ACM Template

Zengarden

Last build at June 21, 2013

Contents

1 数据结构

1.1 树状数组

注意 init 的时候要小一个 1

```
template<int MAXN=300000, typename T = int>
struct BIT {
    int n;
    T a[MAXN];

    void init(int n) {
        this->n = n;
        fill(a, a + n + 1, T());
    }
    void add(int i, T v) {
        for (int j = i; j <= n; j = (j | (j - 1)) + 1) {
            a[j] += v;
        }
    }
    //(0..i];
    T sum(int i) const {
        T ret = T();
        for (int j = i; j > 0; j = j & (j - 1)) {
            ret += a[j];
        }
        return ret;
    }
    T get(int i) const {
        return sum(i ) - sum(i-1);
    }
    void set(int i, T v) {
        add(i, v - get(i));
    }

    void add(int l , int r ,T v)  //need sum is ith val;get && set can'
    {
        add(l,v);add(r+1,-v);
    }
};
```

1.2 坐标离散

注意下标～

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

```
void Lisan(int N)
{
    for(int i=0; i<N; i++)  r[i] = i;
    sort(r, r+N, cmp);
   mp[1] = axis[r[0]];
    axis[r[0]] = M = 1;
    for(int i=1; i<N; i++)
    {
        if(axis[r[i]] == mp[M])  axis[r[i]] = M;
        else     mp[++M] = axis[r[i]], axis[r[i]] = M;
    }
}
int main(){
    for (int i=0;i<5;i++) scanf("%d",&axis[i]);
    Lisan(5);
    return 0;
}
```

1.3 lca-rmq

```
using namespace std;
typedef long long ll;
const int N=10010;
int n;
struct E{
    int u,v,nxt,w;
}edg[2000010];

int tote,head[N];
void init(){
    tote=0;
    memset(head,-1,sizeof(head));
}
inline void addedg(int u,int v){
    edg[tote].u=u;edg[tote].v=v;edg[tote].nxt=head[u];head[u]=tote++;
};

int vst[N],e[N<<1],r[N],d[N<<1];
int cnt;
int fa[N];
void dfs(int u, int depth)  {
    vst[u] = true;
    e[cnt] = u;
    d[cnt] = depth;
    r[u] = cnt++;
    for(int i=head[u];i!=-1;i=edg[i].nxt){
        int v=edg[i].v;
        if (!vst[v]){
            dfs(v,depth+1);
            e[cnt]=u;
            d[cnt++]=depth;
```

```
            }
        }
}
inline int _min(int i, int j)  {
    if (d[i] < d[j]) return i;
    return j;
}
int dp[2*N][16];
void rmpinit(){
    int nn = 2 * n - 1;
    for (int i = 0; i < nn; ++i)   //下标是从开始的0
        dp[i][0] = i;
    int k = (int)(log(nn * 1.0) / log(2.0));
    for (int j = 1; j <= k; ++j)  {
        for (int i = 0; i + (1 << j) - 1 < nn; ++i)
            dp[i][j] = _min(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
    }
}
inline int query(int l, int r)  {
    int k = (int)(log(r * 1.0 - l + 1) / log(2.0));
    return _min(dp[l][k], dp[r-(1<<k)+1][k]);
}
int main(){
    //fi;
    int t;
    scanf("%d",&t);
    int u,v;
    while(t--){
        scanf("%d",&n);
        init();
        fr(i,0,n+1) fa[i]=i;
        fr(i,0,n-1){
            scanf("%d%d",&u,&v);
            addedg(u,v);//addedg(v,u);
            fa[v]=u;
        }
        int root;
        fr(i,1,n+1) if (fa[i]==i) {root=i;break;}
        cnt=0;cl(vst);
        dfs(root,0);
        rmpinit();

        scanf("%d%d",&u,&v);
        if (r[u]<=r[v])  printf("%d\n", e[query(r[u], r[v])]);
        else printf("%d\n", e[query(r[v], r[u])]);
    }
    return 0;
}
```

1.4 rmq2d

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

```
const int N = 301;
int t,n;
int dp[N][N][9][9];
void in(int &a)
{
    char c,f;
    while(((f=getchar())<'0'||f>'9')&&f!='-');
    c=(f=='-')?getchar():f;
    for(a=0;c>='0'&&c<='9';c=getchar())a=a*10+c-'0';
    if(f=='-')a=-a;
}
void initrmq(){
    int i,j;
    int m = log(double(n)) / log(2.0);
    fr(i,0,m+1){
        fr(j,0,m+1){
            if (i==0 && j==0) continue;
            for(int r = 0; r+(1<<i)-1 < n; ++r){
                for(int c = 0; c+(1<<j)-1 < n; ++c){
                    if(i == 0) dp[r][c][i][j] = min(dp[r][c][i][j-1] ,
                    else dp[r][c][i][j] = min(dp[r][c][i-1][j] , dp[r+(
                }
            }
        }
    }
}
int rmq_2d_query(int X1,int Y1,int X2,int Y2){
    int x = log(double(X2 - X1 +1)) / log(2.0);
    int y = log(double(Y2 - Y1 +1)) / log(2.0);
    int m1 = dp[X1][Y1][x][y];
    int m2 = dp[X2-(1<<x)+1][Y1][x][y];
    int m3 = dp[X1][Y2-(1<<y)+1][x][y];
    int m4 = dp[X2-(1<<x)+1][Y2-(1<<y)+1][x][y];
    return min(min(m1,m2),min(m3,m4));
}

void inp(){
    int i,j,m,X1,Y1,X2,Y2;
    in(n);
    fr(i , 0,n){
        fr(j,0,n){
            in(dp[i][j][0][0]);
        }
    }
    initrmq();
    sfint(m);
    while(m--){
        in(X1);in(Y1);in(X2);in(Y2);
        printf("%d\n",rmq_2d_query(X1-1,Y1-1,X2-1,Y2-1));
    }
}
```

```
int main(){
    fi;
    sfint(t);
    while(t--){
        inp();
    }
    return 0;
}
```

