Team notebook

CU BadToTheBone

September 10, 2020

Contents

1 adhoc

2 combinatorics

3 data-structure

3.1 ordered-set

```
/**
 * Policy Based Data Structure (PBDS)
 *
 * The following code works like a set
    and a multiset respectively.
 * ordered_set is used to find the
    number of elements less or greater
    than a certain element, and the
    relative position of the element.
 * In order to declare a multiset, you
    should declare a variable of type
    ordered_set< array<int, 2> >.
 */

#include "../utility/template.cpp"

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

template<class T> using ordered_set =
    tree<T, null_type, less<T>,
    rb_tree_tag,
    tree_order_statistics_node_update>;

int main() {
  ordered_set<int> ost; // ordered set
  ost.insert(2);
  ost.insert(3);
  ost.insert(3);
  ost.insert(5);

  // find_by_order(x) returns the
     element in position x
  cout << *ost.find_by_order(1) << endl;
     // 3
  cout << *ost.find_by_order(2) << endl;
     // 5

  // order_of_key(x) returns the
     relative position of x in the
     set/multiset.
  cout << ost.order_of_key(4) << endl;
     // 2
  cout << ost.order_of_key(3) << endl;
     // 1

  ordered_set<array<int, 2>> omst; //
     ordered multiset
  omst.insert({2, 1});
  omst.insert({4, 2});
  omst.insert({3, 3});
  omst.insert({4, 4});
  omst.insert({3, 5});

  cout << omst.order_of_key({5, 0}) <<
     endl; // 5
  cout << omst.order_of_key({3, 0}) <<
     endl; // 1

  omst.clear();

  omst.insert({1, 1});
  omst.insert({1, 2});
  omst.insert({2, 3});
  omst.insert({2, 4});

  cout << sz(omst) -
      omst.order_of_key({1, 200}) <<
      endl; // 2
}
```

3.2 segment$_t$ree

```
const int mx = 2e5+123;
ll t[mx*3], a[mx], prop[3*mx];
bool vis[3*mx];

void shift ( int id, int b, int e )
{
    int mid = ( b + e ) >> 1;
    t[id*2] += ( mid-b+1 * prop[id] );
    t[id*2+1] += ( e-(mid+1)+1 *
        prop[id] );

    prop[id*2] += prop[id];
    prop[id*2+1] += prop[id];

    vis[id*2] = vis[id*2+1] = 1;
    prop[id] = vis[id] = 0;
}

void init ( int id, int b, int e )
{
    if ( b == e ) {
        t[id] = a[b];
        return;
    }

    int mid = ( b + e ) >> 1;
```

```
    init ( id*2, b, mid );
    init ( id*2+1, mid+1, e );

    t[id] = t[id*2] + t[id*2+1];
}

void upd ( int id, int b, int e, int i,
    int j, ll val )
{
    if ( b > j || e < i ) return;
    if ( b >= i && e <= j ) {
        t[id] += ( val * e-b+1 );
        prop[id] += ( prop[id] * val );
        vis[id] = 1;
        return;
    }

    if ( vis[id] ) shift ( id, b, e );
    int mid = ( b + e ) >> 1;

    upd ( id*2, b, mid, i, j, val );
    upd ( id*2+1, mid+1, e, i, j, val );

    t[id] = t[id*2] + t[id*2+1];
}

ll ask ( int id, int b, int e, int i,
    int j )
{
    if ( b > j || e < i ) return 0;
    if ( b >= i && e <= j ) return t[id];

    if ( vis[id] ) shift ( id, b, e );
    int mid = ( b + e ) >> 1;

    ll ret1 = ask ( id*2, b, mid, i, j );
    ll ret2 = ask ( id*2+1, mid+1, e, i,
        j );

    return ret1 + ret2;
}
```

3.3 segment$_tree_2D$

```
/*//////////////////////////////
2D Segment Tree
Needs O(16nm) memory!!!
Be carefull writing lx, rx, ly, ry!
//////////////////////////////*/

const int maxn = 2001;
int tree[4*maxn][4*maxn], n, m,
    arr[maxn][maxn];
void buildY(int ndx, int lx, int rx, int
    ndy, int ly, int ry) {
        if(ly == ry) {
                if(lx == rx)
                    tree[ndx][ndy] =
                    arr[lx][ly];
                else tree[ndx][ndy] =
                    tree[ndx*2][ndy] +
                    tree[ndx*2+1][ndy];
                return;
        } int mid = ly + ry >> 1;
        buildY(ndx, lx, rx, ndy*2, ly,
            mid);
        buildY(ndx, lx, rx, ndy*2+1,
            mid+1, ry);
        tree[ndx][ndy] = tree[ndx][ndy*2]
            + tree[ndx][ndy*2+1];
}
void buildX(int ndx, int lx, int rx) {
        if(lx != rx) {
                int mid = lx + rx >> 1;
                buildX(ndx*2, lx, mid);
                buildX(ndx*2+1, mid+1, rx);
        } buildY(ndx, lx, rx, 1, 0, m-1);
}
void updateY(int ndx, int lx, int rx,
    int ndy, int ly, int ry, int y, int
    val) {
        if(ly == ry) {
                if(lx == rx)
                    tree[ndx][ndy] = val;
                else tree[ndx][ndy] =
                    tree[ndx*2][ndy] +
                    tree[ndx*2+1][ndy];
                return;
        } int mid = ly + ry >> 1;
        if(y <= mid) updateY(ndx, lx, rx,
            ndy*2, ly, mid, y, val);
        else updateY(ndx, lx, rx,
            ndy*2+1, mid+1, ry, y, val);
        tree[ndx][ndy] = tree[ndx][ndy*2]
            + tree[ndx][ndy*2+1];
}
void updateX(int ndx, int lx, int rx,
    int x, int y, int val) {
        if(lx != rx) {
                int mid = lx + rx >> 1;
                if(x <= mid)
                    updateX(ndx*2, lx,
                    mid, x, y, val);
```

```
            else updateX(ndx*2+1,
                mid+1, rx, x,y, val);
        } updateY(ndx, lx, rx, 1, 0, m-1,
            y,val);
}

int queryY(int ndx, int ndy, int ly, int
    ry, int y1, int y2) {
        if(ry < y1 || ly > y2) return 0;
        if(y1 <= ly && ry <= y2)
                return tree[ndx][ndy];
        int mid = ly + ry >> 1;
        return queryY(ndx, ndy*2, ly,
            mid, y1, y2) +
                queryY(ndx, ndy*2+1,
                    mid+1, ry, y1, y2);
}
int queryX(int ndx, int lx, int rx, int
    x1, int y1, int x2, int y2) {
        if(rx < x1 || lx > x2) return 0;
        if(x1 <= lx && rx <= x2) {
                return queryY(ndx, 1, 0,
                    m-1, y1, y2);
        } int mid = lx + rx >> 1;
        return queryX(ndx*2, lx, mid,
            x1,y1,x2,y2) +
                queryX(ndx*2+1, mid+1,
                    rx, x1,y1,x2,y2);
}
```

3.4 segment$_u$nion

```
int length_union(const vector<pair<int,
    int>> &a) {
    int n = a.size();
    vector<pair<int, bool>> x(n*2);
    for (int i = 0; i < n; i++) {
        x[i*2] = {a[i].first, false};
        x[i*2+1] = {a[i].second, true};
    }

    sort(x.begin(), x.end());

    int result = 0;
    int c = 0;
    for (int i = 0; i < n * 2; i++) {
        if (i > 0 && x[i].first >
            x[i-1].first && c > 0)
            result += x[i].first -
                x[i-1].first;
        if (x[i].second)
            c--;
        else
            c++;
    }
    return result;
}

int point_union ( vii v )
{
    int req_time = 0;
    sort ( all ( v ) );
    int lastr = 0;

        for (auto s : v){

                if (s.F <= lastr) {
                        req_time += max(0,
                            s.S - lastr);
                        lastr = max(s.S,
                            lastr);
                }

                else {
                        req_time += s.S -
                            s.F + 1;
                        lastr = s.S;
                }
        }

        return req_time;
}
```

3.5 sparse$_t$able

```
int st[MAXN][K];

void preprocess()
{
    for (int i = 1; i <= N; i++)
        st[i][0] = array[i];

    for (int j = 1; j <= K; j++) {
        for (int i = 1; i + (1 << j) <=
            N; i++) {
            st[i][j] = min(st[i][j-1],
                st[i + (1 << (j - 1))][j -
                1]);
        }
    }
}

int getMinimum(int L, int R)
```

```
{
    int j = log2(R - L + 1);
    return min(st[L][j], st[R - (1 << j)
        + 1][j]);
}
```

