

---

# ACM TEMPLATE

---

Zengarden

Last build at June 21, 2013

# Contents

<b>1</b>	<b>数据结构</b>	<b>3</b>
1.1	树状数组 . . . . .	3
1.2	坐标离散 . . . . .	3
1.3	lca-rmq . . . . .	4
1.4	rmq2d . . . . .	5
1.5	划分树 . . . . .	7
1.6	扫描线矩形面积并 . . . . .	8
<b>2</b>	<b>数学</b>	<b>11</b>
2.1	素数 . . . . .	11
2.1.1	筛素数 . . . . .	11
2.1.2	Miller-Rabbin . . . . .	11
2.2	Gcd . . . . .	12
2.3	extend-gcd . . . . .	12
2.3.1	求模线性方程 . . . . .	12
2.3.2	求逆 . . . . .	12
2.4	快速加 . . . . .	13
2.5	快速幂 . . . . .	13
2.6	矩阵类 . . . . .	13
2.7	容斥 . . . . .	14
2.8	组合数 . . . . .	15
2.8.1	暴力求解 . . . . .	15
2.8.2	打表 . . . . .	15
2.8.3	质因数分解 . . . . .	15
2.8.4	Lucas . . . . .	17
2.9	pollardRho . . . . .	18
2.10	欧拉函数 . . . . .	19
2.10.1	一般的求法 . . . . .	19
2.10.2	递推 . . . . .	19
2.10.3	单独求 . . . . .	19
2.11	高斯消元 . . . . .	20
2.11.1	模二消元 . . . . .	20
2.11.2	浮点 . . . . .	21
2.12	格雷码 . . . . .	21
2.13	离散对数 . . . . .	22
2.14	字典序 . . . . .	24
2.14.1	排列 . . . . .	24
2.14.2	组合 . . . . .	24
2.15	置换 polya . . . . .	25
<b>3</b>	<b>string</b>	<b>26</b>
3.1	kmp . . . . .	26
3.2	kmp . . . . .	26
3.3	manacher . . . . .	27
3.4	ac 自动机 . . . . .	29
3.5	后缀数组 . . . . .	30
3.6	后缀自动机 . . . . .	33
3.7	elfhash . . . . .	34
3.8	散列 hash . . . . .	34
3.9	可获取任意段字符串的 hash . . . . .	35

---

<b>4</b>	<b>图论</b>	<b>36</b>
4.1	前向星 . . . . .	36
4.2	并差集 . . . . .	36
4.3	spfa . . . . .	36
4.4	LCA . . . . .	37
4.5	Dinic . . . . .	38
4.6	sap . . . . .	39
4.7	费用流 . . . . .	41
<b>5</b>	<b>计算几何</b>	<b>44</b>
5.1	动态凸包 . . . . .	44

# 1 数据结构

## 1.1 树状数组

注意 init 的时候要小一个 1

```

1  template<int MAXN=300000, typename T = int>
2  struct BIT {
3      int n;
4      T a[MAXN];
5
6      void init(int n) {
7          this->n = n;
8          fill(a, a + n + 1, T());
9      }
10     void add(int i, T v) {
11         for (int j = i; j <= n; j = (j | (j - 1)) + 1) {
12             a[j] += v;
13         }
14     }
15     //(0..i];
16     T sum(int i) const {
17         T ret = T();
18         for (int j = i; j > 0; j = j & (j - 1)) {
19             ret += a[j];
20         }
21         return ret;
22     }
23     T get(int i) const {
24         return sum(i) - sum(i - 1);
25     }
26     void set(int i, T v) {
27         add(i, v - get(i));
28     }
29
30     void add(int l, int r, T v) //need sum is ith val; get && set can'
31     {
32         add(l, v); add(r + 1, -v);
33     }
34 };

```

## 1.2 坐标离散

注意下标~

```

1  const int MaxN=100;
2  int axis[MaxN];
3  int r[MaxN]; //排序用到的数组;
4  int mp[MaxN]; //离散值到原始值的映射;
5  int M; //离散值的最大值, [1, M];
6  bool cmp(int a, int b) {return axis[a] < axis[b];}

```

```

7 void Lisan(int N)
8 {
9     for(int i=0; i<N; i++) r[i] = i;
10    sort(r, r+N, cmp);
11    mp[1] = axis[r[0]];
12    axis[r[0]] = M = 1;
13    for(int i=1; i<N; i++)
14    {
15        if(axis[r[i]] == mp[M]) axis[r[i]] = M;
16        else mp[++M] = axis[r[i]], axis[r[i]] = M;
17    }
18 }
19 int main(){
20     for (int i=0;i<5;i++) scanf("%d",&axis[i]);
21     Lisan(5);
22     return 0;
23 }

```

### 1.3 lca-rmq

```

1 using namespace std;
2 typedef long long ll;
3 const int N=10010;
4 int n;
5 struct E{
6     int u,v,nxt,w;
7 }edg[2000010];
8
9 int tote,head[N];
10 void init(){
11     tote=0;
12     memset(head,-1,sizeof(head));
13 }
14 inline void addedg(int u,int v){
15     edg[tote].u=u;edg[tote].v=v;edg[tote].nxt=head[u];head[u]=tote++;
16 };
17
18 int vst[N],e[N<<1],r[N],d[N<<1];
19 int cnt;
20 int fa[N];
21 void dfs(int u, int depth) {
22     vst[u] = true;
23     e[cnt] = u;
24     d[cnt] = depth;
25     r[u] = cnt++;
26     for(int i=head[u]; i!=-1; i=edg[i].nxt){
27         int v=edg[i].v;
28         if (!vst[v]){
29             dfs(v, depth+1);
30             e[cnt]=u;
31             d[cnt++]=depth;

```

```

32     }
33 }
34 }
35 inline int _min(int i, int j) {
36     if (d[i] < d[j]) return i;
37     return j;
38 }
39 int dp[2*N][16];
40 void rmpinit(){
41     int nn = 2 * n - 1;
42     for (int i = 0; i < nn; ++i) //下标是从开始的0
43         dp[i][0] = i;
44     int k = (int)(log(nn * 1.0) / log(2.0));
45     for (int j = 1; j <= k; ++j) {
46         for (int i = 0; i + (1 << j) - 1 < nn; ++i)
47             dp[i][j] = _min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
48     }
49 }
50 inline int query(int l, int r) {
51     int k = (int)(log(r * 1.0 - l + 1) / log(2.0));
52     return _min(dp[l][k], dp[r-(1<<k)+1][k]);
53 }
54 int main(){
55     //fi;
56     int t;
57     scanf("%d",&t);
58     int u,v;
59     while(t--){
60         scanf("%d",&n);
61         init();
62         fr(i,0,n+1) fa[i]=i;
63         fr(i,0,n-1){
64             scanf("%d%d",&u,&v);
65             addedg(u,v); //addedg(v,u);
66             fa[v]=u;
67         }
68         int root;
69         fr(i,1,n+1) if (fa[i]==i) {root=i; break;}
70         cnt=0; cl(vst);
71         dfs(root,0);
72         rmpinit();
73
74         scanf("%d%d",&u,&v);
75         if (r[u]<=r[v]) printf("%d\n", e[query(r[u], r[v])]);
76         else printf("%d\n", e[query(r[v], r[u])]);
77     }
78     return 0;
79 }

```

## 1.4 rmq2d

```
1 | const dl _eps=1e-6;
```

```

2  const int N = 301;
3  int t,n;
4  int dp[N][N][9][9];
5  void in(int &a)
6  {
7      char c,f;
8      while(((f=getchar())<'0' || f>'9')&&f!='-');
9      c=(f=='-')?getchar():f;
10     for(a=0;c>='0'&&c<='9';c=getchar()) a=a*10+c-'0';
11     if(f=='-') a=-a;
12 }
13 void initrmq(){
14     int i,j;
15     int m = log(double(n)) / log(2.0);
16     fr(i,0,m+1){
17         fr(j,0,m+1){
18             if(i==0 && j==0) continue;
19             for(int r = 0; r+(1<<i)-1 < n; ++r){
20                 for(int c = 0; c+(1<<j)-1 < n; ++c){
21                     if(i == 0) dp[r][c][i][j] = min(dp[r][c][i][j-1],
22                     else dp[r][c][i][j] = min(dp[r][c][i-1][j], dp[r+(
23                 }
24             }
25         }
26     }
27 }
28 int rmq_2d_query(int X1,int Y1,int X2,int Y2){
29     int x = log(double(X2 - X1 + 1)) / log(2.0);
30     int y = log(double(Y2 - Y1 + 1)) / log(2.0);
31     int m1 = dp[X1][Y1][x][y];
32     int m2 = dp[X2-(1<<x)+1][Y1][x][y];
33     int m3 = dp[X1][Y2-(1<<y)+1][x][y];
34     int m4 = dp[X2-(1<<x)+1][Y2-(1<<y)+1][x][y];
35     return min(min(m1,m2),min(m3,m4));
36 }
37
38 void inp(){
39     int i,j,m,X1,Y1,X2,Y2;
40     in(n);
41     fr(i,0,n){
42         fr(j,0,n){
43             in(dp[i][j][0][0]);
44         }
45     }
46     initrmq();
47     sfint(m);
48     while(m--){
49         in(X1); in(Y1); in(X2); in(Y2);
50         printf("%d\n",rmq_2d_query(X1-1,Y1-1,X2-1,Y2-1));
51     }
52 }

