\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*头文件\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<cstdio>

#include<cmath>

#include<stack>

#include<map>

#include<set>

#include<queue>

#include<deque>

#include<string>

#include<utility>

#include<sstream>

#include<cstring>

#include<iostream>

#include<algorithm>

using namespace std;

typedef long long lld;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*输入挂\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

inline void \_read(int &x){

char t=getchar();bool sign=true;

while(t<'0'||t>'9')

{if(t=='-')sign=false;t=getchar();}

for(x=0;t>='0'&&t<='9';t=getchar())x=x\*10+t-'0';

if(!sign)x=-x;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*输入挂\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

inline void \_read(int &x){

char t=getchar();bool sign=true;

while(t<'0'||t>'9')

{if(t=='-')sign=false;t=getchar();}

for(x=0;t>='0'&&t<='9';t=getchar())x=x\*10+t-'0';

if(!sign)x=-x;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*并查集\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int find(int x){

return x == f[x] ? x : f[x] = find(f[x]);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*树状数组\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int lowbit(int x){

return x & (-x);

}

void add(int pos, int v){

while(pos <= all){

c[pos] += v;

//c[pos]%=1000000009;

pos += lowbit(pos);

}

}

int sum(int pos){

int s = 0;

while(pos >= 1){ // not 0

s += c[pos];

s %= 1000000009;

pos -= lowbit(pos);

}

return s;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RMQ\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int rmq\_init(int n){

for(int i = 1; i <= n; i++) d[i][0]=a[i];

int t=log(n) / log(2);

for(int j = 1; j <= t; j++)

for(int i = 1; i - 1+ (1 << j) <= n; i++)

d[i][j] = max(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);

}

int rmq\_query(int a, int b){

int k = log(b - a + 1) / log(2);

return max(d[a][k], d[b - (1 <<k ) + 1][k]);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*线段树\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef long long ll;

const ll maxn = 112101;

const ll mod = 1000000009, inf = 1LL << 50;

int lc(int x){ return x << 1;}

int rc(int x){ return x << 1 | 1;}

struct seg {

int l,r;

ll mx;

int mid() {

return l+r>>1;

};

} tree[maxn<<2];

void init\_tree(int l,int r,int x)

{

tree[x].l=l,tree[x].r=r;

tree[x].mx=-1;

if(l==r)return;

int mid=l+r>>1;

init\_tree(l,mid,lc(x));

init\_tree(mid+1,r,rc(x));

}

void insert(int pos,int x,ll num)

{

if(tree[x].l==tree[x].r) {

tree[x].mx=num;

return;

}

int mid=tree[x].mid();

if(pos<=mid)insert(pos,lc(x),num);

else insert(pos,rc(x),num);

tree[x].mx=max(tree[lc(x)].mx,tree[rc(x)].mx);

}

ll query(int l,int r,int x)

{

if(l<=tree[x].l && tree[x].r<=r)

return tree[x].mx;

int mid=tree[x].mid();

ll res=-1;

if(l<=mid)res=max(res,query(l,r,lc(x)));

if(r>mid)res=max(res,query(l,r,rc(x)));

return res;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*数学\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//费马小定理：已知p是质数且gcd(a, p) = 1，则 a^(p-1) ≡ 1 (mod p), 所以 a\*a^(p-2) ≡ 1 (mod p)。

lld find(lld x) //x在%mod下的逆元

{

lld k = mod - 2, ans = 1;

while(k)

{

if (k & 1) {

ans = ans \* x % mod;

}

x = x \* x % mod;

k >>= 1;

}

return ans;

}

// 前n个数的逆元

Inv[1=1;

Inv[i]=(LL)Inv[mo%i]\*(mo-mo/i)%mo;

//最大公约数

int gcd(int m,int n){

int t;

if(m<n){t = n;n = m;m = t;}

if(n == 0)

return m;

else

return gcd(n,m % n);

}

//最小公倍数

int lcm(int a,int b){

return a/gcd(a,b) \* b;

}

//扩展欧几里得

如果gcd(a，b)=d，则存在m，n，使得d = ma + nb，称呼这种关系为a、b组合整数d，m，n称为组合系数。当d=1时，有 ma + nb = 1 ，此时可以看出m是a模b的乘法逆元，n是b模a的乘法逆元。

