22    inline int lca(int u,int v){
23        if(dep[u]>dep[v])swap(u,v);
24        for(int i=t;~i;--i)if(dep[fa[v][i]]>=dep[u])v=fa[v][i];
25        if(u==v)return u;
26        for(int i=t;~i;--i)if(fa[u][i]!=fa[v][i])u=fa[u][i],v=fa[v][i];
27        return fa[u][0];
28    }
29 }

```

连通性

有向图强联通分量

- tarjan $O(n)$

```

1 namespace SCC{
2     int dfn[N],clk,low[N];
3     bool ins[N];int sta[N],tot; //栈 存正在构建的强连通块
4     vector<int>scc[N];int c[N],cnt;//cnt 为强联通块数 scc[i] 存放每个块内点 c[i] 为原图每个结点属于的块
5     void dfs(int x){
6         dfn[x]=low[x]=(++clk);//low[] 在这里初始化
7         ins[x]=1;sta[++tot]=x;
8         go(i,x,to){
9             if(!dfn[to]){dfs(to);low[x]=min(low[x],low[to]);};//走树边

```

```

10         else if (ins[to]) low[x] = min(low[x], dfn[to]); // 走返祖边
11     }
12     if (dfn[x] == low[x]) { // 该结点为块的代表元
13         ++cnt; int u;
14         do { u = sta[tot--]; ins[u] = 0; c[u] = cnt; scc[cnt].push_back(u); } while (x != u);
15     }
16 }
17 inline void tarjan(int n) { // n 是点数
18     for (int i = 1; i <= cnt; ++i) scc[i].clear(); // 清除上次的 scc 防止被卡 MLE
19     for (int i = 1; i <= n; ++i) dfn[i] = ins[i] = 0; tot = clk = cnt = 0; // 全部变量初始化
20     for (int i = 1; i <= n; ++i) if (!dfn[i]) dfs(i);
21     for (int i = 1; i <= n; ++i) c[i] += n; // 此行 (可以省略) 便于原图上加点建新图 加新点前要初始化 Head[] = 0
22 }
23 }

```

二分图匹配

匈牙利算法求二分图无权最大匹配

- 复杂度 $O(nm)$

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  namespace Hungary{
4      const int N = 1000009;
5      // 图中应储存左部到右部的边 左点编号 1..n 右点编号 1..m
6      int Head[N], Ver[N*2], Next[N*2], Gtot = 1; // 注意边数开够
7      inline void graphinit(int n) { Gtot = 1; for (int i = 1; i <= n; ++i) Head[i] = 0; }
8      inline void edge(int u, int v) { Ver[++Gtot] = v, Next[Gtot] = Head[u], Head[u] = Gtot; }
9      #define go(i, st, to) for (int i = Head[st], to = Ver[i]; i; i = Next[i], to = Ver[i]) // st 是左点, to 是 st 能到达的右点
10
11      int match[N], vis[N]; // 右点的匹配点和访问标记
12
13      bool dfs(int x) { // x 是左点
14          go(i, x, to) if (!vis[to]) {
15              vis[to] = 1;
16              if (!match[to] || dfs(match[to])) {
17                  match[to] = x; return 1;
18              }
19          }
20          return 0;
21      }
22
23      inline int ask(int n, int m) { // 左点数和右点数 返回最大匹配数
24          for (int i = 1; i <= m; ++i) match[i] = 0; int res = 0;
25          for (int i = 1; i <= n; ++i) {
26              for (int j = 1; j <= m; ++j) vis[j] = 0;
27              res += dfs(i);
28          }
29          return res;
30      }
31  }

```

KM 算法求二分图带权最大匹配

- 要求该图存在完美匹配该算法将最大化完美匹配的权值和
- 复杂度 $O(n^3)$
- naive 的写法复杂度为 $O((n^2)m)$ 在完全图上会退化至 $O(n^4)$ luogu P6577

