Union de Area de Rectangulos

```
struct T
{
    int x,y1,y2,IoF;
    T(int a=0,int b=0,int c=0,int d=0)
    {
        x=a;
        y1=b;
        y2=c;
        IoF=d;
    }
} L[200005];

int B[200005],r1;
int B1[200005],r2;

bool com(const T &s,const T &p)
{
    return s.x<p.x;
}

int MAXY;
int Stree[3000005];
long long cant[3000005];
int r;

void update(int node,int ini,int fin,int y1,int y2,int IoF)
{
    if(ini>y2 || fin<y1)
        return;
    if(ini>=y1 && fin<=y2)
        Stree[node]+=IoF;
    else
    {
        int piv=(ini+fin)/2;
        update(2*node,ini,piv,y1,y2,IoF);
        update(2*node+1,piv+1,fin,y1,y2,IoF);
    }
    if(Stree[node]==0)
    {
        if(ini==fin)
            cant[node]=0;
        else
            cant[node]=(long long)cant[2*node]+cant[2*node+1];
    }
    else
        cant[node]=(long long)B[fin]-B[ini-1];
}

int main()
{
    int N;
    scanf("%d",&N);

    for(int i=1; i<=N; i++)
    {
        int x1,y1,x2,y2;
        scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
        if(x1>x2)
            swap(x1,x2);
        if(y1>y2)
            swap(y1,y2);
        L[++r]=T(x1,y1,y2,1);
        L[++r]=T(x2,y1,y2,-1);
        B1[++r2]=y1;
        B1[++r2]=y2;
    }
    B1[0]=-1;
    sort(B1+1,B1+r2+1);
    for(int i=1; i<=r2; i++)
        if(B1[i]!=B1[i-1])
            B[++r1]=B1[i];

    sort(L+1,L+r+1,com);

    int last=L[1].x;
    long long area=0;
    for(int i=1; i<=r; i++)
    {
        long long temp=(long long)L[i].x-last;
        temp=(long long)temp*cant[1];
        area=(long long)area+temp;
        last=L[i].x;
        int I=lower_bound(B+1,B+r1+1,L[i].y1)-B;
        int F=lower_bound(B+1,B+r1+1,L[i].y2)-B;
        update(1,1,r1,I+1,F,L[i].IoF);
    }
    cout << area;
    return 0;
}
```

Articulation Points

```
int TD[1005],LOW[1005];
bool mark[1005];
int dc_time;
vector<int>ady[1005];

void A_Points(int nod)
{
    LOW[nod]=TD[nod]=++dc_time;

    int t=ady[nod].size();
    for(int i=0; i<t; i++)
    {
        int nn=ady[nod][i];
        if(!LOW[nn])
        {
            A_Points(nn);
            LOW[nod]=min(LOW[nod],LOW[nn]);
            if(nod==1)
            {
                if(TD[nn]>2)
                    mark[1]=1;
                continue;
            }
            if(TD[nod]<=LOW[nn])
                mark[nod]=1;
        }
        else
            LOW[nod]=min(LOW[nod],TD[nn]);
    }
}

int main()
{
    int n,m;
    scanf("%d%d",&n,&m);

    int a,b;
    for(int i=1; i<=m; i++)
    {
        scanf("%d%d",&a,&b);
```

```
      ady[a].push_back(b);
      ady[b].push_back(a);
   }

   A_Points(1);

   for(int i=1; i<=n; i++)
      if(mark[i])
         printf("%d\n",i);

   printf("TD ->");
   for(int i=1; i<=n; i++)
      printf(" %d",TD[i]);

   printf("\nLOW->");
   for(int i=1; i<=n; i++)
      printf(" %d",LOW[i]);

   return 0;
}
```

Aho Corasick

```
struct node
{
   intpos;
   node* fail;
   node* link;
   node* next[26];
   node()
   {
      pos = -1;
      fail = link = NULL;
      for (int i = 0; i < 26; i++) next[ i ] = NULL;
   }
};
node* root = new node();

voidinsert(char* patt, int idx)
{
   node* curr=root;
   for (int j=0; patt[j]; j++)
   {
      if (curr->next[patt[j] - 'a'] == NULL)
         curr->next[patt[j] - 'a'] = new node();
      curr = curr->next[patt[j] - 'a'];
   }
   curr->pos = idx;
}

voidaho_corasick()
{
   queue<node*> Q;
   for (int i = 0; i<26; i++)
      if ( root->next[i] )
      {
         root->next[i]->fail = root;
         Q.push( root->next[i] );
      }
      else root->next[i] = root;
   while ( !Q.empty() )
   {
      node* t = Q.front();
      Q.pop();
      for (int i = 0; i < 26; i++)
         if ( t->next[i] )
         {
            Q.push( t->next[i] );
```

```
        node* r = t->fail;
        while ( !r->next[i] ) r = r->fail;
        t->next[i]->fail = r->next[i];
        if ( r->next[i]->pos != -1 ) t->next[i]->link = r->next[i];
        else t->next[i]->link = r->next[i]->link; /////multiple matches
      }
  }
}
voidmatch(char text[])
{

  n = strlen( text );
  node* state = root;
  for (int i = 0; i < n; i++)
  {
    while (state->next[ text[i]-'a' ] == NULL)
      state = state->fail;
    state = state->next[ text[i]-'a' ];
    if (state->pos != -1)
      cout<< state->pos<<" found at "<< i << endl;
    for (node* r = state->link; r != NULL; r = r->link)
      cout<< r->pos<<" found in position "<< i << endl;
  }
}
```

ANTIPODAL PAIRS (FOR CONVEX POLYGONS)

```
pair<int,int> q[maxn];
//for each i q[i].first is the first index for whichthe area of (i - 1, i, qi.first) is
largestand q[i].second is one past the last index for which the area of(i - 1, i,
qi.second) is largest
#define next(a, n) ((a) + 1)%n
voidcompute_antipodal(point* P, int n)
{
  int k = 1;
  for(int i=0; i < n; i++)  ///cada iteracion: second de i y first de i + 1
  {
    while( area(P[i], P[next(i,n)], P[k]) - area(P[i], P[next(i,n)], P[next(k,n)]) <
-(1e-9) )k = next(k,n);
    q[next(i,n)].first = k;
    while( fabs(area(P[i], P[next(i,n)], P[k]) - area(P[i], P[next(i,n)],
P[next(k,n)])) < 1e-9 )
      k = next(k,n);
    q[i].second = next(k, n);
  }
}
```