```

```

53 int main(){
54     fi;
55     sfint(t);
56     while(t--){
57         inp();
58     }
59     return 0;
60 }

```

## 1.5 划分树

```

1  int a[100001];
2  int b[21][100001];
3  int sum[21][100001];    //sum[i表示]l——这些点中有多少个进入了左子树。i
4  int n,m;
5
6  void build(int l,int r,int d){    //代表在树上第几层d
7      if (l==r) return;
8      int i ,mid=(l+r)>>1,id1=l , id2=mid+1,midsum=0;
9      for ( i=mid; i>=l&&a[i]==a[mid]; i--)midsum+=1;
10     for ( i=l; i<=r; i++){
11         sum[d][i]=i==l?0:sum[d][i-1];
12         if (b[d][i]<a[mid]){
13             b[d+1][id1++]=b[d][i];
14             sum[d][i]+=1;
15         }
16         else if (b[d][i]==a[mid]&&midsum){
17             midsum-=1;
18             b[d+1][id1++]=b[d][i];
19             sum[d][i]+=1;
20         }
21         else b[d+1][id2++]=b[d][i];
22     }
23     build(l,mid,d+1);
24     build(mid+1,r,d+1);
25 }
26
27 int search(int x,int y,int k){
28     int l=1,r=n,d=0;
29     int ls,rs,mid;
30     while (x!=y){
31         ls=x==l?0:sum[d][x-1];    //因为要包含x
32         rs=sum[d][y];
33         mid=(l+r)>>1;
34         if (k<=rs-ls) {            //在左子树上
35             x=l+ls;
36             y=l+rs-1;
37             r=mid;
38         }
39         else                        //在右子树上
40         {

```



```

41 |         x=mid+1+x-l-ls;    // (x-l-ls是指处在)前面且进入右子树的个数,
    | 因为在子树中保持位置顺序不变,所以在右子树中前面有xx(x-l-ls个数。)
42 |         y=mid+1+y-l-rs;
43 |         k--rs-ls;
44 |         l=mid+1;
45 |     }
46 |     d+=1;
47 | }
48 | return b[d][x];
49 | }
50 |
51 | int main(){
52 |     freopen("in.txt","r",stdin);
53 |     int cnt=1;
54 |     while( scanf("%d",&n)!=EOF){
55 |         int i,x,y,t;
56 |         for (i=1;i<=n;i++){
57 |             scanf("%d",&t);
58 |             a[i]=b[0][i]=t;
59 |         }
60 |         sort(a+1,a+n+1);
61 |         build(1,n,0);
62 |         scanf("%d",&m);
63 |         printf("Case %d:\n",cnt++);
64 |         while(m--){
65 |             scanf("%d%d",&x,&y);
66 |             int k=(y-x+1)/2+1;
67 |             printf("%d\n",search(x,y,k));
68 |         }
69 |     }
70 |     return 0;
71 | }

```

## 1.6 扫描线矩形面积并

```

1 | const int N = 400000;
2 | int n;
3 | struct ARR {
4 |     int a[N];
5 |     int tot;
6 |     void init(){ tot = 0;}
7 |     void add(int x){
8 |         a[tot++] = x;
9 |     }
10 |     void uni() {
11 |         sort(a,a+tot);
12 |         tot = unique(a,a+tot)-a;
13 |     }
14 |     int fd(int x){
15 |         return lower_bound(a,a+tot,x)-a;
16 |     }
17 | }A;

```

```

18 struct Line{
19     int s,e,y,f;
20     bool operator < (const Line & l) const {
21         if( y == l.y) return s < l.s;
22         return y < l.y;
23     }
24 }l[N];
25 int tot;
26 void add_line(int s,int e,int y,int f){
27     if (s == e) return ;
28     A.add(s);A.add(e);
29     l[tot].s = s;l[tot].e = e;l[tot].y = y;l[tot++].f = f;
30 }
31 void init(){
32     tot = 0;A.init();
33     sfint(n);
34     int x,y,h;
35     fr(i , 0 ,n){
36         sfint3(x,y,h);
37         add_line(x,y,0,1);
38         add_line(x,y,h,-1);
39     }
40     /*int x1, y1, x2, y2, x3, y3, x4, y4;
41     fr(i , 0 ,n){
42         scanf("%d%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3,&x4,&y4);
43         add_line(x1,x3,y1,1); add_line(x1,x3,y2,-1);
44         add_line(x3,x4,y1,1); add_line(x3,x4,y3,-1);
45         add_line(x3,x4,y4,1); add_line(x3,x4,y2,-1);
46         add_line(x4,x2,y1,1); add_line(x4,x2,y2,-1);
47     }*/
48     A.uni();
49 }
50
51 struct SEGT{
52     struct SEGtr
53     {
54         int l,r,cov;
55         ll len;
56     }tr[N*4];
57     void build(int rt,int l,int r){
58         tr[rt].l = l;tr[rt].r = r;tr[rt].cov = 0;tr[rt].len = 0;
59         if(l == r){
60             return;
61         }
62         int mid = (l+r)>>1;
63         build(rt<<1,l,mid);
64         build(rt<<1|1,mid+1,r);
65     }
66     void up(int rt){
67         if(tr[rt].cov != 0) tr[rt].len = A.a[tr[rt].r+1]-A.a[tr[rt].l];
68         else if( tr[rt].l == tr[rt].r) tr[rt].len = 0;

```

```

69         else {
70             tr[rt].len=tr[rt<<1].len+tr[rt<<1|1].len;
71         }
72     }
73     void update(int rt,int l,int r,int add){
74         if(tr[rt].l >= l && tr[rt].r <= r) {
75             tr[rt].cov += add;
76             up(rt);
77             return ;
78         }
79         int mid = (tr[rt].l + tr[rt].r)>>1;
80         if(r <= mid)
81             update(rt<<1,l,r,add);
82         else if(l >mid)
83             update(rt<<1|1,l,r,add);
84         else{
85             update(rt<<1,l,mid,add);
86             update(rt<<1|1,mid+1,r,add);
87         }
88         up(rt);
89     }
90 }S;
91 void sol(){
92     sort(l,l+tot);
93     S.build(1,0,A.tot-2);
94     S.update(1,A.fd(l[0].s),A.fd(l[0].e)-1,l[0].f);
95     ll ans = 0;
96     fr(i,l,tot){
97         ans += (ll(l[i].y - l[i-1].y))*ll(S.tr[1].len);
98         S.update(1,A.fd(l[i].s),A.fd(l[i].e)-1,l[i].f);
99     }
100     printf("%lld\n",ans);
101 }

```

## 2 数学

### 2.1 素数

#### 2.1.1 筛素数

```

1 bool flag[N+1];
2 int prime[N+1];
3 int totpri;
4 void getpri(){
5     int n=N;
6     int i,j;totpri=0;
7     for(i=2;i<=n;++i) { /*筛选素数快速的方法*/
8         if(!flag[i]) prime[totpri++]=i;
9         for(j=0;j<totpri&&i*prime[j]<=n;++j)
10            {
11                flag[i*prime[j]]=1;
12                if(i%prime[j]==0) break;
13            }
14     }
15 }
```

#### 2.1.2 Miller-Rabbin

```

1 bool primeTest(ll n, ll b) {
2     ll m = n - 1;
3     ll counter = 0;
4     while ((m & 1) == 0) {
5         m >>= 1;
6         counter ++;
7     }
8     ll ret = pow_mod(b, m, n);
9     if (ret == 1 || ret == n - 1) {
10         return true;
11     }
12     counter --;
13     while (counter >= 0) {
14         ret = add_mod(ret, ret, n);
15         if (ret == n - 1) {
16             return true;
17         }
18         counter --;
19     }
20     return false;
21 }
22
23 const int BASIC[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
24
25 bool isPrime(ll n) {
26     if (n < 2) {
27         return false;
28     }
```

```

29     if (n < 4) {
30         return true;
31     }
32     if (n == 3215031751LL) {
33         return false;
34     }
35     for (int i = 0; i < 12 && BASIC[i] < n; ++ i) {
36         if (!primeTest(n, BASIC[i])) {
37             return false;
38         }
39     }
40     return true;
41 }

