

# 代码库

Spear of Longinus

2016 年 11 月 10 日

## 目录

<b>1</b>	<b>数论</b>	<b>5</b>
1.1	快速求逆元	5
1.2	扩展欧几里德算法	5
1.3	中国剩余定理	5
1.4	中国剩余定理 2	6
1.5	组合数取模	6
1.6	扩展小步大步	7
1.7	卢卡斯定理	7
1.8	小步大步	8
1.9	Miller Rabin 素数测试	8
1.10	Pollard Rho 大数分解	9
1.11	快速数论变换 (zky)	10
1.12	快速数论变换 (lyx)	11
1.13	原根	12
1.14	线性递推	13
1.15	线性筛	15
1.16	直线下整点个数	15
<b>2</b>	<b>数值</b>	<b>16</b>
2.1	高斯消元	16
2.2	快速傅立叶变换	18
2.3	单纯形法求解线性规划	19
2.4	自适应辛普森	21
2.5	多项式求根	21
<b>3</b>	<b>数据结构</b>	<b>23</b>
3.1	平衡的二叉查找树	23
3.1.1	Treap	23
3.1.2	Splay	26
3.2	坚固的数据结构	27
3.2.1	坚固的平衡树	27
3.2.2	坚固的字符串	28

3.2.3 坚固的左偏树	32
3.2.4 不坚固的斜堆	33
3.3 树上的魔术师	33
3.3.1 轻重树链剖分 (zky)	33
3.3.2 轻重树链剖分 (lyx)	34
3.3.3 Link Cut Tree(zky)	35
3.3.4 Link Cut Tree(lyx)	37
3.3.5 AAA Tree	39
3.4 ST	44
3.5 可持久化线段树	45
3.6 可持久化 Trie	46
3.7 k-d 树	49
3.8 莫队算法	52
3.9 树上在线莫队	53
3.10 整体二分	54
3.11 树状数组 kth	55
3.12 虚树	55
3.13 点分治 (zky)	56
3.14 元芳树	57
<b>4 图论</b>	<b>61</b>
4.1 强连通分量	61
4.1.1 点双连通分量 (lyx)	62
4.2 2-SAT 问题	63
4.3 二分图最大匹配	64
4.3.1 Hungary 算法	64
4.3.2 Hopcroft Karp 算法	65
4.4 二分图最大权匹配	66
4.5 最大流 (dinic)	68
4.6 最大流 (sap)	69
4.7 上下界网络流	71
4.7.1 无源汇的上下界可行流	71
4.7.2 有源汇的上下界可行流	71
4.7.3 有源汇的上下界最大流	71
4.7.4 有源汇的上下界最小流	71
4.8 最小费用最大流	72
4.8.1 稀疏图	72
4.8.2 稠密图	73
4.9 一般图最大匹配	76
4.10 无向图全局最小割	78
4.11 有根树的同构	80
4.12 哈密尔顿回路 (ORE 性质的图)	81
4.13 必经点树	83

<b>5</b>	<b>字符串</b>	<b>85</b>
5.1	模式匹配	85
5.1.1	KMP 算法	85
5.1.2	扩展 KMP 算法	86
5.1.3	AC 自动机	86
5.2	后缀三姐妹	87
5.2.1	后缀数组	87
5.2.2	后缀数组 (dc3)	89
5.2.3	后缀自动机-多串 LCS	91
5.2.4	后缀自动机-各长度字串出现次数最大值	92
5.2.5	后缀自动机-两串 LCS	94
5.3	回文三兄弟	95
5.3.1	马拉车	95
5.3.2	回文树 (lyx)	96
5.3.3	回文自动机 (zky)	97
5.4	循环串最小表示	98
<b>6</b>	<b>计算几何</b>	<b>99</b>
6.1	二维基础	99
6.1.1	点类	99
6.1.2	凸包	101
6.1.3	半平面交	101
6.1.4	最近点对	103
6.1.5	最小圆覆盖	103
6.1.6	凸包快速询问	104
6.2	多边形	107
6.2.1	判断点在多边形内部	107
<b>7</b>	<b>其他</b>	<b>107</b>
7.1	斯坦纳树	107
7.2	无敌的读入优化	108
7.3	最小树形图	109
7.4	DLX	110
7.5	插头 DP	112
7.6	某年某月某日是星期几	115
7.7	枚举大小为 $k$ 的子集	116
7.8	环状最长公共子串	116
7.9	LLMOD	118
<b>8</b>	<b>Java</b>	<b>118</b>
8.1	基础模板	118
<b>9</b>	<b>gedit</b>	<b>131</b>

<b>10 数学</b>	<b>131</b>
10.1 常用数学公式	131
10.1.1 求和公式	131
10.1.2 斐波那契数列	131
10.1.3 错排公式	132
10.1.4 莫比乌斯函数	132
10.1.5 伯恩赛德引理	132
10.1.6 五边形数定理	132
10.1.7 树的计数	132
10.1.8 欧拉公式	133
10.1.9 皮克定理	133
10.1.10 牛顿恒等式	133
10.2 平面几何公式	134
10.2.1 三角形	134
10.2.2 四边形	134
10.2.3 正 $n$ 边形	135
10.2.4 圆	135
10.2.5 棱柱	135
10.2.6 棱锥	136
10.2.7 棱台	136
10.2.8 圆柱	136
10.2.9 圆锥	137
10.2.10 圆台	137
10.2.11 球	137
10.2.12 球台	137
10.2.13 球扇形	138
10.3 立体几何公式	138
10.3.1 球面三角公式	138
10.3.2 四面体体积公式	138

## 1 数论

### 1.1 快速求逆元

返回结果:

$$x^{-1}(\text{mod})$$

使用条件:  $x \in [0, \text{mod})$  并且  $x$  与  $\text{mod}$  互质

---

```
1 LL inv(LL a, LL p){
2     LL d, x, y;
3     d = exgcd(a, p, x, y);
4     return d == 1 ? (x + p) % p : -1;
5 }
```

---

### 1.2 扩展欧几里德算法

返回结果:

$$ax + by = \text{gcd}(a, b)$$

时间复杂度:  $\mathcal{O}(n \log n)$

---

```
1 LL exgcd(LL a, LL b, LL &x, LL &y){
2     if(!b){
3         x = 1; y = 0; return a;
4     } else {
5         LL d = exgcd(b, a % b, x, y);
6         LL t = x; x = y; y = t - a / b * y;
7         return d;
8     }
9 }
```

---

### 1.3 中国剩余定理

返回结果:

$$x \equiv r_i(\text{mod } p_i) \quad (0 \leq i < n)$$

使用条件:  $p_i$  需两两互质

---

```
1 LL china(int n, int *a, int *m){
2     LL M = 1, d, x = 0, y;
3     for(int i = 0; i < n; i++)
4         M *= m[i];
```

```

5         for(int i=0;i<n;i++){
6             LL w=M/m[i];
7             d=exgcd(m[i],w,d,y);
8             y=(y%M+M)%M;
9             x=(x+y*w%M*a[i])%M;
10        }
11        while(x<0)x+=M;
12        return x;
13    }

```

---

## 1.4 中国剩余定理 2

```

1  //merge Ax=B and ax=b to A'x=B'
2  void merge(LL &A,LL &B,LL a,LL b){
3      LL x,y;
4      sol(A,-a,b-B,x,y);
5      A=lcm(A,a);
6      B=(a*y+b)%A;
7      B=(B+A)%A;
8  }

```

---

## 1.5 组合数取模

```

1  LL prod=1,P;
2  pair<LL,LL> comput(LL n,LL p,LL k){
3      if(n<=1)return make_pair(0,1);
4      LL ans=1,cnt=0;
5      ans=pow(prod,n/P,P);
6      cnt=n/p;
7      pair<LL,LL>res=comput(n/p,p,k);
8      cnt+=res.first;
9      ans=ans*res.second%P;
10     for(int i=n-n%P+1;i<=n;i++)if(i%p){
11
12         ans=ans*i%P;
13     }
14     return make_pair(cnt,ans);
15 }
16 pair<LL,LL> calc(LL n,LL p,LL k){
17     prod=1;P=pow(p,k,1e18);
18     for(int i=1;i<P;i++)if(i%p)prod=prod*i%P;

```

```

19     pair<LL,LL> res=comput(n,p,k);
20     // res.second=res.second*pow(p,res.first%k,P)%P;
21     // res.first-=res.first%k;
22     return res;
23 }
24 LL calc(LL n,LL m,LL p,LL k){
25     pair<LL,LL>A,B,C;
26     LL P=pow(p,k,1e18);
27     A=calc(n,p,k);
28     B=calc(m,p,k);
29     C=calc(n-m,p,k);
30     LL ans=1;
31     ans=pow(p,A.first-B.first-C.first,P);
32     ans=ans*A.second%P*inv(B.second,P)%P*inv(C.second,P)%P;
33     return ans;
34 }

```

---

## 1.6 扩展小步大步

```

1 LL solve2(LL a,LL b,LL p){
2     //a^x=b (mod p)
3     b%=p;
4     LL e=1%p;
5     for(int i=0;i<100;i++){
6         if(e==b)return i;
7         e=e*a%p;
8     }
9     int r=0;
10    while(gcd(a,p)!=1){
11        LL d=gcd(a,p);
12        if(b%d)return -1;
13        p/=d;b/=d;b=b*inv(a/d,p);
14        r++;
15    }LL res=BSGS(a,b,p);
16    if(res==-1)return -1;
17    return res+r;
18 }

```

---

## 1.7 卢卡斯定理

```

1 LL Lucas(LL n,LL m,LL p){
2     LL ans=1;
3     while(n&& m){
4         LL a=n%p,b=m%p;
5         if(a<b) return 0;
6         ans=(ans*C(a,b,p))%p;
7         n/=p;m/=p;
8     }return ans%p;
9 }

```

---

## 1.8 小步大步

返回结果:

$$a^x = b \pmod{p}$$

使用条件:  $p$  为质数

时间复杂度:  $\mathcal{O}(\sqrt{n})$

```

1 LL BSGS(LL a,LL b,LL p){
2     LL m=sqrt(p)+.5,v=inv(pw(a,m,p),p),e=1;
3     map<LL,LL>hash;hash[1]=0;
4     for(int i=1;i<m;i++){
5         e=e*a%p,hash[e]=i;
6     }for(int i=0;i<=m;i++){
7         if(hash.count(b)) return i*m+hash[b];
8         b=b*v%p;
9     }return -1;
10 }

```

---

## 1.9 Miller Rabin 素数测试

```

1 const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
2 bool check(long long n,int base) {
3     long long n2=n-1,res;
4     int s=0;
5     while(n2%2==0) n2>>=1,s++;
6     res=pw(base,n2,n);
7     if((res==1)|| (res==n-1)) return 1;
8     while(s--){
9         res=mul(res,res,n);
10        if(res==n-1) return 1;
11    }

```



```

12     return 0; // n is not a strong pseudo prime
13 }
14 bool isprime(const long long &n) {
15     if(n==2)
16         return true;
17     if(n<2 || n%2==0)
18         return false;
19     for(int i=0;i<12&&BASE[i]<n;i++){
20         if(!check(n,BASE[i]))
21             return false;
22     }
23     return true;
24 }

```

---

### 1.10 Pollard Rho 大数分解

时间复杂度:  $\mathcal{O}(n^{1/4})$

---

```

1 LL prho(LL n,LL c){
2     LL i=1,k=2,x=rand()%(n-1)+1,y=x;
3     while(1){
4         i++;x=(x*x%n+c)%n;
5         LL d=__gcd((y-x+n)%n,n);
6         if(d>1&&d<n)return d;
7         if(y==x)return n;
8         if(i==k)y=x,k<<=1;
9     }
10 }
11 void factor(LL n,vector<LL>&fat){
12     if(n==1)return;
13     if(isprime(n)){
14         fat.push_back(n);
15         return;
16     }LL p=n;
17     while(p>=n)p=prho(p,rand()%(n-1)+1);
18     factor(p,fat);
19     factor(n/p,fat);
20 }

```

---

## 1.11 快速数论变换 (zky)

返回结果:

$$c_i = \sum_{0 \leq j \leq i} a_j \cdot b_{i-j}(\text{mod}) \quad (0 \leq i < n)$$

使用说明: *magic* 是 *mod* 的原根

时间复杂度:  $\mathcal{O}(n \log n)$

---

```
1  /*
2  {(mod,G)}={(81788929,7),(101711873,3),(167772161,3)
3              ,(377487361,7),(998244353,3),(1224736769,3)
4              ,(1300234241,3),(1484783617,5)}
5  */
6  int mo=998244353,G=3;
7  void NTT(int a[],int n,int f){
8      for(register int i=0;i<n;i++)
9          if(i<rev[i])
10             swap(a[i],a[rev[i]]);
11     for (register int i=2;i<=n;i<=1){
12         static int exp[maxn];
13         exp[0]=1;exp[1]=pw(G,(mo-1)/i);
14         if(f==-1)exp[1]=pw(exp[1],mo-2);
15         for(register int k=2;k<(i>>1);k++)
16             exp[k]=1LL*exp[k-1]*exp[1]%mo;
17         for(register int j=0;j<n;j+=i){
18             for(register int k=0;k<(i>>1);k++){
19                 register int &pA=a[j+k],&pB=a[j+k+(i>>1)];
20                 register int A=pA,B=1LL*pB*exp[k]%mo;
21                 pA=(A+B)%mo;
22                 pB=(A-B+mo)%mo;
23             }
24         }
25     }
26     if(f==-1){
27         int rv=pw(n,mo-2)%mo;
28         for(int i=0;i<n;i++)
29             a[i]=1LL*a[i]*rv%mo;
30     }
31 }
32 void mul(int m,int a[],int b[],int c[]){
33     int n=1,len=0;
34     while(n<m)n<=1,len++;
35     for (int i=1;i<n;i++)
```

```

36         rev[i]=(rev[i>>1]>>1)|((i&1)<<(len-1));
37     NTT(a,n,1);
38     NTT(b,n,1);
39     for(int i=0;i<n;i++)
40         c[i]=1LL*a[i]*b[i]%mo;
41     NTT(c,n,-1);
42 }

```

---

## 1.12 快速数论变换 (lyx)

---

```

1  int Pow(int x,int y,int z){
2      if (y==0) return 1;
3      LL ret=Pow(x,y>>1,z); (ret*=ret)%=z;
4      if (y & 1) (ret*=x)%=z;
5      return ret;
6  }
7
8
9  void Prep(){
10     for (len=1, ci=0; len<=N+N; len<=<=1, ci++);
11     Wi[0]=1, Wi[1]=Pow(G,(Mo-1)/len,Mo);
12     for (int i=2; i<=len; i++) Wi[i]=(Wi[i-1]*Wi[1])% Mo;
13     for (int i=0; i<len; i++){
14         int tmp=0;
15         for (int j=i, c=0; c<ci; c++, j>>=1 ) tmp=(tmp <= 1)|= (j & 1);
16         Bel[i]=tmp;
17     }
18 }
19
20 void Dft(Arr &a,int sig)
21 {
22     for (int i=0; i<len; i++) tp[Bel[i]]=a[i];
23     for (int m=1; m<=len; m<=<=1){
24         int half=m>>1, bei=len/m;
25         for (int i=0; i<half; i++){
26             LL wi=(sig>0)?Wi[i*bei]:Wi[len-i*bei];
27             for (int j=i; j<len; j+=m){
28                 int u=tp[j],v=wi*LL(tp[j+half]) % Mo;
29                 tp[j]=(u+v) % Mo; tp[j+half]=(u-v+Mo)% Mo;
30             }
31         }
32     }

```

```

33         for (int i=0; i<len; i++) a[i]=tp[i];
34     }
35
36     void Mul(Arr &x,Arr &y,Arr &c,bool same)
37     {
38         if (!same){
39             for(int i=0; i<len; i++) a[i]=x[i], b[i]=y[i];
40             Dft(a,1),Dft(b,1);
41             for(int i=0; i<len; i++) a[i]=a[i]*111*b[i] % Mo;
42             Dft(a,-1);
43             for(int i=0; i<=M; i++) c[i]=a[i]*111*Rev % Mo;
44         } else
45         {
46             for(int i=0; i<len; i++) a[i]=x[i];
47             Dft(a,1);
48             for(int i=0; i<len; i++) a[i]=a[i]*111*a[i] % Mo;
49             Dft(a,-1);
50             for(int i=0; i<=M; i++) c[i]=a[i]*111*Rev % Mo;
51         }
52     }
53
54     Prep();
55     Ans[0]=1; Rev=Pow(len,Mo-2,Mo);
56     for(; K; K>>=1){
57         if (K & 1) Mul(Ans,F,Ans,0);
58         if (K > 1) Mul(F,F,F,1);
59     }
60
61     printf("%d\n",Ans[M]);

```

---

### 1.13 原根

---

```

1     vector<LL>fct;
2     bool check(LL x,LL g){
3         for(int i=0;i<fct.size();i++)
4             if(pw(g,(x-1)/fct[i],x)==1)
5                 return 0;
6         return 1;
7     }
8     LL findrt(LL x){
9         LL tmp=x-1;
10        for(int i=2;i*i<=tmp;i++){

```

```

11         if(tmp%i==0){
12             fct.push_back(i);
13             while(tmp%i==0)tmp/=i;
14         }
15     }if(tmp>1)fct.push_back(tmp);
16     // x is 1,2,4,p^n,2p^n
17     // x has phi(phi(x)) primitive roots
18     for(int i=2;i<int(1e9);i++)if(check(x,i))
19         return i;
20     return -1;
21 }
22 const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
23 bool check(long long n,int base) {
24     long long n2=n-1,res;
25     int s=0;
26     while(n2%2==0) n2>>=1,s++;
27     res=pw(base,n2,n);
28     if((res==1)|| (res==n-1)) return 1;
29     while(s--){
30         res=mul(res,res,n);
31         if(res==n-1) return 1;
32     }
33     return 0; // n is not a strong pseudo prime
34 }
35 bool isprime(const long long &n) {
36     if(n==2)
37         return true;
38     if(n<2 || n%2==0)
39         return false;
40     for(int i=0;i<12&&BASE[i]<n;i++){
41         if(!check(n,BASE[i]))
42             return false;
43     }
44     return true;
45 }

```

---

## 1.14 线性递推

---

```

1 //已知  $a_0, a_1, \dots, a_{m-1} \setminus \setminus$ 
2  $a_n = c_0 * a_{n-m} + \dots + c_{m-1} * a_{n-1} \setminus \setminus$ 
3 求  $a_n = v_0 * a_0 + v_1 * a_1 + \dots + v_{m-1} * a_{m-1} \setminus \setminus$ 
4

```

```

5 void linear_recurrence(long long n, int m, int a[], int c[], int p) {
6     long long v[M] = {1 % p}, u[M << 1], msk = !!n;
7     for(long long i(n); i > 1; i >>= 1) {
8         msk <<= 1;
9     }
10    for(long long x(0); msk; msk >>= 1, x <<= 1) {
11        fill_n(u, m << 1, 0);
12        int b(!!(n & msk));
13        x |= b;
14        if(x < m) {
15            u[x] = 1 % p;
16        }else {
17            for(int i(0); i < m; i++) {
18                for(int j(0), t(i + b); j < m; j++, t++) {
19                    u[t] = (u[t] + v[i] * v[j]) % p;
20                }
21            }
22            for(int i((m << 1) - 1); i >= m; i--) {
23                for(int j(0), t(i - m); j < m; j++, t++) {
24                    u[t] = (u[t] + c[j] * u[i]) % p;
25                }
26            }
27        }
28        copy(u, u + m, v);
29    }
30    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
31    for(int i(m); i < 2 * m; i++) {
32        a[i] = 0;
33        for(int j(0); j < m; j++) {
34            a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
35        }
36    }
37    for(int j(0); j < m; j++) {
38        b[j] = 0;
39        for(int i(0); i < m; i++) {
40            b[j] = (b[j] + v[i] * a[i + j]) % p;
41        }
42    }
43    for(int j(0); j < m; j++) {
44        a[j] = b[j];
45    }
46 }

```

---

## 1.15 线性筛

```
1 void sieve(){
2     f[1]=mu[1]=phi[1]=1;
3     for(int i=2;i<maxn;i++){
4         if(!minp[i]){
5             minp[i]=i;
6             minpw[i]=i;
7             mu[i]=-1;
8             phi[i]=i-1;
9             f[i]=i-1;
10            p[++p[0]]=i;//Case 1 prime
11        }
12        for(int j=1;j<=p[0]&&(LL)i*p[j]<maxn;j++){
13            minp[i*p[j]]=p[j];
14            if(i%p[j]==0){
15                //Case 2 not coprime
16                minpw[i*p[j]]=minpw[i]*p[j];
17                phi[i*p[j]]=phi[i]*p[j];
18                mu[i*p[j]]=0;
19                if(i==minpw[i]){
20                    f[i*p[j]]=i*p[j]-i;//Special Case for f(p^k)
21                }else{
22                    f[i*p[j]]=f[i/minpw[i]]*f[minpw[i]*p[j]];
23                }
24                break;
25            }else{
26                //Case 3 coprime
27                minpw[i*p[j]]=p[j];
28                f[i*p[j]]=f[i]*f[p[j]];
29                phi[i*p[j]]=phi[i]*(p[j]-1);
30                mu[i*p[j]]=-mu[i];
31            }
32        }
33    }
34 }
```

## 1.16 直线下整点个数

返回结果:

$$\sum_{0 \leq i < n} \lfloor \frac{a + b \cdot i}{m} \rfloor$$

使用条件:  $n, m > 0, a, b \geq 0$

时间复杂度:  $\mathcal{O}(n \log n)$

---

```
1 //calc \sum_{i=0}^{n-1} [(a+bi)/m]
2 // n,a,b,m >0
3 LL solve(LL n,LL a,LL b,LL m){
4     if(b==0)
5         return n*(a/m);
6     if(a>=m || b>=m)
7         return n*(a/m)+(n-1)*n/2*(b/m)+solve(n,a%m,b%m,m);
8     return solve((a+b*n)/m,(a+b*n)%m,m,b);
9 }
```

---

## 2 数值

### 2.1 高斯消元

---

```
1 void Gauss(){
2     int r,k;
3     for(int i=0;i<n;i++){
4         r=i;
5         for(int j=i+1;j<n;j++){
6             if(fabs(A[j][i])>fabs(A[r][i]))r=j;
7             if(r!=i)for(int j=0;j<=n;j++)swap(A[i][j],A[r][j]);
8             for(int k=i+1;k<n;k++){
9                 double f=A[k][i]/A[i][i];
10                for(int j=i;j<=n;j++)A[k][j]-=f*A[i][j];
11            }
12        }
13        for(int i=n-1;i>=0;i--){
14            for(int j=i+1;j<n;j++){
15                A[i][n]-=A[j][n]*A[i][j];
16            }
17            A[i][n]/=A[i][i];
18        }
19        for(int i=0;i<n-1;i++){
20            cout<<fixed<<setprecision(3)<<A[i][n]<<" ";
21        }
22    }
23    bool Gauss(){
24        for(int i=1;i<=n;i++){
25            int r=0;
26            for(int j=i;j<=m;j++){
```



```

26         if(a[j][i]){r=j;break;}
27         if(!r)return 0;
28         ans=max(ans,r);
29         swap(a[i],a[r]);
30         for(int j=i+1;j<=m;j++)
31             if(a[j][i])a[j]^=a[i];
32     }for(int i=n;i>=1;i--){
33         for(int j=i+1;j<=n;j++)if(a[i][j])
34             a[i][n+1]=a[i][n+1]^a[j][n+1];
35     }return 1;
36 }
37 LL Gauss(){
38     for(int i=0;i<n;i++)for(int j=0;j<n;j++)A[i][j]%m;
39     for(int i=0;i<n;i++)for(int j=0;j<n;j++)A[i][j]=(A[i][j]+m)%m;
40     LL ans=n%2?-1:1;
41     for(int i=0;i<n;i++){
42         for(int j=i+1;j<n;j++){
43             while(A[j][i]){
44                 LL t=A[i][i]/A[j][i];
45                 for(int k=0;k<n;k++)
46                     A[i][k]=(A[i][k]-A[j][k]*t%m+m)%m;
47                 swap(A[i],A[j]);
48                 ans=-ans;
49             }
50             ans=ans*A[i][i]%m;
51         }return (ans%m+m)%m;
52     }
53     int Gauss(){//求秩
54         int r,now=-1;
55         int ans=0;
56         for(int i = 0; i < n; i++){
57             r = now + 1;
58             for(int j = now + 1; j < m; j++)
59                 if(fabs(A[j][i]) > fabs(A[r][i]))
60                     r = j;
61             if (!sgn(A[r][i])) continue;
62             ans++;
63             now++;
64             if(r != now)
65                 for(int j = 0; j < n; j++)
66                     swap(A[r][j], A[now][j]);
67

```

```

68         for(int k = now + 1; k < m; k++){
69             double t = A[k][i] / A[now][i];
70             for(int j = 0; j < n; j++){
71                 A[k][j] -= t * A[now][j];
72             }
73         }
74     }
75     return ans;
76 }