Componentes Biconexas

```
const int
MaxV = 1001,
MaxE = 10001;

typedef pair<int, int> pii;
int V, E;
int i, j;
int a, b;
int size;
int gtime;
stack <pii> Q;
int disc[MaxV];
int back[MaxV];
bool mark[MaxE];
vector<pii> bic[MaxV];
vector<pii> graph[MaxV];

void dfs(int v)
{
  gtime++;
  disc[v] = gtime;
```

```
    back[v] = gtime;
    for (int k = graph[v].size() - 1; k >= 0; k--)
    {
        int next = graph[v][k].first;
        int edge = graph[v][k].second;
        if (!mark[edge])
        {
            Q.push(pii(v, next));
            mark[edge] = true;
        }
        if (!disc[next])
        {
            dfs(next);
            back[v] = min(back[v], back[next]);
            if (back[next] >= disc[v])
            {
                size++;
                for (;;)
                {
                    pii x = Q.top();
                    Q.pop();
                    bic[size].push_back(x);
                    if (x == pii(v, next))
                        break;
                }
            }
        }
        else back[v] = min(back[v], disc[next]);
    }
}

int main()
{
    cin >> V >> E;
    for (i = 0; i < E; i++)
    {
        cin >> a >> b;
        graph[a].push_back(pii(b, i));
        graph[b].push_back(pii(a, i));
    }

    for (i = 1; i <= V; i++)
        if (!disc[i]) dfs(i);

    for (i = 1; i <= size; i++)
    {
        cout << "Biconnected Component: " << i << endl;
        for (j = bic[i].size() - 1; j >= 0; j--)
            cout << bic[i][j].first << " " << bic[i][j].second << endl;
    }

    return 0;
}
```

Bridges

```
typedef pair<int,int>par;
vector<int>ID[1005];//id de las aristas en que esta presente cada nodo
int TD[1005],LOW[1005];
int dc_time;
bool mark[10005];
stack<par>S;
int a,b;
struct T
{
    int nod,nn;
    T(int x=0,int y=0)
```

```
   {
      nod=x;
      nn=y;
   }
   int nextn(int x)
   {
      if(x==nod)
         return nn;
      else
         return nod;
   }
} edge[10005];

void Bridges(int nod)
{
   TD[nod]=LOW[nod]=++dc_time;

   int t=ID[nod].size();
   for(int i=0; i<t; i++)
   {
      int id=ID[nod][i];
      int nn=edge[id].nextn(nod);
      if(!LOW[nn])
      {
         mark[id]=1;
         Bridges(nn);
         if(TD[nod]<LOW[nn])
            S.push(par(nod,nn));
         LOW[nod]=min(LOW[nod],LOW[nn]);
      }
      else if(!mark[id])
         LOW[nod]=min(LOW[nod],TD[nn]);
   }
}

int main()
{
   int n,m;
   scanf("%d%d",&n,&m);

   for(int i=1; i<=m; i++)
   {
      scanf("%d%d",&a,&b);
      ID[a].push_back(i);
      ID[b].push_back(i);
      edge[i]=T(a,b);
   }

   Bridges(1);

   while(!S.empty())
   {
      par A=S.top();
      S.pop();
      printf("%d %d\n",A.first,A.second);
   }
   return 0;
}
```

Closest Pair Points

```
int square(int n)
{
   return n*n;
}
struct T
{
   int x,y,id;
```

```
    T(int a=0,int b=0)
    {
        x=a;
        y=b;
    }
    bool operator <(const T &p)const
    {
        return x<p.x;
    }
} P[100005];
double dist(T a,T b)
{
    return sqrt(square(a.x-b.x)+square(a.y-b.y));
}

struct compy
{
    bool operator()(const T &s,const T &p)const
    {
        return s.y<p.y;
    }
};
multiset<T,compy>MS;
multiset<T,compy>::iterator I,F;

int main()
{
    int N;
    scanf("%d",&N);

    for(int i=1; i<=N; i++)
        scanf("%d%d",&P[i].x,&P[i].y),P[i].id=i;
    sort(P+1,P+N+1);

    double min_dist=1<<30;
    int s1,s2;
    int p=1;
    for(int i=1; i<=N; i++)
    {
        while(p<i && P[i].x-P[p].x>=min_dist)
        {
            MS.erase(MS.find(P[p]));
            p++;
        }
        I=MS.lower_bound(T(P[i].x,P[i].y-min_dist));
        F=MS.upper_bound(T(P[i].x,P[i].y+min_dist));

        while(I!=F)
        {
            //min_dist=min(min_dist,dist(P[i],*I));
            T x=*I;
            if(min_dist>dist(P[i],x))
            {
                min_dist=dist(P[i],x);
                s1=P[i].id;
                s2=x.id;
            }
            I++;
        }
        MS.insert(P[i]);
    }
    printf("%d %d",s1,s2);
    return 0;
}
```

Convex Hull

```
typedef pair<int,int>par;
par P[10005];
int A[10005],r;

int ABS(int x)
{
   if(x<0)
      return -x;
   return x;
}

int cross(int p1,int p2,int p3)
{
   int m1=(P[p3].second-P[p1].second)*(P[p2].first-P[p1].first);
   int m2=(P[p2].second-P[p1].second)*(P[p3].first-P[p1].first);
   return m1-m2;
}

bool com(const par &s,const par &p)
{
   if(s.first!=p.first)
      return s.first < p.first;

   return s.second<p.second;
}

int main()
{
   int n;
   scanf("%d",&n);
   for(int i=1; i<=n; i++)
      scanf("%d%d",&P[i].first,&P[i].second);
   sort(P+1,P+n+1,com);

   int top=2;
   for(int i=1; i<=n; i++)
   {
      while(r>=top && cross(A[r-1],A[r],i)<=0)
         r--;
      A[++r]=i;
   }
   top=r;
   for(int i=n; i>=1; i--)
   {
      while(r>top && cross(A[r-1],A[r],i)<=0)
         r--;
      A[++r]=i;
   }
   for(int i=1; i<r; i++)
      printf("%d -> %d %d\n",A[i],P[A[i]].first,P[A[i]].second);
   return 0;
}
```