```

## 2.2 Gcd

```

1 int gcd(int a, int b)
2 {
3     if (b == 0) return a;
4     return gcd(b, a % b);
5 }

```

## 2.3 extend-gcd

### 2.3.1 求模线性方程

```

1 int e_gcd(int a, int b, int &x, int &y)
2 {
3     if (b == 0)
4         {x = 1; y = 0; return a;}
5     int ans = e_gcd(b, a % b, x, y);
6     int temp = x;
7     x = y;
8     y = temp - (a / b) * y;
9     return ans;
10 }

```

### 2.3.2 求逆

```

1 // a * x + b * y = gcd(a, b)
2 long long extGcd(long long a, long long b, long long& x, long long& y)
3     if (b == 0) {
4         x = 1;
5         y = 0;
6         return a;
7     } else {
8         int g = extGcd(b, a % b, y, x);
9         y -= a / b * x;
10        return g;
11    }
12 }
13
14 // ASSUME: gcd(a, m) == 1

```

```

15 long long modInv(long long a, long long m) {
16     long long x, y;
17     extGcd(a, m, x, y);
18     return (x % m + m) % m;
19 }

```

## 2.4 快速加

```

1 ll add_mod(ll a, ll b, ll m){
2     ll ans=0;
3     a%=m;
4     while(b){
5         if(b&1) ans=(ans+a)%m;
6         a=(a+a)%m;
7         b>>=1;
8     }
9     return ans%m;
10 }

```

## 2.5 快速幂

```

1 ll pow_mod(ll a, ll b, ll m) {
2     ll ans=1;
3     a%=m;
4     while(b)
5     {
6         if(b&1) ans=(ans*a)%m;
7         a=(a*a)%m;
8         b>>=1;
9     }
10    return ans;
11 }

```

## 2.6 矩阵类

```

1 ll MOD=10000;
2 template<int MAXN=1010,int MAXM=1010, typename T = int>
3 struct Mat{
4     int n,m;
5     T a[MAXN][MAXM];
6     Mat(int _n=0,int _m=0):n(_n),m(_m){}
7     void clear(){
8         memset(a,0,sizeof(a));
9     }
10    void identity(){
11        memset(a,0,sizeof(a));
12        for(i=0,n){
13            a[i][i] = 1;
14        }
15    }
16    Mat operator + (const Mat &b) const{
17        Mat tmp(n,m);
18        for(int i=0;i<n;++i){

```

```

19         for (int j = 0; j < m; ++j) {
20             tmp.a[i][j] = a[i][j] + b.a[i][j];
21         }
22     }
23     return tmp;
24 }
25 Mat operator - (const Mat &b) const {
26     Mat tmp(n, m);
27     for (int i = 0; i < n; ++i) {
28         for (int j = 0; j < m; ++j) {
29             tmp.a[i][j] = a[i][j] - b.a[i][j];
30         }
31     }
32     return tmp;
33 }
34 Mat operator * (const Mat &b) const {
35     Mat tmp(n, m);
36     tmp.clear();
37     for (int i = 0; i < n; ++i) {
38         for (int j = 0; j < m; ++j) {
39             for (int k = 0; k < n; ++k) {
40                 tmp.a[i][j] = (tmp.a[i][j] + a[i][k] * b.a[k][j]) % MOD;
41             }
42         }
43     }
44     return tmp;
45 }
46 Mat operator ^ (int b) {
47     Mat ret(n, m);
48     ret.identity();
49     while (b) {
50         if (b & 1)
51             ret = (*this) * ret;
52         (*this) = (*this) * (*this);
53         b >>= 1;
54     }
55     return ret;
56 }
57 void disp() {
58     for (int i = 0; i < n; ++i) {
59         for (int j = 0; j < m; ++j) {
60             printf("%d ", a[i][j]);
61         }
62         puts("");
63     }
64 };

```

## 2.7 容斥

```

1 ll lim;
2 ll dfs(int pos, ll d) {
3     ll ret = 0;

```

```

4     while(pos<totpri&&prime[pos] <= k&& prime[pos] * d<=lim){
5         ret += lim/(prime[pos]*d) - dfs(pos+1,d*prime[pos]);
6         pos++;
7     }
8     return ret;
9 }

```

## 2.8 组合数

### 2.8.1 暴力求解

```

1 C(n,m)=n*(n-1)*...*(n-m+1)/m, !n<=15
2 int Combination(int n, int m)
3 {
4     const int M = 10007;
5     int ans = 1;
6     for(int i=n; i>=(n-m+1); --i)
7         ans *= i;
8     while(m)
9         ans /= m--;
10    return ans % M;
11 }

```

### 2.8.2 打表

```

1 C(n,m)=C(n-1,m-1)+C(n-1,m, )n<=10,000
2 const int M = 10007;
3 const int N = 1000;
4 ll C[N][N];
5 void initc(){
6     int i,j;
7     for(i=0; i<N; ++i){
8         C[0][i] = 0;
9         C[i][0] = 1;
10    }
11    for(i=1; i<N; ++i){
12        for(j=1; j<N; ++j)
13            C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
14    }
15 }

```

### 2.8.3 质因数分解

$C(n,m)=n!/(m!(n-m)!)$ ,  $C(n,m)=p_1a_1-b_1-c_1p_2a_2-b_2-c_2p_kak-b_k-ck, n \leq 10,000,000$

```

1 //用筛法生成素数
2 const int MAXN = 1000000;
3 bool arr[MAXN+1] = {false};
4 vector<int> produce_prim_number(){
5     vector<int> prim;
6     prim.push_back(2);
7     int i,j;
8     for(i=3; i*i<=MAXN; i+=2){
9         if(!arr[i]){

```



```

10         prim.push_back(i);
11         for(j=i*i; j<=MAXN; j+=i)
12             arr[j] = true;
13     }
14 }
15 while(i<=MAXN){
16     if(!arr[i])
17         prim.push_back(i);
18     i+=2;
19 }
20 return prim;
21 }
22
23 //计算n中素因子!的指数p
24 int Cal(int x, int p){
25     int ans = 0;
26     long long rec = p;
27     while(x>=rec){
28         ans += x/rec;
29         rec *= p;
30     }
31     return ans;
32 }
33
34 //计算的次方对取模, 二分法nkM
35 int Pow(long long n, int k, int M){
36     long long ans = 1;
37     while(k){
38         if(k&1){
39             ans = (ans * n) % M;
40         }
41         n = (n * n) % M;
42         k >>= 1;
43     }
44     return ans;
45 }
46
47 //计算C(n,m)
48 int Combination(int n, int m){
49     const int M = 10007;
50     vector<int> prim = produce_prim_number();
51     long long ans = 1;
52     int num;
53     for(int i=0; i<prim.size() && prim[i]<=n; ++i){
54         num = Cal(n, prim[i]) - Cal(m, prim[i]) - Cal(n-m, prim[i]);
55         ans = (ans * Pow(prim[i], num, M)) % M;
56     }
57     return ans;
58 }

```

## 2.8.4 Lucas

/\* 定理, 将  $m, n$  化为  $p$  进制, 有:  $C(n, m) = C(n_0, m_0) * C(n_1, m_1) \dots \pmod{p}$ , 算一个不是很大的  $C(n, m) \% p$ ,  $p$  为素数, 化为线性同余方程, 用扩展的欧几里德定理求解,  $n$  在 `int` 范围内, 修改一下可以满足 `long long` 范围内。\*/

```

1
2
3 const int M = 10007;
4 int ff[M+5]; //打表, 记录n, 避免重复计算!
5
6 //求最大公因数
7 int gcd(int a, int b){
8     if(b==0)
9         return a;
10    else
11        return gcd(b, a%b);
12 }
13
14 //解线性同余方程, 扩展欧几里德定理
15 int x, y;
16 void Extended_gcd(int a, int b){
17     if(b==0) {
18         x=1;
19         y=0;
20     }
21     else{
22         Extended_gcd(b, a%b);
23         long t=x;
24         x=y;
25         y=t-(a/b)*y;
26     }
27 }
28
29 //计算不大的C(n,m)
30 int C(int a, int b){
31     if(b>a)
32         return 0;
33     b=(ff[a-b]*ff[b])%M;
34     a=ff[a];
35     int c=gcd(a, b);
36     a/=c;
37     b/=c;
38     Extended_gcd(b, M);
39     x=(x+M)%M;
40     x=(x*a)%M;
41     return x;
42 }
43
44 //定理Lucas
45 int Combination(int n, int m){