4 dynamic-programming

4.1 1D1D$_{o}$*ptimization*

```
/// Here l, r is range and p is optimal
    solution
struct node {
    int l, r, p;
    node(){}
    node ( int _l, int _r, int _p ) : l
        (_l), r (_r), p (_p) {

    }
};

node que[mx];

int dp[mx], n, c[mx][mx], a[mx];


/// This function calculates the cost of
    (i, j).
int calc ( int j, int i )
{
    return c[j][i];
}
```

```
/// This function compares if i is
    better ans than j for k
bool cmp ( int i, int j, int k )
{
    int v1 = dp[i] + calc ( i, k ), v2 =
        dp[j] + calc ( j, k );
    return ( v1 <= v2 );
}


/// This function finds the lowest
    position where i is optimal solution
    in node cur
int find ( node cur, int i )
{
    int l = cur.l, r = cur.r+1;
    while ( l < r ) {
        int mid = ( l + r ) >> 1;
        if ( cmp ( i, cur.p, mid ) ) r =
            mid;
        else l = mid+1;
    }

    return r;
}


void solve ()
{
    int s = 1, t = 1;
    dp[0] = 0;
    que[1] = node ( 1, n, 0 ); ///
        Initializing optimal value of all
        index as 0.
```

```
for ( int i = 1; i <= n; i++ ) {
    while ( s < t && que[s].r < i )
        s++; /// Deleting ranges from
        front until we get the range
        where i index lies
    dp[i] = dp[que[s].p] +
        calc(que[s].p, i ); ///
        calculation dp[i]

    if ( cmp ( i, que[t].p, n ) ) {
        /// Checking if i is better
        than the current optimal
        value of last range
        while ( s <= t && cmp ( i,
            que[t].p, que[t].l ) )
            t--; /// Deleting all
            range from back of deque
            where i is better.
        if ( s > t ) que[++t] = node
            ( i+1, n, i ); ///
            Creating new range when
            deque is empty.
        else {
            int pos = find( que[t], i
                ); /// Finding lowest
                position where i is
                optimal solution.
            que[t].r = pos-1;
            que[++t] = node ( pos, n,
                i ); /// Creating new
                range.
        }
    }
}
```

```
}


int main()
{

    cin >> n;
    for ( int i = 1; i <= n; i++ ) cin
        >> a[i];

    for ( int i = 1; i <= n; i++ ) {
        for ( int j = i+1; j <= n; j++ ) {
            cin >> c[i][j];
        }
    }

    solve();
    cout << dp[n] << endl;

    return 0;
}
```

4.2 $\mathbf{divide}_{a}\mathbf{nd}_{c}\mathbf{onquer}_{u}\mathbf{sing}_{k}\mathbf{nuth}$

```
const int mx = 5e3+123;
ll dp[mx][mx], a[mx], cost[mx][mx],
    opt[mx][mx];

int main()
{
    optimize();

    int t;
    cin >> t;

    for ( int tc = 1; tc <= t; tc++ ) {
        int n, k;
        cin >> n >> k;
        for ( int i = 1; i <= n; i++ )
            cin >> a[i];

        for ( int i = 1; i <= n; i++ ) {
            cost[i][i] = a[i];
            for ( int j = i+1; j <= n;
                j++ ) {
                cost[i][j] = cost[i][j-1]
                    | a[j];
            }
        }

        for ( int i = 1; i <= n; i++ ) {
            dp[1][i] = cost[1][i];
            opt[1][i] = 1;
        }

        for ( int i = 1; i <= k; i++ )
            opt[i][n+1] = n;

        int pre = -1;
        for ( int i = 2; i <= k; i++ ) {
            for ( int j = n; j >= 1; j--
                ) {

                int ml = opt[i-1][j];
                int mr = opt[i][j+1];

                if ( pre > mr ) return 0;
                pre = ml;

                dp[i][j] = 0;

                for ( int k = ml; k <= mr;
                    k++ ) {
                    ll d = dp[i-1][k] +
                        cost[k+1][j];

                    if ( d > dp[i][j] ) {
                        dp[i][j] = d;
                        opt[i][j] = k;
                    }
                }
            }
        }

        cout << dp[k][n] << endl;
    }

    return 0;
}
```

4.3 $\mathbf{knuth}_{o}\mathit{ptimization}$

```
const int mx = 1e3+123;
long long dp[mx][mx], c[mx];
int opt[mx][mx];

int main()
{
    optimize();

    ll m, n;
    while ( cin >> m >> n ) {
        mem ( dp, 0 );
        c[n+1] = m;
```

```
    for ( int i = 1; i <= n; i++ )
        cin >> c[i];

    for ( int i = 0; i <= n+1; i++ ) {
        for ( int l = 0; l+i <= n+1;
            l++ ) {
            int r = l + i;

            if ( i < 2 ) {
                dp[l][r] = 0;
                opt[l][r] = l;
                continue;
            }

            int ml = opt[l][r-1];
            int mr = opt[l+1][r];

            dp[l][r] = inf;
            for ( int k = ml; k <= mr;
                k++ ) {
                int d = dp[l][k] +
                    dp[k][r] + c[r] -
                    c[l];
                if ( dp[l][r] > d ) {
                    dp[l][r] = d;
                    opt[l][r] = k;
                }
            }
        }
    }

    cout << dp[0][n+1] << endl;
}

return 0;
}
```

4.4 partition$_{dp_trick}$

```
const int mx = 5e3+123;
int n, num[mx], dp[mx], c[mx][mx], a[mx];

/// Here l, r is range and p is optimal
    solution
struct node {
    int l, r, p;
    node(){}
    node ( int _l, int _r, int _p ) : l
        (_l), r (_r), p (_p) {

    }
};

node que[mx];

/// This function compares if i is
    better ans than j for k
bool cmp ( int i, int j, int k )
{
    int v1 = dp[i] + c[i+1][k], v2 =
        dp[j] + c[j+1][k];
    if ( v1 == v2 ) return num[i] <=
        num[j];
    return ( v1 < v2 );
}

/// This function finds the lowest
    position where i is optimal solution
    in node cur
int find ( node cur, int i )
{
    int l = cur.l, r = cur.r+1;
    while ( l < r ) {
        int mid = ( l + r ) >> 1;
        if ( cmp ( i, cur.p, mid ) ) r =
            mid;
        else l = mid+1;
    }

    return r;
}

int solve ( int mid )
{
    int s = 1, t = 1;
    dp[0] = num[0] = 0;
    que[1] = node ( 1, n, 0 ); ///
        Initilaising optimal value of all
        index as 0.

    for ( int i = 1; i <= n; i++ ) {
        while ( s < t && que[s].r < i )
            s++; /// Deleting ranges from
            front until we get the range
            where i index lies
        dp[i] = dp[que[s].p] +
            c[que[s].p+1][i] + mid; ///
            calculating dp[i] with slop
            mid
        num[i] = num[que[s].p] + 1; ///
            calculating num[i].

        if ( cmp ( i, que[t].p, n ) ) {
            /// Checking if i is better
```

```
            than the current optimal
            value of last range
            while ( s <= t && cmp ( i,
                que[t].p, que[t].l ) )
                t--; /// Deleting all
                range from back of queue
                where i is better.
            if ( s > t ) que[++t] = node
                ( i+1, n, i ); ///
                Creating new range when
                deque is empty.
            else {
                int pos = find( que[t], i
                    ); /// Finding lowest
                    position where i is
                    optimal solution.
                que[t].r = pos-1;
                que[++t] = node ( pos, n,
                    i ); /// Creating new
                    range.
            }
        }
    }

    return num[n];
}

int main()
{
    int k;
    cin >> n >> k;
    for ( int i = 1; i <= n; i++ ) cin
        >> a[i];

    for ( int i = 1; i <= n; i++ ) {
        for ( int j = i; j <= n; j++ )
            cin >> c[i][j];
    }

    int l = 0, r = 3e7+123, ans = 0;

    /// Binary search on slop
    while ( l <= r ) {
        int mid = ( l + r ) >> 1;
        if ( solve ( mid ) <= k ) {
            ans = dp[n] - ( k * mid );
                /// As mid is added in
                dp[n], k times.
            r = mid-1;
        }

        else l = mid+1;
    }

    cout << ans << endl;


    return 0;
}
```

5 fft

5.1 FFT

```
typedef complex<dl> cd;
typedef vector<cd> vcd;

void fft ( vcd &a, bool invert )
{
    int n = sz ( a );

    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 ) {
        dl ang = ( ( 2.0 * PI ) / (dl)len
            ) * ( invert ? -1 : 1 );
        cd wlen ( cos ( ang ), sin ( ang
            ) );

        for ( int i = 0; i < n; i += len
            ) {
            cd  w(1);
            for ( int j = 0; j < ( len >>
                1 ); j++ ) {
                cd u = a[i+j], v = w *
                    a[i+j+(len>>1)];
                a[i+j] = u + v;
                a[i+j+(len>>1)] = u - v;
                w *= wlen;
            }
        }
    }

    if ( invert ) {
```

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

vl mul ( vi a, vi b )
{
    vcd fa ( all ( a ) ), fb ( all ( b )
        );
    int n = 1;
    while ( n < sz ( a ) + sz ( b ) ) {
        n <<= 1;
    }

    fa.resize ( n ), fb.resize ( n );

    fft ( fa, 0 );
    fft ( fb, 0 );

    for ( int i = 0; i < n; i++ ) {
        fa[i] *= fb[i];
    }

    fft ( fa, 1 );

    vl ret(n+1);
    for ( int i = 0; i < n; i++ ) {
        ret[i] = round ( fa[i].real() );
    }

    return ret;
}

int main()
{
    optimize();

    int t;
    cin >> t;

    while ( t-- ) {
        int n;
        cin >> n;
        vi a(n+1), b(n+1);
        for ( int i = 0; i < n+1; i++ )
            cin >> a[i];
        for ( int i = 0; i < n+1; i++ )
            cin >> b[i];

        vl ans = mul ( a, b );

        for ( int i = 0; i < (2*n)+1; i++
            ) cout << ans[i] << " ";
        cout << endl;
    }
    return 0;
}
```

5.2 NTT

```
///   ***   ---   |||        In the name
    of ALLAH     |||   ---   ***   ///


#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<vi> vvi;
typedef vector<vl> vvl;
typedef pair<int,int> pii;
typedef pair<double, double> pdd;
typedef pair<ll, ll> pll;
typedef vector<pii> vii;
typedef vector<pll> vll;
typedef double dl;

#define endl '\n'
#define PB push_back
#define F first
#define S second
#define all(a) (a).begin(),(a).end()
#define rall(a) (a).rbegin(),(a).rend()
#define sz(x) (int)x.size()

const double PI = acos(-1);
const double eps = 1e-9;
const int inf = 2000000000;
const ll infLL = 9000000000000000000;
#define MOD 998244353