Diametro de un Grafo

```
const int MAXN=1e5+10;
bool mark[MAXN];
typedef pair<int,int>par;
vector<par>ady[MAXN];
int sol;

int diam(int nod)
{
   int max_path=0;
   mark[nod]=1;
   int t=ady[nod].size();
   for(int i=0; i<t; i++)
   {
      int nn=ady[nod][i].first;
      int nc=ady[nod][i].second;
      if(mark[nn])continue;
      int temp=nc+diam(nn);
      sol=max(sol,max_path+temp);
      max_path=max(max_path,temp);
   }

   return max_path;
}

int main()
{
   int N,M;
   scanf("%d%d",&N,&M);

   for(int i=1; i<=M; i++)
   {
      int a,b,c;
      scanf("%d%d%d%s",&a,&b,&c);
      ady[a].push_back(par(b,c));
      ady[b].push_back(par(a,c));
   }
   diam(1);
   cout << sol << '\n';

   return 0;
}
```

DINIC

```
char M[35][35];
int NODO[35][35];
int SOURCE,SINK;
const int mf[]= {0,0,1,-1},
          mc[]= {1,-1,0,0};
const  int maxn = 2000 ;  // number of vertices
const  int INF =  1000000000 ;  // constant-Infinity

struct edge
{
   int a, b, cap, Flow ;
} ;

int n, s, t, d [ maxn ] , ptr [ maxn ] , q [ maxn ] ;
vector <edge>E;
vector <int> G[maxn] ;

void add_edge ( int a, int b, int cap )
{
   edge e1 =  { a, b, cap, 0 } ;
   edge e2 =  { b, a, 0, 0 } ;
   G [ a ] . push_back  ( ( int ) E. size ( ) ) ;
   E. push_back  ( e1 ) ;
   G [ b ] . push_back  ( ( int ) E. size ( ) ) ;
   E. push_back  ( e2 ) ;
}

bool bfs ( )
{
   int QH = 0, Qt = 0 ;
   q [ Qt ++ ]  = s ;
   memset  ( d, -1, sizeof(d) ) ;
   d [ s ]  =  0 ;
   while  ( QH < Qt && d [ t ]  ==  - 1 )
```

```
    {
        int V = q [ QH ++ ] ;
        for  ( size_t I = 0 ; I < G [ V ] . size ( ) ;  ++ I )
        {
            int ID = G [ V ] [ I ] , to = E [ ID ] . b ;
            if  ( d [ to ]  ==  -1 && E [ ID ] . Flow  < E [ ID ] . cap )
            {
                q [ Qt ++ ]  = to ;
                d [ to ]  = d [ V ]  +  1 ;
            }
        }
    }
    return d [ t ]  !=  -1 ;
}

int DFS ( int V, int Flow )
{
    if  ( ! Flow )   return  0 ;
    if  ( V == t )   return Flow ;
    for  ( ; ptr [ V ] < ( int ) G [ V ] . size ( ) ;  ++ ptr [ V ] )
    {
        int ID = G [ V ] [ ptr [ V ] ] , to = E [ ID ]. b ;
        if  ( d [ to ]  != d [ V ]  + 1 )   continue ;
        int pushed = DFS ( to, min ( Flow, E [ ID ]. cap  - E [ ID ]. Flow ) ) ;
        if  ( pushed )
        {
            E [ ID ]. Flow  += pushed ;
            E [ ID ^ 1 ]. Flow  -= pushed ;
            return pushed ;
        }
    }
    return  0 ;
}
```

```
int EN(int X)
{
    return 2*X-1;
}
int SA(int X)
{
    return 2*X;
}

int main()
{
    int N;
    scanf("%d",&N);

    int cont=0,L;
    for(int i=1; i<=N; i++)
    {
        scanf("%s",M[i]+1);
        L=strlen(M[i]+1);
        for(int j=1; j<=L; j++)
            NODO[i][j]=++cont,add_edge(EN(cont),SA(cont),1);
    }

    SINK=t=2*cont+1;
    for(int i=1; i<=N; i++)
        for(int j=1; j<=L; j++)
        {
            for(int k=0; k<4; k++)
            {
                int nf=i+mf[k];
                int nc=j+mc[k];
                int nod=NODO[i][j];
                if(M[i][j]=='1')add_edge(0,EN(nod),1);
            if(i==1 || i==N || j==1 || j==L)add_edge(SA(nod),SINK,1);

            if(nf<1 || nf>N || nc<1 || nc>L)continue;
            int nn=NODO[nf][nc];
            if(M[nf][nc]=='0')add_edge(SA(nod),EN(nn),1);
         }
      }

   int Flow =  0 ;
   for  ( ;; )
   {
      if  ( ! bfs ( ) )   break ;
      memset  ( ptr, 0, sizeof(ptr)) ;
      while  ( int pushed = DFS ( s, INF ) )
         Flow += pushed ;
   }
   printf("%d",Flow);
   return 0;
}
```

Camino o circuito euleriano

```
int n,m;
int a,b,c;
vector<int>ID[1001];
int start;
int G[1001];
stack<int>pila;
struct edge
{
   int nod,nn;
   bool mark;
   edge(int a=0,int b=0,bool c=0)
   {
      nod=a;
      nn=b;
      mark=c;
   }

   int next(int x)
   {
      if(x==nod)
         return nn;

      return nod;
   }

} A[1001];

void euler(int nod)
{
   int t=ID[nod].size();
   for(int i=0; i<t; i++)
   {
      int id=ID[nod][i];
      if(A[id].mark==0)
      {
         A[id].mark=1;
         euler(A[id].next(nod));
      }
   }
   pila.push(nod);
}

int main()
{
   scanf("%d%d",&n,&m);
```

```
for(int i=1; i<=m; i++)
{
   scanf("%d%d",&a,&b);
   ID[a].push_back(i);
   ID[b].push_back(i);
   G[a]++;
   G[b]++;
   A[i]=edge(a,b,0);
}

int I=0;
for(int i=1; i<=n; i++)
{
   if(G[i]%2==1)
      I++;
   if(I>2)
   {
      printf("NO HAY CAMINO EULERIANO\n");
      return 0;
   }
}

scanf("%d",&start);
euler(start);
if(I)
   printf("EXISTE UN CAMINO EULERIANO\n");
else
   printf("EXISTE UN CIRCUITO EULERIANO\n");

for(; !pila.empty();)
{
   printf("%d\n",pila.top());
   pila.pop();
```

```
   }
   return 0;
}
```

