```
Ellipsoids
                                                                      for(int i=0; i<3; i++)
int X[100],Y[100],Z[100],R[100];
                                                                         Norm(w[i]);
int n;
                                                                       double N=3.0;
                                                                       C=C+W[0]*(1./(N+1.));
const double eps=1e-4;
                                                                       v[0]=w[0]*((N+0.0)/(N+1.0));
const int BUBEN=100000;
                                                                      for(int i=1; i<3; i++)
                                                                         v[i]=w[i]*((N+0.0)/sqrt(N*N-1.0));
P v[3];
P w[3];
                                                                    printf("NO\n");
Pc;
                                                                    return 0;
void Norm(P& p)
                                                                  Euler paths
{
                                                                  // Copyright 2007 x13.
  double d[3];
  double s=0.0;
                                                                  // All rights reserved.
  for(int i=0; i<3; i++)
                                                                  // Havka - papstvo!
                                                                  // Porvite tam vseh!
     d[i]=(p&v[i])/(v[i]&v[i]);
     s+=d[i]*d[i];
                                                                  //Eulerian circuit in directed graph
                                                                  void Euler (int a)
  double koef=1./sqrt(s);
                                                                  {
  p=p*koef;
                                                                           for (int b = 0; b < N; b++) if (G[a][b])
                                                                                    G[a][b] = false;
int main()
                                                                                    Euler(b);
                                                                           Path.push_back(a);
  int n;
                                                                 } // and reverse(Path.begin(), Path.end()) after that
  cin >> n;
  for(int i=0;i< n;i++)
    cin >> X[i] >> Y[i] >> Z[i] >> R[i];
                                                                  FFT
                                                                  int rev(int x) {
  c=P(0,0,0);
  v[0]=P(100000,0,0);
                                                                           int res=0;
  v[1]=P(0,100000,0);
                                                                           for (int i=0; i<S; i++)
  v[2]=P(0,0,100000);
                                                                                    if (x&(1<<i))
  for(int ic=0;ic<BUBEN;ic++)</pre>
                                                                                             res|=1<<(S-i-1);
                                                                           return res;
  {
     int ind=-1;
                                                                 }
     for(int i=0;i< n;i++)
                                                                  Poly FFT(Poly a, int to=1) {
       if((c-P(X[i],Y[i],Z[i])).len()>R[i]+eps)
                                                                           Poly A;
                                                                           for (int i=0; i<n; i++)
         ind=i;
                                                                                    A[rev(i)]=a[i];
         break;
                                                                           for (int s=1; s<=S; s++) {
                                                                                    int m=1<<s:
       }
                                                                  wm=C(cos(to*2*pi/m),sin(to*2*pi/m));
     if(ind==-1)
                                                                                    for (int k=0; k<n; k+=m) {
       printf("YES\n");
                                                                                             C w=C(1,0);
       return 0;
                                                                                              for (int j=0; j< m/2; j++) {
                                                                                                       C u=A[k+j];
    if(v[0].len()<eps | | v[1].len()<eps | |
                                                                                                       С
v[2].len()<eps) break;
                                                                  t=w^*A[k+j+m/2];
    P t=(c-P(X[ind],Y[ind],Z[ind]));
                                                                                                       A[k+j]=u+t;
     w[0]=t;