#define mem(a,b) memset(a, b, sizeof(a) )
#define sqr(a) ((a) * (a))

#define optimize()
    ios_base::sync_with_stdio(0);cin.tie(0);cou
#define fraction()
    cout.unsetf(ios::floatfield);
    cout.precision(10);
```

```
    cout.setf(ios::fixed,ios::floatfield);
#define file()
    freopen("input.txt","r",stdin);freopen("output.txt","w",stdout);

#define dbg(args...) do {cerr << #args
    << " : "; faltu(args); } while(0)
void faltu () {          cerr << endl;}
template < typename T, typename ...
    hello>void faltu( T arg, const hello
    &... rest) {cerr << arg << '
    ';faltu(rest...);}

ll gcd ( ll a, ll b ) { return __gcd (
    a, b ); }
ll lcm ( ll a, ll b ) { return a * ( b /
    gcd ( a, b ) ); }

inline void normal(ll &a) { a %= MOD; (a
    < 0) && (a += MOD); }
inline ll modMul(ll a, ll b) { a %= MOD,
    b %= MOD; normal(a), normal(b);
    return (a*b)%MOD; }
inline ll modAdd(ll a, ll b) { a %= MOD,
    b %= MOD; normal(a), normal(b);
    return (a+b)%MOD; }
inline ll modSub(ll a, ll b) { a %= MOD,
    b %= MOD; normal(a), normal(b); a -=
    b; normal(a); return a; }
inline ll modPow(ll b, ll p) { ll r = 1;
    while(p) { if(p&1) r = modMul(r, b);
    b = modMul(b, b); p >>= 1; } return
    r; }
inline ll modInverse(ll a) { return
    modPow(a, MOD-2); }
```

```
inline ll modDiv(ll a, ll b) { return
    modMul(a, modInverse(b)); }
int getK ( int m )
{
    for ( int i = 30; i >= 0; i-- ) {
        if ( (m-1) % ( 1 << i ) == 0 )
            return i;
    }
}

int generator (int p) {
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() &&
            ok; ++i)
            ok &= (int)modPow( res, phi /
                fact[i]) != 1;
        if (ok) return res;
    }
    return -1;
}

const int mod = MOD;
```

```
const int K = getK ( mod );
const int root = modPow( generator( mod
    ), ( mod-1 ) / ( 1 << K ) );
const int root_1 = modInverse( root );
const int root_pw = 1 << K;


void fft(vector<int> & a, bool invert) {
    int n = 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)(1LL * wlen *
                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)(1LL *
                    a[i+j+len/2] * 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)(1LL * w * wlen %
                mod);
        }
    }
}

if (invert) {
    int n_1 = modInverse(n);
    for (int & x : a)
        x = (int)(1LL * x * n_1 %
            mod);
}
}

vector<int> multiply(vector<int> const&
    a, vector<int> const& b) {
    vector<int> fa(a.begin(), a.end()),
        fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] = modMul( fa[i], fb[i] );
    fft(fa, true);
    vector<int> result(n);
    for (int i = 0; i < n; i++) {
        result[i] = fa[i];
    }
    return result;
}


int main()
{
    optimize();


    return 0;
}
```

5.3 fft$_s$tring$_m$atching

```
using cd = complex<double>;

int reverse(int num, int lg_n) {
    int res = 0;
    for (int i = 0; i < lg_n; i++) {
        if (num & (1 << i))
            res |= 1 << (lg_n - 1 - i);
    }
    return res;
}

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    int lg_n = 0;
    while ((1 << lg_n) < n)
        lg_n++;

    for (int i = 0; i < n; i++) {
        if (i < reverse(i, lg_n))
            swap(a[i], a[reverse(i,
                lg_n)]);
    }

    for (int len = 2; len <= n; len <<=
        1) {
        double ang = 2 * PI / len *
            (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2;
                j++) {
                cd u = a[i+j], v =
                    a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}


vector<int> multiply(vector<int> const&
    a, vector<int> const& b) {
```

```
    vector<cd> fa(a.begin(), a.end()),
        fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}

const int mx = 5e5+123;
int ans[mx], n, m;
string s, p;

void solve ( char ch )
{
    vector < int > a, b, c;
    for ( auto u : s ) {
        a.push_back ( ( u == ch ) );
    }

    for ( auto u : p ) {
        b.push_back ( ( u == ch ) );
    }

    c = multiply( a, b );

    for ( int i = m-1; i < n; i++ ) {
        ans[i-m+1] += c[i];
    }

}

int main()
{
    optimize();

    cin >> s >> p;

    reverse( p.begin(), p.end() );

    n = s.size();
    m = p.size();

    solve ( 'A' );
    solve ( 'T' );
    solve ( 'G' );
    solve ( 'C' );

    int sol = INT_MAX;
    for ( int i = 0; i <= n-m; i++ ) {
        sol = min ( sol, m - ans[i] );
    }

    cout << sol << endl;


    return 0;
}
```

6 flow

6.1 ford$_{fulkerson}$

```
/**
Ford-Fulkerson method
Complexity O ( V * E^2 )
**/

struct Ford {

    int n, s, t;
    const int inf = 2147483647;
    vector < vector < int > > capacity;
    vector < vector < int > > adj;
    int parent[mx];

    Ford ( int n, int s, int t ): n(n),
        s(s), t(t), adj(n+1), capacity(
        n+1, vector < int > (n+1, 0) ) {}

    void addEdge( int u, int v, int cap
        ) {
        adj[u].push_back ( v );
        adj[v].push_back ( u );
        capacity[u][v] = cap;
        /**
        For undirected graph :
        capacity[u][v] = cap;
        capacity[v][u] = cap;
        **/
    }

    int bfs() {
        mem ( parent, -1 );
```

```
        parent[s] = -2;
        queue<pair<int, int>> q;
        q.push({s, inf});

        while (!q.empty()) {
            int cur = q.front().first;
            int flow = q.front().second;
            q.pop();

            for (int next : adj[cur]) {
                if (parent[next] == -1 &&
                    capacity[cur][next] >
                    0 ) {
                    parent[next] = cur;
                    int new_flow =
                        min(flow,
                        capacity[cur][next]);
                    if (next == t)
                        return new_flow;
                    q.push({next,
                        new_flow});
                }
            }
        }

        return 0;
}

    int maxflow() {
        int flow = 0;
        int new_flow;

        while (new_flow = bfs()) {
            flow += new_flow;
            int cur = t;
            while (cur != s) {
                int prev = parent[cur];
                capacity[prev][cur] -=
                    new_flow;
                capacity[cur][prev] +=
                    new_flow;
                cur = prev;
            }
        }

        return flow;
    }

};



int main()
{
    optimize();
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    Ford ford ( n, s, t );

    for ( int i = 1; i <= m; i++ ) {
        int u, v, w;
        cin >> u >> v >> w;
        ford.addEdge( u, v, w );
    }

    cout << ford.maxflow();

    return 0;
}
```

6.2 highest$_l$abel$_p$reflow$_p$ush

```
/*
 * Highest Label Preflow Push
 * Complexity : O(V^2 * sqrt(E))
 *
 * Fastest max flow implementation
 * 1. Works on directed graph
 * 2. Works on undirected graph
 * 3. Works on
     multi-edge(directed/undirected) graph
 * 4. Works on
     self-loop(directed/undirected) graph
 *
 * Can't find the actual flow.
 *
 * Status: Tested and OK
 */

template <class flow_t> ///int/long long;
struct HighestLabelPreflowPush {
    struct Edge {
        int v, rev;
        flow_t cap, tot;
        Edge(int a, flow_t b, int c) :
            v(a), rev(c), cap(b), tot(b)
            {}
    };

    const flow_t maxf =
        numeric_limits<flow_t>::max();
    int ht, S, T, N, H, labelcnt;

    vector<flow_t> exflow;
    vector< vector<Edge> > G;
```

```
vector< vector<int> > hq, gap;
vector<int> h, cnt;

HighestLabelPreflowPush(int NN) :
    exflow(NN), G(NN), hq(NN),
    gap(NN) {}

void addEdge(int u, int v, flow_t
    cap) {
    G[u].emplace_back(v, cap,
        G[v].size());
    G[v].emplace_back(u, 0,
        G[u].size() - 1);
}

void update(int u, int newh) {
    ++labelcnt;
    if (h[u] != H)
        --cnt[h[u]];
    h[u] = newh;
    if (newh == H)
        return;
    ++cnt[ht = newh];
    gap[newh].push_back(u);
    if (exflow[u] > 0)
        hq[newh].push_back(u);
}

void globalRelabel() {
    queue<int> q;
    for (int i = 0; i <= H; i++)
        hq[i].clear(), gap[i].clear();
    h.assign(H, H);
    cnt.assign(H, 0);
    q.push(T);
    labelcnt = ht = h[T] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (Edge& e : G[u]) {
            if (h[e.v] == H &&
                G[e.v][e.rev].cap) {
                update(e.v, h[u] + 1);
                q.push(e.v);
            }
        }
        ht = h[u];
    }
}

void push(int u, Edge& e) {
    if (exflow[e.v] == 0)
        hq[h[e.v]].push_back(e.v);
    flow_t df = min(exflow[u], e.cap);
    e.cap -= df;
    G[e.v][e.rev].cap += df;
    exflow[u] -= df;
    exflow[e.v] += df;
}

void discharge(int u) {
    int nxth = H;
    if (h[u] == H)
        return;
    for (Edge& e : G[u])
        if (e.cap) {
            if (h[u] == h[e.v] + 1) {
                push(u, e);
                if (exflow[u] <= 0)
                    return;
            } else if (nxth > h[e.v] +
                1)
                nxth = h[e.v] + 1;
        }
        if (cnt[h[u]] > 1)
            update(u, nxth);
        else
            for (; ht >= h[u];
                gap[ht--].clear()) {
                for (int& j : gap[ht])
                    update(j, H);
            }
    }

    flow_t maxFlow(int s, int t, int n) {
        S = s, T = t, N = n, H = N + 1;
        fill( exflow.begin(),
            exflow.end(), 0 );
        exflow[S] = maxf;
        exflow[T] = -maxf;
        globalRelabel();
        for (Edge& e : G[S]) push(S, e);
        for (; ~ht; --ht) {
            while (!hq[ht].empty()) {
                int u = hq[ht].back();
                hq[ht].pop_back();
                discharge(u);
                if (labelcnt > (N << 2))
                    globalRelabel();
            }
        }
        return exflow[T] + maxf;
    }
};
```