```

```

46     int ans=1;
47     int a,b;
48     while(m|n){
49         a=n%M;
50         b=m%M;
51         n/=M;
52         m/=M;
53         ans=(ans*C(a,b))%M;
54     }
55     return ans;
56 }
57
58 int main(void){
59     int i,m,n;
60     ff[0]=1;
61     for(i=1;i<=M;i++) //预计算n!
62         ff[i]=(ff[i-1]*i)%M;
63
64     scanf("%d%d",&n,&m);
65     printf("%d\n",func(n,m));
66
67     return 0;
68 }

```

## 2.9 pollardRho

```

1  vector <ll> divisors;
2  ll pollardRho(ll n, ll seed) {
3      ll x, y;
4      x = y = rand() % (n - 1) + 1;
5      ll head = 1;
6      ll tail = 2;
7      while (true) {
8          x = pow_mod(x,2, n);
9          x = add_mod(x, seed, n);
10         if (x == y) {
11             return n;
12         }
13         ll d = gcd(abs(x - y), n);
14         if (1 < d && d < n) {
15             return d;
16         }
17         head ++;
18         if (head == tail) {
19             y = x;
20             tail <<= 1;
21         }
22     }
23 }
24 void factorize(ll n) {
25     if (n > 1) {

```

```

26         if (isPrime(n)) {
27             divisors.push_back(n);
28         } else {
29             ll d = n;
30             while (d >= n) {
31                 d = pollardRho(n, rand() % (n - 1) + 1);
32             }
33             factorize(n / d);
34             factorize(d);
35         }
36     }
37 }

```

## 2.10 欧拉函数

### 2.10.1 一般的求法

```

1  const int N = 100010;
2  bool is_prime[N];
3  ll phi[N];
4  ll prime[N];
5  void init(){
6      ll i, j, k = 0;
7      phi[1] = 1;
8      for(i = 2; i < N; i++){
9          if(is_prime[i] == false){
10             prime[k++] = i;
11             phi[i] = i - 1;
12         }
13         for(j = 0; j < k && i * prime[j] < N; j++){
14             is_prime[i * prime[j]] = true;
15             if(i % prime[j] == 0){
16                 phi[i * prime[j]] = phi[i] * prime[j];
17                 break;
18             }
19             else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
20         }
21     }
22 }

```

### 2.10.2 递推

```

1  for (i = 1; i <= maxn; i++) phi[i] = i;
2  for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
3  for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
4      for (j = i; j <= maxn; j += i)
5          phi[j] = phi[j] / i * (i - 1);
6  }

```

### 2.10.3 单独求

```

1  ll Euler_Phi(ll n)
2  {
3      ll t = n, p = n;

```

```

4      ll sq = sqrt (n);
5
6      for (int i=0;prime[i]<=sq && i<totpri;i++)
7      {
8          if (t%prime[i]==0)
9          {
10             p = p/prime[i]*(prime[i]-1);
11
12             while (t%prime[i]==0)
13                 t/=prime[i];
14
15             //sq = sqrt(t);
16         }
17
18         if (t == 1)
19             break;
20     }
21
22     if (t > 1)
23         p = p/t*(t-1);
24
25     return p;
26 }

```

## 2.11 高斯消元

### 2.11.1 模二消元

```

1  int gauss(int n){
2      int r,c;
3      for(r = 0, c=0;r<n,c<n;++r,++c){
4          int p = r;
5          fr(i, r+1,n){
6              if(a[i][c] > a[p][c]) p = i;
7          }
8          if (p != r){
9              fr(i, c,n+1){
10                 swap(a[p][i],a[r][i]);
11             }
12         }
13         if(a[r][c] == 0){
14             r--;continue;
15         }
16         fr(i, 0,n){
17             if (a[i][c] == 0||i == r) continue;
18             fr(j,c,n+1) a[i][j] = a[i][j]^a[r][j];
19         }
20     }
21     fr(i, r,n) if (a[i][n]) return -1;
22     return n-r;
23 }

```

## 2.11.2 浮点

```

1  const double eps = 1e-12;
2  const int MAXN = 30;
3  inline int gauss(double a[][4], bool l[], double ans[], const int &n){
4      int res = 0, r = 0;
5      for(int i = 0; i < n; ++i) l[i] = false;
6      for(int i = 0; i < n; ++i) {
7          for(int j = r; j < n; ++j){
8              if( fabs(a[j][i]) > eps) {
9                  for(int k = i; k <= n; ++k) swap(a[j][k], a[r][k]);
10                 break;
11             }
12         }
13         if( fabs(a[r][i]) < eps){
14             ++res;
15             continue;
16         }
17         for(int j = 0; j < n; ++j){
18             if( j != r && fabs(a[j][i]) > eps){
19                 double tmp = a[j][i] / a[r][i];
20                 for(int k = i; k <= n; ++k){
21                     a[j][k] -= tmp * a[r][k];
22                 }
23             }
24         }
25         l[i] = true; ++r;
26     }
27
28     fr(i, 0, n){
29         fr(j, 0, n+1){
30             printf("%lf ", a[i][j]);
31         }
32         puts("");
33     }
34     for(int i = 0; i < n; ++i){ //有问题
35         if (l[i])
36             for(int j = 0; j < n; ++j){
37                 if (fabs(a[j][i]) > 0){
38                     ans[i] = a[j][n] / a[j][i];
39                 }
40             }
41     }
42     return res;
43 }

```

## 2.12 格雷码

生成 reflected gray code

每次调用 gray 取得下一个码

000...000 是第一个码, 100...000 是最后一个码

```

1 void gray(int n,int *code){
2     int t=0,i;
3     for (i=0;i<n;t+=code[i++]);
4     if (t&1)
5         for (n--;!code[n];n--);
6         code[n-1]=1-code[n-1];
7 }

```

## 2.13 离散对数

```

1 #define MAXN 131071
2 struct HashNode {
3     ll data, id, next;
4 };
5 HashNode hash[MAXN<<1];
6 bool flag[MAXN<<1];
7 ll top;
8
9 void Insert ( ll a, ll b )
10 {
11     ll k = b & MAXN;
12     if ( flag[k] == false )
13     {
14         flag[k] = true;
15         hash[k].next = -1;
16         hash[k].id = a;
17         hash[k].data = b;
18         return;
19     }
20     while( hash[k].next != -1 )
21     {
22         if( hash[k].data == b ) return;
23         k = hash[k].next;
24     }
25     if ( hash[k].data == b ) return;
26     hash[k].next = ++top;
27     hash[top].next = -1;
28     hash[top].id = a;
29     hash[top].data = b;
30 }
31
32 ll Find ( ll b )
33 {
34     ll k = b & MAXN;
35     if( flag[k] == false ) return -1;
36     while ( k != -1 )
37     {
38         if( hash[k].data == b ) return hash[k].id;
39         k = hash[k].next;
40     }
41     return -1;

```

```

42 }
43
44 ll gcd ( ll a, ll b )
45 {
46     return b ? gcd ( b, a % b ) : a;
47 }
48
49 ll ext_gcd ( ll a, ll b, ll& x, ll& y )
50 {
51     ll t, ret;
52     if ( b == 0 )
53     {
54         x = 1, y = 0;
55         return a;
56     }
57     ret = ext_gcd ( b, a % b, x, y );
58     t = x, x = y, y = t - a / b * y;
59     return ret;
60 }
61 ll mod_exp ( ll a, ll b, ll n )
62 {
63     ll ret = 1;
64     a = a % n;
65     while ( b >= 1 )
66     {
67         if ( b & 1 )
68             ret = ret * a % n;
69         a = a * a % n;
70         b >>= 1;
71     }
72     return ret;
73 }
74
75 ll BabyStep_GiantStep ( ll A, ll B, ll C ) //  $A^x \equiv B \pmod C$ 
76 {
77     memset(flag, 0, sizeof(flag));
78     top = MAXN; B %= C;
79     ll tmp = 1, i;
80     for ( i = 0; i <= 100; tmp = tmp * A % C, i++ )
81         if ( tmp == B % C ) return i;
82
83     ll D = 1, cnt = 0;
84     while( (tmp = gcd(A,C)) != 1 )
85     {
86         if ( B % tmp ) return -1;
87         C /= tmp;
88         B /= tmp;
89         D = D * A / tmp % C;
90         cnt++;
91     }
92

