Università della Svizzera italiana

Team Notebook
illUSIon

Bezdrighin Marcel, Eljon Harlicaj, Guerra Fabio

Contents

1 Template

Write it before the contest begins and then reuse.

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <set>
#include <map>
#include <cmath>
#include <cstring>
#include <ctime>
#include <unordered_map>
#include <iomanip>
#include <complex>
#include <cassert>
using namespace std;

#define fi first
#define se second
#define pb push_back
#define all(v) (v).begin(),(v).end()
#define deb(a) cerr<< #a << "= " << (a)<<"\n";

typedef long long ll;
typedef unsigned long long ull;
typedef unsigned int uint;
typedef long double ld;
typedef vector<int> vi;
typedef pair<int,int> pii;

template<class T>
ostream& operator<<(ostream& stream, const vector<T> v
    ){ stream << "[ "; for (int i=0; i<(int)v.size();
    i++) stream << v[i] << " "; stream << "]"; return
    stream; }

ll fpow(ll x, ll p, ll m){ll r=1; for (;p;p>>=1){ if (
    p&1) r=r*x%m; x=x*x%m; } return r;}
int gcd(int a, int b){ if (!b) return a; return gcd(b,
    a%b);}
ll gcd(ll a, ll b){ if (!b) return a; return gcd(b,a%b
    );}


int main(){
  ios::sync_with_stdio(0);
  cin.tie(0);

  return 0;
}
```

2 Mathematics

2.1 Number Theory

2.1.1 Mobius inversion formula

$$g = f \star 1 \Leftrightarrow f = \mu \star g$$

Example:

$$\sum_{d|n} \phi(d) = n \Leftrightarrow \phi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d}$$

2.1.2 Cipolla's Algorithm

Computes the square root of an elements in a field of prime order.

Let $p \geq 3$ prime and $n \in F_p$. To compute x with $x^2 = n$, take $a \in F_p$ such that $a^2 - n$ is not a square. Take the extension $F_{p^2} = F_p(\sqrt{a^2 - n})$ and compute $x = (a + \sqrt{a^2 - n})^{(p+1)/2}$

2.1.3 Extendend Euclidean Algorithm

```
void gcdext(LL a, LL b,LL &x,LL &y){
  if (b==0){
    x=1,y=0,gcd=a;
    return;
  }

  LL x0,y0;
  gcdext(b,a%b,x0,y0);
  x=y0;
  y=x0-y0*(a/b);
}
```

2.1.4 Lucas's theorem

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} (mod\ p)$$

m_i and n_i are the i^{th} digits in the base p representation.

2.1.5 Bezout identity

Let $a \neq 0, b \neq 0$. $d = \gcd(a, b)$ is the smallest positive integer for which there integer solutions to:

$$xa + yb = d$$

If (x, y) is one solution, the other solutions are given by:

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{kb}{\gcd(a,b)}\right), k \in \mathbb{Z}$$

2.1.6 Chinese remainder theorem

Let m_1 and m_2 such that $\gcd(m_1, m_2) = 1$. Then the following system:

$$x \equiv a_1 (mod\ m_1)$$

$$x \equiv a_2 (mod\ m_2)$$

has an unique solution $mod\ m_1 m_2$, given by:

$$a_1 \cdot m_2 \cdot m_2^{-1} + a_2 \cdot m_1 \cdot m_1^{-1}$$

2.1.7 Miller-Rabin

To check wether n is prime, write $n = 2^s d$. Now if:

$$a^d \not\equiv 1 (mod\ n)$$

and

$$a^{2^r d} \not\equiv -1 (mod\ n)$$

for $0 \leq r \leq s - 1$ then n is not prime. Use around 10 iterations for accuracy.

```
def checkPrime(n):
  if n==2: return 1
  if n%2==0: return 0

  d=n-1
  s=0

  while d%2==0:
    s+=1
    d//=2


  for a in range(2,min(17,n-1)):
    if not a==2 and a%2==0: continue

    v=fpow(a,d,n)
    if v==1 or v==n-1: continue

    i=1
    while i<=s-1:
      v=v*v%n
      if v==1: return 0
      if v==n-1: break
      i+=1

    if i==s:
      return 0

  return 1
```

2.2 Equations

2.2.1 Cramer's Rule

$$\begin{cases} ax + by = e \\ cx + dy = f \end{cases} \Rightarrow \begin{cases} x = \frac{ed-bf}{ad-bc} \\ y = \frac{af-ec}{ad-bc} \end{cases}$$

For a general system of equations, use Cramer's rule. If we have a system $Ax = b$, then the solutions are:

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is formed by replacing the i^{th} column with b.

2.2.2 Gaussian Elimination

```
for (j=1; j<=min(N,M); j++){
    int ind=-1;
    for (i=j; i<=N; i++)
      if (a[i][j]<-EPS || a[i][j]>EPS){
        ind=i;
        break;
      }

    if (ind==-1) continue;

    LD val=a[ind][j];
    for (k=j; k<=M+1; k++){
      swap(a[j][k],a[ind][k]);
```

```
      a[ind][k]/=val;
    }

    ind = j;
    for (i=ind+1; i<=N; i++){
      LD coef=-a[i][j]/a[ind][j];
      for (k=j; k<=M+1; k++)
        a[i][k]+=coef*a[ind][k];
    }
  }



  for (i=N; i>0; i--){
    for (j=1; j<=M+1; j++)
      if (a[i][j]<-EPS || a[i][j]>EPS){
        if (j==M+1){
          printf("Imposibil\n");
          return 0;
        }

        x[j]=a[i][M+1];
        for (k=j+1; k<=M; k++)
          x[j]-=x[k]*a[i][k];
        x[j]/=a[i][j];
        break;
      }
  }
```

2.3 Combinatorics

2.3.1 Catalan numbers:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Number of lattice paths that don't go above $x = y$, number of rooted ordered trees, number of full binary trees with $n+1$ leaves, etc.

2.3.2 Stirling numbers

$$S(n,k) = kS(n-1,k) + S(n,k-1)$$

$$B_n = \sum_{k=0}^{n} S(n,k)$$

Number of ways to partition n elements into k sets.

2.3.3 Inclusion-Exclusion

$$\left|\bigcup_{1\le i\le n} A_i\right| = \sum_{1\le i_1\le n} |A_{i_1}| - \sum_{1\le i_1<i_2\le n} |A_{i_1}\cap A_{i_2}|$$

$$+ \sum_{1\le i_1<i_2<i_3\le n} |A_{i_1}\cap A_{i_2}\cap A_{i_3}| - \ldots + (-1)^{n+1}\left|\bigcap_{i=1}^{n} A_i\right|.$$

2.3.4 Burnside lemma

Let G be a finite group which acts on X. X^g - number of elements fixed by g. We have:

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

Polya enumeration theorem follows from Burnside - look at the number of cycles.

2.3.5 Cayley's formula

Number of labeled trees on n vertices:

$$n^{n-2}$$

2.3.6 Generalized Cayley formula

If there are k connected components, each of size s_i, the number of possible trees on these components is:

$$\sum_{\substack{d_i\ge 1,\\ \sum_{i=1}^{k} d_i = 2k-2}} s_1^{d_1}\cdot\ldots\cdot s_k^{d_k}\binom{k-2}{d_1-1,\ldots,d_k-1} =$$

$$= s_1\cdot\ldots\cdot s_k\cdot n^{k-2}$$

2.3.7 Matrix tree theorem

D - degree matrix, A adjacency matrix. $L = D - A$ - laplacian. The number of spanning trees is calculated as the determinant of L after removing the last row and column.

2.4 Geometry

2.4.1 Pick's theorem

$$A = i + \frac{b}{2} - 1$$

i - number of lattice points in the interior, b - number of lattice points on the boundary. Works for polygons with integer coordinates.

2.4.2 Lattice points on segment

Number of points on a lattice segment from (a,b) to (c,d) is:

$$\gcd(c-a, d-b) + 1$$

2.4.3 Triangles

Side lengths: a, b, c
Semiperimeter: $p = \frac{a+b+c}{2}$
Area: $A = \sqrt{p(p-a)(p-b)(p-c)}$
Circumradius: $R = \frac{abc}{4A}$
Inradius: $r = \frac{A}{p}$
Law of sines: $\frac{\sin\alpha}{a} = \frac{1}{2R}$
Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$

2.5 Numerical methods

2.5.1 Simpson's rule

We have:

$$\int_a^b f(x)dx \approx$$

$$(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 4f(x_{2n-1}) + f(x_{2n}))\frac{h}{3}$$

Where we break up the interval $[a,b]$ into $2n$ parts: $x_i = a + ih$, $h = \frac{b-a}{2n}$. The error is:

$$\le \frac{1}{180}h^4(b-a)\max_{x\in[a,b]}|f^{(4)}(x)|$$

3 Graph Algorithms

3.1 LCA in $O(\log n)$

