TABLE OF CONTENTS

PRECODE

```
//#pragma comment(linker, "/stack:200000000")
//#pragma GCC optimize("Ofast")
//#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
//#pragma GCC optimize("unroll-loops")

#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

#define ll long long
#define ull unsigned long long
#define ld long double
#define pii pair<int,int>
#define pll pair<ll,ll>
#define vi vector<int>
#define vll vector<ll>
#define vc vector<char>
#define vs vector<string>
#define vpll vector<pll>
#define vpii vector<pii>
#define umap unordered_map
#define uset unordered_set
#define PQ priority_queue
#define printa(a,L,R) for(int i=L;i<R;i++) cout<<a[i]<<(i==R-1?'\n':' ')
#define printv(a) printa(a,0,a.size())
#define print2d(a,r,c) for(int i=0;i<r;i++) for(int j=0;j<c;j++) cout<<a[i][j]<<(j==c-1?'\n':' ')
#define pb push_back
#define eb emplace_back
#define mt make_tuple
#define fbo find_by_order
#define ook order_of_key
#define MP make_pair
#define UB upper_bound
#define LB lower_bound
#define SQ(x) ((x)*(x))
#define issq(x) ((((ll)(sqrt((x))))*((ll)(sqrt((x))))==(x))
#define F first
#define S second
#define mem(a,x) memset(a,x,sizeof(a))
#define inf 1e18
#define E 2.71828182845904523536
#define gamma 0.5772156649
#define nl "\n"
#define lg(r,n) (int)(log2(n)/log2(r))
#define pf printf
#define sf scanf
#define sf1(a)          scanf("%d",&a)
#define sf2(a,b)        scanf("%d %d",&a,&b)
#define sf3(a,b,c)      scanf("%d %d %d",&a,&b,&c)
#define pf1(a)          printf("%d\n",a);
#define pf2(a,b)        printf("%d %d\n",a,b)
```

```
#define pf3(a,b,c)          printf("%d %d %d\n",a,b,c)
#define sf1ll(a)            scanf("%lld",&a)
#define sf2ll(a,b)          scanf("%I64d %I64d",&a,&b)
#define sf3ll(a,b,c)        scanf("%I64d %I64d %I64d",&a,&b,&c)
#define pf1ll(a)            printf("%lld\n",a);
#define pf2ll(a,b)          printf("%I64d %I64d\n",a,b)
#define pf3ll(a,b,c)        printf("%I64d %I64d %I64d\n",a,b,c)
#define _ccase printf("Case %lld: ",++cs)
#define _case cout<<"Case "<<++cs<<": "
#define by(x) [](const auto& a, const auto& b) { return a.x < b.x; }

#define asche cerr<<"Ekhane asche\n";
#define rev(v) reverse(v.begin(),v.end())
#define srt(v) sort(v.begin(),v.end())
#define grtsrt(v) sort(v.begin(),v.end(),greater<ll>())
#define all(v) v.begin(),v.end()
#define mnv(v) *min_element(v.begin(),v.end())
#define mxv(v) *max_element(v.begin(),v.end())
#define toint(a) atoi(a.c_str())
#define BeatMeScanf ios_base::sync_with_stdio(false)
#define valid(tx,ty) (tx>=0&&tx<n&&ty>=0&&ty<m)
#define one(x) __builtin_popcount(x)
#define Unique(v) v.erase(unique(all(v)),v.end())
#define stree l=(n<<1),r=l+1,mid=b+(e-b)/2
#define fout(x) fixed<<setprecision(x)
string tostr(int n) {stringstream rr;rr<<n;return rr.str();}
inline void yes(){cout<<"YES\n";exit(0);}
inline void no(){cout<<"NO\n";exit(0);}
```

```
template <typename T> using o_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
//ll dx[]={1,0,-1,0,1,-1,-1,1};
//ll dy[]={0,1,0,-1,1,1,-1,-1};
//random_device rd;
//mt19937 random(rd());
int sc()
{
   register int c = getchar();
   register int x = 0;
   int neg = 0;
   for(;((c<48 || c>57) && c != '-');c = getchar());
   if(c=='-') {neg=1;c=getchar();}
   for(;c>47 && c<58;c = getchar()) {x = (x<<1) + (x<<3) + c - 48;}
   if(neg) x=-x;
   return x;
}
#define debug(args...) { string _s = #args; replace(_s.begin(), _s.end(),
',', ' '); stringstream _ss(_s); istream_iterator<string> _it(_ss); deb(_it,
args); }
void deb(istream_iterator<string> it) {}
template<typename T, typename... Args>
void deb(istream_iterator<string> it, T a, Args... args) {
   cerr << *it << " = " << a << endl;
   deb(++it, args...);
}

const int mod=1e9+7;
const int N=3e5+9;
```

```
const ld eps=1e-9;
const ld PI=acos(-1.0);
//ll gcd(ll a,ll b){while(b){ll x=a%b;a=b;b=x;}return a;}
//ll lcm(ll a,ll b){return a/gcd(a,b)*b;}
//ll qpow(ll n,ll k) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}



int main()
{
   BeatMeScanf;
   ll i,j,k,n,m;

   return 0;
}
```

THE ART OF DEBUGGING

- ✓ Long Longs?
- ✓ Check if m,n aren't misused
- ✓ Printed enough new line or extra new line?
- ✓ Make sure output format is right(includeing YES/NO vs Yes/No or newline vs spaces
- ✓ Run with n=1
- ✓ Have you cleared the vectors ?
- ✓ Make sure two ints aren't multiplied to get a long long
- ✓ Output enough digits after decimal point
- ✓ The exact constraints
- ✓ Check overflow(ll vs ints)
- ✓ Check all array bounds
- ✓ When using multiple dfs recursions check if inside one dfs another dfs is not called
- ✓ Case number print?
- ✓ Are you using the correct mod value?
- ✓ I spent a lot of my time debugging my solution without any success, after the contest I discovered that the obstacles in the input is 'x' (small one) while I was thinking it was 'X' (capital), I lost a bronze medal because of it :(
  -kingofnumbers (a Codeforces id)
- ✓ Set or multiset?
- ✓ Different Variables with same name ?
- ✓ Inside 2d loop are you using i++ instead of j++?
- ✓ Are you using ceil function? Then remove it!
- ✓ Is inf large enough?
- ✓ For multiple queries are you returning 0 inside the queries?
- ✓ For max and min have you initialized the values by a good enough value?
- ✓ Using the local variable of the same name when global variable was required to be used.
- ✓ declared a counter of type char instead of int ,resulted in passing of pretests and failing of system test. :)
- ✓ I subtracted 1 in a for loop from v.size(). Guess what happened when the input vector empty?
- ✓ for (int i = n - 1; i--; i >= 0)
  instead of:
  for (int i = n - 1; i >= 0; i--)

It passed pretests and failed systests

- ✓ in 2d grid for valid(x,y) check if x>=0&&x<n hobe or x>=1&&x<=n hobe. Again x,y duitai ki n,m er sathe compare hobe naki n,n er sathe hobe
- ✓ Are you using memset correctly?
- ✓ Use bool operators using brackets.Beware!!!
- ✓ Have you deleted debug(x) lines?
- ✓ 1 is not a prime number
- ✓ It may be scanf("%d" , x). where &x is missing.
- ✓ Is Y vowel or consonant?
- ✓ Instead of printing NO printed N0.(with a zero).

LOOKUP DATA

Primes Under N

Power of 10	Number of Primes
1	4
2	25
3	168
4	1,229
5	9,592
6	78,498
7	664,579
8	5,761,455
9	50,847,534
10	455,052,511
11	4,118,054,813
12	37,607,912,018
13	346,065,536,839
14	3,204,941,750,802
15	29,844,570,422,669
16	279,238,341,033,925
17	2,623,557,157,654,233
18	24,739,954,287,740,860

NUMBER THEORY

1. Combinatorics

Notes

- $\sum_{0 \le k \le n} \binom{n-k}{k} = \text{Fib}_{n+1}$
- Number of different binary sequences of length n such that no two 0's are adjacent=Fib_{n+1}
- Combination with repetition: Let's say we choose k elements from an n-element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is: $\binom{n+k-1}{k}$
- Number of ways to divide n different persons in n/k equal groups i.e. each having size k is $\binom{n-1}{k-1}$
- The number non-negative solution of the equation

  $x_1+x_2+x_3+\ldots+x_k=n$ is $\binom{n+k-1}{n}$
- Number of binary sequence of length n and with k '1' is $\binom{n}{k}$

- The number of ordered pairs (a, b) of binary sequences of length n, such that the distance between them is k, can be calculated as follows: $\binom{n}{k} \cdot 2^n$.
  The distance between a and b is the number of components that differs in a and b — for example, the distance between (0, 0, 1, 0) and (1, 0, 1, 1) is 2).

Catalan numbers

- ✓ $C_n = \frac{1}{n+1}\binom{2n}{n}$
- ✓ $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- ✓ 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786
- ✓ Number of correct bracket sequence consisting of n opening and n closing brackets.
- ✓ The number of ways to completely parenthesize n+1 factors.
- ✓ The number of triangulations of a convex polygon with +2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- ✓ The number of ways to connect the 2n points on a circle to form n disjoint i.e. non-intersecting chords.
- ✓ The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size n×n, which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
- ✓ Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index i<j<k, such that $a_k < a_i < a_j$).
- ✓ The number of non-crossing partitions of a set of n elements.
- ✓ The number of rooted full binary trees with n+1 leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- ✓ The number of Dyck words of length 2n. A Dyck word is a string consisting of n X's and n Y's such that no initial segment of the string has more Y's than X's For example, the following are the Dyck words of length 6: XXXYYY XYXXYY XYXYXY XXYYXY XXYXYY.
- ✓ The number of different ways a convex polygon with n + 2 sides can be cut into triangles by connecting vertices with straight lines (a form of Polygon triangulation)
- ✓ Number of permutations of {1, …, n} that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing subsequence. For n = 3, these permutations are 132, 213, 231, 312 and 321. For n = 4, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321
- ✓ Number of ways to tile a stairstep shape of height n with n rectangles.