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  namespace KM{
4      const int N = 509;
5      const long long INFLL = 0x3f3f3f3f3f3f3f3f; // 注意 INF 应足够大 至少远大于 n*n*Max{W}
6
7      int n; // 左右点数均为 n 标号均为 1..n
8      int w[N][N]; // 邻接矩阵存边权
9      bool r[N][N]; // 记录 i, j 之间是否有边连接 完全图/非 0 权图则不必要
10

```

```

11 inline void init() { //记得重写初始化
12     int m; scanf("%d%d", &n, &m);
13     for (int i=1; i<=m; ++i) {
14         int u, v, wt; scanf("%d%d%d", &u, &v, &wt);
15         w[u][v]=wt; r[u][v]=1;
16     }
17 }
18
19 long long la[N], lb[N], upd[N]; //左右顶标 每个右点对应的最小 delta 要开 longlong
20 int last[N]; //每个右点对应的回溯右点
21 bool va[N], vb[N];
22 int match[N]; //每个右点对应的左匹配点
23 bool dfs(int x, int fa) {
24     va[x]=1;
25     for (int y=1; y<=n; ++y) if (r[x][y] && !vb[y]) {
26         const long long dt=la[x]+lb[y]-w[x][y];
27         if (dt==0) { //相等子图
28             vb[y]=1; last[y]=fa;
29             if (!match[y] || dfs(match[y], y)) {
30                 match[y]=x;
31                 return 1;
32             }
33         } else if (upd[y]>dt) { //下次 dfs 直接从最小 delta 处开始
34             upd[y]=dt;
35             last[y]=fa; //用 last 回溯该右点的上一个右点以更新增广路
36         }
37     }
38     return 0;
39 }
40
41 inline void KM() {
42     for (int i=1; i<=n; ++i) { //初始化顶标
43         la[i]=-INFLL; lb[i]=0;
44         for (int j=1; j<=n; ++j) if (r[i][j]) la[i]=max(la[i], (long long)w[i][j]);
45     }
46     for (int i=1; i<=n; ++i) { //尝试给每一个左点匹配上右点 匹配失败则扩展相等子图重试至成功
47         memset(va, 0, sizeof va); //注意复杂度
48         memset(vb, 0, sizeof vb);
49         memset(last, 0, sizeof last);
50         memset(upd, 0x3f, sizeof upd);
51         int st=0; match[0]=i; //给起点 i 连一个虚右点 标号为 0
52         //不断尝试将右点 st 从已有的匹配中解放 以获得增广路
53         while (match[st]) { //当 st 到达非匹配右点直接退出
54             long long delta=INFLL;
55             if (dfs(match[st], st)) break; //st 的左点匹配到了新的右点则退出
56             for (int j=1; j<=n; ++j)
57                 if (!vb[j] && upd[j]<delta) { //下次从最小的 delta 处开始 DFS
58                     delta=upd[j];
59                     st=j;
60                 }
61             for (int j=1; j<=n; ++j) { //将交错树上的左顶标加 delta 右顶标减 delta 使更多的边转为相等边
62                 if (va[j]) la[j]-=delta;
63                 if (vb[j]) lb[j]+=delta;
64                 else upd[j]-=delta; //小问题: 这里在干啥 每次修改顶标后重新计算 upd 是否可行
65             }
66             vb[st]=1;
67         }
68         while (st) { //更新增广路
69             match[st]=match[last[st]];
70             st=last[st];
71         }
72     }
73
74     long long ans=0;
75     for (int i=1; i<=n; ++i) ans+=w[match[i]][i];
76     printf("%lld\n", ans);
77     for (int i=1; i<=n; ++i) printf("%d%c", match[i], " \n"[i==n]);
78 }
79 //signed main() {init(); KM(); return 0;}
80 }

```

数据结构

手写整数哈希

- 防止自带哈希被卡 T

```
1 struct custom_hash {
2     static uint64_t splitmix64(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
4         x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8     size_t operator()(uint64_t x) const {
9         static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
10        return splitmix64(x + FIXED_RANDOM);
11    }
12};
```

堆式线段树

- 区间求和和区间修改
- 空间估算
- 所有数组务必初始化