1.5 划分树

```
int a[100001];
int b[21][100001];
int sum[21][100001];    //sum[i表示]l——这些点中有多少个进入了左子树。i
int n,m;

void build(int l,int r,int d){      //代表在树上第几层d
    if (l==r) return;
    int i,mid=(l+r)>>1,id1=l,id2=mid+1,midsum=0;
    for (i=mid;i>=l&&a[i]==a[mid];i--)midsum+=1;
    for (i=l;i<=r;i++){
        sum[d][i]=i==l?0:sum[d][i-1];
        if (b[d][i]<a[mid]){
            b[d+1][id1++]=b[d][i];
            sum[d][i]+=1;
        }
        else if(b[d][i]==a[mid]&&midsum){
            midsum-=1;
            b[d+1][id1++]=b[d][i];
            sum[d][i]+=1;
        }
        else b[d+1][id2++]=b[d][i];
    }
    build(l,mid,d+1);
    build(mid+1,r,d+1);
}

int search(int x,int y,int k){
    int l=1,r=n,d=0;
    int ls,rs,mid;
    while (x!=y){
        ls=x==l?0:sum[d][x-1];      //因为要包含x
        rs=sum[d][y];
        mid=(l+r)>>1;
        if (k<=rs-ls) {             //在左子树上
            x=l+ls;
            y=l+rs-1;
            r=mid;
        }
        else                        //在右子树上
        {
```

```
            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。)
            y=mid+1+y-l-rs;
            k-=rs-ls;
            l=mid+1;
        }
        d+=1;
    }
    return b[d][x];
}

int main(){
    freopen("in.txt","r",stdin);
    int cnt=1;
    while(scanf("%d",&n)!=EOF){
        int i,x,y,t;
        for (i=1;i<=n;i++){
            scanf("%d",&t);
            a[i]=b[0][i]=t;
        }
        sort(a+1,a+n+1);
        build(1,n,0);
        scanf("%d",&m);
        printf("Case %d:\n",cnt++);
        while(m--){
            scanf("%d%d",&x,&y);
            int k=(y-x+1)/2+1;
            printf("%d\n",search(x,y,k));
        }
    }
    return 0;
}
```

1.6 扫描线矩形面积并

```
const int N = 400000;
int n;
struct ARR {
    int a[N];
    int tot;
    void init(){tot = 0;}
    void add(int x){
        a[tot++] = x;
    }
    void uni() {
        sort(a,a+tot);
        tot = unique(a,a+tot)-a;
    }
    int fd(int x){
        return lower_bound(a,a+tot,x)-a;
    }
}A;
```

```
struct Line{
    int s,e,y,f;
    bool operator < (const Line & l) const {
        if( y == l.y) return s < l.s;
        return y < l.y;
    }
}l[N];
int tot;
void add_line(int s,int e,int y,int f){
    if (s == e) return ;
    A.add(s);A.add(e);
    l[tot].s = s;l[tot].e = e;l[tot].y = y;l[tot++].f = f;
}
void init(){
    tot = 0;A.init();
    sfint(n);
    int x,y,h;
    fr(i , 0 ,n){
        sfint3(x,y,h);
        add_line(x,y,0,1);
        add_line(x,y,h,-1);
    }
    /*int x1, y1, x2, y2, x3, y3, x4, y4;
    fr(i , 0 ,n){
        scanf("%d%d%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3,&x4,&y4);
        add_line(x1,x3,y1,1); add_line(x1,x3,y2,-1);
        add_line(x3,x4,y1,1); add_line(x3,x4,y3,-1);
        add_line(x3,x4,y4,1); add_line(x3,x4,y2,-1);
        add_line(x4,x2,y1,1); add_line(x4,x2,y2,-1);
    }*/
    A.uni();
}

struct SEGT{
    struct SEGtr
    {
        int l,r,cov;
        ll len;
    }tr[N*4];
    void build(int rt,int l,int r){
        tr[rt].l = l;tr[rt].r = r;tr[rt].cov = 0;tr[rt].len = 0;
        if(l == r){
            return;
        }
        int mid = (l+r)>>1;
        build(rt<<1,l,mid);
        build(rt<<1|1,mid+1,r);
    }
    void up(int rt){
        if(tr[rt].cov != 0) tr[rt].len = A.a[tr[rt].r+1]-A.a[tr[rt].l];
        else if( tr[rt].l == tr[rt].r) tr[rt].len = 0;
```

```
            else {
                tr[rt].len=tr[rt<<1].len+tr[rt<<1|1].len;
            }
        }
        void update(int rt,int l,int r,int add){
            if(tr[rt].l >= l && tr[rt].r <= r) {
                tr[rt].cov += add;
                up(rt);
                return ;
            }
            int mid = (tr[rt].l + tr[rt].r)>>1;
            if(r <= mid)
                update(rt<<1,l,r,add);
            else if(l >mid)
                update(rt<<1|1,l,r,add);
            else{
                update(rt<<1,l,mid,add);
                update(rt<<1|1,mid+1,r,add);
            }
            up(rt);
        }
}S;
void sol(){
    sort(l,l+tot);
    S.build(1,0,A.tot-2);
    S.update(1,A.fd(l[0].s),A.fd(l[0].e)-1,l[0].f);
    ll ans = 0;
    fr(i , 1 ,tot){
        ans += (ll(l[i].y - l[i-1].y))*ll(S.tr[1].len);
        S.update(1, A.fd(l[i].s),A.fd(l[i].e)-1,l[i].f);
    }
    printf("%lld\n",ans);
}
```

2 数学

2.1 素数

2.1.1 筛素数

```
bool flag[N+1];
int prime[N+1];
int totpri;
void getpri(){
    int n=N;
    int i,j;totpri=0;
    for(i=2;i<=n;++i) { /*筛选素数快速的方法*/
        if(!flag[i]) prime[totpri++]=i;
        for(j=0;j<totpri&&i*prime[j]<=n;++j)
        {
            flag[i*prime[j]]=1;
            if(i%prime[j]==0) break;
        }
    }
}
```

2.1.2 Miller-Rabbin

```
bool primeTest(ll n, ll b) {
    ll m = n - 1;
    ll counter = 0;
    while ((m & 1) == 0) {
        m >>= 1;
        counter ++;
    }
    ll ret = pow_mod(b, m, n);
    if (ret == 1 || ret == n - 1) {
        return true;
    }
    counter --;
    while (counter >= 0) {
        ret = add_mod(ret, ret, n);
        if (ret == n - 1) {
            return true;
        }
        counter --;
    }
    return false;
}

const int BASIC[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

bool isPrime(ll n) {
    if (n < 2) {
        return false;
    }
```

```
    if (n < 4) {
        return true;
    }
    if (n == 3215031751LL) {
        return false;
    }
    for (int i = 0; i < 12 && BASIC[i] < n; ++ i) {
        if (!primeTest(n, BASIC[i])) {
            return false;
        }
    }
    return true;
}
```

2.2 Gcd

```
int gcd(int a,int b)
{
    if(b==0) return a;
    return gcd(b,a%b);
}
```

2.3 extend-gcd

2.3.1 求模线性方程

```
int e_gcd(int a,int b,int &x,int &y)
{
    if(b==0)
    {x=1;y=0;return a;}
    int ans=e_gcd(b,a%b,x,y);
    int temp=x;
    x=y;
    y=temp-(a/b)*y;
    return ans;
}
```