Complex FFT O (n * log (n))

```
typedefcomplex<double>Complex;
// phase: 0 for DFT and 1 for the inverse, n must be a power of 2
constComplex I(0, 1);
voidfft(int n, Complex a[], bool phase)
{
   double theta = 2*M_PI / n;
   if(phase) theta *= -1;
   for (int m = n; m >= 2; m >>= 1)
   {
      int mh = m >> 1;
      for (int i = 0; i < mh; i++)
      {
         Complex w = exp(i*theta*I);
         for (int j = i; j < n; j += m)
         {
            int k = j + mh;
            Complex x = a[j] - a[k];
            a[j] += a[k];
            a[k] = w * x;
         }
      }
      theta *= 2;
   }
   for (int j = 1, i=0; j < n - 1; j++)
   {
      for (int k = n >> 1; k >  (i ^= k); k >>= 1);
      if (j < i) swap(a[i], a[j]);
   }
   if(phase)for(int i=0; i<n; i++)a[i]/=n;
}
intmain()
{
   Complex ar[4] = {7, 3, 0, 0};
   int n = 4;
   fft(n, ar, 0);
   for(int i=0; i<n; i++)ar[i]*=ar[i];
   fft(n, ar, 1);
   for(int i=0; i<n; i++)printf("%lf %lf\n", ar[i].real(), ar[i].imag());
}
```

Hashing

```
using namespace std;
const int MAXN = 2e4 + 10;
int N,K ;
int A[MAXN];

typedef unsigned long long ull;
ull h[2][MAXN];
ull bas[2] = {1e9 + 7 , 1e9 + 11};
ull po[2][MAXN];

//hash desde i a f sin incluir f, con el primo u
ull hash_to(int i , int f , int u){
          return h[u][f-1] - h[u][i-1]*po[u][f -i];
}

int main(){
           scanf("%d %d",&N,&K);
          for(int i =1 ; i <= N ;i++)
                    scanf("%d",&A[i]);

          po[0][0] = po[1][0] = 1;
          for(int j = 0 ; j < 2 ;j++)
                    for(int i = 1 ; i <=N ;i++)
                              po[j][i] = po[j][i-1]*bas[j];

          h[1][0] = h[0][0] = 1;
          for(int j = 0 ; j < 2 ;j++)
                    for(int i =1 ; i < N ;i++)
                              h[j][i] = (h[j][i-1]*bas[j]) + A[i];
}
```

Heavy-Light descomposition

```
int n, q;
vector<int> G[MAX], bit[MAX];
int c[MAX], pad[MAX], h[MAX], path[MAX], psize[MAX];
int P[MAX][20];

bool vis[MAX];

void dfs(int v)
{
   c[v] = 1;
   vis[v]=1;
   for(int i = 0; i < (int)G[v].size(); i++)
   {
      int w = G[v][i];
      if(vis[w]) continue;
      pad[w]=v;
      h[w] = h[v] + 1;
      dfs(w);
      c[v] += c[w];
   }
}
void HLD(int v)
{
    vis[v]=1;
    for(int i = 0; i < (int)G[v].size(); i++)
    {
        int w = G[v][i];
        if(vis[w]) continue;
        if(2 * c[w] > c[v])
            path[w] = path[v];
        else
            path[w] = w;
        psize[path[w]]++;
        HLD(w);
    }
}
void process3()
{
    int i, j;
    for(i = 1; i <= n; ++i)
        for(j = 0; 1 << j <= n; ++j) P[i][j] = -1;
    for(i = 2; i <= n; ++i) P[i][0] = pad[i];
    for(j = 1; 1 << j <= n; ++j)
        for(i = 2; i <= n; ++i)
            if(P[i][j-1] != -1)
                P[i][j] = P[P[i][j-1]][j-1];
}

int lca_HLD(int p, int q)
{
    int i,log;
    if(h[p] < h[q]) swap(p,q);
    for(log = 1; 1 << log <= h[p]; ++log);
    log--;
    for(i = log; i >= 0; --i)
        if(h[p]-(1<<i) >= h[q]) p = P[p][i];
    if(p==q) return p;
    for(i = log; i >= 0; --i)
        if(P[p][i] != -1 && P[p][i] != P[q][i])
            p = P[p][i], q = P[q][i];
    return pad[p];
}

struct segment_tree
{
    struct node
    {
        int sum, d;
        int b, e;
    };
    vector<node> M;
    int n, N;
    segment_tree(int nx)
    {
        n = nx;
        N = 1 << (33 - __builtin_clz(n - 1));
        M.resize(N);
        for(int i(0); i < N; i++)
            M[i].sum = M[i].d = 0;
        for(int i(N / 2); i < N; i++)
            M[i].b = M[i].e = i - N / 2;
        for(int i(N / 2 - 1); i >= 0; i--)
            M[i].b = M[2 * i].b, M[i].e = M[2 * i + 1].e;
    }
    inline void update_lazily(int d, int nod)
    {
        M[nod].d+=d;
        M[nod].sum+=d*(M[nod].e-M[nod].b +1);
    }
    void lazy_stuff(int nod)
    {
        update_lazily(M[nod].d, 2*nod);
        update_lazily(M[nod].d, 2*nod+1);
        M[nod].d=0;
    }
    int query(int left, int right, int nod = 1)
    {
        if(left > M[nod].e || right < M[nod].b) return 0;
        if(M[nod].b >= left && M[nod].e <= right) return M[nod].sum;
        lazy_stuff(nod);
        int p1=query(left, right, 2 * nod);
        int p2=query(left, right, 2 * nod + 1);
        return p1 + p2;
    }
    void update(int left, int right, int d, int nod = 1)
    {
        if(left > M[nod].e || right < M[nod].b) return;
        if(M[nod].b >= left && M[nod].e <= right)
        {
            update_lazily(d, nod);
            return;
        }
        lazy_stuff(nod);
        update(left, right, d, 2 * nod);
        update(left, right, d, 2 * nod + 1);
        M[nod].sum = M[2*nod].sum + M[2*nod+1].sum;
    }
};

segment_tree *T[MAX];

int query_path(int v)
{
    int sum = 0, p, pos;
    while(v)
    {
        p = path[v], pos = h[v] - h[p];
        sum += T[p]->query(0,pos);
        v = pad[p];
    }
    return sum;
}

void update_path(int v,int val)
{
    int p, pos;
    while(v)
    {
        p = path[v], pos = h[v] - h[p];
        T[p]->update(0,pos,val);
        v = pad[p];
    }
}

char buff[50];
int main()
{
    scanf("%d %d",&n,&q);
    int u,v;
    for(int i = 1; i < n; i++)
    {
        scanf("%d %d",&u,&v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(1);
    process3();
    path[1] = 1, psize[1] = 1, pad[1]=0;
    for(int i=0; i<=n; i++) vis[i]=0;
    HLD(1);
    for(int i = 1; i < n + 1; i++)
        T[i]= new segment_tree(psize[i]+1);
    while(q--)
    {
        scanf("%s",buff);
        if(buff[0] == 'P')
        {
            scanf("%d %d",&u,&v);
            int la=lca_HLD(u,v);
            update_path(u,1);
            update_path(v,1);
            update_path(la,-2);
        }
        else
        {
            scanf("%d %d",&u,&v);
            int la=lca_HLD(u,v);
            int res=query_path(v) + query_path(u) - 2*query_path(la);
            printf("%d\n",res);
        }
    }
}
```