```
1 struct SegmentTree_Heap{
2     #define TreeLen (N<<2)          //N
3     #define lc(x) ((x)<<1)
4     #define rc(x) ((x)<<1|1)
5     #define sum(x) (tr[x].sum)      //
6     #define t(x) (t[x])            //
7
8     struct dat{
9         LL sum;
10        /* 这里写区间加法 */
11        dat operator+(const dat&brother){
12            dat result;
13            result.sum=sum+brother.sum;
14            return result;
15        }
16    }tr[TreeLen];
17    LL t[TreeLen]; //lazy tag
18
19    /* 单区间修改 */
20    inline void change(const int&x,const int&l,const int&r,const LL&d){
21        tr[x].sum=tr[x].sum+d*(r-l+1);
22        t[x]=t[x]+d;
23    }
24
25    inline void pushup(int x){tr[x]=tr[lc(x)]+tr[rc(x)];}
26
27    inline void pushdown(int x,const int&l,const int&r,const int&mid){
28        if(t(x)){//注意区间修改细节
29            change(lc(x),l,mid,t(x));
30            change(rc(x),mid+1,r,t(x));
31            t(x)=0;
32        }
33    }
34
35    void build(int x,int l,int r){
36        t(x)=0; // 记得初始化!!!
37        if(l==r){
38            sum(x)=0;
39            return;
40        }
41        int mid=(l+r)>>1;
42        build(lc(x),l,mid);
43        build(rc(x),mid+1,r);
44        pushup(x);
45    }
```

```

46
47 void add(int x,int l,int r,const int&L,const int&R,const LL&d){
48     if(L<=l&&r<=R){
49         change(x,l,r,d);
50         return;
51     }
52     int mid=(l+r)>>1;pushdown(x,l,r,mid);
53     if(L<=mid)add(lc(x),l,mid,L,R,d);
54     if(R>mid)add(rc(x),mid+1,r,L,R,d);
55     pushup(x);
56 }
57
58 LL ask(int x,int l,int r,const int&L,const int&R){
59     if(L<=l&&r<=R)return sum(x);
60     int mid=(l+r)>>1;pushdown(x,l,r,mid);
61     LL res=0;
62     if(L<=mid)res=(res+ask(lc(x),l,mid,L,R));
63     if(mid<R)res=(res+ask(rc(x),mid+1,r,L,R));
64     return res;
65 }
66 };

```

小根堆

```

1 namespace MyPQ{
2     typedef int pqdat;    ///
3     pqdat q[N];
4     int tot;
5     void up(int x){
6         while(x>1)
7             if(q[x]<q[x/2]){
8                 swap(q[x],q[x/2]);
9                 x/=2;
10            }else return;
11    }
12    void down(int x){
13        int ls=x*2;
14        while(ls<=tot){
15            if(ls<tot&&q[ls+1]<q[ls])++ls;
16            if(q[ls]<q[x]){
17                swap(q[x],q[ls]);x=ls;ls=x*2;
18            }else return;
19        }
20    }
21    void push(pqdat x){q[++tot]=x;up(tot);}
22    pqdat top(){return q[1];}
23    void pop(){if(!tot)return;q[1]=q[tot--];down(1);}
24    void pop(int k){if(!tot)return;q[k]=q[tot--];up(k);down(k);}
25 }

```

Treap

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=2000007,INF=(1ll<<30)+7;
4
5 //Treap 维护升序多重集
6 //支持操作: 数 <-> 排名 查询某数前驱后继
7 //操作数 x 可以不在集合中
8 //x 的排名: 集合中 <x 的数的个数 +1
9 //排名 x 的数: 集合中排名 <=x 的数中的最大数
10 //x 的前驱 比 x 小的最大数
11
12 struct treap{//所有点值不同 用副本数实现多重集
13     int l,r;
14     int v,w;//v 是数据 w 是维护堆的随机值
15     int num,sz;//num 是该点副本数 sz 是孩子树副本总数
16 }tr[N];int tot,rt;//tr[0] 始终全 0 使用范围 tr[1..n]
17 #define lc(x) tr[x].l
18 #define rc(x) tr[x].r