//hash

typedef long long lld;

const int N=1301171;

const int M=44444;

struct Hash{

int a[N] , cnt;

lld nxt[M],head[N],val[M];

void clear(){

cnt=1;

memset(head ,0,sizeof(head));

}

void insert(lld x){

int p=x%N;

if(p<0)p+=N;

nxt[cnt]=head[p];

val[cnt]=x;

head[p]=cnt++;

}

bool find(lld x){

int p=x%N ;

if(p<0)p+=N;

for(int e=head[p];e;e=nxt[e]){

if(val[e]==x) return true;

}

return false;

}

}ha;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SPFA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct Edge

{

lld s, t, c;

lld next;

}e[M];

lld cnt;

lld sumFlow;

lld pre[N], dist[N], eat[N];

lld inq[N], vt[N], a[N], b[N], c[N], d[N], x[N];

void init()

{

cnt = 0;

memset(pre, -1, sizeof(pre));

}

void addedge(lld s, lld t, lld c)

{

e[cnt].s = s, e[cnt].t = t, e[cnt].c = c, e[cnt].next = pre[s];

pre[s] = cnt++;

}

lld SPFA(lld s, lld t, lld n)

{

queue <lld> q;

memset(inq, 0, sizeof(inq));

memset(eat, -1, sizeof(eat));

memset(vt, 0 ,sizeof(vt));

for(lld i = 0; i <= n; i ++) dist[i] = inf;

inq[s] = 1;

vt[s]++;

dist[s] = 0;

q.push(s);

while(!q.empty()){

lld u = q.front();

q.pop();

inq[u] = 0;

if(vt[u] > n) break;

for(lld i = pre[u]; i != -1; i = e[i].next){

lld v = e[i].t;

if(dist[v] > dist[u] + e[i].c){

dist[v] = dist[u] + e[i].c;

eat[v] = i;

if(!inq[v]){

q.push(v);

vt[v]++;

inq[v] = 1;

}

}

}

}

if(dist[t] == inf || !q.empty()) return inf;

return dist[t];

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*强连通分量\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

stack<int> S;

vector<int> g[N];

int sv[N], dfn[N], low[N];

int in[N], out[N], num[N];

int id; //连通分支个数

void dfs(int u, int dep){

low[u] = dfn[u] = dep;

S.push(u);

for(int i = 0; i < g[u].size(); ++i){

int v = g[u][i];

if(dfn[v] == -1) dfs(v, dep + 1);

if(sv[v] == -1) low[u] = min(low[u], low[v]); // 判断不可少

}

if(low[u] == dfn[u]){

++id;

while(S.top() != u){

sv[S.top()] = id;

S.pop();

}

sv[S.top()] = id;

S.pop();

}

}

int tarjan(int n){

while(!S.empty()) S.pop();

memset(dfn, -1, sizeof(dfn));

memset(sv, -1, sizeof(sv));

id = 0;

for(int i = 1; i <= n; ++i) if (dfn[i] == -1) dfs(i, 0);

return id;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*割点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

割点：如果在图G中删去一个结点u后，图G的连通分枝数增加，即W(G-u)>W(G)，则称结点u为G的割点，又称关节点。

桥：如果在图G中删去一条边e后，图G的连通分支数增加，即W(G-e)>W(G)，则称边u为G的桥，又称割边或关节边。

双连通分支：G中不含割点的极大连通子图称为G的双连通分支，又称为G的块。

注意是否有多重边

对于边的双连通，若边(u,v)，dfn[u]<low[v]（即不在环内），则为桥。对于点的双连通，若dfn[u]<=low[v]（小于是不在环内的点，等于是环与外界的唯一连接点），或者u为根且u有多个子树（说明u不在环内，因为没有横叉边）则为割点。

