Ellipsoids

```
int X[100],Y[100],Z[100],R[100];
int n;

const double eps=1e-4;
const int BUBEN=100000;

P v[3];
P w[3];
P c;

void Norm(P& p)
{
  double d[3];
  double s=0.0;
  for(int i=0;i<3;i++)
  {
    d[i]=(p&v[i])/(v[i]&v[i]);
    s+=d[i]*d[i];
  }
  double koef=1./sqrt(s);
  p=p*koef;
}

int main()
{
  int n;
  cin >> n;
  for(int i=0;i<n;i++)
    cin >> X[i] >> Y[i] >> Z[i] >> R[i];
  c=P(0,0,0);
  v[0]=P(100000,0,0);
  v[1]=P(0,100000,0);
  v[2]=P(0,0,100000);
  for(int ic=0;ic<BUBEN;ic++)
  {
    int ind=-1;
    for(int i=0;i<n;i++)
    {
      if((c-P(X[i],Y[i],Z[i])).len()>R[i]+eps)
      {
        ind=i;
        break;
      }
    }
    if(ind==-1)
    {
      printf("YES\n");
      return 0;
    }
    if(v[0].len()<eps || v[1].len()<eps ||
v[2].len()<eps) break;
    P t=(c-P(X[ind],Y[ind],Z[ind]));
    w[0]=t;
    if(fabs(t.z)<eps)
      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];
    for(int i=0;i<3;i++)
      Norm(w[i]);
    double N=3.0;
    c=c+w[0]*(1./(N+1.));
    v[0]=w[0]*((N+0.0)/(N+1.0));
    for(int i=1;i<3;i++)
      v[i]=w[i]*((N+0.0)/sqrt(N*N-1.0));
  }
  printf("NO\n");
  return 0;
}
```

Euler paths

```
// Copyright 2007 x13.
// All rights reserved.
//
// Havka - papstvo!
// Porvite tam vseh!
//Eulerian circuit in directed graph
void Euler (int a)
{
        for (int b = 0; b<N; b++) if (G[a][b])
        {
                G[a][b] = false;
                Euler(b);
        }
        Path.push_back(a);
} // and reverse(Path.begin(), Path.end()) after that
```

FFT

```
int rev(int x) {
        int res=0;
        for (int i=0; i<S; i++)
                if (x&(1<<i))
                        res|=1<<(S-i-1);
        return res;
}

Poly FFT(Poly a, int to=1) {
        Poly A;
        for (int i=0; i<n; i++)
                A[rev(i)]=a[i];
        for (int s=1; s<=S; s++) {
                int m=1<<s;
                C
wm=C(cos(to*2*pi/m),sin(to*2*pi/m));
                for (int k=0; k<n; k+=m) {
                        C w=C(1,0);
                        for (int j=0; j<m/2; j++) {
                                C u=A[k+j];
                                C
t=w*A[k+j+m/2];
                                A[k+j]=u+t;
                                A[k+j+m/2]=u-t;
                                w*=wm;
                        }
                }
        }
        if (to==-1)
```

```
                for (int i=0; i<n; i++)
                        A[i]/=n;
        return A;
}
```