2.3.2 求逆

```
// a * x + b * y = gcd(a, b)
long long extGcd(long long a, long long b, long long& x, long long& y)
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    } else {
        int g = extGcd(b, a % b, y, x);
        y -= a / b * x;
        return g;
    }
}

// ASSUME: gcd(a, m) == 1
```

```
long long modInv(long long a, long long m) {
    long long x, y;
    extGcd(a, m, x, y);
    return (x % m + m) % m;
}
```

2.4 快速加

```
ll add_mod(ll a,ll b,ll m){
    ll ans=0;
    a%=m;
    while(b){
        if(b&1) ans=(ans+a)%m;
        a=(a+a)%m;
        b>>=1;
    }
    return ans%m;
}
```

2.5 快速幂

```
ll pow_mod(ll a,ll b,ll m) {
    ll ans=1;
    a%=m;
    while(b)
    {
        if(b&1) ans=(ans*a)%m;
        a=(a*a)%m;
        b>>=1;
    }
    return ans;
}
```

2.6 矩阵类

```
ll MOD=10000;
template<int MAXN=1010,int MAXM=1010, typename T = int>
struct Mat{
    int n,m;
    T a[MAXN][MAXM];
    Mat(int _n=0,int _m=0):n(_n),m(_m){}
    void clear(){
        memset(a,0,sizeof(a));
    }
    void identity(){
        memset(a,0,sizeof(a));
        fr(i , 0 ,n){
            a[i][i] = 1;
        }
    }
    Mat operator + (const Mat &b) const{
        Mat tmp(n,m);
        for(int i = 0;i<n;++i){
```

```
                for(int j = 0;j<m;++j){
                    tmp.a[i][j] = a[i][j] + b.a[i][j];
                }
            }
            return tmp;
    }
    Mat operator - (const Mat &b) const{
            Mat tmp(n,m);
            for(int i = 0;i<n;++i){
                for(int j = 0;j<m;++j){
                    tmp.a[i][j] = a[i][j] - b.a[i][j];
                }
            }
            return tmp;
    }
    Mat operator * (const Mat &b) const{
            Mat tmp(n,m);
            tmp.clear();
            for(int i = 0;i<n;++i){
                for(int j = 0;j<n;++j)
                    for(int k = 0;k < n;++k){
                    tmp.a[i][j] = (tmp.a[i][j] + a[i][k] * b.a[k][j]) % MOD
                }
            }
            return tmp;
    }
    Mat operator ^ (int b) {
            Mat ret(n,m);
            ret.identity();
            while(b){
                if(b&1)
                    ret = (*this)*ret;
                (*this) = (*this)*(*this);
                b>>=1;
            }
            return ret;
    }
    void disp(){
            fr(i , 0 ,n){
                fr(j , 0 ,m){
                    printf("%d ",a[i][j]);
                }
                puts("");
            }
    }
};
```

2.7 容斥

```
ll lim;
ll dfs(int pos,ll d){
    ll ret = 0;
```

```
    while(pos<totpri&&prime[pos] <= k&& prime[pos] * d<=lim){
        ret += lim/(prime[pos]*d) - dfs(pos+1,d*prime[pos]);
        pos++;
    }
    return ret;
}
```

2.8 组合数

2.8.1 暴力求解

```
C(n,m)=n*(n-1)*...*(n-m+1)/m, !n<=15
int Combination(int n, int m)
{
    const int M = 10007;
    int ans = 1;
    for(int i=n; i>=(n-m+1); --i)
        ans *= i;
    while(m)
        ans /= m--;
    return ans % M;
}
```

2.8.2 打表

```
C(n,m)=C(n-1,m-1)+C(n-1,m, )n<=10,000
const int M = 10007;
const int N = 1000;
ll C[N][N];
void initc(){
    int i,j;
    for(i=0; i<N; ++i){
        C[0][i] = 0;
        C[i][0] = 1;
    }
    for(i=1; i<N; ++i){
        for(j=1; j<N; ++j)
            C[i][j] = (C[i-1][j] + C[i-1][j-1]) % MOD;
    }
}
```

2.8.3 质因数分解

C(n,m)=n!/(m!*(n-m)!)，C(n,m)=p1a1-b1-c1p2a2-b2-c2pkak-bk-ck,n<=10,000,000

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

```
                prim.push_back(i);
                for(j=i*i; j<=MAXN; j+=i)
                arr[j] = true;
            }
        }
        while(i<=MAXN){
            if(!arr[i])
            prim.push_back(i);
            i+=2;
        }
        return prim;
}

//计算n中素因子!的指数p
int Cal(int x, int p){
        int ans = 0;
        long long rec = p;
        while(x>=rec){
            ans += x/rec;
            rec *= p;
        }
        return ans;
}

//计算的次方对取模，二分法nkM
int Pow(long long n, int k, int M){
        long long ans = 1;
        while(k){
            if(k&1){
                ans = (ans * n) % M;
            }
            n = (n * n) % M;
            k >>= 1;
        }
        return ans;
}

//计算C(n,m)
int Combination(int n, int m){
        const int M = 10007;
        vector<int> prim = produce_prim_number();
        long long ans = 1;
        int num;
        for(int i=0; i<prim.size() && prim[i]<=n; ++i){
            num = Cal(n, prim[i]) - Cal(m, prim[i]) - Cal(n-m, prim[i]);
            ans = (ans * Pow(prim[i], num, M)) % M;
        }
        return ans;
}
```

2.8.4 Lucas

/* 定理，将 m,n 化为 p 进制, 有:C(n,m)=C(n0,m0)*C(n1,m1)...(mod 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。*/

```cpp


const int M = 10007;
int ff[M+5];    //打表，记录n，避免重复计算!

//求最大公因数
int gcd(int a,int b){
    if(b==0)
        return a;
    else
        return gcd(b,a%b);
}

//解线性同余方程，扩展欧几里德定理
int x,y;
void Extended_gcd(int a,int b){
    if(b==0)    {
       x=1;
       y=0;
    }
    else{
       Extended_gcd(b,a%b);
       long t=x;
       x=y;
       y=t-(a/b)*y;
    }
}

//计算不大的C(n,m)
int C(int a,int b){
    if(b>a)
    return 0;
    b=(ff[a-b]*ff[b])%M;
    a=ff[a];
    int c=gcd(a,b);
    a/=c;
    b/=c;
    Extended_gcd(b,M);
    x=(x+M)%M;
    x=(x*a)%M;
    return x;
}

//定理Lucas
int Combination(int n, int m){
```

```
    int ans=1;
    int a,b;
    while(m||n){
                a=n%M;
        b=m%M;
        n/=M;
        m/=M;
        ans=(ans*C(a,b))%M;
    }
    return ans;
}

int main(void){
    int i,m,n;
    ff[0]=1;
    for(i=1;i<=M;i++)   //预计算n!
    ff[i]=(ff[i-1]*i)%M;

    scanf("%d%d",&n, &m);
    printf("%d\n",func(n,m));

    return 0;
}
```