                                                                                                       A[k+j+m/2]=u-t;
     if(fabs(t.z)<eps)
                                                                                                       w*=wm:
       w[1]=P(t.y,-t.x,0.0);
                                                                                             }
                                                                                    }
     else
       w[1]=P(t.z,0.0,-t.x);
     w[2]=w[0]*w[1];
                                                                           if (to==-1)
```

| for (int i=0; i <n; i++)<br="">A[i]/=n;</n;> | while qh <qt do<br="">begin</qt> |
|--|---|
| return A; | x:=get; |
| } | if I[x]=I[0] then |
| , | break; |
| Hopcraft-Karp | for j:=1 to m do |
| {\$I-,Q-,R-,S-} | if g[x,j] and not f[b[j]] then |
| var n,m,k,i,x,y,cnt,qh,qt,j:longint; | put(b[j],l[x]+1); |
| | pat(bjj];i[x]+1); end; |
| g:array [11000,11000] of boolean; | • |
| a,b,l,q:array [01000] of longint; | if not f[0] then |
| f:array [01000] of boolean; | break; |
| function dfs(u:longint):boolean; | for i:=1 to n do |
| var j:longint; | if (a[i]=0) and dfs(i) then |
| begin | inc(cnt); |
| dfs:=true; | until false; |
| if u=0 then | writeIn(cnt); |
| exit; | end. |
| f[u]:=false; | |
| for j:=1 to m do | <u>KMP</u> |
| if g[u,j] and f[b[j]] and (l[b[j]]=l[u]+1) and dfs(b[j]) | void KMP(const string &s, vector< int > &Len) |
| then | { |
| begin | Len.resize(s.length()); |
| b[j]:=u; | Len[0] = 0; |
| a[u]:=j; | for (int i = 1; i < s.length(); i++) |
| exit; | { |
| end; | int len = Len[i - 1]; |
| dfs:=false; | while (len > 0 && s[i] != s[len]) |
| end; | len = Len[len - 1]; |
| procedure put(x,d:longint); | if (s[i] == s[len]) |
| begin | len++; |
| f[x]:=true; | Len[i] = len; |
| I[x]:=d; | } |
| q[qt]:=x; | } |
| inc(qt); | 7.6 |
| end; | Z-function |
| function get:longint; | vi Z(string s) { |
| begin | vi z(s.size()); |
| get:=q[qh]; | int j=0; |
| inc(qh); | for (int i=1; i <s.size(); i++)="" td="" {<=""></s.size();> |
| end; | int r=min(j+z[j]-1,i+z[i-j]-1); |
| begin | if (r <i) else="" z[i]="r-i+1;</td"></i)> |
| read(n,m,k); | while (i+z[i] <s.size() &&<="" td=""></s.size()> |
| fillchar(g,sizeof(g),false); | S[i+Z[i]] == S[Z[i]]) |
| for i:=1 to k do | Z[i]++; if (i, z[i]> i, z[i]) |
| begin | if (i+z[i]>j+z[j]) : :. |
| read(x,y); | j=i; |
| g[x,y]:=true; | } |
| end; fillchar(a,sizeof(a),0); | return z; |
| • | } |
| fillchar(b,sizeof(b),0); | Preflow-push |
| cnt:=0; | class Network { |
| repeat | |
| qh:=0; | public: |
| qt:=0; fillchar(!sizoof(!) 62); | struct Arc (|
| fillchar(l,sizeof(l),63); fillchar(f,sizeof(f),0); | struct Arc { |
| for i:=1 to n do | int u; |
| if a[i]=0 then | int u, |
| put(i,0); | int c; |
| PACTION | HILO, |

```
int f:
                                                                                     }
                  Arc *rev;
                                                                            delete[] edges;
                  Arc(int u, int v, int c): u(u), v(v),
                                                                            delete[] d;
c(c), f(0), rev(0) {
                                                                            delete[] exc;
                                                                            delete[] curArc;
         };
                                                                            delete[] used;
                                                                  }
         Network(int n);
         ~Network();
                                                                  void Network::addEdge(int u, int v, int c) {
                                                                            assert(0 \le u \&\& u < n);
         void addEdge(int u, int v, int c);
                                                                            assert(0 \le v \&\& v \le n);
         int getMaximumFlow(int s, int t);
                                                                            assert(u != v);
         std::vector<int> getMinimumCut(int s, int t);
                                                                            assert(c >= 0);
                                                                            Arc *a = \text{new Arc}(u, v, c);
         int getMinimumCut();
                                                                            Arc *b = new Arc(v, u, c);
                                                                            a \rightarrow rev = b;
private:
                                                                            b \rightarrow rev = a:
         static const double RATIO = 2.0;
                                                                            edges[u].push_back(a);
                                                                            edges[v].push_back(b);
         Network(const Network&);