HUNGARIAN

```
int N,A[MAXN+1][MAXN+1],p,q, oo;
int fx[MAXN+1],fy[MAXN+1],x[MAXN+1],y[MAXN+1];
int hng(int oo)
{
memset(fx,0,sizeof(fx));
memset(fy,0,sizeof(fy));
memset(x,-1,sizeof(x));
memset(y,-1,sizeof(y));
for(int i = 0; i < N; ++i)
    for(int j = 0; j < N; ++j) fx[i] = max(fx[i],A[i][j]);
for(int i = 0; i < N; )
{
    vector<int> t(N,-1), s(N+1,i);
    for(p = q = 0; p <= q && x[i]<0; ++p)
        for(int k = s[p], j = 0; j < N && x[i]<0; ++j)
            if (fx[k]+fy[j]==A[k][j] && t[j]<0)
            {
                s[++q]=y[j];
                t[j]=k;
                if(s[q]<0)
                    for(p=j; p>=0; j=p)
                        y[j]=k=t[j], p=x[k], x[k]=j;
            }
    if (x[i]<0)
    {
        int d = oo;
        for(int k = 0; k < q+1; ++k)
            for(int j = 0; j < N; ++j)
                if(t[j]<0) d=min(d,fx[s[k]]+fy[j]-A[s[k]][j]);
        for(int j = 0; j < N; ++j) fy[j]+=(t[j]<0?0:d);
        for(int k = 0; k < q+1; ++k) fx[s[k]]-=d;
    }
    else ++i;
}
int ret = 0;
for(int i = 0; i < N; ++i) ret += A[i][x[i]];
return ret;
}
```

KMP

```
char TEXT[500005],PATT[500005];
int F[500005];

int main()
{
   int i = 0, j = -1;
   b[0] = -1; // starting values
   while (i < m)   // pre-process the pattern string P
   {
      while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b
      i++;
      j++; // if same, advance both pointers
      b[i] = j;
   }

   int i = 0, j = 0; // starting values
   while (i < n)   // search through string T
   {
      while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b
      i++;
      j++; // if same, advance both pointers
      if (j == m)   // a match found when j == m
      {
         printf("P is found at index %d in T\n", i - j);
         j = b[j]; // prepare j for the next possible match
      }
   }

   return 0;
}
```

K-th element

```
int A[1000005];
int partition(int l,int F)
{
   int piv=A[l];
   int p=l-1,q=F+1;
   for(;;)
   {
      p++;
      while(A[p]<piv)p++;
      q--;
      while(A[q]>piv)q--;
      if(p<q)
         swap(A[p],A[q]);
      else
         return q;
   }
}

int Kth_element(int l,int F,int K)
{
   if(l==F)
      return A[l];
   int piv=partition(l,F);
   if(piv-l+1==K)
      return A[piv];

   if(piv-l+1>K)
      Kth_element(l,piv-1,K);
   else
      Kth_element(piv+1,F,K-piv);
}
int main()
```

```
{
   int N,K;
   scanf("%d%d",&N,&K);

   for(int i=1; i<=N; i++)
      scanf("%d",&A[i]);

   printf("%d",Kth_element(1,N,K));
   return 0;
}
```

Longest Square(Tandems)

```
voidbfail(char *l,int n,char *r,int m)  //fail[i] guarda el mayor sufijo de l, que
es sufijo para la posición i en r
{
   int it=0;
   for(int i=n-1; i>=0; i--) temp[it++]=l[i]; //invierte las dos cadenas y las
concatena
   for(int i=m-1; i>=0; i--) temp[it++]=r[i];
   Zfunction(temp,it);
   for(int i=0; i<m; i++) fail[i]=min(z[m+n-i-1],n);
}

voidsqfind(char *s1,int l1,char *s2,int l2)  //encuentra los cuadrados de la
concatenación de l2 y
{
   bfail(s1,l1,s2,l2);                                        // l1  centrados en l2
o entre las dos cadenas, que abarcan a l1
   Zfunction(s2,l2);
   for(int i=l2-1; i>resz; i--)
      if(z[i]+fail[i-1]>=i) //implica que hay un cuadrado centrado entre i e i-1
         resz=max(resz,i);
   for(int i=l2-1; i>=resz; i--)
      if(fail[i]>=i+1)         //implica que hay un cuadrado entre l1 y l2;
            resz=max(resz,i+1);
}
voidlsquare(char *txt,int len)
{
   if(len==1) return;
   if(len==2)
   {
      resz=max(resz,int(txt[0]==txt[1]));
      return;
   }
   int n=len/2,m=len-len/2;
   char *s1=txt,*s2=txt+n;
   lsquare(s1,n);
   lsquare(s2,m);
   sqfind(s1,n,s2,m);
   reverse(s1,s1+n);
   reverse(s2,s2+m);
   sqfind(s2,m,s1,n);
   reverse(s1,s1+n);
   reverse(s2,s2+m);
}
```

Largest zero submatriz

```
#define MAXN 5005
int N,M;
int D[MAXN];
int A[MAXN][MAXN];

int max_submatr()
{
   int h[MAXN], s[MAXN], ptr = 0;
   int ret = 0;
   for(int i=0; i<M; i++)
   {
        int l=i;
        while(ptr>0 && D[i]<h[ptr-1])
        {
            ret=max(ret,(i-s[ptr-1])*(h[ptr-1]));
            l=s[ptr-1];
            ptr--;
        }
        h[ptr]=D[i];
        s[ptr++]=l;
    }
    while(ptr>0)
    {
        ret=max(ret,(M-s[ptr-1])*(h[ptr-1]));
        ptr--;
    }

    return ret;
}

int main()
{
    scanf("%d%d",&N,&M);

    for(int i=0; i<N; i++)
        for(int j=0; j<M; j++)
            scanf("%d",&A[i][j]);

    int sol=0;
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<M; j++)
            if(!A[i][j])
                D[j]++;
            else
                D[j]=0;
        sol=max(sol,max_submatr());
    }

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

    return 0;
}
```