Narayana numbers

- ✓ N(n,k) = $\frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- ✓ The number of expressions containing *n* pairs of parentheses, which are correctly matched and which contain *k* distinct nestings. For instance, *N*(4, 2) = 6 as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '()':

  ()((())) (())(()) (()(())) ((()())) ((())()) ((()))()

- ✓ The number of paths from (0, 0) to (2*n*, 0), with steps only northeast and southeast, not straying below the *x*-axis, with *k* peaks. And sum of all number of peaks is Catalan number.

Stirling numbers of the first kind

- ✓ The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
- ✓ S(n,k) counts the number of permutations of n elements with k disjoint cycles.
- ✓ $S(n,k) = (n-1) * S(n-1,k) + S(n-1,k-1),$
  $where\ S(0,0) = 1, S(n,0) = S(0,n) = 0$
- ✓ $\sum_{k=0}^{n} S(n,k) = n!$

Stirling numbers of the second kind

- ✓ Stirling number of the second kind is the number of ways to partition a set of *n* objects into *k* non-empty subsets.
- ✓ $S(n,k) = k * S(n-1,k) + S(n-1,k-1),$
  $where\ S(0,0) = 1, S(n,0) = S(0,n) = 0$
- ✓ S(n,2)=2^{n-1}-1

Bell number

- ✓ Counts the number of partitions of a set.
- ✓ $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} * B_k$
- ✓ $B_n = \sum_{k=0}^{n} S(n,k)$,where S(n,k) is stirling number of second kind.
- ✓ The number of multiplicative partitions of a squarefree number with i prime factors is the i-th Bell number, B_i.
- ✓ If a deck of *n* cards is shuffled by repeatedly removing the top card and reinserting it anywhere in the deck (including its original position at the top of the deck), with exactly *n* repetitions of this operation, then there are n^n different shuffles that can be performed. Of these, the number that return the deck to its original sorted order is exactly B_n. Thus, the probability that the deck is in its original order after shuffling it in this way is B_n/n^n.

Lucas Theorem

- ✓ If p is prime the $\binom{p^a}{k} \equiv 0\ (mod\ p)$
- ✓ For non-negative integers m and n and a prime p, the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

$where,$
$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$
and
$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$
are the base *p* expansions of *m* and *n* respectively. This uses the convention that $\binom{m}{n} = 0, when\ m < n.$

Derangement

- ✓ A derangement is a permutation of the elements of a set, such that no element appears in its original position.
- ✓ $d(n) = (n-1) * \big(d(n-1) + d(n-2)\big),$
  $where\ d(0) = 1, d(1) = 0$
- ✓ $d(n) = \left\lfloor \frac{n!}{e} \right\rfloor, n \geq 1$

2. Burnside Lemma

The task is to count the number of different necklaces from n beads, each of which can be painted in one of the k colors. When comparing two necklaces, they can be rotated, but not reversed (i.e. a cyclic shift is permitted).
Solution:

$$ans = \frac{1}{n} \sum_{i=1}^{n} k^{\gcd(i,n)} = \frac{1}{n} \sum_{d|n} \phi\left(\frac{n}{d}\right) k^d$$

3. Number of Solutions of a Equation

- Number of solutions of
  $$x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \geq 0$$
  is: $\binom{n+k-1}{k-1}$
- Number of solutions of
  $$x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \geq a_i$$
  is: $\binom{n-\sum_{i=1}^{k} a_i + k - 1}{k-1}$
- Number of solutions of
  $$x_1 + x_2 + x_3 + \cdots + x_k = n, x_i \leq a_i$$
  is:

Problem : Codeforces 451E

```
ll f[25];
ll ncr(ll n,ll k)
{
   if(k>n) return 0;
   ll ans=1;
   k=min(n-k,k);
   for(ll i=n-k+1;i<=n;i++) ans=(ans*(i%mod))%mod;
   ans=ans*qpow(f[k],mod-2)%mod;
   return ans;
}
ll a[25];
int main()
{
   fast;
   ll i,j,k,n,m,s;
   cin>>n>>s;//n elements // sum is s
   for(i=0;i<n;i++) cin>>a[i];
```

```
    f[0]=1;
    for(i=1;i<25;i++) f[i]=i*f[i-1]%mod;
    ll ans=0;
    for(i=0;i<(1<<n);i++){
       ll sum=s,cnt=0;
       for(j=0;j<n;j++) if((i>>j)&1){
          sum-=a[j]+1;
          cnt++;
       }
       ll res=ncr(sum+n-1,n-1);
       if(cnt%2==1) res*=-1;
       ans=(ans+res)%mod;
       ans=(ans+mod)%mod;
    }
    cout<<ans<<nl;
    return 0;
}
```

- Number of solutions of

$$x_1 + x_2 + x_3 + \cdots + x_k = n,$$
$$1 \le x_i \le mx, all\ x_i\ are\ distinct$$

is:

Problem : Codeforces 403D

```
ll f[50];
int dp[N][N][50];
//number of solutions of equation c1+c2+c3+...c(len)=sum
//such that c1<c2<c3<...<c(len) ans 1<=c(i)<=mx
int yo(int sum,int mx,int len)
{
    if(len==0) return 1;
    if(sum<=0) return 0;
    if(mx<=0) return 0;
    int &ret=dp[sum][mx][len];
    if(ret!=-1) return ret;
    ll ans=0;
    if(sum>=mx) ans+=yo(sum-mx,mx-1,len-1);
    ans+=yo(sum-1,mx,len);
    ans+=yo(sum,mx-1,len);
    ans-=yo(sum-1,mx-1,len);
    if(ans<0) ans+=mod;
    ans%=mod;
    return ret=ans;
}
int main()
{
    fast;
    int i,j,k,n,m,t;
    f[0]=1;
    for(i=1;i<50;i++) f[i]=1LL*i*f[i-1]%mod;
    mem(dp,-1);
    cin>>n>>k;//mx=k
    ll ans=yo(n,n,k);
    ans=(ans*f[k])%mod;
    cout<<ans<<nl;
    return 0;
}
```

4. Expected value

Mathematically, for a discrete variable X with probability function P(X), the expected value E[X] is given by $\Sigma\, x_i P(x_i)$ the summation runs over all the distinct values x_i that the variable can take.
The rule of "linearity of of the expectation" says that E[x1+x2] = E[x1] + E[x2].

- the expected number of coin flips for getting N consecutive heads is $(2^{N+1} - 2)$.
- A fair coin flip experiment is carried out N times. The expected number of heads is N/2.
- The expected number of coin flips to ensure that there are atleast N heads in 2N.
- Bernoulli trial is a random experiment with exactly two possible outcomes, "success" and "failure".
- If the probability of success of a bernaulli trial is p then the expected number of trials to get a success is 1/p.
- If probability of success in a bernaulli trial is p, then the expected number of trials to guaranttee N successes is N/p.
- A random variable corresponding to a binomial is denoted by B(n,k) and is said to have a *binomial distribution*. If the probability of success is p and failure is q then the probability of exactly k successes in the experiment B(n,k) is given by:

$$B(n,k) = \binom{n}{k} p^k q^{n-k}$$

5. Prime Factorization

```
// Integer factorization in O(N^{1/4}
// uses squfof from msieve https://github.com/radii/msieve
// with fixes to work for n = p^3
// works up to 10^18
// probably fails on 5003^5 which is ~10^{18.5}

namespace NT{
  template<typename T>
  struct bigger_type{};
  template<typename T> using bigger_type_t = typename bigger_type<T>::type;
  template<> struct bigger_type<int>{using type = long long;};
  template<> struct bigger_type<unsigned int>{using type = unsigned long long;};
  //template<> struct bigger_type<int64_t>{using type = __int128;};
  //template<> struct bigger_type<uint64_t>{using type = unsigned __int128;};

  template<typename int_t = unsigned long long>
  struct Mod_Int{
    static inline int_t add(int_t const&a, int_t const&b, int_t const&mod){
      int_t ret = a+b;
      if(ret>=mod) ret-=mod;
      return ret;
    }
    static inline int_t sub(int_t const&a, int_t const&b, int_t const&mod){
      return add(a, mod-b);
    }
```

```
    static inline int_t mul(int_t const&a, int_t const&b, int_t const&mod){
        uint64_t ret = a * (uint64_t)b - (uint64_t)((long double)a * b / mod - 1.1) * mod;
                        if(-ret < ret){
                                ret = mod-1-(-ret-1)%mod;
                        } else {
                                ret%=mod;
                        }

                        //ret = min(ret, ret+mod);
                        int64_t out = ret;
                        /*if(out != a*(__int128) b % mod){
                                cerr << (long double)a * b / mod << " " << (uint64_t)((long double)a * b / mod - 0.1) << "\n";
                                cerr << mod << " " << ret << " " << ret+mod << " " << out << " " << (int64_t)(a*(__int128) b % mod) << "\n";
                                assert(0);
                        }*/
                        return out;
        //return a*static_cast<bigger_type_t<int_t>>(b)%mod;
    }
    static inline int_t pow(int_t const&a, int_t const&b, int_t const&mod){
        int_t ret = 1;
        int_t base = a;
        for(int i=0;b>>i;++i){
            if((b>>i)&1) ret = mul(ret, base, mod);
            base = mul(base, base, mod);
        }
        return ret;
    }
};

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type is_prime(T x){
    if(x<T(4)) return x>T(1);
    for(T i=2;i*i<=x;++i){
        if(x%i == 0) return false;
    }
    return true;
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type miller_rabin_single(T const&x, T base){
    if(x<T(4)) return x>T(1);
    if(x%2 == 0) return false;
    base%=x;
    if(base == 0) return true;

    T xm1 = x-1;
    int j = 1;
    T d = xm1/2;
    while(d%2 == 0){ // could use __builtin_ctz
        d/=2;
```

```
        ++j;
    }
    T t = Mod_Int<T>::pow(base, d, x);
    if(t==T(1) || t==T(xm1)) return true;
    for(int k=1;k<j;++k){
        t = Mod_Int<T>::mul(t, t, x);
        if(t == xm1) return true;
        if(t<=1) break;
    }
    return false;
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type miller_rabin_multi(T const&){return true;}
template<typename T, typename... S>
typename enable_if<is_integral<T>::value, bool>::type miller_rabin_multi(T const&x, T const&base, S const&...bases){
    if(!miller_rabin_single(x, base)) return false;
    return miller_rabin_multi(x, bases...);
}

template<typename T>
typename enable_if<is_integral<T>::value, bool>::type miller_rabin(T const&x){
    if(x < 316349281) return miller_rabin_multi(x, T(11000544), T(31481107));
    if(x < 4759123141ull) return miller_rabin_multi(x, T(2), T(7), T(61));
    return miller_rabin_multi(x, T(2), T(325), T(9375), T(28178), T(450775), T(9780504), T(1795265022));
}

template<typename T>
typename enable_if<is_integral<T>::value, T>::type isqrt(T const&x){
    assert(x>=T(0));
    T ret = static_cast<T>(sqrtl(x));
    while(ret>0 && ret*ret>x) --ret;
    while(x-ret*ret>2*ret)
        ++ret;
    return ret;
}
template<typename T>
typename enable_if<is_integral<T>::value, T>::type icbrt(T const&x){
    assert(x>=T(0));
    T ret = static_cast<T>(cbrt(x));
    while(ret>0 && ret*ret*ret>x) --ret;
    while(x-ret*ret*ret>3*ret*(ret+1))
        ++ret;
    return ret;
}
/*uint64_t isqrt(unsigned __int128 const&x){
    unsigned __int128 ret = sqrtl(x);
    while(ret>0 && ret*ret>x) --ret;
    while(x-ret*ret>2*ret)
        ++ret;
```

```
    return ret;
  }*/
  vector<uint16_t> saved;
  // fast prime factorization from
  // https://github.com/radii/msieve
  uint64_t squfof_iter_better(uint64_t const&x, uint64_t const&k,
uint64_t const&it_max, uint32_t cutoff_div){
    if(__gcd((uint64_t)k, x)!=1) return __gcd((uint64_t)k, x);
    //cerr << "try: " << x << " " << k << "\n";
    saved.clear();
    uint64_t scaledn = k*x;
    if(scaledn>>62) return 1;
    uint32_t sqrtn = isqrt(scaledn);
    uint32_t cutoff = isqrt(2*sqrtn)/cutoff_div;
    uint32_t q0 = 1;
    uint32_t p1 = sqrtn;
    uint32_t q1 = scaledn-p1*p1;

    if(q1 == 0){
      uint64_t factor = __gcd(x, (uint64_t)p1);
      return factor==x ? 1:factor;
    }

    uint32_t multiplier = 2*k;
    uint32_t coarse_cutoff = cutoff * multiplier;
    //cerr << "at: " << multiplier << "\n";

    uint32_t i, j;
    uint32_t p0 = 0;
    uint32_t sqrtq = 0;

    for(i=0;i<it_max;++i){
      uint32_t q, bits, tmp;

      tmp = sqrtn + p1 - q1;
      q = 1;
      if (tmp >= q1)
        q += tmp / q1;

      p0 = q * q1 - p1;
      q0 = q0 + (p1 - p0) * q;

      if (q1 < coarse_cutoff) {
        tmp = q1 / __gcd(q1, multiplier);

        if (tmp < cutoff) {
          saved.push_back((uint16_t)tmp);
        }
      }

      bits = 0;
      tmp = q0;
      while(!(tmp & 1)) {
        bits++;
        tmp >>= 1;
      }
      if (!(bits & 1) && ((tmp & 7) == 1)) {
```

```
      sqrtq = (uint32_t)isqrt(q0);

      if (sqrtq * sqrtq == q0) {
         for(j=0;j<saved.size();++j){
            if(saved[j] == sqrtq) break;
         }
         if(j == saved.size()) break;
         //else cerr << "skip " << i << "\n";;
      }
   }
   tmp = sqrtn + p0 - q0;
   q = 1;
   if (tmp >= q0)
      q += tmp / q0;

   p1 = q * q0 - p0;
   q1 = q1 + (p0 - p1) * q;

   if (q0 < coarse_cutoff) {
      tmp = q0 / __gcd(q0, multiplier);

      if (tmp < cutoff) {
         saved.push_back((uint16_t) tmp);
      }
   }
}

if(sqrtq == 1) { return 1;}
if(i == it_max) { return 1;}
```

```
q0 = sqrtq;
p1 = p0 + sqrtq * ((sqrtn - p0) / sqrtq);
q1 = (scaledn - (uint64_t)p1 * (uint64_t)p1) / (uint64_t)q0;

for(j=0;j<it_max;++j) {
   uint32_t q, tmp;

   tmp = sqrtn + p1 - q1;
   q = 1;
   if (tmp >= q1)
      q += tmp / q1;

   p0 = q * q1 - p1;
   q0 = q0 + (p1 - p0) * q;

   if (p0 == p1) {
      q0 = q1;
      break;
   }

   tmp = sqrtn + p0 - q0;
   q = 1;
   if (tmp >= q0)
      q += tmp / q0;

   p1 = q * q0 - p0;
   q1 = q1 + (p0 - p1) * q;
```

```
        if (p0 == p1)
            break;
      }
      if(j==it_max) {cerr << "RNG\n"; return 1;} // random fail

      uint64_t factor = __gcd((uint64_t)q0, x);
      if(factor == x) factor=1;
      return factor;
   }
   uint64_t squfof(uint64_t const&x){
//for using only squfof don't comment the following lines.
//for factorizing comment these, no problem.
//       for(uint64_t i=2;i<=min((int64_t)5000,(int64_t)x/2-1);i++){
//          uint64_t p=(uint64_t)((1.0*x)/(i*1.0)+eps);
//          if(p*i==x) return i;
//       }
      static array<uint32_t, 16> multipliers{1, 3, 5, 7, 11, 3*5, 3*7,
3*11, 5*7, 5*11, 7*11, 3*5*7, 3*5*11, 3*7*11, 5*7*11, 3*5*7*11};

      uint64_t cbrt_x = icbrt(x);
      if(cbrt_x*cbrt_x*cbrt_x == x) return cbrt_x;

      //uint32_t iter_lim = isqrt(isqrt(x))+10;
      uint32_t iter_lim = 300;
      for(uint32_t iter_fact = 1;iter_fact<20000;iter_fact*=4){
         for(uint32_t const&k : multipliers){
            if(numeric_limits<uint64_t>::max()/k<=x) continue; //would
overflow
            uint32_t const it_max = iter_fact*iter_lim;
            uint64_t factor = squfof_iter_better(x, k, it_max, 1);
            if(factor==1 || factor==x) continue;
            return factor;
         }
      }
      cerr << "failed to factor: " << x << "\n";
      assert(0);
      assert(0);
      return 1;
   }

   template<typename T>
   typename      enable_if<is_integral<T>::value,      vector<T>>::type
factorize_brute(T x){
      vector<T> ret;
      while(x%2 == 0){
         x/=2;
         ret.push_back(2);
      }
      for(uint32_t i=3;i*(T)i <= x;i+=2){
         while(x%i == 0){
            x/=i;
            ret.push_back(i);
         }
      }
      if(x>1) ret.push_back(x);
      return ret;
   }
```

```
  template<typename T>
  typename enable_if<is_integral<T>::value, vector<T>>::type
factorize(T x){
    //cerr << "factor: " << x << "\n";
    vector<T> ret;
    const uint32_t trial_limit = 5000;
            auto trial = [&](uint32_t const&i){
                    while(x%i == 0){
      x/=i;
      ret.push_back(i);
    }
            };
            trial(2);
            trial(3);
    for(uint32_t i=5, j=2;i<trial_limit && i*i <= x;i+=j, j=6-j){
      trial(i);
    }
    if(x>1){
      static stack<T> s;
      s.push(x);
      while(!s.empty()){
        x = s.top(); s.pop();
        if(!miller_rabin(x)){
          T factor = squfof(x);
          if(factor == 1 || factor == x){assert(0); return ret;}
          //cerr << x << " -> " << factor << "\n";
          s.push(factor);
          s.push(x/factor);
        } else {
          ret.push_back(x);
        }
      }
    }
    sort(ret.begin(), ret.end());
    return ret;
  }
}

using ll = int64_t;

int main()
{
      fast;
      ll i,j,k,n,m;
  cin>>n;
  auto v=NT::factorize(n);
  for(auto x:v) cout<<x<<' ';
  return 0;
}
```

6. Power Tower

```
///Given an array for each query (l,r) find
pow(a[l],pow(a[l+1],pow….a[r])%mod
#define MOD(a,b) ((a<b)?a:b+a%b)
ll qpow(ll n,ll k,ll mod) {ll ans=1;while(k){if(k&1)
ans=MOD(ans*n,mod);n=MOD(n*n,mod);k>>=1;}return ans;}
ll a[N];
map<ll,ll>mp;
```

```
ll phi(ll n)
{
   if(mp.count(n)) return mp[n];
   ll i,ans=n,store=n;
   for(i=2;i*i<=n;i++){
      if(n%i==0){
         while(n%i==0) n/=i;
         ans=ans/i*(i-1);
      }
   }
   if(n>1) ans=ans/n*(n-1);
   return mp[store]=ans;
}
ll yo(ll l,ll r,ll mod)
{
   if(l==r) return MOD(a[l],mod);
   if(mod==1) return 1;
   else return qpow(a[l],yo(l+1,r,phi(mod)),mod);
}
int main()
{
   BeatMeScanf;
   ll i,j,k,n,m,mod,q,l,r;
   cin>>n>>mod;
   for(i=1;i<=n;i++) cin>>a[i];
   cin>>q;
   while(q--){
      cin>>l>>r;
      cout<<yo(l,r,mod)%mod<<nl;
   }
   return 0;
}
```