```



```

93     ll M = (ll) ceil(sqrt(C+0.0));
94     for ( tmp = 1, i = 0; i <= M; tmp = tmp * A % C, i++ )
95         Insert ( i, tmp );
96
97     ll x, y, K = mod_exp( A, M, C );
98     for ( i = 0; i <= M; i++ )
99     {
100         ext_gcd ( D, C, x, y ); // D * X = 1 ( mod C )
101         tmp = ((B * x) % C + C) % C;
102         if( (y = Find(tmp)) != -1 )
103             return i * M + y + cnt;
104         D = D * K % C;
105     }
106     return -1;
107 }

```

## 2.14 字典序

### 2.14.1 排列

```

1  int perm2num(int n, int *p) {
2      int i, j, ret = 0, k = 1;
3      for ( i = n - 2; i >= 0; k *= n - (i--) )
4          for ( j = i + 1; j < n; j++ )
5              if (p[j] < p[i])
6                  ret += k;
7      return ret;
8  }
9  void num2perm(int n, int *p, int t) {
10     int i, j;
11     for ( i = n - 1; i >= 0; i-- )
12         p[i] = t % (n - i), t /= n - i;
13     for ( i = n - 1; i; i-- )
14         for ( j = i - 1; j >= 0; j-- )
15             if (p[j] <= p[i])
16                 p[i]++;
17 }

```

### 2.14.2 组合

```

1  int comb(int n, int m) {
2      int ret = 1, i;
3      m = m < (n - m) ? m : (n - m);
4      for ( i = n - m + 1; i <= n; ret *= (i++));
5      for ( i = 1; i <= m; ret /= (i++));
6      return m < 0 ? 0 : ret;
7  }
8  int comb2num(int n, int m, int *c) {
9      int ret = comb(n, m), i;
10     for ( i = 0; i < m; i++ )
11         ret -= comb(n - c[i], m - i);
12     return ret;
13 }

```

```

14 void num2comb(int n, int m, int* c, int t) {
15     int i, j = 1, k;
16     for (i = 0; i < m; c[i++] = j++)
17         for (; t > (k = comb(n - j, m - i - 1)); t -= k, j++);
18 }

```

## 2.15 置换 polya

求置换的循环节, polya 原理

perm[0..n-1] 为 0..n-1 的一个置换 (排列)

返回置换最小周期, num 返回循环节个数

```

1 #define MAXN 1000
2 int polya(int* perm, int n, int& num) {
3     int i, j, p, v[MAXN] = {0}, ret = 1;
4     for (num = i = 0; i < n; i++)
5         if (!v[i]) {
6             for (num++, j = 0, p = i; !v[p = perm[p]]; j++)
7                 v[p] = 1;
8             ret *= j / gcd(ret, j);
9         }
10    return ret;
11 }

```

## 3 string

### 3.1 kmp

$\text{next}[j]$  的值表示  $P[0..j-1]$  中最长后缀的长度等于相同字符序列的前缀。

$j$  为最远位置使得  $\text{mod}[0..j-1] == \text{mod}[i-j+1..i]$

$\text{next}$  的值就是每个  $j$   $\text{next}$  往前

重要的是理解那个往前

例: ababa  $\text{next}$  为: 0 0 1 2 3

那么  $\text{next}[\text{next}[5]] = \text{next}[3] = 2$ ; 即  $s[0..1] = s[2..3]$

性质:  $\text{len} - \text{next}[\text{len}-1]$  就是从 0 开始最短的串能重复出整个串, 比如 abcabcab 就只需要 abc 就能重复完

```

1 | int len1, len2, nxt[10005];
2 | char mod[10005], s[1000005];
3 | void get_nxt(char mod[], int len){
4 |     int i, j=0;
5 |     nxt[0]=0;
6 |     for(i=1; i<len; i++){
7 |         while(j>0 && mod[j]!=mod[i]) j=nxt[j-1];
8 |         if(mod[j]==mod[i]) j++;
9 |         nxt[i]=j;
10 |    }
11 | }
12 | int KMP(int len1, int len2, char s[], char mod[], int pos = 0)
13 | {
14 |     int i=pos, j=0, ret=0;
15 |     while(i<len1){
16 |         while(j && mod[j]!=s[i]) j=nxt[j-1];
17 |         if(mod[j]==s[i]){
18 |             if(++j==(len2)) ret++;
19 |         }
20 |     }
21 |     return ret;
22 | }
```

### 3.2 kmp

$q$  是 B 串继续向后匹配的指针,  $p$  是 A 串继续向后匹配的指针, 也是曾经到达过的最远位置 +1

$q$  在每次计算后会减小 1, 直观的讲就是 B 串向后错了一位

```

1 | const int N=100010;
2 |
3 | int len_s, len_t;
4 | int nxt[N], extend[N];
5 | char S[N], T[N];
6 | void build_nxt()
7 | {
8 |     int k, q, p, a;
9 |     nxt[0] = len_t;
```

```

10     for (k = 1, q = -1; k < len_t; k ++, q --) {
11         if (q < 0 || k + nxt[k - a] >= p) {
12             if (q < 0) q = 0, p = k;
13             while (p < len_t && T[p] == T[q]) {
14                 p ++, q ++;
15             }
16             nxt[k] = q, a = k;
17         }
18         else {
19             nxt[k] = nxt[k - a];
20         }
21     }
22 }
23 void extend_KMP()
24 {
25     int k, q, p, a;
26     for (k = 0, q = -1; k < len_s; k ++, q --) {
27         if (q < 0 || k + nxt[k - a] >= p) {
28             if (q < 0) q = 0, p = k;
29             while (p < len_s && q < len_t && S[p] == T[q]) {
30                 p ++, q ++;
31             }
32             extend[k] = q, a = k;
33         }
34         else {
35             extend[k] = nxt[k - a];
36         }
37     }
38 }
39 int main() {
40     fi;
41     scanf("%s", S);
42     scanf("%s", T);
43
44     len_t = strlen(T);
45     len_s = strlen(S);
46     build_nxt();
47     extend_KMP();
48
49     return 0;
50 }

```

### 3.3 manacher

最长回文子串模板

hdu3068, 最长回文子串模板, Manacher 算法, 时间复杂度  $O(n)$ , 相当快  
str 是这样一串字符串 (下标从 1 开始):

举例: 若原字符串为 "abcd", 则 str 为 "\$#a#b#c#d#", 最后还有一个终止符。

n 为 str 的长度, 若原字符串长度为 nn, 则  $n = 2 * nn + 2$ 。

rad[i] 表示回文的半径, 即最大的 j 满足  $str[i-j+1...i] = str[i+1...i+j]$ ,

而 rad[i]-1 即为以 str[i] 为中心的回文子串在原串中的长度

```

1 #define M 20000050
2 char str1[M],str[2*M]; //start from index 1
3 int rad[M],nn,n;
4 void Manacher(int *rad,char *str,int n)
5 {
6     int i;
7     int mx = 0;
8     int id;
9     for(i=1; i<n; i++)
10    {
11        if( mx > i ) rad[i] = rad[2*id-i]<mx-i?rad[2*id-i]:mx-i;
12        else rad[i] = 1;
13        for(; str[i+rad[i]] == str[i-rad[i]]; rad[i]++);
14        if( rad[i] + i > mx )
15        {
16            mx = rad[i] + i;
17            id = i;
18        }
19    }
20 }
21 struct PLD{
22     int l,r;
23     PLD(int x=0,int y = -1):l(x),r(y){}
24 }p[N];
25
26 void getlr(int n){
27     fr(i,2,n){
28         p[i].l = i-rad[i]+1;
29         p[i].l = (p[i].l+1)/2-1;
30         p[i].r = p[i].l+rad[i]-2;
31     }
32 }
33
34
35 int main()
36 {
37     int i,ans,Case=1;
38     while( scanf("%s",str1)!=EOF)
39     {
40         nn=strlen(str1);
41         n=2*nn+2;
42         str[0]='$';
43         for(i=0;i<=nn;i++)
44         {
45             str[2*i+1]='#';
46             str[2*i+2]=str1[i];
47         }
48         Manacher(rad,str,n);
49         ans=1;
50         for(i=0;i<n;i++)

```

```

51         ans=rad[i]>ans?rad[i]:ans;
52         printf("%d\n",ans-1);
53     }
54     return 0;
55 }