```

---

## 2.2 快速傅立叶变换

返回结果:

$$c_i = \sum_{0 \leq j \leq i} a_j \cdot b_{i-j} \quad (0 \leq i < n)$$

时间复杂度:  $\mathcal{O}(n \log n)$

---

```

1  typedef complex<double> cp;
2  const double pi = acos(-1);
3  void FFT(vector<cp>&num, int len, int ty){
4      for(int i=1, j=0; i<len-1; i++){
5          for(int k=len; j^=k>=1, ~j&k);
6          if(i<j)
7              swap(num[i], num[j]);
8      }
9      for(int h=0; (1<<h)<len; h++){
10         int step=1<<h, step2=step<<1;
11         cp w0(cos(2.0*pi/step2), ty*sin(2.0*pi/step2));
12         for(int i=0; i<len; i+=step2){
13             cp w(1, 0);
14             for(int j=0; j<step; j++){
15                 cp &x=num[i+j+step];
16                 cp &y=num[i+j];
17                 cp d=w*x;
18                 x=y-d;
19                 y=y+d;
20                 w=w*w0;
21             }
22         }
23     }
24     if(ty== -1)
25         for(int i=0; i<len; i++)
26             num[i]=cp(num[i].real()/(double)len, num[i].imag());

```

```

27 }
28 vector<cp> mul(vector<cp>a,vector<cp>b){
29     int len=a.size()+b.size();
30     while((len&-len)!=len)len++;
31     while(a.size()<len)a.push_back(cp(0,0));
32     while(b.size()<len)b.push_back(cp(0,0));
33     FFT(a,len,1);
34     FFT(b,len,1);
35     vector<cp>ans(len);
36     for(int i=0;i<len;i++)
37         ans[i]=a[i]*b[i];
38     FFT(ans,len,-1);
39     return ans;
40 }

```

---

## 2.3 单纯形法求解线性规划

返回结果:

$$\max\{c_{1 \times m} \cdot x_{m \times 1} \mid x_{m \times 1} \geq 0_{m \times 1}, a_{n \times m} \cdot x_{m \times 1} \leq b_{n \times 1}\}$$

---

```

1 namespace LP{
2     const int maxn=233;
3     double a[maxn][maxn];
4     int Ans[maxn],pt[maxn];
5     int n,m;
6     void pivot(int l,int i){
7         double t;
8         swap(Ans[l+n],Ans[i]);
9         t=-a[l][i];
10        a[l][i]=-1;
11        for(int j=0;j<=n;j++)a[l][j]/=t;
12        for(int j=0;j<=m;j++){
13            if(a[j][i]&&j!=1){
14                t=a[j][i];
15                a[j][i]=0;
16                for(int k=0;k<=n;k++)a[j][k]+=t*a[l][k];
17            }
18        }
19    }
20    vector<double> solve(vector<vector<double> >A,vector<double>B,vector<double>C){
21        n=C.size();
22        m=B.size();
23        for(int i=0;i<C.size();i++)

```

```

24         a[0][i+1]=C[i];
25     for(int i=0;i<B.size();i++)
26         a[i+1][0]=B[i];
27
28     for(int i=0;i<m;i++)
29         for(int j=0;j<n;j++)
30             a[i+1][j+1]=-A[i][j];
31
32     for(int i=1;i<=n;i++)Ans[i]=i;
33
34     double t;
35     for(;;){
36         int l=0;t=-eps;
37         for(int j=1;j<=m;j++)if(a[j][0]<t)t=a[l=j][0];
38         if(!l)break;
39         int i=0;
40         for(int j=1;j<=n;j++)if(a[l][j]>eps){i=j;break;}
41         if(!i){
42             puts("Infeasible");
43             return vector<double>();
44         }
45         pivot(l,i);
46     }
47     for(;;){
48         int i=0;t=eps;
49         for(int j=1;j<=n;j++)if(a[0][j]>t)t=a[0][i=j];
50         if(!i)break;
51         int l=0;
52         t=1e30;
53         for(int j=1;j<=m;j++)if(a[j][i]<-eps){
54             double tmp;
55             tmp=-a[j][0]/a[j][i];
56             if(t>tmp)t=tmp,l=j;
57         }
58         if(!l){
59             puts("Unbounded");
60             return vector<double>();
61         }
62         pivot(l,i);
63     }
64     vector<double>x;
65     for(int i=n+1;i<=n+m;i++)pt[Ans[i]]=i-n;

```

```

66         for(int i=1;i<=n;i++)x.push_back(pt[i]?a[pt[i]][0]:0);
67         return x;
68     }
69 }

```

---

## 2.4 自适应辛普森

```

1  double area(const double &left, const double &right) {
2      double mid = (left + right) / 2;
3      return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
4  }
5
6  double simpson(const double &left, const double &right,
7                const double &eps, const double &area_sum) {
8      double mid = (left + right) / 2;
9      double area_left = area(left, mid);
10     double area_right = area(mid, right);
11     double area_total = area_left + area_right;
12     if (std::abs(area_total - area_sum) < 15 * eps) {
13         return area_total + (area_total - area_sum) / 15;
14     }
15     return simpson(left, mid, eps / 2, area_left)
16         + simpson(mid, right, eps / 2, area_right);
17 }
18
19 double simpson(const double &left, const double &right, const double &eps) {
20     return simpson(left, right, eps, area(left, right));
21 }

```

---

## 2.5 多项式求根

```

1  const double eps=1e-12;
2  double a[10][10];
3  typedef vector<double> vd;
4  int sgn(double x) { return x < -eps ? -1 : x > eps; }
5  double mypow(double x,int num){
6      double ans=1.0;
7      for(int i=1;i<=num;++i)ans*=x;
8      return ans;
9  }
10 double f(int n,double x){

```

```

11     double ans=0;
12     for(int i=n;i>=0;--i)ans+=a[n][i]*mypow(x,i);
13     return ans;
14 }
15 double getRoot(int n,double l,double r){
16     if(sgn(f(n,l))==0)return l;
17     if(sgn(f(n,r))==0)return r;
18     double temp;
19     if(sgn(f(n,l))>0)temp=-1;else temp=1;
20     double m;
21     for(int i=1;i<=10000;++i){
22         m=(l+r)/2;
23         double mid=f(n,m);
24         if(sgn(mid)==0){
25             return m;
26         }
27         if(mid*temp<0)l=m;else r=m;
28     }
29     return (l+r)/2;
30 }
31 vd did(int n){
32     vd ret;
33     if(n==1){
34         ret.push_back(-1e10);
35         ret.push_back(-a[n][0]/a[n][1]);
36         ret.push_back(1e10);
37         return ret;
38     }
39     vd mid=did(n-1);
40     ret.push_back(-1e10);
41     for(int i=0;i+1<mid.size();++i){
42         int t1=sgn(f(n,mid[i])),t2=sgn(f(n,mid[i+1]));
43         if(t1*t2>0)continue;
44         ret.push_back(getRoot(n,mid[i],mid[i+1]));
45     }
46     ret.push_back(1e10);
47     return ret;
48 }
49 int main(){
50     int n; scanf("%d",&n);
51     for(int i=n;i>=0;--i){
52         scanf("%lf",&a[n][i]);

```

```

53     }
54     for(int i=n-1;i>=0;--i)
55         for(int j=0;j<=i;++j)a[i][j]=a[i+1][j+1]*(j+1);
56     vd ans=did(n);
57     sort(ans.begin(),ans.end());
58     for(int i=1;i+1<ans.size();++i)printf("%.10f\n",ans[i]);
59     return 0;
60 }

```

---

## 3 数据结构

### 3.1 平衡的二叉查找树

#### 3.1.1 Treap

---

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn=1e5+5;
4  #define sz(x) (x?x->siz:0)
5  struct Treap{
6      struct node{
7          int key,val;
8          int siz,s;
9          node *c[2];
10         node(int v=0){
11             val=v;
12             key=rand();
13             siz=1,s=1;
14             c[0]=c[1]=0;
15         }
16         void rz(){siz=s;if(c[0])siz+=c[0]->siz;if(c[1])siz+=c[1]->siz;}
17     }pool[maxn],*cur,*root;
18     Treap(){cur=pool;}
19     node* newnode(int val){return *cur=node(val),cur++;}
20     void rot(node *&t,int d){
21         if(!t->c[d])t=t->c[!d];
22         else{
23             node *p=t->c[d];t->c[d]=p->c[!d];
24             p->c[!d]=t;t->rz();p->rz();t=p;
25         }
26     }
27     void insert(node *&t,int x){

```

```

28         if(!t){t=newnode(x);return;}
29         if(t->val==x){t->s++;t->siz++;return;}
30         insert(t->c[x>t->val],x);
31         if(t->key<t->c[x>t->val]->key)
32             rot(t,x>t->val);
33         else t->rz();
34     }
35     void del(node *&t,int x){
36         if(!t)return;
37         if(t->val==x){
38             if(t->s>1){t->s--;t->siz--;return;}
39             if(!t->c[0]||!t->c[1]){
40                 if(!t->c[0])t=t->c[1];
41                 else t=t->c[0];
42                 return;
43             }
44             int d=t->c[0]->key<t->c[1]->key;
45             rot(t,d);
46             del(t,x);
47             return;
48         }
49         del(t->c[x>t->val],x);
50         t->rz();
51     }
52     int pre(node *t,int x){
53         if(!t)return INT_MIN;
54         int ans=pre(t->c[x>t->val],x);
55         if(t->val<x)ans=max(ans,t->val);
56         return ans;
57     }
58     int nxt(node *t,int x){
59         if(!t)return INT_MAX;
60         int ans=nxt(t->c[x>=t->val],x);
61         if(t->val>x)ans=min(ans,t->val);
62         return ans;
63     }
64     int rank(node *t,int x){
65         if(!t)return 0;
66         if(t->val==x)return sz(t->c[0]);
67         if(t->val<x)return sz(t->c[0])+t->s+rank(t->c[1],x);
68         if(t->val>x)return rank(t->c[0],x);
69     }

```



```

70     int kth(node *t, int x){
71         if(sz(t->c[0])>=x) return kth(t->c[0], x);
72         if(sz(t->c[0])+t->s>=x) return t->val;
73         return kth(t->c[1], x-t->s-sz(t->c[0]));
74     }
75     void deb(node *t){
76         if(!t) return;
77         deb(t->c[0]);
78         printf("%d ", t->val);
79         deb(t->c[1]);
80     }
81     void insert(int x){insert(root, x);}
82     void del(int x){del(root, x);}
83     int pre(int x){return pre(root, x);}
84     int nxt(int x){return nxt(root, x);}
85     int rank(int x){return rank(root, x);}
86     int kth(int x){return kth(root, x);}
87     void deb(){deb(root); puts("");}
88 }T;
89 int main(){
90     srand(12121);
91     int m;
92     scanf("%d", &m);
93     while(m--){
94         int opt, x;
95         scanf("%d", &opt);
96         switch(opt){
97             case 1:
98                 scanf("%d", &x);
99                 T.insert(x);
100                 break;
101             case 2:
102                 scanf("%d", &x);
103                 T.del(x);
104                 break;
105             case 3:
106                 scanf("%d", &x);
107                 printf("%d\n", T.rank(x)+1);
108                 break;
109             case 4:
110                 scanf("%d", &x);
111                 printf("%d\n", T.kth(x));

```

```

112         break;
113     case 5:
114         scanf("%d",&x);
115         printf("%d\n",T.pre(x));
116         break;
117     case 6:
118         scanf("%d",&x);
119         printf("%d\n",T.nxt(x));
120         break;
121     }
122 }
123     return 0;
124 }

```

---

### 3.1.2 Splay

---

```

1  void Rotate(int x, int c){
2      int y = T[x].c[c];
3      int z = T[y].c[1 - c];
4
5      if (T[x].fa){
6          if (T[T[x].fa].c[0] == x) T[T[x].fa].c[0] = y;
7          else T[T[x].fa].c[1] = y;
8      }
9
10     T[z].fa = x; T[x].c[c] = z;
11     T[y].fa = T[x].fa; T[x].fa = y; T[y].c[1 - c] = x;
12
13     Update(x);
14     Update(y);
15 }
16
17 int stack[M], fx[M];
18
19 void Splay(int x, int fa){
20     int top = 0;
21     for (int u = x; u != fa; u = T[u].fa)
22         stack[++top] = u;
23     for (int i = 2; i <= top; i++)
24         if (T[stack[i]].c[0] == stack[i - 1]) fx[i] = 0;
25         else fx[i] = 1;
26

```

```

27     for (int i = 2; i <= top; i += 2){
28         if (i == top) Rotate(stack[i], fx[i]);
29         else {
30             if (fx[i] == fx[i + 1]){
31                 Rotate(stack[i + 1], fx[i + 1]);
32                 Rotate(stack[i], fx[i]);
33             } else {
34                 Rotate(stack[i], fx[i]);
35                 Rotate(stack[i + 1], fx[i + 1]);
36             }
37         }
38     }
39
40     if (fa == 0) Root = x;
41 }

```

---

## 3.2 坚固的数据结构

### 3.2.1 坚固的平衡树

---

```

1  #define sz(x) (x?x->siz:0)
2  struct node{
3      int siz,key;
4      LL val,sum;
5      LL mu,a,d;
6      node *c[2],*f;
7      void split(int ned,node *&p,node *&q);
8      node* rz(){
9          sum=val;siz=1;
10         if(c[0])sum+=c[0]->sum,siz+=c[0]->siz;
11         if(c[1])sum+=c[1]->sum,siz+=c[1]->siz;
12         return this;
13     }
14     void make(LL _mu,LL _a,LL _d){
15         sum=sum*_mu+_a*siz+_d*siz*(siz-1)/2;
16         val=val*_mu+_a+_d*sz(c[0]);
17         mu*=_mu;a=_a*_mu+_a;d=_d*_mu+_d;
18     }
19     void pd(){
20         if(mu==1&&a==0&&d==0)return;
21         if(c[0])c[0]->make(mu,a,d);
22         if(c[1])c[1]->make(mu,a+d*_d*sz(c[0]),d);

```

```

23         mu=1;a=d=0;
24     }
25     node(){mu=1;}
26 }nd[maxn*2],*root;
27 node *merge(node *p,node *q){
28     if(!p||!q)return p?p->rz():(q?q->rz():0);
29     p->pd();q->pd();
30     if(p->key<q->key){
31         p->c[1]=merge(p->c[1],q);
32         return p->rz();
33     }else{
34         q->c[0]=merge(p,q->c[0]);
35         return q->rz();
36     }
37 }
38 void node::split(int ned,node *&p,node *&q){
39     if(!ned){p=0;q=this;return;}
40     if(ned==siz){p=this;q=0;return;}
41     pd();
42     if(sz(c[0])>=ned){
43         c[0]->split(ned,p,q);c[0]=0;rz();
44         q=merge(q,this);
45     }else{
46         c[1]->split(ned-sz(c[0])-1,p,q);c[1]=0;rz();
47         p=merge(this,p);
48     }
49 }
50 int main(){
51     for(int i=1;i<=n;i++){
52         nd[i].val=in();
53         nd[i].key=rand();
54         nd[i].rz();
55         root=merge(root,nd+i);
56     }
57 }

```

---

### 3.2.2 坚固的字符串

#### 1. ext 库中的 rope

---

```

1  #include <ext/rope>
2
3  using __gnu_cxx::crope;

```

```

4  using __gnu_cxx::rope;
5
6  crope a, b;
7
8  int main(void) {
9      a = b.substr(pos, len);    // [pos, pos + len)
10     a = b.substr(pos);         // [pos, pos]
11     b.c_str();                 // might lead to memory leaks
12     b.delete_c_str();          // delete the c_str that created before
13     a.insert(pos, text);        // insert text before position pos
14     a.erase(pos, len);         // erase [pos, pos + len)
15 }

```

---

## 2. 可持久化平衡树实现的 rope

---

```

1  class Rope {
2  private:
3      class Node {
4      public:
5          Node *left, *right;
6          int size;
7          char key;
8
9          Node(char key = 0, Node *left = NULL, Node *right = NULL)
10             : key(key), left(left), right(right) {
11                 update();
12             }
13
14             void update() {
15                 size = (left ? left->size : 0) + 1 + (right ? right->size : 0);
16             }
17
18             std::string to_string() {
19                 return (left ? left->to_string() : "") + key
20                     + (right ? right->to_string() : "");
21             }
22         };
23
24         bool random(int a, int b) {
25             return rand() % (a + b) < a;
26         }
27
28         Node* merge(Node *x, Node *y) {

```

```

29     if (!x) {
30         return y;
31     }
32     if (!y) {
33         return x;
34     }
35     if (random(x->size, y->size)) {
36         return new Node(x->key, x->left, merge(x->right, y));
37     } else {
38         return new Node(y->key, merge(x, y->left), y->right);
39     }
40 }
41
42 std::pair<Node*, Node*> split(Node *x, int size) {
43     if (!x) {
44         return std::make_pair<Node*, Node*>(NULL, NULL);
45     }
46     if (size == 0) {
47         return std::make_pair<Node*, Node*>(NULL, x);
48     }
49     if (size > x->size) {
50         return std::make_pair<Node*, Node*>(x, NULL);
51     }
52     if (x->left && size <= x->left->size) {
53         std::pair<Node*, Node*> part =
54             split(x->left, size);
55         return std::make_pair(part.first, new Node(x->key, part.second, x->right));
56     } else {
57         std::pair<Node*, Node*> part =
58             split(x->right, size - (x->left ? x->left->size : 0) - 1);
59         return std::make_pair(new Node(x->key, x->left, part.first), part.second);
60     }
61 }
62
63 Node* build(const std::string &text, int left, int right) {
64     if (left > right) {
65         return NULL;
66     }
67     int mid = left + right >> 1;
68     return new Node(text[mid],
69                     build(text, left, mid - 1),
70                     build(text, mid + 1, right));

```

```

71     }
72
73     public:
74         Node *root;
75
76         Rope() {
77             root = NULL;
78         }
79
80         Rope(const std::string &text) {
81             root = build(text, 0, (int)text.length() - 1);
82         }
83
84         Rope(const Rope &other) {
85             root = other.root;
86         }
87
88         Rope& operator = (const Rope &other) {
89             if (this == &other) {
90                 return *this;
91             }
92             root = other.root;
93             return *this;
94         }
95
96         int size() {
97             return root ? root->size : 0;
98         }
99
100        void insert(int pos, const std::string &text) {
101            if (pos < 0 || pos > size()) {
102                throw "Out of range";
103            }
104            std::pair<Node*, Node*> part = split(root, pos);
105            root = merge(merge(part.first, build(text, 0, (int)text.length() - 1)),
106                        part.second);
107        }
108
109        void erase(int left, int right) {
110            if (left < 0 || left >= size() ||
111                right < 1 || right > size()) {
112                throw "Out of range";

```

```

113     }
114     if (left >= right) {
115         return;
116     }
117     std::pair<Node*, Node*> part = split(root, left);
118     root = merge(part.first, split(part.second, right - left).second);
119 }
120
121 std::string substr(int left, int right) {
122     if (left < 0 || left >= size() ||
123         right < 1 || right > size()) {
124         throw "Out of range";
125     }
126     if (left >= right) {
127         return "";
128     }
129     return split(split(root, left).second, right - left).first->to_string();
130 }
131
132 void copy(int left, int right, int pos) {
133     if (left < 0 || left >= size() ||
134         right < 1 || right > size() ||
135         pos < 0 || pos > size()) {
136         throw "Out of range";
137     }
138     if (left >= right) {
139         return;
140     }
141     std::pair<Node*, Node*> part = split(root, pos);
142     root = merge(merge(part.first,
143                       split(split(root, left).second, right - left).first),
144                 part.second);
145 }
146 };

```

---

### 3.2.3 坚固的左偏树

```

1 int Merge(int x, int y){
2     if (x == 0 || y == 0) return x + y;
3     if (Heap[x].Key < Heap[y].Key) swap(x, y);
4     Heap[x].Ri = Merge(Heap[x].Ri, y);
5     if (Heap[Heap[x].Le].Dis < Heap[Heap[x].Ri].Dis) swap(Heap[x].Le, Heap[x].Ri);

```



```

6   if (Heap[x].Ri == 0) Heap[x].Dis = 0;
7   else Heap[x].Dis = Heap[Heap[x].Ri].Dis + 1;
8   return x;
9 }
10
11 for (int i = 0; i <= n; i++){
12     Heap[i].Le = Heap[i].Ri = 0;
13     Heap[i].Dis = 0;
14     Heap[i].Key = Cost[i];
15 }
16 Heap[0].Dis = -1;

```

---

### 3.2.4 不坚固的斜堆

```

1  struct node;
2  node *Null,*root[maxn];
3  struct node{
4      node* c[2];
5      int val,ind;
6      node(int _val=0,int _ind=0){
7          val=_val;c[0]=c[1]=Null;ind=_ind;
8      }
9  };
10 node* merge(node *p,node *q){
11     if(p==Null)return q;
12     if(q==Null)return p;
13     if(p->val>q->val)swap(p,q);
14     p->c[1]=merge(p->c[1],q);
15     swap(p->c[0],p->c[1]);
16     return p;
17 }
18
19 Null=new node(0);
20 Null->c[0]=Null->c[1]=Null;

```

---

## 3.3 树上的魔术师

### 3.3.1 轻重树链剖分 (zky)

```

1  vector<int>G[maxn];
2  int fa[maxn],top[maxn],siz[maxn],son[maxn],mp[maxn],z,dep[maxn];
3  void dfs(int u){

```

```

4      siz[u]=1;
5      for(int i=0;i<G[u].size();i++){
6          int v=G[u][i];
7          if(v!=fa[u]){
8              fa[v]=u;dep[v]=dep[u]+1;
9              dfs(v);
10             siz[u]+=siz[v];
11             if(siz[son[u]]<siz[v])son[u]=v;
12         }
13     }
14 }
15 void build(int u,int tp){
16     top[u]=tp;mp[u]=++z;
17     if(son[u])build(son[u],tp);
18     for(int v,i=0;i<G[u].size();i++)if((v=G[u][i])!=son[u]&&v!=fa[u])build(v,v);
19 }

```

---

### 3.3.2 轻重树链剖分 (lyx)

---

```

1 void Prep(int x){
2     dep[x] = dep[fa[x]] + 1;
3     size[x] = 1;
4     son[x] = 0;
5     for (int i = g[x]; i; i = nxt[i]){
6         int y = adj[i];
7         if (y == fa[x]) continue;
8         fa[y] = x;
9         Prep(y);
10        size[x] += size[y];
11        if (size[y] > size[son[x]]) son[x] = y;
12    }
13 }
14 void Dfs(int x){
15     dfn[x] = ++dfc;
16     if (son[x] != 0){
17         top[son[x]] = top[x];
18         Dfs(son[x]);
19     }
20     for (int i = g[x]; i; i = nxt[i]){
21         int y = adj[i];
22         if (y != fa[x] && y != son[x]){
23             top[y] = y;

```

```

24         Dfs(y);
25     }
26     if (y != fa[x]){
27         Bel[(i + 1) >> 1] = dfn[y];
28         val[dfn[y]] = len[i];
29     }
30 }
31 }
32 int Ask(int x, int y){
33     int Ret = -1000000001;
34     while (top[x] != top[y]){
35         if (dep[top[y]] > dep[top[x]]) swap(x, y);
36         Ret = max(Ret, Query(1, 1, n, dfn[top[x]], dfn[x]));
37         x = fa[top[x]];
38     }
39     if (dep[y] > dep[x]) swap(x, y);
40     if (x != y)
41         Ret = max(Ret, Query(1, 1, n, dfn[y] + 1, dfn[x]));
42     return Ret;
43 }
44 //Hints : Ask 部分具体的求值或者修改要稍作变动

```

---

### 3.3.3 Link Cut Tree(zky)

---

```

1 struct LCT{
2     struct node{
3         bool rev;
4         int mx,val;
5         node *f,*c[2];
6         bool d(){return this==f->c[1];}
7         bool rt(){return !f|| (f->c[0]!=this&&f->c[1]!=this);}
8         void sets(node *x,int d){pd();if(x)x->f=this;c[d]=x;rz();}
9         void makerv(){rev^=1;swap(c[0],c[1]);}
10        void pd(){
11            if(rev){
12                if(c[0])c[0]->makerv();
13                if(c[1])c[1]->makerv();
14                rev=0;
15            }
16        }
17        void rz(){
18            mx=val;

```

```

19         if(c[0])mx=max(mx,c[0]->mx);
20         if(c[1])mx=max(mx,c[1]->mx);
21     }
22 }nd[int(1e4)+1];
23 void rot(node *x){
24     node *y=x->f;if(!y->rt())y->f->pd();
25     y->pd();x->pd();bool d=x->d();
26     y->sets(x->c[!d],d);
27     if(y->rt())x->f=y->f;
28     else y->f->sets(x,y->d());
29     x->sets(y,!d);
30 }
31 void splay(node *x){
32     while(!x->rt())
33         if(x->f->rt())rot(x);
34         else if(x->d()==x->f->d())rot(x->f),rot(x);
35         else rot(x),rot(x);
36 }
37 node* access(node *x){
38     node *y=0;
39     for(;x;x=x->f){
40         splay(x);
41         x->sets(y,1);y=x;
42     }return y;
43 }
44 void makert(node *x){
45     access(x)->makerv();
46     splay(x);
47 }
48 void link(node *x,node *y){
49     makert(x);
50     x->f=y;
51     access(x);
52 }
53 void cut(node *x,node *y){
54     makert(x);access(y);splay(y);
55     y->c[0]=x->f=0;
56     y->rz();
57 }
58 void link(int x,int y){link(nd+x,nd+y);}
59 void cut(int x,int y){cut(nd+x,nd+y);}
60 }T;

```

---

### 3.3.4 Link Cut Tree(lyx)

---