7. Mobius Function

```
int mob[N];
void mobius()
{
   for(int i=1;i<N;i++) mob[i]=3;
   mob[1]=1;
   for(int i=2;i<N;i++){
      if(mob[i]==3){
         mob[i]=-1;
         for(int j=2*i;j<N;j+=i) mob[j]=(mob[j]==3?-1:mob[j]*(-1));
         if(i<N/i) for(int j=i*i;j<N;j+=i*i) mob[j]=0;
      }
   }
}
int main()
{
   fast;
   ll i,j,k,n,m;
   mobius();
   return 0;
}
```

8. Modular Inverse

```
ll inv[N];
int main()
```

```
{
   BeatMeScanf;
   ll i,j,k,n,m;
   inv[1] = 1;
   for (i = 2; i <N; i++ ) {
      inv[i] = (-(1LL*mod/i) * inv[mod%i] ) % mod;
      inv[i] = inv[i] + mod;
   }
   for(i=1;i<10;i++) cout<<inv[i]<<' ';
   return 0;
}
```

9. Factoradic Number

```
vi decimal_to_factoradic(int n)
{
   vi v;
   int i=1;
   while(n){
      v.eb(n%i);
      n/=i;
      i++;
   }
   rev(v);
   return v;
}
int factoradic_to_decimal(vi &v)
{
   int n=v.size();
   int ans=0;
   for(int i=0,mul=n;i<n;i++,mul--) ans=(ans*mul%mod+v[i])%mod;
   return ans;
}
vi permutation(int n,vi &v)
{
   o_set<int>se;
   int sz=v.size();
   vi p;
   for(int i=0;i<n-sz;i++) p.eb(i);
   for(int i=n-sz;i<n;i++) se.insert(i);
   for(int i=0;i<sz;i++){
      int nw=*se.fbo(v[i]);
      p.eb(nw);
      se.erase(nw);
   }
   return p;
}
///returns k-th lexicographically smallest permutation of size n
///0-th permutation is the unit permutation i.e. 0,1,2,....n-1
vi kth_perm(int n,int k)
{
   ///need to return something when k>=n!
   vi v=decimal_to_factoradic(k);
   return permutation(n,v);
}
vi factoradic_order(vi &p)
{
   o_set<int>se;
   int n=p.size();
```

```
    for(int i=0;i<n;i++) se.insert(p[i]);
    vi fac;
    for(int i=0;i<n;i++){
        int x=se.ook(p[i]);
        fac.eb(x);
        se.erase(p[i]);
    }
    return fac;
}
///?-th lexicographically smallest permutation of size n
int order(vi &p)
{
    vi fac=factoradic_order(p);
    return factoradic_to_decimal(fac);
}
```

```
///Given two permutations of size n, find Perm((ord(P)+ord(Q))mod
n!)
///where Perm(k) is k-th lexicographically smallest permutation
///and ord(P) is the number k of the permutation
int main()
{
    BeatMeScanf;
    int i,j,k,n,m;
    cin>>n;
    vi p(n);
    for(i=0;i<n;i++) cin>>p[i];
    vi q(n);
    for(i=0;i<n;i++) cin>>q[i];
    vi ordp=factoradic_order(p);
    vi ordq=factoradic_order(q);
    vi sum=ordp;
    int carry=0;
    for(i=n-1;i>=0;i--){
        sum[i]+=ordq[i]+carry;
        carry=sum[i]/(n-i);
        sum[i]%=(n-i);
    }
    vi perm=permutation(n,sum);
    for(i=0;i<n;i++) cout<<perm[i]<<' ';
    return 0;
}
```

10. Linear Sieve

```
int lps[N];
vi prime;
void linear_sieve()
{
    for(int i=2;i<N;i++){
        if(lps[i]==0) lps[i]=i,prime.eb(i);
        int sz=prime.size();
        for(int j=0;j<sz&&i*prime[j]<N&&prime[j]<=lps[i];j++)
lps[i*prime[j]]=prime[j];
    }
}
int main()
```

```
{
   fast;
   ll i,j,k,n,m;
   linear_sieve();
   return 0;
}
```

11. Palindromic Numbers

```
ull toint(string s)
{
   ull ans=0;
   for(ull i=0;i<s.size();i++) ans=ans*10+(s[i]-'0');
   return ans;
}
string tostr(ull n)
{
   if(n==0) return "0";
   string ans="";
   int d;
   char c;
   while(n){
      d=n%10;
      c=d+'0';
      ans=ans+c;
      n/=10;
   }
   rev(ans);
   return ans;
}
ull power(ull a,ull b)
{
   ull ans=1;
   while(b--) ans*=a;
   return ans;
}
ull yo(string n)
{
   if(n.size()==1&&n<="9"){
      ull p=toint(n);
      return p+1;
   }
   //debug(n);
   ull len,num,dis,cnt,d;
   string k,tmp,dump,next;
   len=n.size();
   dis=(len+1)/2;
   k=n.substr(0,dis);
   num=toint(k);
   cnt=num-power(10,dis-1);
   tmp=k;
   if(len&1){
      dump=n.substr(0,dis-1);
      rev(dump);
      if(k+dump<=n) cnt++;
   }
   else{
      rev(tmp);
      if(k+tmp<=n) cnt++;
```

```
    }
    //debug(k);
    //debug(cnt);
    //debug(len);
    len--;
    d=power(10,len)-1;
    //debug(d);
   // cout<<nl;
    next=tostr(d);
    return cnt+yo(next);
}
int main()
{
    ull n,i,j,k,t,a,b,res1,res2,cs=0;
    cin>>t;
    while(t--){
        cin>>a>>b;
        _case;
        if(a>b) swap(a,b);
        a--;
        if(a==-1) res1=0;
        else{
           res1=yo(tostr(a));
        }
        res2=yo(tostr(b));
        cout<<res2-res1<<nl;
    }
    return 0;
}
```

DATA STRUCTURE

12. Policy Based Data Structure

Ordered Set and Ordered Map

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

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

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

int main()
{
    fast;
    ll i,j,k,n,m;
    o_set<ll>se;
    se.insert(1);
    se.insert(2);
    cout<<*se.find_by_order(0)<<nl;///k th element
    cout<<se.order_of_key(2)<<nl;///number of elements less than k
    o_map<ll,ll>mp;
    mp.insert({1,10});
    mp.insert({2,20});
```

```
   cout<<mp.find_by_order(0)->second<<nl;///k th element
   cout<<mp.order_of_key(2)<<nl;///number of first elements less
than k
   return 0;
}
```

13. Monotonous Queue

///Complexity: $O(n)$
///Given an array and an integer k,
///find the maximum for each and every contiguous subarray of size k.
///monotonous queue for minimum is similar

```
struct monotonous_queue_max
{
   ///pair.first - the actual value,
   ///pair.second- how many elements were deleted between it and
the one before it.
   deque<pair<int, int>> q;
   void push(int val)
   {
      int cnt = 0;
      while(!q.empty() && q.back().F < val)
      {
         cnt += q.back().S + 1;
         q.pop_back();
      }
      q.eb(val, cnt);
   };
   int top()
   {
      return q.front().F;
   }
   void pop ()
   {
      if (q.front().S > 0)
      {
         q.front().S --;
         return;
      }
      q.pop_front();
   }
};
monotonous_queue_max q;
int a[N];
int main()
{
   BeatMeScanf;
   int i,j,k,n,m;
   cin>>n;
   for(i=1;i<=n;i++) cin>>a[i];
   cin>>k;
   for(i=1;i<k;i++) q.push(a[i]);
   int ans=0;
   for(i=k;i<=n;i++){
      q.push(a[i]);
      cout<<q.top()<<' ';
      q.pop();
   }
   return 0;
}
```

14. Binary Indexed Tree

BIT Standard

```
template <class T>
struct BIT
{
   ///1-indexed
        int sz; ///max size of array+1
        vector<T> t;

        void init(int n)
        {
                sz = n;
                t.assign(sz,0);
        }

        T query(int idx)
        {
                T ans = 0;
                for(; idx >= 1; idx -= (idx & -idx)) ans += t[idx];
                return ans;
        }

        void upd(int idx, T val)
        {
                if(idx <= 0) return;
                for(; idx <sz; idx += (idx & -idx)) t[idx] += val;
        }

        T query(int l, int r) { return query(r) - query(l - 1); }
};
int main()
{
   BeatMeScanf;
   ll i,j,k,n,m;
   BIT<ll>t;
   t.init(N);
   t.upd(7,5);
   t.upd(3,10);
   cout<<t.query(1,10)<<nl;
   return 0;
}
```

BIT 2D

```
template<typename T>
struct BIT
{
   ///1-indexed
   int szr,szc;///max size of array+1
   vector<vector<T>>t;
   void init(int n,int m)
   {
      szr=n,szc=m;
      t.assign(szr,vector<T>(szc,0));
   }
   void upd(int r,int c,T val) ///add val to a[i][j]
   {
      for(int i=r;i<szr;i+=i&-i) for(int j=c;j<szc;j+=j&-j) t[i][j]+=val;
   }
   T query(int r,int c)
```

```
    {
        T sum=0;
        for(int i=r;i>0;i-=i&-i) for(int j=c;j>0;j-=j&-j) sum+=t[i][j];
        return sum;
    }
    T query(int x1,int y1,int x2,int y2) ///returns sum of the
corresponding rectangle
    {
        return query(x2,y2)-query(x2,y1-1)-query(x1-1,y2)+query(x1-
1,y1-1);
    }
};
///in case of range update single query
///for range update use upd(x1,y1,val),upd(x1,y2+1,-
val),upd(x2+1,y1,-val),upd(x2+1,y2+1,val)
int main()
{
    fast;
    ll i,j,k,n,m,q,x1,y1,x2,y2,typ;
    cin>>n>>m;
    BIT<ll>t;
    t.init(N,N);
    for(i=1;i<=n;i++) for(j=1;j<=m;j++) cin>>k,t.upd(i,j,k);
    cin>>q;
    while(q--){
        cin>>typ;
        if(typ==1){
            cin>>i>>j>>k;
            ///add k to a[i][j]
            t.upd(i,j,k);
        }
        else{
            cin>>x1>>y1>>x2>>y2;
            ///make sure that (x1,y1) is top-left and (x2,y2) is bottom-
right
            cout<<t.query(x1,y1,x2,y2)<<nl;
        }
    }
    return 0;
}
```

BIT 2D with range update and range sum