```
int main() {
    optimize();
    int T;
    cin >> T;
    for( int test = 1; test <= T; ++test
        ) {
            int N, M, s, t; ///no. of
                nodes; no. of edges;
                source; sink;
        cin >> N >> M >> s >> t;
            HighestLabelPreflowPush<int>
                hlpp(N+2); ///int to
                long long for flow of
                long long; total no.
                of nodes+2(nodes+1
                does not work);
            for( int i = 1; i <= M;
                ++i ) {
                int u, v, w;
                cin >> u >> v >> w;
                hlpp.addEdge(u, v,
                    w); ///For
                    directed graph

                /**
                        For
                            undirected
                            graph:
                        hlpp.addEdge(u,
                            v, w);
                        hlpp.addEdge(v,
                            u, w);
                **/
            }
                cout << hlpp.maxFlow(s, t,
                    N) << endl; ///source;
                    sink; number of nodes;
    }
    return 0;
}
```

6.3 hungarian

```
const int INF = inf;
int a[123][123];

/// for set s1 and s2 what is maximum
    matching with minimum cost


int main()
{
    optimize();

    int T;
    scanf ( "%d", &T );
    for ( int tc = 1; tc <= T; tc++ ) {
        int n;
        scanf ( "%d", &n );
        for ( int i = 1; i <= n; i++ ) {
            for ( int j = 1; j <= n; j++
                ) {
                scanf( "%d", &a[i][j] );
                a[i][j] *= -1; /// for max
                    cost.
            }
        }
        int m = n;
```

```
vector<int> u (n+1), v (m+1), p
    (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vector<int> minv (m+1, INF);
    vector<char> used (m+1,
        false);
    do {
        used[j0] = true;
        int i0 = p[j0], delta =
            INF, j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur =
                    a[i0][j]-u[i0]-v[j];
                if (cur < minv[j])
                    minv[j] = cur,
                        way[j] = j0;
                if (minv[j] <
                    delta)
                    delta =
                        minv[j], j1
                        = j;
            }
        for (int j=0; j<=m; ++j)
            if (used[j])
                u[p[j]] += delta,
                    v[j] -= delta;
            else
                minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
```

```
            int j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }

    printf ( "Case %d: %d\n", tc,
        v[0] );

    /// v[0] is the cost.
    /// -v[0] for min cost
    /// v[0] for max cost

}

return 0;
}
```

6.4 mcmf

```
namespace mcmf {
    const int MAX = 1000010;
    const ll INF = 1LL << 60;

    ll cap[MAX], flow[MAX], cost[MAX],
        dist[MAX];
    int n, m, s, t, Q[10000010],
        adj[MAX], link[MAX], last[MAX],
        from[MAX], visited[MAX];

    void init(int nodes, int source, int
        sink) {
        m = 0, n = nodes, s = source, t =
            sink;
        for (int i = 0; i <= n; ++i)
            last[i] = -1;
    }

    int addEdge(int u, int v, ll c, ll
        w) {
        adj[m] = v, cap[m] = c, flow[m] =
            0, cost[m] = +w, link[m] =
            last[u], last[u] = m++;
        adj[m] = u, cap[m] = 0, flow[m] =
            0, cost[m] = -w, link[m] =
            last[v], last[v] = m++;
        return m - 2;
    }

    bool spfa() {
        int i, j, x, f = 0, l = 0;
        for (i = 0; i <= n; ++i)
            visited[i] = 0, dist[i] = INF;
        dist[s] = 0, Q[l++] = s;
        while (f < l) {
            i = Q[f++];
            for (j = last[i]; j != -1; j
                = link[j]) {
                if (flow[j] < cap[j]) {
                    x = adj[j];
                    if (dist[x] > dist[i]
                        + cost[j]) {
                        dist[x] = dist[i]
                            + cost[j],
                        from[x] = j;
                        if (!visited[x]) {
                            visited[x] = 1;
                            if (f && rand()
                                & 7) Q[--f]
                                = x;
                            else Q[l++] = x;
                        }
                    }
                }
            }
            visited[i] = 0;
        }
        return (dist[t] != INF);
    }

    pll solve() {
        int i, j;
        ll maxFlow = 0, minCost = 0;
        while (spfa()) {
            ll aug = INF;
            for (i = t, j = from[i]; i !=
                s; i = adj[j ^ 1], j =
                from[i]) {
                aug = min(aug, cap[j] -
                    flow[j]);
            }
            for (i = t, j = from[i]; i !=
                s; i = adj[j ^ 1], j =
                from[i]) {
                flow[j] += aug, flow[j ^
                    1] -= aug;
            }
            maxFlow += aug, minCost +=
                aug * dist[t];
        }
        return {minCost, maxFlow};
    }
}
```

7 geometry

7.1 geometry$_{t}emplate$

```
double INF = 1e100;
double EPS = 1e-12;

struct PT {
        double x, y;
        PT() {}
        PT(double x, double y) : x(x),
            y(y) {}
        PT(const PT &p) : x(p.x), y(p.y)
            {}
        PT operator + (const PT &p) const
            { return PT(x+p.x, y+p.y); }
        PT operator - (const PT &p) const
            { return PT(x-p.x, y-p.y); }
        PT operator * (double c)  const {
            return PT(x*c, y*c ); }
        PT operator / (double c)  const {
            return PT(x/c, y/c ); }
        bool operator <(const PT &p)
            const {
                return x < p.x || (x ==
                    p.x && y < p.y);
        }
};

double dot(PT p, PT q)   { return
    p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return
    dot(p-q,p-q); }
double cross(PT p, PT q) { return
    p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const
    PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// checks if a-b-c is CW or not.
bool isPointsCW(PT a, PT b, PT c) {
    return
        a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)+EPS
        < 0;
}
// checks if a-b-c is CCW or not.
bool isPointsCCW(PT a, PT b, PT c) {
    return
        a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)
        > EPS;
}
// checks if a-b-c is collinear or not.
bool isPointsCollinear(PT a, PT b, PT c)
    {
    return
        abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y))
        <= EPS;
}

// rotate a point CCW or CW around the
    origin
PT RotateCCW90(PT p) { return
    PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return
    PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) { // rotate
    a point CCW t degrees around the
    origin
  return PT(p.x*cos(t)-p.y*sin(t),
      p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a
    and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
    return a + (b-a)*dot(c-a,
        b-a)/dot(b-a, b-a);
}

// project point c onto line segment
    through a and b
PT ProjectPointSegment(PT a, PT b, PT c)
    {
        double r = dot(b-a,b-a);
        if (fabs(r) < EPS) return a;
        r = dot(c-a, b-a)/r;
        if (r < 0) return a;
        if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment
    between a and b
double DistancePointSegment(PT a, PT b,
    PT c) {
        return sqrt(dist2(c,
            ProjectPointSegment(a, b,
            c)));
}

// compute distance between point
    (x,y,z) and plane ax+by+cz=d
```

```
double DistancePointPlane(double x,
    double y, double z,
                          double a, double
                              b, double c,
                              double d)
{
       return
           fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c
    to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT
    d) {
       return fabs(cross(b-a, c-d)) <
           EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT
    d) {
       return LinesParallel(a, b, c, d)
     && fabs(cross(a-b, a-c)) < EPS
     && fabs(cross(c-d, c-a)) < EPS;
}

// compute intersection of line passing
    through a and b
// with line passing through c and d,
    assuming that unique
// intersection exists;
PT ComputeLineIntersection(PT a, PT b,
    PT c, PT d) {
       b=b-a; d=c-d; c=c-a;
       assert(dot(b, b) > EPS && dot(d,
           d) > EPS);
       return a + b*cross(c, d)/cross(b,
           d);
}

// shift the straight line passing
    through points a and b
// by distance Dist.
// If Dist is negative the line is
    shifted rightwards or upwards.
// If Dist is positive the line is
    shifted leftwards or downwards.
// The new line passes through points c
    and d
//
    https://math.stackexchange.com/questions/2593627/i-have-a-line-i-want-to-move-the-line-a-ce
pair<PT,PT> ShiftLineByDist(PT a, PT b,
    double Dist) {
       double r = sqrt( dist2(a, b) );
       double delx = (Dist*(a.y-b.y))/r;
       double dely = (Dist*(b.x-a.x))/r;
       PT c = PT(a.x+delx, a.y+dely);
       PT d = PT(b.x+delx, b.y+dely);
       return MP(c, d);
}

// This code computes the area or
    centroid of a (possibly nonconvex)
// polygon, assuming that the
    coordinates are listed in a clockwise
    or
// counterclockwise fashion. Note that
    the centroid is often known as
// the "center of gravity" or "center of
    mass".