```
1  struct node{
2      bool Rev;
3      int c[2], fa;
4  }T[N];
5
6
7  inline void Rev(int x){
8      if (!x) return;
9      swap(T[x].c[0], T[x].c[1]);
10     T[x].Rev ^= 1;
11 }
12
13 inline void Lazy_Down(int x){
14     if (!x) return;
15     if (T[x].Rev) Rev(T[x].c[0]), Rev(T[x].c[1]), T[x].Rev = 0;
16 }
17
18 void Rotate(int x, int c){
19     int y = T[x].c[c];
20     int z = T[y].c[1 - c];
21
22     if (T[x].fa){
23         if (T[T[x].fa].c[0] == x) T[T[x].fa].c[0] = y;
24         else T[T[x].fa].c[1] = y;
25     }
26
27     T[z].fa = x; T[x].c[c] = z;
28     T[y].fa = T[x].fa; T[x].fa = y; T[y].c[1 - c] = x;
29
30     //Update(x);
31     //Update(y);
32 }
33
34 int stack[N], fx[N];
35
36 void Splay(int x){
37     int top = 0;
38     for (int u = x; u; u = T[u].fa)
39         stack[++top] = u;
40     for (int i = top; i >= 1; i--)
```

```

41         Lazy_Down(stack[i]);
42     for (int i = 2; i <= top; i++)
43         if (T[stack[i]].c[0] == stack[i - 1]) fx[i] = 0;
44         else fx[i] = 1;
45
46     for (int i = 2; i <= top; i += 2){
47         if (i == top) Rotate(stack[i], fx[i]);
48         else {
49             if (fx[i] == fx[i + 1]){
50                 Rotate(stack[i + 1], fx[i + 1]);
51                 Rotate(stack[i], fx[i]);
52             } else {
53                 Rotate(stack[i], fx[i]);
54                 Rotate(stack[i + 1], fx[i + 1]);
55             }
56         }
57     }
58
59     if (x != stack[top]) Par[x] = Par[stack[top]], Par[stack[top]] = 0;
60 }
61
62 inline int Access(int u){
63     int Nxt = 0;
64     while (u){
65         Splay(u);
66         if (T[u].c[1]){
67             T[T[u].c[1]].fa = 0;
68             Par[T[u].c[1]] = u;
69         }
70         T[u].c[1] = Nxt;
71         if (Nxt){
72             T[Nxt].fa = u;
73             Par[Nxt] = 0;
74         }
75         //Update(u)
76         Nxt = u;
77         u = Par[u];
78     }
79     return Nxt;
80 }
81
82 inline void Root(int u){

```

```

83     Access(u);
84     Splay(u);
85     Rev(u);
86 }
87
88 inline void Link(int u, int v){
89     Root(u);
90     Par[u] = v;
91 }
92
93 inline void Cut(int u, int v){
94     Access(u);
95     Splay(v);
96     if (Par[v] != u){
97         swap(u, v);
98         Access(u);
99         Splay(v);
100     }
101     Par[v] = 0;
102 }
103
104 inline int Find_Root(int x){
105     Access(x);
106     Splay(x);
107
108     int y = x;
109     while (T[y].c[0]){
110         Lazy_Down(y);
111         y = T[y].c[0];
112     }
113     return y;
114 }

```

---

### 3.3.5 AAA Tree

---

```

1  #define rep(i,a,n) for(int i=a;i<n;i++)
2  int n,m;
3  struct info{
4      int mx,mn,sum,sz;
5      info(){ }
6      info(int mx,int mn,int sum,int sz):
7          mx(mx),mn(mn),sum(sum),sz(sz){ }

```

```

8     void deb(){printf("sum:%d size:%d",(int)sum,sz);}
9 };
10 struct flag{
11     int mul,add;
12     flag(){mul=1;}
13     flag(int mul,int add):
14         mul(mul),add(add){}
15     bool empty(){return mul==1&&add==0;}
16 };
17 info operator+(const info &a,const flag &b) {
18     return a.sz?info(a.mx*b.mul+b.add,a.mn*b.mul+b.add,a.sum*b.mul+b.add*a.sz,a.sz):a;
19 }
20 info operator+(const info &a,const info &b) {
21     return info(max(a.mx,b.mx),min(a.mn,b.mn),a.sum+b.sum,a.sz+b.sz);
22 }
23 flag operator+(const flag &a,const flag &b) {
24     return flag(a.mul*b.mul,a.add*b.mul+b.add);
25 }
26 struct node{
27     node *c[4],*f;
28     flag Cha,All;
29     info cha,sub,all;
30     bool rev,inr;
31     int val;
32     void makerev(){rev^=1;swap(c[0],c[1]);}
33     void makec(const flag &a){
34         Cha=Cha+a;cha=cha+a;val=val*a.mul+a.add;
35         all=cha+sub;
36     }
37     void makes(const flag &a,bool _=1){
38         All=All+a;all=all+a;sub=sub+a;
39         if(_)makec(a);
40     }
41     void rz(){
42         cha=all=sub=info(-(1<<30),1<<30,0,0);
43         if(!inr)all=cha=info(val,val,val,1);
44         rep(i,0,2)if(c[i])cha=cha+c[i]->cha,sub=sub+c[i]->sub;
45         rep(i,0,4)if(c[i])all=all+c[i]->all;
46         rep(i,2,4)if(c[i])sub=sub+c[i]->all;
47     }
48     void pd(){
49         if(rev){

```



```

50         if(c[0])c[0]->makerev();
51         if(c[1])c[1]->makerev();
52         rev=0;
53     }
54     if(!All.empty()){
55         rep(i,0,4)if(c[i])c[i]->makes(All,i>=2);
56         All=flag(1,0);
57     }
58     if(!Cha.empty()){
59         rep(i,0,2)if(c[i])c[i]->makec(Cha);
60         Cha=flag(1,0);
61     }
62
63 }
64 node *C(int i){if(c[i])c[i]->pd();return c[i];}
65 bool d(int ty){return f->c[ty+1]==this;}
66 int D(){rep(i,0,4)if(f->c[i]==this)return i;}
67 void sets(node *x,int d){if(x)x->f=this;c[d]=x;}
68 bool rt(int ty){
69     if(ty==0)return !f||(f->c[0]!=this&&f->c[1]!=this);
70     else return !f||!f->inr||!inr;
71 }
72 }nd[maxn*2],*cur=nd+maxn,*pool[maxn],**Cur=pool;
73 int _cnt;
74 node *newnode(){
75     _cnt++;
76     node *x=(Cur==pool)?cur++:*(--Cur);
77     rep(i,0,4)x->c[i]=0;x->f=0;
78     x->All=x->Cha=flag(1,0);
79     x->all=x->cha=info(-(1<<30),(1<<30),0,0);
80     x->inr=1;x->rev=0;x->val=0;
81     return x;
82 }
83 void dele(node *x){*(Cur++)=x;}
84 void rot(node *x,int ty){
85     node *p=x->f;int d=x->d(ty);
86     if(!p->f)x->f=0;else p->f->sets(x,p->D());
87     p->sets(x->c[!d+ty],d+ty);x->sets(p,!d+ty);p->rz();
88 }
89 void splay(node *x,int ty=0){
90     while(!x->rt(ty)){
91         if(x->f->rt(ty))rot(x,ty);

```

```

92         else if(x->d(ty)==x->f->d(ty))rot(x->f,ty),rot(x,ty);
93         else rot(x,ty),rot(x,ty);
94     }x->rz();
95 }
96 void add(node *u,node *w){
97     w->pd();
98     rep(i,2,4)if(!w->c[i]){w->sets(u,i);return;}
99     node *x=newnode(),*v;
100     for(v=w;v->c[2]->inr;v=v->C(2));
101     x->sets(v->c[2],2);x->sets(u,3);
102     v->sets(x,2);splay(x,2);
103 }
104 void del(node *w){
105     if(w->f->inr){
106         w->f->f->sets(w->f->c[5-w->D()],w->f->D());
107         dele(w->f);splay(w->f->f,2);
108     }else w->f->sets(0,w->D());
109     w->f=0;
110 }
111 void access(node *w){
112     static node *sta[maxn];
113     static int top=0;
114     node *v=w,*u;
115     for(u=w;u;u=u->f)sta[top++]=u;
116     while(top)sta[--top]->pd();
117     splay(w);
118     if(w->c[1])u=w->c[1],w->c[1]=0,add(u,w),w->rz();
119     while(w->f){
120         for(u=w->f;u->inr;u=u->f);
121         splay(u);
122         if(u->c[1])w->f->sets(u->c[1],w->D()),splay(w->f,2);
123         else del(w);
124         u->sets(w,1);
125         (w=u)->rz();
126     }splay(v);
127 }
128 void makert(node *x){
129     access(x);x->makerev();
130 }
131 node *findp(node *u){
132     access(u);u=u->C(0);
133     while(u&&u->c[1])u=u->C(1);

```

```

134     return u;
135 }
136 node *findr(node *u){for(;u->f;u=u->f);return u;}
137 node* cut(node *u){
138     node *v=findp(u);
139     if(v)access(v),del(u),v->rz();
140     return v;
141 }
142 void link(node *u,node *v) {
143     node* p=cut(u);
144     if(findr(u)!=findr(v))p=v;
145     if(p)access(p),add(u,p),p->rz();
146 }
147 int main(){
148     // freopen("bzoj3153.in","r",stdin);
149     n=getint();m=getint();
150     static int _u[maxn],_v[maxn];
151     rep(i,1,n)_u[i]=getint(),_v[i]=getint();
152     rep(i,1,n+1){
153         nd[i].val=getint();
154         nd[i].rz();
155     }
156     rep(i,1,n)makert(nd+_u[i]),link(nd+_u[i],nd+_v[i]);
157     int root=getint();
158     makert(nd+root);
159     // deb();
160     int x,y,z;
161     node *u,*v;
162     while(m--){
163         int k=getint();x=getint();
164         u=nd+x;
165         if(k==0 || k==3 || k==4 || k==5 || k==11){
166             access(u);
167             if(k==3 || k==4 || k==11){
168                 int ans=u->val;
169                 rep(i,2,4)if(u->c[i]){
170                     info res=u->c[i]->all;
171                     if(k==3) ans=min(ans,res.mn);
172                     else if(k==4) ans=max(ans,res.mx);
173                     else if(k==11) ans+=res.sum;
174                 }printf("%d\n",ans);
175             }else{

```

```

176         y=getint();
177         flag fg(k==5,y);
178         u->val=u->val*fg.mul+fg.add;
179         rep(i,2,4)if(u->c[i])u->c[i]->makes(fg);
180         u->rz();
181     }
182 }else if(k==2 || k==6 || k==7 || k==8 || k==10){
183     y=getint();
184     makert(u),access(nd+y),splay(u);
185     if (k==7 || k==8 || k==10) {
186         info ans=u->cha;
187         if (k==7) printf("%d\n",ans.mn);
188         else if (k==8) printf("%d\n",ans.mx);
189         else printf("%d\n",ans.sum);
190     }else u->makec(flag(k==6,getint()));
191     makert(nd+root);
192 }else if(k==9)link(u,nd+getint());
193 else if(k==1)makert(u),root=x;
194 }
195 return 0;
196 }

```

---

### 3.4 ST

---

```

1  for (int i = 1; i <= n; i++)
2      Log[i] = int(log2(i));
3
4  for (int i = 1; i <= n; i++)
5      Rmq[i][0] = i;
6
7  for (int k = 1; (1 << k) <= n; k++)
8      for (int i = 1; i + (1 << k) - 1 <= n; i++){
9          int x = Rmq[i][k - 1], y = Rmq[i + (1 << (k - 1))][k - 1];
10         if (a[x] < a[y])
11             Rmq[i][k] = x;
12         else
13             Rmq[i][k] = y;
14     }
15
16 int Smallest(int l, int r){
17     int k = Log[r - l + 1];
18

```

```

19     int x = Rmq[l][k];
20     int y = Rmq[r - (1 << k) + 1][k];
21
22     if (a[x] < a[y]) return x;
23     else return y;
24 }

```

---

### 3.5 可持久化线段树

---

```

1  struct node1 {
2      int L, R, Lson, Rson, Sum;
3  } tree[N * 40];
4  int root[N], a[N], b[N];
5  int tot, n, m;
6  int Real[N];
7  int Same(int x) {
8      ++tot;
9      tree[tot] = tree[x];
10     return tot;
11 }
12 int build(int L, int R) {
13     ++tot;
14     tree[tot].L = L;
15     tree[tot].R = R;
16     tree[tot].Lson = tree[tot].Rson = tree[tot].Sum = 0;
17     if (L == R) return tot;
18     int s = tot;
19     int mid = (L + R) >> 1;
20     tree[s].Lson = build(L, mid);
21     tree[s].Rson = build(mid + 1, R);
22     return s;
23 }
24 int Ask(int Lst, int Cur, int L, int R, int k) {
25     if (L == R) return L;
26     int Mid = (L + R) >> 1;
27     int Left = tree[tree[Cur].Lson].Sum - tree[tree[Lst].Lson].Sum;
28     if (Left >= k) return Ask(tree[Lst].Lson, tree[Cur].Lson, L, Mid, k);
29     k -= Left;
30     return Ask(tree[Lst].Rson, tree[Cur].Rson, Mid + 1, R, k);
31 }
32 int Add(int Lst, int pos) {
33     int root = Same(Lst);

```

```

34     tree[root].Sum++;
35     if (tree[root].L == tree[root].R) return root;
36     int mid = (tree[root].L + tree[root].R) >> 1;
37     if (pos <= mid) tree[root].Lson = Add(tree[root].Lson, pos);
38     else tree[root].Rson = Add(tree[root].Rson, pos);
39     return root;
40 }
41 int main() {
42     scanf("%d%d", &n, &m);
43     int up = 0;
44     for (int i = 1; i <= n; i++){
45         scanf("%d", &a[i]);
46         b[i] = a[i];
47     }
48     sort(b + 1, b + n + 1);
49     up = unique(b + 1, b + n + 1) - b - 1;
50     for (int i = 1; i <= n; i++){
51         int tmp = lower_bound(b + 1, b + up + 1, a[i]) - b;
52         Real[tmp] = a[i];
53         a[i] = tmp;
54     }
55     tot = 0;
56     root[0] = build(1, up);
57     for (int i = 1; i <= n; i++){
58         root[i] = Add(root[i - 1], a[i]);
59     }
60     for (int i = 1; i <= m; i++){
61         int u, v, w;
62         scanf("%d%d%d", &u, &v, &w);
63         printf("%d\n", Real[Ask(root[u - 1], root[v], 1, up, w)]);
64     }
65     return 0;
66 }

```

---

### 3.6 可持久化 Trie

```

1  int Pre[N];
2  int n, q, Len, cnt, Lstans;
3  char s[N];
4  int First[N], Last[N];
5  int Root[N];
6  int Trie_tot;

```

```

7  struct node{
8      int To[30];
9      int Lst;
10 }Trie[N];
11 int tot;
12 struct node1{
13     int L, R, Lson, Rson, Sum;
14 }tree[N * 25];
15 int Build(int L, int R){
16     ++tot;
17     tree[tot].L = L;
18     tree[tot].R = R;
19     tree[tot].Lson = tree[tot].Rson = tree[tot].Sum = 0;
20     if (L == R) return tot;
21     int s = tot;
22     int mid = (L + R) >> 1;
23     tree[s].Lson = Build(L, mid);
24     tree[s].Rson = Build(mid + 1, R);
25     return s;
26 }
27 int Same(int x){
28     ++tot;
29     tree[tot] = tree[x];
30     return tot;
31 }
32 int Add(int Lst, int pos){
33     int s = Same(Lst);
34     tree[s].Sum++;
35     if (tree[s].L == tree[s].R) return s;
36     int Mid = (tree[s].L + tree[s].R) >> 1;
37     if (pos <= Mid) tree[s].Lson = Add(tree[Lst].Lson, pos);
38     else tree[s].Rson = Add(tree[Lst].Rson, pos);
39     return s;
40 }
41
42 int Ask(int Lst, int Cur, int L, int R, int pos){
43     if (L >= pos) return 0;
44     if (R < pos) return tree[Cur].Sum - tree[Lst].Sum;
45     int Mid = (L + R) >> 1;
46     int Ret = Ask(tree[Lst].Lson, tree[Cur].Lson, L, Mid, pos);
47     Ret += Ask(tree[Lst].Rson, tree[Cur].Rson, Mid + 1, R, pos);
48     return Ret;

```

```

49  }
50
51  int main(){
52      while (scanf("%d", &n) == 1){
53          for (int i = 1; i <= Trie_tot; i++){
54              for (int j = 1; j <= 26; j++){
55                  Trie[i].To[j] = 0;
56                  Trie[i].Lst = 0;
57              }
58              Trie_tot = 1;
59              cnt = 0;
60              for (int ii = 1; ii <= n; ii++){
61                  scanf("%s", s + 1);
62                  Len = strlen(s + 1);
63                  int Cur = 1;
64                  First[ii] = cnt + 1;
65                  for (int i = 1; i <= Len; i++){
66                      int ch = s[i] - 'a' + 1;
67                      if (Trie[Cur].To[ch] == 0){
68                          ++Trie_tot;
69                          Trie[Cur].To[ch] = Trie_tot;
70                      }
71                      Cur = Trie[Cur].To[ch];
72                      Pre[++cnt] = Trie[Cur].Lst;
73                      Trie[Cur].Lst = ii;
74                  }
75                  Last[ii] = cnt;
76              }
77              tot = 0;
78              Root[0] = Build(0, n);
79              for (int i = 1; i <= cnt; i++){
80                  Root[i] = Add(Root[i - 1], Pre[i]);
81              }
82              Lstans = 0;
83              scanf("%d", &q);
84              for (int ii = 1; ii <= q; ii++){
85                  int L, R;
86                  scanf("%d%d", &L, &R);
87                  L = (L + Lstans) % n + 1;
88                  R = (R + Lstans) % n + 1;
89                  if (L > R) swap(L, R);
90                  int Ret = Ask(Root[First[L] - 1], Root[Last[R]], 0, n, L);

```



```

91         printf("%d\n", Ret);
92         Lstans = Ret;
93     }
94 }
95 return 0;
96 }

```

---

### 3.7 k-d 树

---

```

1  long long norm(const long long &x) {
2      // For manhattan distance
3      return std::abs(x);
4      // For euclid distance
5      return x * x;
6  }
7
8  struct Point {
9      int x, y, id;
10
11      const int& operator [] (int index) const {
12          if (index == 0) {
13              return x;
14          } else {
15              return y;
16          }
17      }
18
19      friend long long dist(const Point &a, const Point &b) {
20          long long result = 0;
21          for (int i = 0; i < 2; ++i) {
22              result += norm(a[i] - b[i]);
23          }
24          return result;
25      }
26 } point[N];
27
28 struct Rectangle {
29     int min[2], max[2];
30
31     Rectangle() {
32         min[0] = min[1] = INT_MAX;
33         max[0] = max[1] = INT_MIN;

```

```

34     }
35
36     void add(const Point &p) {
37         for (int i = 0; i < 2; ++i) {
38             min[i] = std::min(min[i], p[i]);
39             max[i] = std::max(max[i], p[i]);
40         }
41     }
42
43     long long dist(const Point &p) {
44         long long result = 0;
45         for (int i = 0; i < 2; ++i) {
46             // For minimum distance
47             result += norm(std::min(std::max(p[i], min[i]), max[i]) - p[i]);
48             // For maximum distance
49             result += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
50         }
51         return result;
52     }
53 };
54
55 struct Node {
56     Point separator;
57     Rectangle rectangle;
58     int child[2];
59
60     void reset(const Point &p) {
61         separator = p;
62         rectangle = Rectangle();
63         rectangle.add(p);
64         child[0] = child[1] = 0;
65     }
66 } tree[N << 1];
67
68 int size, pivot;
69
70 bool compare(const Point &a, const Point &b) {
71     if (a[pivot] != b[pivot]) {
72         return a[pivot] < b[pivot];
73     }
74     return a.id < b.id;
75 }

```

```

76
77 int build(int l, int r, int type = 1) {
78     pivot = type;
79     if (l >= r) {
80         return 0;
81     }
82     int x = ++size;
83     int mid = l + r >> 1;
84     std::nth_element(point + l, point + mid, point + r, compare);
85     tree[x].reset(point[mid]);
86     for (int i = l; i < r; ++i) {
87         tree[x].rectangle.add(point[i]);
88     }
89     tree[x].child[0] = build(l, mid, type ^ 1);
90     tree[x].child[1] = build(mid + 1, r, type ^ 1);
91     return x;
92 }
93
94 int insert(int x, const Point &p, int type = 1) {
95     pivot = type;
96     if (x == 0) {
97         tree[++size].reset(p);
98         return size;
99     }
100     tree[x].rectangle.add(p);
101     if (compare(p, tree[x].separator)) {
102         tree[x].child[0] = insert(tree[x].child[0], p, type ^ 1);
103     } else {
104         tree[x].child[1] = insert(tree[x].child[1], p, type ^ 1);
105     }
106     return x;
107 }
108
109 // For minimum distance
110 void query(int x, const Point &p, std::pair<long long, int> &answer, int type = 1) {
111     pivot = type;
112     if (x == 0 || tree[x].rectangle.dist(p) > answer.first) {
113         return;
114     }
115     answer = std::min(answer,
116         std::make_pair(dist(tree[x].separator, p), tree[x].separator.id));
117     if (compare(p, tree[x].separator)) {

```

```

118         query(tree[x].child[0], p, answer, type ^ 1);
119         query(tree[x].child[1], p, answer, type ^ 1);
120     } else {
121         query(tree[x].child[1], p, answer, type ^ 1);
122         query(tree[x].child[0], p, answer, type ^ 1);
123     }
124 }
125
126 std::priority_queue<std::pair<long long, int> > answer;
127
128 void query(int x, const Point &p, int k, int type = 1) {
129     pivot = type;
130     if (x == 0 ||
131         (int)answer.size() == k && tree[x].rectangle.dist(p) > answer.top().first) {
132         return;
133     }
134     answer.push(std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
135     if ((int)answer.size() > k) {
136         answer.pop();
137     }
138     if (compare(p, tree[x].seperator)) {
139         query(tree[x].child[0], p, k, type ^ 1);
140         query(tree[x].child[1], p, k, type ^ 1);
141     } else {
142         query(tree[x].child[1], p, k, type ^ 1);
143         query(tree[x].child[0], p, k, type ^ 1);
144     }
145 }

```

---

### 3.8 莫队算法

---

```

1 struct node{
2     int l, r, id;
3     friend bool operator < (const node &a, const node &b){
4         if (a.l / Block == b.l / Block) return a.r / Block < b.r / Block;
5         return a.l / Block < b.l / Block;
6     }
7 }q[N];
8 Block = int(sqrt(n));
9 for (int i = 1; i <= m; i++){
10     scanf("%d%d", &q[i].l, &q[i].r);
11     q[i].id = i;

```

```

12 }
13 sort(q + 1, q + 1 + m);
14 Cur = a[1]; /// Hints: adjust by yourself
15 Le = Ri = 1;
16 for (int i = 1; i <= m; i++){
17     while (q[i].r > Ri) Ri++, ChangeRi(1, Le, Ri);
18     while (q[i].l > Le) ChangeLe(-1, Le, Ri), Le++;
19     while (q[i].l < Le) Le--, ChangeLe(1, Le, Ri);
20     while (q[i].r < Ri) ChangeRi(-1, Le, Ri), Ri--;
21     Ans[q[i].id] = Cur;
22 }

```

---

### 3.9 树上在线莫队

```

1 bool operator<(qes a,qes b){
2     if(dfn[a.x]/B!=dfn[b.x]/B)return dfn[a.x]/B<dfn[b.x]/B;
3     if(dfn[a.y]/B!=dfn[b.y]/B)return dfn[a.y]/B<dfn[b.y]/B;
4     if(a.tm/B!=b.tm/B)return a.tm/B<b.tm/B;
5     return a.tm<b.tm;
6 }
7 void vxor(int x){
8     if(vis[x])ans-=(LL)W[cnt[col[x]]]*V[col[x]],cnt[col[x]]--;
9     else cnt[col[x]]++,ans+=(LL)W[cnt[col[x]]]*V[col[x]];
10    vis[x]^=1;
11 }
12 void change(int x,int y){
13     if(vis[x]){
14         vxor(x);col[x]=y;vxor(x);
15     }else col[x]=y;
16 }
17 void TimeMachine(int tar){///XD
18     for(int i=now+1;i<=tar;i++)change(C[i].x,C[i].y);
19     for(int i=now;i>tar;i--)change(C[i].x,C[i].pre);
20     now=tar;
21 }
22 void vxor(int x,int y){
23     while(x!=y)if(dep[x]>dep[y])vxor(x),x=fa[x];
24     else vxor(y),y=fa[y];
25 }
26 for(int i=1;i<=q;i++){
27     int ty=getint(),x=getint(),y=getint();
28     if(ty&&dfn[x]>dfn[y])swap(x,y);

```