```
///works for range xor update and range xor sum too
ll multree[N][N][2],addtree[N][N][2];
ll yo(ll x)
{
    ///for range sum
    return x;
    ///for range xor
    ///return (x%2);
}
ll query2(ll tree[N][N][2],ll x,ll y)
{
    ll mul=0,add=0;
    for(ll i=y;i>0;i-=i&-i){
        mul+=tree[x][i][0];
        add+=tree[x][i][1];
        ///mul^=tree[x][i][0];
        ///add^=tree[x][i][1];
    }
```

```
    return (mul*yo(y))+add;
    ///return (mul*yo(y))^add;
}
ll query1(ll x,ll y)
{
    ll mul=0,add=0;
    for(ll i=x;i>0;i-=i&-i){
        mul+=query2(multree,i,y);
        add+=query2(addtree,i,y);
        ///mul^=query2(multree,i,y);
        ///add^=query2(addtree,i,y)
    }
    return (mul*yo(x))+add;
    ///return (mul*yo(x))^add;
}
ll query(ll x1,ll y1,ll x2,ll y2)
{
    return (query1(x2,y2)-query1(x1-1,y2)-query1(x2,y1-
1)+query1(x1-1,y1-1));
    ///return (query1(x2,y2)^query1(x1-1,y2)^query1(x2,y1-
1)^query1(x1-1,y1-1));
}
void upd2(ll tree[N][N][2],ll x,ll y,ll mul,ll add)
{
    for(ll i=x;i<N;i+=i&-i){
        for(ll j=y;j<N;j+=j&-j){
            tree[i][j][0]+=mul;
            tree[i][j][1]+=add;
            ///tree[i][j][0]^=mul;
            ///tree[i][j][1]^=add;
        }
    }
}
void upd1(ll x,ll y1,ll y2,ll mul,ll add)
{
    ///for range sum
    upd2(multree,x,y1,mul,-mul*yo(y1-1));
    upd2(multree,x,y2,-mul,mul*yo(y2));
    upd2(addtree,x,y1,add,-add*yo(y1-1));
    upd2(addtree,x,y2,-add,add*yo(y2));
    ///for range xor
    ///upd2(multree,x,y1,mul,mul*yo(y1-1));
    ///upd2(multree,x,y2,mul,mul*yo(y2));
    ///upd2(addtree,x,y1,add,add*yo(y1-1));
    ///upd2(addtree,x,y2,add,add*yo(y2));
}
void upd(ll x1,ll y1,ll x2,ll y2,ll val)
{
    ///for range sum
    upd1(x1,y1,y2,val,-val*yo(x1-1));
    upd1(x2,y1,y2,-val,val*yo(x2));
    ///for range xor
    ///upd1(x1,y1,y2,val,val*yo(x1-1));
    ///upd1(x2,y1,y2,val,val*yo(x2));
}
int main()
{
    fast;
    ll i,j,k,n,m,tt,x1,y1,x2,y2,q,val;
    cin>>n>>m;
```

```
    for(i=1;i<=n;i++){
        for(j=1;j<=m;j++){
            cin>>k;
            upd(i,j,i,j,k);
        }
    }
    cin>>q;
    while(q--){
        cin>>tt;
        if(tt==1){
            cin>>x1>>y1>>x2>>y2>>val;
            /// add val from top-left(x1,y1) to bottom-right (x2,y2);
            upd(x1,y1,x2,y2,val);
        }
        else{
            cin>>x1>>y1>>x2>>y2;
            /// output sum from top-left(x1,y1) to bottom-right (x2,y2);
            cout<<query(x1,y1,x2,y2)<<nl;
        }
    }
    return 0;
}
```

15. Binary Search Tree

```
///the code returns a BST which will create if we add the values one by one
///here nodes are indicated by values and every node must be distinct
set<ll>se;
map<ll,ll>le,ri;///le contains the left child of the node,ri contains right child of the node
int main()
{
    fast;
    ll i,j,k,n,m,ans;
    cin>>n;
    cin>>k;///root of the tree
    se.insert(k);
    for(i=1;i<n;i++){
        cin>>k;
        auto it=se.UB(k);
        if(it!=se.end()&&le.find(*it)==le.end()) le[*it]=k;
        else --it,ri[*it]=k;
        se.insert(k);
    }
    for(i=1;i<=n;i++) cout<<le[i]<<' '<<ri[i]<<nl;
    return 0;
}
```

16. Segment Tree

Persistent Segment Tree

```
struct node
{
    int l,r,val;
    node(){l=r=val=0;}
    node(int _l,int _r,int _val){
        l=_l,r=_r,val=_val;
    }
}t[10*N];
```

```
int root[N],a[N],cnt;
void build(int cur,int b,int e)
{
   if(b==e){
      t[cur]=node(0,0,0);
      return ;
   }
   int left,right,mid=(b+e)/2;
   t[cur].l=left=++cnt;
   t[cur].r=right=++cnt;
   build(left,b,mid);
   build(right,mid+1,e);
   t[cur].val=t[left].val+t[right].val;
}
void upd(int pre,int cur,int b,int e,int i,int v)
{
   if(i<b||i>e) return;
   if(b==e){
      t[cur].val+=v;
      return;
   }
   int left,right,mid=(b+e)/2;
   if(i<=mid){
      t[cur].r=right=t[pre].r;
      t[cur].l=left=++cnt;
      upd(t[pre].l,t[cur].l,b,mid,i,v);
   }
   else{
      t[cur].l=left=t[pre].l;
      t[cur].r=right=++cnt;
      upd(t[pre].r,t[cur].r,mid+1,e,i,v);
   }
   t[cur].val=t[left].val+t[right].val;
}
int query(int pre,int cur,int b,int e,int k)
{
   if(b==e) return b;
   int cnt=t[t[cur].l].val-t[t[pre].l].val;
   int mid=(b+e)/2;
   if(cnt>=k) return query(t[pre].l,t[cur].l,b,mid,k);
   else return query(t[pre].r,t[cur].r,mid+1,e,k-cnt);
}
///1 2 2 3 , 3rd number is 3
///the code returns k-th unique number in a range l to r if the range
were sorted
int flag[N];
int main()
{
   fast;
   int i,j,k,n,m,q,t,x,l,r,c=0;
   map<int,int>mp;
   cin>>n>>q;
   for(i=1;i<=n;i++) cin>>a[i],mp[a[i]];
   for(auto x:mp) mp[x.F]=++c,flag[c]=x.F;
   root[0]=++cnt;
   build(root[0],1,n);
   for(i=1;i<=n;i++){
      root[i]=++cnt;
      upd(root[i-1],root[i],1,n,mp[a[i]],1);
   }
```

```
while(q--){
    cin>>l>>r>>k;
    cout<<flag[query(root[l-1],root[r],1,n,k)]<<nl;
}
```

Dynamic Segment Tree

///Complexity: $O(q \log n)$

```
///Given an zero array of size 1e9
///0 x y v,add v to segment [x,y]
///1 x y,output the sum of segment [x,y]
struct node
{
    node *l,*r;
    ll lazy;
    ll sum;
    node()
    {
        l=NULL;
        r=NULL;
        lazy=sum=0;
    }
};
void propagate(node* t,int b,int e)
{
    if(t->lazy==0) return;
    t->sum+=1LL*(e-b+1)*t->lazy;
    if(!t->l) t->l=new node();
    if(!t->r) t->r=new node();
    t->l->lazy+=t->lazy;
    t->r->lazy+=t->lazy;
    t->lazy=0;
}
void upd(node* t,int b,int e,int i,int j,ll v)
{
    propagate(t,b,e);
    if(!t||b>j||e<i) return;
    if(b>=i&&e<=j){
        t->lazy+=v;
        propagate(t,b,e);
        return;
    }
    int mid=(b+e)/2;
    if(!t->l) t->l=new node();
    if(!t->r) t->r=new node();
    upd(t->l,b,mid,i,j,v);
    upd(t->r,mid+1,e,i,j,v);
    t->sum=t->l->sum+t->r->sum;
}
ll query(node* t,int b,int e,int i,int j)
{
    propagate(t,b,e);
    if(!t||b>j||e<i) return 0;
    if(b>=i&&e<=j) return t->sum;
    int mid=(b+e)/2;
    if(!t->l) t->l=new node();
    if(!t->r) t->r=new node();
    return query(t->l,b,mid,i,j)+query(t->r,mid+1,e,i,j);
}
int main()
{
    BeatMeScanf;
```

```
    int i,j,k,n,m,ty,l,r,q,tt;
    ll v;
    node* root;
    cin>>tt;
    while(tt--){
        root=new node();
        cin>>n;///max index of array,can be upto 1e9 or more
        cin>>q;
        while(q--){
            cin>>ty>>l>>r;
            if(ty==0){
                cin>>v;
                upd(root,1,n,l,r,v);
            }
            else cout<<query(root,1,n,l,r)<<nl;
        }
    }
    return 0;
}
```

Segment Tree 2D

```
int n,m;
struct segtree
{
    int a[N*4];
    segtree()
    {
        for(int i=0;i<N*4;i++) a[i]=0;
    }
    void reset()
    {
        for(int i=0;i<N*4;i++) a[i]=0;
    }
    // update i-th column by val
    void upd(int node,int b,int e,int i,int val,vi &v)
    {
        v.pb(node);
        if(b==e)
        {
            a[node]=val;
            return;
        }
        int stree;
        if(i<=mid) upd(l,b,mid,i,val,v);
        else upd(r,mid+1,e,i,val,v);
        a[node]=a[l]+a[r];
    }
    //sum from column i to j
    int query(int node,int b,int e,int i,int j)
    {
        if(j<b||i>e) return 0;
        if(b>=i&&e<=j) return a[node];
        int stree;
        return query(l,b,mid,i,j)+query(r,mid+1,e,i,j);
    }
};
struct segtree2d
{
    segtree a[N*4];
    void reset()
```