Hopcraft-Karp

```
{$I-,Q-,R-,S-}
var n,m,k,i,x,y,cnt,qh,qt,j:longint;
  g:array [1..1000,1..1000] of boolean;
  a,b,l,q:array [0..1000] of longint;
  f:array [0..1000] of boolean;
function dfs(u:longint):boolean;
var j:longint;
begin
 dfs:=true;
 if u=0 then
  exit;
 f[u]:=false;
 for j:=1 to m do
  if g[u,j] and f[b[j]] and (l[b[j]]=l[u]+1) and dfs(b[j])
then
  begin
   b[j]:=u;
   a[u]:=j;
   exit;
  end;
 dfs:=false;
end;
procedure put(x,d:longint);
begin
 f[x]:=true;
 l[x]:=d;
 q[qt]:=x;
 inc(qt);
end;
function get:longint;
begin
 get:=q[qh];
 inc(qh);
end;
begin
 read(n,m,k);
 fillchar(g,sizeof(g),false);
 for i:=1 to k do
 begin
  read(x,y);
  g[x,y]:=true;
 end;
 fillchar(a,sizeof(a),0);
 fillchar(b,sizeof(b),0);
 cnt:=0;
 repeat
  qh:=0;
  qt:=0;
  fillchar(l,sizeof(l),63);
  fillchar(f,sizeof(f),0);
  for i:=1 to n do
   if a[i]=0 then
    put(i,0);
  while qh<qt do
  begin
   x:=get;
   if l[x]=l[0] then
    break;
   for j:=1 to m do
    if g[x,j] and not f[b[j]] then
     put(b[j],l[x]+1);
  end;
  if not f[0] then
   break;
  for i:=1 to n do
   if (a[i]=0) and dfs(i) then
    inc(cnt);
 until false;
 writeln(cnt);
end.
```

KMP

```
void KMP( const string &s, vector< int > &Len)
{
        Len.resize(s.length());
        Len[ 0] = 0;
        for (int i = 1; i < s.length(); i++)
        {
                int len = Len[i - 1];
                while (len > 0 && s[i] != s[len])
                        len = Len[len - 1];
                if (s[i] == s[len])
                        len++;
                Len[i] = len;
        }
}
```

Z-function

```
vi Z(string s) {
        vi z(s.size());
        int j=0;
        for (int i=1; i<s.size(); i++) {
                int r=min(j+z[j]-1,i+z[i-j]-1);
                if (r<i) z[i]=0; else z[i]=r-i+1;
                while (i+z[i]<s.size() &&
s[i+z[i]]==s[z[i]])
                        z[i]++;
                if (i+z[i]>j+z[j])
                        j=i;
        }
        return z;
}
```