```
const int lg=18;

void dfs(int nod, int t, int ad){
  f[nod][0]=t,d[nod]=ad;
  for (int i=1; i<=lg; i++)
    f[nod][i]=f[f[nod][i-1]][i-1];

  for (int nxt : g[nod])
    if (nxt!=t)
      dfs(nxt,nod,ad+1);
}

int lca(int x, int y){
  if (d[x]>d[y]) swap(x,y);
  for (int i=0; i<=lg; i++)
    if ((d[y]-d[x])&(1<<i))
      y=f[y][i];

  if (x==y) return x;
  for (int i=lg; i>=0; i--)
    if (f[x][i]!=f[y][i])
      x=f[x][i],y=f[y][i];

  return f[x][0];
}
```

3.2 Max-Flow - Edmonds-Karp - $O(N \cdot M^2)$

```
struct MaxFlow{
  vector<vi> c,f,g;
  vi tt;
  int s,t,N;

  MaxFlow(int N, int s, int t){
    this->N = N, this->s = s, this->t = t;

    c.resize(N),f.resize(N),g.resize(N);
    for (int i=0; i<N; i++)
      c[i].resize(N),f[i].resize(N);

  }

  int getmaxflow(){
    int res=0;
    for (int i=0; i<N; i++)
      fill(all(f[i]),0);

    while (bfs()){
      int nod;
      for (nod=0; nod<N; nod++){
        if (c[nod][t]-f[nod][t]==0 || tt[nod]==-1)
    continue;

        tt[t]=nod;
        int cr,fmin=(1<<30);
        for (cr=t; cr!=s; cr=tt[cr])
          fmin=min(fmin,c[tt[cr]][cr]-f[tt[cr]][cr]);

        res+=fmin;

        for (cr=t; cr!=s; cr=tt[cr]){
          f[tt[cr]][cr]+=fmin;
          f[cr][tt[cr]]-=fmin;
        }


      }
    }

    return res;
  }

  void add_uedge(int from, int to, int cap){
    g[from].pb(to);
    g[to].pb(from);
    c[from][to]+=cap;
  }

  void add_bedge(int from, int to, int cap){
    g[from].pb(to);
    g[to].pb(from);

    c[from][to]+=cap;
    c[to][from]+=cap;
  }
  void print(){
    cout << N << "\n";
    for (int i=0; i<N; i++, cout << "\n")
      for (int j=0; j<N; j++)
        cout << c[i][j] << " ";
  }
  bool bfs(){
    tt.clear(),tt.resize(N,-1);
    int *q = new int[N+10],K=0,i;
    bool *v = new bool[N+10];

    for (i=0; i<N; i++) v[i]=0;

    q[K++]=s; v[s]=1;
    bool ok=0;
    for (i=0; i<K; i++){
      int nod = q[i];

      for (int nxt : g[nod]){
        if (c[nod][nxt]-f[nod][nxt]>0 && !v[nxt]){
          if (nxt==t){
            ok=1;
            continue;
          }

          tt[nxt]=nod;
          q[K++]=nxt;
          v[nxt]=1;
        }
      }
    }

    delete[] q;
    delete[] v;
    return ok;
  }
};
```

3.3 Max Flow - Dinic -$O(N^2 \cdot M)$

```
struct MaxFlow{
  vector<vi> c,f,g;
  ll s,t,N;
  vi ptr,d;
  MaxFlow(int N, int s, int t){
    this->N = N, this->s = s, this->t = t;

    c.resize(N),f.resize(N),g.resize(N);
    for (int i=0; i<N; i++)
      c[i].resize(N),f[i].resize(N);

  }

  ll getmaxflow(){
    ll res=0;
    for (int i=0; i<N; i++)
      fill(all(f[i]),0);

    while (bfs()){

      ptr.clear();
      ptr.resize(N,0);

      while (ll pushed=dfs(s,(1LL<<60)))
        res+=pushed;
```

```
    }

    return res;
  }

  ll dfs(int nod, ll flow){
    if (!flow) return 0;
    if (nod==t) return flow;

    for (ll &i=ptr[nod]; i<g[nod].size(); i++){
      int nxt=g[nod][i];
      if (d[nxt]!=d[nod]+1) continue;

      ll pushed=dfs(nxt,min(flow,c[nod][nxt]-f[nod][
          nxt]));
      if (pushed){
        f[nod][nxt]+=pushed;
        f[nxt][nod]-=pushed;
        return pushed;
      }
    }

    return 0;
  }

  void add_uedge(int from, int to, ll cap){
    g[from].pb(to);
    g[to].pb(from);
    c[from][to]+=cap;
  }

  void add_bedge(int from, int to, ll cap){
    c[from][to]+=cap;
    c[to][from]+=cap;
    g[from].pb(to);
    g[to].pb(from);
  }
  void print(){
    cout << N << "\n";
    for (int i=0; i<N; i++, cout << "\n")
      for (int j=0; j<N; j++)
        cout << c[i][j] << " ";
  }
  bool bfs(){

    d.clear(),d.resize(N,-1);
    int *q = new int[N+10],K=0,i;

    q[K++]=s; d[s]=0;
    for (i=0; i<K; i++){
      int nod = q[i];

      for (int nxt : g[nod]){

        if (c[nod][nxt]-f[nod][nxt]>0 && d[nxt]==-1){
          d[nxt]=d[nod]+1;
          q[K++]=nxt;
        }
      }
    }

    delete[] q;

    return d[t]!=-1;
  }
};
```

3.4 Min Cost Max Flow

```
const int inf = (1<<29);

struct MCMaxFlow{
  struct edge{
    int to,c,cst;
    edge(int to=0, int c=0, int cst=0) : to(to), c(c),
     cst(cst) {}
    edge(){}
  };

  vector<edge> e;
  set<pii> st;
  vi di;
  vector<vi> g;

  int n,S,F,flow,cflow;