2.9 pollardRho

```
vector <ll> divisors;
ll pollardRho(ll n, ll seed) {
    ll x, y;
    x = y = rand() % (n - 1) + 1;
    ll head = 1;
    ll tail = 2;
    while (true) {
        x = pow_mod(x,2, n);
        x = add_mod(x, seed, n);
        if (x == y) {
            return n;
        }
        ll d = gcd(abs(x - y), n);
        if (1 < d && d < n) {
            return d;
        }
        head ++;
        if (head == tail) {
            y = x;
            tail <<= 1;
        }
    }
}
void factorize(ll n) {
    if (n > 1) {
```

```
            if (isPrime(n)) {
                divisors.push_back(n);
            } else {
                ll d = n;
                while (d >= n) {
                    d = pollardRho(n, rand() % (n - 1) + 1);
                }
                factorize(n / d);
                factorize(d);
            }
        }
}
```

2.10 欧拉函数

2.10.1 一般的求法

```
const int N = 100010;
bool is_prime[N];
ll phi[N];
ll prime[N];
void init(){
    ll i, j, k = 0;
    phi[1] = 1;
    for(i = 2; i < N; i++){
        if(is_prime[i] == false){
            prime[k++] = i;
            phi[i] = i-1;
        }
        for(j = 0; j<k && i*prime[j]<N; j++){
            is_prime[ i*prime[j] ] = true;
            if(i%prime[j] == 0){
                phi[ i*prime[j] ] = phi[i] * prime[j];
                break;
            }
            else phi[ i*prime[j] ] = phi[i] * (prime[j]-1);
        }
    }
}
```

2.10.2 递推

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

2.10.3 单独求

```
ll Euler_Phi(ll n)
{
    ll t = n,p = n;
```

```
    ll sq = sqrt (n);

    for (int i=0;prime[i]<=sq && i<totpri;i++)
    {
        if (t%prime[i]==0)
        {
            p = p/prime[i]*(prime[i]-1);

            while (t%prime[i]==0)
                t/=prime[i];

            //sq = sqrt(t);
        }

        if (t == 1)
            break;
    }

    if (t > 1)
        p = p/t*(t-1);

    return p;
}
```

2.11 高斯消元

2.11.1 模二消元

```
int gauss(int n){
    int r,c;
    for(r = 0, c=0;r<n,c<n;++r,++c){
        int p = r;
        fr(i , r+1,n){
            if(a[i][c] > a[p][c]) p = i;
        }
        if (p != r){
            fr(i , c,n+1){
                swap(a[p][i],a[r][i]);
            }
        }
        if(a[r][c] == 0){
            r--;continue;
        }
        fr(i , 0,n){
            if (a[i][c] == 0||i == r) continue;
            fr(j,c,n+1) a[i][j] = a[i][j]^a[r][j];
        }
    }
    fr(i , r,n) if (a[i][n]) return -1;
    return n-r;
}
```

2.11.2 浮点

```
const double eps = 1e-12;
const int MAXN = 30;
inline int gauss(double a[][4], bool l[], double ans[], const int &n){
    int res = 0, r = 0;
    for(int i = 0; i < n; ++i) l[i] = false;
    for(int i = 0; i < n; ++i) {
        for(int j = r; j < n; ++j){
            if( fabs(a[j][i]) > eps) {
                for(int k = i; k<= n; ++k) swap(a[j][k],  a[r][k]);
                break;
            }
        }
        if( fabs(a[r][i]) < eps){
            ++res;
            continue;
        }
        for(int j = 0; j< n; ++j){
            if( j != r && fabs(a[j][i])>eps){
                double tmp = a[j][i] / a[r][i];
                for(int k = i ; k<=n ; ++k){
                    a[j][k] -= tmp * a[r][k];
                }
            }
        }
        l[i] = true; ++r;
    }

    fr(i ,0 ,n){
        fr(j , 0,n+1){
            printf("%lf ",a[i][j]);
        }
        puts("");
    }
    for(int i = 0; i< n ;++i){  //有问题
        if (l[i])
            for(int j = 0; j< n; ++j){
                if (fabs(a[j][i]) > 0){
                    ans[i] = a[j][n] / a[j][i];
                }
            }
    }
    return res;
}
```

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

```
void gray(int n,int *code){
    int t=0,i;
    for (i=0;i<n;t+=code[i++]);
    if (t&1)
        for (n--;!code[n];n--);
            code[n-1]=1-code[n-1];
}
```

2.13 离散对数

```
#define MAXN 131071
struct HashNode {
    ll data, id, next;
};
HashNode hash[MAXN<<1];
bool flag[MAXN<<1];
ll top;

void Insert ( ll a, ll b )
{
    ll k = b & MAXN;
    if ( flag[k] == false )
    {
        flag[k] = true;
        hash[k].next = -1;
        hash[k].id = a;
        hash[k].data = b;
        return;
    }
    while( hash[k].next != -1 )
    {
        if( hash[k].data == b ) return;
        k = hash[k].next;
    }
    if ( hash[k].data == b ) return;
    hash[k].next = ++top;
    hash[top].next = -1;
    hash[top].id = a;
    hash[top].data = b;
}

ll Find ( ll b )
{
    ll k = b & MAXN;
    if( flag[k] == false ) return -1;
    while ( k != -1 )
    {
        if( hash[k].data == b ) return hash[k].id;
        k = hash[k].next;
    }
    return -1;
```

```
}

ll gcd ( ll a, ll b )
{
    return b ? gcd ( b, a % b ) : a;
}

ll ext_gcd (ll a, ll b, ll& x, ll& y )
{
    ll t, ret;
    if ( b == 0 )
    {
        x = 1, y = 0;
        return a;
    }
    ret = ext_gcd ( b, a % b, x, y );
    t = x, x = y, y = t - a / b * y;
    return ret;
}
ll mod_exp ( ll a, ll b, ll n )
{
    ll ret = 1;
    a = a % n;
    while ( b >= 1 )
    {
        if( b & 1 )
            ret = ret * a % n;
        a = a * a % n;
        b >>= 1;
    }
    return ret;
}

ll BabyStep_GiantStep ( ll A, ll B, ll C ) //A^X %C == B
{
    memset(flag,0,sizeof(flag));
    top = MAXN;  B %= C;
    ll tmp = 1, i;
    for ( i = 0; i <= 100; tmp = tmp * A % C, i++ )
        if ( tmp == B % C ) return i;

    ll D = 1, cnt = 0;
    while( (tmp = gcd(A,C)) !=1 )
    {
        if( B % tmp ) return -1;
        C /= tmp;
        B /= tmp;
        D = D * A / tmp % C;
        cnt++;
    }