double ComputeSignedArea(const
    vector<PT> &p) {
       double area = 0;
       for(int i = 0; i < p.size(); i++)
           {
           int j = (i+1) % p.size();
           area += p[i].x*p[j].y -
               p[j].x*p[i].y;
       }
       return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
       return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
       PT c(0,0);
       double scale = 6.0 *
           ComputeSignedArea(p);
       for (int i = 0; i < p.size();
           i++){
           int j = (i+1) % p.size();
           c = c +
               (p[i]+p[j])*(p[i].x*p[j].y
               - p[j].x*p[i].y);
       }
       return c / scale;
}

// angle from p2->p1 to p2->p3, returns
    -PI to PI
double angle(PT p1, PT p2, PT p3)
{
    PT va = p1-p2,vb=p3-p2;
```

```
    double x,y;
    x=dot(va,vb);
    y=cross(va,vb);
    return(atan2(y,x));
}


int main()
{
        // expected: (-5,2)
        cerr << RotateCCW90(PT(2,5)) <<
            endl;

        // expected: (5,-2)
        cerr << RotateCW90(PT(2,5)) <<
            endl;

        // expected: (-5,2)
        cerr << RotateCCW(PT(2,5),M_PI/2)
            << endl;

        // expected: (5,2)
        cerr <<
            ProjectPointLine(PT(-5,-2),
            PT(10,4), PT(3,7)) << endl;

        // expected: (5,2) (7.5,3) (2.5,1)
        cerr <<
            ProjectPointSegment(PT(-5,-2),
            PT(10,4), PT(3,7)) << " "
                <<
                    ProjectPointSegment(PT(7.5,3),
                    PT(10,4), PT(3,7)) <<
                    " "
                <<
                    ProjectPointSegment(PT(-5,-2),
                    PT(2.5,1), PT(3,7)) <<
                    endl;

        // expected: 6.78903
        cerr <<
            DistancePointPlane(4,-4,3,2,-2,5,-8)
            << endl;


        // expected: 1 0 1
        cerr << LinesParallel(PT(1,1),
            PT(3,5), PT(2,1), PT(4,5)) <<
            " "
                << LinesParallel(PT(1,1),
                    PT(3,5), PT(2,0),
                    PT(4,5)) << " "
                << LinesParallel(PT(1,1),
                    PT(3,5), PT(5,9),
                    PT(7,13)) << endl;

        // expected: 0 0 1
        cerr << LinesCollinear(PT(1,1),
            PT(3,5), PT(2,1), PT(4,5)) <<
            " "
                << LinesCollinear(PT(1,1),
                    PT(3,5), PT(2,0),
                    PT(4,5)) << " "
                << LinesCollinear(PT(1,1),
                    PT(3,5), PT(5,9),
                    PT(7,13)) << endl;

        // expected: (1,2)
        cerr <<
            ComputeLineIntersection(PT(0,0),
            PT(2,4), PT(3,1), PT(-1,3))
            << endl;

        // area should be 5.0
        // centroid should be (1.1666666,
            1.166666)
        PT pa[] = { PT(0,0), PT(5,0),
            PT(1,1), PT(0,5) };
        vector<PT> p(pa, pa+4);
        PT c = ComputeCentroid(p);
        cerr << "Area: " <<
            ComputeArea(p) << endl;
        cerr << "Centroid: " << c << endl;

        // expected: 0
        cerr << isPointsCCW( PT(5, 6),
            PT(10, 10), PT(11, 5) ) <<
            endl;
        // expected: 1
        cerr << isPointsCCW( PT(5, 6),
            PT(10, 2), PT(11, 5) ) <<
            endl;
        // expected: 1
        cerr << isPointsCW( PT(5, 6),
            PT(10, 10), PT(11, 5) ) <<
            endl;
        // expected: 0
        cerr << isPointsCW( PT(5, 6),
            PT(10, 2), PT(11, 5) ) <<
            endl;
        // expected: 0
        cerr << isPointsCollinear( PT(5,
            6), PT(10, 2), PT(11, 5) ) <<
```

```
        endl;
    // expected: 1
    cerr << isPointsCollinear( PT(5,
        6), PT(10, 6), PT(11, 6) ) <<
        endl;

    // expected: (-0.437602,12.6564)
        (2.5624,14.6564)
    cerr << ShiftLineByDist( PT(4,
        6), PT(7, 8), 8 ).F << " " <<
        ShiftLineByDist( PT(4, 6),
        PT(7, 8), 8 ).S << endl;
    // expected: (8.4376,-0.656402)
        (11.4376,1.3436)
    cerr << ShiftLineByDist( PT(4,
        6), PT(7, 8), -8 ).F << " "
        << ShiftLineByDist( PT(4, 6),
        PT(7, 8), -8 ).S << endl;


}
```

7.2 half$_{p}$*lanner*

```
// OFFLINE
// Complexity: O(NlgN)
// very easy concept and implementation
//
    https://codeforces.com/blog/entry/61710

double INF = 1e100;
double EPS = 1e-12;

struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x),
        y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)
        {}
    PT operator + (const PT &p) const
        { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const
        { return PT(x-p.x, y-p.y); }
    PT operator * (double c)  const {
        return PT(x*c, y*c ); }
    PT operator / (double c)  const {
        return PT(x/c, y/c ); }
    bool operator <(const PT &p)
        const {
            return x < p.x || (x ==
                p.x && y < p.y);
        }
};

ostream &operator<<(ostream &os, const
    PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}


int steps = 600;

vector<PT> lower_hull, upper_hull;
int lower_hull_sz, upper_hull_sz;

bool leBorder = 0, riBorder = 0;

double func( double xx, double val )
{
    double ans1 = INF, ans2 = -INF,
        ans;
    for( int i = 0; i <
        lower_hull_sz-1; ++i ) {
            if( leBorder && (i == 0) )
                continue;
            PT a = lower_hull[i], b =
                lower_hull[i+1];
                    // straight
                line passes through
                points a and b
            double m =
                (a.y-b.y)/(a.x-b.x);
                    // slope of the
                straight line; if the
                TL is strict, then
                better precalculate
                all the slopes and
                store them beforehand
            double c = a.y -
                a.x*(m);            //
                intercept of the
                straight line; if the
                TL is strict, then
                better precalculate
                all the intercepts and
                store them beforehand
            double aa = m*xx;
            double bb = c;
            double cc = aa+bb;
            ans1 = min( ans1, cc );
    }
    for( int i = 0; i <
        upper_hull_sz-1; ++i ) {
```

```
            if( riBorder && (i ==
                upper_hull_sz-2) )
                continue;
            PT a = upper_hull[i], b =
                upper_hull[i+1];
                       // straight
                line passes through
                points a and b
            double m =
                (a.y-b.y)/(a.x-b.x);
                    // slope of the
                straight line; if the
                TL is strict, then
                better precalculate
                all the slopes and
                store them beforehand
            double c = a.y -
                a.x*(m);           //
                intercept of the
                straight line; if the
                TL is strict, then
                better precalculate
                all the intercepts and
                store them beforehand
            double aa = m*xx;
            double bb = c;
            double cc = aa+bb;
            ans2 = max( ans2, cc );
        }
        ans = ans1-ans2;
        return ans;
}

bool Ternary_Search(double val)
{
        double lo = -INF, hi = INF, mid1,
            mid2;
        leBorder = 0, riBorder = 0;
        if( lower_hull[0].x ==
            lower_hull[1].x ) lo =
            lower_hull[0].x+val, leBorder
            = 1;
        if( upper_hull[upper_hull_sz-2].x
            ==
            upper_hull[upper_hull_sz-1].x
            ) hi =
            upper_hull[upper_hull_sz-1].x-val,
            riBorder = 1;
        if( lo > hi ) return 0;
        for( int i = 0; i < steps; ++i ) {
                mid1 = (lo*2.0 + hi)/3.0;
                mid2 = (lo + 2.0*hi)/3.0;
                double ff1 = func(mid1,
                    val);
                double ff2 = func(mid2,
                    val);
                if( ff1 >= 0 || ff2 >= 0 )
                    return 1;
                if( ff1 > ff2 ) hi = mid2;
                else lo = mid1;
        }
        if( func(lo, val) >= 0 ) return 1;
        return 0;
}
```

7.3 ternary$_s$earch

```
double ternary_search(double l, double
    r) {
    double eps = 1e-9;           //set
        the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);    //evaluates
            the function at m1
        double f2 = f(m2);    //evaluates
            the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                 //return
        the maximum of f(x) in [l, r]
}


double ternary_search( int l, int r) {
                  //set the error limit
                     here
    while (r - l <= 3) {
        int m1 = l + (r - l) / 3;
        int m2 = r - (r - l) / 3;
        int f1 = f(m1);    //evaluates
            the function at m1
        int f2 = f(m2);    //evaluates
            the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
```

```
    int ret = inf;
    for ( int i = l; i <= r; i++ ) tet =
        max ( ret, f(l) )
    return f(l);                    //return
        the maximum of f(x) in [l, r]
}
```

8 graph

8.1 HLD