LIS

```
set<int>S;
set<int>::iterator it;
int main()
{
    int N;
    scanf("%d",&N);

    for(int i=1; i<=N; i++)
    {
        int a;
        scanf("%d",&a);
        S.insert(a);
        it=S.find(a);
        it++;
        if(it!=S.end())
            S.erase(it);
    }

    printf("%d\n",S.size());

    return 0;
}
```

MANACHER

```
char s[100005];
int r[100005];

int main()
{

   scanf("%s",s);
   int n=strlen(s);

   int i,j,k=0;
   for(i=0,j=0; i<2*n; i+=k,j=max(j-k,0))
   {
      while(i-j>=0 && i+j+1<2*n && s[(i-j)/2]==s[(i+j+1)/2])
         ++j;
      r[i]=j;
      for(k=1; i>=k && r[i]>=k && r[i-k]!=r[i]-k; ++k)
         r[i+k] = min(r[i-k],r[i]-k);
   }

   for(int i=0; i<2*n; i++)
      printf("%d ",r[i]);

   return 0;
}
//posiciones pares->palindromes de tamaño impar
```

Exponenciación de Matrices

```
class matriz
{
   int CF,CC;
   int **M;

public:
matriz(int f,int c)
{
   CF=f;
   CC=c;
   M=new int *[f];
   for(int i=0; i<f; i++)
      M[i]=new int[c];
}
matriz(int f,int c,int **C)
{
   CF=f;
   CC=c;
   M=new int *[f];
   for(int i=0; i<f; i++)
      M[i]=new int[c];
   for(int i=0; i<f; i++)
      for(int j=0; j<c; j++)
         M[i][j]=C[i][j];
}
matriz operator *(const matriz &);
int getF()const
{
   return CF;
}
int getC()const
{
   return CC;
}
int** getM()const
{
   return M;
}
friend matriz POT(matriz,int);
};

matriz matriz::operator*(const matriz &X)
{
  matriz SOL(X.getF(),X.getC());

  for(int i=0; i<CF; i++)
    for(int j=0; j<CC; j++)
    {
      SOL.M[i][j]=0;
      for(int k=0; k<CF; k++)
        SOL.M[i][j]=(SOL.M[i][j]+M[i][k]*X.M[k][j])%10007;
    }
  return SOL;
}
matriz square(matriz X)
{
  return X*X;
}
matriz POT(matriz X,int K)
{
  if(K==1)
    return X;

  if(K%2==0)
    return square(POT(X,K/2));

  return X*POT(X,K-1);
}

int main()
{
  int **A;
  A=new int *[3];
  for(int i=0; i<3; i++)
    A[i]=new int[3];
  A[0][0]=A[0][1]=A[2][0]=A[1][1]=0;
  A[1][0]=A[2][1]=1;
  A[0][2]=A[1][2]=A[2][2]=2;

  matriz X(3,3,A);
  int K;
  while(cin >> K)
  {
    if(!K)return 0;
    if(K>3)
    {
      matriz Z=POT(X,K-3);
      long long sol=0;

      sol=(sol+3*Z.getM()[0][2])%10007;
      sol=(sol+9*Z.getM()[1][2])%10007;
      sol=(sol+26*Z.getM()[2][2])%10007;
      printf("%lld\n",sol);
    }
    else
    {
      if(K==1)printf("3\n");
      if(K==2)printf("9\n");
      if(K==3)printf("26\n");
    }
  }
  return 0;
}
```

Maximun Matching

```
int parent[1005];
int N,M;
bool mark[1005];
bool G[1005][1005];

bool dfs(int nod)
{
   if(mark[nod])
      return 0;
   mark[nod]=1;
   for(int i=N+1; i<=2*N; i++)
      if(G[nod][i] && (parent[i]==0 || dfs(parent[i])))
      {
         parent[i]=nod;
         return 1;
      }
   return 0;
}

int main()
{
   scanf("%d%d",&N,&M);

   for(int i=1; i<=M; i++)
   {
      int a,b;
      scanf("%d%d",&a,&b);
      b+=N;
      G[a][b]=1;
   }

   for(int i=1; i<=N; i++)
      G[0][i]=1,G[i+N][2*N+1]=1;
   int SOL=0;
   for(int i=1; i<=N; i++)
   {
      memset(mark,0,sizeof(mark));
      if(dfs(i))
         SOL++;
   }
   printf("%d",SOL);
   return 0;
}
```

Period

```
char S[1000005];
bool B;
int main()
{
   int N;
   scanf("%d",&N);
   scanf("%s",S+1);

   int l=1;
   for(int i=2; i<=N; i++)
   {
      if(S[i]==S[i-l])
      {
         if(i%l==0)
            printf("%d %d\n",i,i/l),B=1;
      }
      else
      {
         if(S[i]==S[1])
            l=i-1;
         else
            l=i;
```

```
        }
    }
    if(!B)
        printf("0");
    return 0;
}
```

Persistent_Segment_Tree

```
#define MAXN 500005

int sum[3000005], L[3000005], R[3000005];
int root[MAXN];
int A[MAXN],aux[MAXN];
int sz = 1;

int newnode(int s = 0)
{
    sum[sz] = s;
    return sz++;
}

int build(int I, int F)
{
    if(I == F)
        return newnode();
    int piv=(I+F)/2;
    int nod = newnode();
    L[nod] = build(I, piv);
    R[nod] = build(piv+1, F);
    return nod;
}

int update(int nod, int I, int F, int pos)
{
    if(I==F)
        return newnode(sum[nod]+1);

    int piv=(I+F)/2;

    int nnod = newnode();
    if(pos<=piv)
    {
        L[nnod] = update(L[nod],I,piv,pos);
        R[nnod] = R[nod];
    }
    else
    {
        R[nnod] = update(R[nod],piv+1,F, pos);
        L[nnod] = L[nod];
    }

    sum[nnod] = sum[L[nnod]] + sum[R[nnod]];

    return nnod;
}
int query(int nod1,int nod2,int I,int F,int k)
{
    if(I==F)
        return I;
    int suma = sum[L[nod2]] - sum[L[nod1]];
    int piv=(I+F)/2;

    if(suma >= k)
        return query(L[nod1], L[nod2],I,piv,k);
```

```
   else
      return query(R[nod1], R[nod2],piv+1,F, k-suma);
}

int main()
{
   int N, M;
   cin >> N >> M;

   root[0]=build(1, N);

   for(int i=0; i<N; i++)
   {
      cin >> A[i];
      aux[i]=A[i];
   }

   sort(aux, aux+N);
   for(int i=0; i<N; i++)
      A[i]=lower_bound(aux,aux+N,A[i])-aux;

   for(int i=0; i<N; i++)
      root[i+1] = update(root[i],1,N, A[i]+1);

   for(int i=1; i<=M; i++)
   {
      int a,b,k;
      cin >> a >> b >> k;
      cout << aux[query(root[a-1], root[b], 1, N, k)-1] << '\n';
   }

   return 0;
}
```