                                                                            m += 2:
         Network& operator=(const Network&);
                                                                  void checkArray(std::vector<int> &a) {
         int n. m:
                                                                            int old = a.size();
         std::vector<Arc*> *edges;
                                                                            std::sort(a.begin(), a.end());
                                                                            a.erase(std::unique(a.begin(), a.end()),
         int *d:
         int *exc;
                                                                  a.end());
         int *curArc;
                                                                            assert(a.size() == old);
         bool *used;
                                                                  }
         int work:
                                                                  int Network::push(Arc *a) {
                                                                            work++:
         std::multimap<int, int> active;
                                                                            int delta = std::min(a->c - a->f, exc[a->u]);
                                                                            exc[a->u] -= delta;
         bool isAdmissible(Arc *a);
                                                                            exc[a->v] += delta;
         int push(Arc *a);
                                                                            a->f += delta;
         void relabel(int v);
                                                                            a->rev->f -= delta:
         void discharge(int v);
                                                                            return delta;
         void dfs(int v);
         void globalRelabeling(int s, int t);
};
                                                                  void Network::relabel(int v) {
                                                                            d[v] = 2 * n;
Network::Network(int n): n(n) {
                                                                            for (std::vector<Arc*>::iterator it =
                                                                  edges[v].begin(); it != edges[v].end(); it++) {
         assert(n > 0);
         edges = new std::vector<Arc*>[n];
                                                                                     work++;
                                                                                     if ((*it)->f < (*it)->c && d[(*it)->v] +
         d = new int[n];
         exc = new int[n];
                                                                  1 < d[v]) {
         curArc = new int[n];
                                                                                               d[v] = d[(*it)->v] + 1;
         used = new bool[n];
                                                                                     }
         work= 0;
                                                                            }
         m = 0;
}
                                                                  bool Network::isAdmissible(Arc *a) {
Network::~Network() {
                                                                            return a - > f < a - > c & d[a - > u] == d[a - > v] + 1;
         for (int i = 0; i < n; i++) {
                  for (std::vector<Arc*>::iterator it =
edges[i].begin(); it != edges[i].end(); it++) {
                                                                  void Network::discharge(int v) {
                            delete *it;
                                                                            bool needRelabel = false;
```

```
for (;;) {
                                                                    int Network::getMaximumFlow(int s, int t) {
(isAdmissible(edges[v][curArc[v]])) {
                                                                             assert(0 <= s && s < n);
                             int delta =
                                                                             assert(0 \le t \&\& t < n);
push(edges[v][curArc[v]]);
                                                                             assert(s != t);
                            if (exc[edges[v][curArc[v]]-
                                                                             for (int i = 0; i < n; i++) {
>v] == delta) {
                                                                                       d[i] = 0;
                                      if
(d[edges[v][curArc[v]]->v] < n) {
                                                                             d[s] = n;
                                                                             for (int i = 0; i < n; i++) {
         active.insert(std::make_pair(d[edges[v][cur