```

```
    ll M = (ll)ceil(sqrt(C+0.0));
    for ( tmp = 1, i = 0; i <= M; tmp = tmp * A % C, i++ )
        Insert ( i, tmp );

    ll x, y, K = mod_exp( A, M, C );
    for ( i = 0; i <= M; i++ )
    {
        ext_gcd ( D, C, x, y ); // D * X = 1 ( mod C )
        tmp = ((B * x) % C + C) % C;
        if( (y = Find(tmp)) != -1 )
            return i * M + y + cnt;
        D = D * K % C;
    }
    return -1;
}
```

2.14 字典序

2.14.1 排列

```
int perm2num(int n, int *p) {
    int i, j, ret = 0, k = 1;
    for (i = n - 2; i >= 0; k *= n - (i--))
        for (j = i + 1; j < n; j++)
            if (p[j] < p[i])
                ret += k;
    return ret;
}
void num2perm(int n, int *p, int t) {
    int i, j;
    for (i = n - 1; i >= 0; i--)
        p[i] = t % (n - i), t /= n - i;
    for (i = n - 1; i; i--)
        for (j = i - 1; j >= 0; j--)
            if (p[j] <= p[i])
                p[i]++;
}
```

2.14.2 组合

```
int comb(int n, int m) {
    int ret = 1, i;
    m = m < (n - m) ? m : (n - m);
    for (i = n - m + 1; i <= n; ret *= (i++));
    for (i = 1; i <= m; ret /= (i++));
    return m < 0 ? 0 : ret;
}
int comb2num(int n, int m, int *c) {
    int ret = comb(n, m), i;
    for (i = 0; i < m; i++)
        ret -= comb(n - c[i], m - i);
    return ret;
}
```

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

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

```
#define MAXN 1000
int polya(int* perm, int n, int& num) {
    int i, j, p, v[MAXN] = {0}, ret = 1;
    for (num = i = 0; i < n; i++)
        if (!v[i]) {
            for (num++, j = 0, p = i; !v[p = perm[p]]; j++)
                v[p] = 1;
            ret *= j / gcd(ret, j);
        }
    return ret;
}
```

3 string

3.1 kmp

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 mod[0..j-1] == mod[i - j+1 .. i]
next 的值就是每个 j nxt 往前
重要的是理解那个往前
例：ababa nxt 为：0 0 1 2 3
那么 nxt[nxt[5]] = nxt[3] = 2; 即 s[0..1] = s[2..3]

性质：Len-nxt[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

```
int len1,len2, nxt[10005];
char mod[10005], s[1000005];
void get_nxt(char mod[],int len){
    int i,j=0;
    nxt[0]=0;
    for(i=1;i<len;i++){
        while(j>0 && mod[j]!=mod[i]) j=nxt[j-1];
        if(mod[j]==mod[i]) j++;
        nxt[i]=j;
    }
}
int KMP(int len1,int len2,char s[],char mod[],int pos = 0)
{
    int i=pos,j=0,ret=0;
    while(i<len1){
        while(j && mod[j]!=s[i]) j=nxt[j-1];
        if(mod[j]==s[i++]){
            if((++j)==(len2)) ret++;
        }
    }
    return ret;
}
```

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

```
const int N=100010;

int len_s,len_t;
int nxt[N],extend[N];
char S[N],T[N];
void build_nxt()
{
    int k, q, p, a;
    nxt[0] = len_t;
```

```
    for (k = 1, q = -1; k < len_t; k ++, q --) {
        if (q < 0 || k + nxt[k - a] >= p) {
            if (q < 0)q = 0, p = k;
            while (p < len_t && T[p] == T[q]) {
                p ++, q ++;
            }
            nxt[k] = q, a = k;
        }
        else {
            nxt[k] = nxt[k - a];
        }
    }
}
void extend_KMP()
{
    int k, q, p, a;
    for (k = 0, q = -1; k < len_s; k ++, q --) {
        if (q < 0 || k + nxt[k - a] >= p) {
            if (q < 0)q = 0, p = k;
            while (p < len_s && q < len_t && S[p] == T[q]) {
                p ++, q ++;
            }
            extend[k] = q, a = k;
        }
        else {
            extend[k] = nxt[k - a];
        }
    }
}
int main(){
    fi;
    scanf("%s",S);
    scanf("%s",T);

    len_t=strlen(T);
    len_s=strlen(S);
    build_nxt();
    extend_KMP();

    return 0;
}
```

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 是这样一个字符串（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

```
#define M 20000050
char str1[M],str[2*M];//start from index 1
int rad[M],nn,n;
void Manacher(int *rad,char *str,int n)
{
    int i;
    int mx = 0;
    int id;
    for(i=1; i<n; i++)
    {
        if( mx > i ) rad[i] = rad[2*id-i]<mx-i?rad[2*id-i]:mx-i;
        else rad[i] = 1;
        for(; str[i+rad[i]] == str[i-rad[i]]; rad[i]++);
        if( rad[i] + i > mx )
        {
            mx = rad[i] + i;
            id = i;
        }
    }
}
struct PLD{
    int l,r;
    PLD(int x=0,int y = -1):l(x),r(y){}
}p[N];

void getlr(int n){
    fr(i,2,n){
        p[i].l = i-rad[i]+1;
        p[i].l = (p[i].l+1)/2-1;
        p[i].r = p[i].l+rad[i]-2;
    }
}


int main()
{
    int i,ans,Case=1;
    while(scanf("%s",str1)!=EOF)
    {
        nn=strlen(str1);
        n=2*nn+2;
        str[0]='$';
        for(i=0;i<=nn;i++)
        {
            str[2*i+1]='#';
            str[2*i+2]=str1[i];
        }
        Manacher(rad,str,n);
        ans=1;
        for(i=0;i<n;i++)
```