Preflow-push

```
class Network {
public:

        struct Arc {

                int u;
                int v;
                int c;
```

```
		int f;
		Arc *rev;

		Arc(int u, int v, int c): u(u), v(v),
c(c), f(0), rev(0) {
		}
	};

	Network(int n);
	~Network();

	void addEdge(int u, int v, int c);
	int getMaximumFlow(int s, int t);
	std::vector<int> getMinimumCut(int s, int t);
	int getMinimumCut();

private:

	static const double RATIO = 2.0;

	Network(const Network&);
	Network& operator=(const Network&);

	int n, m;
	std::vector<Arc*> *edges;

	int *d;
	int *exc;
	int *curArc;
	bool *used;

	int work;

	std::multimap<int, int> active;

	bool isAdmissible(Arc *a);
	int push(Arc *a);
	void relabel(int v);
	void discharge(int v);
	void dfs(int v);
	void globalRelabeling(int s, int t);
};

Network::Network(int n): n(n) {
	assert(n > 0);
	edges = new std::vector<Arc*>[n];
	d = new int[n];
	exc = new int[n];
	curArc = new int[n];
	used = new bool[n];
	work= 0;
	m = 0;
}

Network::~Network() {
	for (int i = 0; i < n; i++) {
		for (std::vector<Arc*>::iterator it =
edges[i].begin(); it != edges[i].end(); it++) {
			delete *it;
		}
	}
	delete[] edges;
	delete[] d;
	delete[] exc;
	delete[] curArc;
	delete[] used;
}

void Network::addEdge(int u, int v, int c) {
	assert(0 <= u && u < n);
	assert(0 <= v && v < n);
	assert(u != v);
	assert(c >= 0);
	Arc *a = new Arc(u, v, c);
	Arc *b = new Arc(v, u, c);
	a->rev = b;
	b->rev = a;
	edges[u].push_back(a);
	edges[v].push_back(b);
	m += 2;
}

void checkArray(std::vector<int> &a) {
	int old = a.size();
	std::sort(a.begin(), a.end());
	a.erase(std::unique(a.begin(), a.end()),
a.end());
	assert(a.size() == old);
}

int Network::push(Arc *a) {
	work++;
	int delta = std::min(a->c - a->f, exc[a->u]);
	exc[a->u] -= delta;
	exc[a->v] += delta;
	a->f += delta;
	a->rev->f -= delta;
	return delta;
}

void Network::relabel(int v) {
	d[v] = 2 * n;
	for (std::vector<Arc*>::iterator it =
edges[v].begin(); it != edges[v].end(); it++) {
		work++;
		if ((*it)->f < (*it)->c && d[(*it)->v] +
1 < d[v]) {
			d[v] = d[(*it)->v] + 1;
		}
	}
}

bool Network::isAdmissible(Arc *a) {
	return a->f < a->c && d[a->u] == d[a->v] + 1;
}

void Network::discharge(int v) {
	bool needRelabel = false;
	for (;;) {
		if
(isAdmissible(edges[v][curArc[v]])) {
			int delta =
push(edges[v][curArc[v]]);
			if (exc[edges[v][curArc[v]]-
>v] == delta) {
				if
(d[edges[v][curArc[v]]->v] < n) {

	active.insert(std::make_pair(d[edges[v][cur
Arc[v]]->v], edges[v][curArc[v]]->v));
				}
			}
		}
		else {
			if (curArc[v] !=
edges[v].size() - 1) curArc[v]++;
			else {
				curArc[v] = 0;
				needRelabel =
true;
				break;
			}
		}
		if (!exc[v] || needRelabel) break;
	}
	if (needRelabel) relabel(v);
}

void Network::globalRelabeling(int s, int t) {
	std::queue<int>   q;
	q.push(t);
	for (int i = 0; i < n; i++) {
		d[i] = n;
	}
	d[t] = 0;
	while (!q.empty()) {
		int v = q.front();
		q.pop();
		for (vector<Arc*>::iterator it =
edges[v].begin(); it != edges[v].end(); it++) {
			Arc *a = (*it)->rev;
			if (a->f < a->c && d[v] + 1 <
d[a->u]) {
				d[a->u] = d[v] + 1;
				q.push(a->u);
			}
		}
	}
	active.clear();
	for (int i = 0; i < n; i++) {
		if (i == s || i == t) continue;
		if (!exc[i]) continue;
		if (d[i] >= n) continue;
		active.insert(std::make_pair(d[i],
i));
	}
}

int Network::getMaximumFlow(int s, int t) {
	assert(0 <= s && s < n);
	assert(0 <= t && t < n);
	assert(s != t);
	for (int i = 0; i < n; i++) {
		d[i] = 0;
	}
	d[s] = n;
	for (int i = 0; i < n; i++) {
		exc[i] = 0;
		for (std::vector<Arc*>::iterator it =
edges[i].begin(); it != edges[i].end(); it++) {
			(*it)->f = 0;
		}
	}
	for (std::vector<Arc*>::iterator it =
edges[s].begin(); it != edges[s].end(); it++) {
		(*it)->f = (*it)->c;
		(*it)->rev->f = -(*it)->c;
		exc[(*it)->u] -= (*it)->c;
		exc[(*it)->v] += (*it)->c;
	}
	active.clear();
	for (int i = 0; i < n; i++) {
		curArc[i] = 0;
		if (i == s || i == t) continue;
		if (!exc[i]) continue;
		active.insert(std::make_pair(d[i],
i));
	}
	while (!active.empty()) {
		std::multimap<int, int>::iterator it =
active.end();
		it--;
		int node = it->second;
		assert(d[node] == it->first);
		assert(d[node] < n);
		//printf("%d\n", d[node]);
		active.erase(it);
		if (node == s || node == t)
continue;
		discharge(node);
		if (exc[node] > 0) {
			if (d[node] < n) {

	active.insert(std::make_pair(d[node],
node));
			}
		}
		if (work > RATIO * m) {
			work = 0;
			globalRelabeling(s, t);
		}
	}
	int res = 0;
	for (std::vector<Arc*>::iterator it =
edges[t].begin(); it != edges[t].end(); it++) {
		res += (*it)->rev->f;
	}
	return res;
}

void Network::dfs(int v) {
	if (used[v]) return;
	used[v] = true;
	for (std::vector<Arc*>::iterator it =
edges[v].begin(); it != edges[v].end(); it++) {
		Arc *a = (*it)->rev;
		assert(a->v == v);
		if (a->f < a->c) dfs(a->u);
	}
}

std::vector<int> Network::getMinimumCut(int s, int
t) {
	getMaximumFlow(s, t);
	memset(used, 0, n * sizeof(bool));
	dfs(t);
	std::vector<int> res;
	for (int i = 0; i < n; i++) {
		if (used[i]) res.push_back(i);
	}
	return res;
}

int Network::getMinimumCut() {
	int res = 1e9;
	for (int i = 1; i < n; i++) {
		res = std::min(res,
getMaximumFlow(0, i));
	}
	return res;
}
```