POSTFIJA

```
char S[1005];
int V[256];
stack<char>pila;
char SOL[10005];
int r;
double VALOR[1005];

int main()
{
   V['+']=V['-']=1;
   V['*']=V['/']=2;
   V['^']=3;

   scanf("%s",S);
   int l=strlen(S);
   for(int i=0; i<l; i++)
   {
      if(S[i]=='(')pila.push(S[i]);
      if(S[i]>='a' && S[i]<='z')SOL[++r]=S[i];
      if(S[i]==')')
      {
         for(; pila.top()!='('; pila.pop())SOL[++r]=pila.top();
         pila.pop();
      }
      if(V[S[i]])
      {

         while(!pila.empty())
            if(pila.top()!='(' && V[S[i]]<=V[pila.top()])
               SOL[++r]=pila.top(),pila.pop();
            else
               break;
```

```
    pila.push(S[i]);
  }

}
while(!pila.empty())SOL[++r]=pila.top(),pila.pop();

char c;
double a;
while(scanf("%c=%lf",&c,&a)!=EOF)
  VALOR[c]=a;

stack<double>P;
for(int i=1; i<=r; i++)
{
  printf("%c",SOL[i]);
  if(SOL[i]>='a' && SOL[i]<='z')
    P.push(VALOR[SOL[i]]);
  else
  {
    double v1,v2;
    if(SOL[i]=='+')
      v1=P.top(),P.pop(),v2=P.top(),P.pop(),P.push(v1+v2);
    if(SOL[i]=='-')
      v1=P.top(),P.pop(),v2=P.top(),P.pop(),P.push(v2-v1);
    if(SOL[i]=='*')
      v1=P.top(),P.pop(),v2=P.top(),P.pop(),P.push(v1*v2);
    if(SOL[i]=='/')
      v1=P.top(),P.pop(),v2=P.top(),P.pop(),P.push(v2/v1);
    if(SOL[i]=='^')
      v1=P.top(),P.pop(),v2=P.top(),P.pop(),P.push(pow(v2,v1));
  }
}
  printf("\n%.2lf",P.top());
  return 0;
}
```

RMQ

```
int A[10005],M[10005][20];

int main()
{
  int N;
  scanf("%d",&N);

  for(int i=0; i<N; i++)
    scanf("%d",&A[i]),M[i][0]=i;

  for(int i=1; (1<<i)-1<N; i++)
    for(int j=0; j+(1<<i)-1<N; j++)
      if(A[M[j][i-1]]<A[M[j+(1<<(i-1))][i-1]])
        M[j][i]=M[j][i-1];
      else
        M[j][i]=M[j+(1<<(i-1))][i-1];

  int Q;
  scanf("%d",&Q);

  int a,b;
  for(int i=1; i<=Q; i++)
  {
    scanf("%d%d",&a,&b);
    a--;
    b--;
    if(a>b)swap(a,b);
    int lg=(int)log2(b-a+1);
    int sol=min(A[M[a][lg]],A[M[b-(1<<lg)+1][lg]]);
```

```
        printf("%d\n",sol);
    }

    return 0;
}
```

Segment Tree

```
struct STREE
{
    int V;
    bool B;
} ST[3000005];
void build(int nod,int I,int F)
{
    if(I==F)   ST[nod].V=0,ST[nod].B=0;
    else
    {
        int piv=(I+F)/2;
        build(2*nod,I,piv);
        build(2*nod+1,piv+1,F);
        ST[nod].V=0,ST[nod].B=0;
    }
}
void lazy(int nod,int I,int F)
{
    ST[nod].B=0;
    if(I==F)return;
    ST[2*nod].B^=1;
    ST[2*nod+1].B^=1;
    int piv=(I+F)/2;
    ST[2*nod].V=(piv-I+1)-ST[2*nod].V;
    ST[2*nod+1].V=(F-piv)-ST[2*nod+1].V;
}
void update(int nod,int I,int F,int A,int B)
{
    if(ST[nod].B)   lazy(nod,I,F);
    if(I>=A && F<=B)
    {
        ST[nod].V=(F-I+1)-ST[nod].V;
        ST[nod].B=1;
        return;
    }
    if(F<A || I>B)return;
    int piv=(I+F)/2;
    update(2*nod,I,piv,A,B);
    update(2*nod+1,piv+1,F,A,B);
    ST[nod].V=ST[2*nod].V+ST[2*nod+1].V;
}
int query(int nod,int I,int F,int A,int B)
{
    if(ST[nod].B)   lazy(nod,I,F);
    if(F<A || I>B)   return 0;
    if(I>=A && F<=B)   return ST[nod].V;
    int piv=(I+F)/2;
    int p1=query(2*nod,I,piv,A,B);
    int p2=query(2*nod+1,piv+1,F,A,B);
    ST[nod].V=ST[2*nod].V+ST[2*nod+1].V;
    return p1+p2;
}
```

SUFFIX_ARRAY(N log^2 N)

```
struct T
{
    int nr[2],p;
} L[200005];
```

```
bool com(const T &s,const T &p)
{
   if(s.nr[0]!=p.nr[0])
      return s.nr[0]<p.nr[0];
   return s.nr[1]<p.nr[1];
}

int N,K,stp,delta;
char st[200005];
int P[20][200005];
int pos[200005];

int LCP(int x,int y)
{
   int ret=0;
   for(int k=stp-1; k>=0 && x<N && y<N; k--)
      if (P[k][x]==P[k][y])
      {
         x+=(1<<k);
         y+=(1<<k);
         ret+=(1<<k);
      }
   return ret;
}

int main ()
{
   gets( st );
   N = strlen( st );

   /*copy( st, st + N , st + N );
   reverse( st + N, st + 2 * N );
   N *= 2;*/
```

```
/* Suffix Array Computation */
for(int i=0; i<N; i++)
   P[0][i]=st[i]-'A';

/* build suffix array */
for(stp=1,delta=1; (delta>>1) < N; stp++,delta<<=1)
{
   for(int i=0; i<N; i++)
   {
      L[i].nr[0]=P[stp - 1][i];
      L[i].p = i;
      if(i+delta<N)
         L[i].nr[1]=P[stp-1][i+delta];
      else
         L[i].nr[1]=-1;
   }
   sort(L,L+N,com);

   for(int i=0; i<N; i++)
      if(i>0 && L[i].nr[0] == L[i - 1].nr[0] && L[i].nr[1] == L[i - 1].nr[1] )
         P[stp][L[i].p]=P[stp][L[i - 1].p];
      else
         P[stp][L[i].p]=i;
}

/* pos gives me the position of suffix with order at P[stp - 1][i] */

for(int i=0; i<N; i++)
   pos[P[stp - 1][i]]=i;

for(int i=0; i<N; i++)
   printf("%d %s\n",pos[i],st+pos[i]);
```

```
    /*Computing the LCP ( Longest Comon Prefix ) between 2 suffixes, one
starting at
    a, and the other starting at b ( a & b are provided by queries ) */

    /*int solution = 1;
    for (int i = 0 ; i < ( N / 2 ) - 1 ; i++ ) {
        // odd & even length
        if ( i ) // n - i < n
            solution =max( 2 * LCP( i + 1, N - i ) + 1, solution);

        solution = max( 2 * LCP( i + 1, N - i - 1), solution );
    }*/

//printf("%d",solution);

// $ < # < @
// LCP 3 suffixes
    return 0;
}
```