```

29         if(ty==0) C[++Csize]=(oper){x,y,pre[x],i},pre[x]=y;
30         else Q[Qsize+1]=(qes){x,y,Qsize+1,Csize},Qsize++;
31     }sort(Q+1,Q+1+Qsize);
32     int u=Q[1].x,v=Q[1].y;
33     TimeMachine(Q[1].tm);
34     vxor(Q[1].x,Q[1].y);
35     int LCA=lca(Q[1].x,Q[1].y);
36     vxor(LCA);anss[Q[1].id]=ans;vxor(LCA);
37     for(int i=2;i<=Qsize;i++){
38         TimeMachine(Q[i].tm);
39         vxor(Q[i-1].x,Q[i].x);
40         vxor(Q[i-1].y,Q[i].y);
41         int LCA=lca(Q[i].x,Q[i].y);
42         vxor(LCA);
43         anss[Q[i].id]=ans;
44         vxor(LCA);
45     }

```

---

### 3.10 整体二分

```

1  struct BIT{
2      LL d[maxn];
3      inline int lowbit(int x){return x&-x;}
4      LL get(int x){
5          LL ans=0;
6          while(x)ans+=d[x],x-=lowbit(x);
7          return ans;
8      }
9      void updata(int x,LL f){
10         while(x<=m)d[x]+=f,x+=lowbit(x);
11     }
12     void add(int l,int r,LL f){
13         updata(l,f);
14         updata(r+1,-f);
15     }
16 }T,T2;
17 int anss[maxn],wana[maxn];
18 struct qes{
19     LL x,y,z;
20     qes(LL _x=0,LL _y=0,LL _z=0):
21         x(_x),y(_y),z(_z){}
22 }q[maxn],p[maxn];

```

```

23 bool part(qes &q){
24     if(q.y+q.z>=wana[q.x])return 1;
25     q.z+=q.y;q.y=0;return 0;
26 }
27 void solve(int lef,int rig,int l,int r){
28     if(l==r){
29         for(int i=lef;i<=rig;i++)if(anss[p[i].x]!=-1)
30             anss[p[i].x]=1;return;
31     }int mid=(l+r)>>1;
32     for(int i=l;i<=mid;i++){
33         if(q[i].x<=q[i].y)T.add(q[i].x,q[i].y,q[i].z);
34         else T.add(1,q[i].y,q[i].z),T.add(q[i].x,m,q[i].z);
35     }for(int i=lef;i<=rig;i++){
36         p[i].y=0;
37         for(int j=0;j<0[p[i].x].size()&&p[i].y<=int(1e9)+1;j++)
38             p[i].y+=T.get(0[p[i].x][j]);
39     }for(int i=l;i<=mid;i++){
40         if(q[i].x<=q[i].y)T.add(q[i].x,q[i].y,-q[i].z);
41         else T.add(1,q[i].y,-q[i].z),T.add(q[i].x,m,-q[i].z);
42     }int dv=stable_partition(p+lef,p+rig+1,part)-p-1;
43     if(lef<=dv)
44         solve(lef,dv,l,mid);
45     if(dv+1<=rig)
46         solve(dv+1,rig,mid+1,r);
47 }

```

---

### 3.11 树状数组 kth

```

1 int find(int k){
2     int cnt=0,ans=0;
3     for(int i=22;i>=0;i--){
4         ans+=(1<<i);
5         if(ans>n || cnt+d[ans]>=k)ans-=(1<<i);
6         else cnt+=d[ans];
7     }
8     return ans+1;
9 }

```

---

### 3.12 虚树

```

1  int a[maxn*2],sta[maxn*2];
2  int top=0,k;
3  void build(){
4      top=0;
5      sort(a,a+k,bydfn);
6      k=unique(a,a+k)-a;
7      sta[top++]=1;_n=k;
8      for(int i=0;i<k;i++){
9          int LCA=lca(a[i],sta[top-1]);
10         while(dep[LCA]<dep[sta[top-1]]){
11             if(dep[LCA]>=dep[sta[top-2]]){
12                 add_edge(LCA,sta[top-1]);
13                 if(sta[top-1]!=LCA)sta[top++]=LCA;
14                 break;
15             }add_edge(sta[top-2],sta[top-1]);top--;
16         }if(sta[top-1]!=a[i])sta[top++]=a[i];
17     }
18     while(top>1)
19         add_edge(sta[top-2],sta[top-1]),top--;
20     for(int i=0;i<k;i++)inr[a[i]]=1;
21 }

```

---

### 3.13 点分治 (zky)

---

```

1  int siz[maxn],f[maxn],dep[maxn],cant[maxn],root,All,d[maxn];
2  void makert(int u,int fa){
3      siz[u]=1;f[u]=0;
4      for(int i=0;i<G[u].size();i++){
5          edge e=G[u][i];
6          if(e.v!=fa&&!cant[e.v]){
7              dep[e.v]=dep[u]+1;
8              makert(e.v,u);
9              siz[u]+=siz[e.v];
10             f[u]=max(f[u],siz[e.v]);
11         }
12     }f[u]=max(f[u],All-f[u]);
13     if(f[root]>f[u])root=u;
14 }
15 void dfs(int u,int fa){
16     //Gain data
17     for(int i=0;i<G[u].size();i++){
18         edge e=G[u][i];

```



```

19         if(e.v==fa || cant[e.v])continue;
20         d[e.v]=d[u]+e.w;
21         dfs(e.v,u);
22     }
23 }
24 void calc(int u){
25     d[u]=0;
26     for(int i=0;i<G[u].size();i++){
27         edge e=G[u][i];
28         if(cant[e.v])continue;
29         d[e.v]=e.w;
30         dfs(e.v,u);
31     }
32 }
33 }
34 void solve(int u){
35     calc(u);cant[u]=1;
36     for(int i=0;i<G[u].size();i++){
37         edge e=G[u][i];
38         if(cant[e.v])continue;
39         All=siz[e.v];
40         f[root=0]=n+1;
41         makert(e.v,0);
42         solve(root);
43     }
44 }
45 All=n
46 f[root=0]=n+1;
47 makert(1,1);
48 solve(root);

```

---

### 3.14 元芳树

---

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn=1e4+1e4+233;
4  const int BIT=18;
5  int n,m,q;
6  struct edge{
7      int u,v,w;
8      bool operator==(edge oth)const{
9          return u==oth.u && v==oth.v && w==oth.w;

```

```

10         }
11         bool operator!=(edge oth)const{
12             return !(*this==oth);
13         }
14     };
15     vector<edge>G[maxn],T[maxn];
16
17     int dfn[maxn],low[maxn],tot,rlen[maxn];
18     bool ins[maxn];
19     stack<edge>S;
20     int Rcnt=0;
21     vector<edge>ring[maxn];
22     vector<int>bel[maxn],sum[maxn],dis[maxn];
23     int fa[maxn][BIT];
24     int dep[maxn],dep2[maxn],fw[maxn];
25     vector<pair<int,int> >ind[maxn];
26     map<pair<int,int>,int>Mw;
27     pair<int,int>pack(int a,int b){
28         if(a>b)swap(a,b);
29         return make_pair(a,b);
30     }
31     void tarjan(int u){
32         dfn[u]=low[u]=++tot;
33         for(int i=0;i<G[u].size();i++){
34             edge e=G[u][i];
35             if(dfn[e.v])
36                 low[u]=min(low[u],dfn[e.v]);
37             else{
38                 S.push(e);
39                 tarjan(e.v);
40                 if(low[e.v]==dfn[u]){
41
42                     if(S.top()==e){
43                         fa[e.v][0]=u;
44                         fw[e.v]=e.w;
45                         S.pop();
46                         continue;
47                     }
48
49                     Rcnt++;
50                     edge ed;
51                     do{

```

```

52         ed=S.top();S.pop();
53         ring[Rcnt].push_back(ed);
54     }while(ed!=e);
55     reverse(ring[Rcnt].begin(),ring[Rcnt].end());
56     int last=ring[Rcnt].back().v;
57     ring[Rcnt].push_back((edge){last,u,Mw[pack(last,u)]});
58     }
59     low[u]=min(low[u],low[e.v]);
60 }
61 }
62 }
63 void up(int u){
64     if(dep[u]||u==1)return ;
65     if(fa[u][0])up(fa[u][0]);
66     dep[u]=dep[fa[u][0]]+1;
67     fw[u]+=fw[fa[u][0]];
68 }
69 void build(){
70     S.push((edge){0,1,0});
71     tarjan(1);
72
73     for(int i=1;i<=Rcnt;i++){
74         rlen[i]=0;
75         sum[i].resize(ring[i].size());
76         dis[i].resize(ring[i].size());
77         for(int j=0;j<ring[i].size();j++){
78             rlen[i]+=ring[i][j].w;
79             ind[i].push_back(make_pair(ring[i][j].u,j));
80         }
81         sum[i][0]=0;
82         fw[i+n]=0;
83         fa[i+n][0]=ring[i][0].u;
84         for(int j=1;j<ring[i].size();j++){
85             sum[i][j]=sum[i][j-1]+ring[i][j-1].w;
86             dis[i][j]=min(sum[i][j],rlen[i]-sum[i][j]);
87             fw[ring[i][j].u]=dis[i][j];
88             fa[ring[i][j].u][0]=i+n;
89         }
90         sort(ind[i].begin(),ind[i].end());
91     }
92
93     for(int i=1;i<=n+Rcnt;i++)

```

```

94         up(i);
95
96     for(int j=1;j<BIT;j++)
97     for(int i=1;i<=n+Rcnt;i++)if(fa[i][j-1])
98         fa[i][j]=fa[fa[i][j-1]][j-1];
99
100 }
101 pair<int,int>second_lca;
102 int lca(int u,int v){
103     if(dep[u]<dep[v])swap(u,v);
104     int d=dep[u]-dep[v];
105     for(int i=0;i<BIT;i++)if(d>>i&1)
106         u=fa[u][i];
107     if(u==v)return u;
108     for(int i=BIT-1;i>=0;i--)if(fa[u][i]!=fa[v][i]){
109         u=fa[u][i];
110         v=fa[v][i];
111     }
112     second_lca=make_pair(u,v);
113     return fa[u][0];
114 }
115 int main(){
116
117     freopen("bzoj2125.in","r",stdin);
118
119     scanf("%d%d%d",&n,&m,&q);
120     for(int i=1;i<=m;i++){
121         int u,v,w;scanf("%d%d%d",&u,&v,&w);
122         G[u].push_back((edge){u,v,w});
123         G[v].push_back((edge){v,u,w});
124         Mw[pack(u,v)]=w;
125     }
126
127     build();
128     while(q--){
129         int u,v;
130         scanf("%d%d",&u,&v);
131         int LCA=lca(u,v);
132         if(LCA<=n)printf("%d\n",fw[u]+fw[v]-2*fw[LCA]);
133         else{
134             if(dep[u]<dep[v])swap(u,v);
135             int R=LCA-n;

```

```

136         int uu=second_lca.first;
137         int vv=second_lca.second;
138         int ans=fw[u]-fw[uu]+fw[v]-fw[vv];
139         int uid,vid;
140         uid=lower_bound(ind[R].begin(),ind[R].end(),make_pair(uu,-1))->second;
141         vid=lower_bound(ind[R].begin(),ind[R].end(),make_pair(vv,-1))->second;
142         ans+=min(abs(sum[R][uid]-sum[R][vid]),rlen[R]-abs(sum[R][uid]-sum[R][vid]));
143         printf("%d\n",ans);
144     }
145 }
146 return 0;
147 }

```

---

## 4 图论

### 4.1 强连通分量

---

```

1  int stamp, comps, top;
2  int dfn[N], low[N], comp[N], stack[N];
3
4  void tarjan(int x) {
5      dfn[x] = low[x] = ++stamp;
6      stack[top++] = x;
7      for (int i = 0; i < (int)edge[x].size(); ++i) {
8          int y = edge[x][i];
9          if (!dfn[y]) {
10             tarjan(y);
11             low[x] = std::min(low[x], low[y]);
12         } else if (!comp[y]) {
13             low[x] = std::min(low[x], dfn[y]);
14         }
15     }
16     if (low[x] == dfn[x]) {
17         comps++;
18         do {
19             int y = stack[--top];
20             comp[y] = comps;
21         } while (stack[top] != x);
22     }
23 }
24
25 void solve() {

```

```

26     stamp = comps = top = 0;
27     std::fill(dfn, dfn + n, 0);
28     std::fill(comp, comp + n, 0);
29     for (int i = 0; i < n; ++i) {
30         if (!dfn[i]) {
31             tarjan(i);
32         }
33     }
34 }

```

---

#### 4.1.1 点双连通分量 (lyx)

---

```

1 void Dfs(int x, int lst){
2     dfn[x] = ++dfc;
3     low[x] = dfn[x];
4     stack[++cnt] = x;
5     int son = 0;
6
7     for (int i = g[x]; i; i = nxt[i]){
8         if (!dfn[adj[i]]){
9             ++son;
10            Dfs(adj[i], i);
11            low[x] = min(low[x], low[adj[i]]);
12            if (low[adj[i]] >= dfn[x]){
13                int Tmp;
14                iscut[x] = 1;
15                ++block;
16                E[x].push_back(block + n);
17                do{
18                    Tmp = stack[cnt--];
19                    belong[Tmp] = block + n;
20
21                    E[Tmp].push_back(block + n);
22                }while (Tmp != adj[i]);
23            }
24        }
25        else
26            if (i ^ lst != 1) low[x] = min(low[x], dfn[adj[i]]);
27    }
28    if (x == Root && son == 1) iscut[x] = 0, belong[x] = E[x][0];
29    if (x == Root && son == 0){
30        ++block;

```

```

31         belong[x] = block + n;
32     }
33 }
34
35     tot = 1;////////////////////1
36     block = 0;
37     cnt = 0;
38     for (int i = 1; i <= n * 3; i++){
39         g[i] = 0;
40         iscut[i] = 0;
41         G[i].clear();
42         E[i].clear();
43         belong[i] = 0;
44         size[i] = 0;
45         low[i] = dfn[i] = 0;
46     }
47     for (int i = 1; i <= m; i++){
48         int x, y;
49         scanf("%d%d", &x, &y);
50         Ins(x,y);
51     }
52
53     dfc = 0;
54     for (int i = 1; i <= n; i++)
55         if (dfn[i] == 0){
56             Root = i;
57             Dfs(i, 0);
58         }

```

---

## 4.2 2-SAT 问题

---

```

1  int stamp, comps, top;
2  int dfn[N], low[N], comp[N], stack[N];
3
4  void add(int x, int a, int y, int b) {
5      edge[x << 1 | a].push_back(y << 1 | b);
6  }
7
8  void tarjan(int x) {
9      dfn[x] = low[x] = ++stamp;
10     stack[top++] = x;
11     for (int i = 0; i < (int)edge[x].size(); ++i) {

```

```

12     int y = edge[x][i];
13     if (!dfn[y]) {
14         tarjan(y);
15         low[x] = std::min(low[x], low[y]);
16     } else if (!comp[y]) {
17         low[x] = std::min(low[x], dfn[y]);
18     }
19 }
20 if (low[x] == dfn[x]) {
21     comps++;
22     do {
23         int y = stack[--top];
24         comp[y] = comps;
25     } while (stack[top] != x);
26 }
27 }
28
29 bool solve() {
30     int counter = n + n + 1;
31     stamp = top = comps = 0;
32     std::fill(dfn, dfn + counter, 0);
33     std::fill(comp, comp + counter, 0);
34     for (int i = 0; i < counter; ++i) {
35         if (!dfn[i]) {
36             tarjan(i);
37         }
38     }
39     for (int i = 0; i < n; ++i) {
40         if (comp[i << 1] == comp[i << 1 | 1]) {
41             return false;
42         }
43         answer[i] = (comp[i << 1 | 1] < comp[i << 1]);
44     }
45     return true;
46 }

```

---

## 4.3 二分图最大匹配

### 4.3.1 Hungary 算法

时间复杂度:  $\mathcal{O}(V \cdot E)$

---



```

1  vector<int>G[maxn];
2  int Link[maxn],vis[maxn],T;
3  bool find(int x){
4      for(int i=0;i<G[x].size();i++){
5          int v=G[x][i];
6          if(vis[v]==T)continue;
7          vis[v]=T;
8          if(!Link[v]||find(Link[v])){
9              Link[v]=x;
10             return 1;
11         }
12     }return 0;
13 }
14 int Hungarian(int n){
15     int ans=0;
16     memset(Link,0,sizeof Link);
17     for(int i=1;i<=n;i++){
18         T++;
19         ans+=find(i);
20     }return ans;
21 }

```

---

#### 4.3.2 Hopcroft Karp 算法

时间复杂度:  $\mathcal{O}(\sqrt{V} \cdot E)$

---

```

1  int matchx[N], matchy[N], level[N];
2
3  bool dfs(int x) {
4      for (int i = 0; i < (int)edge[x].size(); ++i) {
5          int y = edge[x][i];
6          int w = matchy[y];
7          if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
8              matchx[x] = y;
9              matchy[y] = x;
10             return true;
11         }
12     }
13     level[x] = -1;
14     return false;
15 }
16
17 int solve() {

```

```

18     std::fill(matchx, matchx + n, -1);
19     std::fill(matchy, matchy + m, -1);
20     for (int answer = 0; ; ) {
21         std::vector<int> queue;
22         for (int i = 0; i < n; ++i) {
23             if (matchx[i] == -1) {
24                 level[i] = 0;
25                 queue.push_back(i);
26             } else {
27                 level[i] = -1;
28             }
29         }
30         for (int head = 0; head < (int)queue.size(); ++head) {
31             int x = queue[head];
32             for (int i = 0; i < (int)edge[x].size(); ++i) {
33                 int y = edge[x][i];
34                 int w = matchy[y];
35                 if (w != -1 && level[w] < 0) {
36                     level[w] = level[x] + 1;
37                     queue.push_back(w);
38                 }
39             }
40         }
41         int delta = 0;
42         for (int i = 0; i < n; ++i) {
43             if (matchx[i] == -1 && dfs(i)) {
44                 delta++;
45             }
46         }
47         if (delta == 0) {
48             return answer;
49         } else {
50             answer += delta;
51         }
52     }
53 }

```

---

#### 4.4 二分图最大权匹配

时间复杂度:  $\mathcal{O}(V^4)$

---

```

1  int labelx[N], labely[N], match[N], slack[N];
2  bool visitx[N], visity[N];

```

```

3
4  bool dfs(int x) {
5      visitx[x] = true;
6      for (int y = 0; y < n; ++y) {
7          if (visity[y]) {
8              continue;
9          }
10         int delta = labelx[x] + labely[y] - graph[x][y];
11         if (delta == 0) {
12             visity[y] = true;
13             if (match[y] == -1 || dfs(match[y])) {
14                 match[y] = x;
15                 return true;
16             }
17         } else {
18             slack[y] = std::min(slack[y], delta);
19         }
20     }
21     return false;
22 }
23
24 int solve() {
25     for (int i = 0; i < n; ++i) {
26         match[i] = -1;
27         labelx[i] = INT_MIN;
28         labely[i] = 0;
29         for (int j = 0; j < n; ++j) {
30             labelx[i] = std::max(labelx[i], graph[i][j]);
31         }
32     }
33     for (int i = 0; i < n; ++i) {
34         while (true) {
35             std::fill(visitx, visitx + n, 0);
36             std::fill(visity, visity + n, 0);
37             for (int j = 0; j < n; ++j) {
38                 slack[j] = INT_MAX;
39             }
40             if (dfs(i)) {
41                 break;
42             }
43             int delta = INT_MAX;
44             for (int j = 0; j < n; ++j) {

```

```

45         if (!visity[j]) {
46             delta = std::min(delta, slack[j]);
47         }
48     }
49     for (int j = 0; j < n; ++j) {
50         if (visitx[j]) {
51             labelx[j] -= delta;
52         }
53         if (visity[j]) {
54             labely[j] += delta;
55         } else {
56             slack[j] -= delta;
57         }
58     }
59 }
60 }
61 int answer = 0;
62 for (int i = 0; i < n; ++i) {
63     answer += graph[match[i]][i];
64 }
65 return answer;
66 }

```

---

## 4.5 最大流 (dinic)

时间复杂度:  $\mathcal{O}(V^2 \cdot E)$

---

```

1  struct edge{int u,v,cap,flow;};
2  vector<edge>edges;
3  vector<int>G[maxn];
4  int s,t;
5  int cur[maxn],d[maxn];
6  void add(int u,int v,int cap){
7      edges.push_back((edge){u,v,cap,0});
8      G[u].push_back(edges.size()-1);
9      edges.push_back((edge){v,u,0,0});
10     G[v].push_back(edges.size()-1);
11 }
12 bool bfs(){
13     static int vis[maxn];
14     memset(vis,0,sizeof vis);vis[s]=1;
15     queue<int>q;q.push(s);d[s]=0;
16     while(!q.empty()){

```

```

17         int u=q.front();q.pop();
18         for(int i=0;i<G[u].size();i++){
19             edge e=edges[G[u][i]];if(vis[e.v]||e.cap==e.flow)continue;
20             d[e.v]=d[u]+1;vis[e.v]=1;q.push(e.v);
21         }
22     }return vis[t];
23 }
24 int dfs(int u,int a){
25     if(u==t||!a)return a;
26     int flow=0,f;
27     for(int &i=cur[u];i<G[u].size();i++){
28         edge e=edges[G[u][i]];
29         if(d[e.v]==d[u]+1&&(f=dfs(e.v,min(a,e.cap-e.flow)))>0){
30             edges[G[u][i]].flow+=f;
31             edges[G[u][i]^1].flow-=f;
32             flow+=f;a-=f;if(!a)break;
33         }
34     }return flow;
35 }
36 int dinic(){
37     int flow=0,x;
38     while(bfs()){
39         memset(cur,0,sizeof cur);
40         while(x=dfs(s,INT_MAX)){
41             flow+=x;
42             memset(cur,0,sizeof cur);
43         }
44     }return flow;
45 }

```

---

## 4.6 最大流 (sap)

时间复杂度:  $\mathcal{O}(V^2 \cdot E)$

---

```

1  int g[T], adj[M], nxt[M], f[M];
2  int cnt[T], dist[T], cur[T], fa[T], dat[T];
3  void Ins(int x, int y, int ff, int rf){
4      adj[++tot] = y; nxt[tot] = g[x]; g[x] = tot; f[tot] = ff;
5      adj[++tot] = x; nxt[tot] = g[y]; g[y] = tot; f[tot] = rf;
6  }
7  int sap(int s, int t){
8      int x, sum;
9      for (int i = 1; i <= t; i++){

```

```

10         dist[i] = 1;
11         cur[i] = g[i];
12         fa[i] = 0;
13         dat[i] = 0;
14         cnt[i] = 0;
15     }
16     cnt[0] = 1; cnt[1] = t - 1;
17     dist[t] = 0;
18     dat[s] = INF;
19     x = s;
20     sum = 0;
21     while (1){
22         int p;
23         for (p = cur[x]; p; p = nxt[p]){
24             if (f[p] > 0 && dist[adj[p]] == dist[x] - 1) break;
25         }
26         if (p > 0){
27             cur[x] = p;
28             fa[adj[p]] = p;
29             dat[adj[p]] = min(dat[x], f[p]);
30             x = adj[p];
31             if (x == t){
32                 sum += dat[x];
33                 while (x != s){
34                     f[fa[x]] -= dat[t];
35                     f[fa[x] ^ 1] += dat[t];
36                     x = adj[fa[x] ^ 1];
37                 }
38             }
39         } else {
40             cnt[dist[x]]--;
41             if (cnt[dist[x]] == 0) return sum;
42             dist[x] = t + 1;
43             for (int p = g[x]; p; p = nxt[p]){
44                 if (f[p] > 0 && dist[adj[p]] + 1 < dist[x]){
45                     dist[x] = dist[adj[p]] + 1;
46                     cur[x] = p;
47                 }
48             }
49             cnt[dist[x]]++;
50             if (dist[s] > t) return sum;
51             if (x != s) x = adj[fa[x] ^ 1];

```

```

52         }
53     }
54 }
55 /*
56 tot = 1
57 edges' id start from 2
58 remember to clean g
59 t is the number of points
60 */