  MCMaxFlow(int n, int S, int F){
    this->n=n;
    g.resize(n);
    di.resize(n);
    this->S=S,this->F=F;
  }

  void add_edge(int x, int y, int cp, int cst){
    edge e1(y,cp,cst);
    edge e2(x,0,-cst);
    g[x].pb(e.size());
    e.pb(e1);
    g[y].pb(e.size());
    e.pb(e2);
  }

  bool dijkstra(){
    vi d(n),p(n),pe(n);
    fill(all(d),inf);
    d[S]=0;
    st.clear();
    st.insert({0,S});
    vi real_d(n);

    while (!st.empty()){
      pii cr = *st.begin();
      st.erase(st.begin());

      int nod=cr.se;
      real_d[nod]=d[nod]-di[nod];

      for (int id : g[nod]){
        int nxt = e[id].to;
        int cst=d[nod]+e[id].cst-di[nod]+di[nxt];

        if (e[id].c && cst<d[nxt]){
          if (d[nxt]!=inf) st.erase({d[nxt],nxt});
          d[nxt] = cst;
          p[nxt]=nod;
          pe[nxt]=id;
          st.insert({d[nxt],nxt});
        }
      }
    }

    di = real_d;
    if (d[F] == inf) return 0;

    int minv = inf,cr=0;
    for (cr=F; cr!=S; cr=p[cr])
      minv=min(minv,e[pe[cr]].c);

    flow+=minv;
    cflow += minv*real_d[F];

    for (cr = F; cr!=S; cr=p[cr]){
      e[pe[cr]].c-=minv;
      e[pe[cr]^1].c+=minv;
    }

    return 1;
  }

  int get_flow(){
    flow=cflow=0;
    bellman();

    while (dijkstra()) ;
    return flow;
```

```
  }

  void bellman(){
    fill(all(di),inf);
    di[S]=0;

    vi q,nq;
    vi v(n,0);
    q.pb(S);

    for (int i=1; i<=n; i++){
      nq.clear();
      for (int x : q){
        for (int id : g[x]){
          int nxt = e[id].to;
          if (e[id].c && e[id].cst+di[x]<di[nxt]){
            di[nxt]=e[id].cst+di[x];

            if (!v[nxt]) nq.pb(nxt);
            v[nxt]=1;
          }
        }
      }

      q.clear();
      for (int x : nq){
        v[x]=0;
        q.pb(x);
      }
    }

  }
};
```

3.5 Heavy-Light Decomposition

```
int N,M,a[100100],sz[100100],L,d[100100],lid[100100],
    pl[100100]; vi g[100100];
int pos[100100],off[100100];

int t[800100];

vi lant[100100];

void bdfs(int nod, int f, int h){
  d[nod]=h;
  sz[nod]=1;

  if ((nod!=1 && g[nod].size()==1) || N==1){
    lid[nod]=++L;
    lant[L].pb(nod);
    return;
  }

  int bst=0;
  for (int nxt : g[nod]){
    if (nxt==f) continue;

    bdfs(nxt,nod,h+1);
    if (sz[bst]<sz[nxt]) bst=nxt;
    sz[nod]+=sz[nxt];
  }

  lid[nod]=lid[bst];
  lant[lid[nod]].pb(nod);

  for (int nxt : g[nod])
    if (nxt!=f && nxt!=bst) pl[lid[nxt]]=nod;
}



void upd(int nod, int l, int r, int off, int p, int
    val){
  if (l==r){
    t[nod+off]=val;
    return ;
  }

  int mid=(l+r)/2;
  if (p<=mid) upd(nod*2,l,mid,off,p,val);
  else upd(nod*2+1,mid+1,r,off,p,val);
  t[nod+off]=max(t[nod*2+off],t[nod*2+1+off]);
}

void upd(int x, int val){
  upd(1,1,lant[lid[x]].size(),off[lid[x]],pos[x],val);
}

int query(int nod, int l, int r, int ql, int qr, int
    off){
  if (ql<=l && r<=qr) return t[nod+off];

  int mid=(l+r)/2,lv=0,rv=0;
  if (ql<=mid) lv=query(nod*2,l,mid,ql,qr,off);
  if (mid<qr) rv=query(nod*2+1,mid+1,r,ql,qr,off);

  return max(lv,rv);
}

void build(){
  int i,j;

  bdfs(1,0,1);
  int crsz=0;
  for (i=1; i<=L; i++){
    reverse(all(lant[i]));
    off[i]=crsz;
    for (j=1; j<=(int)lant[i].size(); j++){
      pos[lant[i][j-1]]=j;
      upd(lant[i][j-1],a[lant[i][j-1]]);
    }

    crsz+=4*lant[i].size();
  }
}

int query(int x, int y){
  int res=0;
  while (x!=-1){

    if (d[x]>d[y]) swap(x,y);

    if (lid[x]==lid[y]){
      res=max(res,query(1,1,lant[lid[x]].size(),pos[x
    ],pos[y],off[lid[x]]));
      x=-1;
    }
    else {
      if (d[pl[lid[x]]]<d[pl[lid[y]]]) swap(x,y);

      res=max(res,query(1,1,lant[lid[x]].size(),1,pos[
    x],off[lid[x]]));
      x=pl[lid[x]];
    }
  }

  return res;
}
```

3.6 Max Bipartite matching - $O(N\sqrt{N})$

```
  int N,M,E,l[10100],r[10100]; vector<int> g[10100];
  bool v[10100];

  bool try_match(int nod){
    if (v[nod]) return 0;
    v[nod]=1;

    int i,nd;
    for (i=0; i<g[nod].size(); i++){
      nd=g[nod][i];
      if (!r[nd] || try_match(r[nd])){
        l[nod]=nd;
        r[nd]=nod;
        return 1;
```

```
    }
  }

  return 0;
}

//inside code
bool ex=1;
while (ex){
  ex=0;
  memset(v,0,sizeof(v));
  for (i=1; i<=N; i++)
    if (!l[i])
      ex|=try_match(i);
}
```

3.7 Centroid Decomposition

```
int N,K,sz[100010],cnt[100010]; vi g[100010];
bool v[100010]; ll res;

void dfs_sz(int nod, int f){
  sz[nod]=1;
  for (int nxt : g[nod]){
    if (v[nxt] || f==nxt) continue;
    dfs_sz(nxt,nod);
    sz[nod]+=sz[nxt];
  }
}

void dfs(int nod, int f, int d, bool add){
  if (d>K) return;

  if (add) cnt[d]++;
  if (!add) res+=cnt[K-d];
  for (int nxt : g[nod])
    if (!v[nxt] && nxt!=f) dfs(nxt,nod,d+1,add);
}

void calc(int nod){
  int i;

  cnt[0]=1;
  for (int nxt : g[nod])
    if (!v[nxt]){
      dfs(nxt,nod,1,0);
      dfs(nxt,nod,1,1);
    }

  for (i=0; i<=sz[nod]; i++)
    cnt[i]=0;
}

void decomp(int nod){
  dfs_sz(nod,-1);

  int tot=sz[nod];
  bool f=0;
  while (!f){
    f=1;

    for (int nxt : g[nod])
      if (!v[nxt] && 2*sz[nxt]>tot){
        f=0;
        sz[nod]-=sz[nxt];
        sz[nxt]+=sz[nod];
        nod=nxt;
        break;
      }
  }

  calc(nod);
  v[nod]=1;

  for (int nxt : g[nod])
    if (!v[nxt])
      decomp(nxt);
  }
```

3.8 Euler tour