```
const int mx = 2e5+123;
ll t[mx*3], a[mx], prop[3*mx];
bool vis[3*mx];

int baseArry[mx], basePos[mx], chainNO,
    chainHead[mx], parent[mx], level[mx],
    chainInd[mx], ptr, p[mx][40], sz[mx];
vii adj[mx];

void shift ( int id, int b, int e )
{
    int mid = ( b + e ) >> 1;
    t[id*2] += ( mid-b+1 * prop[id] );
    t[id*2+1] += ( e-(mid+1)+1 *
        prop[id] );

    prop[id*2] += prop[id];
    prop[id*2+1] += prop[id];

    vis[id*2] = vis[id*2+1] = 1;
    prop[id] = vis[id] = 0;
}

void init ( int id, int b, int e )
{
    if ( b == e ) {
        t[id] = baseArry[b];
        return;
    }

    int mid = ( b + e ) >> 1;

    init ( id*2, b, mid );
    init ( id*2+1, mid+1, e );

    t[id] = t[id*2] + t[id*2+1];
}

void upd ( int id, int b, int e, int i,
    int j, ll val )
{
    if ( b > j || e < i ) return;
    if ( b >= i && e <= j ) {
        t[id] += ( val * e-b+1 );
        prop[id] += ( prop[id] * val );
        vis[id] = 1;
        return;
    }

    if ( vis[id] ) shift ( id, b, e );
    int mid = ( b + e ) >> 1;

    upd ( id*2, b, mid, i, j, val );
    upd ( id*2+1, mid+1, e, i, j, val );

    t[id] = t[id*2] + t[id*2+1];
}

ll ask ( int id, int b, int e, int i,
    int j )
{
    if ( b > j || e < i ) return 0;
    if ( b >= i && e <= j ) return t[id];

    if ( vis[id] ) shift ( id, b, e );
    int mid = ( b + e ) >> 1;

    ll ret1 = ask ( id*2, b, mid, i, j );
    ll ret2 = ask ( id*2+1, mid+1, e, i,
        j );

    return ret1 + ret2;
}

int dfs ( int u, int lev )
{
    int ret = 1;
    level[u] = lev;
    for ( auto v : adj[u] ) {
        if ( parent[u] != v.F ) {
            parent[v.F] = u;
            ret += dfs ( v.F, lev+1 );
        }
    }

    sz[u] = ret;
    return ret;
}


void HLD ( int u, int cost, int pU )
{
```

```
    if ( chainHead[chainNO] == -1 ) {
        chainHead[chainNO] = u;
    }

    chainInd[u] = chainNO;
    basePos[u] = ++ptr;
    baseArry[ptr] = cost;

    int m = -1, id = -1, c = -1;

    for ( auto v : adj[u] ) {
        if ( v.F != pU ) {
            if ( sz[v.F] > m ) {
                m = sz[v.F];
                id = v.F;
                c = v.S;
            }
        }
    }


    if ( id != -1 ) HLD ( id, c, u );

    for ( auto v : adj[u] ) {
        if ( v.F != pU && v.F != id ) {
            chainNO++;
            HLD ( v.F, v.S, u );
        }
    }
}

void preprocess ( int n )
{
    for ( int i = 1; i <= n; i++ )
        p[i][0] = parent[i];

    for ( int j = 1; (1 << j) <= n; j++
        ) {
        for ( int i = 1; i <= n; i++ ) {
            if ( p[i][j-1] != -1 )
                p[i][j] =
                p[p[i][j-1]][j-1];
        }
    }
}

int LCA ( int u, int v )
{
    if ( level[u] < level[v] ) swap ( u,
        v );

    int dist = level[u] - level[v];

    int rise;
    while ( dist > 0 ) {
        rise = log2( dist );
        u = p[u][rise];
        dist -= ( 1 << rise );
    }

    if ( u == v ) return u;

    for ( int i = 20; i >= 0; i-- ) {
        if ( p[u][i] != p[v][i] &&
            p[u][i] != -1 ) {
            u = p[u][i];
            v = p[v][i];
        }
    }
    return parent[u];
}


void query_upd ( int u, int v, ll val )
{
    if ( u == v ) return;
    int chainU, chainV = chainInd[v];

    while ( 1 ) {
        chainU = chainInd[u];
        if ( chainU == chainV ) {
            upd ( 1, 1, ptr,
                basePos[v]+1, basePos[u],
                val );
            break;
        }

        upd ( 1, 1, ptr,
            basePos[chainHead[chainU]],
            basePos[u], val );
        u = chainHead[chainU];
        u = parent[u];
    }

    return;
}


void queryUpd ( int u, int v, ll val )
{
    int lca = LCA ( u, v );
    query_upd ( u, lca, val );
    query_upd ( v, lca, val );
}
```

```
ll query_ask ( int u, int v )
{
    if ( u == v ) return 0;
    int chainU, chainV = chainInd[v];
    ll ans = 0;

    while ( 1 ) {
        chainU = chainInd[u];
        if ( chainU == chainV ) {
            ans += ask ( 1, 1, ptr,
                basePos[v]+1, basePos[u] );
            break;
        }

        ans += ask ( 1, 1, ptr,
            basePos[chainHead[chainU]],
            basePos[u] );
        u = chainHead[chainU];
        u = parent[u];
    }

    return ans;
}

ll queryAsk ( int u, int v )
{
    int lca = LCA ( u, v );
    return query_ask ( u, lca ) +
        query_ask ( v, lca );
}
```

```
int main()
{
    optimize();

    int n;

    ptr = 0, chainNO = 1;
    mem ( p, -1 );
    mem ( chainHead, -1 );


    dfs ( 1, 0 );
    HLD ( 1, 0, -1 );
    preprocess( n );
    init ( 1, 1, ptr );

    return 0;
}
```

8.2 LCA

```
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u]
        >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
```

```
    dfs(root, root);
}
```

8.3 dijkstra

```
const int INF = 2147483647;
const int MAX = 5005;
int D[MAX], N; // Keeps minimum distance
    to each node
vector<pair<int,int>> E[MAX]; //
    Adjacency list

void dijkstra()
{
    for(int i = 1; i <= N; i++) D[i] =
        INF;
    D[1] = 0;
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
        q;
    q.push({0,1});

    while(!q.empty())
    {
        pair<int,int> p = q.top();
        q.pop();

        int u = p.second, dist = p.first;
        if(dist > D[u]) continue;

        for(pair<int,int> pr : E[u])
        {
            int v = pr.first;
            int next_dist = dist +
                pr.second;
            if(next_dist < D[v])
            {
                D[v] = next_dist;
                q.push({next_dist,v});
            }
        }
    }
}
```

8.4 dynamic$_c$*onnectivity*

```
const int mx = 100100;
int n, m, par[mx], sz[mx];
bool ans[mx];
pii queries[mx];
int t[mx*5];
map<pii, int> M;
stack<int> st;

void update(int cur, int s, int e, int
    l, int r, pii val) {
    if (s > r || e < l) return;
    if (l <= s && e <= r) {
        t[cur].PB(val);
        return;
    }
    int c1 = (cur << 1), c2 = c1 | 1, m
        = (s + e) >> 1;
    update(c1, s, m, l, r, val);
    update(c2, m + 1, e, l, r, val);
}

int Find(int u) { return (par[u] == u ?
    u : Find(par[u])); }

bool isSame(int u, int v) { return
    Find(u) == Find(v); }

bool makeAns(int i) {
    if (queries[i].F != -1) {
        return isSame(queries[i].F,
            queries[i].S);
    }
    return 0;
}

void Merge(pii edge) {
    int u = Find(edge.F), v =
        Find(edge.S);
    if (u == v) return;
    if (sz[u] < sz[v]) swap(u, v);
    sz[u] += sz[v];
    par[v] = u;
    st.push(v);
}

void rollback(int moment) {
    while (st.size() > moment) {
        int cur = st.top();
        st.pop();
        sz[Find(cur)] -= sz[cur];
        par[cur] = cur;
    }
}

void dfs(int cur, int s, int e) {
    if (s > e) return;
```

```
    int moment = st.size();
    for (pii edge : t[cur]) {
        Merge(edge);
    }
    if (s == e) ans[s] = makeAns(s);
    else {
        int c1 = (cur << 1), c2 = c1 | 1,
            m = (s + e) >> 1;
        dfs(c1, s, m);
        dfs(c2, m + 1, e);
    }
    rollback(moment);
}

int main() {
    optimize();
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) {
        par[i] = i;
        sz[i] = 1;
    }
    for (int i = 1; i <= m; ++i)
        queries[i] = MP(-1, -1);
    for (int i = 1; i <= m; ++i) {
        string q;
        int u, v;
        cin >> q >> u >> v;
        if (u < v) swap(u, v);
        if (q == "conn") queries[i] =
            MP(u, v);
        else {
            if (q == "rem") {
                update(1, 1, m, M[MP(u,
                    v)], i, MP(u, v));
                M.erase(MP(u, v));
            }
            else M[MP(u, v)] = i;
        }
    }
    for (auto it : M) update(1, 1, m,
        it.S, m, it.F);
    dfs(1, 1, m);
    for (int i = 1; i <= m; ++i) {
        if (queries[i].F != -1) {
            cout << (ans[i] ? "YES" :
                "NO") << endl;
        }
    }
    return 0;
}
```

8.5 $\mathbf{mst}_{kruskal}$

```
struct edge {
    int u, v, w;
    bool operator<(const edge& p) const
    {
        return w < p.w;
    }
};

int par[MAXN], size[MAXN];
vector<edge> e;

int find_root(int i) { return (par[i] ==
    i ? i : par[i] = find_root(par[i])); }

void unite(int u, int v) {
    u = find_root(u), v = find_root(v);
    if (u != v) {
        if (size[u] < size[v]) swap(u, v);
        par[v] = u;
        size[u] += size[v];
    }
}

int mst(int n) {
    sort(e.begin(), e.end());
    for (int i = 1; i <= n; i++) {
        par[i] = i;
        size[i] = 1;
    }
    int s = 0;
    for (int i = 0; i < (int)e.size();
        i++) {
        int u = find_root(e[i].u);
        int v = find_root(e[i].v);
        if (u != v) {
            unite(u, v);
            s += e[i].w;
        }
    }
    return s;
}
```

8.6 $\mathbf{scc}_{(kosaraju)}$