```

### 3.4 ac 自动机

```

1  char str[2000010];
2  char c[1010][55];
3
4  namespace AC {
5  const int dict = 26;
6  const int root = 0;
7  const int maxn = 3000000;
8  struct node {
9      int son[dict], fail, idx;
10 } tree[maxn];
11 int apr[10010];
12 bool vis[3000000];
13 int sz;
14 int initNode(int idx) {
15     memset(tree[idx].son, 0, sizeof(tree[idx]));
16     tree[idx].fail = tree[idx].idx = 0;
17     return idx;
18 }
19 void init() {
20     sz = initNode(0);
21     memset(apr, 0, sizeof(apr));
22 }
23 void ins(char *s, int idx) {
24     int cur = root, t;
25     while (*s) {
26         t = *s - 'A';
27         if (!tree[cur].son[t]) tree[cur].son[t] = initNode(++sz);
28         cur = tree[cur].son[t];
29         s++;
30     }
31     tree[cur].idx = idx;
32 }
33 queue<int> q;
34 void buildac() {
35     while(!q.empty()) q.pop();
36     int i, cur, nxt, f;
37     for (i = 0; i < dict; i++)
38         if (tree[root].son[i]) q.push(tree[root].son[i]);
39
40     while (!q.empty()) {
41         cur = q.front();
42         q.pop();
43         f = tree[cur].fail;

```

```

44         for ( i = 0 ; i < dict ; i++ )
45             if (tree[cur].son[i]) {
46                 nxt = tree[cur].son[i];
47                 tree[nxt].fail = tree[f].son[i];
48                 q.push(nxt);
49             } else tree[cur].son[i] = tree[f].son[i];
50     }
51 }
52 void search(char *s) {
53     int i, cur = 0;
54     for ( ; *s ; s++ ) {
55         if( (*s) >= 'A' && (*s) <= 'Z' ) {
56             cur = tree[cur].son[*s - 'A'];
57             for ( i = cur ; i ; i = tree[i].fail ) { //用于优化vis
58                 apr[tree[i].idx]++;
59             }
60         } else {
61             cur = 0;
62         }
63     }
64     for(int i = 1; i <= 1010; ++i) {
65         if(apr[i]) {
66             printf("%s: %d\n", c[i], apr[i]);
67         }
68     }
69 }
70 };
71
72 int main() {
73     // freopen("input.txt", "r", stdin);
74     int n;
75
76     while(scanf("%d", &n) != EOF) {
77         AC::init();
78         for(int i = 1; i <= n ; ++i) {
79             scanf("%s", c[i]);
80             AC::ins(c[i], i);
81         }
82         AC::buildac();
83         getchar();
84         gets(str);
85         AC::search(str);
86     }
87
88     return 0;
89 }

```

### 3.5 后缀数组

da 函数的参数  $m$  代表字符串中字符的取值范围，是基数排序的一个参数，如果原序列都是字母可以直接取 128，如果原序列本身都是整数的话，则  $m$  可以取比最大的整数大 1 的值。  
 $height[i] = LCP(i-1, i)$   $LCP(i, j) = lcp(\text{Suffix}(SA[i]), \text{Suffix}(SA[j]))$

就是从  $sa[i]$  开始的后缀与从  $sa[j]$  开始的后缀的最长公共前缀  
 $LCP(i,j)=\min height[k] \mid i+1 \leq k \leq j$  此时的  $i, j$  为 suffix 的对应值  
 例: abaca  
 rk 2 4 3 5 1 0  
 sa 5 4 0 2 1 3  
 height 0 0 1 1 0 0

```

1  const int maxn = 2010;
2  int wa[maxn], wb[maxn], wv[maxn], wss[maxn], sa[maxn];
3  bool cmp(int *r, int a, int b, int l)
4  {
5      return r[a]==r[b]&&r[a+l]==r[b+l];
6  }
7  void da(int *r, int *sa, int n, int m)
8  {
9      int i, j, p, *x=wa, *y=wb, *t;
10
11     for(i=0; i<m; i++) wss[i]=0;
12     for(i=0; i<n; i++) wss[x[i]=r[i]]++;
13     for(i=1; i<m; i++) wss[i]+=wss[i-1];
14     for(i=n-1; i>=0; i--) sa[--wss[x[i]]]=i;
15     for(j=1, p=1; p<n; j*=2, m=p)
16     {
17         for(p=0, i=n-j; i<n; i++) y[p++]=i;
18         for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;
19         for(i=0; i<n; i++) wv[i]=x[y[i]];
20         for(i=0; i<m; i++) wss[i]=0;
21         for(i=0; i<n; i++) wss[wv[i]]++;
22         for(i=1; i<m; i++) wss[i]+=wss[i-1];
23         for(i=n-1; i>=0; i--) sa[--wss[wv[i]]]=y[i];
24         for(t=x, x=y, y=t, p=1, x[sa[0]]=0, i=1; i<n; i++)
25             x[sa[i]]=cmp(y, sa[i-1], sa[i], j)?p-1:p++;
26     }
27 }
28 int rk[maxn], height[maxn];
29 void calheight(int *r, int *sa, int n)
30 {
31     int i, j, k=0;
32     cl(rk);
33     for(i=1; i<=n; i++) rk[sa[i]]=i;
34     for(i=0; i<n; height[rk[i++]]=k)
35         for(k?k--:0, j=sa[rk[i]-1]; r[i+k]==r[j+k]; k++);
36 }
37 int dp[100010][18];
38 void rmqInit(int n){
39     fr(i, 0, n) dp[i][0] = height[i+2];
40     int k = (int)(log(n * 1.0) / log(2.0)); k++;
41     fr(j, 1, k){
42         for(int i = 0; i+(1<<j)-1 < n; ++i){
43             dp[i][j] = min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
44         }
45     }
46 }

```



```

45     }
46 }
47 inline int query(int l ,int r){
48     int k = (int)(log(r * 1.0 - l + 1) / log(2.0));
49     return min(dp[l][k] , dp[r-(1<<k)+1][k]);
50 }
51 int lcp(int l,int r){
52     int t ;
53     l = rk[l] , r = rk[r];
54     if(l>r) l^=r^=l^=r;
55     return query(l-1,r-2);
56 }
57
58 bool check(int x,int k){
59     fr(i , 0 ,k-1){
60         if(lcp(i*x,(i+1)*x) < x) return 0;
61     }
62     return 1;
63 }
64 int c[maxn];
65 char str[maxn];
66 int maxrep[maxn];
67 int main(){
68     fi;
69     while(sfstr(str)!=EOF){
70         int len = strlen(str);
71         int k;
72         sfint(k);
73         if( k == 1 ){
74             printf("%lld\n", (long long)len*(long long)(len+1)/2);
75             continue;
76         }
77         ll ans = 0;
78         fr(i , 0 ,len) c[i] = str[i]-'a'+1;
79         c[len] = 0;
80         da(c,sa,len+1,27);
81         calheight(c,sa,len);
82         rmqInit(len-1);
83
84         for(int i = 0; i < len; ++i )    maxrep[i] = 1;
85
86         for(int L = 1; L*k <= len; ++L ) //rep[L次的有多少个]
87         {
88             for(int i = L; i < len; i += L )    if( maxrep[i-L] == 1 )
89             {
90                 int t = lcp(i-L, i);
91                 if( t )
92                 {
93                     int j = 0;
94                     while( j < L && i-L >= j && str[i-L-j] == str[i-j]
95

```

```

96         if( t >= L && lcp(i-L-j, i-j) >= (k-1)*L )
97             maxrep[i-L-j] = max(maxrep[i-L-j], t/L+1);
98         ++t, ++j;
99     }
100 }
101 }
102 }
103 for(int i = 0; i < len; ++i )
104     if( maxrep[i] >= k )
105         ans += ll(maxrep[i] - k + 1);
106 printf("%lld\n", ans);
107 }
108 return 0;
109 }

```

### 3.6 后缀自动机

```

1  const int  maxn=2000010;
2  const int  kinds=26;
3
4  char  ch[maxn];
5
6  struct Sam{
7      Sam *son[kinds],*fa;
8      int  l ,cnt;
9      bool  vst;
10 }a[maxn],*head,*last;
11
12 int  top=-1;
13 void add(int x){
14     Sam *p=&a[++top],*bj=last;
15     p->l=last->l+1;last=p;
16     for(; bj && !bj->son[x]; bj = bj->fa) bj->son[x] = p;
17     if (!bj) p->fa = head;
18     else if (bj->l+1 == bj->son[x]->l) p->fa = bj->son[x];
19     else{
20         Sam *r = &a[ ++ top],*q = bj->son[x];
21         *r = *q ,r->l= bj->l+1, p->fa = q->fa = r;
22         for( ; bj && bj->son[x] == q; bj = bj->fa) bj->son[x] = r;
23     }
24 }
25
26 Sam *b[maxn];
27 Sam *sta[maxn];
28 int  dws[maxn];
29 void caltimes(int n){ // n = lenstr;
30     int  i;
31     for (i = 0; i <= top; ++i) ++dws[a[i].l];
32     for (i = 1; i <= n; ++i) dws[i] += dws[i - 1];
33     for (i = 0; i <= top; ++i) b[--dws[a[i].l]] = &a[i];
34     for (last = head, i = 0; i < n; ++i)