                                                                                       exc[i] = 0;
Arc[v]]->v], edges[v][curArc[v]]->v));
                                                                                       for (std::vector<Arc*>::iterator it =
                                                                    edges[i].begin(); it != edges[i].end(); it++) {
                                                                                                (*it)->f=0;
                   }
                   else {
                            if (curArc[v] !=
                                                                             for (std::vector<Arc*>::iterator it =
edges[v].size() - 1) curArc[v]++;
                                                                    edges[s].begin(); it != edges[s].end(); it++) {
                             else {
                                                                                       (*it)->f = (*it)->c;
                                      curArc[v] = 0;
                                                                                       (*it)->rev->f = -(*it)->c;
                                      needRelabel =
                                                                                       exc[(*it)->u] -= (*it)->c;
true;
                                                                                       exc[(*it)->v] += (*it)->c;
                                      break;
                                                                             active.clear();
                            }
                                                                             for (int i = 0; i < n; i++) {
                   if (!exc[v] | | needRelabel) break;
                                                                                       curArc[i] = 0;
                                                                                       if (i == s \mid | i == t) continue;
                                                                                       if (!exc[i]) continue;
         if (needRelabel) relabel(v);
}
                                                                                       active.insert(std::make_pair(d[i],
                                                                    i));
void Network::globalRelabeling(int s, int t) {
         std::queue<int> q;
                                                                             while (!active.empty()) {
         q.push(t);
                                                                                       std::multimap<int, int>::iterator it =
         for (int i = 0; i < n; i++) {
                                                                    active.end();
                   d[i] = n;
                                                                                       it--:
         }
                                                                                       int node = it->second;
         d[t] = 0;
                                                                                       assert(d[node] == it->first);
         while (!q.empty()) {
                                                                                       assert(d[node] < n);
                   int v = q.front();
                                                                                       //printf("%d\n", d[node]);
                   q.pop();
                                                                                       active.erase(it);
                   for (vector<Arc*>::iterator it =
                                                                                       if (node == s \mid | node == t)
edges[v].begin(); it != edges[v].end(); it++) {
                                                                    continue;
                                                                                       discharge(node);
                            Arc *a = (*it)->rev;
                            if (a->f < a->c && d[v] + 1 <
                                                                                       if (exc[node] > 0) {
d[a->u]) {
                                                                                                if (d[node] < n) {
                                      d[a->u] = d[v] + 1;
                                                                             active.insert(std::make_pair(d[node],
                                      q.push(a->u);
                            }
                                                                    node));
         }
                                                                                       }
                                                                                       if (work > RATIO * m) {
         active.clear();
         for (int i = 0; i < n; i++) {