```
void euler(){
  int nod;
  T=1,st[1]=1;

  while (T){
    nod=st[T];
    while (cr[nod]<g[nod].size())
      if (!ve[g[nod][cr[nod]].ind]) break;
      else cr[nod]++;

      if (cr[nod]<g[nod].size()){
        ve[g[nod][cr[nod]].ind]=1;
        st[++T]=g[nod][cr[nod]].d;
      }
      else{
        ans[++K]=nod;
        T--;
      }
    }
}
```

3.9 Biconnected components

```
void cache_bc(const int x, const int y)
{
  vector <int> con; int tx, ty;
  do {
    tx = stk.top().first, ty = stk.top().second;
    stk.pop();
    con.push_back(tx), con.push_back(ty);
  }
  while (tx != x || ty != y);
  C.push_back(con);
}

void DF(const int n, const int fn, int number)
{
  vector <int>::iterator it;

  dfn[n] = low[n] = number;
  for (it = adj[n].begin(); it != adj[n].end(); ++
    it) {
    if (*it == fn)  continue ;
    if (dfn[*it] == -1) {
      stk.push( make_pair(n, *it) );
      DF(*it, n, number + 1);
      low[n] = Min(low[n], low[*it]);
      if (low[*it] >= dfn[n])
        cache_bc(n, *it);
    }
    else
      low[n] = Min(low[n], dfn[*it]);
  }
}

int main(void)
{
  int n;
  read_in(adj, n);
  dfn.resize(n + 1), dfn.assign(n + 1, -1);
  low.resize(n + 1);
  DF(1, 0, 0);

  ofstream out(oname);
  out << C.size() << "\n";
  for (size_t i = 0; i < C.size(); ++ i) {
    sort(C[i].begin(), C[i].end());
    C[i].erase(unique(C[i].begin(), C[i].end()), C[i
    ].end());
    for (size_t j = 0; j < C[i].size(); ++ j)
      out << C[i][j] << " ";
    out << "\n";
  }
```

4 Data structures

4.1 Treap

```
struct Treap{
  ll x,p,sum,sumi;
  int sz;
  Treap *l,*r;

  Treap(ll x, ll p, Treap* l, Treap* r, int sz);
  void upd();


} *nil=new Treap(0,0,nullptr,nullptr,0);

typedef Treap* tp;
tp root = nil;

void Treap::upd(){
  if (this==nil) return;

  sum=x+l->sum+r->sum;
  sz=1+l->sz+r->sz;
  sumi=l->sumi+1LL*(l->sz+1)*x+r->sumi+(l->sz+1)*r->
    sum;
}

Treap::Treap(ll x, ll p, tp l, tp r, int sz=1){
  this->x = x, this->p = p;
  this->sum=this->sumi=x;
  this->l = l, this->r = r;
  this->sz=sz;
}

void split(tp root, int x, tp &L, tp &R){
  if (root==nil){
    L=R=nil;
    return;
  }

  root->upd();

  if (root->x<=x){
    split(root->r,x,root->r,R);
    L=root;
    L->upd();
  }
  else {
    split(root->l,x,L,root->l);
    R=root;
    R->upd();
  }
}

tp merge(tp l, tp r){
  if (l==nil || r==nil)
    return (l!=nil ? l : r);

  if (l->p>r->p){
    l->r=merge(l->r,r);
    l->upd();
    return l;
  }
  else {
    r->l=merge(l,r->l);
    r->upd();
    return r;
  }

}

void insert(tp &root, tp nod){
  if (root==nil){
    root = nod;
    nod->upd();
    return;
  }
  if (root->p<nod->p){
    split(root,nod->x,nod->l,nod->r);
    root=nod;
  }
  else if (root->x>nod->x) insert(root->l,nod);
  else insert(root->r,nod);

  root->upd();
}

void del(tp &root, ll x){
  if (root==nil) return;

  if (root->x==x){
    tp cr = root;
    root = merge(root->l, root->r);

    delete cr;
    return;
  }
  else if (root->x>x) del(root->l,x);
  else del(root->r,x);

  root->upd();
}
```

4.2 Persistent Segment Tree

```
  struct tree{
    int l,r,s;
    tree (int l=0, int r=0, int s=0) : l(l), r(r), s(s
      ) {}
  };

  int N,Q,a[300010],T,rt[300010]; tree t[7000000];

  int upd(int nod, int l, int r, int p){
    tree &cr=t[++T];
    cr=t[nod];

    if (l==r){
      cr.s++;
      cr.l=cr.r=0;
      return T;
    }

    int mid=(l+r)/2,id=T;
    if (p<=mid) cr.l=upd(t[nod].l,l,mid,p);
    else cr.r=upd(t[nod].r,mid+1,r,p);
    cr.s=t[cr.l].s+t[cr.r].s;
    return id;
  }

  int query(int nod, int l, int r, int ql, int qr){
    if (nod==0) return 0;
    if (ql<=l && r<=qr) return t[nod].s;

    int mid=(l+r)/2,res=0;
    if (ql<=mid) res+=query(t[nod].l,l,mid,ql,qr);
    if (mid<qr) res+=query(t[nod].r,mid+1,r,ql,qr);
    return res;
  }

  int find_kth(int p, int q, int l, int r, int k){
    if (l==r) return l;

    int mid=(l+r)/2;
    int totl=(t[t[p].l].s-t[t[q].l].s);
    if (totl>=k) return find_kth(t[p].l,t[q].l,l,mid,k
      );
    else return find_kth(t[p].r,t[q].r,mid+1,r,k-totl)
      ;
  }
```

4.3 2D Segment Tree

Basically use treaps for the second dimension.

```
/* insert treap implementation here */

struct tree2d{
  tree2d *l, *r;
  Treap* root;

  tree2d(){
    root=nil;
  }
} *root;


ll getmxv(tp root, int x1, int x2, int l, int r, int
    k){
  ll res=-inf;
  if (root==nil || r<x1 || l>x2) return res;

  if (x1<=l && r<=x2) return root->mv[k];

  if (root->x<x1) res=getmxv(root->r,x1,x2,root->x
    +1,r,k);
  else if (root->x>x2) res=getmxv(root->l,x1,x2,l,
    root->x-1,k);
  else{
    res=root->v[k];

    ll lres=getmxv(root->l,x1,x2,l,root->x-1,k);
    ll rres=getmxv(root->r,x1,x2,root->x+1,r,k);

    res=max(res,max(lres,rres));
  }

  return res;
}

void upd2d(tree2d *&nod, int l, int r, int x, int y)
    {
  if (nod==NULL) nod=new tree2d();

  //change this in case the value can be deleted
  ins(nod->root,new Treap(y,rnd()+1,nil,nil,x,y));

  if (l==r)
    return;

  int mid=(l+r)/2;
  if (x<=mid) upd2d(nod->l,l,mid,x,y);
  else upd2d(nod->r,mid+1,r,x,y);