```

---

## 4.7 上下界网络流

$B(u, v)$  表示边  $(u, v)$  流量的下界,  $C(u, v)$  表示边  $(u, v)$  流量的上界,  $F(u, v)$  表示边  $(u, v)$  的流量。设  $G(u, v) = F(u, v) - B(u, v)$ , 显然有

$$0 \leq G(u, v) \leq C(u, v) - B(u, v)$$

### 4.7.1 无源汇的上下界可行流

建立超级源点  $S^*$  和超级汇点  $T^*$ , 对于原图每条边  $(u, v)$  在新网络中连如下三条边:  $S^* \rightarrow v$ , 容量为  $B(u, v)$ ;  $u \rightarrow T^*$ , 容量为  $B(u, v)$ ;  $u \rightarrow v$ , 容量为  $C(u, v) - B(u, v)$ 。最后求新网络的最大流, 判断从超级源点  $S^*$  出发的边是否都满流即可, 边  $(u, v)$  的最终解中的实际流量为  $G(u, v) + B(u, v)$ 。

### 4.7.2 有源汇的上下界可行流

从汇点  $T$  到源点  $S$  连一条上界为  $\infty$ , 下界为 0 的边。按照无源汇的上下界可行流一样做即可, 流量即为  $T \rightarrow S$  边上的流量。

### 4.7.3 有源汇的上下界最大流

1. 在有源汇的上下界可行流中, 从汇点  $T$  到源点  $S$  的边改为连一条上界为  $\infty$ , 下届为  $x$  的边。 $x$  满足二分性质, 找到最大的  $x$  使得新网络存在无源汇的上下界可行流即为原图的最大流。
2. 从汇点  $T$  到源点  $S$  连一条上界为  $\infty$ , 下界为 0 的边, 变成无源汇的网络。按照无源汇的上下界可行流的方法, 建立超级源点  $S^*$  和超级汇点  $T^*$ , 求一遍  $S^* \rightarrow T^*$  的最大流, 再将 从汇点  $T$  到源点  $S$  的这条边拆掉, 求一次  $S \rightarrow T$  的最大流即可。

### 4.7.4 有源汇的上下界最小流

1. 在有源汇的上下界可行流中, 从汇点  $T$  到源点  $S$  的边改为连一条上界为  $x$ , 下界为 0 的边。 $x$  满足二分性质, 找到最小的  $x$  使得新网络存在无源汇的上下界可行流即为原图的最小流。

2. 按照无源汇的上下界可行流的方法，建立超级源点  $S^*$  与超级汇点  $T^*$ ，求一遍  $S^* \rightarrow T^*$  的最大流，但是注意这一次不加上汇点  $T$  到源点  $S$  的这条边，即不使之改为无源汇的网络去求解。求完后，再加上那条汇点  $T$  到源点  $S$  上界  $\infty$  的边。因为这条边下界为 0，所以  $S^*$ ， $T^*$  无影响，再直接求一次  $S^* \rightarrow T^*$  的最大流。若超级源点  $S^*$  出发的边全部满流，则  $T \rightarrow S$  边上的流量即为原图的最小流，否则无解。

## 4.8 最小费用最大流

### 4.8.1 稀疏图

时间复杂度： $\mathcal{O}(V \cdot E^2)$

---

```
1 struct EdgeList {
2     int size;
3     int last[N];
4     int succ[M], other[M], flow[M], cost[M];
5     void clear(int n) {
6         size = 0;
7         std::fill(last, last + n, -1);
8     }
9     void add(int x, int y, int c, int w) {
10         succ[size] = last[x];
11         last[x] = size;
12         other[size] = y;
13         flow[size] = c;
14         cost[size++] = w;
15     }
16 } e;
17
18 int n, source, target;
19 int prev[N];
20
21 void add(int x, int y, int c, int w) {
22     e.add(x, y, c, w);
23     e.add(y, x, 0, -w);
24 }
25
26 bool augment() {
27     static int dist[N], occur[N];
28     std::vector<int> queue;
29     std::fill(dist, dist + n, INT_MAX);
30     std::fill(occur, occur + n, 0);
31     dist[source] = 0;
```



```

32     occur[source] = true;
33     queue.push_back(source);
34     for (int head = 0; head < (int)queue.size(); ++head) {
35         int x = queue[head];
36         for (int i = e.last[x]; ~i; i = e.succ[i]) {
37             int y = e.other[i];
38             if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
39                 dist[y] = dist[x] + e.cost[i];
40                 prev[y] = i;
41                 if (!occur[y]) {
42                     occur[y] = true;
43                     queue.push_back(y);
44                 }
45             }
46         }
47         occur[x] = false;
48     }
49     return dist[target] < INT_MAX;
50 }
51
52 std::pair<int, int> solve() {
53     std::pair<int, int> answer = std::make_pair(0, 0);
54     while (augment()) {
55         int number = INT_MAX;
56         for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
57             number = std::min(number, e.flow[prev[i]]);
58         }
59         answer.first += number;
60         for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
61             e.flow[prev[i]] -= number;
62             e.flow[prev[i] ^ 1] += number;
63             answer.second += number * e.cost[prev[i]];
64         }
65     }
66     return answer;
67 }

```

---

#### 4.8.2 稠密图

使用条件：费用非负

时间复杂度： $\mathcal{O}(V \cdot E^2)$

```

1  struct EdgeList {
2      int size;
3      int last[N];
4      int succ[M], other[M], flow[M], cost[M];
5      void clear(int n) {
6          size = 0;
7          std::fill(last, last + n, -1);
8      }
9      void add(int x, int y, int c, int w) {
10         succ[size] = last[x];
11         last[x] = size;
12         other[size] = y;
13         flow[size] = c;
14         cost[size++] = w;
15     }
16 } e;
17
18 int n, source, target, flow, cost;
19 int slack[N], dist[N];
20 bool visit[N];
21
22 void add(int x, int y, int c, int w) {
23     e.add(x, y, c, w);
24     e.add(y, x, 0, -w);
25 }
26
27 bool relabel() {
28     int delta = INT_MAX;
29     for (int i = 0; i < n; ++i) {
30         if (!visit[i]) {
31             delta = std::min(delta, slack[i]);
32         }
33         slack[i] = INT_MAX;
34     }
35     if (delta == INT_MAX) {
36         return true;
37     }
38     for (int i = 0; i < n; ++i) {
39         if (visit[i]) {
40             dist[i] += delta;
41         }
42     }

```

```

43     return false;
44 }
45
46 int dfs(int x, int answer) {
47     if (x == target) {
48         flow += answer;
49         cost += answer * (dist[source] - dist[target]);
50         return answer;
51     }
52     visit[x] = true;
53     int delta = answer;
54     for (int i = e.last[x]; ~i; i = e.succ[i]) {
55         int y = e.other[i];
56         if (e.flow[i] > 0 && !visit[y]) {
57             if (dist[y] + e.cost[i] == dist[x]) {
58                 int number = dfs(y, std::min(e.flow[i], delta));
59                 e.flow[i] -= number;
60                 e.flow[i ^ 1] += number;
61                 delta -= number;
62                 if (delta == 0) {
63                     dist[x] = INT_MIN;
64                     return answer;
65                 }
66             } else {
67                 slack[y] = std::min(slack[y], dist[y] + e.cost[i] - dist[x]);
68             }
69         }
70     }
71     return answer - delta;
72 }
73
74 std::pair<int, int> solve() {
75     flow = cost = 0;
76     std::fill(dist, dist + n, 0);
77     do {
78         do {
79             fill(visit, visit + n, 0);
80         } while (dfs(source, INT_MAX));
81     } while (!relabel());
82     return std::make_pair(flow, cost);
83 }

```

---

## 4.9 一般图最大匹配

时间复杂度:  $\mathcal{O}(V^3)$

---

```
1  int match[N], belong[N], next[N], mark[N], visit[N];
2  std::vector<int> queue;
3
4  int find(int x) {
5      if (belong[x] != x) {
6          belong[x] = find(belong[x]);
7      }
8      return belong[x];
9  }
10
11 void merge(int x, int y) {
12     x = find(x);
13     y = find(y);
14     if (x != y) {
15         belong[x] = y;
16     }
17 }
18
19 int lca(int x, int y) {
20     static int stamp = 0;
21     stamp++;
22     while (true) {
23         if (x != -1) {
24             x = find(x);
25             if (visit[x] == stamp) {
26                 return x;
27             }
28             visit[x] = stamp;
29             if (match[x] != -1) {
30                 x = next[match[x]];
31             } else {
32                 x = -1;
33             }
34         }
35         std::swap(x, y);
36     }
37 }
38
39 void group(int a, int p) {
```

```

40     while (a != p) {
41         int b = match[a], c = next[b];
42         if (find(c) != p) {
43             next[c] = b;
44         }
45         if (mark[b] == 2) {
46             mark[b] = 1;
47             queue.push_back(b);
48         }
49         if (mark[c] == 2) {
50             mark[c] = 1;
51             queue.push_back(c);
52         }
53         merge(a, b);
54         merge(b, c);
55         a = c;
56     }
57 }
58
59 void augment(int source) {
60     queue.clear();
61     for (int i = 0; i < n; ++i) {
62         next[i] = visit[i] = -1;
63         belong[i] = i;
64         mark[i] = 0;
65     }
66     mark[source] = 1;
67     queue.push_back(source);
68     for (int head = 0; head < (int)queue.size() && match[source] == -1; ++head) {
69         int x = queue[head];
70         for (int i = 0; i < (int)edge[x].size(); ++i) {
71             int y = edge[x][i];
72             if (match[x] == y || find(x) == find(y) || mark[y] == 2) {
73                 continue;
74             }
75             if (mark[y] == 1) {
76                 int r = lca(x, y);
77                 if (find(x) != r) {
78                     next[x] = y;
79                 }
80                 if (find(y) != r) {
81                     next[y] = x;

```

```

82         }
83         group(x, r);
84         group(y, r);
85     } else if (match[y] == -1) {
86         next[y] = x;
87         for (int u = y; u != -1; ) {
88             int v = next[u];
89             int mv = match[v];
90             match[v] = u;
91             match[u] = v;
92             u = mv;
93         }
94         break;
95     } else {
96         next[y] = x;
97         mark[y] = 2;
98         mark[match[y]] = 1;
99         queue.push_back(match[y]);
100     }
101 }
102 }
103 }
104
105 int solve() {
106     std::fill(match, match + n, -1);
107     for (int i = 0; i < n; ++i) {
108         if (match[i] == -1) {
109             augment(i);
110         }
111     }
112     int answer = 0;
113     for (int i = 0; i < n; ++i) {
114         answer += (match[i] != -1);
115     }
116     return answer;
117 }

```

---

#### 4.10 无向图全局最小割

时间复杂度:  $\mathcal{O}(V^3)$

注意事项: 处理重边时, 应该对边权累加

```

1  int node[N], dist[N];
2  bool visit[N];
3
4  int solve(int n) {
5      int answer = INT_MAX;
6      for (int i = 0; i < n; ++i) {
7          node[i] = i;
8      }
9      while (n > 1) {
10         int max = 1;
11         for (int i = 0; i < n; ++i) {
12             dist[node[i]] = graph[node[0]][node[i]];
13             if (dist[node[i]] > dist[node[max]]) {
14                 max = i;
15             }
16         }
17         int prev = 0;
18         memset(visit, 0, sizeof(visit));
19         visit[node[0]] = true;
20         for (int i = 1; i < n; ++i) {
21             if (i == n - 1) {
22                 answer = std::min(answer, dist[node[max]]);
23                 for (int k = 0; k < n; ++k) {
24                     graph[node[k]][node[prev]] =
25                         (graph[node[prev]][node[k]] += graph[node[k]][node[max]]);
26                 }
27                 node[max] = node[--n];
28             }
29             visit[node[max]] = true;
30             prev = max;
31             max = -1;
32             for (int j = 1; j < n; ++j) {
33                 if (!visit[node[j]]) {
34                     dist[node[j]] += graph[node[prev]][node[j]];
35                     if (max == -1 || dist[node[max]] < dist[node[j]]) {
36                         max = j;
37                     }
38                 }
39             }
40         }
41     }
42     return answer;

```

43 }

---

#### 4.11 有根树的同构

时间复杂度:  $\mathcal{O}(V \log V)$

---

```
1  const unsigned long long MAGIC = 4423;
2
3  unsigned long long magic[N];
4  std::pair<unsigned long long, int> hash[N];
5
6  void solve(int root) {
7      magic[0] = 1;
8      for (int i = 1; i <= n; ++i) {
9          magic[i] = magic[i - 1] * MAGIC;
10     }
11     std::vector<int> queue;
12     queue.push_back(root);
13     for (int head = 0; head < (int)queue.size(); ++head) {
14         int x = queue[head];
15         for (int i = 0; i < (int)son[x].size(); ++i) {
16             int y = son[x][i];
17             queue.push_back(y);
18         }
19     }
20     for (int index = n - 1; index >= 0; --index) {
21         int x = queue[index];
22         hash[x] = std::make_pair(0, 0);
23
24         std::vector<std::pair<unsigned long long, int> > value;
25         for (int i = 0; i < (int)son[x].size(); ++i) {
26             int y = son[x][i];
27             value.push_back(hash[y]);
28         }
29         std::sort(value.begin(), value.end());
30
31         hash[x].first = hash[x].first * magic[1] + 37;
32         hash[x].second++;
33         for (int i = 0; i < (int)value.size(); ++i) {
34             hash[x].first = hash[x].first * magic[value[i].second] + value[i].first;
35             hash[x].second += value[i].second;
36         }
37         hash[x].first = hash[x].first * magic[1] + 41;
```



```

38         hash[x].second++;
39     }
40 }

```

---

#### 4.12 哈密尔顿回路（ORE 性质的图）

ORE 性质:

$$\forall x, y \in V \wedge (x, y) \notin E \quad s.t. \quad deg_x + deg_y \geq n$$

返回结果: 从顶点 1 出发的一个哈密尔顿回路

使用条件:  $n \geq 3$

---

```

1  int left[N], right[N], next[N], last[N];
2
3  void cover(int x) {
4      left[right[x]] = left[x];
5      right[left[x]] = right[x];
6  }
7
8  int adjacent(int x) {
9      for (int i = right[0]; i <= n; i = right[i]) {
10         if (graph[x][i]) {
11             return i;
12         }
13     }
14     return 0;
15 }
16
17 std::vector<int> solve() {
18     for (int i = 1; i <= n; ++i) {
19         left[i] = i - 1;
20         right[i] = i + 1;
21     }
22     int head, tail;
23     for (int i = 2; i <= n; ++i) {
24         if (graph[1][i]) {
25             head = 1;
26             tail = i;
27             cover(head);
28             cover(tail);
29             next[head] = tail;
30             break;
31         }

```

```

32     }
33     while (true) {
34         int x;
35         while (x = adjacent(head)) {
36             next[x] = head;
37             head = x;
38             cover(head);
39         }
40         while (x = adjacent(tail)) {
41             next[tail] = x;
42             tail = x;
43             cover(tail);
44         }
45         if (!graph[head][tail]) {
46             for (int i = head, j; i != tail; i = next[i]) {
47                 if (graph[head][next[i]] && graph[tail][i]) {
48                     for (j = head; j != i; j = next[j]) {
49                         last[next[j]] = j;
50                     }
51                     j = next[head];
52                     next[head] = next[i];
53                     next[tail] = i;
54                     tail = j;
55                     for (j = i; j != head; j = last[j]) {
56                         next[j] = last[j];
57                     }
58                     break;
59                 }
60             }
61         }
62         next[tail] = head;
63         if (right[0] > n) {
64             break;
65         }
66         for (int i = head; i != tail; i = next[i]) {
67             if (adjacent(i)) {
68                 head = next[i];
69                 tail = i;
70                 next[tail] = 0;
71                 break;
72             }
73         }

```

```

74     }
75     std::vector<int> answer;
76     for (int i = head; ; i = next[i]) {
77         if (i == 1) {
78             answer.push_back(i);
79             for (int j = next[i]; j != i; j = next[j]) {
80                 answer.push_back(j);
81             }
82             answer.push_back(i);
83             break;
84         }
85         if (i == tail) {
86             break;
87         }
88     }
89     return answer;
90 }

```

---

#### 4.13 必经点树

---

```

1  vector<int>G[maxn],rG[maxn],dom[maxn];
2  int n,m;
3  int dfn[maxn],rdfs[maxn],dfs_c,semi[maxn],idom[maxn],fa[maxn];
4  struct ufsets{
5      int fa[maxn],best[maxn];
6      int find(int x){
7          if(fa[x]==x)
8              return x;
9          int f=find(fa[x]);
10         if(dfn[semi[best[x]]]>dfn[semi[best[fa[x]]]])
11             best[x]=best[fa[x]];
12         fa[x]=f;
13         return f;
14     }
15     int getbest(int x){
16         find(x);
17         return best[x];
18     }
19     void init(){
20         for(int i=1;i<=n;i++)
21             fa[i]=best[i]=i;
22     }

```

```

23 }uf;
24 void init(){
25     uf.init();
26     for(int i=1;i<=n;i++){
27         semi[i]=i;
28         idom[i]=0;
29         fa[i]=0;
30         dfn[i]=rdfn[i]=0;
31     }
32     dfs_c=0;
33 }
34 void dfs(int u){
35     dfn[u]=++dfs_c;
36     rdfn[dfn[u]]=u;
37     for(int i=0;i<G[u].size();i++){
38         int v=G[u][i];
39         if(!dfn[v]){
40             fa[v]=u;
41             dfs(v);
42         }
43     }
44 }
45
46 void tarjan(){
47     for(int i=n;i>1;i--){
48         int tmp=1e9;
49         int y=rdfn[i];
50         for(int i=0;i<rG[y].size();i++){
51             int x=rG[y][i];
52             tmp=min(tmp,dfn[semi[uf.getbest(x)]]);
53         }
54         semi[y]=rdfn[tmp];
55         int x=fa[y];
56         dom[semi[y]].push_back(y);
57         uf.fa[y]=x;
58         for(int i=0;i<dom[x].size();i++){
59             int z=dom[x][i];
60             if(dfn[semi[uf.getbest(z)]]<dfn[x])
61                 idom[z]=uf.getbest(z);
62             else
63                 idom[z]=semi[z];
64         }

```

```

65         dom[x].clear();
66     }
67     semi[rdfn[1]]=1;
68     for(int i=2;i<=n;i++){
69         int x=rdfn[i];
70         if(idom[x]!=semi[x])
71             idom[x]=idom[idom[x]];
72
73     }
74     idom[rdfn[1]]=0;
75 }
76 init();
77 dfs(1);
78 tarjan();

```

---

## 5 字符串

### 5.1 模式匹配

#### 5.1.1 KMP 算法

---

```

1 void build(char *pattern) {
2     int length = (int)strlen(pattern + 1);
3     fail[0] = -1;
4     for (int i = 1, j; i <= length; ++i) {
5         for (j = fail[i - 1]; j != -1 && pattern[i] != pattern[j + 1]; j = fail[j]);
6         fail[i] = j + 1;
7     }
8 }
9
10 void solve(char *text, char *pattern) {
11     int length = (int)strlen(text + 1);
12     for (int i = 1, j; i <= length; ++i) {
13         for (j = match[i - 1]; j != -1 && text[i] != pattern[j + 1]; j = fail[j]);
14         match[i] = j + 1;
15     }
16 }
17 ///Hint: 1 - Base

```

---

### 5.1.2 扩展 KMP 算法

返回结果:

$$next_i = lcp(text, text_{i..n-1})$$

---

```
1 void solve(char *text, int length, int *next) {
2     int j = 0, k = 1;
3     for (; j + 1 < length && text[j] == text[j + 1]; j++);
4     next[0] = length - 1;
5     next[1] = j;
6     for (int i = 2; i < length; ++i) {
7         int far = k + next[k] - 1;
8         if (next[i - k] < far - i + 1) {
9             next[i] = next[i - k];
10        } else {
11            j = std::max(far - i + 1, 0);
12            for (; i + j < length && text[j] == text[i + j]; j++);
13            next[i] = j;
14            k = i;
15        }
16    }
17 }
18 /// 0 - Base
```

---

### 5.1.3 AC 自动机

---

```
1 struct Node{
2     int Next[30], fail, mark;
3 }Tree[N];
4
5 void Init(){
6     memset(Tree, 0, sizeof Tree);
7     cnt = 1;
8
9     for (int i = 1; i <= n; i++){
10         char c;
11         int now = 1;
12         scanf("%s", s + 1);
13         int Length = strlen(s + 1);
14         for (int j = 1; j <= Length; j++){
15             c = s[j];
16             if (Tree[now].Next[c - 'a']) now = Tree[now].Next[c - 'a']; else
17                 Tree[now].Next[c - 'a'] = ++ cnt, now = cnt;
```

```

18         }
19     }
20 }
21
22 void Build_Ac(){
23     int en = 0;
24     Q[0] = 1;
25     for (int fi = 0; fi <= en; fi++){
26         int now = Q[fi];
27         for (int next = 0; next < 26; next++){
28             if (Tree[now].Next[next])
29             {
30                 int k = Tree[now].Next[next];
31                 if (now == 1) Tree[k].fail = 1; else
32                 {
33                     int h = Tree[now].fail;
34                     while (h && !Tree[h].Next[next]) h = Tree[h].fail;
35                     if (!h) Tree[k].fail = 1;
36                     else Tree[k].fail = Tree[h].Next[next];
37                 }
38                 Q[++ en] = k;
39             }
40         }
41     }
42
43     /// Hints : when not match , fail = 1

```

---

## 5.2 后缀三姐妹

### 5.2.1 后缀数组

---

```

1 struct Sa{
2     int heap[N],s[N],sa[N],r[N],tr[N],sec[N],m,cnt;
3     int h[19][N];
4
5     void Prep(){
6         for (int i=1; i<=m; i++) heap[i]=0;
7         for (int i=1; i<=n; i++) heap[s[i]]++;
8         for (int i=2; i<=m; i++) heap[i]+=heap[i-1];
9         for (int i=n; i>=1; i--) sa[heap[s[i]]--]=i;
10        r[sa[1]]=1; cnt=1;
11        for (int i=2; i<=n; i++){

```

```

12         if (s[sa[i]]!=s[sa[i-1]]) cnt++;
13         r[sa[i]]=cnt;
14     }
15     m=cnt;
16 }
17
18 void Suffix(){
19     int j=1;
20     while (cnt<n){
21         cnt=0;
22         for (int i=n-j+1; i<=n; i++) sec[++cnt]=i;
23         for (int i=1; i<=n; i++) if (sa[i]>j)
24             sec[++cnt]=sa[i]-j;
25         for (int i=1; i<=n; i++) tr[i]=r[sec[i]];
26         for (int i=1; i<=m; i++) heap[i]=0;
27         for (int i=1; i<=n; i++) heap[tr[i]]++;
28         for (int i=2; i<=m; i++) heap[i]+=heap[i-1];
29         for (int i=n; i>=1; i--)
30             sa[heap[tr[i]]--]=sec[i];
31         tr[sa[1]]=1; cnt=1;
32         for (int i=2; i<=n; i++){
33             if ((r[sa[i]]!=r[sa[i-1]]) || (r[sa[i]+j]!=r[sa[i-1]+j]))
34                 cnt++;
35             tr[sa[i]]=cnt;
36         }
37         for (int i=1; i<=n; i++) r[i]=tr[i];
38         m=cnt; j=j+j;
39     }
40 }
41
42 void Calc(){
43     int k=0;
44     for (int i=1; i<=n; i++){
45         if (r[i]==1) continue;
46         int j=sa[r[i]-1];
47         while ((i+k<=n) && (j+k<=n) && (s[i+k]==s[j+k])) k++;
48         h[0][r[i]]=k;
49         if (k) k--;
50     }
51     for (int i=1; i<19; i++)
52         for (int j=1; j+(1<<i)-1<=n; j++)
53             h[i][j]=min(h[i-1][j],h[i-1][j + (1<<(i - 1)) + 1]);

```



```

54     }
55
56     int Query(int L,int R){
57         L=r[L], R=r[R];
58         if (L>R) swap(L,R);
59         L++;
60         int l0 = Lg[R-L+1];
61         return min(h[l0][L],h[l0][R-(1<<l0)+1]);
62     }
63
64     void Work(){
65         Prep(); Suffix(); Calc();
66     }
67 }P,S;
68
69 /// Hints : 1 - Base

```

---

### 5.2.2 后缀数组 (dc3)

---

```

1  ///`DC3` 待排序的字符串放在 r 数组中, 从 r[0] 到 r[n-1], 长度为 n, 且最大值小于 m.`
2  ///`约定除 r[n-1] 外所有的 r[i] 都大于 0, r[n-1]=0.`
3  ///`函数结束后, 结果放在 sa 数组中, 从 sa[0] 到 sa[n-1]`.`
4  ///`r 必须开长度乘 3`
5  #define maxn 10000
6  #define F(x) ((x)/3+((x)%3==1?0:tb))
7  #define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
8
9  int wa[maxn],wb[maxn],wv[maxn],wss[maxn];
10 int s[maxn*3],sa[maxn*3];
11 int c0(int *r,int a,int b)
12 {
13     return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
14 }
15 int c12(int k,int *r,int a,int b)
16 {
17     if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
18     else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
19 }
20 void sort(int *r,int *a,int *b,int n,int m)
21 {
22     int i;
23     for(i=0;i<n;i++) wv[i]=r[a[i]];