                                                                                                work = 0;
                   if (i == s \mid | i == t) continue;
                                                                                                globalRelabeling(s, t);
                   if (!exc[i]) continue;
                                                                                       }
                   if (d[i] >= n) continue;
                   active.insert(std::make_pair(d[i],
                                                                             int res = 0;
i));
                                                                             for (std::vector<Arc*>::iterator it =
                                                                    edges[t].begin(); it != edges[t].end(); it++) {
         }
}
                                                                                       res += (*it)->rev->f;
```

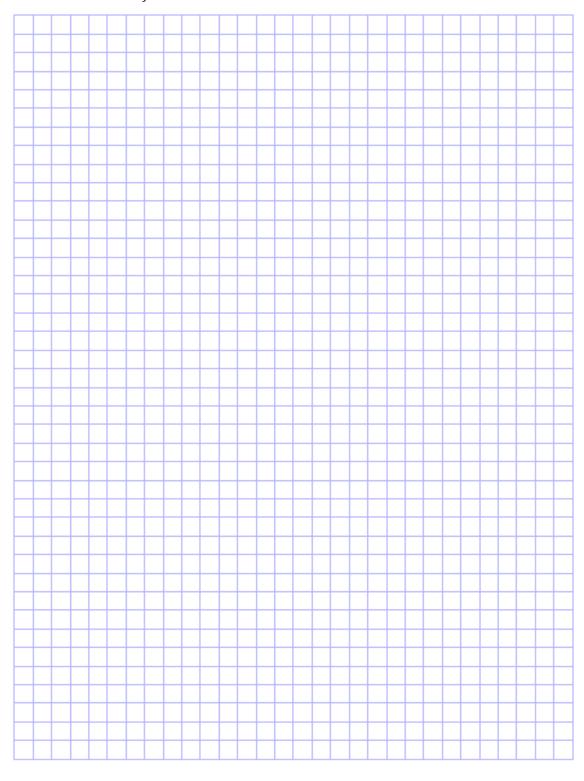
```
Moscow State University
```

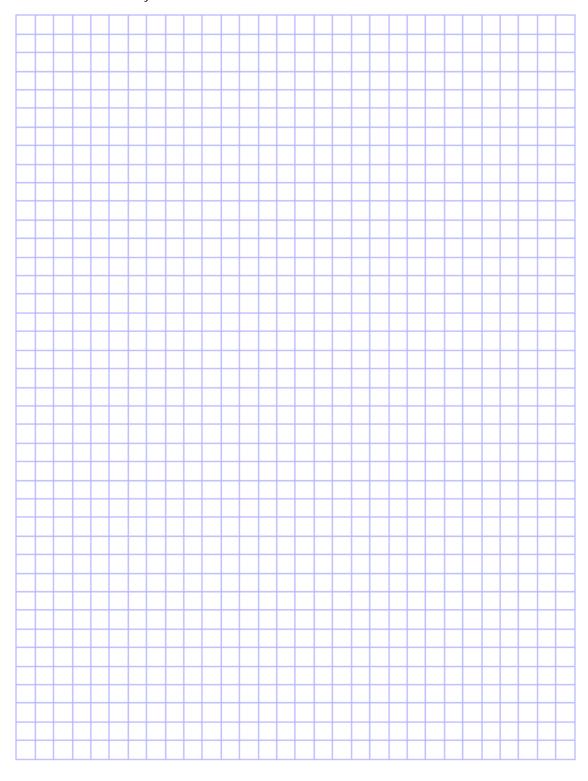
```
5
```

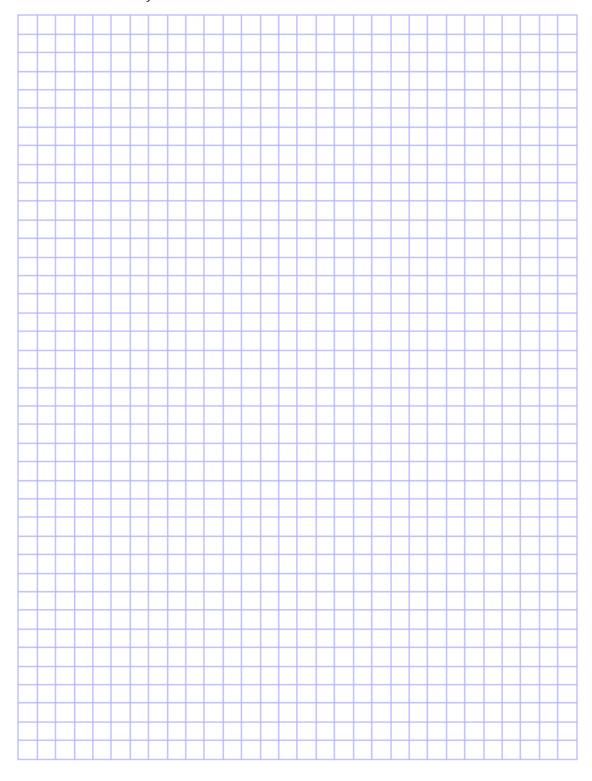
```
}
                                                                                                     I=r+1;
         return res;
                                                                                            }
}
                                                                                   } else {
                                                                                            if (Seq[R[v]-k+1]==Seq[I]) {
void Network::dfs(int v) {
                                                                                                     I++;
         if (used[v]) return;
                                                                                                     k--;
         used[v] = true;
                                                                                            } else {
         for (std::vector<Arc*>::iterator it =
                                                                                                     NV++;
edges[v].begin(); it != edges[v].end(); it++) {
                                                                                                     w=NV;
                  Arc *a = (*it)->rev;
                                                                                                     u=F[v]; F[v]=w;
                  assert(a->v == v);
                                                                F[w]=u;
                  if (a->f < a->c) dfs(a->u);
                                                                                                     L[w]=L[v];
         }
                                                                R[w]=R[v]-k;
}
                                                                                                     L[v]=R[v]-k+1;
std::vector<int> Network::getMinimumCut(int s, int
                                                                          C[u][Seq[L[w]]]=w;
                                                                                                     C[w][Seq[L[v]]]=v;
         getMaximumFlow(s, t);
                                                                                                     V=W;
         memset(used, 0, n * sizeof(bool));
                                                                                                     k=0;
         dfs(t);
                                                                                            }
         std::vector<int> res;
                                                                                   }
         for (int i = 0; i < n; i++) {
                                                                          }
                  if (used[i]) res.push_back(i);
         }
         return res;
                                                                void BravelyAdd(int V, int I, int r, int &v, int &k)
}
                                                                          int i=I;
int Network::getMinimumCut() {
                                                                         v=V; k=0;
         int res = 1e9;
                                                                          while (i<=r) {
                                                                                   v=C[v][Seq[i]];
         for (int i = 1; i < n; i++) {
                                                                                   i+=R[v]-L[v]+1;
                  res = std::min(res,
getMaximumFlow(0, i));
                                                                                   if (i>r)
         }
                                                                                            k=i-r-1;
         return res;
                                                                          }
}
                                                                }
Suffix tree