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

3.4 ac 自动机

```
char str[2000010];
char c[1010][55];

namespace AC {
const int dict = 26;
const int root = 0;
const int maxn = 3000000;
struct node {
    int son[dict], fail, idx;
} tree[maxn];
int apr[10010];
bool vis[3000000];
int sz;
int initNode(int idx) {
    memset(tree[idx].son, 0, sizeof(tree[idx]));
    tree[idx].fail = tree[idx].idx = 0;
    return idx;
}
void init() {
    sz = initNode(0);
    memset(apr, 0, sizeof(apr));
}
void ins(char *s, int idx) {
    int cur = root, t;
    while (*s) {
        t = *s - 'A';
        if (!tree[cur].son[t]) tree[cur].son[t] = initNode(++sz);
        cur = tree[cur].son[t];
        s++;
    }
    tree[cur].idx = idx;
}
queue<int> q;
void buildac() {
    while(!q.empty()) q.pop();
    int i, cur, nxt, f;
    for ( i = 0 ; i < dict ; i++ )
        if (tree[root].son[i])  q.push(tree[root].son[i]);

    while (!q.empty()) {
        cur = q.front();
        q.pop();
        f = tree[cur].fail;
```

```
        for ( i = 0 ; i < dict ; i++ )
            if (tree[cur].son[i]) {
                nxt = tree[cur].son[i];
                tree[nxt].fail = tree[f].son[i];
                q.push(nxt);
            } else tree[cur].son[i] = tree[f].son[i];
    }
}
void search(char *s) {
    int i, cur = 0;
    for ( ; *s ; s++ ) {
        if( (*s) >= 'A' && (*s) <= 'Z' ) {
            cur = tree[cur].son[*s - 'A'];
            for ( i = cur ; i ; i = tree[i].fail ) { //用于优化vis
                apr[tree[i].idx]++;
            }
        } else {
            cur = 0;
        }
    }
    for(int i = 1; i <= 1010; ++i) {
        if(apr[i]) {
            printf("%s: %d\n", c[i], apr[i]);
        }
    }
}
};

int main() {
 //  freopen("input.txt", "r", stdin);
    int n;

    while(scanf("%d", &n) != EOF) {
        AC::init();
        for(int i = 1; i <= n ; ++i) {
            scanf("%s", c[i]);
            AC::ins(c[i], i);
        }
        AC::buildac();
        getchar();
        gets(str);
        AC::search(str);
    }

    return 0;
}
```

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(Suffix(SA[i]),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)=minheight[k] | i+1 k 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

```
const int maxn = 2010;
int wa[maxn],wb[maxn],wv[maxn],wss[maxn],sa[maxn];
bool cmp(int *r,int a,int b,int l)
{
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m)
{
    int i,j,p,*x=wa,*y=wb,*t;

    for(i=0;i<m;i++) wss[i]=0;
    for(i=0;i<n;i++) wss[x[i]=r[i]]++;
    for(i=1;i<m;i++) wss[i]+=wss[i-1];
    for(i=n-1;i>=0;i--) sa[--wss[x[i]]]=i;
    for(j=1,p=1;p<n;j*=2,m=p)
    {
        for(p=0,i=n-j;i<n;i++) y[p++]=i;
        for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
        for(i=0;i<n;i++) wv[i]=x[y[i]];
        for(i=0;i<m;i++) wss[i]=0;
        for(i=0;i<n;i++) wss[wv[i]]++;
        for(i=1;i<m;i++) wss[i]+=wss[i-1];
        for(i=n-1;i>=0;i--) sa[--wss[wv[i]]]=y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
    }
}
int rk[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
    int i,j,k=0;
    cl(rk);
    for(i=1;i<=n;i++) rk[sa[i]]=i;
    for(i=0;i<n;height[rk[i++]]=k)
        for(k?k--:0,j=sa[rk[i]-1];r[i+k]==r[j+k];k++);
}
int dp[100010][18];
void rmqInit(int n){
    fr(i , 0 ,n) dp[i][0] = height[i+2];
    int k = (int)(log(n * 1.0) / log(2.0)); k++;
    fr(j , 1 , k){
        for(int i = 0;i+(1 << j)-1 < n;++i){
            dp[i][j] = min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
        }
```

```
    }
}
inline int query(int l ,int r){
    int k = (int)(log(r * 1.0 - l + 1) / log(2.0));
    return min(dp[l][k], dp[r-(1<<k)+1][k]);
}
int lcp(int l,int r){
    int t ;
    l = rk[l],r = rk[r];
    if(l>r) l^=r^=l^=r;
    return query(l-1,r-2);
}

bool check(int x,int k){
    fr(i , 0 ,k-1){
        if(lcp(i*x,(i+1)*x) < x) return 0;
    }
    return 1;
}
int c[maxn];
char str[maxn];
int maxrep[maxn];
int main(){
    fi;
    while(sfstr(str)!=EOF){
        int len = strlen(str);
        int k;
        sfint(k);
        if( k == 1 ){
            printf("%lld\n", (long long)len*(long long)(len+1)/2);
            continue;
        }
        ll ans = 0;
        fr(i , 0 ,len) c[i] = str[i]-'a'+1;
        c[len] = 0;
        da(c,sa,len+1,27);
        calheight(c,sa,len);
        rmqInit(len-1);

        for(int i = 0; i < len; ++i )    maxrep[i] = 1;

        for(int L = 1; L*k <= len; ++L ) //rep[L次的有多少个]
        {
            for(int i = L; i < len; i += L )    if( maxrep[i-L] == 1 )
            {
                int t = lcp(i-L, i);
                if( t )
                {
                    int j = 0;
                    while( j < L && i-L >= j && str[i-L-j] == str[i-j]
                    {
```

```
                        if( t >= L && lcp(i-L-j, i-j) >= (k-1)*L )
                            maxrep[i-L-j] = max(maxrep[i-L-j], t/L+1);
                        ++t, ++j;
                    }
                }
            }
        }
        for(int i = 0; i < len; ++i )
            if( maxrep[i] >= k )
                ans += ll(maxrep[i] - k + 1);
        printf("%lld\n", ans);
    }
    return 0;
}
```

3.6 后缀自动机

```
const int  maxn=2000010;
const int kinds=26;

char ch[maxn];

struct Sam{
    Sam *son[kinds],*fa;
    int l ,cnt;
    bool vst;
}a[maxn],*head,*last;

int top=-1;
void add(int x){
    Sam *p=&a[++top],*bj=last;
    p->l=last->l+1;last=p;
    for(; bj && !bj->son[x] ; bj = bj->fa) bj->son[x] = p;
    if (!bj) p->fa = head;
    else if (bj->l+1 == bj->son[x]->l) p->fa = bj->son[x];
    else{
        Sam *r = &a[ ++ top],*q = bj->son[x];
        *r = *q ,r->l= bj->l+1, p->fa = q->fa = r;
        for( ; bj && bj->son[x] == q; bj = bj->fa) bj->son[x] = r;
    }
}

Sam *b[maxn];
Sam *sta[maxn];
int dws[maxn];
void caltimes(int n){ // n = lenstr;
    int i;
    for (i = 0; i <= top; ++i) ++dws[a[i].l];
    for (i = 1; i <= n; ++i)    dws[i] += dws[i - 1];
    for (i = 0; i <= top; ++i)   b[--dws[a[i].l]] = &a[i];
    for (last = head, i = 0; i < n; ++i)
```

```
            (last = last->son[ch[i] - 'a'])->cnt++;

      for (i = top; i > 0; --i){
            b[i]->fa->cnt += b[i]->cnt;
      }
}

int main(){
      scanf("%s",ch);
      head = last = &a[++top];
      int n=strlen(ch);
      fr(i,0,n) add( ch[i] - 'a');
      int i;
      caltimes(n);
      return 0;
}
```