```
    {
        for(int i=0;i<N*4;i++) a[i].reset();
    }
    vi v;
    //set a[i][j]=val
    void upd(int node,int b,int e,int i,int j,int val)
    {
        if(b==e)
        {
            v.clear();
            a[node].upd(1,1,m,j,val,v);
            return;
        }
        int stree;
        if(i>=b&&i<=mid) upd(l,b,mid,i,j,val);
        else upd(r,mid+1,e,i,j,val);
        for(auto x:v) a[node].a[x]=a[l].a[x]+a[r].a[x];
    }
    //return sum from top-left (i,y1) to bottom-right (j,y2)
    int query(int node,int b,int e,int i,int y1,int j,int y2)
    {
        if(j<b||i>e) return 0;
        if(b>=i&&e<=j) return a[node].query(1,1,m,y1,y2);
        int stree;
        return query(l,b,mid,i,y1,j,y2)+query(r,mid+1,e,i,y1,j,y2);
    }
};
segtree2d t;
int main()
{
    // fast;
    int i,j,k,x,x1,y1,x2,y2,typ,q;
//    cin>>n>>m;
//    for(i=1;i<=n;i++){
//        for(j=1;j<=m;j++){
//            cin>>k;
//            t.upd(1,1,n,i,j,k);
//        }
//    }
    int tt;
    tt=sc();
    while(tt--){
        n=sc();
        m=n;
        while(1){
            char s[5];
            sf("%s",&s);
            if(s[2]=='D') break;///end
            else if(s[2]=='T'){///set a[i][j] as k
                i=sc(),j=sc(),x=sc();
                i++,j++;
                t.upd(1,1,n,i,j,x);
            }
            else{

                x1=sc(),y1=sc(),x2=sc(),y2=sc();
                x1++,y1++,x2++,y2++;
                // make sure (x1,y1) is top-left and (x2,y2) is bottom-right
                pf("%d\n",t.query(1,1,n,x1,y1,x2,y2));///sum of the
rectangle
```

```
            }
        }
        t.reset();
    }
    return 0;
}
```

Quad Tree

```
#define Max 501
#define INF (1 << 30)
int P[Max][Max]; // container for 2D grid

/* 2D Segment Tree node */
struct Point {
    int x, y, mx;
    Point() {}
    Point(int x, int y, int mx) : x(x), y(y), mx(mx) {}

    bool operator < (const Point& other) const {
        return mx < other.mx;
    }
};

struct Segtree2d {

    // I didn't calculate the exact size needed in terms of 2D container size.
    // If anyone, please edit the answer.
    // It's just a safe size to store nodes for MAX * MAX 2D grids which won't cause stack overflow :)
    Point T[500000]; // TODO: calculate the accurate space needed

    int n, m;

    // initialize and construct segment tree
    void init(int n, int m) {
        this -> n = n;
        this -> m = m;
        build(1, 1, 1, n, m);
    }

    // build a 2D segment tree from data [ (a1, b1), (a2, b2) ]
    // Time: O(n logn)
    Point build(int node, int a1, int b1, int a2, int b2) {
        // out of range
        if (a1 > a2 or b1 > b2)
            return def();

        // if it is only a single index, assign value to node
        if (a1 == a2 and b1 == b2)
            return T[node] = Point(a1, b1, P[a1][b1]);

        // split the tree into four segments
        T[node] = def();
        T[node] = maxNode(T[node], build(4 * node - 2, a1, b1, (a1 + a2) / 2, (b1 + b2) / 2 ) );
        T[node] = maxNode(T[node], build(4 * node - 1, (a1 + a2) / 2 + 1, b1, a2, (b1 + b2) / 2 ));
        T[node] = maxNode(T[node], build(4 * node + 0, a1, (b1 + b2) / 2 + 1, (a1 + a2) / 2, b2) );
```

```
      T[node] = maxNode(T[node], build(4 * node + 1, (a1 + a2) / 2 +
1, (b1 + b2) / 2 + 1, a2, b2) );
      return T[node];
   }

   // helper function for query(int, int, int, int);
   Point query(int node, int a1, int b1, int a2, int b2, int x1, int y1, int
x2, int y2) {
      // if we out of range, return dummy
      if (x1 > a2 or y1 > b2 or x2 < a1 or y2 < b1 or a1 > a2 or b1 > b2)
         return def();

      // if it is within range, return the node
      if (x1 <= a1 and y1 <= b1 and a2 <= x2 and b2 <= y2)
         return T[node];

      // split into four segments
      Point mx = def();
      mx = maxNode(mx, query(4 * node - 2, a1, b1, (a1 + a2) / 2, (b1
+ b2) / 2, x1, y1, x2, y2) );
      mx = maxNode(mx, query(4 * node - 1, (a1 + a2) / 2 + 1, b1, a2,
(b1 + b2) / 2, x1, y1, x2, y2) );
      mx = maxNode(mx, query(4 * node + 0, a1, (b1 + b2) / 2 + 1, (a1
+ a2) / 2, b2, x1, y1, x2, y2) );
      mx = maxNode(mx, query(4 * node + 1, (a1 + a2) / 2 + 1, (b1 +
b2) / 2 + 1, a2, b2, x1, y1, x2, y2));

      // return the maximum value
      return mx;
   }

   // query from range [ (x1, y1), (x2, y2) ]
   // Time: O(logn)
   Point query(int x1, int y1, int x2, int y2) {
      return query(1, 1, 1, n, m, x1, y1, x2, y2);
   }

   // helper function for update(int, int, int);
   Point update(int node, int a1, int b1, int a2, int b2, int x, int y, int
value) {
      if (a1 > a2 or b1 > b2)
         return def();

      if (x > a2 or y > b2 or x < a1 or y < b1)
         return T[node];

      if (x == a1 and y == b1 and x == a2 and y == b2)
         return T[node] = Point(x, y, value);

      Point mx = def();
      mx = maxNode(mx, update(4 * node - 2, a1, b1, (a1 + a2) / 2,
(b1 + b2) / 2, x, y, value) );
      mx = maxNode(mx, update(4 * node - 1, (a1 + a2) / 2 + 1, b1,
a2, (b1 + b2) / 2, x, y, value));
      mx = maxNode(mx, update(4 * node + 0, a1, (b1 + b2) / 2 + 1,
(a1 + a2) / 2, b2, x, y, value));
      mx = maxNode(mx, update(4 * node + 1, (a1 + a2) / 2 + 1, (b1 +
b2) / 2 + 1, a2, b2, x, y, value) );
      return T[node] = mx;
   }
```

```
    // update the value of (x, y) index to 'value'
    // Time: O(logn)
    Point update(int x, int y, int value) {
        return update(1, 1, 1, n, m, x, y, value);
    }

    // utility functions; these functions are virtual because they will be
overridden in child class
    virtual Point maxNode(Point a, Point b) {
        return max(a, b);
    }

    // dummy node
    virtual Point def() {
        return Point(0, 0, -INF);
    }
};

/* 2D Segment Tree for range minimum query; a override of
Segtree2d class */
struct Segtree2dMin : Segtree2d {
    // overload maxNode() function to return minimum value
    Point maxNode(Point a, Point b) {
        return min(a, b);
    }

    Point def() {
        return Point(0, 0, INF);
    }
};

// initialize class objects
Segtree2d Tmax;
Segtree2dMin Tmin;


/* Drier program */
int main(void) {
    int n, m;
    // input
    scanf("%d %d", &n, &m);
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            scanf("%d", &P[i][j]);

    // initialize
    Tmax.init(n, m);
    Tmin.init(n, m);

    // query
    int x1, y1, x2, y2;
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2);

    cout<<Tmax.query(x1, y1, x2, y2).mx<<endl;
    cout<<Tmin.query(x1, y1, x2, y2).mx<<endl;

    // update
    int x, y, v;
    scanf("%d %d %d", &x, &y, &v);
```

```
    Tmax.update(x, y, v);
    Tmin.update(x, y, v);

    return 0;
}
```

17. Disjoint Set Union

Persistent DSU

///Standard DSU with 'last added edges can be removed' capability
///Here is u,v connected? Query er answer deowa ache
///for connected component query see Dynamic Connectivity Problem

```
struct persistent_dsu
{
        struct state
        {
                int u, v, rnku, rnkv;
                state() {u = -1; v = -1; rnkv = -1; rnku = -1;}
                state(int _u, int _rnku, int _v, int _rnkv)
                {
                        u = _u;
                        rnku = _rnku;
                        v = _v;
                        rnkv = _rnkv;
                }
        };

        stack<state> st;
        int par[N], depth[N];

        persistent_dsu() {memset(par, -1, sizeof(par));
memset(depth, 0, sizeof(depth));}

        int root(int x)
        {
                if(x == par[x]) return x;
                return root(par[x]);
        }

        void init(int n)
        {
                for(int i = 0; i <= n; i++)
                {
                        par[i] = i;
                        depth[i] = 1;
                }
        }

        bool connected(int x, int y)
        {
                return root(x) == root(y);
        }

        void unite(int x, int y)
        {
                int rx = root(x), ry = root(y);
                st.push(state(rx, depth[rx], ry, depth[ry]));

                if(depth[rx] < depth[ry])
                        par[rx] = ry;
```

```
                else if(depth[ry] < depth[rx])
                        par[ry] = rx;
                else
                {
                        par[rx] = ry;
                        depth[rx]++;
                }
        }
        ///how many last added edges you want to erase
        void backtrack(int c)
        {
                while(!st.empty() && c)
                {
                        par[st.top().u] = st.top().u;
                        par[st.top().v] = st.top().v;
                        depth[st.top().u] = st.top().rnku;
                        depth[st.top().v] = st.top().rnkv;
                        st.pop();
                        c--;
                }
        }
};
persistent_dsu d;
int main()
{
        BeatMeScanf;
        int i,j,k,n,m,u,v;
        cin>>n>>m;
        d.init(n);
        while(m--){
        int ty;
        cin>>ty;
        if(ty==1){
           ///add an edge
           cin>>u>>v;
           d.unite(u,v);
        }
        else if(ty==2){
           ///remove last added k edges
           cin>>k;
           d.backtrack(k);
        }
        else{
           ///if u and v is connected
           cin>>u>>v;
           if(d.connected(u,v)) cout<<"YES\n";
           else cout<<"NO\n";
        }
         }
}
```

Dynamic Connectivity Problem

Complexity:$O(m \log m)$, $where\ m\ is\ number\ of\ edges\ in\ the\ graph$

/// +,Add an edge to the graph
/// -, Delete an edge from the graph
/// ?, Count the number of connected components in the graph