```
/**
  * Strongly Connected Component
  * Kosaraju's Algorighm
  * Complexity: O(V + E)
  */
```

```
int n;
stack<int> Stack;
vi adj[mxN], tadj[mxN]; // tadj contains
    the reverse directions of all the
    directed edges of adj.
bool visited[mxN];
vvi scc; // Contains all the scc(s).

void dfs(int u) {
    if (visited[u]) return;
    visited[u] = 1;
    for (int v : adj[u]) {
        dfs(v);
    }
    Stack.push(u);
}

void dfs2(int id, int u) {
    if (visited[u]) return;
    visited[u] = 1;
    scc[id].PB(u);
    for (int v : tadj[u]) {
        dfs2(id, v);
    }
}

// Just call the function when finding
    all the scc(s).
// This stores all the scc(s) in the scc
    vector.
void kosaraju() {
    mem(visited, 0);
    for (int u = 1; u <= n; ++u) {
        if (!visited[u]) {
            dfs(u);
        }
    }
    mem(visited, 0);
    int cnt = 0;
    while (!Stack.empty()) {
        int u = Stack.top();
        Stack.pop();
        if (!visited[u]) {
            scc.PB(vi(0));
            dfs2(cnt, u);
            cnt++;
        }
    }
}
```

9 math

9.1 big*integer*

```
#include <bits/stdc++.h>

using namespace std;

struct Bigint {
    // representations and structures
    string a; // to store the digits
    int sign; // sign = -1 for negative
        numbers, sign = 1 otherwise

    // constructors
    Bigint() {} // default constructor
    Bigint( string b ) { (*this) = b; }
        // constructor for string

    // some helpful methods
    int size() { // returns number of
        digits
        return a.size();
    }

    Bigint inverseSign() { // changes
        the sign
        sign *= -1;
        return (*this);
    }

    Bigint normalize( int newSign ) { //
        removes leading 0, fixes sign
        for( int i = a.size() - 1; i > 0
            && a[i] == '0'; i-- )
            a.erase(a.begin() + i);
        sign = ( a.size() == 1 && a[0] ==
            '0' ) ? 1 : newSign;
        return (*this);
    }

    // assignment operator
    void operator = ( string b ) { //
        assigns a string to Bigint
        a = b[0] == '-' ? b.substr(1) : b;
        reverse( a.begin(), a.end() );
        this->normalize( b[0] == '-' ? -1
            : 1 );
    }

    // conditional operators
```

```
bool operator < ( const Bigint &b )
    const { // less than operator
    if( sign != b.sign ) return sign
        < b.sign;
    if( a.size() != b.a.size() )
        return sign == 1 ? a.size() <
            b.a.size() : a.size() >
            b.a.size();
    for( int i = a.size() - 1; i >=
        0; i-- ) if( a[i] != b.a[i] )
        return sign == 1 ? a[i] <
            b.a[i] : a[i] > b.a[i];
    return false;
}

bool operator == ( const Bigint &b )
    const { // operator for equality
    return a == b.a && sign == b.sign;
}

// mathematical operators
Bigint operator + ( Bigint b ) { //
    addition operator overloading
    if( sign != b.sign ) return
        (*this) - b.inverseSign();
    Bigint c;
    for(int i = 0, carry = 0;
        i<a.size() || i<b.size() ||
        carry; i++ ) {
        carry+=(i<a.size() ? a[i]-48
            : 0)+(i<b.a.size() ?
            b.a[i]-48 : 0);
        c.a += (carry % 10 + 48);
        carry /= 10;
    }
    return c.normalize(sign);
}

Bigint operator - ( Bigint b ) { //
    subtraction operator overloading
    if( sign != b.sign ) return
        (*this) + b.inverseSign();
    int s = sign; sign = b.sign = 1;
    if( (*this) < b ) return ((b -
        (*this)).inverseSign()).normalize(-s);
    Bigint c;
    for( int i = 0, borrow = 0; i <
        a.size(); i++ ) {
        borrow = a[i] - borrow - (i <
            b.size() ? b.a[i] : 48);
        c.a += borrow >= 0 ? borrow +
            48 : borrow + 58;
        borrow = borrow >= 0 ? 0 : 1;
    }
    return c.normalize(s);
}

Bigint operator * ( Bigint b ) { //
    multiplication operator
    overloading
    Bigint c("0");
    for( int i = 0, k = a[i] - 48; i
        < a.size(); i++, k = a[i] -
        48 ) {
        while(k--) c = c + b; // ith
            digit is k, so, we add k
            times
        b.a.insert(b.a.begin(), '0');
            // multiplied by 10
    }
    return c.normalize(sign * b.sign);
}

Bigint operator / ( Bigint b ) { //
    division operator overloading
    if( b.size() == 1 && b.a[0] ==
        '0' ) b.a[0] /= ( b.a[0] - 48
        );
    Bigint c("0"), d;
    for( int j = 0; j < a.size(); j++
        ) d.a += "0";
    int dSign = sign * b.sign; b.sign
        = 1;
    for( int i = a.size() - 1; i >=
        0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c -
            b, d.a[i]++;
    }
    return d.normalize(dSign);
}

Bigint operator % ( Bigint b ) { //
    modulo operator overloading
    if( b.size() == 1 && b.a[0] ==
        '0' ) b.a[0] /= ( b.a[0] - 48
        );
    Bigint c("0");
    b.sign = 1;
    for( int i = a.size() - 1; i >=
        0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c - b;
```

```
        }
        return c.normalize(sign);
    }

    // output method
    void print() {
        if( sign == -1 ) putchar('-');
        for( int i = a.size() - 1; i >=
            0; i-- ) putchar(a[i]);
    }
};

int main() {
    Bigint a, b, c; // declared some
        Bigint variables

    ////////////////////////
    // taking Bigint input //
    ////////////////////////

    string input; // string to take input
    cin >> input; // take the Big
        integer as string
    a = input; // assign the string to
        Bigint a
    cin >> input; // take the Big
        integer as string
    b = input; // assign the string to
        Bigint b

    ////////////////////////////////
    // Using mathematical operators //
    ////////////////////////////////

    c = a + b; // adding a and b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a - b; // subtracting b from a
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a * b; // multiplying a and b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a / b; // dividing a by b
    c.print(); // printing the Bigint
    puts(""); // newline
    c = a % b; // a modulo b
    c.print(); // printing the Bigint
    puts(""); // newline

    ////////////////////////////////
    // Using conditional operators //
    ////////////////////////////////

    if( a == b ) puts("equal"); //
        checking equality
    else puts("not equal");
    if( a < b ) puts("a is smaller than
        b"); // checking less than
        operator
    return 0;
}
```

9.2 bitwise$_{sieve}$

```
#define mx 100001010
long long a[mx / 64 + 200];
int prime[5800000];
int cnt = 0;

void sieveGen( int limit )
{
    limit += 100;
    int sq = sqrt ( limit );

        for (long long i = 3; i <= sq; i
            += 2) {
            if(!(a[i/64]&(1LL<<(i%64))))
                {
                    for(long long j =
                        i * i; j <=
                        limit; j += 2 *
                        i) {
                        a[j/64] |=
                            (1LL<<(j%64));
                    }
                }
        }

        prime[cnt++] = 2;
        for (long long i = 3; i <= limit;
            i += 2) {
            if(!(a[i / 64] & (1LL <<
                (i % 64)))) {
                    prime[cnt++] = i;
            }

        }

}
```

9.3 nominator$_d$enominator

```
struct frac {
    ll n, d;

    frac(ll _n = 0, ll _d = 1) {
        if ( _d < 0 ) _n = -_n, _d = -_d;
        ll g = gcd ( abs(_n), _d );
        n = _n / g;
        d = _d / g;
    }

    friend frac operator + ( const frac
        &a, const frac &b ) {
        return frac ( ( a.n * b.d ) + (
            b.n * a.d ), ( a.d * b.d ) );
    }

    friend frac operator - ( const frac
        &a, const frac &b ) {
        return frac ( ( a.n * b.d ) - (
            b.n * a.d ), ( a.d * b.d ) );
    }

    friend frac operator * ( const frac
        &a, const frac &b ) {
        return frac ( ( a.n * b.n ), (
            a.d * b.d ) );
    }

    friend frac operator / ( const frac
        &a, const frac &b ) {
        return frac ( a.n * b.d, b.n *
            a.d );
    }

    friend bool operator < ( const frac
        &a, const frac &b ) {
        frac ret = a - b;
        return ret.n < 0;
    }

    friend bool operator > ( const frac
        &a, const frac &b ) {
        frac ret = a - b;
        return ret.n >= 0;
    }

    friend void swap ( frac &a, frac &b
        ) {
        frac tmp = b;
        b = a;
        a = tmp;
    }

};



int main()
{

    frac f1 = frac ( 1, 2 ), f2 = frac(
        2, 3 );
    frac ans;

    ans = f1 + f2;
    cout << ans.n << " " << ans.d <<
        endl; ///7 6

    ans = f1 - f2;
    cout << ans.n << " " << ans.d <<
        endl; ///-1 6

    ans = f1 * f2;
    cout << ans.n << " " << ans.d <<
        endl; ///1 3

    ans = f1 / f2;
    cout << ans.n << " " << ans.d <<
        endl; ///3 4

    swap ( f1, f2 );
    cout << f1.n << " " << f1.d << endl;
        ///2 3
    cout << f2.n << " " << f2.d << endl;
        ///1 2


    if ( f1 > f2 ) cout <<
        "Greater\n";///Greater
    if ( f1 < f2 ) cout <<
        "Smaller\n";///Condition is not
        true.