定理:当把边的双连通分支缩点后形成了一个有向无环图，叶子（度为1的点）的个数为n，则需要在原图中添加(n + 1)/2条边，可以使原图变为没有桥的双连通图。

边的双连通，可以求出桥，不经过桥dfs。

// 割点

void dfs(int u, int dep){

low[u] = dfn[u] = dep; //访问时间

for (int i = 0; i < g[u].size(); ++i) {

int v = g[u][i];

if(dfn[v] == -1){

dfs(v, dep + 1);

if(u == root) son++;

else{

low[u] = min(low[u], low[v]);

if(low[v] >= dfn[u]) cut[u]++; //无法回溯到u的父亲节点， 注意无向图的两条有向边，所以有v -> u。须看清题目是否有多重边

}

}

else low[u] = min(low[u], dfn[v]); //回边

}

}

int tarjan(int n){

memset(dfn, -1, sizeof(dfn));

ans = 0;

for(int i = 0; i < n; i ++)

if(dfn[i] == -1){

son = 0;

root = i;

ans++;

dfs(root, 0);

cut[root] = son - 1;

}

return 0;

}

// 桥

void dfs(int u, int dep, int id){

low[u] = dfn[u] = dep; //访问时间

for (int i = pre[u]; i != -1; i = e[i].next){

int v = e[i].to;

if(e[i].id == id) continue;

if(dfn[v] == -1) dfs(v, dep + 1, e[i].id);

low[u] = min(low[u], low[v]);

if(low[v] > dfn[u] && cut[e[i].id] == 0) cut[e[i].id] = 1, ++ans;

}

}

int tarjan(int n){

memset(dfn, -1, sizeof(dfn));

memset(cut, 0, sizeof(cut));

ans = 0;

dfs(1, 0, -1);

return 0;

}

// 强连通分量

void dfs(int u, int dep){

low[u] = dfn[u] = dep;

S.push(u);

for(int i = 0; i < g[u].size(); ++i){

int v = g[u][i];

if(dfn[v] == -1) dfs(v, dep + 1);

if(sv[v] == -1) low[u] = min(low[u], low[v]); // 判断不可少

}

if(low[u] == dfn[u]){

++id;

while(S.top() != u){

sv[S.top()] = id;

S.pop();

}

sv[S.top()] = id;

S.pop();

}

}

int tarjan(int n){

while(!S.empty()) S.pop();

memset(dfn, -1, sizeof(dfn));

memset(sv, -1, sizeof(sv));

id = 0;

for(int i = 1; i <= n; ++i) if (dfn[i] == -1) dfs(i, 0);

return id;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*网络流dinic\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const int N = 310, M = 500000, inf = 100100100; //注意修改点的范围

struct Edge{

int s, t, f, id;

int next;

}e[M];

int pre[N], tpre[N], dis[N], ans[M], cnt;

int a[N][N];

void addedge(int s, int t, int f, int id){ //有上下界的最大流模板

e[cnt].s = s, e[cnt].t = t, e[cnt].f = f, e[cnt].id = id, e[cnt].next = pre[s];

pre[s] = cnt++;

e[cnt].s = t, e[cnt].t = s, e[cnt].f = 0, e[cnt].id = 0, e[cnt].next = pre[t];

pre[t] = cnt++;

}

int bfs(int s, int t){

memset(dis, 0, sizeof(dis));

queue<int> q;

dis[s] = 1;

q.push(s);

while(!q.empty()){

int id = q.front();

q.pop();

for(int i = pre[id]; i != -1; i = e[i].next)

if(dis[e[i].t] == 0 && e[i].f > 0){

dis[e[i].t] = dis[id] + 1;

q.push(e[i].t);

}

}

return dis[t] != 0;

}

int dfs(int s, int t, int cur\_flow){

int lev = cur\_flow;

if(s == t) return cur\_flow;

for(int &i = tpre[s]; i != -1; i = e[i].next)

if(dis[e[i].t] == dis[s] + 1 && e[i].f > 0 && lev > 0){

int temp = dfs(e[i].t, t, min(lev, e[i].f));

e[i].f -= temp;

e[i^1].f += temp;

lev -= temp;

}

return cur\_flow - lev;

}

int dinic\_flow(int s, int t){ //注意是从s到t，故而s最小，t最大

int sum = 0;

while(bfs(s, t)){

int flow;

for(int i = s; i <= t; i++) tpre[i] = pre[i];

while((flow = dfs(s, t, inf))) sum += flow;

}

return sum;

}

void flow\_init(){

memset(pre, -1, sizeof(pre));

memset(ans, 0, sizeof(ans));

cnt = 0;