}

ll query2d(tree2d *nod, int l, int r, int ql, int qr
    , int y1, int y2, int k){
  if (nod==NULL) return -inf;
  if (ql<=l && r<=qr) return getmxv(nod->root,y1,y2
    ,0,C, k);

  int mid=(l+r)/2; ll res=-inf;
  if (ql<=mid) res=max(res,query2d(nod->l,l,mid,ql,
    qr,y1,y2,k));
  if (mid<qr) res=max(res,query2d(nod->r,mid+1,r,ql,
    qr,y1,y2,k));
  return res;
}
```

4.4 Merging Segment Trees

```
struct stree{
  stree *l,*r;
  int sz;

  stree();
} *nil = new stree();
typedef stree* st;
stree::stree(){
  l=r=nil;
  sz=0;
}

void insert(st &nod, int l, int r, int p){
  cnt++;
  if (nod==nil) nod=new stree();

  if (l!=r){
    int mid=(l+r)/2;
    if (p<=mid) insert(nod->l,l,mid,p);
    else insert(nod->r,mid+1,r,p);
  }

  nod->sz++;
}

st merge(st &a, st b){
  cnt++;
  if (a==nil || b==nil)
    return (a!=nil ? a : b);

  a->sz+=b->sz;
  a->l=merge(a->l,b->l);
  a->r=merge(a->r,b->r);

  delete b;
  return a;
}

void split(st &nod, int l, int r, int p, st &rn){
  cnt++;
  if (nod==nil || l==r){
    rn=nil;
    return;
  }

  int mid=(l+r)/2;
  rn=new stree();

  if (p<=mid){
    split(nod->l,l,mid,p,rn->l);
    rn->r=nod->r;
    nod->r=nil;
  }
  else
    split(nod->r,mid+1,r,p,rn->r);

  rn->sz=rn->l->sz+rn->r->sz;
  nod->sz-=rn->sz;
}

int query(st nod, int l, int r, int ql, int qr){
  cnt++;
  if (nod==nil) return 0;
  if (ql<=l && r<=qr) return nod->sz;

  int mid=(l+r)/2,res=0;
  if (ql<=mid) res+=query(nod->l,l,mid,ql,qr);
  if (mid<qr) res+=query(nod->r,mid+1,r,ql,qr);
  return res;
}
```

4.5 Binary Indexed Tree

```
int N,M,tree[100010];

int sum(int r){
  int res=0;
  while (r){
    res+=tree[r];
    r-=(r&-r);
  }
  return res;
}

int get_sum(int l,int r){
```

```
    return sum(r)-sum(l-1);
  }

  void update(int ind, int val){
    while (ind<=N){
      tree[ind]+=val;
      ind+=(ind&-ind);
    }
  }

  int find_min(int sum){
    int lg=0,p=0;
    for (lg=0; (1<<(lg+1))<=N; lg++) ;

    for (; lg>=0; lg--){
      if (p+(1<<lg)<=N)
        if (sum>=tree[p+(1<<lg)]){
          p+=(1<<lg), sum-=tree[p];
          if (!sum) return p;
        }
    }
    return -1;
  }
```

4.6 SQRT Decomposition

```
struct block{
  vector<ll> arr,nxt,nrpasi,fnxt,nxtb;
  int lzadd=0,id;

  int getp(int p){
    if (p==-1) return -1;
    return sz*id+p;
  }

  void init(){
    int sz = arr.size();
    //the elements should already be pushed to array
    //do necessary initialization here for any
    additional operations

  }

 void unlz(){
    if (lzadd==0) return;
    int i;
    for (i=0; i<(int)arr.size(); i++)
      arr[i]+=lzadd;
    lzadd=0;
  }

  //augment the structure with addtional operations
  //call unlz() when needed
}
int bid(int x){
  return x/sz;
}

int bpos(int x){
  return x%sz;
}


 cin >> N >> Q;

 int i,j;
 sz=102;
 for (i=0; i<N; i++) cin >> a[i];

 if (sz*sz<N){
   while (sz*sz<=N) sz++;
   sz--;
 }

 for (i=0; i<N; i+=sz){
   int ind = i/sz;
   for (j=i; j<min(i+sz,N); j++)
     b[ind].arr.pb(a[j]);
   b[ind].init();
   b[ind].id = ind;
   b[ind].calc_nxt();
   if (ind) b[ind-1].calc_nxtb(b[ind]);
 }

 //an example of how an update to a range should look
    like
 //NOTE: this is a rough sketch
 //the upd function updates a range of elements inside
     a block
  int l,r; ll x;

    cin >> l >> r >> x;
    l--,r--;
    int idl=bid(l),idr=bid(r);

    if (idl==idr){
      b[idl].upd(l,r,x);
    }
    else {
      b[idl].upd(l,(idl+1)*sz-1,x);
      b[idr].upd(idr*sz,r,x);

      for (i=idl+1; i<idr; i++)
        b[i].lzadd+=x;

    }
```

4.7 Mo's Algorithm

```
  void mvr(int &r, int pos){
    while (r!=pos){
      if (r<pos){
        r++;
        cnte[p[r-1]]++;
        cnt[p[r]]++;
        res+=cnte[p2[r][0]];
      }
      else{
        res-=cnte[p2[r][0]];
        cnt[p[r]]--;
        cnte[p[r-1]]--;
        r--;
      }
    }
  }

  void mvl(int &l, int pos){
    while (l!=pos){
      if (l>pos){
        l--;
        cnte[p[l-1]]++;
        cnt[p[l]]++;
        res+=cnt[p2[l-1][1]];
      }
      else{
        res-=cnt[p2[l-1][1]];
        cnte[p[l-1]]--;
        cnt[p[l]]--;
        l++;
      }
    }
  }

  //main code

  while (sr*sr<=N)
    sr++;
  sr--;
  cin >> Q;
  for (i=1; i<=Q; i++){
    cin >> q[i].l >> q[i].r;
    q[i].id=i;
  }
```

```
  sort(q+1,q+Q+1,[sr](query A, query B) { return (A.l/
    sr==B.l/sr ? A.r<B.r : A.l/sr<B.l/sr);});

  int l=q[1].l,r=q[1].r;
  for (i=l; i<=r; i++){
    cnte[p[i-1]]++;
    cnt[p[i]]++;
    res+=cnte[p2[i][0]];
  }

  for (i=1; i<=Q; i++){
    if (r<q[i].r) mvr(r,q[i].r),mvl(l,q[i].l);
    else mvl(l,q[i].l),mvr(r,q[i].r);

    ans[q[i].id]=res;
  }
```

5 Transformations

5.1 Fast Fourier Transform - $O(n \log n)$

```
const ld pi = acos(-1);
typedef complex<double> base;