```
struct persistent_dsu
{
        struct state
```

```
	{
		int u, v, rnku, rnkv;
		state() {u = -1; v = -1; rnkv = -1; rnku = -1;}
		state(int _u, int _rnku, int _v, int _rnkv)
		{
			u = _u;
			rnku = _rnku;
			v = _v;
			rnkv = _rnkv;
		}
	};

	stack<state> st;
	int par[N], depth[N];
	int comp;
	persistent_dsu()	{comp=0;memset(par,	-1,	sizeof(par));
memset(depth, 0, sizeof(depth));}

	int root(int x)
	{
		if(x == par[x]) return x;
		return root(par[x]);
	}

	void init(int n)
	{
	   comp=n;
		for(int i = 0; i <= n; i++)
		{
			par[i] = i;
			depth[i] = 1;
		}
	}

	bool connected(int x, int y)
	{
		return root(x) == root(y);
	}

	void unite(int x, int y)
	{
		int rx = root(x), ry = root(y);
		if(rx==ry){
st.push(state());
return;
		}
		if(depth[rx] < depth[ry])
			par[rx] = ry;
		else if(depth[ry] < depth[rx])
			par[ry] = rx;
		else
		{
			par[rx] = ry;
			depth[rx]++;
		}
		comp--;
		st.push(state(rx, depth[rx], ry, depth[ry]));
	}
```

```
        ///how many last added edges you want to erase
        void backtrack(int c)
        {
                while(!st.empty() && c)
                {
                   if(st.top().u==-1){
          st.pop();
          c--;
          continue;
                    }
                        par[st.top().u] = st.top().u;
                        par[st.top().v] = st.top().v;
                        depth[st.top().u] = st.top().rnku;
                        depth[st.top().v] = st.top().rnkv;
                        st.pop();
                        c--;
                        comp++;
                }
        }
};
persistent_dsu d;
vpii alive[4*N];
void upd(int n,int b,int e,int i,int j,pii &p)
{
   if(b>j||e<i) return;
   if(b>=i&&e<=j){
      alive[n].eb(p);///this edge was alive in this time range
      return;
   }
   int stree;
   upd(l,b,mid,i,j,p);
   upd(r,mid+1,e,i,j,p);
}
int ans[N];
void query(int n,int b,int e)
{
   if(b>e) return;
   int prevsz=d.st.size();
   ///add edges which were alive in this range
   for(auto p:alive[n]) d.unite(p.F,p.S);
   if(b==e){
      ans[b]=d.comp;
      d.backtrack(d.st.size()-prevsz);
      return;
   }
   int stree;
   query(l,b,mid);
   query(r,mid+1,e);
   d.backtrack(d.st.size()-prevsz);
}
struct HASH{
  size_t operator()(const pair<int,int>&x)const{
    return hash<long long>()(((long long)x.first)^(((long
long)x.second)<<32));
  }
};
set<pii>se;
bool isquery[N];
```

```
map<pii,int>st;
int main()
{
   BeatMeScanf;
   //freopen("connect.in", "r", stdin);
   //freopen("connect.out", "w", stdout);
   int i,j,k,n,m,u,v;
   cin>>n>>m;
   d.init(n);
   for(i=1;i<=m;i++){
      string ty;
      cin>>ty;
      if(ty=="?"){
         isquery[i]=1;
      }
      else if(ty=="+"){
         cin>>u>>v;
         if(u>v) swap(u,v);
         pii p={u,v};
         se.insert(p);
         st[p]=i;
      }
      else{
         cin>>u>>v;
         if(u>v) swap(u,v);
         pii p={u,v};
         se.erase(p);
         upd(1,1,m,st[p],i-1,p);///in this time range this edge was in the
DSU
      }
   }
   for(auto p:se) upd(1,1,m,st[p],m,p);///update rest of the edges
   se.clear();
   query(1,1,m);
   for(i=1;i<=m;i++) if(isquery[i]) cout<<ans[i]<<nl;
   return 0;
}
```

Augmented DSU

///Application:- used for maintaining a system of equations of the form (y-x = d) along
///with their consistencial queries dynamically using disjoint set union and find data structure.

```
int flaw;  ///counting numbers of inconsistent assertions
int val[N]; ///val[i]=a[i]-a[root[i]] where root[i]=root of the
corresponding dsu of i
int par[N]; ///adding a[i]-a[j]=d means setting j=par[i] and updating
val[i]

void init(int n)
{
        flaw=0;
        for(int i=1;i<=n;++i)
        {
                par[i]=i;
                val[i]=0;
        }
}
```

```
int find_(int x)
{
        if(par[x]==x) return x;
        int rx=find_(par[x]);      /// rx is the root of x
        val[x]=val[par[x]]+val[x];      ///update all val along the path,i.e.,val calculated wrt root
        par[x]=rx;
        return rx;
}

void merge_(int a,int b,int d)
{
        int ra=find_(a);
        int rb=find_(b);
        if(ra==rb && val[a]-val[b]!=d) flaw++;
        else if(ra!=rb)
        {
           if(rand()%2){
        val[ra]=d+val[b]-val[a];
        par[ra]=rb;
           }
           else{
        val[rb]=d+val[a]-val[b];
        par[rb]=ra;
           }
        }
}
```

```
int main()
{
   int i,j,k,n,m;
        cin>>n; ///no. of variables
        cin>>m; ///no. of equations
        init(n);
        for(int  i=1;i<=m;++i)        ///consider 1-based indexing of variables
        {
                int a,b,d;        ///asserting a-b=d;
                cin>>a>>b>>d;
                merge_(a,b,d);
        }
        cout<<"No. of inconsistencies= "<<flaw;
        ///queries of type y-x=? can be given through val[y]-val[x]
        ///(only when then are in same component
   ///i.e., can be extracted from the information so far )
        return 0;
}
```

18. MO's Algorithm

MO's Algorithm Standard

Complexity: $O((n+q)\sqrt{n})$

```
struct sj
{
   int l,r,idx;
}q[mxn];
int block,a[mxn],curl,curr,cnt[mxn*5];
ll ans,sol[mxn];
```

```
bool cmp(sj a,sj b)
{
   if(a.l/block==b.l/block) return a.r<b.r;
   else return a.l/block<b.l/block;
}
inline void add(int x)//careful x is the number not the index
{
// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
   ans-=1LL*cnt[x]*cnt[x]*x;
   cnt[x]++;
   ans+=1LL*cnt[x]*cnt[x]*x;
}
inline void remov(int x)
{
// Removing is much like adding.
   ans-=1LL*cnt[x]*cnt[x]*x;
   cnt[x]--;
   ans+=1LL*cnt[x]*cnt[x]*x;
}
int main()
{
   fast;
   int i,j,k,n,m,t;
   cin>>n>>t;
   for(i=1;i<=n;i++) cin>>a[i];
   for(i=0;i<t;i++){
      cin>>q[i].l>>q[i].r;
      q[i].idx=i;
   }
   block=(int)sqrt(n);
   sort(q,q+t,cmp);
   for(i=0;i<t;i++){
      while(curl<q[i].l) remov(a[curl++]);
      while(curl>q[i].l) add(a[--curl]);
      while(curr<q[i].r) add(a[++curr]);
      while(curr>q[i].r) remov(a[curr--]);
      sol[q[i].idx]=ans;
   }
   for(i=0;i<t;i++) cout<<sol[i]<<nl;
   return 0;
}
```

MO's Algorithm GilbertOrder

Complexity: $O(n\sqrt{q})$

```
//always better than standard algorithm
//far better for small amount of queries
struct sj
{
   int l,r,idx;
   int64_t ord;
}q[N];
int a[N],curl,curr,cnt[N*5];
ll ans,sol[N];
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
        if (pow == 0) {
                return 0;
        }
        int hpow = 1 << (pow-1);
```

```
        int seg = (x < hpow) ? (
                (y < hpow) ? 0 : 3
        ) : (
                (y < hpow) ? 1 : 2
        );
        seg = (seg + rotate) & 3;
        const int rotateDelta[4] = {3, 0, 0, 1};
        int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
        int nrot = (rotate + rotateDelta[seg]) & 3;
        int64_t subSquareSize = int64_t(1) << (2*pow - 2);
        int64_t ans = seg * subSquareSize;
        int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
        ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add -
1);
        return ans;
}
bool cmp(sj a,sj b)
{
   return a.ord<b.ord;
}
inline void add(int x)//careful x is the number not the index
{
// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
   ans-=1LL*cnt[x]*cnt[x]*x;
   cnt[x]++;
   ans+=1LL*cnt[x]*cnt[x]*x;
}
inline void remov(int x)
{
// Removing is much like adding.
   ans-=1LL*cnt[x]*cnt[x]*x;
   cnt[x]--;
   ans+=1LL*cnt[x]*cnt[x]*x;
}
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,t;
   cin>>n>>t;
   for(i=1;i<=n;i++) cin>>a[i];
   for(i=0;i<t;i++){
      cin>>q[i].l>>q[i].r;
      q[i].idx=i;
   }
   for(i=0;i<t;i++) q[i].ord=gilbertOrder(q[i].l,q[i].r,21,0);
   sort(q,q+t,cmp);
   for(i=0;i<t;i++){
      while(curl<q[i].l) remov(a[curl++]);
      while(curl>q[i].l) add(a[--curl]);
      while(curr<q[i].r) add(a[++curr]);
      while(curr>q[i].r) remov(a[curr--]);
      sol[q[i].idx]=ans;
   }
   for(i=0;i<t;i++) cout<<sol[i]<<nl;
   return 0;
}
```

Mo’s on Tree