```

```

24     for(i=0;i<m;i++) wss[i]=0;
25     for(i=0;i<n;i++) wss[wv[i]]++;
26     for(i=1;i<m;i++) wss[i]+=wss[i-1];
27     for(i=n-1;i>=0;i--) b[--wss[wv[i]]]=a[i];
28 }
29 void dc3(int *r,int *sa,int n,int m)
30 {
31     int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
32     r[n]=r[n+1]=0;
33     for(i=0;i<n;i++)
34         if(i%3!=0) wa[tbc++]=i;
35     sort(r+2,wa,wb,tbc,m);
36     sort(r+1,wb,wa,tbc,m);
37     sort(r,wa,wb,tbc,m);
38     for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
39         rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
40     if (p<tbc) dc3(rn,san,tbc,p);
41     else for (i=0;i<tbc;i++) san[rn[i]]=i;
42     for (i=0;i<tbc;i++)
43         if(san[i]<tb) wb[ta++]=san[i]*3;
44     if(n%3==1) wb[ta++]=n-1;
45     sort(r,wb,wa,ta,m);
46     for(i=0;i<tbc;i++)
47         wv[wb[i]=G(san[i])]=i;
48     for(i=0,j=0,p=0;i<ta && j<tbc;p++)
49         sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
50     for(;i<ta;p++) sa[p]=wa[i++];
51     for(;j<tbc;p++) sa[p]=wb[j++];
52 }
53
54 int main(){
55     int n,m=0;
56     scanf("%d",&n);
57     for (int i=0;i<n;i++) scanf("%d",&s[i]),s[i]++,m=max(s[i]+1,m);
58     printf("%d\n",m);
59     s[n++]=0;
60     dc3(s,sa,n,m);
61     for (int i=0;i<n;i++) printf("%d ",sa[i]);printf("\n");
62 }

```

---

### 5.2.3 后缀自动机-多串 LCS

对一个串建后缀自动机，其他串在上面匹配，因为是求所有串的公共子串，所以每个点记录每个串最长匹配长度的最小值，最后找到所有点中最长的一个即可。一个注意事项就是，当走到一个点时，还要更新它的 `parent` 树上的祖先的匹配长度，数组开两倍啦啦啦！

---

```
1 struct Node{
2     int len, fail;
3     int To[30];
4 }T[N];
5 int Lst, Root, tot, ans;
6 char s[N];
7 int Len[N], Ans[N], Ord[N];
8 void Add(int x, int l){
9     int Nt = ++tot, p = Lst;
10    T[Nt].len = l;
11    for (;p && !T[p].To[x]; p = T[p].fail) T[p].To[x] = Nt;
12    if (!p) T[Nt].fail = Root; else
13    if (T[T[p].To[x]].len == T[p].len + 1) T[Nt].fail = T[p].To[x];
14    else{
15        int q = ++tot, qt = T[p].To[x];
16        T[q] = T[qt];
17        T[q].len = T[p].len + 1;
18        T[qt].fail = T[Nt].fail = q;
19        for (;p && T[p].To[x] == qt; p = T[p].fail) T[p].To[x] = q;
20    }
21    Lst = Nt;
22 }
23 bool cmp(int a, int b){
24     return T[a].len < T[b].len;
25 }
26 int main(){
27     scanf("%s", s + 1);
28     int n = strlen(s + 1);
29     ans = n;
30     Root = tot = Lst = 1;
31     for (int i = 1; i <= n; i++)
32         Add(s[i] - 'a' + 1, i);
33     for (int i = 1; i <= tot; i++)
34         Ord[i] = i;
35     sort(Ord + 1, Ord + tot + 1, cmp);
36     for (int i = 1; i <= tot; i++)
37         Ans[i] = T[i].len;
```

```

38     bool flag = 0;
39     while (scanf("%s", s + 1) != EOF){
40         flag = 1;
41         int n = strlen(s + 1);
42         int p = Root, len = 0;
43         for (int i = 1; i <= tot; i++) Len[i] = 0;
44         for (int i = 1; i <= n; i++){
45             int x = s[i] - 'a' + 1;
46             if (T[p].To[x]) len++, p = T[p].To[x];
47             else {
48                 while (p && !T[p].To[x]) p = T[p].fail;
49                 if (!p) p = Root, len = 0;
50                 else len = T[p].len + 1, p = T[p].To[x];
51             }
52             Len[p] = max(Len[p], len);
53         }
54         for (int i = tot; i >= 1; i--){
55             int Cur = Ord[i];
56             Ans[Cur] = min(Ans[Cur], Len[Cur]);
57             if (Len[Cur] && T[Cur].fail)
58                 Len[T[Cur].fail] = T[T[Cur].fail].len;
59         }
60     }
61     if (flag){
62         ans = 0;
63         for (int i = 1; i <= tot; i++){
64             ans = max(ans, Ans[i]);
65         }
66     }
67     printf("%d\n", ans);
68     return 0;
69 }

```

---

#### 5.2.4 后缀自动机-各长度字串出现次数最大值

给一个字符串  $S$ ，令  $F(x)$  表示  $S$  的所有长度为  $x$  的子串中，出现次数的最大值。构建字符串的自动机，对于每个节点， $\text{right}$  集合大小就是出现次数， $\text{maxs}$  就是它代表的最长长度，那么我们用  $|\text{right}(x)|$  去更新  $f[\text{maxs}[x]]$  的值，最后从大到小用  $f[i]$  去更新  $f[i-1]$  的值即可

---

```

1 struct Node{
2     int len, fail;
3     int To[30];

```

```

4  }T[N];
5  int Lst, Root, tot, n;
6  char s[N];
7  int Ord[N], Ans[N], Ways[N], heap[N];
8  void Add(int x, int l){
9      int Nt = ++tot, p = Lst;
10     T[Nt].len = 1;
11     for (;p && !T[p].To[x]; p = T[p].fail) T[p].To[x] = Nt;
12     if (!p) T[Nt].fail = Root; else
13     if (T[T[p].To[x]].len == T[p].len + 1) T[Nt].fail = T[p].To[x];
14     else{
15         int q = ++tot, qt = T[p].To[x];
16         T[q] = T[qt];
17         T[q].len = T[p].len + 1;
18         T[qt].fail = T[Nt].fail = q;
19         for (;p && T[p].To[x] == qt; p = T[p].fail) T[p].To[x] = q;
20     }
21     Lst = Nt;
22 }
23 bool cmp(int a, int b){
24     return T[a].len < T[b].len;
25 }
26 void sort(){
27     for (int i = 1; i <= tot; i++) heap[T[i].len]++;
28     for (int i = 1; i <= n; i++) heap[i] += heap[i-1];
29     for (int i = 1; i <= tot; i++) Ord[heap[T[i].len]--]=i;
30 }
31 int main(){
32     scanf("%s", s + 1);
33     n = strlen(s + 1);
34     Root = tot = Lst = 1;
35     for (int i = 1; i <= n; i++)
36         Add(s[i] - 'a' + 1, i);
37     sort();
38     memset(Ways, 0, sizeof(Ways));
39     for (int i = 1, p = Root; i <= n; i++)
40         p = T[p].To[s[i] - 'a' + 1], Ways[p] = 1;
41     for (int i = tot; i >= 1; i--){
42         int Cur = Ord[i];
43         if (T[Cur].fail == 0) continue;
44         Ways[T[Cur].fail] += Ways[Cur];
45     }

```

```

46     for (int i = 1; i <= tot; i++)
47         Ans[T[i].len] = max(Ans[T[i].len], Ways[i]);
48     for (int i = n; i >= 1; i--)
49         Ans[i] = max(Ans[i + 1], Ans[i]);
50     for (int i = 1; i <= n; i++)
51         printf("%d\n", Ans[i]);
52     return 0;
53 }

```

---

### 5.2.5 后缀自动机-两串 LCS

---

```

1  struct node{
2      int len, fail;
3      int To[27];
4  }T[N];
5  char a[N], b[N];
6  int Lst, Root, tot;
7  void add(int x, int l){
8      int Nt = ++tot, p = Lst;
9      T[Nt].len = l;
10     for (; p && !T[p].To[x]; p = T[p].fail) T[p].To[x] = Nt;
11     if (!p) T[Nt].fail = Root;
12     else
13         if (T[T[p].To[x]].len == T[p].len + 1) T[Nt].fail = T[p].To[x];
14         else{
15             int q = ++tot, qt = T[p].To[x];
16             T[q] = T[qt];
17             T[q].len = T[p].len + 1;
18             T[qt].fail = T[Nt].fail = q;
19             for (; p && T[p].To[x] == qt; p = T[p].fail) T[p].To[x] = q;
20         }
21     Lst = Nt;
22 }
23 int main(){
24     while (scanf("%s%s", a + 1, b + 1) == 2){
25         int n = strlen(a + 1);
26         Lst = Root = tot = 1;
27         for (int i = 1; i <= n; i++)
28             add(a[i] - 'a' + 1, i);
29         int m = strlen(b + 1);
30         int p = Root, len = 0;
31         int Ans = 0;

```

```

32     for (int i = 1; i <= m; i++){
33         int x = b[i] - 'a' + 1;
34         if (T[p].To[x]) len++, p = T[p].To[x];
35         else {
36             while (p && !T[p].To[x]) p = T[p].fail;
37             if (!p) p = Root, len = 0;
38             else len = T[p].len + 1, p = T[p].To[x];
39         }
40         if (len > Ans) Ans = len;
41     }
42     printf("%d\n", Ans);
43     for (int i = 1; i <= tot; i++){
44         T[i].len = T[i].fail = 0;
45         for (int j = 1; j <= 26; j++)
46             T[i].To[j] = 0;
47     }
48 }
49 return 0;
50 }
51 //HintsoSAM + Longest common subsequence

```

---

## 5.3 回文三兄弟

### 5.3.1 马拉车

---

```

1 void Manacher(){
2     R[1] = 1;
3     for (int i = 2, j = 1; i <= length; i++){
4         if (j + R[j] <= i){
5             R[i] = 0;
6         } else {
7             R[i] = min(R[j * 2 - i], j + R[j] - i);
8         }
9         while (i - R[i] >= 1 && i + R[i] <= length
10             && text[i - R[i]] == text[i + R[i]]){
11             R[i]++;
12         }
13         if (i + R[i] > j + R[j]){
14             j = i;
15         }
16     }
17 }

```

```

18     length = 0;
19     int n = strlen(s + 1);
20     for (int i = 1; i <= n; i++){
21         text[++length] = '*';
22         text[++length] = s[i];
23     }
24     text[++length] = '*';
25     /// Hints: 1 - Base

```

---

### 5.3.2 回文树 (lyx)

---

```

1  const int N = 400005;
2
3  char s[N];
4  int Len;
5
6  struct Palindromic_Tree {
7      int next[N][27];
8      int fail[N];
9      int cnt[N];
10     int num[N];
11     int len[N];
12     char S[N];
13     int last;
14     int n;
15     int p;
16
17     int newnode(int l)
18     {
19         for(int i = 1; i <= 26; i++) next[p][i] = 0;
20         cnt[p] = 0;
21         num[p] = 0;
22         len[p] = l;
23         fail[p] = 0;
24         return p++;
25     }
26     void init()
27     {
28         p = 0;
29         newnode(0);
30         newnode(-1);
31         last = 0;

```



```

32     n = 0;
33     S[n] = -1;
34     fail[0] = 1;
35 }
36 int get_fail(int x)
37 {
38     while (S[n - len[x] - 1] != S[n]) x = fail[x];
39     return x;
40 }
41 void add(char c, int pos)
42 {
43     c = c - 'a' + 1;
44     S[++ n] = c ;
45     int cur = get_fail(last);
46     if (!next[cur][c])
47     {
48         int now = newnode(len[cur] + 2);
49         fail[now] = next[get_fail(fail[cur])][c];
50         next[cur][c] = now;
51         num[now] = num[fail[now]] + 1;
52     }
53     last = next[cur][c] ;
54     cnt[last]++;
55 }
56 void count()
57 {
58     for (int i = p - 1 ; i >= 0 ; -- i) cnt[fail[i]] += cnt[i] ;
59 }
60 }T;
61
62 Len = strlen(s + 1);
63
64 T.init();
65 for (int i = 1; i <= Len; i++)
66     T.add(s[i], i);
67 T.count();

```

---

### 5.3.3 回文自动机 (zky)

---

```

1 struct PAM{
2     int tot,last,str[maxn],nxt[maxn][26],n;
3     int len[maxn],suf[maxn],cnt[maxn];

```

```

4      int newNode(int l){
5          len[tot]=1;
6          return tot++;
7      }
8      void init(){
9          tot=0;
10         newNode(0); // tree 0 is node 0
11         newNode(-1); // tree -1 is node 1
12         str[0]=-1;
13         suf[0]=1;
14     }
15     int find(int x){
16         while(str[n-len[x]-1]!=str[n])x=suf[x];
17         return x;
18     }
19     void add(int c){
20         str[++n]=c;
21         int u=find(last);
22         if(!nxt[u][c]){
23             int v=newNode(len[u]+2);
24             suf[v]=nxt[find(suf[u])][c];
25             nxt[u][c]=v;
26         }last=nxt[u][c];
27         cnt[last]++;
28     }
29     void count(){
30         for(int i=tot-1;i>=0;i--)cnt[suf[i]]+=cnt[i];
31     }
32 }P;
33 int main(){
34     P.init();
35     for(int i=0;i<n;i++)
36         P.add(s[i]-'a');
37     P.count();

```

---

## 5.4 循环串最小表示

```

1 string sol(char *s){
2     int n=strlen(s);
3     int i=0,j=1,k=0,p;
4     while(i<n&&j<n&&k<n){
5         int t=s[(i+k)%n]-s[(j+k)%n];

```

```

6         if(t==0)k++;
7         else if(t<0)j+=k+1,k=0;
8         else i+=k+1,k=0;
9         if(i==j)j++;
10    }p=min(i,j);
11    string S;
12    for(int i=p;i<p+n;i++)S.push_back(s[i%n]);
13    return S;
14 }

```

---

## 6 计算几何

### 6.1 二维基础

#### 6.1.1 点类

---

```

1  int sgn(double x){return (x>eps)-(x<-eps);}
2  int sgn(double a,double b){return sgn(a-b);}
3  double sqr(double x){return x*x;}
4  struct P{
5      double x,y;
6      P(){}
7      P(double x,double y):x(x),y(y){}
8      double len2(){
9          return sqr(x)+sqr(y);
10     }
11     double len(){
12         return sqrt(len2());
13     }
14     void print(){
15         printf("%.3f,%.3f\n",x,y);
16     }
17     P turn90(){return P(-y,x);}
18     P norm(){return P(x/len(),y/len());}
19 };
20 bool operator==(P a,P b){
21     return !sgn(a.x-b.x) and !sgn(a.y-b.y);
22 }
23 P operator+(P a,P b){
24     return P(a.x+b.x,a.y+b.y);
25 }
26 P operator-(P a,P b){

```

```

27         return P(a.x-b.x,a.y-b.y);
28     }
29     P operator*(P a,double b){
30         return P(a.x*b,a.y*b);
31     }
32     P operator/(P a,double b){
33         return P(a.x/b,a.y/b);
34     }
35     double operator^(P a,P b){
36         return a.x*b.x + a.y*b.y;
37     }
38     double operator*(P a,P b){
39         return a.x*b.y - a.y*b.x;
40     }
41     double det(P a,P b,P c){
42         return (b-a)*(c-a);
43     }
44     double dis(P a,P b){
45         return (b-a).len();
46     }
47     double Area(vector<P>poly){
48         double ans=0;
49         for(int i=1;i<poly.size();i++)
50             ans+=(poly[i]-poly[0])*(poly[(i+1)%poly.size()]-poly[0]);
51         return fabs(ans)/2;
52     }
53     struct L{
54         P a,b;
55         L(){}
56         L(P a,P b):a(a),b(b){}
57         P v(){return b-a;}
58     };
59     bool onLine(P p,L l){
60         return sgn((l.a-p)*(l.b-p))==0;
61     }
62     bool onSeg(P p,L s){
63         return onLine(p,s) and sgn((s.b-s.a)^(p-s.a))>=0 and sgn((s.a-s.b)^(p-s.b))>=0;
64     }
65     bool parallel(L l1,L l2){
66         return sgn(l1.v()*l2.v())==0;
67     }
68     P intersect(L l1,L l2){

```

```

69     double s1=det(l1.a,l1.b,l2.a);
70     double s2=det(l1.a,l1.b,l2.b);
71     return (l2.a*s2-l2.b*s1)/(s2-s1);
72 }
73 P project(P p,L l){
74     return l.a+l.v()*((p-l.a)^l.v())/l.v().len2();
75 }
76 double dis(P p,L l){
77     return fabs((p-l.a)*l.v())/l.v().len();
78 }

```

---

### 6.1.2 凸包

```

1 vector<P> convex(vector<P>p){
2     sort(p.begin(),p.end());
3     vector<P>ans,S;
4     for(int i=0;i<p.size();i++){
5         while(S.size())>=2
6             && sgn(det(S[S.size()-2],S.back(),p[i]))<=0)
7             S.pop_back();
8         S.push_back(p[i]);
9     }//dw
10    ans=S;
11    S.clear();
12    for(int i=(int)p.size()-1;i>=0;i--){
13        while(S.size())>=2
14            && sgn(det(S[S.size()-2],S.back(),p[i]))<=0)
15            S.pop_back();
16        S.push_back(p[i]);
17    }//up
18    for(int i=1;i+1<S.size();i++)
19        ans.push_back(S[i]);
20    return ans;
21 }

```

---

### 6.1.3 半平面交

```

1 struct P{
2     int quad() const { return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) >= 0);}
3 };
4 struct L{

```

```

5         bool onLeft(const P &p) const { return sgn((b - a)*( p - a)) > 0; }
6         L push() const{ // push out eps
7             const double eps = 1e-10;
8             P delta = (b - a).turn90().norm() * eps;
9             return L(a - delta, b - delta);
10        }
11    };
12    bool sameDir(const L &l0, const L &l1) {
13        return parallel(l0, l1) && sgn((l0.b - l0.a)^(l1.b - l1.a)) == 1;
14    }
15    bool operator < (const P &a, const P &b) {
16        if (a.quad() != b.quad())
17            return a.quad() < b.quad();
18        else
19            return sgn((a*b)) > 0;
20    }
21    bool operator < (const L &l0, const L &l1) {
22        if (sameDir(l0, l1))
23            return l1.onLeft(l0.a);
24        else
25            return (l0.b - l0.a) < (l1.b - l1.a);
26    }
27    bool check(const L &u, const L &v, const L &w) {
28        return w.onLeft(intersect(u, v));
29    }
30    vector<P> intersection(vector<L> &l) {
31        sort(l.begin(), l.end());
32        deque<L> q;
33        for (int i = 0; i < (int)l.size(); ++i) {
34            if (i && sameDir(l[i], l[i - 1])) {
35                continue;
36            }
37            while (q.size() > 1
38                && !check(q[q.size() - 2], q[q.size() - 1], l[i]))
39                q.pop_back();
40            while (q.size() > 1
41                && !check(q[1], q[0], l[i]))
42                q.pop_front();
43            q.push_back(l[i]);
44        }
45        while (q.size() > 2
46            && !check(q[q.size() - 2], q[q.size() - 1], q[0]))

```

```

47         q.pop_back();
48     while (q.size() > 2
49         && !check(q[1], q[0], q[q.size() - 1]))
50         q.pop_front();
51     vector<P> ret;
52     for (int i = 0; i < (int)q.size(); ++i)
53         ret.push_back(intersect(q[i], q[(i + 1) % q.size()]));
54     return ret;
55 }

```

---

#### 6.1.4 最近点对

```

1  bool byY(P a,P b){return a.y<b.y;}
2  LL solve(P *p,int l,int r){
3      LL d=1LL<<62;
4      if(l==r)
5          return d;
6      if(l+1==r)
7          return dis2(p[l],p[r]);
8      int mid=(l+r)>>1;
9      d=min(solve(l,mid),d);
10     d=min(solve(mid+1,r),d);
11     vector<P>tmp;
12     for(int i=l;i<=r;i++)
13         if(sqr(p[mid].x-p[i].x)<=d)
14             tmp.push_back(p[i]);
15     sort(tmp.begin(),tmp.end(),byY);
16     for(int i=0;i<tmp.size();i++)
17         for(int j=i+1;j<tmp.size()&&j-i<10;j++)
18             d=min(d,dis2(tmp[i],tmp[j]));
19     return d;
20 }

```

---

#### 6.1.5 最小圆覆盖

```

1  struct line{
2      point p,v;
3  };
4  point Rev(point v){return point(-v.y,v.x);}
5  point operator*(line A,line B){
6      point u=B.p-A.p;

```

```

7         double t=(B.v*u)/(B.v*A.v);
8         return A.p+A.v*t;
9     }
10    point get(point a,point b){
11        return (a+b)/2;
12    }
13    point get(point a,point b,point c){
14        if(a==b)return get(a,c);
15        if(a==c)return get(a,b);
16        if(b==c)return get(a,b);
17        line ABO=(line){(a+b)/2,Rev(a-b)};
18        line BCO=(line){(c+b)/2,Rev(b-c)};
19        return ABO*BCO;
20    }
21    int main(){
22        scanf("%d",&n);
23        for(int i=1;i<=n;i++)scanf("%lf%lf",&p[i].x,&p[i].y);
24        random_shuffle(p+1,p+1+n);
25        O=p[1];r=0;
26        for(int i=2;i<=n;i++){
27            if(dis(p[i],O)<r+1e-6)continue;
28            O=get(p[1],p[i]);r=dis(O,p[i]);
29            for(int j=1;j<i;j++){
30                if(dis(p[j],O)<r+1e-6)continue;
31                O=get(p[i],p[j]);r=dis(O,p[i]);
32                for(int k=1;k<j;k++){
33                    if(dis(p[k],O)<r+1e-6)continue;
34                    O=get(p[i],p[j],p[k]);r=dis(O,p[i]);
35                }
36            }
37        }printf("%.21f %.21f %.21f\n",O.x,O.y,r);
38        return 0;
39    }s

```

---

### 6.1.6 凸包快速询问

---

```

1  /*
2      给定凸包， $\log n$  内完成各种询问，具体操作有：
3      1. 判定一个点是否在凸包内
4      2. 询问凸包外的点到凸包的两个切点
5      3. 询问一个向量关于凸包的切点
6      4. 询问一条直线和凸包的交点

```



```

7     INF 为坐标范围，需要定义点类大于号
8     改成实数只需修改 sign 函数，以及把 Long Long 改为 double 即可
9     构造函数时传入凸包要求无重点，面积非空，以及 pair(x,y) 的最小点放在第一个
10  */
11  const int INF = 1000000000;
12  struct Convex
13  {
14      int n;
15      vector<Point> a, upper, lower;
16      Convex(vector<Point> _a) : a(_a) {
17          n = a.size();
18          int ptr = 0;
19          for(int i = 1; i < n; ++ i) if (a[ptr] < a[i]) ptr = i;
20          for(int i = 0; i <= ptr; ++ i) lower.push_back(a[i]);
21          for(int i = ptr; i < n; ++ i) upper.push_back(a[i]);
22          upper.push_back(a[0]);
23      }
24      int sign(long long x) { return x < 0 ? -1 : x > 0; }
25      pair<long long, int> get_tangent(vector<Point> &convex, Point vec) {
26          int l = 0, r = (int)convex.size() - 2;
27          for( ; l + 1 < r; ) {
28              int mid = (l + r) / 2;
29              if (sign((convex[mid + 1] - convex[mid]).det(vec)) > 0) r = mid;
30              else l = mid;
31          }
32          return max(make_pair(vec.det(convex[r]), r)
33              , make_pair(vec.det(convex[0]), 0));
34      }
35      void update_tangent(const Point &p, int id, int &i0, int &i1) {
36          if ((a[i0] - p).det(a[id] - p) > 0) i0 = id;
37          if ((a[i1] - p).det(a[id] - p) < 0) i1 = id;
38      }
39      void binary_search(int l, int r, Point p, int &i0, int &i1) {
40          if (l == r) return;
41          update_tangent(p, l % n, i0, i1);
42          int s1 = sign((a[l % n] - p).det(a[(l + 1) % n] - p));
43          for( ; l + 1 < r; ) {
44              int mid = (l + r) / 2;
45              int smid = sign((a[mid % n] - p).det(a[(mid + 1) % n] - p));
46              if (smid == s1) l = mid;
47              else r = mid;
48          }