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

void fft(vector<base> &a, bool inv){
  int lg=1,sz=a.size(),i,j;
  while ((1<<lg)<sz) lg++;

  for (i=0; i<sz; i++)
    if (i<rev(i,lg)) swap(a[i],a[rev(i,lg)]);

  for (int len=2; len<=sz; len<<=1){
    ld ang = 2*pi/len * (inv ? -1 : 1);
    base wlen(cos(ang),sin(ang));

    for (i=0; i<sz; i+=len){
      base w(1,0);
      for (j=0; j<len/2; j++){
        base u=a[i+j],v=w*a[i+len/2+j];
        a[i+j]=u+v;
        a[i+len/2+j]=u-v;

        w*=wlen;
      }
    }
  }

  if (inv)
    for (i=0; i<sz; i++)
      a[i]/=sz;
}

vi conv(vi &a, vi &b){
  vector<base> na,nb,nc;
  for (int x : a)
    na.pb(base(x));
  for (int x : b)
    nb.pb(base(x));

  int sz=2*max(a.size(),b.size());
  int lg=1;
  while ((1<<lg)<sz) lg++;
  sz=1<<lg;

  na.resize(sz),nb.resize(sz),nc.resize(sz);
  fft(na,0),fft(nb,0);

  for (int i=0; i<sz; i++)
    nc[i]=na[i]*nb[i];
  fft(nc,1);

  vi c(a.size()+b.size()-1);
  for (int i=0; i<(int)c.size(); i++)
    c[i]=(int)(nc[i].real()+0.5); //be careful

  return c;
}
```

5.2 FFT with big modulo

```
 namespace fft {
 typedef double dbl;

 struct num {
  dbl x, y;
  num() { x = y = 0; }
  num(dbl x, dbl y) : x(x), y(y) { }
 };

 inline num operator+(num a, num b) { return num(a.x +
     b.x, a.y + b.y); }
 inline num operator-(num a, num b) { return num(a.x -
     b.x, a.y - b.y); }
 inline num operator*(num a, num b) { return num(a.x *
     b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
 inline num conj(num a) { return num(a.x, -a.y); }

 int base = 1;
 vector<num> roots = {{0, 0}, {1, 0}};
 vector<int> rev = {0, 1};

 const dbl PI = acosl(-1.0);

 void ensure_base(int nbase) {
  if (nbase <= base) {
   return;
  }
  rev.resize(1 << nbase);
  for (int i = 0; i < (1 << nbase); i++) {
   rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase -
    1));
  }
  roots.resize(1 << nbase);
  while (base < nbase) {
   dbl angle = 2 * PI / (1 << (base + 1));
//    num z(cos(angle), sin(angle));
   for (int i = 1 << (base - 1); i < (1 << base); i++)
     {
    roots[i << 1] = roots[i];
//     roots[(i << 1) + 1] = roots[i] * z;
    dbl angle_i = angle * (2 * i + 1 - (1 << base));
    roots[(i << 1) + 1] = num(cos(angle_i), sin(
    angle_i));
   }
   base++;
  }
 }

 void fft(vector<num> &a, int n = -1) {
  if (n == -1) {
   n = a.size();
  }
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for (int i = 0; i < n; i++) {
   if (i < (rev[i] >> shift)) {
    swap(a[i], a[rev[i] >> shift]);
   }
  }
  for (int k = 1; k < n; k <<= 1) {
   for (int i = 0; i < n; i += 2 * k) {
    for (int j = 0; j < k; j++) {
     num z = a[i + j + k] * roots[j + k];
     a[i + j + k] = a[i + j] - z;
     a[i + j] = a[i + j] + z;
    }
```

```
    }
  }
}

vector<num> fa, fb;

vector<int> multiply(vector<int> &a, vector<int> &b)
   {
 int need = a.size() + b.size() - 1;
 int nbase = 0;
 while ((1 << nbase) < need) nbase++;
 ensure_base(nbase);
 int sz = 1 << nbase;
 if (sz > (int) fa.size()) {
  fa.resize(sz);
 }
 for (int i = 0; i < sz; i++) {
  int x = (i < (int) a.size() ? a[i] : 0);
  int y = (i < (int) b.size() ? b[i] : 0);
  fa[i] = num(x, y);
 }
 fft(fa, sz);
 num r(0, -0.25 / sz);
 for (int i = 0; i <= (sz >> 1); i++) {
  int j = (sz - i) & (sz - 1);
  num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
  if (i != j) {
   fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
  }
  fa[i] = z;
 }
 fft(fa, sz);
 vector<int> res(need);
 for (int i = 0; i < need; i++) {
  res[i] = fa[i].x + 0.5;
 }
 return res;
}

vector<int> multiply_mod(vector<int> &a, vector<int>
   &b, int m, int eq = 0) {
 int need = a.size() + b.size() - 1;
 int nbase = 0;
 while ((1 << nbase) < need) nbase++;
 ensure_base(nbase);
 int sz = 1 << nbase;
 if (sz > (int) fa.size()) {
  fa.resize(sz);
 }
 for (int i = 0; i < (int) a.size(); i++) {
  int x = (a[i] % m + m) % m;
  fa[i] = num(x & ((1 << 15) - 1), x >> 15);
 }
 fill(fa.begin() + a.size(), fa.begin() + sz, num {0,
    0});
 fft(fa, sz);
 if (sz > (int) fb.size()) {
  fb.resize(sz);
 }
 if (eq) {
  copy(fa.begin(), fa.begin() + sz, fb.begin());
 } else {
  for (int i = 0; i < (int) b.size(); i++) {
   int x = (b[i] % m + m) % m;
   fb[i] = num(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fb.begin() + b.size(), fb.begin() + sz, num
   {0, 0});
  fft(fb, sz);
 }
 dbl ratio = 0.25 / sz;
 num r2(0, -1);
 num r3(ratio, 0);
 num r4(0, -ratio);
 num r5(0, 1);
 for (int i = 0; i <= (sz >> 1); i++) {
  int j = (sz - i) & (sz - 1);
  num a1 = (fa[i] + conj(fa[j]));
   num a2 = (fa[i] - conj(fa[j])) * r2;
   num b1 = (fb[i] + conj(fb[j])) * r3;
   num b2 = (fb[i] - conj(fb[j])) * r4;
   if (i != j) {
    num c1 = (fa[j] + conj(fa[i]));
    num c2 = (fa[j] - conj(fa[i])) * r2;
    num d1 = (fb[j] + conj(fb[i])) * r3;
    num d2 = (fb[j] - conj(fb[i])) * r4;
    fa[i] = c1 * d1 + c2 * d2 * r5;
    fb[i] = c1 * d2 + c2 * d1;
   }
   fa[j] = a1 * b1 + a2 * b2 * r5;
   fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz);
  fft(fb, sz);
  vector<int> res(need);
  for (int i = 0; i < need; i++) {
   long long aa = fa[i].x + 0.5;
   long long bb = fb[i].x + 0.5;
   long long cc = fa[i].y + 0.5;
   res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30))
     % m;
  }
  return res;
 }

 vector<int> square_mod(vector<int> &a, int m) {
  return multiply_mod(a, a, m, 1);
 }
};
```

5.3 Number Theoretic Transform

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
  int n = (int) a.size();

  for (int i=1, j=0; i<n; ++i) {
    int bit = n >> 1;
    for (; j>=bit; bit>>=1)
      j -= bit;
    j += bit;
    if (i < j)
      swap (a[i], a[j]);
  }

  for (int len=2; len<=n; len<<=1) {
    int wlen = invert ? root_1 : root;
    for (int i=len; i<root_pw; i<<=1)
      wlen = int (wlen * 1ll * wlen % mod);
    for (int i=0; i<n; i+=len) {
      int w = 1;
      for (int j=0; j<len/2; ++j) {
        int u = a[i+j], v = int (a[i+j+len/2] * 1ll
          * w % mod);
        a[i+j] = u+v < mod ? u+v : u+v-mod;
        a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
        w = int (w * 1ll * wlen % mod);
      }
    }
  }
  if (invert) {
    int nrev = reverse (n, mod);
    for (int i=0; i<n; ++i)
      a[i] = int (a[i] * 1ll * nrev % mod);
  }
}
```