}

int limit\_min\_flow(int n, int m, int mid, int k, int &flow){

int src = 0, dest = n + 2 \* m + 1;

for (int i = 1; i <= n; ++i) {

addedge(src, i, 1, 0);

}

if (mid == 1) {

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (a[i][j]) {

addedge(i, n + j, 1, 0);

}

else {

addedge(i, n + m + j, 1, 0);

}

}

}

}

for (int i = 1; i <= m; ++i) {

addedge(n + i, dest, 1, 0);

if (mid == 1) {

addedge(n + m + i, n + i, k, 0);

}

}

flow += dinic\_flow(src, dest);

return flow >= mid \* n;

}

int main(){

int ca = 0, CASE;

cin >> CASE;

while(CASE--){

int n, m, L, k;

cin >> n >> m >> L >> k;

memset(a, 0, sizeof(a));

for (int i = 0; i < L; ++i) {

int u, v;

cin >> u >> v;

a[u][v] = 1;

}

int ans = 0, flow = 0;

flow\_init();

while (true) {

if (limit\_min\_flow(n, m, ans + 1, k, flow)) {

++ans;

}

else {

break;

}

}

printf("Case %d: %d\n", ++ca, ans);

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*最大权闭合图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

构造一个源点S，汇点T。我们将S与所有权值为正的点连一条容量为其权值的边，将所有权值为负的点与T连一条容量为其权值的绝对值的边，原来的边将其容量定为正无穷。

（S包含，那么S指向的点也包含）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*最小费用最大流\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const int N = 502;

const int M = 110002;

const int inf = 100100100;

struct Edge

{

int s, t, f, c;

int next;

};

char str[N][N];

vector<pair<int, int> > mV, hV;

struct MCMF { // minCostMaxFlow

int cnt, sumFlow;

int pre[N], dist[N], eat[N];

int inq[N], vt[N];

struct Edge e[M];

void reset()

{

cnt = 0;

memset(pre, -1, sizeof(pre));

}

void addedge(int s, int t, int f, int c)

{

e[cnt].s = s, e[cnt].t = t, e[cnt].f = f, e[cnt].c = c, e[cnt].next = pre[s];

pre[s] = cnt++;

e[cnt].s = t, e[cnt].t = s, e[cnt].f = 0, e[cnt].c = -c, e[cnt].next = pre[t];

pre[t] = cnt++;

}

bool SPFA(int s, int t, int n)

{

queue <int> q;

memset(inq, 0, sizeof(inq));

memset(eat, -1, sizeof(eat));

memset(vt, 0 ,sizeof(vt));

for(int i = 0; i <= n; i ++) dist[i] = inf;

inq[s] = 1;

vt[s]++;

dist[s] = 0;

q.push(s);

while(!q.empty()){

int u = q.front();

q.pop();

inq[u] = 0;

if(vt[u] > n) break;

for(int i = pre[u]; i != -1; i = e[i].next){

int v = e[i].t;

if(e[i].f && dist[v] > dist[u] + e[i].c){

dist[v] = dist[u] + e[i].c;

eat[v] = i;

if(!inq[v]){

q.push(v);

vt[v]++;

inq[v] = 1;

}

}

}

}

if(dist[t] == inf || !q.empty()) return false;

return true;

}

int look(int s, int t, int n)

{

int flow = 0; // 总流量

int i, minflow, mincost;

mincost = 0;

while(SPFA(s, t, n)){

minflow = inf + 1;

for(i = eat[t]; i != -1; i = eat[e[i].s])

if(e[i].f < minflow) minflow = e[i].f;

flow += minflow;

for(i = eat[t]; i != -1; i = eat[e[i].s]){

e[i].f -= minflow;

e[i^1].f += minflow;

}

mincost += dist[t] \* minflow; //单位流量的最小代价dist[t]

}

sumFlow = flow; // 题目需要流量，用于判断

return mincost;

}

}acm;

int main()

{

int n, m;

while(cin >> n >> m){

if (n + m == 0) {

break;

}

int src = 0, dest = 222;

for (int i = 1; i <= n; ++i) {

cin >> (str[i] + 1);

}

acm.reset();

mV.clear();

hV.clear();

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (str[i][j] == 'm') {

mV.push\_back(make\_pair(i, j));

acm.addedge(src, mV.size(), 1, 0);