Suffix Array(N log N)

```
#define ll long long
#define MAX 500005

char s[MAX];
int SA[MAX],wa[MAX], wb[MAX], we[MAX], wv[MAX],S[MAX],A[MAX];

void Sufix_Array(char *cad,int *SA,int N)
{
    N++;
    int i, j, p, *x = wa, *y = wb, range = 256;
    memset(we, 0, range * sizeof(int));
    for (i = 0; i < N; i++)
        we[ x[i] = cad[i] ]++;
    for (i = 1; i < range; i++) we[i] += we[ i-1 ];
    for (i = N - 1; i >= 0; i--)
        SA[ --we[ x[i] ] ] = i;
    for (j = p = 1; p < N; j <<= 1, range = p)
    {
        for (p = 0, i = N - j; i < N; y[p++] = i , 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] ];
        memset(we, 0, range * sizeof(int));
        for (i = 0; i< N; i++)
            we[ wv[i] ]++;
        for (i = 1; i < range; i++) we[i] += we[i-1];
        for (i = N-1; i >= 0; i--) SA[--we[wv[i]]] = y[i];
        swap(x, y);
        x[SA[0]] = 0;
        for (p = i = 1; i < N; i++)
            if(y[SA[i]] == y[SA[i-1]] && y[SA[i]+j] == y[SA[i-1]+j])
                x[SA[i]] = p - 1;
            else
                x[SA[i]] = p++;
    }
    N--;
}

int rank[MAX], LCP [MAX];
void FindLCP(char *cad, int *SA, int N)
{
    int i, j, k;
    for (i = 1; i <= N; i++)
```

```
      rank[ SA[i] ] = i;
   for (k = i = 0; i < N; LCP [rank[i++]] = k)
      for (k ? k-- : 0,j = SA[rank[i]-1]; cad[i + k] == cad[j + k];
            k++);
}

char cad[MAX];
int n;

int main()
{

   scanf("%s", cad);
   n = strlen(cad);
   Sufix_Array(cad, SA, n);
   FindLCP(cad, SA, n);

   for(int i=1; i<=n; i++)
      printf("%d %s\n",SA[i],cad+SA[i]);

   return 0;
}
```

TRIE

```
int tree[1000005][256];
int pasan[1000005];
int terminan[1000005];
char cad[10005];
int A[100005];

int main()
{
   int n,m;
   scanf("%d%d",&n,&m);
for(int j = 0; j <= 255; ++j)
   tree[0][j]=-1;
int nodos=0;
int t;
for(int i=1; i<=n; i++)
{
   scanf("%d",&t);
   int p = 0;
   for(int j=0; j<t; j++)
   {
      int c;
      scanf("%d",&c);
      if(tree[p][c]==-1)
      {
         tree[p][c]=++nodos;
         for(int k = 0; k <= 255; ++k)
            tree[nodos][k]=-1;
      }
      p = tree[p][c];
      pasan[p]++;
   }
   pasan[p]--;
   terminan[p]++;
}

for(int i=1; i<=m; i++)
{
   int p=0;
   int t;
   scanf("%d",&t);
   bool B=1;
   int SOL=0;
```

```
      int c;
      for(int j=0; j<t; j++)
         scanf("%d",&A[j]);
      for(int j=0; j<t; j++)
      {
         c=A[j];
         if(tree[p][c]==-1)
         {
            B=0;
            break;
         }
         p=tree[p][c];
         SOL+=terminan[p];
      }
      if(B==1)
         SOL+=pasan[p];

      printf("%d\n",SOL);
   }
   return 0;
}
```

Prefix and Z function

```
string s;
int z[100005];

int main()
{
   cin >> s;
   int n =  ( int ) s.length ( ) ;
   vector < int > pi ( n ) ;
   for  ( int I = 1 ; I < n ;  ++ I )
   {
      int j = pi [ I - 1 ] ;
      while  ( j > 0 && s [ I ]  != s [ j ] )
         j = pi [ j - 1 ] ;
      if  ( s [ I ]  == s [ j ] )   ++ j ;
      pi [ I ]  = j ;
   }
//cantidad de veces que aparece el prefijo de tamaño
//i en la cadena
   vector < int > ans ( n + 1 ) ;
   for  ( int I = 0 ; I < n ;  ++ I )
      ++ ans [ pi [ I ] ] ;
   for  ( int I = n - 1 ; I > 0 ;  -- I )
      ans [ pi [ I - 1 ] ]  += ans [ I ] ;

   for(int i=1; i<n; i++)
      printf("%d ",ans[i]+1);

//Given a string S of length n,
//the Z Algorithm produces an array Z
//where Z[i] is the length of the longest
//substring starting from S[i] which is also a prefix of S
   int L = 0, R = 0;
   for (int i = 1; i < n; i++)
   {
      if (i > R)
      {
         L = R = i;
         while (R < n && s[R-L] == s[R]) R++;
         z[i] = R-L;
         R--;
      }
      else
      {
         int k = i-L;
```

```
            if (z[k] < R-i+1) z[i] = z[k];
            else
            {
                L = i;
                while (R < n && s[R-L] == s[R]) R++;
                z[i] = R-L;
                R--;
            }
        }
    }
    cout << '\n';
    for(int i=0; i<n; i++)
        printf("%d ",z[i]);

    return 0;
}
```