<u>Suffix tree</u>

```
//SUFFIX TREE IMPLEMENTATION
const int root=1;
int Seq[max_n],Len;
int
NV,C[max_n][max_char+1],F[max_n],L[max_n],R[ma
x_n],Link[max_n];

void Add(int l, int r, int v, int k)
{
	int w,u;
	while (l<=r) {
		if (k==0) {
			if (C[v][Seq[l]]!=0) {
				v=C[v][Seq[l]];
				l++;
				k=R[v]-L[v];
			} else {
				NV++;
				w=NV;
				L[w]=l; R[w]=r;
				F[w]=v;
				C[v][Seq[l]]=w;
				l=r+1;
			}
		} else {
			if (Seq[R[v]-k+1]==Seq[l]) {
				l++;
				k--;
			} else {
				NV++;
				w=NV;
				u=F[v]; F[v]=w;
F[w]=u;
				L[w]=L[v];
R[w]=R[v]-k;
				L[v]=R[v]-k+1;

	C[u][Seq[L[w]]]=w;
				C[w][Seq[L[v]]]=v;
				v=w;
				k=0;
			}
		}
	}
}

void BravelyAdd(int V, int l, int r, int &v, int &k)
{
	int i=l;
	v=V; k=0;
	while (i<=r) {
		v=C[v][Seq[i]];
		i+=R[v]-L[v]+1;
		if (i>r)
			k=i-r-1;
	}
}

int main()
{
	Len++;
	Seq[Len]=max_char;
	int last=root;
	F[root]=root;
	NV=1;
	int pv,pk;
	for (int i=1; i<=Len; i++) {
		int u=F[last];
		if (u==root) {
			Add(i,Len,root,0);
			last=NV;
		} else {
			int v=F[u];
			if (v==root) {

	BravelyAdd(root,L[u]+1,R[u],pv,pk);

	Add(L[last],R[last],pv,pk);
				last=NV;
			} else {
				int w=Link[v];
	BravelyAdd(w,L[u],R[u],pv,pk);

	Add(L[last],R[last],pv,pk);
					last=NV;
				}
				if (pk==0)
					Link[u]=pv;
				else
					Link[u]=F[last];
			}
		}
		return 0;
}
```

Suffix Array

```
string s;
int n;
int a[2][101000], b[2][101000], c[2][101000];
int cnt[128], cur[128];

void init_steps(int *a, int *b, int *c)
{
	memset(cnt, 0, sizeof(cnt));
	for (int i = 0; i < n; i++) cnt[s[i]]++;
	cur[0] = 0;
	for (int i = 1; i < 128; i++) cur[i] = cur[i - 1] +
cnt[i - 1];
	for (int i = 0; i < n; i++) a[cur[s[i]]++] = i;
	for (int i = 0; i < n; i++)
	{
		b[a[i]] = i;
		if (!i || s[a[i]] != s[a[i - 1]]) c[i] = i;
		else c[i] = c[i - 1];
	}
}

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;
	for (int i = 0; i < n; i++)
	{
		a1[i] = (a1[i] - h + 2 * n) % n;
		a2[c2[c1[b1[a1[i]]]]++] = a1[i];
	}
	for (int i = 0; i < n; i++)
	{
		b2[a2[i]] = i;
		if (!i || c1[b1[a2[i]]] != c1[b1[a2[i -
1]]] || c1[b1[(a2[i] + h) % n]] != c1[b1[(a2[i - 1] + h) %
n]]) c2[i] = i;
		else c2[i] = c2[i - 1];
	}
}

int main()
{
	cin >> s;
	s += '$';
	n = int(s.size());
	init_steps(a[0], b[0], c[0]);
	int u = 0, v = 1, h = 1;
	while (h < n)
	{
		make_step(h, a[u], b[u], c[u], a[v],
b[v], c[v]);
		swap(u, v);
		h *= 2;
	}
	for (int i = 0; i < n; i++)
	{
		for (int j = a[u][i]; j < n; j++)
			printf("%c", s[j]);
		printf("\n");
	}
	return 0;
}
```