}

if (str[i][j] == 'H') {

hV.push\_back(make\_pair(i, j));

acm.addedge(hV.size() + 111, dest, 1, 0);

}

}

}

for (int i = 0; i < mV.size(); ++i) {

for (int j = 0; j < hV.size(); ++j) {

acm.addedge(i + 1, 111 + j + 1, inf, abs(mV[i].first - hV[j].first) + abs(mV[i].second - hV[j].second));

}

}

int ans = acm.look(src, dest, dest);

printf("%d\n", ans);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*二分图匹配\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

求二分图匹配可以用最大流(Maximal Flow)或者匈牙利算法(Hungarian Algorithm)。

//匈牙利算法

struct Hungary {

static const int N = 1000;

int nx, ny, g[N][N], linker[N];

bool used[N];

int dfs(int u) {

for(int v = 0; v < ny; ++v) {

if(g[u][v] && !used[v]) {

used[v]=true;

if(linker[v] == -1 || dfs(linker[v])) {

linker[v] = u;

return 1;

}

}

}

return 0;

}

int look() {

int res = 0;

for(int u = 0; u< nx; ++u) {

memset(used, 0, sizeof(used));

res += dfs(u);

}

return res;

}

void reset() {

memset(g, 0, sizeof(g));

memset(linker, -1, sizeof(linker));

}

void addEdge(int u, int v) {

g[u][v] = 1; //注意，这里是mat[a][b]=mat[b][a]=0

}

}acm;

int main(){

int k;

while(cin >> k >> acm.nx >> acm.ny, k){

acm.reset();

for(int i = 1; i <= k; ++i){

int a, b;

scanf("%d%d", &a, &b);

acm.addEdge(a - 1, b - 1);

}

printf("%d\n", acm.look());

}

}

//匈牙利算法的HK实现

struct HungaryHK {

static const int N = 1000, inf = 1<<28;

bool bmap[N][N], bmask[N];

int nx,ny,dis;

int cx[N], cy[N], dx[N], dy[N];

bool bfs()

{

queue<int> q;

dis = inf;

memset(dx,-1,sizeof(dx));

memset(dy,-1,sizeof(dy));

for(int i = 0;i < nx; ++i) if(cx[i] == -1) dx[i] = 0, q.push(i); //从未匹配点X[i]出发

while(!q.empty()){

int u = q.front();

q.pop();

if(dx[u] > dis) break;

for(int v = 0; v < ny; ++v)

if(bmap[u][v] && dy[v]== -1){

dy[v] = dx[u] + 1;

if(cy[v] == -1) dis = dy[v]; // 找到未匹配点Y，即是增广路存在

else dx[cy[v]] = dy[v] + 1, q.push(cy[v]);

}

}

return dis != inf;

}

int dfs(int u)

{

for(int v = 0; v < ny; ++v){

if(!bmask[v] && bmap[u][v] && dy[v] == dx[u]+1){

bmask[v]=1;

if(cy[v] != -1 && dy[v] == dis) continue;

if(cy[v] == -1 || dfs(cy[v])){

cy[v] = u;

cx[u] = v;

return 1;

}

}

}

return 0;

}

int look(){

int res = 0;

memset(cx, -1, sizeof(cx));

memset(cy, -1, sizeof(cy));

while(bfs()){

memset(bmask, 0, sizeof(bmask));

for(int i = 0; i < nx; ++i) if(cx[i]==-1)

res += dfs(i);

}

return res;

}

void reset() {

for(int i = 0; i <= nx; ++i) for(int j = 0; j <= ny; ++j) bmap[i][j] = 0;

}

void addEdge(int u, int v) {

bmap[u][v] = 1; //注意，这里不是mat[a][b]=mat[b][a]=0

}

}acm;

//KM算法-二分图最优匹配

struct KM {

static const int N = 2010, inf = 10000000;

int lx[N], ly[N], mat[N][N];

int vx[N], vy[N], link[N], slack[N], a[N];

int dfs(int k, int n){

vx[k] = 1;

for(int i = 1; i <= n; i++){

if(vy[i]) continue;

int t = lx[k] + ly[i] - mat[k][i];

if(t == 0){ //注意此处的虚拟边，权值为零不影响最佳匹配的边，方便遍历

vy[i] = 1;

if(link[i] == -1 || dfs(link[i], n)){

link[i] = k;

return 1;

