CODEBOOK CSE 20

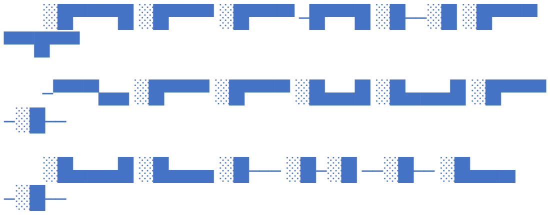
TEAM:NeverEndingHope

*Template:

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

#include<ext/pb_ds/assoc_container.hpp>

#include<ext/pb_ds/tree_policy.hpp>

using namespace std;

using namespace __gnu_pbds;


//VVI

#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

#define pb          push_back

#define ll          long long

#define ff first

#define ss second

#define SZ(a) (int)a.size()

#define UNIQUE(a) (a).erase(unique(all(a)),(a).end())

#define eb emplace_back

#define mp make_pair



///BIT MANIPULATION


#define Set(x, k) (x |= (1LL << k))

#define Unset(x, k) (x &= ~(1LL << k))

#define Check(x, k) (x & (1LL << k))

#define Toggle(x, k) (x ^ (1LL << k))


//LOOPS


#define scl(n)        scanf("%lld", &n)

#define fr(i,n)       for (ll i=0;i<n;i++)

#define fr1(i,n)      for(ll i=1;i<=n;i++)

#define Fo(i,k,n) for(i=k;k<n?i<n:i>n;k<n?i+=1:i-=1)
```

```
///PRINTING


#define deb(x) cout << #x << "=" << x << endl

#define deb2(x, y) cout << #x << "=" << x << "," << #y << "=" << y
<< endl

#define nn '\n'

#define pfl(x)        printf("%lld\n",x)

#define pcas(i)        printf("Case %lld: ",i)

#define Setpre(n) cout<<fixed<<setprecision(n)

#define itr(it, a) for(auto it = a.begin(); it != a.end(); it++)

#define debug          printf("I am here\n")


///SORTING AND FILLING


#define asort(a)        sort(a,a+n)

#define dsort(a)        sort(a,a+n,greater<int>())

#define vasort(v)       sort(v.begin(), v.end());

#define vdsort(v)       sort(v.begin(), v.end(),greater<ll>());

#define rev(x) reverse(all(x))

#define sortall(x) sort(all(x))

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

#define all(x) x.begin(), x.end()

#define rev(x) reverse(all(x))


//CONSTANTS

#define md            10000007

#define PI 3.14159265358979323846 26



///INLINE FUNCTIONS

inline ll GCD(ll a, ll b) { return b == 0 ? a : GCD(b, a % b); }

inline ll LCM(ll a, ll b) { return a * b / GCD(a, b); }

inline ll Ceil(ll p, ll q)  {return p < 0 ? p / q : p / q + !!(p % q);}

inline ll Floor(ll p, ll q) {return p > 0 ? p / q : p / q - !!(p % q);}
inline double logb(ll base,ll num){ return (double)log(num)/(double)log(base);}

inline bool isPerfectSquare(long double x){ if (x >= 0) { long long sr = sqrt(x);return (sr * sr == x); }return false; }

double euclidean_distance(ll x1,ll y1,ll x2,ll y2){double a=(x2-x1)*(x2-x1);double b=(y2-y1)*(y2-y1);double c=(double)sqrt(a+b);return c;}

int popcount(ll x){return __builtin_popcountll(x);};

int poplow(ll x){return __builtin_ctzll(x);};

int pophigh(ll x){return 63 - __builtin_clzll(x);};


typedef unsigned long long ull;

typedef pair<ll, ll>    pll;

typedef vector<ll>      vl;

typedef vector<pll>     vpll;

typedef vector<vl>      vvl;

template <typename T> using PQ = priority_queue<T>;

template <typename T> using QP = priority_queue<T,vector<T>,greater<T>>;


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

template <typename T,typename R> using ordered_map = tree<T, R , less<T>, rb_tree_tag, tree_order_statistics_node_update>;

;

const double EPS = 1e-9;

const ll N = 2e5+10;

const ll M = 1e9+7;



namespace io{

   template<typename First, typename Second> ostream& operator << ( ostream &os, const pair<First, Second> &p ) { return os << p.first << " " << p.second; }

   template<typename First, typename Second> ostream& operator << ( ostream &os, const map<First, Second> &mp ) { for( auto it : mp ) { os << it << endl;  } return os; }

   template<typename First> ostream& operator << ( ostream &os, const vector<First> &v ) { bool space = false; for( First x : v ) { if( space ) os << " "; space = true; os << x; } return os; }

   template<typename First> ostream& operator << ( ostream &os, const set<First> &st ) { bool space = false; for( First x : st ) { if( space ) os << " "; space = true; os << x; } return os; }

   template<typename First> ostream& operator << ( ostream &os, const multiset<First> &st ) { bool space = false; for( First x : st ) { if( space ) os << " "; space = true; os << x; } return os; }


   template<typename First, typename Second> istream& operator >> ( istream &is, pair<First, Second> &p ) { return is >> p.first >> p.second; }

   template<typename First> istream& operator >> ( istream &is, vector<First> &v ) { for( First &x : v ) { is >> x; } return is; }


   long long fastread(){ char c; long long d = 1, x = 0; do c = getchar(); while( c == ' ' || c == '\n' ); if( c == '-' ) c = getchar(), d = -1; while( isdigit( c ) ){ x = x * 10 + c - '0'; c = getchar(); } return d * x; }


   static bool sep = false;


   using std::to_string;


   string to_string( bool x ){ return ( x ? "true" : "false" ); }

   string to_string( const string & s ){ return "\"" + s + "\""; }

   string to_string( const char * s ){ return "\"" + string( s ) + "\""; }

   string to_string ( const char & c ) { string s; s += c; return "\'" + s + "\'"; }


   template<typename Type> string to_string( vector<Type> );

   template<typename First, typename Second> string to_string( pair<First, Second> );

   template<typename Collection> string to_string( Collection );


   template<typename First, typename Second> string to_string( pair<First, Second> p ){ return "{" + to_string( p.first ) + ", " + to_string( p.second ) + "}"; }

   template<typename Type> string to_string( vector<Type> v ) { bool sep = false; string s = "["; for( Type x: v ){ if( sep ) s += ", "; sep = true; s += to_string( x ); } s += "]"; return s; }

   template<typename Collection> string to_string( Collection collection ) { bool sep = false; string s = "{"; for( auto x: collection ){ if( sep ) s += ", "; sep = true; s += to_string( x ); } s += "}"; return s; }
```

```
    void print() { cerr << endl; sep = false; }

    template <typename First, typename... Other> void print( First
first, Other... other ) { if( sep ) cerr << " | "; sep = true; cerr <<
to_string( first ); print( other... ); }


} using namespace io;



/*=====================================================
===============//
```



```
//=====================================================
================*/


void setIO(){

    #ifndef ONLINE_JUDGE

    freopen("input.txt", "r", stdin);


    freopen("output.txt", "w", stdout);

    #endif // ONLINE_JUDGE


}



struct custom_hash {

    static uint64_t splitmix64(uint64_t x) {

        x += 0x9e3779b97f4a7c15;

        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;

        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;

        return x ^ (x >> 31);

    }

    size_t operator()(uint64_t x) const {

        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();

        return splitmix64(x + FIXED_RANDOM);

    }

};


int main()

{

    fast;

     ll t;

    //setIO();

     //ll tno=1;;

     t=1;

    //cin>>t;


    while(t--){


    }



    return 0;

}
```

NAMESPACE POLLARDHO:

```
namespace PollardRho{

    mt19937
rnd(chrono::steady_clock::now().time_since_epoch().count());

    const int P = 1e6 + 9;

    ll seq[P];

    int primes[P], spf[P];

    inline ll add_mod(ll x, ll y, ll m){

        return (x += y) < m ? x : x - m;

    }

    inline ll mul_mod(ll x, ll y, ll m){

        ll res = __int128(x) * y % m;

        return res;
    // ll res = x * y - (ll)((long double)x * y / m + 0.5) * m;
    // return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m){
    ll res = 1 % m;
    for (; n; n >>= 1){
        if (n & 1)
            res = mul_mod(res, x, m);
        x = mul_mod(x, x, m);
    }
    return res;
}
// O(it * (logn)^3), it = number of rounds performed
inline bool miller_rabin(ll n){
    if (n <= 2 || (n & 1 ^ 1))
        return (n == 2);
    if (n < P)
        return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++){}
    // each iteration is a round
    for (int i = 0; primes[i] < n && primes[i] < 32; i++){
        c = pow_mod(primes[i], r, n);
        for (int j = 0; j < s; j++){
            d = mul_mod(c, c, n);
            if (d == 1 && c != 1 && c != (n - 1)) return false;
            c = d;
        }
        if (c != 1) return false;
    }
    return true;
}
void init(){
    int cnt = 0;
    for (int i = 2; i < P; i++){
        if (!spf[i]) primes[cnt++] = spf[i] = i;
        for (int j = 0, k; (k = i * primes[j]) < P; j++){
            spf[k] = primes[j];
            if (spf[i] == spf[k]) break;
        }
    }
}
// returns O(n^(1/4))
ll pollard_rho(ll n){
    while (1){
        ll x = rnd() % n, y = x, c = rnd() % n, u = 1, v, t = 0;
        ll *px = seq, *py = seq;
        while (1){
            *py++ = y = add_mod(mul_mod(y, y, n), c, n);
            *py++ = y = add_mod(mul_mod(y, y, n), c, n);
            if ((x = *px++) == y) break;
            v = u;
            u = mul_mod(u, abs(y - x), n);
            if (!u) return __gcd(v, n);
            if (++t == 32){
                t = 0;
                if ((u = __gcd(u, n)) > 1 && u < n) return u;
            }
        }
        if (t && (u = __gcd(u, n)) > 1 && u < n) return u;
    }
}
vector<ll> factorize(ll n){
    if (n == 1) return vector<ll>();
    if (miller_rabin(n)) return vector<ll>{n};
    vector<ll> v, w;
    while (n > 1 && n < P){
        v.push_back(spf[n]);
        n /= spf[n];
    }
```

```
      if (n >= P){
        ll x = pollard_rho(n);
        v = factorize(x);
        w = factorize(n / x);
        v.insert(v.end(), w.begin(), w.end());
      }
      return v;
   }
}
// using namespace PollardRho;
```

INPUT_OUTPUT:

```
freopen("input.txt","r",stdin);
freopen("output.txt","w",stdout);
```

Clock

```
int st = clock ();
int ed = clock ();
if(ed - st >= CLOCKS_PER_SEC * 1);
```

DISPLAY FUNCTION:

```
template <typename T>
void display (T const& coll)
{
   typename T::const_iterator pos;  // iterator to iterate over coll
   typename T::const_iterator end(coll.end());  // end position
   for (pos=coll.begin(); pos!=end; ++pos) {
      cout << *pos << ' ';
   }
   cout << endl;
}
```

***BASICS: (LOOPS AND RECURSION & ARRAY)

SUBSET GENERATION:

```
vvl subsets;
vl v;

void generate(vl &subset,ll i){
   if(i==v.size()){
      subsets.push_back(subset);
      return;
   }
   generate(subset,i+1);
   subset.push_back(v[i]);
   generate(subset,i+1);
   subset.pop_back();
}
```

PERMUTATION GENERATION:

Use next_permutation(vec.begin(),vec.end());

Generate Combination:

```
vl people;
vl combination;
vvl combinations;

void go(ll offset, ll k) {
   if (k == 0) {
      combinations.pb(combination);
      return;
   }
   for (ll i = offset; i <= people.size() - k; ++i) {
      combination.push_back(people[i]);
      go(i+1, k-1);
      combination.pop_back();
   }
}
```

Subset Sum:

```
bitset<N> can;

cin>>n>>k;

can[0]=true;

fo(i,n){

    int x;cin>>x;

    can|=(can<<x);

}

cout<<(can[k]?"YES\n":"NO\n");
```

PREFIX SUM

Prefix sum:

```
vector<ll> prefixsum(vector<ll>&vec,ll n){

   vector<ll>pref(n);

   pref[0]=vec[0];

   for(ll i=1;i<n;i++){

       pref[i]=vec[i]+pref[i-1];

   }

   return pref;


}
```

2D prefix sum:

```
ll arr[N][N];

ll pfsum[N][N];

void buildPS(){

    for(int i=1;i<N;i++){

        for(int j=1;j<N;j++){

            pfsum[i][j]=arr[i][j]+pfsum[i-1][j]+pfsum[i][j-1]-pfsum[i-1][j-1];

        }

    }

}

ll getSum(ll a,ll b,ll c,ll d){

    return pfsum[c][d]-pfsum[a-1][d]-pfsum[c][b-1]+pfsum[a-1][b-1];

}
```

SUBARRAY RELATED:

Z-FUNCTION:

```
// An element Z[i] of Z array stores length of the longest substring

// starting from str[i] which is also a prefix of str[0..n-1].

// The first entry of Z array is meaning less as complete string is always prefix of itself.

// Here Z[0]=0.

vector<int> z_function(string s) {

  int n = (int) s.length();

  vector<int> z(n);

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

    if (i <= r)

      z[i] = min (r - i + 1, z[i - l]);

    while (i + z[i] < n && s[z[i]] == s[i + z[i]])

      ++z[i];

    if (i + z[i] - 1 > r)

      l = i, r = i + z[i] - 1;

  }

  return z;

}
```

SUBARRAY SUM:

```
ll findSubarraySum(vector<ll> &vec, ll n, ll sum)

{

    ll m=0,cnt=0;

    map<ll,ll>mp;

    for(int i=0;i<n;++i)

    {
```

```
    m+=vec[i];
    if(m==sum)cnt++;
    if(mp.count(m-sum))
    {
      cnt+=mp[m-sum];
    }
    mp[m]++;
  }
  cout<<cnt<<endl;
}
```

TWO POINTERs:

Pair such that sum of the pair=k:

```
bool pairsum(int ar[],int n,int k){
  int L=0,R=n-1;
  while(L<R){
    if(ar[L]+ar[R]==k){
      cout<<L<<" "<<R<<endl;
      return true;
    }
    else if(ar[L]+ar[R] >k)
      R--;
    else
      L++;
  }
  return false;
}
```

KADANE ALGORITHM:

```
ll kadane(vl &vc,ll n){
  ll sum,currSum,i=0;
  sum=currSum=vc[0];
  Fo(i,1,n){
    currSum=max(currSum+vc[i],vc[i]);
    sum=max(sum,currSum);
    // currSum= (currSum<0? 0:currSum);
  }
  return sum;
}
```

SLIDING WINDOW:

Maximum element of subarray length K:

```
vector<int> maxSlidingWindow(vector<int> &nums, int k) {
        multiset<int> s;
        vector<int> ret;
        for (int i = 0; i < k; i++) { s.insert(nums[i]); }
        for (int i = k; i < nums.size(); i++) {
                ret.push_back(*s.rbegin());
                s.erase(s.find(nums[i - k]));
                s.insert(nums[i]);
        }
        ret.push_back(*s.rbegin());
        return ret;
}
```

NEXT GREATER ELEMENT:

```
vector<int> NGE(vector<int> v){
  vector<int> nge(v.size());
  stack<int> st;
  for(int i=0;i<v.size();i++){
    while(!st.empty() && v[i]>v[st.top()]){
      nge[st.top()]=i;
      st.pop();
    }
    st.push(i);
  }
  while(!st.empty()){
    nge[st.top()]=-1;
    st.pop();
  }
  return nge;
}
void print(vector<int> &v,int n){
  vector<int> nge=NGE(v);
  for(int i=0;i<n;i++){
    cout<<v[i]<<" "<<((nge[i]==-1)?-1 :v[nge[i]]) <<endl;
  }
}
```

Inversion count(number of pair of index i,j where i<j && v[i]>v[j]):

Ordered set:

```
// Returns inversion count in v[0..n-1]
ll getInvCount(vl &v,ll n){
    ordered_set<pll> st;
    ll invcount = 0;
    for(ll i = 0; i < n; i++) {
        ll temp=st.order_of_key({v[i], -1});
        temp=(ll)st.size()-temp;
        invcount+=temp;
        st.insert({v[i], i});
    }
    return invcount;
}
```

Using BIT:

```
ll BITree[100009];


///do this for range: getSum(r) - getSum(l - 1)
```

```
ll getSum(ll index){
    ll sum = 0; // Iniialize result
    // Traverse ancestors of BITree[index]
    while (index>0){
        sum += BITree[index];  // Add current element of BITree to sum
        index -= index & (-index);  // Move index to parent node in getSum View
    }
    return sum;
}


void updateBIT(ll n, ll index, ll val){
    // Traverse all ancestors and add 'val'
    while (index <= n){
      // Add 'val' to current node of BI Tree
      BITree[index] += val;
      // Update index to that of parent in update View
      index += index & (-index);
    }
}


void compress(vl &v,ll n){
    ll i;
    set<pll> st;
    fo(i,n){
        st.insert({v[i],i});
    }
    i=1;
    for(auto &it:st){
        v[it.second]=i;
        i++;
    }
}

ll getInvCount(vl &v,ll n){
    ll invcount=0;
    compress(v,n);
    for(int i=1;i<=n;i++){
        BITree[i]=0;
    }
    for(int i=n-1;i>=0;i--){
        invcount+=getSum(v[i]-1);
        updateBIT(n,v[i],1);
    }
    return invcount;
}
```

SUBSEQUENCE RELATED:

Length of LIS(O(nlogn):

```
ll lis(vl &v){
    if (v.empty()) return 0;
    vl tail(v.size(), 0);
    int length = 1; // always points empty slot in tail
    tail[0] = v[0];
    for(int i=1;i<v.size();i++){
        auto b = tail.begin(), e = tail.begin() + length;
        auto it = lower_bound(b, e, v[i]);
        if (it == tail.begin() + length) tail[length++] = v[i];
        else *it = v[i];
    }
    return length;
}
```

***SPARSE TABLE:

```
ll table[N][19], ar[N];//note: ar is 1 based
void build(ll n) {
  for(ll i = 1; i <= n; ++i) table[i][0] = ar[i];
  for(ll k = 1; k < 19; ++k) {
    for(ll i = 1; i + (1 << k) - 1 <= n; ++i) {
      table[i][k] = GCD(table[i][k - 1], table[i + (1 << (k - 1))][k - 1]);
    }
  }
}

ll query(ll l, ll r) {
  ll k = 31 - __builtin_clz(r - l + 1);
  return GCD(table[l][k], table[r - (1 << k) + 1][k]);
}
```

SPARSE TABLE 2D:

```
const int LG = 10;

int st[N][N][LG][LG];
int a[N][N], lg2[N];

int yo(int x1, int y1, int x2, int y2)
{
    x2++;
    y2++;
    int a = lg2[x2 - x1], b = lg2[y2 - y1];
    return max(
        max(st[x1][y1][a][b], st[x2 - (1 << a)][y1][a][b]),
        max(st[x1][y2 - (1 << b)][a][b], st[x2 - (1 << a)][y2 - (1 << b)][a][b]));
}

void build(int n, int m)
{ // 0 indexed
    for (int i = 2; i < N; i++)
        lg2[i] = lg2[i >> 1] + 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            st[i][j][0][0] = a[i][j];
        }
    }
    for (int a = 0; a < LG; a++)
    {
        for (int b = 0; b < LG; b++)
        {
            if (a + b == 0)
                continue;
            for (int i = 0; i + (1 << a) <= n; i++)
            {
                for (int j = 0; j + (1 << b) <= m; j++)
                {
                    if (!a)
                    {
                        st[i][j][a][b] = max(st[i][j][a][b - 1], st[i][j + (1 << (b - 1))][a][b - 1]);
                    }
                    else
                    {
                        st[i][j][a][b] = max(st[i][j][a - 1][b], st[i + (1 << (a - 1))][j][a - 1][b]);
                    }
                }
            }
        }
    }
}
```

***Hashing:

```
#define MAXLEN 1000010
constexpr uint64_t mod = (1ULL << 61) - 1;
const uint64_t seed = chrono::system_clock::now().time_since_epoch().count();
const uint64_t base = mt19937_64(seed)() % (mod / 3) + (mod / 3);
uint64_t base_pow[MAXLEN];
int64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a >> 32, l2 = (uint32_t)b, h2 = b >> 32;
    uint64_t l = l1 * l2, m = l1 * h2 + l2 * h1, h = h1 * h2;
    uint64_t ret = (l & mod) + (l >> 61) + (h << 3) + (m >> 29) + (m << 35 >> 3) + 1;
    ret = (ret & mod) + (ret >> 61);
    ret = (ret & mod) + (ret >> 61);
    return ret - 1;
}
void init(){
    base_pow[0] = 1;
    for (int i = 1; i < MAXLEN; i++){
        base_pow[i] = modmul(base_pow[i - 1], base);
    }
}
struct PolyHash{
    /// Remove suff vector and usage if reverse hash is not required for more speed
    vector<int64_t> pref, suff;
    PolyHash() {}
    template <typename T>
    PolyHash(const vector<T>& ar){
        if (!base_pow[0]) init();
        int n = ar.size();
        assert(n < MAXLEN);
        pref.resize(n + 3, 0), suff.resize(n + 3, 0);
        for (int i = 1; i <= n; i++){
            pref[i] = modmul(pref[i - 1], base) + ar[i - 1] + 997;
            if (pref[i] >= mod) pref[i] -= mod;
        }
        for (int i = n; i >= 1; i--){
            suff[i] = modmul(suff[i + 1], base) + ar[i - 1] + 997;
            if (suff[i] >= mod) suff[i] -= mod;
        }
    }
    PolyHash(const char* str)
        : PolyHash(vector<char> (str, str + strlen(str))) {}
    uint64_t get_hash(int l, int r){
        int64_t h = pref[r + 1] - modmul(base_pow[r - l + 1], pref[l]);
        return h < 0 ? h + mod : h;
    }
    uint64_t rev_hash(int l, int r){
        int64_t h = suff[l + 1] - modmul(base_pow[r - l + 1], suff[r + 2]);
        return h < 0 ? h + mod : h;
    }
};
```

Custom hash for unordered map:

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

DOUBLE HASH:

```
const ll MAX_N = 1e6+10, mod = 2e9+63, base1 = 1e9+21, base2 = 1e9+181;
char s[MAX_N];  // 1-indexed
ll pw1[MAX_N], pw2[MAX_N], slen;

void pw_calc() {
    pw1[0] = pw2[0] = 1;
    for(int i = 1; i < MAX_N; i++) {
        pw1[i] = (pw1[i-1] * base1) % mod;
        pw2[i] = (pw2[i-1] * base2) % mod;
    }
}

struct Hash {
    ll h1[MAX_N], h2[MAX_N];

    void init() {
        h1[0] = h2[0] = 0;
        for(int i = 1; i <= slen; i++) {
            h1[i] = (h1[i-1] * base1 + s[i]) % mod;
            h2[i] = (h2[i-1] * base2 + s[i]) % mod;
        }
    }

    inline ll hashVal(int l, int r) {
        ll hsh1 = (h1[r] - h1[l-1] * pw1[r-l+1]) % mod;
        if(hsh1 < 0) hsh1 += mod;
        ll hsh2 = (h2[r] - h2[l-1] * pw2[r-l+1]) % mod;
        if(hsh2 < 0) hsh2 += mod;
        return (hsh1 << 32) | hsh2;
    }

    inline ll hashOne(int l, int r) {
        ll hsh1 = (h1[r] - h1[l-1] * pw1[r-l+1]) % mod;
        if(hsh1 < 0) hsh1 += mod;
        return hsh1;
    }

    inline ll hashTwo(int l, int r) {
        ll hsh2 = (h2[r] - h2[l-1] * pw2[r-l+1]) % mod;
        if(hsh2 < 0) hsh2 += mod;
        return hsh2;
    }
} fw;
```