                                                                int main()
//SUFFIX TREE IMPLEMENTATION
const int root=1;
                                                                          Len++;
int Seq[max_n],Len;
                                                                          Seq[Len]=max_char;
                                                                          int last=root;
NV,C[max_n][max_char+1],F[max_n],L[max_n],R[ma
                                                                          F[root]=root;
x_n],Link[max_n];
                                                                          NV=1;
                                                                          int pv,pk;
void Add(int I, int r, int v, int k)
                                                                          for (int i=1; i<=Len; i++) {
                                                                                   int u=F[last];
         int w,u;
                                                                                   if (u==root) {
         while (I<=r) {
                                                                                            Add(i,Len,root,0);
                  if (k==0) {
                                                                                            last=NV;
                           if (C[v][Seq[I]]!=0) {
                                                                                   } else {
                                                                                            int v=F[u];
                                    v=C[v][Seq[I]];
                                    1++;
                                                                                            if (v==root) {
                                    k=R[v]-L[v];
                           } else {
                                                                          BravelyAdd(root,L[u]+1,R[u],pv,pk);
                                    NV++:
                                    w=NV;
                                                                          Add(L[last],R[last],pv,pk);
                                    L[w]=I; R[w]=r;
                                                                                                     last=NV;
                                    F[w]=v;
                                                                                            } else {
                                    C[v][Seq[I]]=w;
                                                                                                     int w=Link[v];
```

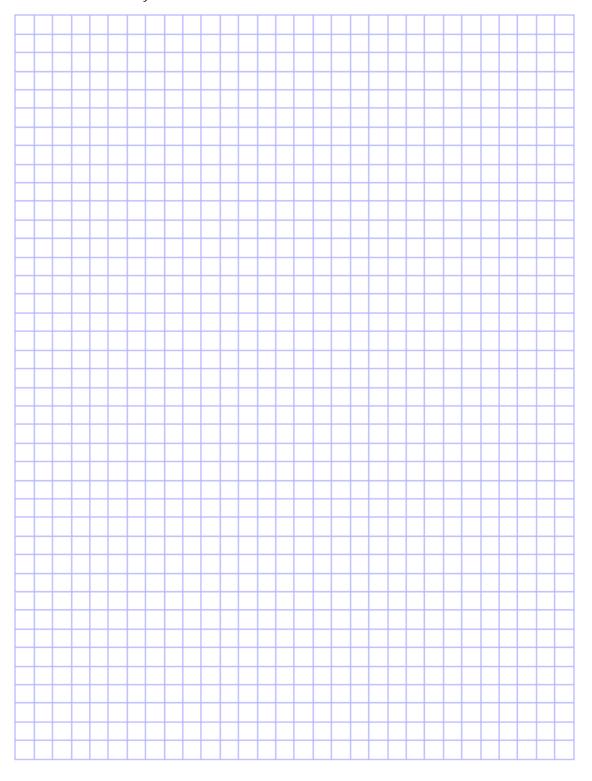
```
S += '$';
          BravelyAdd(w,L[u],R[u],pv,pk);
                                                                              n = int(s.size());
                                                                              init_steps(a[0], b[0], c[0]);
          Add(L[last],R[last],pv,pk);
                                                                              int u = 0, v = 1, h = 1;
                                      last=NV;
                                                                              while (h < n)
                             if (pk==0)
                                                                                        make_step(h, a[u], b[u], c[u], a[v],
                                      Link[u]=pv;
                                                                     b[v], c[v]);
                             else
                                                                                        swap(u, v);
                                                                                        h *= 2;
                                      Link[u]=F[last];
                   }
                                                                              for (int i = 0; i < n; i++)
          }
          return 0;
                                                                              {
}
                                                                                        for (int j = a[u][i]; j < n; j++)
                                                                                                 printf("%c", s[j]);
Suffix Array
                                                                                        printf("\n");
string s;
                                                                              return 0;
int n:
int a[2][101000], b[2][101000], c[2][101000];
int cnt[128], cur[128];
                                                                     Suffix automat
void init_steps(int *a, int *b, int *c)
                                                                     struct state {
                                                                              int length, link, cnt;
                                                                              int endpos;