5.4 XOR convolutin ($\oplus$) - $O(n \log n)$

```
const int lg = 16,sz=(1<<16),mod=30011,i2=15006;
vi a(sz);
```

```
vi conv(const vi &a, const vi &b){
  int len,i,k;
  vi c;
  c.insert(c.end(),all(a));
  c.insert(c.end(),all(b));

  for (len=sz; len>1; len>>=1){
    for (i=0; i<2*sz; i+=2*len){
      for (k=0; k<len/2; k++){
        int a=c[i+k],na=c[i+len/2+k];
        int b=c[i+len+k],nb=c[i+3*len/2+k];

        c[i+k]=(a+na)%mod;
        c[i+len/2+k]=(b+nb)%mod;
        c[i+len+k]=(a-na+mod)%mod,c[i+3*len/2+k]=(b-
    nb+mod)%mod;

        // for OR (a+na), a
        // for AND (a+na),na
      }
    }
  }

  for (i=0; i<2*sz; i+=2)
    c[i]=(c[i]*c[i+1])%mod;


  for (len=2; len<=sz; len<<=1){
    for (i=0; i<2*sz; i+=2*len){
      for (k=0; k<len/2; k++){
        int a=c[i+k],b=c[i+len+k];
        c[i+k]=(a+b)*i2%mod;
        c[i+len/2+k]=(a-b+mod)*i2%mod;

        //for OR b, a-b
        //for AND a-b, b
      }
    }
  }

  c.resize(sz);
  return c;
}
```

6 String Algorithms

6.1 Suffix automaton - $O(n)$

Every state represents an equivalence class of substrings. To get the number of different substrings, compute the sum of differences between the len of node, and node.fail.

```
struct state {
  int len, link;
  map<char,int> next;
};

const int MAXLEN = 100000;
state st[MAXLEN*2];
int sz, last;

void sa_init() {
  sz = last = 0;
  st[0].len = 0;
  st[0].link = -1;
  ++sz;
  //clear next in case of multiple automatons
}

void sa_extend (char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p;
  for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].
    link)
    st[p].next[c] = cur;
  if (p == -1)
    st[cur].link = 0;

  else {
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len)
      st[cur].link = q;

    else {
      int clone = sz++;
      st[clone].len = st[p].len + 1;
      st[clone].next = st[q].next;
      st[clone].link = st[q].link;

      for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
        st[p].next[c] = clone;

      st[q].link = st[cur].link = clone;
    }
  }
  last = cur;
}
```

6.2 Aho-Corasick

```
int to[1000010][26],K=1,cnt[1000010],T,fail
    [1000010],pnod[120],root=1;
int q[1000010];


int ins(int &nod, int p, string &s){
  if (nod==0) nod=++K;

  if (p==s.length())
    return nod;

  return ins(to[nod][s[p]-'a'],p+1,s);
}

void calc_fail(){
  T=1,q[1]=root;
  fail[root]=root;

  int i,j;
  for (i=1; i<=T; i++){
    int cr=q[i];

    for (j=0; j<26; j++)
      if (to[cr][j]!=0){
        int nxt=to[cr][j];

        int f=fail[cr];
        while (f!=1 && to[f][j]==0)
          f=fail[f];

        fail[nxt]=f;
        if (to[f][j]!=0 && to[f][j]!=nxt)
          fail[nxt]=to[f][j];


        q[++T]=nxt;

      }
  }
}

void prop(){
  for (int i=K; i>1; i--){
    cnt[fail[q[i]]]+=cnt[q[i]];
  }
}
```

6.3 Suffix Array

```
const int MAXN = 65536;
const int MAXLG = 17;

char A[MAXN];
struct entry {
  int nr[2], p;
} L[MAXN];
int P[MAXLG][MAXN], N, i, stp, cnt;

bool cmp(const entry &a, const entry &b) {
  return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1]) :
    (a.nr[0] < b.nr[0]);
}

int main() {
  gets(A);
  for (N = strlen(A), i = 0; i < N; ++i)
    P[0][i] = A[i] - 'a';
  for (stp = 1, cnt = 1; cnt >> 1 < N; ++stp, cnt
    <<= 1) {
    for (i = 0; i < N; ++i) {
      L[i].nr[0] = P[stp - 1][i];
      L[i].nr[1] = i + cnt < N ? P[stp - 1][i + cnt]
     : -1;
      L[i].p = i;
    }
    sort(L, L + N, cmp);
    for (i = 0; i < N; ++i)
      P[stp][L[i].p] = i > 0 && L[i].nr[0] == L[i -
    1].nr[0] && L[i].nr[1] == L[i - 1].nr[1] ? P[stp
    ][L[i - 1].p] : i;
  }
  return 0;
}
```

6.4 KMP

Find all occurrences of A in B:

```
int i,q=0;
for (i=2; i<=N; i++){
  while (q && A[q+1]!=A[i])
    q--;

  if (A[q+1]==A[i]) q++;
  P[i]=q;
}

q=0;
for (i=1; i<=M; i++){
  while (q && A[q+1]!=B[i])
    q=P[q];

  if (A[q+1]==B[i]) q++;

  if (q==N){
    K++;
    q=P[N];
    if (K<=1000)
      pos[K]=i-N;

  }
}
```

6.5 Manacher's Algorithm

```
void get_pal(string &s, int *q){
int i,N=s.length(),hr=-1,l,p;

for (i=0; i<N; i++){
  l=0;
  if (hr<i) l=0,hr=i;
  else l=q[2*p-i];
    if (i+l<hr){
      q[i]=l;
      continue;
    }
    else l=hr-i;

    while (i+l<N && i-l>=0 && s[i+l]==s[i-l])
      l++;

    l--;
    q[i]=l;
    hr=i+l;
    p=i;
  }
}
```

7 Geometry

7.1 Common structures

```
struct point{
  ld x,y;
  point (ld x=0, ld y=0) : x(x), y(y) {}
};

struct line{
  ld A,B,C;
  line (ld A=0, ld B=0, ld C=0) : A(A), B(B), C(C) {}
};

ld norm(point P){
  return sqrt(P.x*P.x+P.y*P.y);
}
```

7.2 Angle between vectors

$$\cos\alpha = \frac{u \cdot v}{|u||v|}$$

7.3 Line from 2 points

```
line getline(point u, point v){
  line r;
  r.A=u.y-v.y;
  r.B=v.x-u.x;
  r.C=u.x*(v.y-u.y)-u.y*(v.x-u.x);
  return r;
}
```

7.4 Intersection of 2 lines

```
point intersect(line l1, line l2){
  if (l1.A*l2.B==l2.A*l1.B){
    //parallel, or completely intersecting
    return {0,0};
  }
  else{
    ld x=(-l1.C*l2.B+l1.B*l2.C)/(l1.A*l2.B-l1.B*l2.A);
    ld y=(-l1.A*l2.C+l1.C*l2.A)/(l1.A*l2.B-l1.B*l2.A);
    return point(x,y);
  }
}
```