```
/* call pw_calc() for calculating powers
less than MAX_N

 * slen = strlen(s+1);   --> string length

 * fw.init() will calculate the double
hashes

 * fw.hashVal(l,r) will return [l,,r]
merged double hash value

 * fw.hashOne(l, r) will return [l,,r]
base1 hash

 * fw.hashTwo(l, r) will return [l,,r]
base2 hash

*/
```

***SUFFIX ARRAY RELATED:

O(nlogn):

```
#define MAX_N 1000020
int n, t;
char s[MAX_N];
int SA[MAX_N], LCP[MAX_N];
int RA[MAX_N], tempRA[MAX_N];
int tempSA[MAX_N];
int c[MAX_N];
int Phi[MAX_N], PLCP[MAX_N];


void countingSort(int k) {   // O(n)
    int i, sum, maxi = max(300, n);
    // up to 255 ASCII chars or length of n
    memset(c, 0, sizeof c);
    // clear frequency table
    for (i = 0; i < n; i++)
    // count the frequency of each integer rank
    c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++) {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (i = 0; i < n; i++)
        // shuffle the suffix array if necessary
        tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] =
SA[i];


    for (i = 0; i < n; i++)
        // update the suffix array SA
        SA[i] = tempSA[i];
}


void buildSA() {
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = s[i];
    // initial rankings
    for (i = 0; i < n; i++) SA[i] = i;
    // initial SA: {0, 1, 2, ..., n-1}
    for (k = 1; k < n; k <<= 1) {
        // repeat sorting process log n times
        countingSort(k); // actually radix sort: sort based
on the second item
        countingSort(0);
        // then (stable) sort based on the first item
        tempRA[SA[0]] = r = 0;
        // re-ranking; start from rank r = 0
        for (i = 1; i < n; i++)
            // compare adjacent suffixes
            tempRA[SA[i]] = // if same pair => same rank
r; otherwise, increase r
```

```
            (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] + k]
== RA[SA[i - 1] + k]) ? r : ++r;
    for (i = 0; i < n; i++)
      // update the rank array RA
      RA[i] = tempRA[i];


    if (RA[SA[n - 1]] == n - 1) break;
    // nice optimization trick
  }
}


void buildLCP() {
  int i, L;
  Phi[SA[0]] = -1;
  // default value
  for (i = 1; i < n; i++)
    // compute Phi in O(n)
    Phi[SA[i]] = SA[i - 1];
  // remember which suffix is behind this suffix
  for (i = L = 0; i < n; i++) {
    // compute Permuted LCP in O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }
    // special case
    while (s[i + L] == s[Phi[i] + L]) L++;
    // L increased max n times
    PLCP[i] = L;
    L = max(L - 1, 0);
    // L decreased max n times
  }
  for (i = 0; i < n; i++)
    // compute LCP in O(n)
    LCP[i] = PLCP[SA[i]];
    // put the permuted LCP to the correct position
}
// n = string length + 1
// s = the string
// memset(LCP, 0, sizeof(LCP));  setting all index of
LCP to zero
// buildSA(); for building suffix array
// buildLCP(); for building LCP array
// LCP is the longest common prefix with the
previous suffix here
// SA[0] holds the empty suffix "\0".
```

O(n):

```
#define MAXN 1010


char s[MAXN + 1];


int AN;
int A[3 * MAXN + 100];
int cnt[MAXN + 1]; // Should be >= 256
int SA[MAXN + 1], REVSA[MAXN + 1], LCP[MAXN +
1];


/* Used by suffix_array. */
void radix_pass(int* A, int AN, int* R, int RN, int* D) {
  memset(cnt, 0, sizeof(int) * (AN + 1));
  int* C = cnt + 1;
  for(int i = 0; i < RN; i++) ++C[A[R[i]]];
  for(int i = -1, v = 0; i <= AN && v < RN; v += C[i++])
swap(v, C[i]);
  for(int i = 0; i < RN; i++) D[C[A[R[i]]]++] = R[i];
```

```
}


/* DC3 in O(N) using 20N bytes of memory.  Stores the suffix array of the string
 * [A,A+AN) into SA where SA[i] (0<=i<=AN) gives the starting position of the
 * i-th least suffix of A (including the empty suffix).
 */
void suffix_array(int* A, int AN) {
  // Base case... length 1 string.
  if(!AN) {
    SA[0] = 0;
  } else if(AN == 1) {
    SA[0] = 1; SA[1] = 0;
    return;
  }

  // Sort all strings of length 3 starting at non-multiples of 3 into R.
  int RN = 0;
  int* SUBA = A + AN + 2;
  int* R = SUBA + AN + 2;
  for(int i = 1; i < AN; i += 3) SUBA[RN++] = i;
  for(int i = 2; i < AN; i += 3) SUBA[RN++] = i;
  A[AN + 1] = A[AN] = -1;
  radix_pass(A + 2, AN - 2, SUBA, RN, R);
  radix_pass(A + 1, AN - 1, R, RN, SUBA);
  radix_pass(A + 0, AN - 0, SUBA, RN, R);

  // Compute the relabel array if we need to recursively solve for the
  // non-multiples.
  int resfix, resmul, v;
  if(AN % 3 == 1) {
    resfix = 1; resmul = RN >> 1;
  } else {
    resfix = 2; resmul = RN + 1 >> 1;
  }
  for(int i = v = 0; i < RN; i++) {
    v += i && (A[R[i - 1] + 0] != A[R[i] + 0] ||
           A[R[i - 1] + 1] != A[R[i] + 1] ||
           A[R[i - 1] + 2] != A[R[i] + 2]);
    SUBA[R[i] / 3 + (R[i] % 3 == resfix) * resmul] = v;
  }

  // Recursively solve if needed to compute relative ranks in the final suffix
  // array of all non-multiples.
  if(v + 1 != RN) {
    suffix_array(SUBA, RN);
    SA[0] = AN;
    for(int i = 1; i <= RN; i++) {
      SA[i] = SA[i] < resmul ? 3 * SA[i] + (resfix==1?2:1) :
          3 * (SA[i] - resmul) + resfix;
    }
  } else {
    SA[0] = AN;
    memcpy(SA + 1, R, sizeof(int) * RN);
  }


  // Compute the relative ordering of the multiples.
  int NMN = RN;
```

```
    for(int i = RN = 0; i <= NMN; i++) {
        if(SA[i] % 3 == 1) {
        SUBA[RN++] = SA[i] - 1;
        }
    }
    radix_pass(A, AN, SUBA, RN, R);

    // Compute the reverse SA for what we know so far.
    for(int i = 0; i <= NMN; i++) {
        SUBA[SA[i]] = i;
    }

    // Merge the orderings.
    int ii = RN - 1;
    int jj = NMN;
    int pos;
    for(pos = AN; ii >= 0; pos--) {
        int i = R[ii];
        int j = SA[jj];
        int v = A[i] - A[j];
        if(!v) {
            if(j % 3 == 1) {
                v = SUBA[i + 1] - SUBA[j + 1];
            } else {
                v = A[i + 1] - A[j + 1];
                if(!v) v = SUBA[i + 2] - SUBA[j + 2];
            }
        }
        SA[pos] = v < 0 ? SA[jj--] : R[ii--];
    }
```

```
}

/* Copies the string in s into A and reduces the characters as needed. */
void prep_string() {
    int v = AN = 0;
    memset(cnt, 0, 256 * sizeof(int));
    for(char* ss = s; *ss; ++ss, ++AN) cnt[*ss]++;
    for(int i = 0; i < AN; i++) cnt[s[i]]++;
    for(int i = 0; i < 256; i++) cnt[i] = cnt[i] ? v++ : -1;
    for(int i = 0; i < AN; i++) A[i] = cnt[s[i]];
}

/* Computes the reverse SA index.  REVSA[i] gives the index of the suffix
 * starting a i in the SA array.  In other words, REVSA[i] gives the number of
 * suffixes before the suffix starting at i.  This can be useful in itself but
 * is also used for compute_lcp().
 */
void compute_reverse_sa() {
    for(int i = 0; i <= AN; i++) {
        REVSA[SA[i]] = i;
    }
}

/* Computes the longest common prefix between adjacent suffixes. LCP[i] gives
 * the length of the longest common prefix between the suffix starting at
 * SA[i-1] and SA[i]. Runs in O(N) time.
 */
```

```
void compute_lcp() {
    int len = 0;
    for(int i = 0; i < AN; i++, len = max(0, len - 1)) {
        int s = REVSA[i];
        int j = SA[s - 1];
        for(; i + len < AN && j + len < AN && A[i + len] ==
A[j + len]; len++);
        LCP[s] = len;
    }
}
```

```
/* Call these inside test case:
 *   prep_string();
 *   suffix_array(A, len);
 *   compute_reverse_sa();
 *   compute_lcp();
*/
```

***BIT MANIPULATION % Binary numbers related:

Is set:

```
bool isSet(int x,int i){
   return (x&(1<<i));
}
```

Print binary:

```
void printBin(int num){
   for(int i=10;i>=0;i--){
      cout<<((num>>i)&1);
   }
   cout<<endl;
}
```

Toggle bit:

```
int toggle(int x,int i){
   return (x^(1<<i));
}
```

Unset finction:

```
int unset(int x,int i){
   return (x&(~(1<<i)));
}
```

Set bit:

```
int setBit(int x,int i){
   return (x|(1<<i));
}
```

XOR Basis:

```
const int mxdigit = 50;
vector <ll > b(mxdigit + 1);
void add(ll t){
for(int i = mxdigit; i >= 0; i--){
if(!(1LL << i & t)) continue;
if(b[i] != 0){
t ^= b[i];
continue;
}
for(int j = 0; j < i; j++){
if(1LL << j & t) t ^= b[j];
}
for(int j = i + 1; j <= mxdigit; j
++){
if(1LL << i & b[j]) b[j] ^= t;
}b
[i] = t;
break;
}
}
```

Binary Exponentiation:

```
int binexp(int a, int b){
    int result=1;
    while(b>0){
        if(b&1){
            result=(result * 1LL * a) % M;
        }
        a = (a * 1LL *a) % M;
        b>>=1;
    }
    return result;
}
```

Binary Multiply:

```
ll binMultiply(ll a,ll b){
    ll ans=0;
    while(b>0){
        if(b&1) ans=(ans+a)%M;
        a=(a+a)%M;
        b>>=1;
    }
    return ans;
}
```

XOR OF ALL ELEMENTS OF SUBARRAY=SUM OF ALL ELEMENTS:

```
ll calc(vl arr){
  ll n=SZ(arr);
  ll right = 0, ans = 0,
     num = 0;
  for (ll left = 0; left < n; left++) {
    while (right < n&& num + arr[right]== (num ^ arr[right])) {
      num += arr[right];
      right++;
    }
    ans += right - left;
    if (left == right)
      right++;
    else
      num -= arr[left];
  }
  return ans;
}
```

XOR SUM FIND TWO NUMS:

```
void findNums(ll X, ll Y)
{

  // Initialize the two numbers
  ll A, B;

  // Case 1: X < Y
  if (X < Y) {
    A = -1;
    B = -1;
  }

  // Case 2: X-Y is odd
  else if (abs(X - Y) & 1) {
    A = -1;
    B = -1;
  }
```

```
    // Case 3: If both Sum and XOR
    // are equal
    else if (X == Y) {
        A = 0;
        B = Y;
    }

    // Case 4: If above cases fails
    else {

        // Update the value of A
        A = (X - Y) / 2;

        // Check if A & Y value is 0
        if ((A & Y) == 0) {

            // If true, update B
            B = (A + Y);
        }

        // Otherwise assign -1 to A,
        // -1 to B
        else {
            A = -1;
            B = -1;
        }
    }

    // Print the numbers A and B
    if(A<0 || B<0 ) cout<<-1<<endl;
    else cout << A << " " << B<<endl;
}
```

***Binary Indexed Tree/Fenwick Tree:

```
ll BITree[100009];

///do this for range: getSum(r) - getSum(l - 1)
ll getSum(ll index){
    ll sum = 0; // Iniialize result
    // Traverse ancestors of BITree[index]
    while (index>0){
        sum += BITree[index];  // Add current element of BITree to sum
        index -= index & (-index);  // Move index to parent node in getSum View
    }
    return sum;
}

void updateBIT(ll n, ll index, ll val){
    // Traverse all ancestors and add 'val'
    while (index <= n){
       // Add 'val' to current node of BI Tree
       BITree[index] += val;
       // Update index to that of parent in update View
       index += index & (-index);
    }
}
```

***BIT(2D):

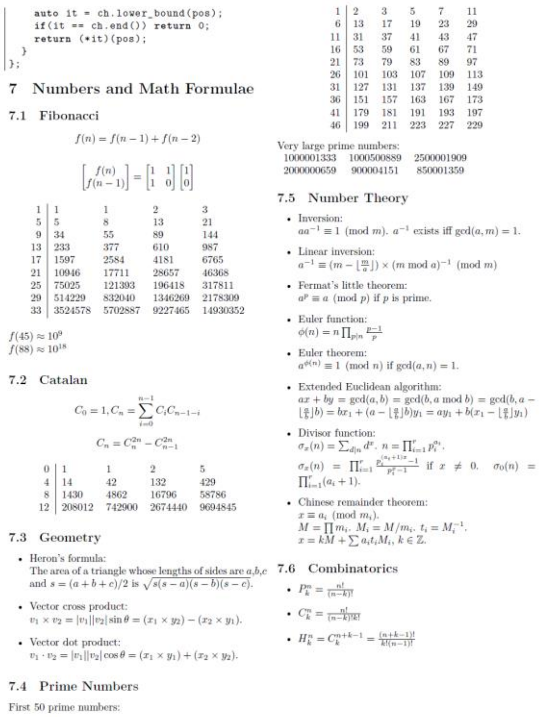
```
struct BIT2D
{
    long long M[N][N][2], A[N][N][2];
    BIT2D()
    {
        memset(M, 0, sizeof M);
        memset(A, 0, sizeof A);
    }
    void upd2(long long t[N][N][2], int x, int y, long long mul, long long add)
    {
        for (int i = x; i < N; i += i & -i)
        {
            for (int j = y; j < N; j += j & -j)
            {
                t[i][j][0] += mul;
                t[i][j][1] += add;
            }
        }
    }
    void upd1(int x, int y1, int y2, long long mul, long long add)
    {
        upd2(M, x, y1, mul, -mul * (y1 - 1));
        upd2(M, x, y2, -mul, mul * y2);
        upd2(A, x, y1, add, -add * (y1 - 1));
        upd2(A, x, y2, -add, add * y2);
    }
    void upd(int x1, int y1, int x2, int y2, long long val) // add val from top-left(x1, y1) to bottom-right (x2, y2);
    {
        upd1(x1, y1, y2, val, -val * (x1 - 1));
        upd1(x2, y1, y2, -val, val * x2);
    }
    long long query2(long long t[N][N][2], int x, int y)
    {
        long long mul = 0, add = 0;
        for (int i = y; i > 0; i -= i & -i)
        {
            mul += t[x][i][0];
            add += t[x][i][1];
        }
        return mul * x + add;
    }
    long long query1(int x, int y)
    {
        long long mul = 0, add = 0;
        for (int i = x; i > 0; i -= i & -i)
        {
            mul += query2(M, i, y);
            add += query2(A, i, y);
        }
        return mul * x + add;
    }
    long long query(int x1, int y1, int x2, int y2) // output sum from top-left(x1, y1) to bottom-right (x2, y2);
    {
        return query1(x2, y2) - query1(x1 - 1, y2) - query1(x2, y1 - 1) + query1(x1 - 1, y1 - 1);
    }
}
```

Search(BIT):

```
// This is equivalent to calculating lower_bound on prefix sums array
// LOGN = log(N)

int bit[N]; // BIT array

int bit_search(int v)
{
    int sum = 0;
    int pos = 0;

    for(int i=LOGN; i>=0; i--)
    {
        if(pos + (1 << i) < N and sum + bit[pos + (1 << i)] < v)
        {
            sum += bit[pos + (1 << i)];
            pos += (1 << i);
        }
```

```
   }

   return pos + 1; // +1 because 'pos' will have position of largest value less than 'v'

}
```

********MATH AND GEOMETRY************



```
      auto it = ch.lower_bound(pos);
      if(it == ch.end()) return 0;
      return (*it)(pos);
   }
};
```

7 Numbers and Math Formulae

7.1 Fibonacci

$$f(n) = f(n-1) + f(n-2)$$

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

1	1	1	2	3
5	5	8	13	21
9	34	55	89	144
13	233	377	610	987
17	1597	2584	4181	6765
21	10946	17711	28657	46368
25	75025	121393	196418	317811
29	514229	832040	1346269	2178309
33	3524578	5702887	9227465	14930352

$f(45) \approx 10^9$
$f(88) \approx 10^{18}$

7.2 Catalan

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

$$C_n = C_n^{2n} - C_{n-1}^{2n}$$

0	1	1	2	5
4	14	42	132	429
8	1430	4862	16796	58786
12	208012	742900	2674440	9694845

7.3 Geometry

- Heron's formula:
  The area of a triangle whose lengths of sides are a,b,c and $s = (a+b+c)/2$ is $\sqrt{s(s-a)(s-b)(s-c)}$.
- Vector cross product:
  $v_1 \times v_2 = |v_1||v_2| \sin\theta = (x_1 \times y_2) - (x_2 \times y_1)$.
- Vector dot product:
  $v_1 \cdot v_2 = |v_1||v_2| \cos\theta = (x_1 \times y_1) + (x_2 \times y_2)$.

7.4 Prime Numbers

First 50 prime numbers:

1	2	3	5	7	11
6	13	17	19	23	29
11	31	37	41	43	47
16	53	59	61	67	71
21	73	79	83	89	97
26	101	103	107	109	113
31	127	131	137	139	149
36	151	157	163	167	173
41	179	181	191	193	197
46	199	211	223	227	229

Very large prime numbers:

1000001333	1000500889	2500001909
2000000659	900004151	850001359

7.5 Number Theory

- Inversion:
  $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.
- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:
  $a^p \equiv a \pmod{p}$ if p is prime.
- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.
- Extended Euclidean algorithm:
  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^{r} p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^{r} (a_i + 1)$.
- Chinese remainder theorem:
  $x \equiv a_i \pmod{m_i}$.
  $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
  $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.

7.6 Combinatorics

- $P_k^n = \frac{n!}{(n-k)!}$
- $C_k^n = \frac{n!}{(n-k)!k!}$
- $H_k^n = C_k^{n+k-1} = \frac{(n+k-1)!}{k!(n-1)!}$

Handling Big and Complex numbers:

```
typedef long double ld;

typedef complex<ld> pt;

const int MOD = 1e9 + 7;


template <class T>

struct cplx

{

   T x, y;

   cplx()

{

   x = 0.0;

   y = 0.0;

}

cplx(T nx, T ny = 0)

{

   x = nx;

   y = ny;

}

cplx operator+(const cplx &c) const

{

   return {x + c.x, y + c.y};

}

cplx operator-(const cplx &c) const

{

   return {x - c.x, y - c.y};

}

cplx operator*(const cplx &c) const

{

   return {x * c.x - y * c.y, x * c.y + y * c.x};

}

cplx &operator*=(const cplx &c)

{

   return *this = {x * c.x - y * c.y, x * c.y + y * c.x};

}

inline T real() const

{

   return x;

}

inline T imag() const

{

   return y;

}

// Only supports right scalar multiplication like p*c

template <class U>
```

```
    cplx operator*(const U &c) const
    {
        return {x * c, y * c};
    }
    template <class U>
    cplx operator/(const U &c) const
    {
        return {x / c, y / c};
    }
    template <class U>
    void operator/=(const U &c)
    {
        x /= c;
        y /= c;
    }
};
#define polar(r, a) \
    (cplx<ld>) { r *cos(a), r *sin(a) }

const int DIG = 9, FDIG = 4;
const int BASE = 1e9, FBASE = 1e4;
typedef cplx<ld> Cplx;

// use mulmod when taking mod by int v and v>2e9
// you can use mod by bigint in that case too
struct BigInt
{
    int sgn;
    vector<int> a;
    BigInt() : sgn(1) {}
    BigInt(ll v)
    {
        *this = v;
    }
    BigInt &operator=(ll v)
    {
        sgn = 1;
        if (v < 0)
            sgn = -1, v = -v;
        a.clear();
        for (; v > 0; v /= BASE)
            a.push_back(v % BASE);
        return *this;
    }
    BigInt(const BigInt &other)
    {
        sgn = other.sgn;
        a = other.a;
    }
    friend void swap(BigInt &a, BigInt &b)
    {
        swap(a.sgn, b.sgn);
        swap(a.a, b.a);
    }
    BigInt &operator=(BigInt other)
    {
        swap(*this, other);
        return *this;
    }
    BigInt(BigInt &&other) : BigInt()
    {
        swap(*this, other);
    }
    BigInt(const string &s)
    {
        read(s);
    }
    void read(const string &s)
    {
        sgn = 1;
        a.clear();
        int k = 0;
        for (; k < s.size() && (s[k] == '-' || s[k] == '+'); k++)
            if (s[k] == '-')
                sgn = -sgn;
        for (int i = s.size() - 1; i >= k; i -= DIG)
        {
            int x = 0;
            for (int j = max(k, i - DIG + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend istream &operator>>(istream &in, BigInt &v)
    {
        string s;
        in >> s;
        v.read(s);
        return in;
    }
    friend ostream &operator<<(ostream &out, const BigInt &v)
    {
        if (v.sgn == -1 && !v.zero())
            out << '-';
        out << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int)v.a.size() - 2; i >= 0; --i)
            out << setw(DIG) << setfill('0') << v.a[i];
        return out;
    }
    bool operator<(const BigInt &v) const
    {
        if (sgn != v.sgn)
            return sgn < v.sgn;
        if (a.size() != v.a.size())
            return a.size() * sgn < v.a.size() * v.sgn;
        for (int i = (int)a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sgn < v.a[i] * sgn;
        return 0;
    }
    bool operator>(const BigInt &v) const
    {
        return v < *this;
    }
    bool operator<=(const BigInt &v) const
    {
        return !(v < *this);
    }
    bool operator>=(const BigInt &v) const
    {
        return !(*this < v);
    }
    bool operator==(const BigInt &v) const
    {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const BigInt &v) const
    {
        return *this < v || v < *this;
    }
    friend int __cmp(const BigInt &x, const BigInt &y)
    {
        if (x.a.size() != y.a.size())
            return x.a.size() < y.a.size() ? -1 : 1;
        for (int i = (int)x.a.size() - 1; i >= 0; --i)
            if (x.a[i] != y.a[i])
                return x.a[i] < y.a[i] ? -1 : 1;
        return 0;
    }
```