Suffix automat

```
struct state {
	int length,link,cnt;
	int endpos;
	int next[255];
	bool clonned;
	bool terminal;
	vi ilink;
	state() {
		length=0;
		link=-1;
		cnt=0;
		endpos=-1;
		cnt=0;
		clonned=false;
		terminal=false;
		memset(next,-1,sizeof(next));
	}
};

state st[1000*1000];
int last,size;

void extend(char ch) {
	int nlast=size++;
	st[nlast].length=st[last].length+1;
	st[nlast].endpos=st[nlast].length-1;
	int p=last;
	while (p!=-1 && st[p].next[ch]==-1) {
		st[p].next[ch]=nlast;
		p=st[p].link;
	}
	if (p==-1)
		st[nlast].link=0;
	else {
		int q=st[p].next[ch];
		if (st[p].length+1==st[q].length)
			st[nlast].link=q;
		else {
					int clone=size++;
					st[clone]=st[q];

		st[clone].length=st[p].length+1;
					st[clone].clonned=true;

		st[nlast].link=st[q].link=clone;
					while (p!=-1 &&
st[p].next[ch]==q) {

		st[p].next[ch]=clone;
						p=st[p].link;
					}
				}
		}
		last=nlast;
}

vs ans;

void out(string cur, int at) {
		if (st[at].terminal)
				cout<<cur<<endl;
		for (int i=0; i<255; i++)
				if (st[at].next[i]!=-1)

		out(cur+(char)i,st[at].next[i]);
}

int calcCnt(int at) {
		int res=!st[at].clonned;
		for (int i=0; i<sz(st[at].ilink); i++)
				res+=calcCnt(st[at].ilink[i]);
		return st[at].cnt=res;
}

int main()
{
		freopen("input.txt","r",stdin);
		freopen("output.txt","w",stdout);

		string s,pat;
		cin>>s>>pat;

		size=1;
		last=0;
		for (int i=0; i<sz(s); i++)
				extend(s[i]);
		int p=last;
		while (p!=-1) {
				st[p].terminal=true;
				p=st[p].link;
		}
		p=0;
		for (int i=0; i<sz(pat); i++)
				if (p!=-1)
					p=st[p].next[pat[i]];
		if (p==-1) {
				printf("No such patterns\n");
				return 0;
		}
		printf("First occurrence at
%d\n",st[p].endpos-sz(pat)+1);
		for (int i=0; i<size; i++)
				if (st[i].link!=-1)
					st[st[i].link].ilink.pb(i);
		calcCnt(0);
		printf("Count of occurrences is
%d\n",st[p].cnt);

		return 0;
}
```

TEMPLATE

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <cassert>
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <set>
#include <map>
#include <queue>

using namespace std;

#define sz(v) ((int) (v).size())
#define all(v) (v).begin(), (v).end()
#define mp make_pair
#define pb push_back

typedef long long ll;
typedef long long int64;
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<string> vs;

template<typename T> T abs(T x) { return x>0 ? x : -
x; }
template<typename T> T sqr(T x) { return x*x;		}

int main()
{
		freopen("","r",stdin);
		freopen("","w",stdout);

		return 0;
}
```

A	
B	
C	
D	
E	
F	
G	
H	
I	
J	
K	
L	