```

```

19 #define sz(x) tr[x].sz
20 #define num(x) tr[x].num
21 #define val(x) tr[x].v
22 #define wt(x) tr[x].w
23
24 inline int New(int x){
25     val(++tot)=x; wt(tot)=rand();
26     num(tot)=sz(tot)=1; return tot;
27 }
28
29 inline void upd(int p){sz(p)=sz(lc(p))+sz(rc(p))+num(p);}
30
31 inline void build(){//初始化 INF 和-INF 两个点
32     srand(time(0));
33     rt=1;tot=2;
34     rc(1)=2;val(1)=-INF;wt(1)=rand();num(1)=1;sz(1)=2;
35     val(2)=INF;wt(2)=rand();num(2)=1;sz(2)=1;
36 }
37
38 //调用时记得减一 askrk(rt,x)-1
39 int askrk(int p,int x){//当前子树中查询 x 的排名
40     if(p==0)return 1;//说明某子树所有数均比 x 大
41     if(x==val(p))return sz(lc(p))+1;
42     return x<val(p)?askrk(lc(p),x):askrk(rc(p),x)+sz(lc(p))+num(p);
43 }
44
45 //调用时记得加一 kth(rt,++rank)
46 int kth(int p,int rk){//当前子树中查询排名 rk 的数
47     if(p==0)return INF;//说明集合大小 <rk
48     if(sz(lc(p))>=rk)return kth(lc(p),rk);
49     rk-=sz(lc(p))+num(p);
50     return (rk>0)?kth(rc(p),rk):val(p);
51 }
52
53 inline void zig(int &p){//与左子节点交换位置
54     int q=lc(p);lc(p)=rc(q);rc(q)=p;
55     upd(p);p=q;upd(p);
56 }
57
58 inline void zag(int &p){//与右子节点交换位置
59     int q=rc(p);rc(p)=lc(q);lc(q)=p;
60     upd(p);p=q;upd(p);
61 }
62
63 //insert(rt,x)
64 void insert(int &p,int x){//当前子树中插入 x
65     if(p==0){p=New(x);return;}//x 首次插入
66     if(x==val(p)){++num(p);++sz(p);return;}
67     if(x<val(p)){
68         insert(lc(p),x);
69         if(wt(p)<wt(lc(p)))zig(p);//维护大根堆
70     }else{
71         insert(rc(p),x);
72         if(wt(p)<wt(rc(p)))zag(p);//维护大根堆
73     }
74     upd(p);
75 }
76
77 //erase(rt,x)
78 void erase(int &p,int x){//当前子树中删除一个 x
79     if(p==0)return;//已经无需删除
80     if(val(p)==x){//如果找到了 x 的位置
81         if(num(p)>1){//无需删点
82             --num(p);--sz(p);return;//如果有多个 x 维护副本数即可
83         }
84         if(lc(p)||rc(p)){//该点不是叶子节点 则不断向下调整至叶子节点
85             if(rc(p)==0||wt(lc(p))>wt(rc(p)))zig(p),erase(rc(p),x);//由于 rand() 的值域 & 大根堆的实现 故省略左子树为空的判断
86             else zag(p),erase(lc(p),x);
87             upd(p);
88         }else p=0;//是叶子节点则直接删除
89         return;