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。

```
unsigned int ELFHash(char *str)
{
      unsigned int hash = 0;
      unsigned int x = 0;

      while (*str)
      {
            hash = (hash << 4) + (*str++);
            if ((x = hash & 0xF0000000L) != 0)
            {
                  hash ^= (x >> 24);
                  hash &= ~x;
            }
      }
      return (hash & 0x7FFFFFFF);
}
```

3.8 散列 hash

```
struct hash_map{
      const static int P = 999887;
```

```
    int head[P], next[N],key[N];
    int sz;
    inline void init(){
        cl(head), sz = 0;
    }
    inline int find(uint val){
        int x = val % P;
        for (int i=head[x]; i; i=next[i])
            if (key[i] == val) return i;
        return 0;
    }
    inline int insert(uint val){
        ++sz; key[sz] = val;
        int x = val % P; next[sz] = head[x]; head[x] = sz;
        return sz;
    }
} hashed;
```

3.9 可获取任意段字符串的 hash

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

4 图论

4.1 前向星

```
const int N = 1010;const int M = 2010;
struct Edg
{
    int u,v,w,nxt;
}edg[M];
int tote,head[N];
void init(){
    tote = 0;
    memset(head,-1,sizeof(head));
}
inline void addedg(int u,int v){
    edg[tote].u=u;edg[tote].v=v;edg[tote].nxt=head[u];head[u]=tote++;
};
inline void addedg(int u,int v,int w){
    edg[tote].u=u;edg[tote].v=v;edg[tote].w=w;edg[tote].nxt=head[u];hea
};
```

4.2 并差集

```
struct DisjointSet{
    int fa[N];
    int tot;
    void init(int n){
        fr(i, 0 ,n){
            fa[i] = i;
        }
        tot = 0;
    }
    int find(int x){
        return x==fa[x]?x:fa[x]=find(fa[x]);
    };
    void un(int x,int y){
        int fx = find(x);
        int fy = find(y);
        if(fx != fy){
            fa[fy] = fx;
            tot--;
        }
    }
}DS;
```

4.3 spfa

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

```
        u = q.front();
        q.pop();
        for all edge(u, v, e)
            if(d[v] > d[u] + e){
                d[v] = d[u] + e;
                if(不在中vq) {  //这里用vst
                    q.push(v);
                    if(入队次数v==n) return false;
                }
            }退出队列

    }
    return true;
}
```

4.4 LCA

```
const int MAXM = 16;
const int MAXN = 1 << MAXM;
struct LCA {
    vector<int> e[MAXN];
    int d[MAXN], p[MAXN][MAXM];
    void dfs_(int v, int f) {
        p[v][0] = f;
        for (int i = 1; i < MAXM; ++i) {
            p[v][i] = p[p[v][i - 1]][i - 1];
        }
        for (int i = 0; i < (int)e[v].size(); ++i) {
            int w = e[v][i];
            if (w != f) {
                d[w] = d[v] + 1;
                dfs_(w, v);
            }
        }
    }

    void init(int n) {//vector<int> e[MAXN]
        //copy(e, e + n, this->e);
        d[0] = 0;
        dfs_(0, 0);
    }

    int up_(int v, int m) {
        for (int i = 0; i < MAXM; ++i) {
            if (m & (1 << i)) {
                v = p[v][i];
            }
        }
        return v;
    }

    int lca(int a, int b) {
```

```
        if (d[a] > d[b]) {
            swap(a, b);
        }
        b = up_(b, d[b] - d[a]);
        if (a == b) {
            return a;
        } else {
            for (int i = MAXM - 1; i >= 0; --i) {
                if (p[a][i] != p[b][i]) {
                    a = p[a][i];
                    b = p[b][i];
                }
            }
            return p[a][0];
        }
    }
    void add(int u,int v){
        e[u].push_back(v);
    }
} lca;
```

4.5 Dinic

```
const int pN=2000,eN=3000000;
struct Edge{
    int u,v,nxt;
    int w;
}e[eN];

int en,head[pN];

void init(){
    memset(head,-1,sizeof(head));
    en=0;
}
void add(int u,int v,int w){
    e[en].u=u;e[en].v=v;e[en].w=w;e[en].nxt=head[u];head[u]=en++;
    e[en].u=v;e[en].v=u;e[en].w=0;e[en].nxt=head[v];head[v]=en++;
}

int cur[pN],sta[pN],dep[pN];
int max_flow(int n,int s,int t){
    int tr,flow = 0;
    int i,u,v,f,r,top;    //即是ffront 队列的头部
    int j;
    while(1){
        memset(dep,-1,n*sizeof(int));
        for( f = dep[ sta[0] = s ] = 0 ,r = 1;f != r;){
            for( u = sta[f++],i = head[u];i != -1;i = e[i].nxt){
                if (e[i].w && dep[ v = e[i].v] == -1){
                    dep[v] = dep[u] +1;
                    sta[r ++] = v; //将入队列v 向后标号法
```

```
                    if (v == t){
                        f = r;
                        break;
                    }
                }
            }
        }
        if (-1 == dep[t]) break;
        memcpy(cur,head,n*sizeof(int));
        for (i = s,top = 0; ;){
            if (i == t){
                for( j =0 , tr = inf; j < top; ++j){   //找出一条增广路的最小边权
                    if (e[ sta[j] ].w < tr){
                        tr = e[ sta[ f = j ] ].w;   //一个简单优化每一次不用从头开始找增广
                    }
                }
                for( j = 0;j < top; ++j){
                    e[ sta[j] ].w -= tr;
                    e[ sta[j]^1].w += tr;
                }
                flow += tr;
                i = e[ sta[top = f] ].u;
            }
            for(j = cur[i]; cur[i] != -1; j = cur[i] = e[cur[i]].nxt) //为当前的栈顶元素i
                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

            if (cur[i] != -1){    //就是这个点还有出度
                sta[ top++ ] = cur[i];
                i = e[ cur[i] ].v;
            }
            else{
                if (top == 0) break;
                dep[i] = -1;
                i = e[sta[--top]].u;
            }
        }
    }
    return flow;
}
```

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)；

```
#define inf 1000000000
using namespace std;