```

```

35         (last = last->son[ch[i] - 'a'])->cnt++;
36
37     for (i = top; i > 0; --i){
38         b[i]->fa->cnt += b[i]->cnt;
39     }
40 }
41
42 int main(){
43     scanf("%s",ch);
44     head = last = &a[++top];
45     int n=strlen(ch);
46     fr(i,0,n) add( ch[i] - 'a' );
47     int i;
48     caltimes(n);
49     return 0;
50 }

```

### 3.7 elfhash

如果最高的四位不为 0，则说明字符多余 7 个，现在正在存第 8 个字符，如果不处理，再加下一个字符时，第一个字符会被移出，因此要有如下处理。

该处理，如果对于字符串 (a-z 或者 A-Z) 就会仅仅影响 5-8 位，否则会影响 5-31 位，因为 C 语言使用的算数移位

因为 1-4 位刚刚存储了新加入到字符，所以不能右移 28

上面这行代码并不会对 X 有影响，本身 X 和 hash 的高 4 位相同，下面这行代码即对 28-31(高 4 位) 位清零。

返回一个符号位为 0 的数，即丢弃最高位，以免函数外产生影响。(我们可以考虑，如果只有字符，符号位不可能为负)

hash 左移 4 位，把当前字符 ASCII 存入 hash 低四位。

```

1 unsigned int ELFHash(char *str)
2 {
3     unsigned int hash = 0;
4     unsigned int x = 0;
5
6     while (*str)
7     {
8         hash = (hash << 4) + (*str++);
9         if ((x = hash & 0xF0000000L) != 0)
10        {
11            hash ^= (x >> 24);
12            hash &= ~x;
13        }
14    }
15    return (hash & 0x7FFFFFFF);
16 }

```

### 3.8 散列 hash

```

1 struct hash_map{
2     const static int P = 999887;

```

```

3   int head[P], next[N], key[N];
4   int sz;
5   inline void init(){
6       cl(head), sz = 0;
7   }
8   inline int find(uint val){
9       int x = val % P;
10      for (int i=head[x]; i; i=next[i])
11          if (key[i] == val) return i;
12      return 0;
13  }
14  inline int insert(uint val){
15      ++sz; key[sz] = val;
16      int x = val % P; next[sz] = head[x]; head[x] = sz;
17      return sz;
18  }
19 } hashed;

```

### 3.9 可获取任意段字符串的 hash

```

1   unsigned int S[N], P[N];
2   void init(char *str, int n){
3       S[0] = 1, P[0] = 1;
4       fr(i, 1, n+1) P[i] = P[i-1]*Z; //是zbase
5       fr(i, 0, n) S[i+1] = S[i]*Z+(str[i]-'a'+1);
6   }
7   int H(PLD x){ //这里获得一段的值hash x.l r 收尾位置
8       int l=x.l; int r=x.r;
9       return S[r+1] - S[l] * P[r-l+1];
10  }

```

## 4 图论

### 4.1 前向星

```

1 | const int N = 1010; const int M = 2010;
2 | struct Edg
3 | {
4 |     int u, v, w, nxt;
5 | } edg[M];
6 | int tote, head[N];
7 | void init(){
8 |     tote = 0;
9 |     memset(head, -1, sizeof(head));
10 | }
11 | inline void addedg(int u, int v){
12 |     edg[tote].u = u; edg[tote].v = v; edg[tote].nxt = head[u]; head[u] = tote++;
13 | };
14 | inline void addedg(int u, int v, int w){
15 |     edg[tote].u = u; edg[tote].v = v; edg[tote].w = w; edg[tote].nxt = head[u]; head[u] = tote++;
16 | };

```

### 4.2 并差集

```

1 | struct DisjointSet{
2 |     int fa[N];
3 |     int tot;
4 |     void init(int n){
5 |         fr(i, 0, n){
6 |             fa[i] = i;
7 |         }
8 |         tot = 0;
9 |     }
10 |     int find(int x){
11 |         return x == fa[x] ? x : fa[x] = find(fa[x]);
12 |     };
13 |     void un(int x, int y){
14 |         int fx = find(x);
15 |         int fy = find(y);
16 |         if(fx != fy){
17 |             fa[fy] = fx;
18 |             tot--;
19 |         }
20 |     }
21 | }DS;

```

### 4.3 spfa

```

1 | bool spfa(int s){
2 |     for(i = 1; i <= n; ++i) d[i] = INF;
3 |     d[s] = 0;
4 |     q.push(s);
5 |     while(不为空q){

```

```

6         u = q.front();
7         q.pop();
8         for all edge(u, v, e)
9             if(d[v] > d[u] + e){
10                 d[v] = d[u] + e;
11                 if(不在中vq) { //这里用vst
12                     q.push(v);
13                     if(入队次数v==n) return false;
14                 }
15             }退出队列
16
17     }
18     return true;
19 }

```

#### 4.4 LCA

```

1  const int MAXM = 16;
2  const int MAXN = 1 << MAXM;
3  struct LCA {
4      vector<int> e[MAXN];
5      int d[MAXN], p[MAXN][MAXM];
6      void dfs_(int v, int f) {
7          p[v][0] = f;
8          for (int i = 1; i < MAXM; ++i) {
9              p[v][i] = p[p[v][i-1]][i-1];
10             }
11             for (int i = 0; i < (int)e[v].size(); ++i) {
12                 int w = e[v][i];
13                 if (w != f) {
14                     d[w] = d[v] + 1;
15                     dfs_(w, v);
16                 }
17             }
18         }
19
20     void init(int n) { //vector<int> e[MAXN]
21         //copy(e, e + n, this->e);
22         d[0] = 0;
23         dfs_(0, 0);
24     }
25
26     int up_(int v, int m) {
27         for (int i = 0; i < MAXM; ++i) {
28             if (m & (1 << i)) {
29                 v = p[v][i];
30             }
31         }
32         return v;
33     }
34
35     int lca(int a, int b) {

```

```

36         if (d[a] > d[b]) {
37             swap(a, b);
38         }
39         b = up_(b, d[b] - d[a]);
40         if (a == b) {
41             return a;
42         } else {
43             for (int i = MAXM - 1; i >= 0; --i) {
44                 if (p[a][i] != p[b][i]) {
45                     a = p[a][i];
46                     b = p[b][i];
47                 }
48             }
49             return p[a][0];
50         }
51     }
52     void add(int u, int v){
53         e[u].push_back(v);
54     }
55 } lca;

```

## 4.5 Dinic

```

1  const int pN=2000,eN=3000000;
2  struct Edge{
3      int u,v,nxt;
4      int w;
5  }e[eN];
6
7  int en,head[pN];
8
9  void init(){
10     memset(head,-1,sizeof(head));
11     en=0;
12 }
13 void add(int u,int v,int w){
14     e[en].u=u;e[en].v=v;e[en].w=w;e[en].nxt=head[u];head[u]=en++;
15     e[en].u=v;e[en].v=u;e[en].w=0;e[en].nxt=head[v];head[v]=en++;
16 }
17
18 int cur[pN],sta[pN],dep[pN];
19 int max_flow(int n,int s,int t){
20     int tr,flow = 0;
21     int i,u,v,f,r,top; //即是ffront 队列的头部
22     int j;
23     while(1){
24         memset(dep,-1,n*sizeof(int));
25         for( f = dep[ sta[0] = s ] = 0 ,r = 1;f != r;){
26             for( u = sta[f++],i = head[u];i != -1;i = e[i].nxt){
27                 if (e[i].w && dep[ v = e[i].v ] == -1){
28                     dep[v] = dep[u] + 1;
29                     sta[r++] = v; //将入队列v 向后标号法

```

```

30         if (v == t){
31             f = r;
32             break;
33         }
34     }
35 }
36 }
37 if (-1 == dep[t]) break;
38 memcpy(cur, head, n * sizeof(int));
39 for (i = s, top = 0; ;){
40     if (i == t){
41         for( j = 0, tr = inf; j < top; ++j){ //找出一条增广路
的最小边权
42             if (e[ sta[j] ].w < tr){
43                 tr = e[ sta[ f = j ] ].w; //一个简单优化每一
次不用从头开始找增广
44             }
45         }
46         for( j = 0; j < top; ++j){
47             e[ sta[j] ].w -= tr;
48             e[ sta[j]^1 ].w += tr;
49         }
50         flow += tr;
51         i = e[ sta[top = f] ].u;
52     }
53     for(j = cur[i]; cur[i] != -1; j = cur[i] = e[cur[i]].nxt)
//为当前的栈顶元素i
54         if (e[j].w && dep[i] + 1 == dep[e[j].v]) break; //找
到了一条路径最短的增广边
55
56         if (cur[i] != -1){ //就是这个点还有出度
57             sta[ top++ ] = cur[i];
58             i = e[ cur[i] ].v;
59         }
60         else{
61             if (top == 0) break;
62             dep[i] = -1;
63             i = e[ sta[--top] ].u;
64         }
65     }
66 }
67 return flow;
68 }