          memset(cnt, 0, sizeof(cnt));
                                                                              int next[255];
          for (int i = 0; i < n; i++) cnt[s[i]]++;
          cur[0] = 0;
                                                                              bool clonned;
          for (int i = 1; i < 128; i++) cur[i] = cur[i - 1] +
                                                                              bool terminal;
                                                                              vi ilink;
cnt[i - 1];
          for (int i = 0; i < n; i++) a[cur[s[i]]++] = i;
                                                                              state() {
          for (int i = 0; i < n; i++)
                                                                                        length=0;
                                                                                        link=-1:
         {
                   b[a[i]] = i;
                                                                                        cnt=0;
                   if (!i \mid | s[a[i]] != s[a[i-1]]) c[i] = i;
                                                                                        endpos=-1;
                   else c[i] = c[i - 1];
                                                                                        cnt=0:
         }
                                                                                        clonned=false:
}
                                                                                        terminal=false;
                                                                                        memset(next,-1,sizeof(next));
void make_step(int h, int *a1, int *b1, int *c1, int
                                                                              }
*a2, int *b2, int *c2)
                                                                     };
{
          for (int i = 0; i < n; i++) c2[i] = i;
                                                                     state st[1000*1000];
          for (int i = 0; i < n; i++)
                                                                     int last, size;
          {
                   a1[i] = (a1[i] - h + 2 * n) % n;
                                                                     void extend(char ch) {
                                                                              int nlast=size++;
                   a2[c2[c1[b1[a1[i]]]]++] = a1[i];
                                                                              st[nlast].length=st[last].length+1;
          for (int i = 0; i < n; i++)
                                                                              st[nlast].endpos=st[nlast].length-1;
                                                                              int p=last;
          {
                   b2[a2[i]] = i;
                                                                              while (p!=-1 && st[p].next[ch]==-1) {
                                                                                        st[p].next[ch]=nlast;
                   if (!i | | c1[b1[a2[i]]] != c1[b1[a2[i -
1]]] || c1[b1[(a2[i] + h) % n]] != c1[b1[(a2[i - 1] + h) %
                                                                                        p=st[p].link;
n]]) c2[i] = i;
                                                                              if (p==-1)
                   else c2[i] = c2[i - 1];
                                                                                        st[nlast].link=0;
         }
}
                                                                              else {
                                                                                        int q=st[p].next[ch];
int main()
                                                                                        if (st[p].length+1==st[q].length)
                                                                                                  st[nlast].link=q;
{
                                                                                        else {
          cin >> s;
```

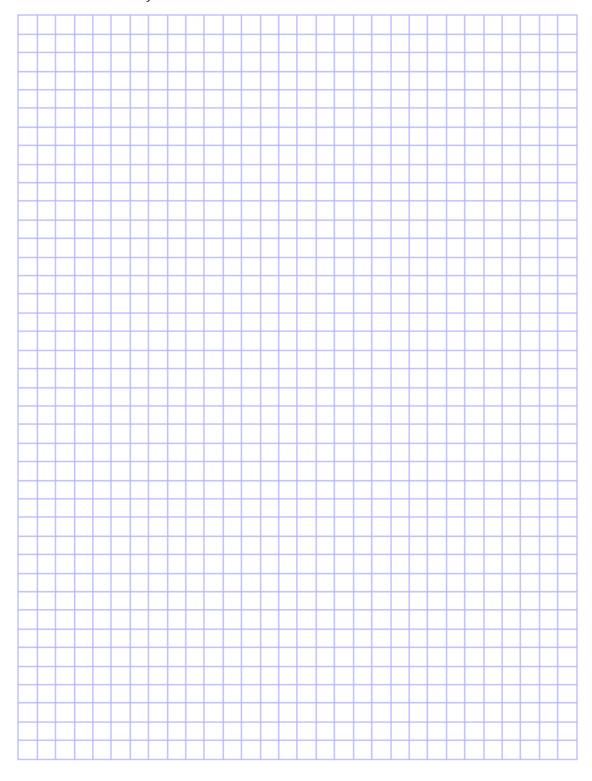
```
if (p==-1) {
                           int clone=size++;
                                                                                   printf("No such patterns\n");
                           st[clone]=st[q];
                                                                                   return 0;
         st[clone].length=st[p].length+1;
                                                                          }
                           st[clone].clonned=true;
                                                                          printf("First occurrence at
                                                                 %d\n",st[p].endpos-sz(pat)+1);
         st[nlast].link=st[q].link=clone;