```
BigInt operator-() const
{
    BigInt res = *this;
    if (zero())
        return res;
    res.sgn = -sgn;
    return res;
}

void __add(const BigInt &v)
{
    if (a.size() < v.a.size())
        a.resize(v.a.size(), 0);
    for (int i = 0, carry = 0; i < max(a.size(), v.a.size()) || carry; ++i)
    {
        if (i == a.size())
            a.push_back(0);
        a[i] += carry + (i < (int)v.a.size() ? v.a[i] : 0);
        carry = a[i] >= BASE;
        if (carry)
            a[i] -= BASE;
    }
}

void __sub(const BigInt &v)
{
    for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i)
    {
        a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
        carry = a[i] < 0;
        if (carry)
            a[i] += BASE;
    }
    this->trim();
}

BigInt operator+=(const BigInt &v)
{
    if (sgn == v.sgn)
        __add(v);
    else if (__cmp(*this, v) >= 0)
        __sub(v);
    else
    {
        BigInt vv = v;
        swap(*this, vv);
        __sub(vv);
    }
    return *this;
}

BigInt operator-=(const BigInt &v)
{
    if (sgn == v.sgn)
    {
        if (__cmp(*this, v) >= 0)
            __sub(v);
        else
        {
            BigInt vv = v;
            swap(*this, vv);
            __sub(vv);
            sgn = -sgn;
        }
    }
    else
        __add(v);
    return *this;
}
```

```
template <typename L, typename R>
typename enable_if<
    is_convertible<L, BigInt>::value &&
        is_convertible<R, BigInt>::value &&
        is_lvalue_reference<R &&>::value,
    BigInt>::type friend
operator+(L &&l, R &&r)
{
    BigInt result(forward<L>(l));
    result += r;
    return result;
}
template <typename L, typename R>
typename enable_if<
    is_convertible<L, BigInt>::value &&
        is_convertible<R, BigInt>::value &&
        is_rvalue_reference<R &&>::value,
    BigInt>::type friend
operator+(L &&l, R &&r)
{
    BigInt result(move(r));
    result += l;
    return result;
}
template <typename L, typename R>
typename enable_if<
    is_convertible<L, BigInt>::value &&
        is_convertible<R, BigInt>::value,
    BigInt>::type friend
operator-(L &&l, R &&r)
{
    BigInt result(forward<L>(l));
    result -= r;
    return result;
```

```
    }

    friend pair<BigInt, BigInt> divmod(const BigInt &a1, const BigInt
&b1)
    {
        ll norm = BASE / (b1.a.back() + 1);
        BigInt a = a1.abs() * norm, b = b1.abs() * norm, q = 0, r = 0;
        q.a.resize(a.a.size());
        for (int i = a.a.size() - 1; i >= 0; i--)
        {
            r *= BASE;
            r += a.a[i];
            ll s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            ll s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            ll d = ((ll)BASE * s1 + s2) / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }
        q.sgn = a1.sgn * b1.sgn;
        r.sgn = a1.sgn;
        q.trim();
        r.trim();
        auto res = make_pair(q, r / norm);
        if (res.second < 0)
            res.second += b1;
        return res;
    }
    BigInt operator/(const BigInt &v) const
    {
        return divmod(*this, v).first;
    }
    BigInt operator%(const BigInt &v) const
    {
```

```
    return divmod(*this, v).second;
}
void operator/=(int v)
{
    if (llabs(v) >= BASE)
    {
        *this /= BigInt(v);
        return;
    }
    if (v < 0)
        sgn = -sgn, v = -v;
    for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i)
    {
        ll cur = a[i] + rem * (ll)BASE;
        a[i] = (int)(cur / v);
        rem = (int)(cur % v);
    }
    trim();
}
BigInt operator/(int v) const
{
    if (llabs(v) >= BASE)
        return *this / BigInt(v);
    BigInt res = *this;
    res /= v;
    return res;
}
void operator/=(const BigInt &v)
{
    *this = *this / v;
}
ll operator%(ll v) const
{
    int m = 0;
    for (int i = a.size() - 1; i >= 0; --i)
```

```
            m = (a[i] + m * (ll)BASE) % v;
        return m * sgn;
    }
    void operator*=(int v)
    {
        if (llabs(v) >= BASE)
        {
            *this *= BigInt(v);
            return;
        }
        if (v < 0)
            sgn = -sgn, v = -v;
        for (int i = 0, carry = 0; i < a.size() || carry; ++i)
        {
            if (i == a.size())
                a.push_back(0);
            ll cur = a[i] * (ll)v + carry;
            carry = (int)(cur / BASE);
            a[i] = (int)(cur % BASE);
        }
        trim();
    }
    BigInt operator*(int v) const
    {
        if (llabs(v) >= BASE)
            return *this * BigInt(v);
        BigInt res = *this;
        res *= v;
        return res;
    }

    static vector<int> convert_base(const vector<int> &a, int
old_digits, int new_digits)
    {
        vector<ll> p(max(old_digits, new_digits) + 1);
    p[0] = 1;
    for (int i = 1; i < (int)p.size(); i++)
        p[i] = p[i - 1] * 10;
    vector<int> res;
    ll cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++)
    {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits)
        {
            res.push_back((ll)(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back())
        res.pop_back();
    return res;
}

void fft(vector<Cplx> &a, bool invert) const
{
    int n = a.size();
    for (int i = 1, j = 0; i < n; ++i)
    {
        int bit = n / 2;
        for (; j >= bit; bit /= 2)
            j -= bit;
        j += bit;
        if (i < j)
            swap(a[i], a[j]);
    }
        for (int len = 2; len <= n; len *= 2)
        {
            ld ang = 2 * PI / len * (invert ? -1 : 1);
            Cplx wlen = polar(1, ang);
            for (int i = 0; i < n; i += len)
            {
                Cplx w(1);
                for (int j = 0; j < len / 2; ++j)
                {
                    Cplx 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 (int i = 0; i < n; ++i)
                a[i] /= n;
    }
    void multiply_fft(const vector<int> &a, const vector<int> &b, vector<int> &res) const
    {
        vector<Cplx> fa(a.begin(), a.end()), fb(b.begin(), b.end());
        int n = 1;
        while (n < max(a.size(), b.size()))
            n *= 2;
        n *= 2;
        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);
    res.resize(n);
    ll carry = 0;
    for (int i = 0; i < n; i++)
    {
        ll t = (ll)(fa[i].real() + 0.5) + carry;
        carry = t / FBASE;
        res[i] = t % FBASE;
    }
}
static inline int rev_incr(int a, int n)
{
    int msk = n / 2, cnt = 0;
    while (a & msk)
    {
        cnt++;
        a <<= 1;
    }
    a &= msk - 1;
    a |= msk;
    while (cnt--)
        a >>= 1;
    return a;
}
static vector<Cplx> FFT(vector<Cplx> v, int dir = 1)
{
    Cplx wm, w, u, t;
    int n = v.size();
    vector<Cplx> V(n);
    for (int k = 0, a = 0; k < n; ++k, a = rev_incr(a, n))
        V[a] = v[k] / ld(dir > 0 ? 1 : n);
    for (int m = 2; m <= n; m <<= 1)
    {
        wm = polar((ld)1, dir * 2 * PI / m);
        for (int k = 0; k < n; k += m)
        {
            w = 1;
            for (int j = 0; j < m / 2; ++j, w *= wm)
            {
                u = V[k + j];
                t = w * V[k + j + m / 2];
                V[k + j] = u + t;
                V[k + j + m / 2] = u - t;
            }
        }
    }
    return V;
}
static void convolution(const vector<int> &a, const vector<int>
&b, vector<int> &c)
{
    int sz = a.size() + b.size() - 1;
    int n = 1 << int(ceil(log2(sz)));
    vector<Cplx> av(n, 0), bv(n, 0), cv;
    for (int i = 0; i < a.size(); i++)
        av[i] = a[i];
    for (int i = 0; i < b.size(); i++)
        bv[i] = b[i];
    cv = FFT(bv);
    bv = FFT(av);
    for (int i = 0; i < n; i++)
        av[i] = bv[i] * cv[i];
    cv = FFT(av, -1);
    c.resize(n);
    ll carry = 0;
    for (int i = 0; i < n; i++)
    {
        ll t = ll(cv[i].real() + 0.5) + carry;
        carry = t / FBASE;
        c[i] = t % FBASE;
    }
}
BigInt mul_simple(const BigInt &v) const
{
    BigInt res;
    res.sgn = sgn * v.sgn;
    res.a.resize(a.size() + v.a.size());
    for (int i = 0; i < a.size(); i++)
        if (a[i])
            for (int j = 0, carry = 0; j < v.a.size() || carry; j++)
            {
                ll cur = res.a[i + j] + (ll)a[i] * (j < v.a.size() ? v.a[j] : 0) + carry;
                carry = (int)(cur / BASE);
                res.a[i + j] = (int)(cur % BASE);
            }
    res.trim();
    return res;
}
BigInt mul_fft(const BigInt &v) const
{
    BigInt res;
    convolution(convert_base(a, DIG, FDIG), convert_base(v.a, DIG, FDIG), res.a);
    res.a = convert_base(res.a, FDIG, DIG);
    res.trim();
    return res;
}
void operator*=(const BigInt &v)
{
    *this = *this * v;
}
BigInt operator*(const BigInt &v) const
{
    if (1LL * a.size() * v.a.size() <= 1000111)
        return mul_simple(v);
    return mul_fft(v);
}

BigInt abs() const
{
    BigInt res = *this;
    res.sgn *= res.sgn;
    return res;
}
void trim()
{
    while (!a.empty() && !a.back())
        a.pop_back();
}
bool zero() const
{
    return a.empty() || (a.size() == 1 && !a[0]);
}
friend BigInt gcd(const BigInt &a, const BigInt &b)
{
    return b.zero() ? a : gcd(b, a % b);
}
};
BigInt power(BigInt a, ll k)
{
BigInt ans = 1;
while (k > 0)
{
    if (k & 1)
        ans *= a;
    a *= a;
    k >>= 1;
}
return ans;
}
```

Vector Operations

```
template <typename T>
pair <T, T> operator +(pair <T, T> a, pair <
T, T> b){
return mp(a.F + b.F, a.S + b.S);
}
template <typename T>
pair <T, T> operator -(pair <T, T> a, pair <
T, T> b){
return mp(a.F - b.F, a.S - b.S);
}
template <typename T>
pair <T, T> operator *(pair <T, T> a, T b){
return mp(a.F * b, a.S * b);
}
template <typename T>
pair <T, T> operator /(pair <T, T> a, T b){
return mp(a.F / b, a.S / b);
}
template <typename T>
T dot(pair <T, T> a, pair <T, T> b){
return a.F * b.F + a.S * b.S;
}
template <typename T>
T cross(pair <T, T> a, pair <T, T> b){
return a.F * b.S - a.S * b.F;
}
template <typename T>
T abs2(pair <T, T> a){
return a.F * a.F + a.S * a.S;
}
```

PRIME NO RELATED:

Sieve optimum:

```
vector<int> smallest_factor;
vector<bool> prime;
vector<int> primes;

void sieve(int maximum) {
  maximum = max(maximum, 2);
  smallest_factor.assign(maximum + 1, 0);
  prime.assign(maximum + 1, true);
  prime[0] = prime[1] = false;
  primes = {2};

  for (int p = 2; p <= maximum; p += 2) {
    prime[p] = p == 2;
    smallest_factor[p] = 2;
  }


  for (int p = 3; p * p <= maximum; p += 2)
    if (prime[p])
      for (int i = p * p; i <= maximum; i += 2 * p)
        if (prime[i]) {
          prime[i] = false;
          smallest_factor[i] = p;
        }


  for (int p = 3; p <= maximum; p += 2)
    if (prime[p]) {
      smallest_factor[p] = p;
      primes.push_back(p);
    }
}
************OR******:
vector<bool> Primes(N,1);
vector<ll>primenos;
void SieveOfEratosthenes(ll n)
{
  Primes[0]=0;
  Primes[1]=0;
  for (ll i=2;i*i<=n;i++) {
  if(Primes[i]==1){
  for(ll j=i*i;j<=n;j+=i)
    Primes[j]=0;
    }
  }
  for(ll i=1;i<n;i++){
    if(Primes[i]){
      primenos.push_back(i);
    }
  }
}
```

NUMBER OF DIVISORS:

```
ll NOD(ll n){
  ll res=1;
  for(int i=0;i<SZ(primes) && primes[i]*primes[i]<=n;i++){
    ll cnt=0;
    ll it=primes[i];
    if(n%it==0){
      while(n%it==0) n/=it,cnt++;
    }
    res*=(cnt+1);
    if(n==1||it>n) break;
  }
  if(n!=1) res*=(1+1);
  return res;
}
```

Lowest prime & Highest primes:

```
vector<bool> isPrime(N,1);
vector<int> lp(N,0),hp(N,0);
void primeSieve(){
  isPrime[0]=isPrime[1]=false;
  for(int i=2;i<N;i++){
    if(isPrime[i]==true){
      lp[i]=hp[i]=i;
      for(int j=2*i;j<N;j+=i) {
        isPrime[j]=false;
        hp[j]=i;
        if(lp[j]==0){
          lp[j]=i;
        }
      }
    }
  }
}
```

MILLER ROBIN Primality Test:

```
/* Miller-Rabin primality test, iteration signifies the accuracy of
the test */
bool Miller(ll p,int iteration){
  if(p<2){
    return false;
  }
  if(p!=2 && p%2==0){
    return false;
  }
  ll s=p-1;
  while(s%2==0){
    s/=2;
  }
  for(int i=0;i<iteration;i++){
    ll a=rand()%(p-1)+1,temp=s;
    ll mod=modulo(a,temp,p);
    while(temp!=p-1 && mod!=1 && mod!=p-1){
      mod=mulmod(mod,mod,p);
      temp *= 2;
    }
    if(mod!=p-1 && temp%2==0){
    return false;
    }
  }
  return true;
}
```

PRIME FACTOCTORIZATIONS:

```
vl primeFactor(ll n){
  vl res;
  while(n>1){
    ll a=smallest_factor[n];
```

```
        res.pb(a);
        while(n%a==0) n/=a;
    }
    return res;
}
```

Divisor Store:

```
vl divisors[N];

void divisor_store(){
    for(int i=2;i<N;i++){
        for(int j=i;j<N;j+=i) {
            divisors[j].push_back(i);
        }
    }
}
```

EULER TOTIENT:

```
const int MAX = 100001;
bool isPrime[MAX+1];

// Stores prime numbers upto MAX - 1 values
vector<ll> p;

// Finds prime numbers upto MAX-1 and
// stores them in vector p
void sieve(){
    for (ll i = 2; i<= MAX; i++){
        // if prime[i] is not marked before
        if (isPrime[i] == 0){
            // fill vector for every newly
            // encountered prime
            p.push_back(i);
            // run this loop till square root of MAX,
            // mark the index i * j as not prime
            for (ll j = 2; i * j<= MAX; j++)
                isPrime[i * j]= 1;
        }
    }
}

// function to find totient of n
ll phi(ll n){
    ll res = n;
    // this loop runs sqrt(n / ln(n)) times
    for (ll i=0; p[i]*p[i] <= n; i++){
        if (n % p[i]== 0){
            // subtract multiples of p[i] from r
            res -= (res / p[i]);
            // Remove all occurrences of p[i] in n
            while (n % p[i]== 0) n /= p[i];
        }
    }
    // when n has prime factor greater
    // than sqrt(n)
    if (n > 1) res -= (res / n);
    return res;
}

// Computes and prints totient of all numbers
// smaller than or equal to n.

#define sz 10000000
ll prime[sz + 9], etf[sz + 9];

void computeTotient(){
    etf[1] = 1;
    for(ll i = 2; i <= sz; i++){
        if(!prime[i]){
            etf[i] = i - 1;
            for(ll j = 1; j * i <= sz; j++)
```

```
                if(!prime[j*i])prime[j*i] = i;
        }
        else{
            etf[i] = etf[prime[i]] * etf[ i/prime[i] ];
            ll g = 1;
            if(i % (prime[i]*prime[i]) == 0) g = prime[i];
            etf[i] *= g;
            etf[i] /= etf[g];
        }
    }
}
```

EULER TOTIENT (1-n) 2:

```
int phi[N];

void computePhi(){
    for(int i=2; i<=N; i++)
        phi[i] = i;
    for(int i=2; i<=N; i++)
        if(phi[i]==i)
            for(int j=i; j<=N; j+=i)
                phi[j]-=phi[j]/i;
}
```

Extended Euclid:

```
// using T = __int128;
// ax + by = __gcd(a, b)
// returns __gcd(a, b)
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
    ll xx = y = 0;
    ll yy = x = 1;
    while (b) {
        ll q = a / b;
        ll t = b; b = a % b; a = t;
        t = xx; xx = x - q * xx; x = t;
        t = yy; yy = y - q * yy; y = t;
    }
    return a;
}
```

Extended gcd:

```
pii extendedEuclid(ll a, ll b)   // returns x, y for ax + by = gcd(a,b)
{
    if(b == 0) return pii(1, 0);
    else
    {
        pii d = extendedEuclid(b, a % b);
        return pii(d.ss, d.ff - d.ss * (a / b));
    }
}


ll modularInverse(ll a, ll m)
{
    pii ret = extendedEuclid(a, m);
    return ((ret.ff % m) + m) % m;
}
```

GCD:

```
ll gcd(ll a,ll b){
    if(a==0) return b;
    if(b==0) return a;
    while(b){
        ll remainder=a%b;
        a=b;
        b=remainder;
    }
    return a;
}
```

POWER:

```
int power(int a, int n){
    int res = 1;
    while(n){if(n%2){res*=a;n--;}else{a*=a;n/=2;}}
    return res;
}
```

JOSEPHUS:

```
// n = total person
// will kill every kth person, if k = 2, 2,4,6,...
// returns the mth killed person
ll josephus(ll n, ll k, ll m) {
    m = n - m;
    if (k <= 1)return n - m;
    ll i = m;
    while (i < n) {
        ll r = (i - m + k - 2) / (k - 1);
        if ((i + r) > n) r = n - i;
        else if (!r) r = 1;
        i += r;
        m = (m + (r * k)) % i;
    } return m + 1;
}
```

MIN_25_Sieve:

```
// credit: min_25
// takes 0.5s for n = 1e9
vector<int> sieve(const int N, const int Q = 17, const int L = 1 << 15)
{
    static const int rs[] = {1, 7, 11, 13, 17, 19, 23, 29};
    struct P
    {
        P(int p) : p(p) {}
        int p;
        int pos[8];
    };
    auto approx_prime_count = [](const int N) -> int
    {
        return N > 60184 ? N / (log(N) - 1.1)
                         : max(1., N / (log(N) - 1.11)) + 1;
    };

    const int v = sqrt(N), vv = sqrt(v);
    vector<bool> isp(v + 1, true);
    for (int i = 2; i <= vv; ++i)
        if (isp[i])
        {
            for (int j = i * i; j <= v; j += i)
                isp[j] = false;
        }

    const int rsize = approx_prime_count(N + 30);
    vector<int> primes = {2, 3, 5};
    int psize = 3;
    primes.resize(rsize);

    vector<P> sprimes;
    size_t pbeg = 0;
    int prod = 1;
    for (int p = 7; p <= v; ++p)
    {
        if (!isp[p])
            continue;
        if (p <= Q)
            prod *= p, ++pbeg, primes[psize++] = p;
        auto pp = P(p);
        for (int t = 0; t < 8; ++t)
        {
            int j = (p <= Q) ? p : p * p;
            while (j % 30 != rs[t])
                j += p << 1;
            pp.pos[t] = j / 30;
        }
        sprimes.push_back(pp);
    }
    vector<unsigned char> pre(prod, 0xFF);
    for (size_t pi = 0; pi < pbeg; ++pi)
    {
        auto pp = sprimes[pi];
        const int p = pp.p;
        for (int t = 0; t < 8; ++t)
        {
            const unsigned char m = ~(1 << t);
            for (int i = pp.pos[t]; i < prod; i += p)
                pre[i] &= m;
        }
    }

    const int block_size = (L + prod - 1) / prod * prod;
    vector<unsigned char> block(block_size);
    unsigned char *pblock = block.data();
    const int M = (N + 29) / 30;

    for (int beg = 0; beg < M; beg += block_size, pblock -= block_size)
    {
        int end = min(M, beg + block_size);
        for (int i = beg; i < end; i += prod)
        {
            copy(pre.begin(), pre.end(), pblock + i);
        }
        if (beg == 0)
            pblock[0] &= 0xFE;
        for (size_t pi = pbeg; pi < sprimes.size(); ++pi)
        {
            auto &pp = sprimes[pi];
            const int p = pp.p;
            for (int t = 0; t < 8; ++t)
            {
                int i = pp.pos[t];
                const unsigned char m = ~(1 << t);
                for (; i < end; i += p)
                    pblock[i] &= m;
                pp.pos[t] = i;
            }
        }
        for (int i = beg; i < end; ++i)
        {
            for (int m = pblock[i]; m > 0; m &= m - 1)
            {
                primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
            }
        }
    }
    assert(psize <= rsize);
    while (psize > 0 && primes[psize - 1] > N)
        --psize;
    primes.resize(psize);
    return primes;
}
```

FLOOR sum of n/1+n/2+…:

```
// formula: floor sum upto n=2*floor sum upto k - k^2[k=sqrt(n)]
ll floorSum(int n){
    ll sum = 0;
    ll k = sqrt(n);
    // Summation of floor(n / i)
    for (int i = 1; i <= k; i++) {
        sum += Floor(n,i);
    }
    // From the formula
    deb2(sum,k);
    sum *= 2;
    sum -= BigMod<ll>(k,2,LLONG_MAX);
    return sum;
}
```

MODULAR ARITHMATICS

POWER MOD:

```
ll power(ll a,ll b,ll mod)
{   int res = 1;
    a=a%mod;
    if (a==0) return 0;
    while (b>0)
    {
        if (b&1) res=(res*a)%mod;
        b /=2;
        a=(a*a)%mod;
    }
    return res;
}
```

ModINT:

```
const int mod = 998244353;
template <const int32_t MOD>
struct modint {
   int32_t value;
   modint() = default;
   modint(int32_t value_) : value(value_) {}
   inline modint<MOD> operator + (modint<MOD> other) const { int32_t c = this->value + other.value; return modint<MOD>(c >= MOD ? c - MOD : c); }
   inline modint<MOD> operator - (modint<MOD> other) const { int32_t c = this->value - other.value; return modint<MOD>(c <    0 ? c + MOD : c); }
   inline modint<MOD> operator * (modint<MOD> other) const { int32_t c = (int64_t)this->value * other.value % MOD; return modint<MOD>(c < 0 ? c + MOD : c); }
   inline modint<MOD> & operator += (modint<MOD> other) { this->value += other.value; if (this->value >= MOD) this->value -= MOD; return *this; }
   inline modint<MOD> & operator -= (modint<MOD> other) { this->value -= other.value; if (this->value <    0) this->value += MOD; return *this; }
   inline modint<MOD> & operator *= (modint<MOD> other) { this->value = (int64_t)this->value * other.value % MOD; if (this->value < 0) this->value += MOD; return *this; }
   inline modint<MOD> operator - () const { return modint<MOD>(this->value ? MOD - this->value : 0); }
   modint<MOD> pow(uint64_t k) const { modint<MOD> x = *this, y = 1; for (; k; k >>= 1) { if (k & 1) y *= x; x *= x; } return y; }
   modint<MOD> inv() const { return pow(MOD - 2); }  // MOD must be a prime
   inline modint<MOD> operator /  (modint<MOD> other) const { return *this *  other.inv(); }
   inline modint<MOD> operator /= (modint<MOD> other)      { return *this *= other.inv(); }
   inline bool operator == (modint<MOD> other) const { return value == other.value; }
   inline bool operator != (modint<MOD> other) const { return value != other.value; }
   inline bool operator < (modint<MOD> other) const { return value < other.value; }
   inline bool operator > (modint<MOD> other) const { return value > other.value; }
};
template <int32_t MOD> modint<MOD> operator * (int64_t value, modint<MOD> n) { return modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD> operator * (int32_t value, modint<MOD> n) { return modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream & operator >> (istream & in, modint<MOD> &n) { return in >> n.value; }
template <int32_t MOD> ostream & operator << (ostream & out, modint<MOD> n) { return out << n.value; }
```

NCR MOD:

```
ll FM[N];
int is_initialized = 0;
ll factorialMod(ll n, ll x){
   if (!is_initialized){
      FM[0] = 1 % x;
      for (int i = 1; i < N; i++)
```

```
            FM[i] = (FM[i - 1] * i) % x;
        is_initialized = 1;
    }
    return FM[n];
}


ll powerMod(ll x, ll y, ll p){
    ll res = 1 % p;
    x = x % p;
    while (y > 0){
        if (y & 1) res = (res * x) % p;
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}


ll inverseMod(ll a, ll x){
    return powerMod(a, x - 2, x);
}


ll nCrMod(ll n, ll r, ll x){
    if (r == 0) return 1;
    if (r > n) return 0;
    ll res = factorialMod(n, x);
    ll fr = factorialMod(r, x);
    ll zr = factorialMod(n - r, x);
    res = (res * inverseMod((fr * zr) % x, x)) % x;
    return res;
}
```

Mathematical Expression:

```
bool isSpace(char c) {
    return (c==' ');
}
```

```
bool is_op(char c) {
    return c=='+'||c=='-'||c=='*'||c=='/';
}


bool is_unary(char c) {
    return c=='+'||c=='-';
}


ll priority (char op) {
    if(op<0) return 4; // unary operator
    if(op=='+' || op=='-') return 1;
    if(op=='*') return 2;
    if(op=='/') return 3;
    return -1;
}


void process_op(stack<ll>& st, char op) {
    if(op<0) {
        ll l=st.top();
        st.pop();
        switch(-op){
            case '+': st.push(l); break;
            case '-': st.push(-l); break;
        }
    }else {
        ll r=st.top();
        st.pop();
        ll l=st.top();
        st.pop();
        switch (op) {
            case '+': st.push(l+r); break;
            case '-': st.push(l-r); break;
            case '*': st.push(l*r); break;
            case '/': st.push(l/r); break;
        }
    }
}

ll evaluate(string& s) {
    stack<ll> st;
    stack<char> op;
    bool has_unary = true;
    for (int i=0;i<SZ(s);i++) {
        if (isSpace(s[i])) continue;
        if (s[i]=='(') {
            op.push('(');
            has_unary=true;
        }else if(s[i]==')') {
            while (op.top()!='(') {
                process_op(st,op.top());
                op.pop();
            }
            op.pop();
            has_unary=false;
        }else if(is_op(s[i])) {
            char cur_op=s[i];
            if (has_unary && is_unary(cur_op))
                cur_op=-cur_op;
            while (!op.empty() && ((cur_op>=0 && priority(op.top())>=priority(cur_op)) ||(cur_op<0 && priority(op.top())>priority(cur_op)))) {
                process_op(st,op.top());
                op.pop();
            }
            op.push(cur_op);
            has_unary=true;
        }else{
            ll number=0;
            while(i<SZ(s) && isalnum(s[i])) number = number * 10 + s[i++] - '0';
            --i;
            st.push(number);
            has_unary=false;
        }
    }

    while(!op.empty()) {
        process_op(st,op.top());
        op.pop();
    }
    return st.top();
}
```

Vector Operations

```
template <typename T>
pair <T, T> operator +(pair <T, T> a, pair < T, T> b){
return mp(a.F + b.F, a.S + b.S);
}
template <typename T>
pair <T, T> operator -(pair <T, T> a, pair < T, T> b){
return mp(a.F - b.F, a.S - b.S);
}
template <typename T>
pair <T, T> operator *(pair <T, T> a, T b){
return mp(a.F * b, a.S * b);
}
template <typename T>
pair <T, T> operator /(pair <T, T> a, T b){
return mp(a.F / b, a.S / b);
}
template <typename T>
T dot(pair <T, T> a, pair <T, T> b){
return a.F * b.F + a.S * b.S;
}
template <typename T>
T cross(pair <T, T> a, pair <T, T> b){
return a.F * b.S - a.S * b.F;
}
template <typename T>
T abs2(pair <T, T> a){
return a.F * a.F + a.S * a.S;
}
```

Convex Hull

```
template <typename T>
pair <T, T> operator -(pair <T, T> a, pair <
T, T> b){
return mp(a.F - b.F, a.S - b.S);
}
template <typename T>
T cross(pair <T, T> a, pair <T, T> b){
return a.F * b.S - a.S * b.F;
}
template <typename T>
vector <pair <T, T>> getConvexHull(vector <
pair <T, T>>& pnts){
int n = pnts.size();
lsort(pnts);
vector <pair <T, T>> hull;
hull.reserve(n);
for(int i = 0; i < 2; i++){
int t = hull.size();
for(pair <T, T> pnt : pnts){
while(hull.size() - t >= 2 &&
cross(hull.back() - hull[hull.size()
- 2], pnt - hull[hull.size() - 2]) <=
0){
hull.pop_back ();
}
hull.pb(pnt);
}
hull.pop_back ();
reverse(iter(pnts));
}
return hull;
}
```

Dynamic Convex Hull

```
struct Line{
ll a, b, l = MIN , r = MAX;
Line(ll a, ll b): a(a), b(b) {}
ll operator ()(ll x) const{
return a * x + b;
}
bool operator <(Line b) const{
return a < b.a;
}
bool operator <(ll b) const{
return r < b;
}
};
ll iceil(ll a, ll b){
if(b < 0) a *= -1, b *= -1;
if(a > 0) return (a + b - 1) / b;
else return a / b;
}
ll intersect(Line a, Line b){
return iceil(a.b - b.b, b.a - a.a);
}
struct DynamicConvexHull{
multiset <Line , less <>> ch;
void add(Line ln){
auto it = ch.lower_bound(ln);
while(it != ch.end()){
Line tl = *it;
if(tl(tl.r) <= ln(tl.r)){
it = ch.erase(it);
}
else break;
}
auto it2 = ch.lower_bound(ln);
while(it2 != ch.begin ()){
Line tl = *prev(it2);
if(tl(tl.l) <= ln(tl.l)){
it2 = ch.erase(prev(it2));
}
else break;
}
it = ch.lower_bound(ln);
if(it != ch.end()){
Line tl = *it;
if(tl(tl.l) >= ln(tl.l)) ln.r = tl
.l - 1;
else{
ll pos = intersect(ln , tl);
tl.l = pos;
ln.r = pos - 1;
ch.erase(it);
ch.insert(tl);
}
}
it2 = ch.lower_bound(ln);
if(it2 != ch.begin ()){
Line tl = *prev(it2);
if(tl(tl.r) >= ln(tl.r)) ln.l = tl
.r + 1;
else{
ll pos = intersect(tl , ln);
tl.r = pos - 1;
ln.l = pos;
ch.erase(prev(it2));
ch.insert(tl);
}
}
if(ln.l <= ln.r) ch.insert(ln);
}
ll query(ll pos){
```

```
auto it = ch.lower_bound(pos);
if(it == ch.end()) return 0;
return (*it)(pos);
```

```
}
};
```

****Combinatorics***

```
struct combi{
 int n; vector<mint> facts, finvs, invs;
 combi(int _n): n(_n), facts(_n), finvs(_n), invs(_n){
  facts[0] = finvs[0] = 1;
  invs[1] = 1;
  for (int i = 2; i < n; i++) invs[i] =  invs[mod % i] * (-mod / i);
  for(int i = 1; i < n; i++){
   facts[i] = facts[i - 1] * i;
   finvs[i] = finvs[i - 1] * invs[i];
  }
 }
 inline mint fact(int n) { return facts[n]; }
 inline mint finv(int n) { return finvs[n]; }
 inline mint inv(int n) { return invs[n]; }
 inline mint ncr(int n, int k) { return n < k or k < 0 ? 0 : facts[n] * finvs[k] * finvs[n-k]; }
};
combi C(N);
```

****GEOMETRY****

2D geometry:

```
const double inf = 1e100;
const double eps = 1e-9;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
  double x, y;
  PT() { x = 0, y = 0; }
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &a) const { return PT(x + a.x, y + a.y); }
  PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
  PT operator * (const double a) const { return PT(x * a, y * a); }
  friend PT operator * (const double &a, const PT &b) { return PT(a * b.x, a * b.y); }
  PT operator / (const double a) const { return PT(x / a, y / a); }
  bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y - y) == 0; }
  bool operator != (PT a) const { return !(*this == a); }
  bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y : x < a.x; }
  bool operator > (PT a) const { return sign(a.x - x) == 0 ? y > a.y : x > a.x; }
  double norm() { return sqrt(x * x + y * y); }
  double norm2() { return x * x + y * y; }
  PT perp() { return PT(-y, x); }
  double arg() { return atan2(y, x); }
  PT truncate(double r) { // returns a vector with norm r and having same direction
    double k = norm();
    if (!sign(k)) return *this;
    r /= k;
    return PT(x * r, y * r);
  }
};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline double cross2(PT a, PT b, PT c) { return cross(b - a, c - a); }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y * sin(t), a.x * sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y * sin(t), -a.x * sin(t) + a.y * cos(t)); }

double SQ(double x) { return x * x; }

double rad_to_deg(double r) { return (r * 180.0 / PI); }

double deg_to_rad(double d) { return (d * PI / 180.0); }

double get_angle(PT a, PT b) {

    double costheta = dot(a, b) / a.norm() / b.norm();

    return acos(max((double)-1.0, min((double)1.0, costheta)));

}

bool is_point_in_angle(PT b, PT a, PT c, PT p) { // does point p lie in angle <bac

    assert(orientation(a, b, c) != 0);

    if (orientation(a, c, b) < 0) swap(b, c);

    return orientation(a, c, p) >= 0 && orientation(a, b, p) <= 0;

}

bool half(PT p) {

    return p.y > 0.0 || (p.y == 0.0 && p.x < 0.0);

}

void polar_sort(vector<PT> &v) { // sort points in counterclockwise

    sort(v.begin(), v.end(), [](PT a,PT b) {

        return make_tuple(half(a), 0.0, a.norm2()) < make_tuple(half(b), cross(a, b), b.norm2());

    });

}

struct line {

    PT a, b; // goes through points a and b

    PT v; double c;  //line form: direction vec [cross] (x, y) = c

    line() {}

    //direction vector v and offset c

    line(PT v, double c) : v(v), c(c) {

        auto p = get_points();

        a = p.first; b = p.second;

    }

    // equation ax + by + c = 0

    line(double _a, double _b, double _c) : v({_b, -_a}), c(-_c) {

        auto p = get_points();

        a = p.first; b = p.second;

    }

    // goes through points p and q

    line(PT p, PT q) : v(q - p), c(cross(v, p)), a(p), b(q) {}

        pair<PT, PT> get_points() { //extract any two points from this line

        PT p, q; double a = -v.y, b = v.x; // ax + by = c

        if (sign(a) == 0) {

            p = PT(0, c / b);

            q = PT(1, c / b);

        }

        else if (sign(b) == 0) {

            p = PT(c / a, 0);

            q = PT(c / a, 1);

        }

        else {

            p = PT(0, c / b);

            q = PT(1, (c - a) / b);

        }

        return {p, q};

    }

    //ax + by + c = 0

    array<double, 3> get_abc() {

        double a = -v.y, b = v.x;

        return {a, b, c};

    }

    // 1 if on the left, -1 if on the right, 0 if on the line

    int side(PT p) { return sign(cross(v, p) - c); }

    // line that is perpendicular to this and goes through point p

    line perpendicular_through(PT p) { return {p, p + perp(v)}; }

    // translate the line by vector t i.e. shifting it by vector t

    line translate(PT t) { return {v, c + cross(v, t)}; }

    // compare two points by their orthogonal projection on this line

    // a projection point comes before another if it comes first according to vector v
    bool cmp_by_projection(PT p, PT q) { return dot(v, p) < dot(v, q); }
    line shift_left(double d) {
        PT z = v.perp().truncate(d);
        return line(a + z, b + z);
    }
};
// find a point from a through b with distance d
PT point_along_line(PT a, PT b, double d) {
    return a + (((b - a) / (b - a).norm()) * d);
}
// projection point c onto line through a and b  assuming a != b
PT project_from_point_to_line(PT a, PT b, PT c) {
    return a + (b - a) * dot(c - a, b - a) / (b - a).norm2();
}
// reflection point c onto line through a and b  assuming a != b
PT reflection_from_point_to_line(PT a, PT b, PT c) {
    PT p = project_from_point_to_line(a,b,c);
    return point_along_line(c, p, 2.0 * dist(c, p));
}
// minimum distance from point c to line through a and b
double dist_from_point_to_line(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) / (b - a).norm());
}
// returns true if  point p is on line segment ab
bool is_point_on_seg(PT a, PT b, PT p) {
    if (fabs(cross(p - b, a - b)) < eps) {
        if (p.x < min(a.x, b.x) || p.x > max(a.x, b.x)) return false;
        if (p.y < min(a.y, b.y) || p.y > max(a.y, b.y)) return false;
        return true;
    }
    return false;
}
// minimum distance point from point c to segment ab that lies on segment ab
PT project_from_point_to_seg(PT a, PT b, PT c) {
    double r = dist2(a, b);
    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;
}
// minimum distance from point c to segment ab
double dist_from_point_to_seg(PT a, PT b, PT c) {
    return dist(c, project_from_point_to_seg(a, b, c));
}
// 0 if not parallel, 1 if parallel, 2 if collinear
int is_parallel(PT a, PT b, PT c, PT d) {
    double k = fabs(cross(b - a, d - c));
    if (k < eps){
        if (fabs(cross(a - b, a - c)) < eps && fabs(cross(c - d, c - a)) < eps) return 2;
        else return 1;
    }
    else return 0;
}
// check if two lines are same
bool are_lines_same(PT a, PT b, PT c, PT d) {
    if (fabs(cross(a - c, c - d)) < eps && fabs(cross(b - c, c - d)) < eps) return true;
    return false;
}
// bisector vector of <abc
PT angle_bisector(PT &a, PT &b, PT &c){
    PT p = a - b, q = c - b;
    return p + q * sqrt(dot(p, p) / dot(q, q));
}
// 1 if point is ccw to the line, 2 if point is cw to the line, 3 if point is on the line
int point_line_relation(PT a, PT b, PT p) {
    int c = sign(cross(p - a, b - a));
    if (c < 0) return 1;
    if (c > 0) return 2;
    return 3;
}
// intersection point between ab and cd assuming unique intersection exists
bool line_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double a1 = a.y - b.y, b1 = b.x - a.x, c1 = cross(a, b);
    double a2 = c.y - d.y, b2 = d.x - c.x, c2 = cross(c, d);
    double det = a1 * b2 - a2 * b1;
    if (det == 0) return 0;
    ans = PT((b1 * c2 - b2 * c1) / det, (c1 * a2 - a1 * c2) / det);
    return 1;
}
// intersection point between segment ab and segment cd assuming unique intersection exists
bool seg_seg_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    double oa = cross2(c, d, a), ob = cross2(c, d, b);
    double oc = cross2(a, b, c), od = cross2(a, b, d);
    if (oa * ob < 0 && oc * od < 0){
        ans = (a * ob - b * oa) / (ob - oa);
        return 1;
    }
    else return 0;
}
// intersection point between segment ab and segment cd assuming unique intersection may not exists
// se.size()==0 means no intersection
// se.size()==1 means one intersection
// se.size()==2 means range intersection
set<PT> seg_seg_intersection_inside(PT a,  PT b,  PT c,  PT d) {
    PT ans;
    if (seg_seg_intersection(a, b, c, d, ans)) return {ans};
    set<PT> se;
    if (is_point_on_seg(c, d, a)) se.insert(a);
    if (is_point_on_seg(c, d, b)) se.insert(b);
    if (is_point_on_seg(a, b, c)) se.insert(c);
    if (is_point_on_seg(a, b, d)) se.insert(d);
    return se;
}
// intersection  between segment ab and line cd
// 0 if do not intersect, 1 if proper intersect, 2 if segment intersect
int seg_line_relation(PT a, PT b, PT c, PT d) {
    double p = cross2(c, d, a);
    double q = cross2(c, d, b);
    if (sign(p) == 0 && sign(q) == 0) return 2;
    else if (p * q < 0) return 1;
    else return 0;
}
// intersection between segament ab and line cd assuming unique intersection exists
bool seg_line_intersection(PT a, PT b, PT c, PT d, PT &ans) {
    bool k = seg_line_relation(a, b, c, d);
    assert(k != 2);
    if (k) line_line_intersection(a, b, c, d, ans);
    return k;
}
// minimum distance from segment ab to segment cd
double dist_from_seg_to_seg(PT a, PT b, PT c, PT d) {
    PT dummy;
    if (seg_seg_intersection(a, b, c, d, dummy)) return 0.0;
    else return min({dist_from_point_to_seg(a, b, c), dist_from_point_to_seg(a, b, d),
        dist_from_point_to_seg(c, d, a), dist_from_point_to_seg(c, d, b)});
}
// minimum distance from point c to ray (starting point a and direction vector b)
double dist_from_point_to_ray(PT a, PT b, PT c) {
    b = a + b;
    double r = dot(c - a, b - a);
    if (r < 0.0) return dist(c, a);
    return dist_from_point_to_line(a, b, c);
```