```

```
const int pN=5000,eN=100000;

struct Edge{
    int u,v,nxt;
    int w;
}e[eN];

int en,head[pN];

void init(){
    memset(head,-1,sizeof(head));
    en=0;
}
void add(int u,int v,int w){
    e[en].u=u;e[en].v=v;e[en].w=w;e[en].nxt=head[u];head[u]=en++;
    e[en].u=v;e[en].v=u;e[en].w=0;e[en].nxt=head[v];head[v]=en++;
}
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
void BFS(int n,int s,int t){
    memset(dep,-1,n * sizeof(int));
    memset(gap, 0 ,n * sizeof (int));
    gap[0] = 1;
    int f = 0,r = 0,u,v;
    dep[ t ] = 0; que[r ++] = t; //从后外前面标号
    while(f != r){
        u = que[f ++];
        if ( f == pN) f = 0;
        for(int i = head[u];i != -1;i = e[i].nxt){
            v = e[i].v;
            if (e[i].w != 0 || dep[v] != -1) continue;  //如果容量为0 就根本到不到它
            que[ r++ ] = v;
            if (r == pN) r = 0;
            dep[ v ] = dep[ u ] + 1;
            ++ gap[dep[ v ]];  //这里的就是每一层有多少个点gap
        }
    }
}

int cur[pN],sta[pN];
int sap(int n,int s,int t){          //为总的点个数n 包括源点和汇点
    int flow = 0;
    BFS(n,s,t);
    int top = 0,u = s,i;
    memcpy(cur,head,n*sizeof(int));   //当前弧
    while( dep[s] < n){
        if ( u == t){
            int tmp = inf;
            int pos;
            for(i = 0;i < top;i++){
```

```
                if (tmp > e[ sta[i] ].w){
                    tmp = e[ sta[i] ].w;
                    pos = i;
                }
            }
            for(i = 0;i < top; ++i){
                e[ sta[i] ].w -= tmp;
                e[ sta[i]^1 ].w += tmp;
            }
            flow += tmp;
            top = pos;
            u = e[sta[top]].u;
        }
        if(u != t && gap[dep[u] - 1] == 0)  break;  //gap 优化出现断层后直接退出
        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
            if(e[i].w != 0 && dep[u] == dep[e[i].v] + 1) break;  //找到了一条最短增广路
        if (i != -1) cur[ u ] = i,sta[top ++] = i, u = e[i].v;
        else{
            //这里与不同dinic
            int mn = n;
            for (i = head[u] ;i != -1;i = e[i].nxt){
                if ( e[i].w != 0 && mn > dep[ e[i].v ] ){
                    mn = dep[ e[i].v ] ;
                    cur[u] = i;
                }
            }
            -- gap[ dep[u] ];
            dep[u] = mn + 1;
            ++ gap[ dep[u] ];
            if (u != s) u = e[sta[--top]].u;
        }
    }
    return flow;
}
```

4.7 费用流

```
const int inf = 0xffffff;
#define M 200001
#define maxx 2000
class Mcmf{
public:
    struct T{
        int u, v, w;
        int nxt, cost;
    }edge[M];
    int en;
    int visit[M], pre[M], dist[M], que[M], vis[M], pos[M];
```

```
    void init(){
        memset(vis,-1,sizeof(vis));
        en=0;
    }
    void add(int u, int v, int w, int cost)
    {
        edge[en].u = u,edge[en].v = v, edge[en].w = w, edge[en].cost =
        edge[en].nxt = vis[u], vis[u] = en++;
        edge[en].u= v, edge[en].v = u, edge[en].w = 0, edge[en].cost = -
        edge[en].nxt = vis[v], vis[v] = en++;
    }
    bool spfa(int n,int s,int t){
        int v,k;
        for (int i = 0;i <= n; i++){
            pre[i] = -1,visit[i] = 0;
        }
        int f = 0,r = 0;
        for (int i = 0;i <= n; ++i) dist[i] = -1;
        que[r ++] = s;pre[s] = s;dist[s] = 0;visit[s] = 1;
        while(f != r){
            int u = que[f ++];
            visit[u] = 0;
            for (k = vis[u] ;k != -1;k = edge[k].nxt){
                v = edge[k].v;
                if (edge[k].w && dist[u] + edge[k].cost > dist[v]){
                    dist[v] = dist[u] + edge[k].cost;
                    pre[v] = u;
                    pos[v] = k;    //是哪一条边到大的v 巧妙呀值得学习一
下 ~~~
                    if (! visit[v]){
                        visit[v] = 1;
                        que[r ++] = v;
                    }
                }
            }
        }
        if (pre[t] != -1 &&dist[t] > -1) return 1;
        return 0;
    }
    int mnCostFlow(int n,int s,int t){
        if (s == t){}
        int flow =0,cost =0;
        while(spfa(n,s,t)){
            int u,mn = inf;
            for( u = t;u != s; u = pre[u])
                if (mn > edge[pos[u]].w) mn = edge[pos[u]].w;
            flow += mn;
            cost += dist[t] * mn;
            for(u = t;u != s;u = pre[u]){
                edge[pos[u]].w -= mn;
                edge[pos[u]^1].w += mn;
```

```
            }
        }
        return cost;
    }
}mcf;
```

5 计算几何

5.1 动态凸包

```
const double eps = 1e-9;
typedef pair<int,int> pii;

struct dynamic_Convex{
    map<int,int> cvex[2]; //cvex[0] upper contex line, cvex[1] lower co
    map<int,int>::iterator p,q,it;
    double cross(pii a,pii b,pii c){
        return (double(b.first - a.first)) * (double(c.second - a.secon
            - (double(b.second - a.second))*(double(c.first - a.first))
    }
    bool IsUnderUpper(map<int,int> &st,int x,int y){  //check if the po
        if( !st.size()) return false;
        if (x < st.begin()->first || x > (--st.end())->first ) return f
        if (st.find(x) != st.end()) return y <= st[x];
        p = st.upper_bound(x);
        q = p;q--;
        return !(cross(make_pair(x,y) , *q,*p) > eps);
    }
    void insUpperConvex(map<int,int> &st, int x,int y){ //insert a poin
        if( IsUnderUpper(st,x,y) ) return ;
        st[x] = y;
        p = st.upper_bound(x);
        it = p;it--;
        if ( p!=st.end()){
            q = p;q++;
            while(q != st.end() && cross(make_pair(x,y) , *p, *q) >-eps
                st.erase(p);p = q;q++;
            }
        }
        if ( it != st.begin() ){
            p = it;p--;q = p ;q--;
            while(p != st.begin() && cross(make_pair(x,y),*q,*p) > -eps
                st.erase(p);p = q;q--;
            }
        }
    }
    bool judge(int x,int y){ //check if the poing is in the convex hull
        return IsUnderUpper(cvex[0],x,y) && IsUnderUpper(cvex[1],x,-y);
    }
    void ins(int x,int y){ //insert a point to convex hull;
        insUpperConvex(cvex[0],x,y);
        insUpperConvex(cvex[1],x,-y);
    }
}dc;
```