}

}

else slack[i] = min(t, slack[i]);

}

return 0;

}

int look(int n){

for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) lx[i] = max(lx[i], mat[i][j]);

for(int i = 1; i <= n; i++){

for(int j = 1; j <= n; j++) slack[j] = inf;

while(true){

memset(vx, 0, sizeof(vx));

memset(vy, 0, sizeof(vy));

if(dfs(i, n)) break;

int Min = inf;

for(int t = 1; t <= n; t++)

if(!vy[t]) Min = min(Min, slack[t]);

for(int t = 1; t <= n; t++)

if(vx[t]) lx[t] -= Min;

for(int t = 1; t <= n; t++)

if(vy[t]) ly[t] += Min;

else slack[t] -= Min;

}

}

int sum = 0;

for(int i = 1; i <= n; i++)

if(link[i] != -1) sum += mat[link[i]][i];

return sum;

}

void reset(int n) {

for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) mat[i][j] = 0; //初始化, 默认存在权值为0的边, 如果是最小值，要初始化成负无穷

for(int i = 1; i <= n; i++) lx[i] = 0; //初始化,如果最小值需修改-inf

memset(link, -1, sizeof(link));

memset(ly, 0, sizeof(ly));

}

}acm;

//二分图多重匹配，建模直接网络流。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KMP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int next[N];

char s[N], str[N];

void getNext(char \*s){

// next[j] = max(k : 0 < k < j s[0...k - 1] = s[j - k, j - 1]

// 最长后缀的长度等于相同字符序列的前缀

int n = strlen(s);

int i = 0, j = -1;

next[0] = -1;

while(i < n){

if(j == -1 || s[i] == s[j])

i++, j++, next[i] = j;

else

j = next[j];

}

}

int main(){

while(scanf("%s", s) != EOF){

scanf("%s", str);

getNext(s);

int n = strlen(s), m = strlen(str);

int i = 0, j = 0;

while(i < m){

if(j == -1 || str[i] == s[j])

i++, j++;

else

j = next[j];

}

if(j == 0) cout << j << endl;

else{

string ss(s, s + j);

cout << ss << ' ' << j << endl;

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AC自动机\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

struct Node {

int word;

Node \*ch[26], \*fail;

};

struct Trie {

static const int N = 1001000, inf = 10000000; // 注意点的大小

char s[N], str[N];

struct Node node[N];

int cnt;

Node \*head;

Node\* need(int t){

memset(node + t, 0, sizeof(Node)); // 注意大小写！！

return &node[t];

}

void reset() {

cnt = 0;

head = need(cnt);

}

void insert(char s[]){ //字典树建树

Node \*p = head;

for(int i = 0; s[i]; i++){

int id = s[i] - 'a';

if(!p->ch[id]) p->ch[id] = need(++cnt);

p = p->ch[id];

}

p->word++;

}

void build(){

queue<Node\*> q;

q.push(head);

while(!q.empty()){

Node \*p = q.front();

q.pop();

for(int i = 0; i < 26; i++){

Node \*fail = (p == head) ? head : p->fail->ch[i]; //如果为头指针，则失败时指向自己

if(!p->ch[i]) p->ch[i] = fail; //很方便的一点

else{

p->ch[i]->fail = fail;

q.push(p->ch[i]);

}

}

}

}

int look(char str[]){

int sum = 0;

Node \*p = head;

for(int i = 0; str[i]; i++){

int id = str[i] - 'a';

p = p->ch[id];

Node\* tp = p;

while(tp != head)

if(tp->word > 0)

sum += tp->word, tp->word = 0;

else

tp = tp->fail;

}

return sum;

}

}acm;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*manachar\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const int maxn = 1010100;

char s[maxn \* 2], str[maxn \* 2];

int p[maxn \* 2];

void manachar(int n){

int id, mx=0;

for(int i = 1; i < n; i++){

if(mx > i) p[i] = min(p[2 \* id - i], mx - i);

else p[i] = 1;

while(str[i + p[i]] == str[i - p[i]]) p[i]++;

if(p[i] + i > mx) mx = p[i] + i, id = i;