```

```

49         update_tangent(p, r % n, i0, i1);
50     }
51     int binary_search(Point u, Point v, int l, int r) {
52         int s1 = sign((v - u).det(a[l % n] - u));
53         for( ; l + 1 < r; ) {
54             int mid = (l + r) / 2;
55             int smid = sign((v - u).det(a[mid % n] - u));
56             if (smid == s1) l = mid;
57             else r = mid;
58         }
59         return l % n;
60     }
61     // 判定点是否在凸包内, 在边界返回 true
62     bool contain(Point p) {
63         if (p.x < lower[0].x || p.x > lower.back().x) return false;
64         int id = lower_bound(lower.begin(), lower.end()
65             , Point(p.x, -INF)) - lower.begin();
66         if (lower[id].x == p.x) {
67             if (lower[id].y > p.y) return false;
68         } else if ((lower[id - 1] - p).det(lower[id] - p) < 0) return false;
69         id = lower_bound(upper.begin(), upper.end(), Point(p.x, INF)
70             , greater<Point>()) - upper.begin();
71         if (upper[id].x == p.x) {
72             if (upper[id].y < p.y) return false;
73         } else if ((upper[id - 1] - p).det(upper[id] - p) < 0) return false;
74         return true;
75     }
76     // 求点 p 关于凸包的两个切点, 如果在凸包外则有序返回编号
77     // 共线的多个切点返回任意一个, 否则返回 false
78     bool get_tangent(Point p, int &i0, int &i1) {
79         if (contain(p)) return false;
80         i0 = i1 = 0;
81         int id = lower_bound(lower.begin(), lower.end(), p) - lower.begin();
82         binary_search(0, id, p, i0, i1);
83         binary_search(id, (int)lower.size(), p, i0, i1);
84         id = lower_bound(upper.begin(), upper.end(), p
85             , greater<Point>()) - upper.begin();
86         binary_search((int)lower.size() - 1, (int)lower.size() - 1 + id, p, i0, i1);
87         binary_search((int)lower.size() - 1 + id
88             , (int)lower.size() - 1 + (int)upper.size(), p, i0, i1);
89         return true;
90     }

```

```

91 // 求凸包上和向量 vec 叉积最大的点，返回编号，共线的多个切点返回任意一个
92 int get_tangent(Point vec) {
93     pair<long long, int> ret = get_tangent(upper, vec);
94     ret.second = (ret.second + (int)lower.size() - 1) % n;
95     ret = max(ret, get_tangent(lower, vec));
96     return ret.second;
97 }
98 // 求凸包和直线 u,v 的交点，如果无严格相交返回 false.
99 //如果有则是和 (i,next(i)) 的交点，两个点无序，交在点上不确定返回前后两条线段其中之一
100 bool get_intersection(Point u, Point v, int &i0, int &i1) {
101     int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
102     if (sign((v - u).det(a[p0] - u)) * sign((v - u).det(a[p1] - u)) < 0) {
103         if (p0 > p1) swap(p0, p1);
104         i0 = binary_search(u, v, p0, p1);
105         i1 = binary_search(u, v, p1, p0 + n);
106         return true;
107     } else {
108         return false;
109     }
110 }
111 };

```

---

## 6.2 多边形

### 6.2.1 判断点在多边形内部

---

```

1 bool InPoly(P p,vector<P>poly){
2     int cnt=0;
3     for(int i=0;i<poly.size();i++){
4         P a=poly[i],b=poly[(i+1)%poly.size()];
5         if(OnLine(p,L(a,b)))
6             return false;
7         int x=sgn(det(a,p,b));
8         int y=sgn(a.y-p.y);
9         int z=sgn(b.y-p.y);
10        cnt+=(x>0&&y<=0&&z>0);
11        cnt-=(x<0&&z<=0&&y>0);
12    }
13    return cnt;
14 }

```

---

## 7 其他

### 7.1 斯坦纳树

---

```
1 priority_queue<pair<int, int> > Q;
2
3 // m is key point
4 // n is all point
5
6 for (int s = 0; s < (1 << m); s++){
7     for (int i = 1; i <= n; i++){
8         if (id[i]) continue;
9         for (int s0 = 0; s0 < s; s0++){
10             if ( (s0 & s) == s0 ){
11                 f[s][i] = min(f[s][i], f[s0][i] + f[s - s0][i]);
12             }
13         }
14         for (int i = 1; i <= n; i++) vis[i] = 0;
15     while (!Q.empty()) Q.pop();
16     for (int i = 1; i <= n; i++){
17         if (id[i]) continue;
18         Q.push(mp(-f[s][i], i));
19     }
20     while (!Q.empty()){
21         while (!Q.empty() && Q.top().first != -f[s][Q.top().second]) Q.pop();
22         if (Q.empty()) break;
23         int Cur = Q.top().second; Q.pop();
24         for (int p = g[Cur]; p; p = nxt[p]){
25             int y = adj[p];
26             if ( f[s][y] > f[s][Cur] + 1){
27                 f[s][y] = f[s][Cur] + 1;
28                 Q.push(mp(-f[s][y], y));
29             }
30         }
31     }
32 }
```

---

### 7.2 无敌的读入优化

---

```
1 namespace Reader {
2     const int L = (1 << 20) + 5;
3     char buffer[L], *S, *T;
```

```

4      __inline bool getchar(char &ch) {
5          if (S == T) {
6              T = (S = buffer) + fread(buffer, 1, L, stdin);
7              if (S == T) {
8                  ch = EOF;
9                  return false;
10             }
11         }
12         ch = *S++;
13         return true;
14     }
15     __inline bool getint(int &x) {
16         char ch;
17         for (; getchar(ch) && (ch < '0' || ch > '9'); );
18         if (ch == EOF) return false;
19         x = ch - '0';
20         for (; getchar(ch), ch >= '0' && ch <= '9'; )
21             x = x * 10 + ch - '0';
22         return true;
23     }
24 }
25 Reader::getint(x);
26 Reader::getint(y);

```

---

### 7.3 最小树形图

---

```

1  const int maxn=1100;
2
3  int n,m , g[maxn][maxn] , used[maxn] , pass[maxn] , eg[maxn] , more , queue[maxn];
4
5  void combine (int id , int &sum ) {
6      int tot = 0 , from , i , j , k ;
7      for ( ; id!=0 && !pass[ id ] ; id=eg[id] ) {
8          queue[tot++]=id ; pass[id]=1;
9      }
10     for ( from=0; from<tot && queue[from]!=id ; from++);
11     if ( from==tot ) return ;
12     more = 1 ;
13     for ( i=from ; i<tot ; i++) {
14         sum+=g[eg[queue[i]]][queue[i]] ;
15         if ( i!=from ) {
16             used[queue[i]]=1;

```

```

17         for ( j = 1 ; j <= n ; j++) if ( !used[j] )
18             if ( g[queue[i]][j]<g[id][j] ) g[id][j]=g[queue[i]][j] ;
19     }
20 }
21 for ( i=1; i<=n ; i++) if ( !used[i] && i!=id ) {
22     for ( j=from ; j<tot ; j++){
23         k=queue[j];
24         if ( g[i][id]>g[i][k]-g[eg[k]][k] ) g[i][id]=g[i][k]-g[eg[k]][k];
25     }
26 }
27 }
28
29 int mdst( int root ) { // return the total length of MDST
30     int i , j , k , sum = 0 ;
31     memset ( used , 0 , sizeof ( used ) ) ;
32     for ( more =1; more ; ) {
33         more = 0 ;
34         memset (eg,0,sizeof(eg)) ;
35         for ( i=1 ; i <= n ; i ++ ) if ( !used[i] && i!=root ) {
36             for ( j=1 , k=0 ; j <= n ; j ++ ) if ( !used[j] && i!=j )
37                 if ( k==0 || g[j][i] < g[k][i] ) k=j ;
38             eg[i] = k ;
39         }
40         memset(pass,0,sizeof(pass));
41         for ( i=1; i<=n ; i++) if ( !used[i] && !pass[i] && i!= root ) combine ( i
42     )
43     for ( i =1; i<=n ; i ++ ) if ( !used[i] && i!= root ) sum+=g[eg[i]][i];
44     return sum ;
45 }

```

---

## 7.4 DLX

```

1  int n,m,K;
2  struct DLX{
3      int L[maxn],R[maxn],U[maxn],D[maxn];
4      int sz,col[maxn],row[maxn],s[maxn],H[maxn];
5      bool vis[233];
6      int ans[maxn],cnt;
7      void init(int m){
8          for(int i=0;i<=m;i++){
9              L[i]=i-1;R[i]=i+1;
10             U[i]=D[i]=i;s[i]=0;

```

```

11         }
12         memset(H, -1, sizeof H);
13         L[0]=m; R[m]=0; sz=m+1;
14     }
15     void Link(int r, int c){
16         U[sz]=c; D[sz]=D[c]; U[D[c]]=sz; D[c]=sz;
17         if(H[r]<0) H[r]=L[sz]=R[sz]=sz;
18         else{
19             L[sz]=H[r]; R[sz]=R[H[r]];
20             L[R[H[r]]]=sz; R[H[r]]=sz;
21         }
22         s[c]++; col[sz]=c; row[sz]=r; sz++;
23     }
24     void remove(int c){
25         for(int i=D[c]; i!=c; i=D[i])
26             L[R[i]]=L[i], R[L[i]]=R[i];
27     }
28     void resume(int c){
29         for(int i=U[c]; i!=c; i=U[i])
30             L[R[i]]=R[L[i]]=i;
31     }
32     int A(){
33         int res=0;
34         memset(vis, 0, sizeof vis);
35         for(int i=R[0]; i; i=R[i]) if(!vis[i]){
36             vis[i]=1; res++;
37             for(int j=D[i]; j!=i; j=D[j])
38                 for(int k=R[j]; k!=j; k=R[k])
39                     vis[col[k]]=1;
40         }
41         return res;
42     }
43     void dfs(int d, int &ans){
44         if(R[0]==0){ ans=min(ans, d); return; }
45         if(d+A()>=ans) return;
46         int tmp=23333, c;
47         for(int i=R[0]; i; i=R[i])
48             if(tmp>s[i]) tmp=s[i], c=i;
49         for(int i=D[c]; i!=c; i=D[i]){
50             remove(i);
51             for(int j=R[i]; j!=i; j=R[j]) remove(j);
52             dfs(d+1, ans);

```

```

53         for(int j=L[i];j!=i;j=L[j])resume(j);
54         resume(i);
55     }
56 }
57 void del(int c){//exactly cover
58     L[R[c]]=L[c];R[L[c]]=R[c];
59     for(int i=D[c];i!=c;i=D[i])
60         for(int j=R[i];j!=i;j=R[j])
61             U[D[j]]=U[j],D[U[j]]=D[j],--s[col[j]];
62 }
63 void add(int c){ //exactly cover
64     R[L[c]]=L[R[c]]=c;
65     for(int i=U[c];i!=c;i=U[i])
66         for(int j=L[i];j!=i;j=L[j])
67             ++s[col[U[D[j]]=D[U[j]]=j]];
68 }
69 bool dfs2(int k){//exactly cover
70     if(!R[0]){
71         cnt=k;return 1;
72     }
73     int c=R[0];
74     for(int i=R[0];i;i=R[i])
75         if(s[c]>s[i])c=i;
76     del(c);
77     for(int i=D[c];i!=c;i=D[i]){
78         for(int j=R[i];j!=i;j=R[j])
79             del(col[j]);
80         ans[k]=row[i];if(dfs2(k+1))return true;
81         for(int j=L[i];j!=i;j=L[j])
82             add(col[j]);
83     }
84     add(c);
85     return 0;
86 }
87 }dlx;
88 int main(){
89     dlx.init(n);
90     for(int i=1;i<=m;i++)
91         for(int j=1;j<=n;j++)
92             if(dis(station[i],city[j])<mid-eps)
93                 dlx.Link(i,j);
94     dlx.dfs(0,ans);

```



95 }

---

## 7.5 插头 DP

---

```
1  int n,m,l;
2  struct L{
3      int d[11];
4      int& operator[](int x){return d[x];}
5      int mc(int x){
6          int an=1;
7          if(d[x]==1){
8              for(x++;x<1;x++)if(d[x]){
9                  an=an+(d[x]==1?1:-1);
10                 if(!an)return x;
11             }
12         }else{
13             for(x--;x>=0;x--)if(d[x]){
14                 an=an+(d[x]==2?1:-1);
15                 if(!an)return x;
16             }
17         }
18     }
19     int h(){int an=0;for(int i=l-1;i>=0;i--)an=an*3+d[i];return an;}
20     L s(int x,int y){
21         L S=*this;
22         S[x]=y;return S;
23     }
24     L operator>>(int _){
25         L S=*this;
26         for(int i=l-1;i>=1;i--)S[i]=S[i-1];
27         S[0]=0;return S;
28     }
29 };
30 struct Int{
31     int len;
32     int a[40];
33     Int(){len=1;memset(a,0,sizeof a);}
34     Int operator+=(const Int &o){
35         int l=max(len,o.len);
36         for(int i=0;i<l;i++)
37             a[i]=a[i]+o.a[i];
38         for(int i=0;i<l;i++)
```

```

39         a[i+1]+=a[i]/10,a[i]%=10;
40         if(a[l])l++;len=l;
41         return *this;
42     }
43     void print(){
44         for(int i=len-1;i>=0;i--)
45             printf("%d",a[i]);
46         puts("");
47     }
48 };
49 struct hashtab{
50     int sz;
51     int tab[177147];
52     Int w[177147];
53     L s[177147];
54     hashtab(){memset(tab,-1,sizeof tab);}
55     void cl(){
56         for(int i=0;i<sz;i++)tab[s[i].h()]=-1;
57         sz=0;
58     }
59     Int& operator[](L S){
60         int h=S.h();
61         if(tab[h]==-1)tab[h]=sz,s[sz]=S,w[sz]=Int(),sz++;
62         return w[tab[h]];
63     }
64 }f[2];
65 bool check(L S){
66     int cn1=0,cn2=0;
67     for(int i=0;i<l;i++){
68         cn1+=S[i]==1;
69         cn2+=S[i]==2;
70     }return cn1==1&&cn2==1;
71 }
72 int main(){
73     Int One;One.a[0]=1;
74     scanf("%d%d",&n,&m);if(n<m)swap(n,m);l=m+1;
75     if(n==1||m==1){puts("1");return 0;}
76     int cur=0;f[cur].cl();
77     for(int i=1;i<=n;i++){
78         for(int j=1;j<=m;j++){
79             if(i==1&&j==1){
80                 f[cur][L().s(0,1).s(1,2)]+=One;

```

```

81         continue;
82     }
83     cur^=1;f[cur].cl();
84     for(int k=0;k<f[!cur].sz;k++){
85         L S=f[!cur].s[k];Int w=f[!cur][S];
86         int d1=S[j-1],d2=S[j];
87         if(d1==0&&d2==0){
88             if(i!=n&&j!=m)f[cur][S.s(j-1,1).s(j,2)]+=w;
89         }else
90         if(d1==0||d2==0){
91             if(i!=n)f[cur][S.s(j-1,d1|d2).s(j,0)]+=w;
92             if(j!=m)f[cur][S.s(j-1,0).s(j,d1|d2)]+=w;
93         }else
94         if(d1==1&&d2==2){
95             if(i==n&&j==m&&check(S))
96                 (w+=w).print();
97         }else
98         if(d1==2&&d2==1){
99             f[cur][S.s(j-1,0).s(j,0)]+=w;
100         }else
101         if((d1==1&&d2==1)|| (d1==2&&d2==2)){
102             int m1=S.mc(j),m2=S.mc(j-1);
103             f[cur][S.s(j-1,0).s(j,0).s(m1,1).s(m2,2)]+=w;
104         }
105     }
106 }
107 cur^=1;f[cur].cl();
108 for(int k=0;k<f[!cur].sz;k++){
109     L S=f[!cur].s[k];Int w=f[!cur][S];
110     f[cur][S>>1]=w;
111 }
112 }
113 return 0;
114 }

```

---

## 7.6 某年某月某日是星期几

```

1 int solve(int year, int month, int day) {
2     int answer;
3     if (month == 1 || month == 2) {
4         month += 12;
5         year--;

```

```

6     }
7     if ((year < 1752) || (year == 1752 && month < 9) ||
8         (year == 1752 && month == 9 && day < 3)) {
9         answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4 + 5) % 7;
10    } else {
11        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4
12                - year / 100 + year / 400) % 7;
13    }
14    return answer;
15 }

```

---

## 7.7 枚举大小为 $k$ 的子集

使用条件:  $k > 0$

```

1 void solve(int n, int k) {
2     for (int comb = (1 << k) - 1; comb < (1 << n); ) {
3         // ...
4         int x = comb & -comb, y = comb + x;
5         comb = (((comb & ~y) / x) >> 1) | y;
6     }
7 }

```

---

## 7.8 环状最长公共子串

```

1 int n, a[N << 1], b[N << 1];
2
3 bool has(int i, int j) {
4     return a[(i - 1) % n] == b[(j - 1) % n];
5 }
6
7 const int DELTA[3][2] = {{0, -1}, {-1, -1}, {-1, 0}};
8
9 int from[N][N];
10
11 int solve() {
12     memset(from, 0, sizeof(from));
13     int ret = 0;
14     for (int i = 1; i <= 2 * n; ++i) {
15         from[i][0] = 2;
16         int left = 0, up = 0;
17         for (int j = 1; j <= n; ++j) {

```

```

18     int upleft = up + 1 + !!from[i - 1][j];
19     if (!has(i, j)) {
20         upleft = INT_MIN;
21     }
22     int max = std::max(left, std::max(upleft, up));
23     if (left == max) {
24         from[i][j] = 0;
25     } else if (upleft == max) {
26         from[i][j] = 1;
27     } else {
28         from[i][j] = 2;
29     }
30     left = max;
31 }
32 if (i >= n) {
33     int count = 0;
34     for (int x = i, y = n; y; ) {
35         int t = from[x][y];
36         count += t == 1;
37         x += DELTA[t][0];
38         y += DELTA[t][1];
39     }
40     ret = std::max(ret, count);
41     int x = i - n + 1;
42     from[x][0] = 0;
43     int y = 0;
44     while (y <= n && from[x][y] == 0) {
45         y++;
46     }
47     for (; x <= i; ++x) {
48         from[x][y] = 0;
49         if (x == i) {
50             break;
51         }
52         for (; y <= n; ++y) {
53             if (from[x + 1][y] == 2) {
54                 break;
55             }
56             if (y + 1 <= n && from[x + 1][y + 1] == 1) {
57                 y++;
58                 break;
59             }

```

```

60         }
61     }
62 }
63 }
64 return ret;
65 }

```

---

## 7.9 LLMOD

```

1 LL multiplyMod(LL a, LL b, LL P) { // `需要保证 a 和 b 非负`
2     LL t = (a * b - LL((long double)a / P * b + 1e-3) * P) % P;
3     return t < 0 : t + P : t;
4 }

```

---

## 8 Java

### 8.1 基础模板

```

1 import java.io.*;
2 import java.util.*;
3 import java.math.*;
4 public class Main {
5     public static void main(String[] args) {
6         InputStream inputStream = System.in;
7         OutputStream outputStream = System.out;
8         InputReader in = new InputReader(inputStream);
9         PrintWriter out = new PrintWriter(outputStream);
10    }
11 }
12 public static class edge implements Comparable<edge>{
13     public int u,v,w;
14     public int compareTo(edge e){
15         return w-e.w;
16     }
17 }
18 public static class cmp implements Comparator<edge>{
19     public int compare(edge a,edge b){
20         if(a.w<b.w)return 1;
21         if(a.w>b.w)return -1;
22         return 0;
23     }

```

```

24 }
25 class InputReader {
26     public BufferedReader reader;
27     public StringTokenizer tokenizer;
28
29     public InputReader(InputStream stream) {
30         reader = new BufferedReader(new InputStreamReader(stream), 32768);
31         tokenizer = null;
32     }
33
34     public String next() {
35         while (tokenizer == null || !tokenizer.hasMoreTokens()) {
36             try {
37                 tokenizer = new StringTokenizer(reader.readLine());
38             } catch (IOException e) {
39                 throw new RuntimeException(e);
40             }
41         }
42         return tokenizer.nextToken();
43     }
44
45     public int nextInt() {
46         return Integer.parseInt(next());
47     }
48
49     public long nextLong() {
50         return Long.parseLong(next());
51     }
52 }

```

---

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)[compact1](#), [compact2](#), [compact3](#)[java.math](#)

## Class BigInteger

[java.lang.Object](#)[java.lang.Number](#)[java.math.BigInteger](#)

### All Implemented Interfaces:

[Serializable](#), [Comparable<BigInteger>](#)

```
public class BigInteger
    extends Number
    implements Comparable<BigInteger>
```

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

Semantics of arithmetic operations exactly mimic those of Java's integer arithmetic operators, as defined in *The Java Language Specification*. For example, division by zero throws an ArithmeticException, and division of a negative by a positive yields a negative (or zero) remainder. All of the details in the Spec concerning overflow are ignored, as BigIntegers are made as large as necessary to accommodate the results of an operation.

Semantics of shift operations extend those of Java's shift operators to allow for negative shift distances. A right-shift with a negative shift distance results in a left shift, and vice-versa. The unsigned right shift operator (>>>) is omitted, as this operation makes little sense in combination with the "infinite word size" abstraction provided by this class.

Semantics of bitwise logical operations exactly mimic those of Java's bitwise integer operators. The binary operators (and, or, xor) implicitly perform sign extension on the shorter of the two operands prior to performing the operation.

Comparison operations perform signed integer comparisons, analogous to those performed by Java's relational and equality operators.

Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. These methods always return a non-negative result, between 0 and (modulus - 1), inclusive.

Bit operations operate on a single bit of the two's-complement representation of their operand. If necessary, the operand is sign-extended so that it contains the designated bit. None of the single-bit operations can produce a BigInteger with a different sign from the BigInteger being operated on, as they affect only a single bit, and the "infinite word size" abstraction provided by this class ensures that there are infinitely many "virtual sign bits"



preceding each BigInteger.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigInteger methods. The pseudo-code expression (i + j) is shorthand for "a BigInteger whose value is that of the BigInteger i plus that of the BigInteger j." The pseudo-code expression (i == j) is shorthand for "true if and only if the BigInteger i represents the same value as the BigInteger j." Other pseudo-code expressions are interpreted similarly.

All methods and constructors in this class throw NullPointerException when passed a null object reference for any input parameter. BigInteger must support values in the range -2<sup>Integer.MAX\_VALUE</sup> (exclusive) to +2<sup>Integer.MAX\_VALUE</sup> (exclusive) and may support values outside of that range. The range of probable prime values is limited and may be less than the full supported positive range of BigInteger. The range must be at least 1 to 2<sup>5000000000</sup>.

**Implementation Note:**

BigInteger constructors and operations throw ArithmeticException when the result is out of the supported range of -2<sup>Integer.MAX\_VALUE</sup> (exclusive) to +2<sup>Integer.MAX\_VALUE</sup> (exclusive).

**Since:**

JDK1.1

**See Also:**

[BigDecimal](#), [Serialized Form](#)

**Field Summary**

**Fields**

Modifier and Type	Field and Description
static <a href="#">BigInteger</a>	<b>ONE</b> The BigInteger constant one.
static <a href="#">BigInteger</a>	<b>TEN</b> The BigInteger constant ten.
static <a href="#">BigInteger</a>	<b>ZERO</b> The BigInteger constant zero.

**Constructor Summary**

**Constructors**

Constructor and Description
<a href="#">BigInteger</a> (byte[] val) Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger.
<a href="#">BigInteger</a> (int signum, byte[] magnitude) Translates the sign-magnitude representation of a BigInteger into a BigInteger.

**BigInteger**(int bitLength, int certainty, **Random** rnd)

Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

**BigInteger**(int numBits, **Random** rnd)

Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ( $2^{\text{numBits}} - 1$ ), inclusive.

**BigInteger**(String val)

Translates the decimal String representation of a BigInteger into a BigInteger.

**BigInteger**(String val, int radix)

Translates the String representation of a BigInteger in the specified radix into a BigInteger.

## Method Summary

**All Methods**    **Static Methods**    **Instance Methods**    **Concrete Methods**

Modifier and Type	Method and Description
<b>BigInteger</b>	<b>abs()</b> Returns a BigInteger whose value is the absolute value of this BigInteger.
<b>BigInteger</b>	<b>add(BigInteger val)</b> Returns a BigInteger whose value is ( <code>this + val</code> ).
<b>BigInteger</b>	<b>and(BigInteger val)</b> Returns a BigInteger whose value is ( <code>this &amp; val</code> ).
<b>BigInteger</b>	<b>andNot(BigInteger val)</b> Returns a BigInteger whose value is ( <code>this &amp; ~val</code> ).
int	<b>bitCount()</b> Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	<b>bitLength()</b> Returns the number of bits in the minimal two's-complement representation of this BigInteger, <i>excluding</i> a sign bit.
byte	<b>byteValueExact()</b> Converts this BigInteger to a byte, checking for lost information.
<b>BigInteger</b>	<b>clearBit(int n)</b> Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	<b>compareTo(BigInteger val)</b> Compares this BigInteger with the specified BigInteger.
<b>BigInteger</b>	<b>divide(BigInteger val)</b> Returns the BigInteger that is the quotient of this BigInteger divided by the specified BigInteger.