                                                                          for (int i=0; i<size; i++)
                           while (p!=-1 &&
                                                                                   if (st[i].link!=-1)
st[p].next[ch]==q) {
                                                                                            st[st[i].link].ilink.pb(i);
                                                                          calcCnt(0);
                                                                          printf("Count of occurrences is
         st[p].next[ch]=clone;
                                                                 %d\n",st[p].cnt);
                                     p=st[p].link;
                           }
                                                                          return 0;
                  }
         }
                                                                 }
         last=nlast;
                                                                                        TEMPLATE
                                                                 #include <cstdio>
vs ans;
                                                                 #include <cstring>
                                                                 #include <cstdlib>
void out(string cur, int at) {
                                                                 #include <cmath>
         if (st[at].terminal)
                                                                 #include <cassert>
                  cout<<cur<<endl;
                                                                 #include <iostream>
         for (int i=0; i<255; i++)
                                                                 #include <sstream>
                  if (st[at].next[i]!=-1)
                                                                 #include <algorithm>
                                                                 #include <vector>
         out(cur+(char)i,st[at].next[i]);
                                                                 #include <set>
}
                                                                 #include <map>
                                                                 #include <queue>
int calcCnt(int at) {
         int res=!st[at].clonned;
                                                                 using namespace std;
         for (int i=0; i<sz(st[at].ilink); i++)
                  res+=calcCnt(st[at].ilink[i]);
                                                                 #define sz(v) ((int) (v).size())
                                                                 #define all(v) (v).begin(), (v).end()
         return st[at].cnt=res;
}
                                                                 #define mp make_pair
                                                                 #define pb push_back
int main()
                                                                 typedef long long II;
         freopen("input.txt","r",stdin);
                                                                 typedef long long int64;
         freopen("output.txt","w",stdout);
                                                                 typedef pair<int,int> ii;
                                                                 typedef vector<int> vi;
                                                                 typedef vector<string> vs;
         string s,pat;
         cin>>s>>pat;
                                                                 template<typename T> T abs(T x) { return x>0 ? x : -
         size=1;
                                                                 template<typename T> T sqr(T x) { return x*x;
         last=0;
                                                                                                                       }
         for (int i=0; i < sz(s); i++)
                  extend(s[i]);
                                                                 int main()
         int p=last;
                                                                 {
                                                                          freopen("","r",stdin);
         while (p!=-1) {
                                                                          freopen("","w",stdout);
                  st[p].terminal=true;
                  p=st[p].link;
         }
                                                                          return 0;
         p=0;
                                                                 }
         for (int i=0; i < sz(pat); i++)
                  if (p!=-1)
                           p=st[p].next[pat[i]];
```

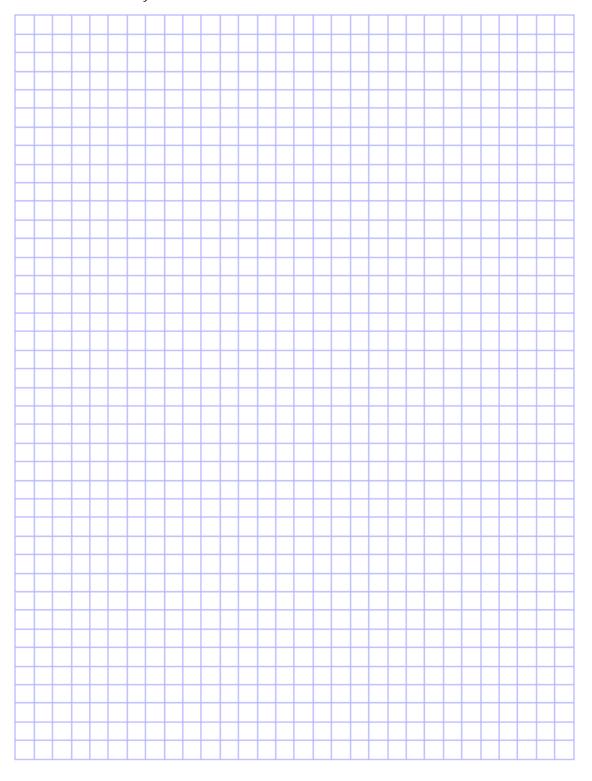


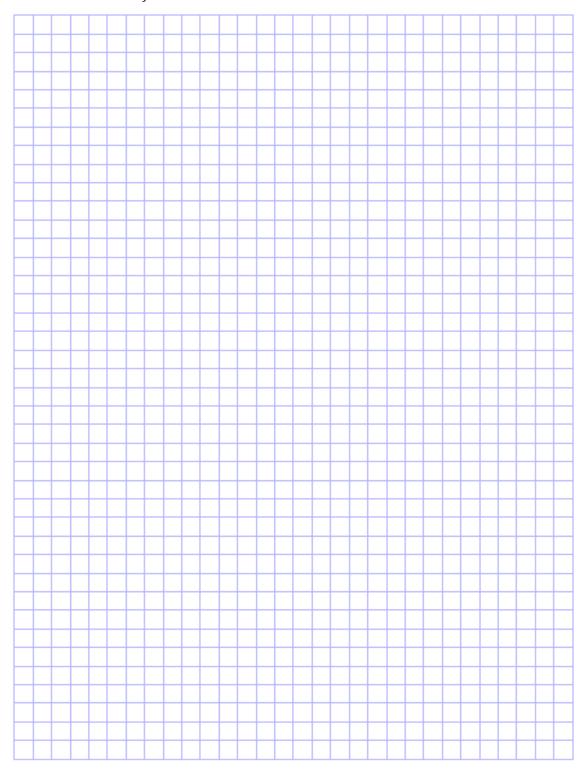


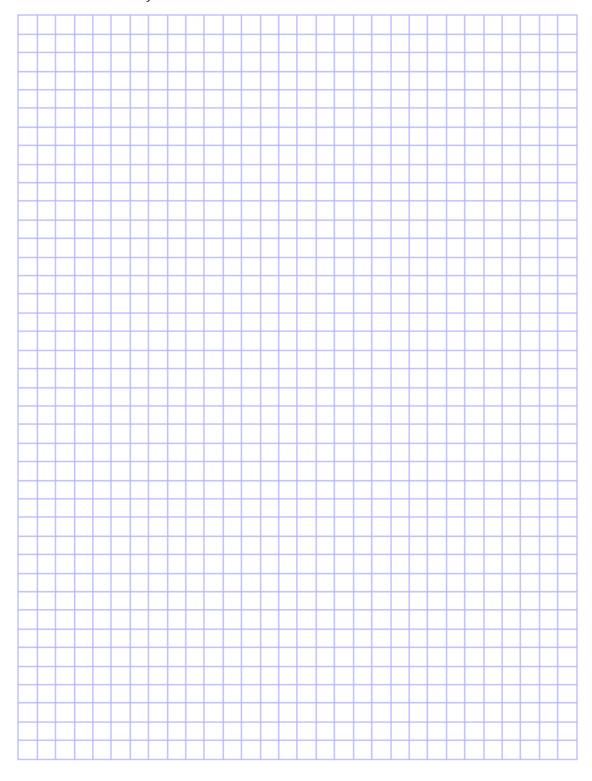












| A | |
|---|--|
| В | |
| С | |
| D | |
| E | |
| F | |
| G | |
| Н | |
| I | |
| J | |
| K | |
| L | |