    swap( f1, f2 );
    if ( f1 > f2 ) cout <<
        "Greater\n";///Condition is not
        true.
    if ( f1 < f2 ) cout <<
        "Smaller\n";///Smaller

    return 0;
}
```

10 number-theory

11 numerical

12 string

12.1 aho$_{c}$*orasick*

```
const int N = 1e4;

///beware! if k distinct patterns are
    given having sum of length m then
    size of ending array and oc array will
///be at most m.sqrt(m) ,But for similar
    patterns one must act with them
    differently
struct aho_corasick
{
       bool  is_end[N];
       int link[N];          ///A suffix
           link for a vertex p is a edge
           that points to
                        ///the longest
                            proper
                            suffix of
                        ///the string
                            corresponding
                            to the
                            vertex p.
   int psz = 1;             ///tracks
       node numbers of the trie
       map<char, int> to[N]; ///tracks
           the next node
vector<int> ending[N];
   ///ending[i] stores the
   indexes of patterns which ends
                ///at node
                    i(from the
                    trie)
vector<int> oc[N];
   ///oc[i] stores ending index
   of all occurrences of
   pattern[i]
                ///so real
                    oc[i][j]=oc[i][j]-pattern[i].size()+1
void clear()
{
       for(int i = 0; i <= psz;
           i++)
               is_end[i] = 0,
                   link[i] = 0,
                   to[i].clear(),ending[i].clear(),oc[i].clear();

       psz = 1;
       is_end[0] = 1;
}

void faho_corasick() { clear(); }

void add_word(string s,int idx)
{
       int u = 0;
       for(char c: s)
       {
              if(!to[u].count(c))
                  to[u][c] =
                  psz++;
              u = to[u][c];
       }

       is_end[u] = 1;
       ending[u].push_back(idx);
   }

   void populate(int cur)
   {
       /// merging the occurrences of
           patterns ending at cur node
           in the trie
       for(auto occ: ending[link[cur]])
           ending[cur].push_back(occ);
   }

   void populate(vector<int> &en, int
       cur)
   {
       /// occurrences of patterns in
           the given string
       for(auto idx: en)
       {
           oc[idx].push_back(cur);
       }
   }

       void push_links()
       {
              queue<int> q;
              int u, v, j;
              char c;

              q.push(0);
              link[0] = -1;
```

```
        while(!q.empty())
        {
                u = q.front();
                q.pop();

                for(auto it: to[u])
                {
                        v =
                            it.second;
                        c =
                            it.first;
                        j = link[u];

                        while(j !=
                            -1 &&
                            !to[j].count(c))
                            {
            j = link[j];
                        }
                        if(j != -1)
                            link[v]
                            =
                            to[j][c];
                        else
                            link[v]
                            = 0;

                        q.push(v);
                        populate(v);
                }
        }
    }

    void traverse(string s)
{
        int n=s.size();
        int cur=0;///root

        for(int i=0;i<n;i++){
            char c=s[i];
            while(cur!=-1 &&
                !to[cur].count(c))
                cur=link[cur];
            if(cur!=-1) cur=to[cur][c];
            else cur=0;
            populate(ending[cur],i);
        }
    }
};

aho_corasick t;

int main()
{

    int T;
    cin >> T;
    for ( int tc = 1; tc <= T; tc++ ) {

        t.faho_corasick();
        string s;
        cin >> s;
        int q;
        cin >> q;


        for ( int k = 1; k <= q; k++ ) {
            string p;
            cin >> p;
            t.add_word( p, k );
        }

        t.push_links();
        t.traverse( s );


        for ( int i = 1; i <= q; i++ ) {
            cout << t.oc[i].size() <<
                endl; /// Ending index of
                patter i in s
            for ( auto u : t.oc[i] ) cout
                << u << " ";
            cout << endl;
        }
    }


    return 0;
}
```

12.2 $\text{aho}_c orasick_e maxx$

```
const int K = 26;

struct Vertex {
    int Next[K];
    bool leaf = 0;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p = -1, char ch = '$') :
        p(p), pch(ch) {
```

```
        fill(begin(Next), end(Next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].Next[c] == -1) {
            t[v].Next[c] = t.size();
            t.push_back(Vertex(v, ch));
        }
        v = t[v].Next[c];
    }
    t[v].leaf = 1;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else t[v].link =
            go(get_link(t[v].p),
            t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].Next[c] != -1)
            t[v].go[c] = t[v].Next[c];
        else t[v].go[c] = (v == 0 ? 0 :
            go(get_link(v), ch));
    }
    return t[v].go[c];
}
```

12.3 hashing

```
struct simpleHash{
    vector<long long>p;
    vector<long long>h;

    long long base,mod,len;

    simpleHash(){}
    simpleHash(string &str, long long b,
        long long m){
        //0 base index array.
        base=b; mod=m; len=str.size();
        p.resize(len,1);
        h.resize(len+1,0);

        for(int
            i=1;i<len;i++)p[i]=(p[i-1]*base)%mod;
        for(int
            i=1;i<=len;i++)h[i]=(h[i-1]*base+(str[i-1]-'a'+3))%mod;
    }

    long long rangeHash(int l,int r){
        //l and r inclusive
        return
            (h[r+1]-((h[l]*p[r-l+1])%mod)+mod)%m
    }
};


struct doubleHashing{
    simpleHash h1,h2;

    doubleHashing(string &str){
        h1=simpleHash(str,43, (long
            long)1e9+7);
        h2=simpleHash(str,97, (long
            long)1e9+7);
    }

    long long rangeHash(int l,int r){
        return
            (h1.rangeHash(l,r)<<32LL)^h2.rangeHa
    }
};

//***Double Hashing***


int pw[123], hash_s[123];

int main()
{
    optimize();

    string s = "asdf";
    doubleHashing d = doubleHashing( s );
    cout << d.rangeHash( 0, sz ( s ) );
```

```
    ///Normal Hashing :

    int p = 31; /// Magical primes : 31,
        41, 37
    pw[0] = 1;
    for ( int i = 1; i <= 12; i++ ) {
        pw[i] = ( p * pw[i-1] ) % MOD;
    }

    hash_s[0] = ( s[0] - 'a' )+1;
    for ( int i = 1; i < sz(s); i++ )
        hash_s[i] = ( hash_s[i-1] + (
        pw[i] * ( s[i] - 'a' + 1 ) ) ) %
        MOD;



    return 0;
}
```

12.4 manachers

```
void manachers(string s, vector<int>
    &d1, vector<int> &d2) {
    int n = s.size();
    d1.resize(n);
    d2.resize(n);
    for (int i = 0, l = 0, r = -1; i <
        n; i++) {
        int k = (i > r) ? 1 : min(d1[l +
            r - i], r - i + 1);
        while (0 <= i - k && i + k < n &&
            s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }
    for (int i = 0, l = 0, r = -1; i <
        n; i++) {
        int k = (i > r) ? 0 : min(d2[l +
            r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k <
            n && s[i - k - 1] == s[i +
            k]) {
            k++;
        }
        d2[i] = k--;
        if (i + k > r) {
            l = i - k - 1;
            r = i + k;
        }
    }
}
```

12.5 suffix$_a$rray

```
vector<int> sort_cyclic_shifts(string
    const& s) {
    int n = s.size();
    const int alphabet = 256;
    vector<int> p(n), c(n),
        cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0) pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() +
            classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur =
                {c[p[i]], c[(p[i] + (1 <<
                h)) % n]};
            pair<int, int> prev =
                {c[p[i-1]], c[(p[i-1] + (1
                << h)) % n]};
```

```
            if (cur != prev) ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int>
    suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts =
        sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

vector<int> lcp_construction(string
    const& s, vector<int> const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n &&
            s[i+k] == s[j+k]) k++;
        lcp[rank[i]] = k;
        if (k) k--;
    }
    return lcp;
}
```

13 utility

13.1 template

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<vi> vvi;
typedef vector<vl> vvl;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<pii> vii;
typedef vector<pll> vll;

#define endl '\n'
#define PB push_back
#define F first
#define S second
#define all(a) (a).begin(), (a).end()
#define rall(a) (a).rbegin(), (a).rend()
#define sz(x) (int) x.size()

const double PI = acos(-1);
const double eps = 1e-9;

const int inf = 2000000000;
const ll infLL = 9000000000000000000;
#define MOD 1000000007

#define mem(a, b) memset(a, b, sizeof(a))
#define sqr(a) ((a) * (a))

#define optimize()
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
#define fraction()
    cout.unsetf(ios::floatfield);
    cout.precision(10);
    cout.setf(ios::fixed,
    ios::floatfield);
#define file() freopen("input.txt", "r",
    stdin); freopen("output.txt", "w",
    stdout);

inline bool checkBit(ll n, int i) {
    return n & (1LL << i); }
inline ll setBit(ll n, int i) { return n
    | (1LL << i); }
inline ll resetBit(ll n, int i) { return
    n & (~(1LL << i)); }

inline void normal(ll &a) { a %= MOD; (a
    < 0) && (a += MOD); }
inline ll modMul(ll a, ll b) { a %= MOD;
    b %= MOD; normal(a); normal(b);
    return (a * b) % MOD; }
inline ll modAdd(ll a, ll b) { a %= MOD;
    b %= MOD; normal(a); normal(b);
    return (a + b) % MOD; }
```

```
inline ll modSub(ll a, ll b) { a %= MOD;
    b %= MOD; normal(a); normal(b); a -=
    b; normal(a); return a; }
inline ll modPow(ll b, ll p) { ll r =
    1LL; while (p) { if (p & 1) r =
    modMul(r, b); b = modMul(b, b); p >>=
    1; } return r; }
inline ll modInverse(ll a) { return
    modPow(a, MOD - 2); }
inline ll modDiv(ll a, ll b) { return
    modMul(a, modInverse(b)); }
```