}

}

int main(){

while(scanf("%s", s) != EOF){

str[0] = '@';

int i, j;

for(i = 0, j = 1; s[i]; i++){

str[j++] = '#';

str[j++] = s[i];

}

str[j++] = '#';

int n = j;

manachar(n);

int ans = 0;

for(i = 1; i < n; i++) ans = max(ans, p[i] - 1);

printf("%d\n", ans);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*后缀自动机\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

max：即代码中 val 变量，它表示该状态能够接受的最长的字符串长度。

min：表示该状态能够接受的最短的字符串长度。实际上等于该状态的 par 指针指向的结点的 val + 1。

max-min+1：表示该状态能够接受的不同的字符串数。

right：即 end-set 的个数，表示这个状态在字符串中出现了多少次，该状态能够表示的所有字符串均出现过 right 次。

par：par 指向了一个能够表示当前状态表示的所有字符串的最长公共后缀的结点。所有的状态的 par 指针构成了一个 parent 树，恰好是字符串的逆序的后缀树。

parent 树的拓扑序：序列中第i个状态的子结点必定在它之后，父结点必定在它之前。

struct Node{

Node \*ch[26], \*f;

int ml, multiLCS, LCS;

};

char s[10][N];

vector<int> g[N];

struct SAM {

Node pool[N \* 2], \*last, \*head;

char str[N];

int tot, n, maxLen;

SAM() {

reset();

}

void reset() {

tot = 0;

maxLen = 0;

n = 0;

last = head = NULL;

}

Node\* need(int t){

memset(pool + t, 0, sizeof(pool[t]));

pool[t].multiLCS = N;

return &pool[t];

}

Node\* need(int t, Node\* p) {

need(t);

pool[t] = \*p;

return pool + t;

}

void add(int c, int loc){

Node \*p = last, \*np = need(++tot);

np->ml = loc;

while (p && !p->ch[c]) {

p->ch[c]=np;

p = p->f;

}

last = np;

if(!p) {

np->f = head;

}

else

if(p->ch[c]->ml == p->ml + 1) {

np->f = p->ch[c];

}

else{

Node \*q = p->ch[c], \*newNode = need(++tot, q);

newNode->ml = p->ml + 1;

q->f = np->f = newNode;

while (p && p->ch[c] == q) {

p->ch[c] = newNode;

p = p->f;

}

}

}

void build() {

last = head = need(++tot);

n = (int)strlen(str);

for(int i = 0; i < n; ++i) add(str[i] - 'a', i + 1);

topsSort();

}

void look(char s[]) {

Node \*p = head;

int len = 0;

for (int i = 0; i <= tot; ++i) {

pool[i].LCS = 0;

}

for (int i = 0; s[i] != '\0'; ++i) {

int id = s[i] - 'a';

if (p->ch[id]) {

p = p->ch[id];

++len;

}

else {

while (p) {

if (p->ch[id]) {

len = p->ml + 1;

p = p->ch[id];

break;

}

p = p->f;

}

if (!p) {

p = head;

len = 0;

}

}

p->LCS = max(p->LCS, len);

}

for (int i = n; i > 0; --i) {

for (int j = 0; j < g[i].size(); ++j) {

Node \*p = pool + g[i][j];

if (p->f) {

int len = min(p->f->ml, p->LCS); //已经访问过的最长序列

p->f->LCS = max(p->f->LCS, len);

}

}

}

for (int i = 0; i <= tot; ++i) {

pool[i].multiLCS = min(pool[i].multiLCS, pool[i].LCS);

}

}

void topsSort() {

for (int i = 1; i <= n; ++i) {

g[i].clear();

}

for (int i = 1; i <= tot; ++i) {

g[pool[i].ml].push\_back(i);

}

}

}sam;

int main(int argc, const char \* argv[]) {

int cnt = 0;

while (scanf("%s", s[cnt]) != EOF) {

++cnt;

}

strcpy(sam.str, s[0]);

sam.build();

for (int i = 1; i < cnt; ++i) {

sam.look(s[i]);

}

int ans = 0;

for (int i = 1; i <= sam.tot; ++i) {

if (sam.pool[i].multiLCS != N) {

ans = max(ans, sam.pool[i].multiLCS);

}

}

cout << ans << endl;

return 0;

}