```

#### 4.6 sap

这里是与 dinic 不同的地方不用每次的 bfs 而是充分利用以前的距离标号的信息有这个定理：从源点到汇点的最短路一定是用允许弧构成。所以每次扩展路径都找允许弧，如果 i 没有允许弧就更新  $dis[i] = \min dis[j] + 1$  或者  $r[i][j]$  大于 0)；

```

1 #define inf 1000000000
2 using namespace std;
3

```



```

4  const int pN=5000,eN=100000;
5
6  struct Edge{
7      int u,v,nxt;
8      int w;
9  }e[eN];
10
11 int en,head[pN];
12
13 void init(){
14     memset(head,-1,sizeof(head));
15     en=0;
16 }
17 void add(int u,int v,int w){
18     e[en].u=u;e[en].v=v;e[en].w=w;e[en].nxt=head[u];head[u]=en++;
19     e[en].u=v;e[en].v=u;e[en].w=0;e[en].nxt=head[v];head[v]=en++;
20 }
21 int dep[pN],gap[pN],que[pN]; //gap 每一次重标号时若出现了断层，则可以证明
    无可行流，此时可以直接退出算法 st
22 void BFS(int n,int s,int t){
23     memset(dep,-1,n * sizeof(int));
24     memset(gap, 0 ,n * sizeof (int));
25     gap[0] = 1;
26     int f = 0,r = 0,u,v;
27     dep[ t ] = 0; que[r++] = t; //从后外前面标号
28     while(f != r){
29         u = que[f++];
30         if ( f == pN) f = 0;
31         for(int i = head[u]; i != -1;i = e[i].nxt){
32             v = e[i].v;
33             if (e[i].w != 0 || dep[v] != -1) continue; //如果容量为0 就
                根本到不到它
34             que[ r++ ] = v;
35             if (r == pN) r = 0;
36             dep[ v ] = dep[ u ] + 1;
37             ++ gap[dep[ v ]]; //这里的就是每一层有多少个点gap
38         }
39     }
40 }
41
42 int cur[pN],sta[pN];
43 int sap(int n,int s,int t){ //为总的点个数n 包括源点和汇点
44     int flow = 0;
45     BFS(n,s,t);
46     int top = 0,u = s,i;
47     memcpy(cur,head,n*sizeof(int)); //当前弧
48     while( dep[s] < n){
49         if ( u == t){
50             int tmp = inf;
51             int pos;
52             for(i = 0;i < top;i++){

```

```

53         if (tmp > e[ sta[i] ].w){
54             tmp = e[ sta[i] ].w;
55             pos = i;
56         }
57     }
58     for(i = 0; i < top; ++i){
59         e[ sta[i] ].w -= tmp;
60         e[ sta[i]^1 ].w += tmp;
61     }
62     flow += tmp;
63     top = pos;
64     u = e[ sta[top] ].u;
65 }
66 if(u != t && gap[dep[u] - 1] == 0) break; //gap 优化出现断层
    后直接退出
67 for(i = cur[u] ; i != -1 ; i = e[i].nxt) //当前弧优化 因
    为以前的弧绝对不满足要求
68 if(e[i].w != 0 && dep[u] == dep[e[i].v] + 1) break; //找
    到了一条最短增广路
69 if (i != -1) cur[ u ] = i, sta[top++] = i, u = e[i].v;
70 else{
71     //这里与不同dinic
72     int mn = n;
73     for (i = head[u] ; i != -1; i = e[i].nxt){
74         if ( e[i].w != 0 && mn > dep[ e[i].v ] ){
75             mn = dep[ e[i].v ] ;
76             cur[u] = i;
77         }
78     }
79     -- gap[ dep[u] ];
80     dep[u] = mn + 1;
81     ++ gap[ dep[u] ];
82     if (u != s) u = e[ sta[--top] ].u;
83 }
84 }
85 return flow;
86 }

```

## 4.7 费用流

```

1  const int inf = 0xfffff;
2  #define M 200001
3  #define maxx 2000
4  class Mcmf{
5  public:
6      struct T{
7          int u, v, w;
8          int nxt, cost;
9      }edge[M];
10     int en;
11     int visit[M], pre[M], dist[M], que[M], vis[M], pos[M];

```

```

12 void init(){
13     memset(vis,-1,sizeof(vis));
14     en=0;
15 }
16 void add(int u, int v, int w, int cost)
17 {
18     edge[en].u = u, edge[en].v = v, edge[en].w = w, edge[en].cost =
19     edge[en].nxt = vis[u], vis[u] = en++;
20     edge[en].u = v, edge[en].v = u, edge[en].w = 0, edge[en].cost =
21     edge[en].nxt = vis[v], vis[v] = en++;
22 }
23 bool spfa(int n, int s, int t){
24     int v, k;
25     for (int i = 0; i <= n; i++){
26         pre[i] = -1, visit[i] = 0;
27     }
28     int f = 0, r = 0;
29     for (int i = 0; i <= n; ++i) dist[i] = -1;
30     que[r++] = s; pre[s] = s; dist[s] = 0; visit[s] = 1;
31     while(f != r){
32         int u = que[f++];
33         visit[u] = 0;
34         for (k = vis[u]; k != -1; k = edge[k].nxt){
35             v = edge[k].v;
36             if (edge[k].w && dist[u] + edge[k].cost > dist[v]){
37                 dist[v] = dist[u] + edge[k].cost;
38                 pre[v] = u;
39                 pos[v] = k;    //是哪一条边到大的v 巧妙呀值得学习一
下 ~~~
40                 if (! visit[v]){
41                     visit[v] = 1;
42                     que[r++] = v;
43                 }
44             }
45         }
46     }
47     if (pre[t] != -1 && dist[t] > -1) return 1;
48     return 0;
49 }
50 int mnCostFlow(int n, int s, int t){
51     if (s == t){}
52     int flow = 0, cost = 0;
53     while(spfa(n, s, t)){
54         int u, mn = inf;
55         for (u = t; u != s; u = pre[u])
56             if (mn > edge[pos[u]].w) mn = edge[pos[u]].w;
57         flow += mn;
58         cost += dist[t] * mn;
59         for (u = t; u != s; u = pre[u]){
60             edge[pos[u]].w -= mn;
61             edge[pos[u]^1].w += mn;

```

```
62         }
63     }
64     return cost;
65 }
66 }mcf;
```

## 5 计算几何

### 5.1 动态凸包

```

1  const double eps = 1e-9;
2  typedef pair<int,int> pii;
3
4  struct dynamic_Convex{
5      map<int,int> cvex[2]; //cvex[0] upper contex line , cvex[1] lower co
6      map<int,int>::iterator p,q,it;
7      double cross(pii a,pii b,pii c){
8          return (double(b.first - a.first)) * (double(c.second - a.second)
9              - (double(b.second - a.second))*(double(c.first - a.first)))
10     }
11     bool IsUnderUpper(map<int,int> &st,int x,int y){ //check if the po
12         if( !st.size()) return false;
13         if (x < st.begin()->first || x > (--st.end()->first ) return f
14         if (st.find(x) != st.end()) return y <= st[x];
15         p = st.upper_bound(x);
16         q = p;q--;
17         return !( cross (make_pair(x,y) , *q,*p) > eps );
18     }
19     void insUpperConvex(map<int,int> &st, int x,int y){ //insert a poin
20         if( IsUnderUpper(st,x,y) ) return ;
21         st[x] = y;
22         p = st.upper_bound(x);
23         it = p;it--;
24         if ( p!=st.end()){
25             q = p;q++;
26             while(q != st.end() && cross (make_pair(x,y) , *p, *q) >=eps
27                 st.erase(p);p = q;q++;
28             }
29         }
30         if ( it != st.begin() ){
31             p = it;p--;q = p ;q--;
32             while(p != st.begin() && cross (make_pair(x,y),*q,*p) > -eps
33                 st.erase(p);p = q;q--;
34             }
35         }
36     }
37     bool judge(int x,int y){ //check if the poing is in the convex hull
38         return IsUnderUpper(cvex[0],x,y) && IsUnderUpper(cvex[1],x,-y);
39     }
40     void ins(int x,int y){ //insert a point to convex hull;
41         insUpperConvex(cvex[0],x,y);
42         insUpperConvex(cvex[1],x,-y);
43     }
44 }dc;

```