	Returns a <code>BigInteger</code> whose value is <code>(this / val)</code> .
<code>BigInteger[]</code>	<b><code>divideAndRemainder(BigInteger val)</code></b> Returns an array of two <code>BigInteger</code> s containing <code>(this / val)</code> followed by <code>(this % val)</code> .
<code>double</code>	<b><code>doubleValue()</code></b> Converts this <code>BigInteger</code> to a <code>double</code> .
<code>boolean</code>	<b><code>equals(Object x)</code></b> Compares this <code>BigInteger</code> with the specified <code>Object</code> for equality.
<code>BigInteger</code>	<b><code>flipBit(int n)</code></b> Returns a <code>BigInteger</code> whose value is equivalent to this <code>BigInteger</code> with the designated bit flipped.
<code>float</code>	<b><code>floatValue()</code></b> Converts this <code>BigInteger</code> to a <code>float</code> .
<code>BigInteger</code>	<b><code>gcd(BigInteger val)</code></b> Returns a <code>BigInteger</code> whose value is the greatest common divisor of <code>abs(this)</code> and <code>abs(val)</code> .
<code>int</code>	<b><code>getLowestSetBit()</code></b> Returns the index of the rightmost (lowest-order) one bit in this <code>BigInteger</code> (the number of zero bits to the right of the rightmost one bit).
<code>int</code>	<b><code>hashCode()</code></b> Returns the hash code for this <code>BigInteger</code> .
<code>int</code>	<b><code>intValue()</code></b> Converts this <code>BigInteger</code> to an <code>int</code> .
<code>int</code>	<b><code>intValueExact()</code></b> Converts this <code>BigInteger</code> to an <code>int</code> , checking for lost information.
<code>boolean</code>	<b><code>isProbablePrime(int certainty)</code></b> Returns <code>true</code> if this <code>BigInteger</code> is probably prime, <code>false</code> if it's definitely composite.
<code>long</code>	<b><code>longValue()</code></b> Converts this <code>BigInteger</code> to a <code>long</code> .
<code>long</code>	<b><code>longValueExact()</code></b> Converts this <code>BigInteger</code> to a <code>long</code> , checking for lost information.
<code>BigInteger</code>	<b><code>max(BigInteger val)</code></b> Returns the maximum of this <code>BigInteger</code> and <code>val</code> .
<code>BigInteger</code>	<b><code>min(BigInteger val)</code></b> Returns the minimum of this <code>BigInteger</code> and <code>val</code> .
<code>BigInteger</code>	<b><code>mod(BigInteger m)</code></b> Returns a <code>BigInteger</code> whose value is <code>(this mod m)</code> .

<b>BigInteger</b>	<b>modInverse(BigInteger m)</b> Returns a BigInteger whose value is $(\text{this}^{-1} \bmod m)$ .
<b>BigInteger</b>	<b>modPow(BigInteger exponent, BigInteger m)</b> Returns a BigInteger whose value is $(\text{this}^{\text{exponent}} \bmod m)$ .
<b>BigInteger</b>	<b>multiply(BigInteger val)</b> Returns a BigInteger whose value is $(\text{this} * \text{val})$ .
<b>BigInteger</b>	<b>negate()</b> Returns a BigInteger whose value is $(-\text{this})$ .
<b>BigInteger</b>	<b>nextProbablePrime()</b> Returns the first integer greater than this BigInteger that is probably prime.
<b>BigInteger</b>	<b>not()</b> Returns a BigInteger whose value is $(\sim \text{this})$ .
<b>BigInteger</b>	<b>or(BigInteger val)</b> Returns a BigInteger whose value is $(\text{this}   \text{val})$ .
<b>BigInteger</b>	<b>pow(int exponent)</b> Returns a BigInteger whose value is $(\text{this}^{\text{exponent}})$ .
static <b>BigInteger</b>	<b>probablePrime(int bitLength, Random rnd)</b> Returns a positive BigInteger that is probably prime, with the specified bitLength.
<b>BigInteger</b>	<b>remainder(BigInteger val)</b> Returns a BigInteger whose value is $(\text{this} \% \text{val})$ .
<b>BigInteger</b>	<b>setBit(int n)</b> Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
<b>BigInteger</b>	<b>shiftLeft(int n)</b> Returns a BigInteger whose value is $(\text{this} \ll n)$ .
<b>BigInteger</b>	<b>shiftRight(int n)</b> Returns a BigInteger whose value is $(\text{this} \gg n)$ .
short	<b>shortValueExact()</b> Converts this BigInteger to a short, checking for lost information.
int	<b>signum()</b> Returns the signum function of this BigInteger.
<b>BigInteger</b>	<b>subtract(BigInteger val)</b> Returns a BigInteger whose value is $(\text{this} - \text{val})$ .
boolean	<b>testBit(int n)</b> Returns true if and only if the designated bit is set.
byte[]	<b>toByteArray()</b> Returns a byte array containing the two's complement binary representation of the BigInteger value.

Returns a byte array containing the two's-complement representation of this BigInteger.

**String**

**toString()**

Returns the decimal String representation of this BigInteger.

**String**

**toString(int radix)**

Returns the String representation of this BigInteger in the given radix.

static **BigInteger** **valueOf(long val)**

Returns a BigInteger whose value is equal to that of the specified long.

**BigInteger**

**xor(BigInteger val)**

Returns a BigInteger whose value is (this ^ val).

### Methods inherited from class java.lang.Number

byteValue, shortValue

### Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

## Field Detail

### ZERO

public static final **BigInteger** ZERO

The BigInteger constant zero.

**Since:**

1.2

### ONE

public static final **BigInteger** ONE

The BigInteger constant one.

**Since:**

1.2

### TEN

public static final **BigInteger** TEN

The BigInteger constant ten.

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [ALL CLASSES](#)[SUMMARY: NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

compact1, compact2, compact3

java.util

## Class `TreeMap<K,V>`

java.lang.Object

java.util.AbstractMap&lt;K,V&gt;

java.util.TreeMap&lt;K,V&gt;

### Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

### All Implemented Interfaces:

`Serializable`, `Cloneable`, `Map<K,V>`, `NavigableMap<K,V>`, `SortedMap<K,V>`

```
public class TreeMap<K,V>
extends AbstractMap<K,V>
implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the **natural ordering** of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

This implementation provides guaranteed  $\log(n)$  time cost for the `containsKey`, `get`, `put` and `remove` operations. Algorithms are adaptations of those in Cormen, Leiserson, and Rivest's *Introduction to Algorithms*.

Note that the ordering maintained by a tree map, like any sorted map, and whether or not an explicit comparator is provided, must be *consistent with equals* if this sorted map is to correctly implement the `Map` interface. (See `Comparable` or `Comparator` for a precise definition of *consistent with equals*.) This is so because the `Map` interface is defined in terms of the `equals` operation, but a sorted map performs all key comparisons using its `compareTo` (or `compare`) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a sorted map is well-defined even if its ordering is inconsistent with `equals`; it just fails to obey the general contract of the `Map` interface.

**Note that this implementation is not synchronized.** If multiple threads access a map concurrently, and at least one of the threads modifies the map structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with an existing key is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be "wrapped" using the `Collections.synchronizedSortedMap` method. This is best done at creation time, to prevent accidental unsynchronized access to the map:

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

The iterators returned by the `iterator` method of the collections returned by all of this

class's "collection view methods" are *fail-fast*: if the map is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs*.

All `Map.Entry` pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do **not** support the `Entry.setValue` method. (Note however that it is possible to change mappings in the associated map using `put`.)

This class is a member of the [Java Collections Framework](#).

**Since:**

1.2

**See Also:**

[Map](#), [HashMap](#), [Hashtable](#), [Comparable](#), [Comparator](#), [Collection](#), [Serialized Form](#)

## ***Nested Class Summary***

### **Nested classes/interfaces inherited from class [java.util.AbstractMap](#)**

[AbstractMap.SimpleEntry<K,V>](#), [AbstractMap.SimpleImmutableEntry<K,V>](#)

## ***Constructor Summary***

### **Constructors**

#### **Constructor and Description**

##### **[TreeMap\(\)](#)**

Constructs a new, empty tree map, using the natural ordering of its keys.

##### **[TreeMap\(Comparator<? super K> comparator\)](#)**

Constructs a new, empty tree map, ordered according to the given comparator.

##### **[TreeMap\(Map<? extends K,? extends V> m\)](#)**

Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.

##### **[TreeMap\(SortedMap<K,? extends V> m\)](#)**

Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

## ***Method Summary***

Modifier and Type	Method and Description
<b>Map.Entry&lt;K, V&gt;</b>	<b>ceilingEntry(K key)</b> Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such key.
<b>K</b>	<b>ceilingKey(K key)</b> Returns the least key greater than or equal to the given key, or null if there is no such key.
<b>void</b>	<b>clear()</b> Removes all of the mappings from this map.
<b>Object</b>	<b>clone()</b> Returns a shallow copy of this TreeMap instance.
<b>Comparator&lt;? super K&gt;</b>	<b>comparator()</b> Returns the comparator used to order the keys in this map, or null if this map uses the <b>natural ordering</b> of its keys.
<b>boolean</b>	<b>containsKey(Object key)</b> Returns true if this map contains a mapping for the specified key.
<b>boolean</b>	<b>containsValue(Object value)</b> Returns true if this map maps one or more keys to the specified value.
<b>NavigableSet&lt;K&gt;</b>	<b>descendingKeySet()</b> Returns a reverse order <b>NavigableSet</b> view of the keys contained in this map.
<b>NavigableMap&lt;K, V&gt;</b>	<b>descendingMap()</b> Returns a reverse order view of the mappings contained in this map.
<b>Set&lt;Map.Entry&lt;K, V&gt;&gt;</b>	<b>entrySet()</b> Returns a <b>Set</b> view of the mappings contained in this map.
<b>Map.Entry&lt;K, V&gt;</b>	<b>firstEntry()</b> Returns a key-value mapping associated with the least key in this map, or null if the map is empty.
<b>K</b>	<b>firstKey()</b> Returns the first (lowest) key currently in this map.
<b>Map.Entry&lt;K, V&gt;</b>	<b>floorEntry(K key)</b> Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key.
<b>K</b>	<b>floorKey(K key)</b> Returns the greatest key less than or equal to the given key, or null if there is no such key.



or null if there is no such key.

void

**forEach**(**BiConsumer**<? super **K**,? super **V**> action)

Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.

**V**

**get**(**Object** key)

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

**SortedMap**<**K**,**V**>

**headMap**(**K** toKey)

Returns a view of the portion of this map whose keys are strictly less than toKey.

**NavigableMap**<**K**,**V**>

**headMap**(**K** toKey, boolean inclusive)

Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey.

**Map.Entry**<**K**,**V**>

**higherEntry**(**K** key)

Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key.

**K**

**higherKey**(**K** key)

Returns the least key strictly greater than the given key, or null if there is no such key.

**Set**<**K**>

**keySet**()

Returns a **Set** view of the keys contained in this map.

**Map.Entry**<**K**,**V**>

**lastEntry**()

Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.

**K**

**lastKey**()

Returns the last (highest) key currently in this map.

**Map.Entry**<**K**,**V**>

**lowerEntry**(**K** key)

Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key.

**K**

**lowerKey**(**K** key)

Returns the greatest key strictly less than the given key, or null if there is no such key.

**NavigableSet**<**K**>

**navigableKeySet**()

Returns a **NavigableSet** view of the keys contained in this map.

**Map.Entry**<**K**,**V**>

**pollFirstEntry**()

Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.

**Map.Entry**<**K**,**V**>

**pollLastEntry**()

Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.

the greatest key in this map, or null if the map is empty.

<b>V</b>	<b>put(K key, V value)</b> Associates the specified value with the specified key in this map.
void	<b>putAll(Map&lt;? extends K,? extends V&gt; map)</b> Copies all of the mappings from the specified map to this map.
<b>V</b>	<b>remove(Object key)</b> Removes the mapping for this key from this TreeMap if present.
<b>V</b>	<b>replace(K key, V value)</b> Replaces the entry for the specified key only if it is currently mapped to some value.
boolean	<b>replace(K key, V oldValue, V newValue)</b> Replaces the entry for the specified key only if currently mapped to the specified value.
void	<b>replaceAll(BiFunction&lt;? super K,? super V,? extends V&gt; function)</b> Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	<b>size()</b> Returns the number of key-value mappings in this map.
<b>NavigableMap&lt;K,V&gt;</b>	<b>subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)</b> Returns a view of the portion of this map whose keys range from fromKey to toKey.
<b>SortedMap&lt;K,V&gt;</b>	<b>subMap(K fromKey, K toKey)</b> Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive.
<b>SortedMap&lt;K,V&gt;</b>	<b>tailMap(K fromKey)</b> Returns a view of the portion of this map whose keys are greater than or equal to fromKey.
<b>NavigableMap&lt;K,V&gt;</b>	<b>tailMap(K fromKey, boolean inclusive)</b> Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.
<b>Collection&lt;V&gt;</b>	<b>values()</b> Returns a <b>Collection</b> view of the values contained in this map.

### Methods inherited from class java.util.AbstractMap

equals, hashCode, isEmpty, toString

## 9 gedit

---

```
1 Compile:
2 #!/bin/sh
3 full=$GEDIT_CURRENT_DOCUMENT_NAME
4 name=`echo $full | cut -d. -f1`
5 g++ $full -o $name -g -Wall
6
7 Debug:
8 #!/bin/bash
9 name=`echo $GEDIT_CURRENT_DOCUMENT_NAME | cut -d. -f1`
10 gnome-terminal -x bash -c "gdb ./$name"
11
12 Run:
13 #!/bin/bash
14 name=`echo $GEDIT_CURRENT_DOCUMENT_NAME | cut -d. -f1`
15 gnome-terminal -x bash -c "time ./$name;echo 'Press any key to continue'; read"
```

---

## 10 数学

### 10.1 常用数学公式

#### 10.1.1 求和公式

1.  $\sum_{k=1}^n (2k-1)^2 = \frac{n(4n^2-1)}{3}$
2.  $\sum_{k=1}^n k^3 = [\frac{n(n+1)}{2}]^2$
3.  $\sum_{k=1}^n (2k-1)^3 = n^2(2n^2-1)$
4.  $\sum_{k=1}^n k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
5.  $\sum_{k=1}^n k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$
6.  $\sum_{k=1}^n k(k+1) = \frac{n(n+1)(n+2)}{3}$
7.  $\sum_{k=1}^n k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$
8.  $\sum_{k=1}^n k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$

#### 10.1.2 斐波那契数列

1.  $fib_0 = 0, fib_1 = 1, fib_n = fib_{n-1} + fib_{n-2}$
2.  $fib_{n+2} \cdot fib_n - fib_{n+1}^2 = (-1)^{n+1}$
3.  $fib_{-n} = (-1)^{n-1} fib_n$

$$4. \text{fib}_{n+k} = \text{fib}_k \cdot \text{fib}_{n+1} + \text{fib}_{k-1} \cdot \text{fib}_n$$

$$5. \gcd(\text{fib}_m, \text{fib}_n) = \text{fib}_{\gcd(m,n)}$$

$$6. \text{fib}_m | \text{fib}_n^2 \Leftrightarrow n \text{fib}_n | m$$

### 10.1.3 错排公式

$$1. D_n = (n-1)(D_{n-2} - D_{n-1})$$

$$2. D_n = n! \cdot (1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + \frac{(-1)^n}{n!})$$

### 10.1.4 莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & \text{若 } n = 1 \\ (-1)^k & \text{若 } n \text{ 无平方数因子, 且 } n = p_1 p_2 \dots p_k \\ 0 & \text{若 } n \text{ 有大于1的平方数因数} \end{cases}$$

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{若 } n = 1 \\ 0 & \text{其他情况} \end{cases}$$

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right)$$

$$g(x) = \sum_{n=1}^{[x]} f\left(\frac{x}{n}\right) \Leftrightarrow f(x) = \sum_{n=1}^{[x]} \mu(n) g\left(\frac{x}{n}\right)$$

### 10.1.5 伯恩赛德引理

设  $G$  是一个有限群, 作用在集合  $X$  上。对每个  $g$  属于  $G$ , 令  $X^g$  表示  $X$  中在  $g$  作用下的不动元素, 轨道数 (记作  $|X/G|$ ) 由如下公式给出:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

### 10.1.6 五边形数定理

设  $p(n)$  是  $n$  的拆分数, 有

$$p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$$

### 10.1.7 树的计数

1. 有根树计数:  $n+1$  个结点的有根树的个数为

$$a_{n+1} = \frac{\sum_{j=1}^n j \cdot a_j \cdot S_{n,j}}{n}$$

其中,

$$S_{n,j} = \sum_{i=1}^{n/j} a_{n+1-ij} = S_{n-j,j} + a_{n+1-j}$$

2. 无根树计数: 当  $n$  为奇数时,  $n$  个结点的无根树的个数为

$$a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$$

当  $n$  为偶数时,  $n$  个结点的无根树的个数为

$$a_n - \sum_{i=1}^{n/2} a_i a_{n-i} + \frac{1}{2} a_{\frac{n}{2}} (a_{\frac{n}{2}} + 1)$$

3.  $n$  个结点的完全图的生成树个数为

$$n^{n-2}$$

4. 矩阵-树定理: 图  $G$  由  $n$  个结点构成, 设  $A[G]$  为图  $G$  的邻接矩阵、 $D[G]$  为图  $G$  的度数矩阵, 则图  $G$  的不同生成树的个数为  $C[G] = D[G] - A[G]$  的任意一个  $n-1$  阶主子式的行列式值。

#### 10.1.8 欧拉公式

平面图的顶点个数、边数和面的个数有如下关系:

$$V - E + F = C + 1$$

其中,  $V$  是顶点的数目,  $E$  是边的数目,  $F$  是面的数目,  $C$  是组成图形的连通部分的数目。当图是单连通图的时候, 公式简化为:

$$V - E + F = 2$$

#### 10.1.9 皮克定理

给定顶点坐标均是整点 (或正方形格点) 的简单多边形, 其面积  $A$  和内部格点数目  $i$ 、边上格点数目  $b$  的关系:

$$A = i + \frac{b}{2} - 1$$

#### 10.1.10 牛顿恒等式

设

$$\prod_{i=1}^n (x - x_i) = a_n + a_{n-1}x + \cdots + a_1x^{n-1} + a_0x^n$$

$$p_k = \sum_{i=1}^n x_i^k$$

则

$$a_0 p_k + a_1 p_{k-1} + \cdots + a_{k-1} p_1 + k a_k = 0$$

特别地, 对于

$$|\mathbf{A} - \lambda \mathbf{E}| = (-1)^n (a_n + a_{n-1} \lambda + \cdots + a_1 \lambda^{n-1} + a_0 \lambda^n)$$

有

$$p_k = \text{Tr}(\mathbf{A}^k)$$

## 10.2 平面几何公式

### 10.2.1 三角形

#### 1. 半周长

$$p = \frac{a+b+c}{2}$$

#### 2. 面积

$$S = \frac{a \cdot H_a}{2} = \frac{ab \cdot \sin C}{2} = \sqrt{p(p-a)(p-b)(p-c)}$$

#### 3. 中线

$$M_a = \frac{\sqrt{2(b^2 + c^2) - a^2}}{2} = \frac{\sqrt{b^2 + c^2 + 2bc \cdot \cos A}}{2}$$

#### 4. 角平分线

$$T_a = \frac{\sqrt{bc \cdot [(b+c)^2 - a^2]}}{b+c} = \frac{2bc \cos \frac{A}{2}}{b+c}$$

#### 5. 高线

$$H_a = b \sin C = c \sin B = \sqrt{b^2 - \left(\frac{a^2 + b^2 - c^2}{2a}\right)^2}$$

#### 6. 内切圆半径

$$\begin{aligned} r &= \frac{S}{p} = \frac{\arcsin \frac{B}{2} \cdot \sin \frac{C}{2}}{\sin \frac{B+C}{2}} = 4R \cdot \sin \frac{A}{2} \sin \frac{B}{2} \sin \frac{C}{2} \\ &= \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} = p \cdot \tan \frac{A}{2} \tan \frac{B}{2} \tan \frac{C}{2} \end{aligned}$$

#### 7. 外接圆半径

$$R = \frac{abc}{4S} = \frac{a}{2 \sin A} = \frac{b}{2 \sin B} = \frac{c}{2 \sin C}$$

### 10.2.2 四边形

$D_1, D_2$  为对角线,  $M$  为对角线中点连线,  $A$  为对角线夹角,  $p$  为半周长

$$1. a^2 + b^2 + c^2 + d^2 = D_1^2 + D_2^2 + 4M^2$$

$$2. S = \frac{1}{2} D_1 D_2 \sin A$$

3. 对于圆内接四边形

$$ac + bd = D_1 D_2$$

4. 对于圆内接四边形

$$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$$

### 10.2.3 正 $n$ 边形

$R$  为外接圆半径,  $r$  为内切圆半径

1. 中心角

$$A = \frac{2\pi}{n}$$

2. 内角

$$C = \frac{n-2}{n}\pi$$

3. 边长

$$a = 2\sqrt{R^2 - r^2} = 2R \cdot \sin \frac{A}{2} = 2r \cdot \tan \frac{A}{2}$$

4. 面积

$$S = \frac{nar}{2} = nr^2 \cdot \tan \frac{A}{2} = \frac{nR^2}{2} \cdot \sin A = \frac{na^2}{4 \cdot \tan \frac{A}{2}}$$

### 10.2.4 圆

1. 弧长

$$l = rA$$

2. 弦长

$$a = 2\sqrt{2hr - h^2} = 2r \cdot \sin \frac{A}{2}$$

3. 弓形高

$$h = r - \sqrt{r^2 - \frac{a^2}{4}} = r(1 - \cos \frac{A}{2}) = \frac{1}{2} \cdot \arctan \frac{A}{4}$$

4. 扇形面积

$$S_1 = \frac{rl}{2} = \frac{r^2 A}{2}$$

5. 弓形面积

$$S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2}{2}(A - \sin A)$$

### 10.2.5 棱柱

1. 体积

$$V = Ah$$

$A$  为底面积,  $h$  为高

2. 侧面积

$$S = lp$$

$l$  为棱长,  $p$  为直截面周长

3. 全面积

$$T = S + 2A$$

### 10.2.6 棱锥

1. 体积

$$V = Ah$$

$A$  为底面积,  $h$  为高

2. 正棱锥侧面积

$$S = lp$$

$l$  为棱长,  $p$  为直截面周长

3. 正棱锥全面积

$$T = S + 2A$$

### 10.2.7 棱台

1. 体积

$$V = (A_1 + A_2 + \sqrt{A_1 A_2}) \cdot \frac{h}{3}$$

$A_1, A_2$  为上下底面积,  $h$  为高

2. 正棱台侧面积

$$S = \frac{p_1 + p_2}{2} l$$

$p_1, p_2$  为上下底面周长,  $l$  为斜高

3. 正棱台全面积

$$T = S + A_1 + A_2$$

### 10.2.8 圆柱

1. 侧面积

$$S = 2\pi r h$$

2. 全面积

$$T = 2\pi r(h + r)$$

3. 体积

$$V = \pi r^2 h$$



### 10.2.9 圆锥

#### 1. 母线

$$l = \sqrt{h^2 + r^2}$$

#### 2. 侧面积

$$S = \pi r l$$

#### 3. 全面积

$$T = \pi r(l + r)$$

#### 4. 体积

$$V = \frac{\pi}{3} r^2 h$$

### 10.2.10 圆台

#### 1. 母线

$$l = \sqrt{h^2 + (r_1 - r_2)^2}$$

#### 2. 侧面积

$$S = \pi(r_1 + r_2)l$$

#### 3. 全面积

$$T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$$

#### 4. 体积

$$V = \frac{\pi}{3}(r_1^2 + r_2^2 + r_1 r_2)h$$

### 10.2.11 球

#### 1. 全面积

$$T = 4\pi r^2$$

#### 2. 体积

$$V = \frac{4}{3}\pi r^3$$

### 10.2.12 球台

#### 1. 侧面积

$$S = 2\pi r h$$

#### 2. 全面积

$$T = \pi(2rh + r_1^2 + r_2^2)$$

#### 3. 体积

$$V = \frac{\pi h[3(r_1^2 + r_2^2) + h^2]}{6}$$

### 10.2.13 球扇形

#### 1. 全面积

$$T = \pi r(2h + r_0)$$

$h$  为球冠高,  $r_0$  为球冠底面半径

#### 2. 体积

$$V = \frac{2}{3}\pi r^2 h$$

## 10.3 立体几何公式

### 10.3.1 球面三角公式

设  $a, b, c$  是边长,  $A, B, C$  是所对的二面角, 有余弦定理

$$\cos a = \cos b \cdot \cos c + \sin b \cdot \sin c \cdot \cos A$$

正弦定理

$$\frac{\sin A}{\sin a} = \frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}$$

三角形面积是  $A + B + C - \pi$

### 10.3.2 四面体体积公式

$U, V, W, u, v, w$  是四面体的 6 条棱,  $U, V, W$  构成三角形,  $(U, u), (V, v), (W, w)$  互为对棱, 则

$$V = \frac{\sqrt{(s-2a)(s-2b)(s-2c)(s-2d)}}{192uvw}$$

其中

$$\left\{ \begin{array}{lcl} a & = & \sqrt{xYZ}, \\ b & = & \sqrt{yZX}, \\ c & = & \sqrt{zXY}, \\ d & = & \sqrt{xyz}, \\ s & = & a + b + c + d, \\ X & = & (w - U + v)(U + v + w), \\ x & = & (U - v + w)(v - w + U), \\ Y & = & (u - V + w)(V + w + u), \\ y & = & (V - w + u)(w - u + V), \\ Z & = & (v - W + u)(W + u + v), \\ z & = & (W - u + v)(u - v + W) \end{array} \right.$$