```
}
// starting point as and direction vector ad
bool ray_ray_intersection(PT as, PT ad, PT bs, PT bd) {
    double dx = bs.x - as.x, dy = bs.y - as.y;
    double det = bd.x * ad.y - bd.y * ad.x;
    if (fabs(det) < eps) return 0;
    double u = (dy * bd.x - dx * bd.y) / det;
    double v = (dy * ad.x - dx * ad.y) / det;
    if (sign(u) >= 0 && sign(v) >= 0) return 1;
    else return 0;
}
double ray_ray_distance(PT as, PT ad, PT bs, PT bd) {
    if (ray_ray_intersection(as, ad, bs, bd)) return 0.0;
    double ans = dist_from_point_to_ray(as, ad, bs);
    ans = min(ans, dist_from_point_to_ray(bs, bd, as));
    return ans;
}
struct circle {
    PT p; double r;
    circle() {}
    circle(PT _p, double _r): p(_p), r(_r) {};
    // center (x, y) and radius r
    circle(double x, double y, double _r): p(PT(x, y)), r(_r) {};
    // circumcircle of a triangle
    // the three points must be unique
    circle(PT a, PT b, PT c) {
        b = (a + b) * 0.5;
        c = (a + c) * 0.5;
        line_line_intersection(b, b + rotatecw90(a - b), c, c + rotatecw90(a - c), p);
        r = dist(a, p);
    }
    // inscribed circle of a triangle
    circle(PT a, PT b, PT c, bool t) {
        line u, v;
        double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
        u.a = a;
        u.b = u.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        v.a = b;
        m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
        v.b = v.a + (PT(cos((n + m)/2.0), sin((n + m)/2.0)));
        line_line_intersection(u.a, u.b, v.a, v.b, p);
        r = dist_from_point_to_seg(a, b, p);
    }
    bool operator == (circle v) { return p == v.p && sign(r - v.r) == 0; }
    double area() { return PI * r * r; }
    double circumference() { return 2.0 * PI * r; }
};
//0 if outside, 1 if on circumference, 2 if inside circle
int circle_point_relation(PT p, double r, PT b) {
    double d = dist(p, b);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
// 0 if outside, 1 if on circumference, 2 if inside circle
int circle_line_relation(PT p, double r, PT a, PT b) {
    double d = dist_from_point_to_line(a, b, p);
    if (sign(d - r) < 0) return 2;
    if (sign(d - r) == 0) return 1;
    return 0;
}
//compute intersection of line through points a and b with
//circle centered at c with radius r > 0
vector<PT> circle_line_intersection(PT c, double r, PT a, PT b) {
    vector<PT> ret;
    b = b - a; a = a - c;
    double A = dot(b, b), B = dot(a, b);
    double C = dot(a, a) - r * r, D = B * B - A * C;
    if (D < -eps) return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + eps)) / A);
    if (D > eps) ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}
//5 - outside and do not intersect
//4 - intersect outside in one point
//3 - intersect in 2 points
//2 - intersect inside in one point
//1 - inside and do not intersect
int circle_circle_relation(PT a, double r, PT b, double R) {
    double d = dist(a, b);
    if (sign(d - r - R) > 0)  return 5;
    if (sign(d - r - R) == 0) return 4;
    double l = fabs(r - R);
    if (sign(d - r - R) < 0 && sign(d - l) > 0) return 3;
    if (sign(d - l) == 0) return 2;
    if (sign(d - l) < 0) return 1;
    assert(0); return -1;
}
vector<PT> circle_circle_intersection(PT a, double r, PT b, double R) {
    if (a == b && sign(r - R) == 0) return {PT(1e18, 1e18)};
    vector<PT> ret;
    double d = sqrt(dist2(a,  b));
    if (d > r + R || d + min(r,  R) < max(r,  R)) return ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x  +  rotateccw90(v) * y);
    if (y > 0) ret.push_back(a + v * x - rotateccw90(v) * y);
    return ret;
}
// returns two circle c1, c2 through points a, b and of radius r
// 0 if there is no such circle, 1 if one circle, 2 if two circle
int get_circle(PT a, PT b, double r, circle &c1, circle &c2) {
    vector<PT> v = circle_circle_intersection(a, r, b, r);
    int t = v.size();
    if (!t) return 0;
    c1.p = v[0], c1.r = r;
    if (t == 2) c2.p = v[1], c2.r = r;
    return t;
}
// returns two circle c1, c2 which is tangent to line u,  goes through
// point q and has radius r1; 0 for no circle, 1 if c1 = c2 , 2 if c1 != c2
int get_circle(line u, PT q, double r1, circle &c1, circle &c2) {
    double d = dist_from_point_to_line(u.a, u.b, q);
    if (sign(d - r1 * 2.0) > 0) return 0;
    if (sign(d) == 0) {
        cout << u.v.x << ' ' << u.v.y << '\n';
        c1.p = q + rotateccw90(u.v).truncate(r1);
        c2.p = q + rotatecw90(u.v).truncate(r1);
        c1.r = c2.r = r1;
        return 2;
    }
    line u1 = line(u.a + rotateccw90(u.v).truncate(r1), u.b + rotateccw90(u.v).truncate(r1));
    line u2 = line(u.a + rotatecw90(u.v).truncate(r1), u.b + rotatecw90(u.v).truncate(r1));
    circle cc = circle(q, r1);
    PT p1, p2; vector<PT> v;
    v = circle_line_intersection(q, r1, u1.a, u1.b);
    if (!v.size()) v = circle_line_intersection(q, r1, u2.a, u2.b);
    v.push_back(v[0]);
    p1 = v[0], p2 = v[1];
    c1 = circle(p1, r1);
    if (p1 == p2) {
        c2 = c1;
        return 1;
    }
    c2 = circle(p2, r1);
    return 2;
}
// returns area of intersection between two circles
double circle_circle_area(PT a, double r1, PT b, double r2) {
    double d = (a - b).norm();
    if(r1 + r2 < d + eps) return 0;
    if(r1 + d < r2 + eps) return PI * r1 * r1;
    if(r2 + d < r1 + eps) return PI * r2 * r2;
    double theta_1 = acos((r1 * r1 + d * d - r2 * r2) / (2 * r1 * d)),
    theta_2 = acos((r2 * r2 + d * d - r1 * r1)/(2 * r2 * d));
    return r1 * r1 * (theta_1 - sin(2 * theta_1)/2.) + r2 * r2 * (theta_2 - sin(2 * theta_2)/2.);
}
// tangent lines from point q to the circle
int tangent_lines_from_point(PT p, double r, PT q, line &u, line &v) {
    int x = sign(dist2(p, q) - r * r);
    if (x < 0) return 0; // point in cricle
    if (x == 0) { // point on circle
        u = line(q, q + rotateccw90(q - p));
        v = u;
        return 1;
    }
    double d = dist(p, q);
    double l = r * r / d;
    double h = sqrt(r * r - l * l);
    u = line(q, p + ((q - p).truncate(l) + (rotateccw90(q - p).truncate(h))));
    v = line(q, p + ((q - p).truncate(l) + (rotatecw90(q - p).truncate(h))));
    return 2;
}
// returns outer tangents line of two circles
// if inner == 1 it returns inner tangent lines
int tangents_lines_from_circle(PT c1, double r1, PT c2, double r2, bool inner, line &u, line &v) {
    if (inner) r2 = -r2;
    PT d = c2 - c1;
    double dr = r1 - r2, d2 = d.norm(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) {
        assert(h2 != 0);
        return 0;
    }
    vector<pair<PT, PT>>out;
    for (int tmp: {- 1, 1}) {
        PT v = (d * dr + rotateccw90(d) * sqrt(h2) * tmp) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    u = line(out[0].first, out[0].second);
    if (out.size() == 2) v = line(out[1].first, out[1].second);
    return 1 + (h2 > 0);
}
//O(n^2 log n)
struct CircleUnion {
    int n;
    double x[2020], y[2020], r[2020];
    int covered[2020];
    vector<pair<double, double> > seg, cover;
    double arc, pol;
    inline int sign(double x) {return x < -eps ? -1 : x > eps;}
    inline int sign(double x, double y) {return sign(x - y);}
    inline double SQ(const double x) {return x * x;}
    inline double dist(double x1, double y1, double x2, double y2) {return sqrt(SQ(x1 - x2) + SQ(y1 - y2));}
    inline double angle(double A, double B, double C) {
        double val = (SQ(A) + SQ(B) - SQ(C)) / (2 * A * B);
        if (val < -1) val = -1;
        if (val > +1) val = +1;
        return acos(val);
    }
    CircleUnion() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void init() {
        n = 0;
        seg.clear(), cover.clear();
        arc = pol = 0;
    }
    void add(double xx, double yy, double rr) {
        x[n] = xx, y[n] = yy, r[n] = rr, covered[n] = 0, n++;
    }
    void getarea(int i, double lef, double rig) {
        arc += 0.5 * r[i] * r[i] * (rig - lef - sin(rig - lef));
        double x1 = x[i] + r[i] * cos(lef), y1 = y[i] + r[i] * sin(lef);
        double x2 = x[i] + r[i] * cos(rig), y2 = y[i] + r[i] * sin(rig);
        pol += x1 * y2 - x2 * y1;
    }
    double solve() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (!sign(x[i] - x[j]) && !sign(y[i] - y[j]) && !sign(r[i] - r[j])) {
                    r[i] = 0.0;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j && sign(r[j] - r[i]) >= 0 && sign(dist(x[i], y[i], x[j], y[j]) - (r[j] - r[i])) <= 0) {
                    covered[i] = 1;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (sign(r[i]) && !covered[i]) {
                seg.clear();
                for (int j = 0; j < n; j++) {
                    if (i != j) {
                        double d = dist(x[i], y[i], x[j], y[j]);
                        if (sign(d - (r[j] + r[i])) >= 0 || sign(d - abs(r[j] - r[i])) <= 0) {
                            continue;
                        }
                        double alpha = atan2(y[j] - y[i], x[j] - x[i]);
                        double beta = angle(r[i], d, r[j]);
                        pair<double, double> tmp(alpha - beta, alpha + beta);
                        if (sign(tmp.first) <= 0 && sign(tmp.second) <= 0) {
                            seg.push_back(pair<double, double>(2 * PI + tmp.first, 2 * PI + tmp.second));
                        }
                        else if (sign(tmp.first) < 0) {
                            seg.push_back(pair<double, double>(2 * PI + tmp.first, 2 * PI));
                            seg.push_back(pair<double, double>(0, tmp.second));
                        }
                        else {
                            seg.push_back(tmp);
                        }
                    }
                }
                sort(seg.begin(), seg.end());
                double rig = 0;
                for (vector<pair<double, double> >::iterator iter = seg.begin(); iter != seg.end(); iter++) {
                    if (sign(rig - iter->first) >= 0) {
                        rig = max(rig, iter->second);
                    }
                    else {
                        getarea(i, rig, iter->first);
                        rig = iter->second;
                    }
                }
                if (!sign(rig)) {
                    arc += r[i] * r[i] * PI;
                }
                else {
                    getarea(i, rig, 2 * PI);
                }
            }
        }
        return pol / 2.0 + arc;
    }
} CU;
double area_of_triangle(PT a, PT b, PT c) {
    return fabs(cross(b - a, c - a) * 0.5);
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
int is_point_in_triangle(PT a, PT b, PT c, PT p) {
    if (sign(cross(b - a,c - a)) < 0) swap(b, c);
    int c1 = sign(cross(b - a,p - a));
    int c2 = sign(cross(c - b,p - b));
    int c3 = sign(cross(a - c,p - c));
    if (c1<0 || c2<0 || c3 < 0) return 1;
    if (c1 + c2 + c3 != 3) return 0;
    return -1;
}
double perimeter(vector<PT> &p) {
    double ans=0; int n = p.size();
    for (int i = 0; i < n; i++) ans += dist(p[i], p[(i + 1) % n]);
    return ans;
}
double area(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i + 1) % n]);
    return fabs(ans) * 0.5;
}
// centroid of a (possibly non-convex) 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".
PT centroid(vector<PT> &p) {
    int n = p.size(); PT c(0, 0);
    double sum = 0;
    for (int i = 0; i < n; i++) sum += cross(p[i], p[(i + 1) % n]);
    double scale = 3.0 * sum;
    for (int i = 0; i < n; i++) {
        int j = (i + 1) % n;
        c = c + (p[i] + p[j]) * cross(p[i], p[j]);
    }
    return c / scale;
}
// 0 if cw, 1 if ccw
bool get_direction(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i + 1) % n]);
    if (sign(ans) > 0) return 1;
    return 0;
}
// it returns a point such that the sum of distances
// from that point to all points in p  is minimum
// O(n log^2 MX)
PT geometric_median(vector<PT> p) {
    auto tot_dist = [&](PT z) {
        double res = 0;
        for (int i = 0; i < p.size(); i++) res += dist(p[i], z);
        return res;
    };
```

```
  auto findY = [&](double x) {
    double yl = -1e5, yr = 1e5;
    for (int i = 0; i < 60; i++) {
      double ym1 = yl + (yr - yl) / 3;
      double ym2 = yr - (yr - yl) / 3;
      double d1 = tot_dist(PT(x, ym1));
      double d2 = tot_dist(PT(x, ym2));
      if (d1 < d2) yr = ym2;
      else yl = ym1;
    }
    return pair<double, double> (yl, tot_dist(PT(x, yl)));
  };
  double xl = -1e5, xr = 1e5;
  for (int i = 0; i < 60; i++) {
    double xm1 = xl + (xr - xl) / 3;
    double xm2 = xr - (xr - xl) / 3;
    double y1, d1, y2, d2;
    auto z = findY(xm1); y1 = z.first; d1 = z.second;
    z = findY(xm2); y2 = z.first; d2 = z.second;
    if (d1 < d2) xr = xm2;
    else xl = xm1;
  }
  return {xl, findY(xl).first };
}
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
  sort(v.begin(), v.end());
  vector<PT> up, dn;
  for (auto& p : v) {
    while (up.size() > 1 && orientation(up[up.size() - 2], up.back(),
p) >= 0) {
      up.pop_back();
    }
    while (dn.size() > 1 && orientation(dn[dn.size() - 2], dn.back(),
p) <= 0) {
      dn.pop_back();
    }
    up.push_back(p);
    dn.push_back(p);
  }
  v = dn;
  if (v.size() > 1) v.pop_back();
  reverse(up.begin(), up.end());
  up.pop_back();
  for (auto& p : up) {
    v.push_back(p);
  }
  if (v.size() == 2 && v[0] == v[1]) v.pop_back();
  return v;
}
 //checks if convex or not
bool is_convex(vector<PT> &p) {
  bool s[3]; s[0] = s[1] = s[2] = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n;
    int k = (j + 1) % n;
    s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
    if (s[0] && s[2]) return 0;
  }
  return 1;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
// it must be strictly convex, otherwise make it strictly convex first
int is_point_in_convex(vector<PT> &p, const PT& x) { // O(log n)
  int n = p.size(); assert(n >= 3);
  int a = orientation(p[0], p[1], x), b = orientation(p[0], p[n - 1], x);
  if (a < 0 || b > 0) return 1;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
        int mid = l + r >> 1;
        if (orientation(p[0], p[mid], x) >= 0) l = mid;
        else r = mid;
    }
    int k = orientation(p[l], p[r], x);
    if (k <= 0) return -k;
    if (l == 1 && a == 0) return 0;
    if (r == n - 1 && b == 0) return 0;
    return -1;
}
bool is_point_on_polygon(vector<PT> &p, const PT& z) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (is_point_on_seg(p[i], p[(i + 1) % n], z)) return 1;
    }
    return 0;
}
// returns 1e9 if the point is on the polygon
int winding_number(vector<PT> &p, const PT& z) { // O(n)
    if (is_point_on_polygon(p, z)) return 1e9;
    int n = p.size(), ans = 0;
    for (int i = 0; i < n; ++i) {
        int j = (i + 1) % n;
        bool below = p[i].y < z.y;
        if (below != (p[j].y < z.y)) {
            auto orient = orientation(z, p[j], p[i]);
            if (orient == 0) return 0;
            if (below == (orient > 0)) ans += below ? 1 : -1;
        }
    }
    return ans;
}
// -1 if strictly inside, 0 if on the polygon, 1 if strictly outside
int is_point_in_polygon(vector<PT> &p, const PT& z) { // O(n)
    int k = winding_number(p, z);
    return k == 1e9 ? 0 : k == 0 ? 1 : -1;
}
// id of the vertex having maximum dot product with z
// polygon must need to be convex
// top - upper right vertex
// for minimum dot prouct negate z and return -dot(z, p[id])
int extreme_vertex(vector<PT> &p, const PT &z, const int top) { // O(log n)
    int n = p.size();
    if (n == 1) return 0;
    double ans = dot(p[0], z); int id = 0;
    if (dot(p[top], z) > ans) ans = dot(p[top], z), id = top;
    int l = 1, r = top - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[mid + 1], z) >= dot(p[mid], z)) l = mid + 1;
        else r = mid;
    }
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    l = top + 1, r = n - 1;
    while (l < r) {
        int mid = l + r >> 1;
        if (dot(p[(mid + 1) % n], z) >= dot(p[mid], z)) l = mid + 1;
        else r = mid;
    }
    l %= n;
    if (dot(p[l], z) > ans) ans = dot(p[l], z), id = l;
    return id;
}
double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j]) >= 0) {
            ans = max(ans, dist2(p[i], p[j]));
            j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
    }
    return sqrt(ans);
}
double width(vector<PT> &p) {
    int n = (int)p.size();
    if (n <= 2) return 0;
    double ans = inf;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j]) >= 0) j = (j + 1) % n;
        ans = min(ans, dist_from_point_to_line(p[i], p[(i + 1) % n], p[j]));
        i++;
    }
    return ans;
}
// minimum perimeter
double minimum_enclosing_rectangle(vector<PT> &p) {
    int n = p.size();
    if (n <= 2) return perimeter(p);
    int mndot = 0; double tmp = dot(p[1] - p[0], p[0]);
    for (int i = 1; i < n; i++) {
        if (dot(p[1] - p[0], p[i]) <= tmp) {
            tmp = dot(p[1] - p[0], p[i]);
            mndot = i;
        }
    }
    double ans = inf;
    int i = 0, j = 1, mxdot = 1;
    while (i < n) {
        PT cur = p[(i + 1) % n] - p[i];
        while (cross(cur, p[(j + 1) % n] - p[j]) >= 0) j = (j + 1) % n;
        while (dot(p[(mxdot + 1) % n], cur) >= dot(p[mxdot], cur)) mxdot = (mxdot + 1) % n;
        while (dot(p[(mndot + 1) % n], cur) <= dot(p[mndot], cur)) mndot = (mndot + 1) % n;
        ans = min(ans, 2.0 * ((dot(p[mxdot], cur) / cur.norm() - dot(p[mndot], cur) / cur.norm()) + dist_from_point_to_line(p[i], p[(i + 1) % n], p[j])));
        i++;
    }
    return ans;
}
// given n points, find the minimum enclosing circle of the points
// call convex_hull() before this for faster solution
// expected O(n)
circle minimum_enclosing_circle(vector<PT> &p) {
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    circle c(p[0], 0);
    for (int i = 1; i < n; i++) {
        if (sign(dist(c.p, p[i]) - c.r) > 0) {
            c = circle(p[i], 0);
            for (int j = 0; j < i; j++) {
                if (sign(dist(c.p, p[j]) - c.r) > 0) {
                    c = circle((p[i] + p[j]) / 2, dist(p[i], p[j]) / 2);
                    for (int k = 0; k < j; k++) {
                        if (sign(dist(c.p, p[k]) - c.r) > 0) {
                            c = circle(p[i], p[j], p[k]);
                        }
                    }
                }
            }
        }
    }
    return c;
}
// returns a vector with the vertices of a polygon with everything
// to the left of the line going from a to b cut away.
vector<PT> cut(vector<PT> &p, PT a, PT b) {
    vector<PT> ans;
    int n = (int)p.size();
    for (int i = 0; i < n; i++) {
        double c1 = cross(b - a, p[i] - a);
        double c2 = cross(b - a, p[(i + 1) % n] - a);
        if (sign(c1) >= 0) ans.push_back(p[i]);
        if (sign(c1 * c2) < 0) {
            if (!is_parallel(p[i], p[(i + 1) % n], a, b)) {
                PT tmp; line_line_intersection(p[i], p[(i + 1) % n], a, b,
tmp);
                ans.push_back(tmp);
            }
        }
    }
    return ans;
}
// not necessarily convex, boundary is included in the intersection
// returns total intersected length
double polygon_line_intersection(vector<PT> p, PT a, PT b) {
    int n = p.size();
    p.push_back(p[0]);
    line l = line(a, b);
    double ans = 0.0;
    vector< pair<double, int> > vec;
    for (int i = 0; i < n; i++) {
        int s1 = sign(cross(b - a, p[i] - a));
        int s2 = sign(cross(b - a, p[i+1] - a));
        if (s1 == s2) continue;
        line t = line(p[i], p[i + 1]);
        PT inter = (t.v * l.c - l.v * t.c) / cross(l.v, t.v);
        double tmp = dot(inter, l.v);
        int f;
        if (s1 > s2) f = s1 && s2 ? 2 : 1;
        else f = s1 && s2 ? -2 : -1;
        vec.push_back(make_pair(tmp, f));
    }
    sort(vec.begin(), vec.end());
    for (int i = 0, j = 0; i + 1 < (int)vec.size(); i++){
        j += vec[i].second;
        if (j) ans += vec[i + 1].first - vec[i].first;
    }
    ans = ans / sqrt(dot(l.v, l.v));
    p.pop_back();
    return ans;
}
pair<PT, PT> convex_line_intersection(vector<PT> &p, PT a, PT b)
{
    return {{0, 0}, {0, 0}};
}
// minimum distance from a point to a convex polygon
// it assumes point does not lie strictly inside the polygon
double dist_from_point_to_polygon(vector<PT> &v, PT p) { //
O(log n)
    int n = (int)v.size();
    if (n <= 3) {
        double ans = inf;
        for(int i = 0; i < n; i++) ans = min(ans,
dist_from_point_to_seg(v[i], v[(i + 1) % n], p));
        return ans;
    }
    PT bscur, bs = angle_bisector(v[n - 1], v[0], v[1]);
    int ok,  i,  pw = 1,  ans = 0,  sgncur,  sgn = sign(cross(bs, p - v[0]));
    while (pw <= n) pw <<= 1;
    while ((pw >>= 1)) {
        if ((i = ans + pw) < n) {
            bscur = angle_bisector(v[i - 1], v[i], v[(i + 1) % n]);
            sgncur = sign(cross(bscur, p - v[i]));
            ok = sign(cross(bs, bscur)) >= 0 ? (sgn >= 0 || sgncur <= 0) : (sgn >= 0 && sgncur <= 0);
            if (ok) ans = i, bs = bscur, sgn = sgncur;
        }
    }
    return dist_from_point_to_seg(v[ans], v[(ans + 1) % n], p);
}
// minimum distance from convex polygon p to line ab
// returns 0 is it intersects with the polygon
// top - upper right vertex
double dist_from_polygon_to_line(vector<PT> &p, PT a, PT b, int top) { //O(log n)
    PT orth = (b - a).perp();
    if (orientation(a, b, p[0]) > 0) orth = (a - b).perp();
    int id = extreme_vertex(p, orth, top);
    if (dot(p[id] - a, orth) > 0) return 0.0; //if orth and a are in the same half of the line, then poly and line intersects
    return dist_from_point_to_line(a, b, p[id]); //does not intersect
}
// minimum distance from a convex polygon to another convex polygon
double dist_from_polygon_to_polygon(vector<PT> &p1, vector<PT> &p2) { // O(n log n)
    double ans = inf;
    for (int i = 0; i < p1.size(); i++) {
        ans = min(ans, dist_from_point_to_polygon(p2, p1[i]));
    }
    for (int i = 0; i < p2.size(); i++) {
        ans = min(ans, dist_from_point_to_polygon(p1, p2[i]));
    }
    return ans;
}
// maximum distance from a convex polygon to another convex polygon
double maximum_dist_from_polygon_to_polygon(vector<PT> &u, vector<PT> &v){ //O(n)
    int n = (int)u.size(), m = (int)v.size();
    double ans = 0;
    if (n < 3 || m < 3) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) ans = max(ans, dist2(u[i], v[j]));
        }
        return sqrt(ans);
    }
    if (u[0].x > v[0].x) swap(n, m), swap(u, v);
    int i = 0, j = 0, step = n + m + 10;
    while (j + 1 < m && v[j].x < v[j + 1].x) j++ ;
    while (step--) {
        if (cross(u[(i + 1)%n] - u[i], v[(j + 1)%m] - v[j]) >= 0) j = (j + 1) % m;
        else i = (i + 1) % n;
        ans = max(ans, dist2(u[i], v[j]));
    }
    return sqrt(ans);
}
pair<PT, int> point_poly_tangent(vector<PT> &p, PT Q, int dir, int l, int r) {
    while (r - l > 1) {
        int mid = (l + r) >> 1;
        bool pvs = orientation(Q, p[mid], p[mid - 1]) != -dir;
        bool nxt = orientation(Q, p[mid], p[mid + 1]) != -dir;
        if (pvs && nxt) return {p[mid], mid};
        if (!(pvs || nxt)) {
            auto p1 = point_poly_tangent(p, Q, dir, mid + 1, r);
            auto p2 = point_poly_tangent(p, Q, dir, l, mid - 1);
            return orientation(Q, p1.first, p2.first) == dir ? p1 : p2;
        }
        if (!pvs) {
            if (orientation(Q, p[mid], p[l]) == dir)  r = mid - 1;
            else if (orientation(Q, p[l], p[r]) == dir) r = mid - 1;
            else l = mid + 1;
        }
        if (!nxt) {
            if (orientation(Q, p[mid], p[l]) == dir)  l = mid + 1;
            else if (orientation(Q, p[l], p[r]) == dir) r = mid - 1;
            else l = mid + 1;
        }
    }
    pair<PT, int> ret = {p[l], l};
    for (int i = l + 1; i <= r; i++) ret = orientation(Q, ret.first, p[i]) != dir ? make_pair(p[i], i) : ret;
    return ret;
}
// (cw, ccw) tangents from a point that is outside this convex polygon
// returns indexes of the points
pair<int, int> tangents_from_point_to_polygon(vector<PT> &p, PT Q){
    int cw = point_poly_tangent(p, Q, 1, 0, (int)p.size() - 1).second;
    int ccw = point_poly_tangent(p, Q, -1, 0, (int)p.size() - 1).second;
    return make_pair(cw, ccw);
}
// calculates the area of the union of n polygons (not necessarily convex).
// the points within each polygon must be given in CCW order.
// complexity: O(N^2), where N is the total number of points
double rat(PT a, PT b, PT p) {
        return !sign(a.x - b.x) ? (p.y - a.y) / (b.y - a.y) : (p.x - a.x) / (b.x - a.x);
 };
double polygon_union(vector<vector<PT>> &p) {
    int n = p.size();
    double ans=0;
    for(int i = 0; i < n; ++i) {
        for (int v = 0; v < (int)p[i].size(); ++v) {
            PT a = p[i][v], b = p[i][(v + 1) % p[i].size()];
            vector<pair<double, int>> segs;
            segs.emplace_back(0,  0), segs.emplace_back(1,  0);
            for(int j = 0; j < n; ++j) {
                if(i != j) {
                    for(size_t u = 0; u < p[j].size(); ++u) {
                        PT c = p[j][u], d = p[j][(u + 1) % p[j].size()];
                        int sc = sign(cross(b - a, c - a)), sd = sign(cross(b - a, d - a));
                        if(!sc && !sd) {
                            if(sign(dot(b - a, d - c)) > 0 && i > j) {
                                segs.emplace_back(rat(a, b, c), 1), segs.emplace_back(rat(a, b, d),  -1);
                            }
                        }
                        else {
                            double sa = cross(d - c, a - c), sb = cross(d - c, b - c);
                            if(sc >= 0 && sd < 0) segs.emplace_back(sa / (sa - sb), 1);
                            else if(sc < 0 && sd >= 0) segs.emplace_back(sa / (sa - sb),  -1);
                        }
                    }
                }
            }
            sort(segs.begin(),  segs.end());
            double pre = min(max(segs[0].first, 0.0), 1.0), now, sum = 0;
            int cnt = segs[0].second;
            for(int j = 1; j < segs.size(); ++j) {
                now = min(max(segs[j].first, 0.0), 1.0);
                if (!cnt) sum += now - pre;
                cnt += segs[j].second;
                pre = now;
            }
            ans += cross(a, b) * sum;
        }
    }
    return ans * 0.5;
}
// contains all points p such that: cross(b - a, p - a) >= 0
struct HP {
    PT a, b;
    HP() {}
    HP(PT a, PT b) : a(a), b(b) {}
    HP(const HP& rhs) : a(rhs.a), b(rhs.b) {}
    int operator < (const HP& rhs) const {
        PT p = b - a;
        PT q = rhs.b - rhs.a;
        int fp = (p.y < 0 || (p.y == 0 && p.x < 0));
        int fq = (q.y < 0 || (q.y == 0 && q.x < 0));
        if (fp != fq) return fp == 0;
        if (cross(p, q)) return cross(p, q) > 0;
        return cross(p, rhs.b - a) < 0;
    }
    PT line_line_intersection(PT a, PT b, PT c, PT d) {
        b = b - a; d = c - d; c = c - a;
        return a + b * cross(c, d) / cross(b, d);
    }
    PT intersection(const HP &v) {
        return line_line_intersection(a, b, v.a, v.b);
    }
};
int check(HP a, HP b, HP c) {
    return cross(a.b - a.a, b.intersection(c) - a.a) > -eps; //-eps to include polygons of zero area (straight lines, points)
}
// consider half-plane of counter-clockwise side of each line
// if lines are not bounded add infinity rectangle
// returns a convex polygon, a point can occur multiple times though
// complexity: O(n log(n))
vector<PT> half_plane_intersection(vector<HP> h) {
    sort(h.begin(), h.end());
    vector<HP> tmp;
    for (int i = 0; i < h.size(); i++) {
        if (!i || cross(h[i].b - h[i].a, h[i - 1].b - h[i - 1].a)) {
            tmp.push_back(h[i]);
        }
    }
    h = tmp;
    vector<HP> q(h.size() + 10);
    int qh = 0, qe = 0;
    for (int i = 0; i < h.size(); i++) {
        while (qe - qh > 1 && !check(h[i], q[qe - 2], q[qe - 1])) qe--;
        while (qe - qh > 1 && !check(h[i], q[qh], q[qh + 1])) qh++;
        q[qe++] = h[i];
    }
    while (qe - qh > 2 && !check(q[qh], q[qe - 2], q[qe - 1])) qe--;
    while (qe - qh > 2 && !check(q[qe - 1], q[qh], q[qh + 1])) qh++;
    vector<HP> res;
    for (int i = qh; i < qe; i++) res.push_back(q[i]);
    vector<PT> hull;
    if (res.size() > 2) {
        for (int i = 0; i < res.size(); i++) {
            hull.push_back(res[i].intersection(res[(i + 1) % ((int)res.size())]));
        }
    }
    return hull;
}
// a and b are strictly convex polygons of DISTINCT points
// returns a convex hull of their minkowski sum with distinct points
vector<PT> minkowski_sum(vector<PT> &a, vector<PT> &b) {
    int n = (int)a.size(), m = (int)b.size();
    int i = 0, j = 0; //assuming a[i] and b[j] both are (left, bottom)-most points
    vector<PT> c;
    c.push_back(a[i] + b[j]);
    while (i + 1 < n || j + 1 < m){
        PT p1 = a[i] + b[(j + 1) % m];
        PT p2 = a[(i + 1) % n] + b[j];
        int t = orientation(c.back(), p1, p2);
        if (t >= 0) j = (j + 1) % m;
        if (t <= 0) i = (i + 1) % n, p1 = p2;
        if (t == 0) p1 = a[i] + b[j];
        if (p1 == c[0]) break;
        c.push_back(p1);
    }
    return c;
}
// system should be translated from circle center
double triangle_circle_intersection(PT c, double r, PT a, PT b) {
    double sd1 = dist2(c, a), sd2 = dist2(c, b);
    if(sd1 > sd2) swap(a, b), swap(sd1, sd2);
    double sd = dist2(a, b);
    double d1 = sqrtl(sd1), d2 = sqrtl(sd2), d = sqrt(sd);
    double x = abs(sd2 - sd - sd1) / (2 * d);
    double h = sqrtl(sd1 - x * x);
    if(r >= d2) return h * d / 2;
    double area = 0;
    if(sd + sd1 < sd2) {
        if(r < d1) area = r * r * (acos(h / d2) - acos(h / d1)) / 2;
        else {
            area = r * r * ( acos(h / d2) - acos(h / r)) / 2;
            double y = sqrtl(r * r - h * h);
            area += h * (y - x) / 2;
        }
    }
    else {
        if(r < h) area = r * r * (acos(h / d2) + acos(h / d1)) / 2;
        else {
            area += r * r * (acos(h / d2) - acos(h / r)) / 2;
            double y = sqrtl(r * r - h * h);
            area += h * y / 2;
            if(r < d1) {
                area += r * r * (acos(h / d1) - acos(h / r)) / 2;
                area += h * y / 2;
            }
            else area += h * x / 2;
        }
    }
    return area;
}
// intersection between a simple polygon and a circle
double polygon_circle_intersection(vector<PT> &v, PT p, double r) {
    int n = v.size();
    double ans = 0.00;
    PT org = {0, 0};
    for(int i = 0; i < n; i++) {
        int x = orientation(p, v[i], v[(i + 1) % n]);
        if(x == 0) continue;
        double area = triangle_circle_intersection(org, r, v[i] - p, v[(i + 1) % n] - p);
        if (x < 0) ans -= area;
        else ans += area;
    }
    return abs(ans);
}
// find a circle of radius r that contains as many points as possible
// O(n^2 log n);
double maximum_circle_cover(vector<PT> p, double r, circle &c) {
    int n = p.size();
    int ans = 0;
    int id = 0; double th = 0;
    for (int i = 0; i < n; ++i) {
        // maximum circle cover when the circle goes through this point
        vector<pair<double, int>> events = {{-PI, +1}, {PI, -1}};
        for (int j = 0; j < n; ++j) {
            if (j == i) continue;
            double d = dist(p[i], p[j]);
            if (d > r * 2) continue;
            double dir = (p[j] - p[i]).arg();
```