7.5 Distance from point to line

$$d = \frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

If line is given as a pair of points, or distance to ray/segment is required, use:

```
ld dist(point A, point B, point X){
  B.x-=A.x,B.y-=A.y;
  X.x-=A.x,X.y-=A.y;

  ld t=(B.x*X.x+B.y*X.y)/(B.x*B.x+B.y*B.y);
```

```
  //in case of line/ray intersection put conditions on
      t
  point P(t*B.x,t*B.y);
  return sqrt(pow(P.x-X.x,2)+pow(P.y-X.y,2));
}
```

7.6 Bisector

```
point bisector(point X, point A, point B){
  A.x-=X.x,A.y-=X.y;
  B.x-=X.x,B.y-=X.y;

  ld v;
  v=norm(A);
  A.x/=v,A.y/=v;

  v=norm(B);
  B.x/=v,B.y/=v;


  if (abs(A.x*B.x+A.y*B.y+1)<eps) return point(X.x-A.y
      ,X.y+A.x);
  //returns vector
  return point(X.x+(A.x+B.x)/2,X.y+(A.y+B.y)/2);

}
```

7.7 Convex hull

```
int N,K,st[120100]; point a[120100];

 double area(point A, point B, point C){
  return (A.x*B.y+B.x*C.y+C.x*A.y-B.y*C.x-C.y*A.x-A.y*
    B.x);
}

double sdist(point A, point B){
  return (A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y);
}

inline bool cmp(point A, point B){
  return (area(a[1],A,B)==0 ? sdist(a[1],A)<sdist(a
    [1],B) : area(a[1],A,B)>0);
}

int i,ind=1;
for (i=1; i<=N; i++){
    scanf("%lf %lf",&a[i].x,&a[i].y);
    if (a[i].x<a[ind].x) ind=i;
 }

swap(a[1],a[ind]);
sort(a+2,a+N+1,cmp);
a[N+1]=a[1];

K=1,st[1]=1;
for (i=2; i<=N; i++){
  st[++K]=i;
  while (area(a[st[K-1]],a[st[K]],a[i+1])<0) K--;
 }
```

7.8 Testing if point is inside convex polygon - $O(\log n)$

```
bool is_between(point A, point B, point X){
  return (min(A.x,B.x)<=X.x && X.x<=max(A.x,B.x) &&
    min(A.y,B.y)<=X.y && X.y<=max(A.y,B.y));
}

bool inside_triangle(point A, point B, point C, point
    X){
  if (sign(ccw(X,A,B))*sign(ccw(X,B,C))>=0 && sign(ccw
    (X,B,C))*sign(ccw(X,C,A))>=0) return 1;
  return 0;
}


bool is_inside(point &A, vector<point> &poly){
  int l=1,r=poly.size()-1,mid;
  if (ccw(poly[0],poly[1],A)<0) return 0;
  while (l<r){
    mid=(l+r+1)/2;
    if (ccw(poly[0],poly[mid],A)>=0) l=mid;
    else r=mid-1;
  }

  if (r==(int)poly.size()-1) return (ccw(poly[0],poly[
    r],A)==0 && is_between(poly[0],poly[r],A));
  return inside_triangle(poly[0],poly[r],poly[r+1],A);
}
```

7.9 Signed area of a polygon

$$A = \frac{1}{2}\sum_{i=1}^{n}(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

7.10 Closest pair of points

```
  int N; point a[100100];
  point v[100100]; int K;

  double mind(int l, int r){
    if (r-l+1<=1) return (1LL<<30);
    if (r-l+1==2){
      if (a[l].y>a[r].y) swap(a[l],a[r]);
      return dist(a[l],a[r]);
    }

    double ans=(1LL<<30);
    int mid=(l+r)/2,mx=a[mid].x;
    ans=min(ans,mind(l,mid));
    ans=min(ans,mind(mid+1,r));



    int i,j;
    merge(a+l,a+mid+1,a+mid+1,a+r+1,v,[](point A,
        point B){return A.y<B.y; });
    memcpy(a+l,v,(r-l+1)*sizeof(point));

    vector<point> med;
    for (i=l; i<=r; i++)
      if (abs(a[i].x-mx)<=ans)
        med.pb(a[i]);


    for (i=0; i<(int)med.size(); i++)
      for (j=i+1; j<(int)med.size() && med[j].y-med[i
          ].y<=ans; j++)
        ans=min(ans,dist(med[i],med[j]));


    return ans;
  }
```

8 Misc

8.1 2-SAT

```
const int MAXN=100010;
int N,M,st[2*MAXN+10],K;
vector<int> g[2*MAXN+10],gc[2*MAXN+10];
int comp[2*MAXN+10],nrC;


int id(int x){
  if (x>0) return 2*x-1;
  return 2*(-x);
}
```

```
void dfs1(int nod){
  comp[nod]=1;
  for (int nxt : g[nod])
    if (!comp[nxt]) dfs1(nxt);
  st[++K]=nod;
}

void dfs2(int nod){
  comp[nod]=nrC;
  for (int nxt : gc[nod])
    if (!comp[nxt]) dfs2(nxt);
}

int main(){
  // N - number of variables, M-clauses

  cin >> N >> M;

  int i,x,y;
  for (i=1; i<=M; i++){
    fcn >> x >> y;
    g[id(-x)].push_back(id(y));
    g[id(-y)].push_back(id(x));

    gc[id(y)].push_back(id(-x));
    gc[id(x)].push_back(id(-y));
  }

  for (i=1; i<=2*N; i++)
    if (!comp[i]) dfs1(i);

  memset(comp,0,sizeof(comp));
  for (i=2*N; i>0; i--)
    if (!comp[st[i]]) nrC++,dfs2(st[i]);

  for (i=1; i<=N; i++)
    if (comp[id(i)]==comp[id(-i)]){
      fout << -1 << "\n";
      return 0;
    }

  for (i=1; i<=N; i++)
    fout << (comp[id(i)]>comp[id(-i)]) << " ";
  fout << "\n";
  return 0;
}
```

8.2 Custom set/multiset comparator

```
  struct lex_compare {
    bool operator() (const int64_t& lhs, const int64_t
      & rhs) const {
      stringstream s1, s2;
      s1 << lhs;
      s2 << rhs;
      return s1.str() < s2.str();
    }
  };

  //use like this
  set<int64_t, lex_compare> s;
```

8.3 Parsing expressions

```
  const long MAX = 100010;
  char S[MAX], *p=S;

  long termen();
  long factor();

  long eval() {
    long r = termen();
    while ( *p=='+' || *p=='-' ) {
      switch ( *p ) {
        case '+':
          ++p;
          r += termen();
          break;
        case '-':
          ++p;
          r -= termen();
          break;
      }
    }
    return r;
  }

  long termen() {
    long r = factor();
    while ( *p=='*' || *p=='/' ) {
      switch ( *p ) {
        case '*' :
          ++p;
          r *= factor();
          break;
        case '/':
          ++p;
          r /= factor();
          break;
      }
    }
    return r;
  }

  long factor() {
    long r=0;
    if ( *p == '(' ) {
      ++p;
      r = eval();
      ++p;
    } else {
      while ( *p>='0' && *p<='9' ) {
        r = r*10 + *p - '0';
        ++p;
      }
    }
    return r;
  }
```