```

```

90     }
91     x<val(p)?erase(lc(p),x):erase(rc(p),x);upd(p);
92 }
93
94 int askpre(int x){
95     int id=1;//-INF 若没有前驱则返回-INF
96     //尝试自顶向下寻找 x 则 x 的前驱有两种情况
97     //1) 未找到 x 或 x 没有左子树 则前驱在搜索路径上
98     //2) 前驱是 x 的左子树中最大值 即 x 的左子树一直向右走
99     int p=rt;
100    while(p){
101        if(x==val(p)){//找到 x
102            if(lc(p)){p=lc(p);while(rc(p))p=rc(p);id=p;}
103            break;
104        }
105        if(val(p)<x&&val(p)>val(id))id=p;//每经过一个点尝试更新前驱
106        p=(val(p)>x?lc(p):rc(p));//找 x
107    }
108    return val(id);
109 }
110
111 int asknxt(int x){
112     int id=2;//INF
113     int p=rt;
114     while(p){
115         if(x==val(p)){
116             if(rc(p)){p=rc(p);while(lc(p))p=lc(p);id=p;}
117             break;
118         }
119         if(val(p)>x&&val(p)<val(id))id=p;
120         p=(val(p)>x?lc(p):rc(p));
121     }
122     return val(id);
123 }
124
125 #include <bits/stdc++.h>
126 using namespace std;
127
128 struct MY_IO{
129     #define DEBUG 1//本地调试
130     #define MAXSIZE (1 << 20)
131     inline bool isdigit(const char &x) { return x >= '0' && x <= '9'; } //字符集 看情况改
132     inline bool blank(const char &c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }
133     #if DEBUG //
134     #else //
135     char buf[MAXSIZE + 3], *p1, *p2, pbuf[MAXSIZE + 3], *pp;
136     MY_IO() : p1(buf), p2(buf), pp(pbuf) {}
137     ~MY_IO() { fwrite(pbuf, 1, pp - pbuf, stdout); }
138     #endif //
139
140     inline char gc(){
141     #if DEBUG //
142         return getchar(); //
143     #else //
144         if (p1 == p2)
145             p2 = (p1 = buf) + fread(buf, 1, MAXSIZE, stdin);
146         return p1 == p2 ? EOF : *p1++;
147     #endif //
148     }
149
150     inline void pc(const char &c){
151     #if DEBUG //
152         putchar(c); //
153     #else //
154         if (pp - pbuf == MAXSIZE)
155             fwrite(pbuf, 1, MAXSIZE, stdout), pp = pbuf;
156         *pp++ = c;
157     #endif //
158     }
159
160     template<typename T>inline bool read(T &x){
161         x = 0; char c = gc(); int f = 1;

```

```

38     while (!isdigit(c) && (c != '-') && (c != EOF)) c = gc();
39     if (c == EOF) return 0;
40     if (c == '-') f = -1, c = gc();
41     while (isdigit(c)) {x = x * 10 + (c & 15); c = gc();}
42     x *= f; return 1;
43 }
44
45 template<typename T, typename... Args>inline bool read(T &x, Args &...args){
46     bool res = 1; res &= read(x); res &= read(args...); return res;
47 }
48
49 inline int gets(char *s){
50     char c = gc(); while (blank(c) && c != EOF) c = gc();
51     if (c == EOF) return 0;
52     int len = 0;
53     while (!blank(c) && c != EOF) *s++ = c, c = gc(), ++len;
54     *s = 0; return len;
55 }
56
57 inline void getc(char &c){for (c = gc(); blank(c) && c != EOF; c = gc());}
58
59 /* 不能输出 (int)(-2^31) */ template<typename T>inline void write(T x){
60     if (x < 0) x = -x, pc('-');
61     static T sta[233];
62     int top = 0;
63     do{
64         sta[top++] = x % 10, x /= 10;
65     } while (x);
66     while (top) pc(sta[--top] + '0');
67 }
68
69 template <typename T>inline void write(T x, const char &Lastchar){write(x); pc(Lastchar);}
70
71 inline void puts(char *s){while ((*s) != 0)pc(*s++);}
72
73 inline int getline(char *s){
74     char c = gc();
75     int len = 0;
76     while (c != '\n' && c != EOF) *s++ = c, c = gc(), ++len;
77     *s = 0; return len;
78 }
79
80 inline void putline(char *s){while ((*s) != 0)pc(*s++); pc('\n');}
81 }IO;
82 #define read IO.read
83 #define write IO.write
84 #define gc IO.gc
85 #define pc IO.pc
86 #define gets IO.gets
87 #define getc IO.getc
88 #define puts IO.puts
89 #define getl IO.getline
90 #define putl IO.putline

```