```
      double ang = acos(d / 2 / r);
      double st = dir - ang, ed = dir + ang;
      if (st > PI) st -= PI * 2;
      if (st <= -PI) st += PI * 2;
      if (ed > PI) ed -= PI * 2;
      if (ed <= -PI) ed += PI * 2;
      events.push_back({st, +1});
      events.push_back({ed, -1});
      if (st > ed) {
        events.push_back({-PI, +1});
        events.push_back({+PI, -1});
      }
    }
    sort(events.begin(), events.end());
    int cnt = 0;
    for (auto &&e: events) {
      cnt += e.second;
      if (cnt > ans) {
        ans = cnt;
        id = i; th = e.first;
      }
    }
  }
  PT w = PT(p[id].x + r * cos(th), p[id].y + r * sin(th));
  c = circle(w, r); //best_circle
  return ans;
}
// radius of the maximum inscribed circle in a convex polygon
double maximum_inscribed_circle(vector<PT> p) {
  int n = p.size();
  if (n <= 2) return 0;
  double l = 0, r = 20000;
  while (r - l > eps) {
    double mid = (l + r) * 0.5;
    vector<HP> h;
    const int L = 1e9;
    h.push_back(HP(PT(-L, -L), PT(L, -L)));
    h.push_back(HP(PT(L, -L), PT(L, L)));
    h.push_back(HP(PT(L, L), PT(-L, L)));
    h.push_back(HP(PT(-L, L), PT(-L, -L)));
    for (int i = 0; i < n; i++) {
      PT z = (p[(i + 1) % n] - p[i]).perp();
      z = z.truncate(mid);
      PT y = p[i] + z, q = p[(i + 1) % n] + z;
      h.push_back(HP(p[i] + z, p[(i + 1) % n] + z));
    }
    vector<PT> nw = half_plane_intersection(h);
    if (!nw.empty()) l = mid;
    else r = mid;
  }
  return l;
}
```

Check if there exists a point that all ranges cover:

```
bool sortby(const pair<ll, ll>& a,
            const pair<ll, ll>& b)
{
    if (a.first != b.first)
        return a.first < b.first;
    return (a.second < b.second);
}


// Function that returns true if any k
// segments overlap at any point
bool kOverlap(vector<pair<ll, ll> > pairs, ll k)
{
    // Vector to store the starting point
    // and the ending point
vector<pair<ll, ll> > vec;
for (ll i = 0; i < pairs.size(); i++) {
   // Starting points are marked by -1
   // and ending points by +1
   vec.push_back({ pairs[i].first, -1 });
   vec.push_back({ pairs[i].second, +1 });
}
// Sort the vector by first element
sort(vec.begin(), vec.end());
// Stack to store the overlaps
stack<pair<ll, ll> > st;
for (int i = 0; i < vec.size(); i++) {
   // Get the current element
   pair<ll, ll> cur = vec[i];

   // If it is the starting point
   if (cur.second == -1) {
      // Push it in the stack
      st.push(cur);
   }
   // It is the ending point
   else {
      // Pop an element from stack
      st.pop();
   }
   // If more than k ranges overlap
   if (st.size() >= k) {
      return true;
   }
}
return false;
}
```

Count the number of rectangle with given points:

```
// Function to find number of possible rectangles
int countRectangles(vector<pair<int, int> >& ob)
{

  // Creating TreeSet containing elements
  set<pair<int, int> > it;

  // Inserting the pairs in the set
  for (int i = 0; i < ob.size(); ++i) {
    it.insert(ob[i]);
  }

  int ans = 0;
  for (int i = 0; i < ob.size(); ++i)
  {
    for (int j = 0; j < ob.size(); ++j)
    {
      if (ob[i].first != ob[j].first
        && ob[i].second != ob[j].second)
      {

        // Searching the pairs in the set
        if (it.count({ ob[i].first, ob[j].second })
          && it.count(
            { ob[j].first, ob[i].second }))
        {

          // Increase the answer
          ++ans;
        }
      }
    }
  }

  // Return the final answer
  return ans / 4;
}
```

Maximum possible rectangles:

```
void maxRectanglesPossible(int N)
{
  // Invalid case
  if (N < 4 || N % 2 != 0) {
    cout << -1 << "\n";
  }
  else
    // Number of distinct rectangles.
    cout << (N / 2) - 1 << "\n";
}
```

Minimum number of straight lines to connect all points:

```
int minimumLines(vector<vector<int> >& arr)
{
  int n = arr.size();

  // Base case when there is only one point,
  // then min lines = 0
  if (n == 1)
    return 0;

  // Sorting in ascending order of X coordinate
  sort(arr.begin(), arr.end());

  int numoflines = 1;

  // Traverse through points and check
  // whether the slopes matches or not.
  // If they does not match
  // increment the count of lines
  for (int i = 2; i < n; i++) {
    int x1 = arr[i][0];
    int x2 = arr[i - 1][0];
    int x3 = arr[i - 2][0];
    int y1 = arr[i][1];
    int y2 = arr[i - 1][1];
    int y3 = arr[i - 2][1];
    int slope1 = (y3 - y2) * (x2 - x1);
    int slope2 = (y2 - y1) * (x3 - x2);

    if (slope1 != slope2)
      numoflines++;
  }

  // Return the num of lines
  return numoflines;
}
```

POINT INSIDE POLYGON?

```
struct Point {
  int x, y;
};
```

```
struct line {
    Point p1, p2;
};

bool onLine(line l1, Point p)
{
    // Check whether p is on the line or not
    if (p.x <= max(l1.p1.x, l1.p2.x)
        && p.x <= min(l1.p1.x, l1.p2.x)
        && (p.y <= max(l1.p1.y, l1.p2.y)
            && p.y <= min(l1.p1.y, l1.p2.y)))
        return true;

    return false;
}

int direction(Point a, Point b, Point c)
{
    int val = (b.y - a.y) * (c.x - b.x)
              - (b.x - a.x) * (c.y - b.y);

    if (val == 0)

        // Colinear
        return 0;

    else if (val < 0)

        // Anti-clockwise direction
        return 2;

    // Clockwise direction
    return 1;
}

bool isIntersect(line l1, line l2)
{
    // Four direction for two lines and points of other line
    int dir1 = direction(l1.p1, l1.p2, l2.p1);
    int dir2 = direction(l1.p1, l1.p2, l2.p2);
    int dir3 = direction(l2.p1, l2.p2, l1.p1);
    int dir4 = direction(l2.p1, l2.p2, l1.p2);

    // When intersecting
    if (dir1 != dir2 && dir3 != dir4)
        return true;

    // When p2 of line2 are on the line1
    if (dir1 == 0 && onLine(l1, l2.p1))
        return true;

    // When p1 of line2 are on the line1
    if (dir2 == 0 && onLine(l1, l2.p2))
        return true;

    // When p2 of line1 are on the line2
    if (dir3 == 0 && onLine(l2, l1.p1))
        return true;

    // When p1 of line1 are on the line2
    if (dir4 == 0 && onLine(l2, l1.p2))
        return true;

    return false;
}

bool checkInside(Point poly[], int n, Point p)
{

    // When polygon has less than 3 edge, it is not polygon
```

```
  if (n < 3)
    return false;

  // Create a point at infinity, y is same as point p
  line exline = { p, { 9999, p.y } };
  int count = 0;
  int i = 0;
  do {

    // Forming a line from two consecutive points of
    // poly
    line side = { poly[i], poly[(i + 1) % n] };
    if (isIntersect(side, exline)) {

      // If side is intersects exline
      if (direction(side.p1, p, side.p2) == 0)
        return onLine(side, p);
      count++;
    }
    i = (i + 1) % n;
  } while (i != 0);

  // When count is odd
  return count & 1;
}
```

*****STRINGS***

Pattern matching:KMP

```
const ll MAX_N = 1e5+10;
char s[MAX_N], pat[MAX_N];  // 1-indexed
ll lps[MAX_N];     // lps[i] = longest proper prefix-suffix in i length's prefix

void gen_lps(ll plen){
  ll now;
  lps[0] = lps[1] = now = 0;
  for(ll i = 2; i <= plen; i++) {
    while(now != 0 && pat[now+1] != pat[i]) now = lps[now];
    if(pat[now+1] == pat[i]) lps[i] = ++now;
    else lps[i] = now = 0;
  }
}
```

Lexicographically compare of two strings:

```
string compare(string s1,string s2){

  ll n=s1.size();
  ll a=0,b=0;
  for(ll i=0;i<n;i++){
    if(s1[i]=='1' && s2[i]=='0') return s1;
    if(s2[i]=='1' && s1[i]=='0') return s2;
  }
  return s2;
}
```

Custom comparator sort numeric strings

```
bool myCmp(string s1, string s2)
{
    if (s1.size() == s2.size()) {
        return s1 < s2;
    }
     else {
        return s1.size() < s2.size();
    }
}

// in main
```

```
//sort(v.begin(), v.end(), myCmp);
```

String operations:

```
string Addition(string a,string b);
string Multiplication(string a,string b);
string Multiplication(string a,ll k);
string Subtraction(string a,string b);
string Division(string a,string b);
string Division(string a,ll k);
string Div_mod(string a,string b);
ll Div_mod(string a,ll k);
string cut_leading_zero(string a);
ll compare(string a,string b);


string Multiplication(string a,string b){
    ll i,j,multi,carry;
    string ans,temp;
    ans="0";
    Fo(j,SZ(b)-1,-1){
        temp="";
        carry=0;
        Fo(i,SZ(a)-1,-1){
            multi=(a[i]-'0')*(b[j]-'0')+carry;
            temp+=(multi%10+'0');
            carry=multi/10;
        }
        if(carry) temp+=(carry+'0');
        rev(temp);
        temp+=string(SZ(b)-j-1,'0');
        ans=Addition(ans,temp);
    }
    ans=cut_leading_zero(ans);
    return ans;
}
```

```
string Multiplication(string a,ll k){
    string ans;
    ll i,sum,carry=0;
    Fo(i,SZ(a)-1,-1){
        sum=(a[i]-'0')*k+carry;
        carry=sum/10;
        ans+=(sum%10)+'0';
    }
    while(carry){
        ans+=(carry%10)+'0';
        carry/=10;
    }
    rev(ans);
    ans=cut_leading_zero(ans);
    return ans;
}


string Addition(string a,string b){
    ll carry=0,i;
    string ans;
    if(SZ(a)>SZ(b)) b=string(SZ(a)-SZ(b),'0')+b;
    if(SZ(b)>SZ(a)) a=string(SZ(b)-SZ(a),'0')+a;
    ans.resize(SZ(a));
    Fo(i,SZ(a)-1,-1){
        ll sum=carry+a[i]+b[i]-96;
        ans[i]=sum%10+'0';
        carry=sum/10;
    }
    if(carry) ans.insert(0,string(1,carry+'0'));
    ans=cut_leading_zero(ans);
    return ans;
}


string Subtraction(string a,string b){
    ll borrow=0,i,sub;
    string ans;
    if(SZ(b)<SZ(a)) b=string(SZ(a)-SZ(b),'0')+b;
    Fo(i,SZ(a)-1,-1){
        sub=a[i]-b[i]-borrow;
        if(sub<0){
            sub+=10;
            borrow=1;
        }else{
            borrow=0;
        }
        ans+=sub+'0';
    }
    rev(ans);
    ans=cut_leading_zero(ans);
    return ans;
}

string Division(string a,string b){
    string mod, temp, ans="0";
    ll i, j;

    fo(i,SZ(a)){
        mod+=a[i];
        mod=cut_leading_zero(mod);
        fo(j,10){
            temp=Multiplication(b,j);
            if(compare(temp,mod)==1) break;
        }
        temp=Multiplication(b,j-1);
        mod=Subtraction(mod,temp);
        ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return ans;
}

string Division(string a,ll k){
    ll i, sum=0;
    string ans = "0";

    fo(i,SZ(a)){
        sum=(sum*10+(a[i]-'0'));
        ans+=(sum/k)+'0';
        sum=sum%k;
    }
    ans=cut_leading_zero(ans);
    return ans;
}

string Div_mod(string a,string b){
    string mod, temp, ans="0";
    ll i, j;

    fo(i,SZ(a)){
        mod+=a[i];
        mod=cut_leading_zero(mod);

        Fo(j,1,10){
            temp=Multiplication(b,j);
            if(compare(temp,mod)>0) break;
        }
        temp=Multiplication(b,j-1);
        mod=Subtraction(mod,temp);
        ans+=(j-1)+'0';
    }
    mod=cut_leading_zero(mod);
    ans=cut_leading_zero(ans);
    return mod;
}
```

```
ll Div_mod(string a,ll k){
   ll i, sum=0;
   fo(i,SZ(a)) sum=(sum*10+(a[i]-'0'))%k;
   return sum;
}

ll compare(string a,string b){
   ll i;
   a=cut_leading_zero(a);
   b=cut_leading_zero(b);

   if(SZ(a)>SZ(b)) return 1;
   if(SZ(a)<SZ(b)) return -1;
   fo(i,SZ(a))
   {
      if( a[i]>b[i] ) return 1;
      else if(a[i]<b[i]) return -1;
   }
   return 0;
}

string cut_leading_zero(string a){
   string s;
   ll i,j;
   if(a[0]!='0') return a;
   fo(i,SZ(a)-1) if(a[i]!='0') break;
   Fo(j,i,SZ(a)) s+=a[j];
   return s;
}


ll KMP(ll slen, ll plen){
   ll now = 0;
   for(ll i = 1; i <= slen; i++) {
      while(now != 0 && pat[now+1] != s[i]) now = lps[now];
      if(pat[now+1] == s[i]) ++now;
      else now = 0;
      // now is the length of the longest prefix of pat, which
      // ends as a substring of s in index i.
      if(now == plen) return 1;
   }
   return 0;
}
// slen = length of s, plen = length of pat
// call gen_lps(plen); to generate LPS (failure) array
// call KMP(slen, plen) to find pat in s
```

Binary to Roman:

```
string Bin_to_Roman(ll n){
   string s="";
   while(n>=1000) s.pb('M'),n-=1000;
   while(n>=900) s.pb('C'),s.pb('M'),n-=900;
   while(n>=500) s.pb('D'),n-=500;
   while(n>=400) s.pb('C'),s.pb('D'),n-=400;
   while(n>=100) s.pb('C'),n-=100;
   while(n>=90) s.pb('X'),s.pb('C'),n-=90;
   while(n>=50) s.pb('L'),n-=50;
   while(n>=40) s.pb('X'),s.pb('L'),n-=40;
   while(n>=10) s.pb('X'),n-=10;
   while(n>=9) s.pb('I'),s.pb('X'),n-=9;
   while(n>=5) s.pb('V'),n-=5;
   while(n>=4) s.pb('I'),s.pb('V'),n-=4;
   while(n) s.pb('I'),n--;
   return s;
}
```

COUNT unique substrings:

```
int count_unique_substrings(string const& s) {
   int n = s.size();
```

```
    const int p = 31;

    const int m = 1e9 + 9;

    vector<long long> p_pow(n);

    p_pow[0] = 1;

    for (int i = 1; i < n; i++)

        p_pow[i] = (p_pow[i-1] * p) % m;



    vector<long long> h(n + 1, 0);

    for (int i = 0; i < n; i++)

        h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;

    int cnt = 0;

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

        set<long long> hs;

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

            long long cur_h = (h[i + l] + m - h[i]) % m;

            cur_h = (cur_h * p_pow[n-i-1]) % m;

            hs.insert(cur_h);

        }

        cnt += hs.size();

    }

    return cnt;

}
```

*****SEARCHING AND SORTING**********

BINARY Search:

```
ll func(ll pos){



}



ll bs(ll low,ll high){

    ll mid;

    while(high-low>=2){

        mid=(high+low)>>1;

        //cout<<mid<<" "<<func(mid)<<endl;

        if(func(mid)){

            low=mid;

        }

        else{

            high=mid-1;

        }

    }

    if(func(high)) return high;

    else return low;

}
```

Binary search(real numbers):

```
double func(double mid){



}



double bs(double l,double r){

    double eps=1e-9;        //set the error limit here

    while(r-l>eps) {

        double mid=l+(r-l)/2;

        if (func(mid)) l=mid;

        else r=mid;

    }

    return l;

}
```

TERNARY SEARCH:

```
double func(double mid){



}



double ts(double l, double r){

    double eps=1e-9;          //set the error limit here

    while (r-l>eps){

        double mid1=l+(r-l)/3;

        double mid2=r-(r-l)/3;

        double f1=func(mid1);     //evaluates the function at mid1

        double f2=func(mid2);     //evaluates the function at mid2
        if (f1<f2) l = mid1;       //change f1>f2 if needed minimum
        else r=mid2;
    }
    return func(l);             //return the maximum of func(x) in [l, r]
}
```

BS on string:

```
string bs(string low,string high){
    string mid;
    while(compare(Subtraction(high,low),"1")!=-1){
        mid=Division(Addition(high,low),2);
        //cout<<mid<<" "<<fnc(mid)<<endl;
        if(func(mid)){
            low=Addition(mid,"1");
        }
        else{
            high=mid;
        }
    }
    return high;
    // if(func(high)) return high;
    // else return low;
}
```

TERNARY SEARCH(Int):

```
long double func(ll pos){


}


ll ts(ll low,ll high){
    ll mid;
    while(high-low>=2){
        mid=(high+low)>>1;
        //cout<<mid<<" "<<func(mid)<<endl;
        if(func(mid)<func(mid+1)){
            high=mid;
        }
        else{
            low=mid;
        }
    }
    if(func(high)<func(low)) return high;
    else return low;
}
```

Comparators:

//for pair

```
bool cmp(pair<ll,ll> a,pair<ll,ll> b){
    if(a.second!=b.second) return a.second>b.second;
    return a.first<b.first;
}
```

//for set

```
struct cmp {
     bool operator() (const pair<int, int> &a, const pair<int, int> &b) const {
         int lena = a.second - a.first + 1;
         int lenb = b.second - b.first + 1;
         if (lena == lenb) return a.first < b.first;
         return lena > lenb;
    }
};
```

//for descending

```
bool cmp(int a,int b){
    return a>b;
}
```

Compress vector:

```
void compress(vl &v,ll n){
    ll i;
    set<pll> st;
    fo(i,n){
        st.insert({v[i],i});
    }
    i=1;
    for(auto &it:st){
        v[it.second]=i;
        i++;
    }
}
```

SORT check:

```
bool check(int ar[],int n){
    if(n==1)
        return 1;
     bool restarray=check(ar+1,n-1);
     return (restarray && (ar[0]<ar[1]));
}
```

Count sort:

```
void countSort(vl &v){
    ll i=0,n=v.size(),mx=*max_element(all(v));
    vl cnt(mx+1,0);
    vl sorted(n);
    fo(i,n) cnt[v[i]]++;
    Fo(i,1,cnt.size()) cnt[i]+=cnt[i-1];
    Fo(i,n-1,-1) sorted[--cnt[v[i]]]=v[i];
    fo(i,v.size()) v[i]=sorted[i];
}
```

TOPOLOGICAL SORT:

```
vector<ll> topoSort(ll n){
    queue<ll> q;
    vector<ll> indegree(n,0);
    for(int i=0;i<n;i++){
        for(auto &it:g[i]){
            indegree[it]++;
        }
    }
    for(int i=0;i<n;i++){
        if(!indegree[i]) q.push(i);
    }
    vector<ll> topo;
    while(!q.empty()){
        auto node=q.front();
        q.pop();
        topo.pb(node);
        for(auto &it:g[node]){
            indegree[it]--;
            if(indegree[it]==0){
                q.push(it);
            }
        }
    }
    return topo;
}
```

******GRAPH AND TREES**********\

Reset function:

```
void reset(ll n){
    for(ll i=0;i<=n;i++){
        g[i].clear();
        dist[i]=INF;
        vis[i]=0;
    }
}
```

DFS:

On graph:

```
bool vis[N];
int subtree_sum[N];
void dfs(ll vertex){
    /*
    take action on vertex after entering the vertex
    */
    vis[vertex]=true;
    for(ll child: g[vertex]){
        /*
        take action on child before entering the child node
        */
        if(vis[child]) continue;
        dfs(child);
        subtree_sum[vertex]+=subtree_sum[child];
        /*
        take action on child after entering the child node
        */
    }
    /*
    take action on vertex before exiting the vertex
    */
}
```

DFS:

On grid:

```
ll n,m,t=0;
vector<string> g;
vpll Move={ {1,0},{-1,0},{0,1},{0,-1} };

bool vis[N][N];
bool isValid(ll x,ll y){
    return (x>=0&&y>=0&&x<n&&y<m&&vis[x][y]==0&&g[x][y]=='.');
}

void dfs(pll vertex){
    /*
    take action on vertex after entering the vertex
    */
    vis[vertex.first][vertex.second]=true;
    for(pll &child: Move){
        /*
        take action on child before entering the child node
        */
        ll x=child.first+vertex.first;
        ll y=child.second+vertex.second;
        if(!isValid(x,y)) continue;
        dfs({x,y});
        /*
        take action on child after entering the child node
        */
    }
    /*
    take action on vertex before exiting the vertex
    */
}
```

DIAMETER OF A TREE:

```
int diameter(int n){
    int i;
    dfs(1);
    int max_depth=-1;
    int max_d_node;
    Fo(i,1,n+1){
        if(max_depth<depth[i]){
            max_depth=depth[i];
            max_d_node=i;
        }
        depth[i]=0;;
    }
    dfs(max_d_node);
    Fo(i,1,n+1){
        if(max_depth<depth[i]){
            max_depth=depth[i];
        }
    }
    return max_depth;
}
```

DIAMETER OF A WEIGHTED TREE:

```
ll t=0;
vpll g[N];
ll depth[N];
void dfs(ll vertex,ll par=-1){
    for(auto child: g[vertex]){
        if(child.first==par) continue;
        depth[child.first]=depth[vertex]+child.second;
        dfs(child.first,vertex);
    }
}

ll diameter(ll n){
    ll i;
    dfs(0);
    ll max_depth=-1;
    ll max_d_node;
    fo(i,n){
        if(max_depth<depth[i]){
            max_depth=depth[i];
            max_d_node=i;
        }
        depth[i]=0;
    }
    dfs(max_d_node);
    fo(i,n){
        if(max_depth<depth[i]){
            max_depth=depth[i];
        }
    }
    return max_depth;
}
```

Depth/height of a tree:

```
int depth[N],height[N];
void dfs(int vertex,int par=0){
    for(int child: g[vertex]){
        if(child==par) continue;
        depth[child]=depth[vertex]+1;
        dfs(child,vertex);
        height[vertex]=max(height[vertex],height[child]+1);
    }
}
```

LCA:

```
vector<int> path(int vertex){
    vector<int> ans;
    while(vertex!=-1){
        ans.push_back(vertex);
        vertex=parent[vertex];
    }
    reverse(ans.begin(),ans.end());
    return ans;
}

int LCA(int n){
    int i;
    dfs(1);
    int x,y;
    cin>>x>>y;
    vector<int> path_x=path(x);
    vector<int> path_y=path(y);
    int mn_ln=min(path_x.size(),path_y.size());

    int lca=-1;
    fo(i,mn_ln){
        if(path_x[i]==path_y[i]){
            lca=path_x[i];
        }else{
            break;
        }
    }
    return lca;
}
```

BFS:

```
bool vis[N];
ll level[N];

void bfs(ll source){
    queue<ll> q;
    q.push(source);
    vis[source]=1;
    level[source]=0;
    while(!q.empty()){
        ll cur_v=q.front();
        q.pop();
        for(ll child:g[cur_v]){
            if(!vis[child]){
                q.push(child);
                vis[child]=1;
                level[child]=1+level[cur_v];
            }
        }
    }
}
```

BFS on grid:

```
vpll Move={ {1,0},{-1,0},{0,1},{0,-1} };

bool vis[N][N];
ll level[N][N];
ll n,m;

bool isValid(ll x,ll y){
    return (x>=0&&x<n&&y>=0&&y<m&&vis[x][y]==0);
}

void bfs(pll source){
    queue<pll> q;
    q.push(source);
    vis[source.first][source.second]=1;
    level[source.first][source.second]=0;
    while(!q.empty()){
        pll cur_v=q.front();
        q.pop();
        for(pll &child:Move){
            ll x=cur_v.first+child.first;
            ll y=cur_v.second+child.second;
            if(isValid(x,y)){
                q.push({x,y});
                vis[x][y]=1;
                level[x][y]=1+level[cur_v.first][cur_v.second];
            }
        }
    }
}
```

Multisource bfs:

```
const ll maxN=1e3+10;//for graph

const ll INF=1e9+10;

#define M 10000



//when edges dont have same weight...0 and 1
weights..use 0-1 bfs

 ll n,m;

 ll val[maxN][maxN];

 ll vis[maxN][maxN];

 ll lev[maxN][maxN];

 void reset(){



    for(ll i=0;i<n;i++){

        for(ll j=0;j<m;j++){

            vis[i][j]=0;

            lev[i][j]=INF;

        }

    }



 }

 bool isvalid(ll i,ll j){

    return i>=0 &&  j>=0 && i< n && j<m;



 }

  vector<pair<ll,ll> >movements={

 {0,1},{0,-1},{1,0},{-1,0},

 {1,1},{1,-1},{-1,1},{-1,-1}



 };
```

```
 ll bfs(){

     ll mx=0;

     for(ll i=0;i<n;i++){

        for(ll j=0;j<m;j++){

          mx=max(mx,val[i][j]);

        }

    }

    queue< pair<ll,ll> >q;



    for(ll i=0;i<n;i++){

        for(ll j=0;j<m;j++){

        if(mx==val[i][j]){

            q.push({i,j});

            lev[i][j]=0;

            vis[i][j]=1;

        }

        }

    }

    ll ans=0;

    while(!q.empty()){

        auto v=q.front();

        ll v_x=v.first;

        ll v_y=v.second;

        q.pop();

        for(auto movement : movements){

            ll child_x=movement.first+v_x;

            ll child_y=movement.second+v_y;

            if(!isvalid(child_x,child_y))
continue;
```

```
            if(vis[child_x][child_y]) continue;

            q.push({child_x,child_y});

lev[child_x][child_y]=lev[v_x][v_y]+1;
            vis[child_x][child_y]=1;
            ans=max(ans,lev[child_x][child_y]);
        }

    }
    return ans;
 }
```

0-1 Bfs:

```
const ll maxN=1e5+10;//for graph
const ll INF=1e9+10;
#define M 10000



vector<pair<ll,ll> >g[maxN];
vector<ll> lev(maxN,INF);
//when edges dont have same weight...0 and 1 weights..use 0-1 bfs
 ll n,m;

ll bfs(){

    deque<ll> q;
    q.push_back(1);
    lev[1]=0;
    while(!q.empty()){
        ll curr_v=q.front();
        q.pop_front();
        for(auto &child : g[curr_v]){
            ll child_v=child.first;
            ll weight=child.second;
            if(lev[curr_v]+weight<lev[child_v]){
                lev[child_v]=lev[curr_v]+weight;
                if(weight==1){
                    q.push_back(child_v);
                }
                else{
                    q.push_front(child_v);
                }
            }
        }
    }
    return lev[n]==INF? -1:lev[n];
}
```

SHORTEST CIRCLE(unwt & undirected):

```
ll bfs(ll source,ll n){
    ll ret=INT_MAX;
    vl par(n+1,-1);
    vl dist(n+1,INT_MAX);
    queue<ll> q;
    q.push(source);
    dist[source]=0;
    while(!q.empty()){
        ll cur_v=q.front();
        q.pop();
        for(ll child:g[cur_v]){
            if(dist[child]==INT_MAX){
                q.push(child);
                par[child]=cur_v;
                dist[child]=1+dist[cur_v];
            }else{
                if(par[cur_v]==child) continue;
                ret=min(ret,dist[child]+dist[cur_v]+1);
            }
        }
    }
    return ret;
}

ll shortest_cycle(ll n){
    ll ans=INT_MAX,i;
    fo(i,n) ans=min(ans,bfs(i,n));
    fo(i,n+1) g[i].clear();
    if(ans==INT_MAX) return -1;
    return ans;
}
```

Dijkstra(+find parent):

```
const ll N=1e5+10;
const ll INF=1e16+9;
vector<pair<ll,ll> > g[N];
vector<ll> dist(N,INF);
ll vis[N];
vector<ll>ans;
vector<ll>par(N);
ll n,m,k;
void dijkstra(int source){
    priority_queue<pair<ll,ll> > pq;
    pq.push({source,0});
    dist[source]=0;
    vis[source]=1;
    while(pq.size()){
        ll v=pq.top().first;
        ll v_dist=pq.top().second;
        pq.pop();
        if(v_dist>dist[v]) continue;
        vis[v]=1;
        for(auto &child:g[v]){
            ll child_v=child.first;
            ll wt=child.second;
          if(vis[child_v] && dist[v]+wt>dist[child_v]) continue;
            if(dist[v]+wt<dist[child_v]){
                dist[child_v]=dist[v]+wt;
                par[child_v]=v;
                pq.push({child_v,dist[child_v]});
            }
        }
    }
}

void func(ll vertex){
   ans.push_back(vertex);
   if(vertex==1) return;
    func(par[vertex]);
}
```

BELLMAN_FORD:

```
void bellman_ford(){
    ll x=-1;
    for(ll i=1;i<=n;i++){ dist[i]=INF;par[i]=-1;}
    dist[1]=0;

    for(ll i=0; i<n; i++){
        x=-1;
        for(ll node=1; node<=n; node++){
            //if(dist[node]==INF) continue;
            for(pair<ll,ll> a : g[node]){
                if(dist[a.first]>dist[node]+a.second){
                    dist[a.first]=dist[node]+a.second;
                    par[a.first]=node;
                    x=a.first;
                }
            }
        }
        //if(!x) break;
    }
    if(x==-1){
        cout<<"NO"<<endl;
    }
    else{
        //x can be on any cycle or reachable from some cycle
        vl path;
        for (ll i=0; i<n; i++) x = par[x];

        for(ll cur=x; ; cur=par[cur]) {
            //cout<<cur<<" ";
            path.push_back (cur);
            if (cur == x && path.size() > 1) break;

        }
        //cout<<endl;
        reverse(path.begin(), path.end());
        cout << "YES"<<endl;
       cout<<path<<endl;
    }

}
```

FLOYD WARSHALL:

```
ll dp[N][N];
const int INF=1e9;

void floyd_warshall(int n){
    ll i,j,k;
    fo(i,n+1){
        dp[i][i]=0;
    }
    Fo(k,1,n+1){
        Fo(i,1,n+1){
            Fo(j,1,n+1){
                if(dp[i][k]!=INF && dp[k][j]!=INF){
                    dp[i][j]=min(dp[i][j],dp[i][k]+dp[k][j]);
                }
            }
        }
    }
}
```

Disjoint set union:

```
int par[N];
int sz[N];
multiset<int> sizes;

void make(int v){
    par[v]=v;
    sz[v]=1;
    sizes.insert(1);
}

int find(int v){
    if(v==par[v]) return v;
    return par[v]=find(par[v]);
}

void merge(int a,int b){
    sizes.erase(sizes.find(sz[a]));
    sizes.erase(sizes.find(sz[b]));
    sizes.insert(sz[a]+sz[b]);
}

void Union(int a,int b){
    a=find(a);
    b=find(b);
    if(a!=b){
        if(sz[a]<sz[b]) swap(a,b);
        par[b]=a;
        // merge(a,b);
        sz[a]+=sz[b];
    }
}
```

DSU ON TREES:

```
#define maxn 100009

vector <ll> graph[maxn];
ll col[maxn], sz[maxn], cnt[maxn], ans[maxn];
bool big[maxn];

void szdfs(ll u, ll p)

{
    sz[u] = 1;
    for(ll i = 0; i < graph[u].size(); i++) {
        ll nd = graph[u][i];
        if(nd == p)
            continue;

        szdfs(nd, u);
        sz[u] += sz[nd];
    }
}

void add(ll u, ll p, ll x)
{
    cnt[ col[u] ] += x;
    for(auto v: graph[u])
        if(v != p && !big[v])
            add(v, u, x);
}

void dfs(ll u, ll p, bool keep)
{
    ll mx = -1, bigChild = -1;
    for(auto v : graph[u])
        if(v != p && sz[v] > mx)
            mx = sz[v], bigChild = v;

    for(auto v : graph[u])
        if(v != p && v != bigChild)
            dfs(v, u, 0);  /// run a dfs on small childs and clear them from cnt

    if(bigChild != -1) {
        dfs(bigChild, u, 1);
        big[bigChild] = 1;  /// bigChild marked as big and not cleared from cnt
    }

    add(u, p, 1);
    ///now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.

    if(bigChild != -1)
        big[bigChild] = 0;
    if(keep == 0)
        add(u, p, -1);
}

//szdfs(1,-1); dfs(1,-1,0);
```

*******SEGMENT TREE*********

```
const double EPS = 1e-9;

const int N = 2e5+10;

ll T=0;


ll tre[3*N];

ll lazy[3*N];


ll merge(ll x,ll y){

   return x+y;

}


void buildSegTree(vector<ll>& arr, ll treeIndex, ll lo, ll hi){


   if (lo == hi) {               // leaf node. store value in node.

      tre[treeIndex] = arr[lo];

      return;

   }


   ll mid = lo + (hi - lo) / 2;   // recurse deeper for children.

   buildSegTree(arr, 2 * treeIndex + 1, lo, mid);

   buildSegTree(arr, 2 * treeIndex + 2, mid + 1, hi);


   // merge build results

   tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex + 2]);

}


// call this method as buildSegTree(arr, 0, 0, n-1);

// Here arr[] is input array and n is its size.

ll querySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j){

   // query for arr[i..j]


   if (lo > j || hi < i)            // segment completely outside range

      return 0;                    // represents a null node


   if (i <= lo && j >= hi)           // segment completely inside range

      return tre[treeIndex];


   ll mid = lo + (hi - lo) / 2;      // partial overlap of current segment and queried range. Recurse deeper.


   if (i > mid)

      return querySegTree(2 * treeIndex + 2, mid + 1, hi, i, j);

   else if (j <= mid)

      return querySegTree(2 * treeIndex + 1, lo, mid, i, j);


   ll leftQuery = querySegTree(2 * treeIndex + 1, lo, mid, i, mid);

   ll rightQuery = querySegTree(2 * treeIndex + 2, mid + 1, hi, mid + 1, j);


   // merge query results

   return merge(leftQuery, rightQuery);

}


// call this method as querySegTree(0, 0, n-1, i, j);

// Here [i,j] is the range/interval you are querying.

// This method relies on "null" nodes being equivalent to storing zero.


void updateValSegTree(ll treeIndex, ll lo, ll hi, ll arrIndex, ll val)

{

   if (lo == hi) {               // leaf node. update element.

      tre[treeIndex] = val;

      return;

   }
```

```
    ll mid = lo + (hi - lo) / 2;   // recurse deeper for appropriate child

    if (arrIndex > mid)
        updateValSegTree(2 * treeIndex + 2, mid + 1, hi, arrIndex, val);
    else if (arrIndex <= mid)
        updateValSegTree(2 * treeIndex + 1, lo, mid, arrIndex, val);

    // merge updates
    tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex + 2]);
}

// call this method as updateValSegTree(0, 0, n-1, i, val);
// Here you want to update the value at index i with value val.

void updateLazySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j, ll val){
    if (lazy[treeIndex] != 0) {                         // this node is lazy
        tre[treeIndex] += (hi - lo + 1) * lazy[treeIndex]; // normalize current node by removing laziness

        if (lo != hi) {                                 // update lazy[] for children nodes
            lazy[2 * treeIndex + 1] += lazy[treeIndex];
            lazy[2 * treeIndex + 2] += lazy[treeIndex];
        }

        lazy[treeIndex] = 0;                            // current node processed. No longer lazy
    }

    if (lo > hi || lo > j || hi < i)
        return;                                         // out of range. escape.

    if (i <= lo && hi <= j) {                           // segment is fully within update range
        tre[treeIndex] += (hi - lo + 1) * val;          // update segment

        if (lo != hi) {                                 // update lazy[] for children
            lazy[2 * treeIndex + 1] += val;
            lazy[2 * treeIndex + 2] += val;
        }

        return;
    }

    ll mid = lo + (hi - lo) / 2;                        // recurse deeper for appropriate child

    updateLazySegTree(2 * treeIndex + 1, lo, mid, i, j, val);
    updateLazySegTree(2 * treeIndex + 2, mid + 1, hi, i, j, val);

    // merge updates
    tre[treeIndex] = tre[2 * treeIndex + 1] + tre[2 * treeIndex + 2];
}
// call this method as updateLazySegTree(0, 0, n-1, i, j, val);
// Here you want to update the range [i, j] with value val.

ll queryLazySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j){
    // query for arr[i..j]

    if (lo > j || hi < i)                               // segment completely outside range
        return 0;                                       // represents a null node

    if (lazy[treeIndex] != 0) {                         // this node is lazy
        tre[treeIndex] += (hi - lo + 1) * lazy[treeIndex]; // normalize current node by removing laziness

        if (lo != hi) {                                 // update lazy[] for children nodes
            lazy[2 * treeIndex + 1] += lazy[treeIndex];
            lazy[2 * treeIndex + 2] += lazy[treeIndex];
        }
        lazy[treeIndex] = 0;                          // current node processed. No longer lazy
    }

    if (i <= lo && j >= hi)                          // segment completely inside range
        return tre[treeIndex];

    ll mid = lo + (hi - lo) / 2;                     // partial overlap of current segment and queried range. Recurse deeper.
    if (i > mid)
        return queryLazySegTree(2 * treeIndex + 2, mid + 1, hi, i, j);
    else if (j <= mid)
        return queryLazySegTree(2 * treeIndex + 1, lo, mid, i, j);

    ll leftQuery = queryLazySegTree(2 * treeIndex + 1, lo, mid, i, mid);
    ll rightQuery = queryLazySegTree(2 * treeIndex + 2, mid + 1, hi, mid + 1, j);
    // merge query results
    return leftQuery + rightQuery;
}
// call this method as queryLazySegTree(0, 0, n-1, i, j);
// Here [i,j] is the range/interval you are querying.
// This method relies on "null" nodes being equivalent to storing zero.
```

SEGMENTED SIEVE:

```
vector<int> smallest_factor;
vector<bool> prime;
vector<int> primes;

void sieve(int maximum) {
    maximum = max(maximum, 2);
    smallest_factor.assign(maximum + 1, 0);
    prime.assign(maximum + 1, true);
    prime[0] = prime[1] = false;
    primes = {2};
    for (int p = 2; p <= maximum; p += 2) {
        prime[p] = p == 2;
        smallest_factor[p] = 2;
    }
    for (int p = 3; p * p <= maximum; p += 2)
        if (prime[p])
            for (int i = p * p; i <= maximum; i += 2 * p)
                if (prime[i]) {
                    prime[i] = false;
                    smallest_factor[i] = p;
                }
    for (int p = 3; p <= maximum; p += 2)
        if (prime[p]) {
            smallest_factor[p] = p;
            primes.push_back(p);
        }
}
vl segmentSieve (ll l, ll r) {
    vector<bool> isPrime(r-l+1,1);
    vl res;
    if (l==1) isPrime[0]=false;
    for (int i=0;primes[i]*primes[i]<=r;++i) {
        int p=primes[i];
        ll k = Ceil(l,p)*p;
        for (ll j=k;j<=r;j+=p) {
            isPrime[j-l]=0;
        }
        if(k==p) isPrime[k-l]=1;
    }
    for (int i=0;i<r-l+1;++i) {
        if(isPrime[i]) res.pb(i+l);
    }
    return res;
}
```

*******TRIE**********

```
const int  m=11;
ll Trie[N][m];
ll nnode;
bool isword[N];
void reset(int k){
    for(int i=0;i<m;i++){
        Trie[k][i]=-1;
    }
}
void Insert(string &s){
    int n=SZ(s),node=0;
    for(int i=0;i<n;i++){
        if(Trie[node][s[i]-'0']==-1){
            Trie[node][s[i]-'0']=++nnode;
            reset(nnode);
        }
        node=Trie[node][s[i]-'0'];
    }
    isword[node]=1;
}

string Search(string &s){
    // print(s);
    ll n=SZ(s),node=0;
    string res;
    for(int i=0;i<n;i++){
        pll temp={-1,-1};
        for(int j=0;j<10;j++){
            if(Trie[node][j]!=-1){
                if(temp.ff<((j+(s[i]-48))%10)){
                    temp={((j+s[i]-48)%10),j};
                }
            }
        }
        res.pb(temp.ss+48);
        node=Trie[node][temp.ss];
    }
    return res;
}
```

TRIE TO NUMBER OF DISTICT SUBSTR:

```
#define MAX_CHAR 26
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
class SuffixTrieNode
{
public:
    SuffixTrieNode *children[MAX_CHAR];
    SuffixTrieNode() // Constructor
    {
        // Initialize all child pointers as NULL
        for (int i = 0; i < MAX_CHAR; i++)
          children[i] = NULL;
    }

    // A recursive function to insert a suffix of the s
    // in subtree rooted with this node
    void insertSuffix(string suffix);
};
```

```
// A Trie of all suffixes
class SuffixTrie
{
    SuffixTrieNode *root;
    int _countNodesInTrie(SuffixTrieNode *);
public:
    // Constructor (Builds a trie of suffies of the given text)
    SuffixTrie(string s)
    {
        root = new SuffixTrieNode();

        // Consider all suffixes of given string and insert
        // them into the Suffix Trie using recursive function
        // insertSuffix() in SuffixTrieNode class
        for (int i = 0; i < s.length(); i++)
            root->insertSuffix(s.substr(i));
    }

    //  method to count total nodes in suffix trie
    int countNodesInTrie() { return _countNodesInTrie(root); }
};

// A recursive function to insert a suffix of the s in
// subtree rooted with this node
void SuffixTrieNode::insertSuffix(string s)
{
    // If string has more characters
    if (s.length() > 0)
    {
        // Find the first character and convert it
        // into 0-25 range.
        char cIndex = s.at(0) - 'a';

        // If there is no edge for this character,
        // add a new edge
        if (children[cIndex] == NULL)
            children[cIndex] = new SuffixTrieNode();

        // Recur for next suffix
        children[cIndex]->insertSuffix(s.substr(1));
    }
}

// A recursive function to count nodes in trie
int SuffixTrie::_countNodesInTrie(SuffixTrieNode* node)
{
    // If all characters of pattern have been processed,
    if (node == NULL)
        return 0;

    int count = 0;
    for (int i = 0; i < MAX_CHAR; i++)
    {
        // if children is not NULL then find count
        // of all nodes in this subtrie
        if (node->children[i] != NULL)
            count += _countNodesInTrie(node->children[i]);
    }

    // return count of nodes of subtrie and plus
    // 1 because of node's own count
    return (1 + count);
}

// Returns count of distinct substrings of str
ll countDistinctSubstring(string str)
{
    // Construct a Trie of all suffixes
    SuffixTrie sTrie(str);
    // Return count of nodes in Trie of Suffixes

    return sTrie.countNodesInTrie();

}
```

DYNAMIC PROGRAMMING***

0-1 knapsack:

```
int knapSack(int W, int wt[], int val[], int n)

{

    // making and initializing dp array

    int dp[W + 1];

    memset(dp, 0, sizeof(dp));


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

        for (int w = W; w >= 0; w--) {


            if (wt[i - 1] <= w)

                // finding the maximum value

                dp[w] = max(dp[w],

                            dp[w - wt[i - 1]] + val[i - 1]);

        }

    }

    return dp[W]; // returning the maximum value of knapsack

}
```

COIN change:

```
ll dp[][];

vl coins;


ll func(ll ind,ll amount){

    if(amount==0) return 1;

    if(ind<0) return 0;

    if(dp[ind][amount] !=-1) return dp[ind][amount];


    ll ways=0;

    for(ll coin_amount=0;coin_amount<=amount;coin_amount+=coins[ind]){

        ways+=func(ind-1,amount-coin_amount);

    }

    return dp[ind][amount]=ways;

}

ll coinChange(ll amount){

    memset(dp,-1,sizeof(dp));

    return func(coins.size()-1,amount);

}
```

Iterative solution of coin change:

```
ll t=0,n;

ll dp[10005];

ll coins[105];


ll func(ll amount){

    dp[0]=1;

    ll i,j;

    fo(i,n){

        for(j=coins[i];j<=amount;j++){

            dp[j]=(dp[j]+dp[j-coins[i]])%M;

        }

    }

    return dp[amount];

}
```

LCS:

```
string s1,s2;

int dp[1005][1005];

int lcs(int i,int j){

    if(i<0 || j<0) return 0;

    if(dp[i][j]!=-1) return dp[i][j];

    //remove 1 char from s1
```

```
   int ans=lcs(i-1,j);
   //remove 1 char from s2
   ans=max(ans,lcs(i,j-1));
   //remove 1 char from s1 and s2
   ans=max(ans,lcs(i-1,j-1))+(s1[i]==s2[j]);
   return dp[i][j]=ans;
}
```

Longest Increasing Subsequence:

```
int ar[N];
int dp[N];

int lis(int n){
   if(dp[n]!=-1) return dp[n];
   int ans=1;
   for(int i=0;i<n;i++){
      if(ar[n]>ar[i]){
         ans=max(ans,1+lis(i));
      }
   }
   return dp[n]=ans;
}//O(n^2)
```

DP right capital small partition:

```
ll dp[100000+5][5];
string s;
ll func(int i,bool flag=0){
   if(i<0) return 0;
   if(dp[i][flag]!=-1) return dp[i][flag];
   ll ans=INT_MAX,ans2=INT_MAX;
   if(!flag){
      if(s[i]>96){
         ans=1+func(i-1,1);
         ans2=func(i-1,0);
      }
      else{
         ans=1+func(i-1,0);
         ans2=func(i-1,1);
      }
   }
   else{
      if(s[i]>96) ans=1+func(i-1,1);
      else ans=func(i-1,1);
   }
   return dp[i][flag]= min(ans,ans2);
}
```

DP Rod cutting:

```
int dp[1000];
vector<int> prices;
int func(int len){
   if(len==0) return 0;
   if(dp[len]!=-1) return dp[len];
   int ans=0;
   for(int len_to_cut=1;len_to_cut<=prices.size();len_to_cut++){
      if(len-len_to_cut>=0){
         ans=max(ans,func(len-len_to_cut)+prices[len_to_cut-1]);
      }
   }
   return dp[len]=ans;
}
```

DP SUBSET SUM:

```
int dp[][];
vector<int> nums;
bool func(int i, int sum){
   if(sum==0) return true;
   if(i<0) return false;
   if(dp[i][sum]!=-1) return dp[i][sum];
   // not consider ith index
```

```
    int isPossible=func(i-1,sum);
    // consider ith index
    if(sum-nums[i]>=0) isPossible|=func(i-1,sum-nums[i]);
    return dp[i][sum]=isPossible;
}
```

DP SUBSET Count:

```
 int dp[105][35];
string str;
int func(string a,string b,int m,int n){
    if(n<0) return 1;
    if(m<0) return 0;
    if(dp[m][n]!=-1) return dp[m][n];

    if(a[m]==b[n]) return dp[m][n]=func(a,b,m-1,n-1)+ func(a,b,m-1,n);
    return dp[m][n]=func(a,b,m-1,n);
}
```

Find minimum number operations to convert str1 to str2:

```
int editDistDP(string str1, string str2, int m, int n)
{
    // Create a table to store results of subproblems
    int dp[m + 1][n + 1];

    // Fill d[][] in bottom up manner
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            // If first string is empty, only option is to
            // insert all characters of second string
            if (i == 0)
                dp[i][j] = j; // Min. operations = j
            // If second string is empty, only option is to
            // remove all characters of second string
            else if (j == 0)
                dp[i][j] = i; // Min. operations = i

            // If last characters are same, ignore last char
            // and recur for remaining string
            else if (str1[i - 1] == str2[j - 1])
                dp[i][j] = dp[i - 1][j - 1];


            // If the last character is different, consider
            // all possibilities and find the minimum
            else
                dp[i][j]
                    = 1
                      + min(dp[i][j - 1], // Insert
                            dp[i - 1][j], // Remove
                            dp[i - 1][j - 1]); // Replace
        }
    }

    return dp[m][n];
}
```

PROBLEMS:

Given a n*n matrix where all numbers are distinct,

find the maximum length path (starting from any cell) such that all cells along the path are in increasing order with a difference of 1.

```
// Returns length of the longest path beginning with
// mat[i][j]. This function mainly uses lookup table
// dp[n][n]
int findLongestFromACell(int i, int j, int mat[n][n],
                         int dp[n][n])
{
    if (i < 0 || i >= n || j < 0 || j >= n)
```

```
        return 0;

    // If this subproblem is already solved
    if (dp[i][j] != -1)
        return dp[i][j];

    // To store the path lengths in all the four directions
    int x = INT_MIN, y = INT_MIN, z = INT_MIN, w = INT_MIN;

    // Since all numbers are unique and in range from 1 to
    // n*n, there is atmost one possible direction from any
    // cell
    if (j < n - 1 && ((mat[i][j] + 1) == mat[i][j + 1]))
        x = 1 + findLongestFromACell(i, j + 1, mat, dp);

    if (j > 0 && (mat[i][j] + 1 == mat[i][j - 1]))
        y = 1 + findLongestFromACell(i, j - 1, mat, dp);

    if (i > 0 && (mat[i][j] + 1 == mat[i - 1][j]))
        z = 1 + findLongestFromACell(i - 1, j, mat, dp);

    if (i < n - 1 && (mat[i][j] + 1 == mat[i + 1][j]))
        w = 1 + findLongestFromACell(i + 1, j, mat, dp);

    // If none of the adjacent fours is one greater we will
    // take 1 otherwise we will pick maximum from all the
    // four directions
    return dp[i][j] = max({x, y, z, w, 1});
}

// Returns length of the longest path beginning with any
// cell
int finLongestOverAll(int mat[n][n])
{
    int result = 1; // Initialize result

    // Create a lookup table and fill all entries in it as
    // -1
    int dp[n][n];
    memset(dp, -1, sizeof dp);

    // Compute longest path beginning from all cells
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dp[i][j] == -1)
                findLongestFromACell(i, j, mat, dp);

            // Update result if needed
            result = max(result, dp[i][j]);
        }
    }

    return result;
}
```

****** DATA STRUCTURES ******

Paranthesis:

```
unordered_map<char,int> symbols ={{'(',-1},{'{',-2},{'[',-3},{')',1},{'}',2},{']',3}};

bool isBalanced(string s){
    stack<char> st;
    for(char bracket : s){
        if(symbols[bracket]<0){
            st.push(bracket);
        }else{
            if(st.empty()) return 0;
            char top=st.top();
            st.pop();
            if(symbols[top] + symbols[bracket] !=0){
```

```
            return 0;
          }
        }
      }
    if(st.empty()) return 1;
    return 0;
}
```

generate balanced paranthesis:

```
vector<string> valid;

void generate(string &s,int open,int close){
    if(open==0&&close==0){
        valid.push_back(s);
        return;
    }
    if(open>0){
        s.push_back('(');
        generate(s,open-1,close);
        s.pop_back();
    }
    if(close>0){
        if(open<close){
            s.push_back(')');
            generate(s,open,close-1);
            s.pop_back();
        }
    }
}
```

LINKED LIST:

```
class node{
    public:
    int data;
    node* next;

    node(int val){
        data=val;
        next=NULL;
    }
};

void insertAthead(node* &head, int val){
    node* n=new node(val);
    n->next=head;
    head=n;
}

void insertAtTail(node* &head,int val){
    node* n=new node(val);

    node* temp=head;

    if(head==NULL){
        head=n;
        return;
    }

    while(temp->next!=NULL){
        temp=temp->next;
    }
    temp->next=n;
}

bool search(node* head,int key){
    node* temp=head;
    while(temp!=NULL){
        if(temp->data==key){
            return true;
        }
      temp=temp->next;
    }
    return false;
}

void deleteAtHead(node* &head){


    node* todelete=head;
    head=head->next;

    delete todelete;
}

void deletion(node* &head, int val){
    if(head==NULL) return;

    if(head->next==NULL){
      deleteAtHead(head);
      return;
    }

    node* temp=head;
    while(temp->next->data!=val){
      temp=temp->next;
    }
    node* todelete= temp->next;
    temp->next=temp->next->next;

    delete todelete;
}


void display(node* head){
    node* temp=head;
    while(temp!=NULL){
        cout<<temp->data<<' ';
        temp=temp->next;
    }
    cout<<endl;
}
```

*********GAME THEORY & EXTRAS*********

ALPHA BETA PRUNING:

```
// Returns optimal value for
// current player(Initially called
// for root and maximizer)
int minimax(int depth, int nodeIndex,bool maximizingPlayer,vi values, int alpha,int beta)
{

  // Terminating condition. i.e
  // leaf node is reached
  if (depth == 3)
    return values[nodeIndex];

  if (maximizingPlayer)
  {
    int best = MIN;

    // Recur for left and
    // right children
    for (int i = 0; i < 2; i++)
    {

      int val = minimax(depth + 1, nodeIndex * 2 + i,false, values, alpha, beta);
      best = max(best, val);
      alpha = max(alpha, best);
      // Alpha Beta Pruning
      if (beta <= alpha)
        break;
    }
    return best;
  }
  else
  {
    int best = MAX;

    // Recur for left and
    // right children
    for (int i = 0; i < 2; i++)
    {
      int val = minimax(depth + 1, nodeIndex * 2 + i,true, values, alpha, beta);
      best = min(best, val);
      beta = min(beta, best);

      // Alpha Beta Pruning
      if (beta <= alpha)
        break;
    }
    return best;
  }
}
```

CRT:

```
/**
  A CRT solver which works even when moduli are not pairwise coprime
  1. Add equations using addEquation() method
  2. Call solve() to get {x, N} pair, where x is the unique solution modulo N.
  Assumptions:
    1. LCM of all mods will fit into long long.
*/
class CRT {
  typedef pair<ll,ll> pll;

  /** CRT Equations stored as pairs of vector. See addEqation()*/
  vector<pll> equations;

public:
  void clear() {
    equations.clear();
  }

  /** Add equation of the form x = r (mod m)*/
  void addEquation( ll r, ll m ) {
    equations.push_back({r, m});
  }
  pll solve() {
    if (equations.size() == 0) return {-1,-1}; /// No equations to solve

    ll a1 = equations[0].first;
    ll m1 = equations[0].second;
    a1 %= m1;
    /** Initially x = a_0 (mod m_0)*/

    /** Merge the solution with remaining equations */
    for ( int i = 1; i < equations.size(); i++ ) {
      ll a2 = equations[i].first;
      ll m2 = equations[i].second;

      ll g = __gcd(m1, m2);
      if ( a1 % g != a2 % g ) return {-1,-1}; /// Conflict in equations

      /** Merge the two equations*/
      ll p, q;
```

```
        euclide<ll>(m1/g, m2/g, p, q);

        ll mod = m1 / g * m2;
        ll x = ( (__int128)a1 * (m2/g) % mod *q % mod +
(__int128)a2 * (m1/g) % mod * p % mod ) % mod;

        /** Merged equation*/
        a1 = x;
        if ( a1 < 0 ) a1 += mod;
        m1 = mod;
      }
      return {a1, m1};
   }
};
```

CHESS MOVES:

```
vpii king={ {1,0},{0,1},{-1,0},{0,-1},{1,-1},{-1,1},{1,1},{-1,-1} };

vpii rook={ {1,0},{2,0},{3,0},{4,0},{5,0},{6,0},{7,0},
        {0,1},{0,2},{0,3},{0,4},{0,5},{0,6},{0,7},
        {-1,0},{-2,0},{-3,0},{-4,0},{-5,0},{-6,0},{-7,0},
        {0,-1},{0,-2},{0,-3},{0,-4},{0,-5},{0,-6},{0,-7} };

vpii bishop={ {1,1},{2,2},{3,3},{4,4},{5,5},{6,6},{7,7},
         {-1,1},{-2,2},{-3,3},{-4,4},{-5,5},{-6,6},{-7,7},
         {1,-1},{2,-2},{3,-3},{4,-4},{5,-5},{6,-6},{7,-7},
         {-1,-1},{-2,-2},{-3,-3},{-4,-4},{-5,-5},{-6,-6},{-7,-7} };

vpii queen={ {1,0},{2,0},{3,0},{4,0},{5,0},{6,0},{7,0},
        {0,1},{0,2},{0,3},{0,4},{0,5},{0,6},{0,7},
        {-1,0},{-2,0},{-3,0},{-4,0},{-5,0},{-6,0},{-7,0},
        {0,-1},{0,-2},{0,-3},{0,-4},{0,-5},{0,-6},{0,-7},
        {1,1},{2,2},{3,3},{4,4},{5,5},{6,6},{7,7},
        {-1,1},{-2,2},{-3,3},{-4,4},{-5,5},{-6,6},{-7,7},
        {1,-1},{2,-2},{3,-3},{4,-4},{5,-5},{6,-6},{7,-7},
        {-1,-1},{-2,-2},{-3,-3},{-4,-4},{-5,-5},{-6,-6},{-7,-7} };

vpii knight={ {1,2},{1,-2},{-1,2},{-1,-2},{2,1},{2,-1},{-2,1},{-2,-1} };
```

***GRUNDY

```
/* Game Description-
"A game is played between two players and there are N piles
of stones such that each pile has certain number of stones.
On his/her turn, a player selects a pile and can take any
non-zero number of stones upto 3 (i.e- 1,2,3)
The player who cannot move is considered to lose the game
(i.e., one who take the last stone is the winner).
Can you find which player wins the game if both players play
optimally (they don't make any mistake)? "

A Dynamic Programming approach to calculate Grundy Number
and Mex and find the Winner using Sprague - Grundy Theorem. */


/* piles[] -> Array having the initial count of stones/coins
in each piles before the game has started.
n-> Number of piles

Grundy[] -> Array having the Grundy Number corresponding to
the initial position of each piles in the game

The piles[] and Grundy[] are having 0-based indexing*/

ll dp[N];

// A Function to calculate Mex of all the values in that set
ll calculateMex(unordered_set<ll> Set){
  ll Mex = 0;
  while (Set.find(Mex) != Set.end()) Mex++;
  return Mex;
}
```

```
// A function to Compute Grundy Number of 'n'
ll calculateGrundy(ll n){
   if(n<=3) return dp[n]=n;
   ll &ret=dp[n];
   if(ret!=-1) return ret;
   unordered_set<ll> Set; // A Hash Table

   for (ll i=1; i<=3; i++) Set.insert (calculateGrundy (n-i));
   // Store the result
   return ret = calculateMex (Set);
}

// A function to declare the winner of the game
bool declareWinner(vl &piles, ll n){
   ll xorValue = dp[piles[0]];
   for(int i=1; i<=n-1; i++) xorValue = xorValue ^ dp[piles[i]];
   return (xorValue!=0);
}
/*
 Game Description-
 The game starts with a number- 'n' and the player to move
 divides the number- 'n' with the primes- 2, 3, 6 and then
 takes the floor. If the integer becomes 0, it is removed.
 The last player to move wins. Which player wins the game?

 A Dynamic Programming (Memoization-based) approach to
 calculate Grundy Number and Mex
*/

ll dp[N];

// A Function to calculate Mex of all the values in that set
// This function remains same
ll calculateMex (unordered_set<ll> &st){
   ll Mex = 0;
   while(st.count(Mex)) Mex++;
   return (Mex);
}

// A function to Compute Grundy Number of 'n'
// Only this function varies according to the game
ll calculateGrundy (ll n){
   if (n == 0) return 0;
   ll &ret=dp[n];
   if(ret!=-1) return ret;
   unordered_set<ll> Set; // A Hash Table
   Set.insert (calculateGrundy (n/2));
   Set.insert (calculateGrundy (n/3));
   Set.insert (calculateGrundy (n/6));
   // Store the result
   return ret = calculateMex (Set);
}
/*  A Dynamic Programming (Memoization-based) approach to
 calculate Grundy Number of a Game
 Game Description-
 Just like a one-pile version of Nim, the game starts with
 a pile of n stones, and the player to move may take any
 positive number of stones.
 The last player to move wins. Which player wins the game? */

ll dp[N];

// A Function to calculate Mex of all the values in that set
// This function remains same
ll calculateMex(unordered_set<ll>& st){
   ll mex = 0;
   while (st.count(mex)) mex++;
   return (mex);
}
```

```
// A function to Compute Grundy Number of 'n'
// Only this function varies according to the game
ll calculateGrundy(ll n){
   if(n==0) return 0;

   ll &ret=dp[n];

   if(ret!=-1) return ret;

   unordered_set<ll> st;

   for (ll i=0;i<=n-1;i++) st.insert(calculateGrundy(i));

   return ret=calculateMex (st);
}
/* A  C++ program to find Grundy Number for
  a game which is one-pile version of Nim.
 Game Description : The game starts with a pile of
 n stones, and the player to move may take any
 positive number of stones upto k only.
 The last player to move wins. */


// A function to Compute Grundy Number of 'n'
// Only this function varies according to the game
int calculateGrundy (int n,int k){
   if(n==0) return 0;
   if(n==1) return 1;
   if(n==2) return 2;
   if(n==3) return 3;
   return (n%(k+1));
}
```