```
///unique elements in path u to v
```

```
int ans,curl,curr,timee,que,block,val[mxn],node[mxn*2],n;
int
st[mxn],en[mxn],level[mxn],par[mxn][20],cnt[mxn],id[mxn],sol[mxm
];
bool vis[mxn];
vi g[mxn];
void remov(int u);
struct sj
{
   int l,r,lc,idx;
}q[mxm];
void dfs(int u,int prev)
{
   par[u][0]=prev;
   level[u]=level[prev]+1;
   node[timee]=u;
   st[u]=timee++;
   for(auto v:g[u]){
      if(v==prev) continue;
      dfs(v,u);
   }
   node[timee]=u;
   en[u]=timee++;
}
int lca(int u,int v)
{
   if(level[u]<level[v]) swap(u,v);
   for(int k=19;k>=0;k--) if(level[par[u][k]]>=level[v]) u=par[u][k];
   if(u==v) return u;
   for(int k=19;k>=0;k--) if(par[u][k]!=par[v][k])
u=par[u][k],v=par[v][k];
   return par[u][0];
}
bool cmp(sj x,sj y)
{
   int l=(x.l-1)/block;
   int r=(y.l-1)/block;
   if(l==r) return x.r<y.r;
   else return l<r;
}
void add(int u)
{
   int x=id[u];
   if(cnt[x]++==0) ans++;
   return;
}
void remov(int u)
{
   int x=id[u];
   if(--cnt[x]==0) ans--;
   return;
}
void add_list(int u)
{
   if(vis[u]==0) add(u);
   else remov(u);
   vis[u]^=1;
}
void remov_list(int u)
```

```
{
   if(vis[u]==0) add(u);
   else remov(u);
   vis[u]^=1;
}
void mos_algo()
{
   int i,j,k,u,v,l,r;
   sort(q,q+que,cmp);
   curl=q[0].l,curr=q[0].l-1;
   ans=0;
   for(i=0;i<que;i++){
      l=q[i].l,r=q[i].r;
      while(curl<l) remov_list(node[curl++]);
      while(curl>l) add_list(node[--curl]);
      while(curr<r) add_list(node[++curr]);
      while(curr>r) remov_list(node[curr--]);
      u=node[curl],v=node[curr];
      if(q[i].lc!=u&&q[i].lc!=v) add_list(q[i].lc);
      sol[q[i].idx]=ans;
      if(q[i].lc!=u&&q[i].lc!=v) remov_list(q[i].lc);
   }
}
void compress()
{
   map<int,int>mp;
   for(int i=1;i<=n;i++){
      if(mp.find(val[i])==mp.end()) mp[val[i]]=mp.size();
      id[i]=mp[val[i]];
   }
}
int main()
{
   fast;
   int i,j,k,m,u,v;
   while(cin>>n>>que){
      timee=1;
      ans=0;
      for(i=1;i<=n;i++) cin>>val[i];
      compress();
      for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
      dfs(1,0);
      for(k=1;k<20;k++) for(i=1;i<=n;i++) par[i][k]=par[par[i][k-1]][k-1];
      timee--;
      block=(int)(sqrt(timee)+eps);
      for(i=0;i<que;i++){
         cin>>u>>v;
         if(level[u]>level[v]) swap(u,v);
         q[i].lc=lca(u,v);
         if(q[i].lc==u) q[i].l=st[u],q[i].r=st[v];
         else q[i].l=en[u],q[i].r=st[v];
         q[i].idx=i;
      }
      mos_algo();
      for(i=0;i<que;i++) cout<<sol[i]<<nl;
      for(i=0;i<=n;i++){
         level[i]=0;
         vis[i]=0,cnt[i]=0,g[i].clear();
         for(k=0;k<20;k++) par[i][k]=0;
      }
```

```
    }
    return 0;
}
```

Mo's with Update

Complexity: $O(n^{\frac{5}{3}} + q.n^{\frac{2}{3}})$

```
///sum of distinct numbers in range with update
int block;
struct query
{
    int l,r;///query range
    int updcnt;///number of update happened before this query
    int idx;///query index
    bool operator<(const query& x)const
    {
        if(l/block==x.l/block){
            if(r/block==x.r/block) return updcnt<x.updcnt;
            else return r/block<x.r/block;
        }
        else return l/block<x.l/block;
    }
}q[N];
struct Upd
{
    int idx;///update index
    int prv;///array had value prv before this update
    int nxt;///array will have value nxt after this update
}upd[N];
int curl,curr,a[N],cnt[5*N],val[5*N];
ll ans;
umap<int,int>mp;
/// When adding a number, we first nullify it's effect on current
/// answer, then update corresponding array, then account for it's
effect again.
inline void add(int x)
{
    cnt[x]++;
    if(cnt[x]==1) ans+=val[x];
}
///removing is quite same as adding
inline void remov(int x)
{
    cnt[x]--;
    if(cnt[x]==0) ans-=val[x];
}
inline void do_upd(int upd_idx)
{
    int i=upd[upd_idx].idx;
    int now=upd[upd_idx].nxt;
    ///if the update is within our current range then re-correct our
ans
    if(i>=curl&&i<=curr) remov(a[i]);
    a[i]=now;
    if(i>=curl&&i<=curr) add(a[i]);
}
inline void undo_upd(int upd_idx)
{
    int i=upd[upd_idx].idx;
    int now=upd[upd_idx].prv;
```

```
   ///if the update is within our current range then re-correct our
ans
   if(i>=curl&&i<=curr) remov(a[i]);
   a[i]=now;
   if(i>=curl&&i<=curr) add(a[i]);
}
int aux[N];
ll res[N];
int main()
{
  //BeatMeScanf;
   int i,j,k,n,m,U=0,Q=0,que,l,r,t,z=0;
   char ch;
   sf1(n);
   for(i=1;i<=n;i++){
      sf1(a[i]);
      if(!mp.count(a[i])) mp[a[i]]=++z,val[z]=a[i];///compressing
      a[i]=mp[a[i]];
      aux[i]=a[i];

   }
   sf1(que);
   for(i=0;i<que;i++){
      sf(" %c",&ch);
      sf2(l,r);
      if(ch=='U'){
         ++U;
         upd[U].idx=l;
         if(!mp.count(r)) mp[r]=++z,val[z]=r;///compressing
         upd[U].nxt=mp[r];
         upd[U].prv=aux[l];
         aux[l]=mp[r];
      }
      else{
         q[Q].l=l;
         q[Q].r=r;
         q[Q].updcnt=U;
         q[Q].idx=i;
         Q++;
      }
   }
   block=(int)(cbrt(n*1.0)+eps);
   block*=block;
   sort(q,q+Q);
   int cur=0;///tracks the number of updates already happened
   mem(res,-1);
   for(i=0;i<Q;i++){
      l=q[i].l,r=q[i].r,t=q[i].updcnt;
      while(cur<t){
         cur++;
         do_upd(cur);
      }
      while(cur>t){
         undo_upd(cur);
         cur--;
      }
      while(curl<l) remov(a[curl++]);
      while(curl>l) add(a[--curl]);
      while(curr<r) add(a[++curr]);
      while(curr>r) remov(a[curr--]);
```

```
    res[q[i].idx]=ans;
  }
  for(i=0;i<que;i++) if(res[i]!=-1) pf1ll(res[i]);
  return 0;
}
```

MO's with DSU

///Complexity: $O((m+q)\sqrt{m}.some\ small\ constant)$

```
///This code runs for 1<=n,m,q<=2e5 in 2 second
///Given n vertices and m edges perform q queries of type (l,r)
///output the number of connected components if we added edges i
such that l<=i<=r
const int N = 2e5+9;
struct query
{
	int l, r, idx;
	query() {l = 0; r = 0; idx = 0;}
	query(int _l, int _r, int _idx)
	{
		l = _l;
		r = _r;
		idx = _idx;
	}
};

struct persistent_dsu
{
	struct state
	{
		int u, ru, v, rv;
		state() {u = 0; ru = 0; v = 0; rv = 0;}
		state(int _u, int _ru, int _v, int _rv)
		{
			u = _u;
			ru = _ru;
			v = _v;
			rv = _rv;
		}
};

int cnt;
int depth[N], par[N];
stack<state> st;

persistent_dsu()
{
	cnt = 0;
	memset(depth, 0, sizeof(depth));
	memset(par, 0, sizeof(par));
	while(!st.empty()) st.pop();
}

void init(int _sz)
{
	cnt = _sz;
	for(int i = 0; i <= _sz; i++)
		par[i] = i, depth[i] = 1;
}

int root(int x)
{
```

```
	if(x == par[x]) return x;
	return root(par[x]);
}

bool connected(int x, int y)
{
	return root(x) == root(y);
}

void unite(int x, int y)
{
	int rx = root(x), ry = root(y);
	if(rx == ry) return;

	if(depth[rx] < depth[ry])
		par[rx] = ry;
	else if(depth[ry] < depth[rx])
		par[ry] = rx;
	else par[rx] = ry, depth[ry]++;

	cnt--;
	st.push(state(rx, depth[rx], ry, depth[ry]));

}

void snapshot() { st.push(state(-1, -1, -1, -1)); }

void rollback()
{
	while(!st.empty())
	{
		if(st.top().u == -1)
			return;

		++cnt;
		par[st.top().u] = st.top().u;
		par[st.top().v] = st.top().v;
		depth[st.top().u] = st.top().ru;
		depth[st.top().v] = st.top().rv;
		st.pop();
	}
}
};

struct edge
{
	int u, v;
	edge() {u = 0; v = 0;}
	edge(int _u, int _v)
	{
		u = _u;
		v = _v;
	}
};

int n, ed, m;
edge a[N];
query q[N];

void read()
```

```
{
	cin >> n >> ed >> m;

	for(int i = 1; i <= ed; i++)
	{
		int u, v;
		cin >> u >> v;
		a[i] = edge(u, v);
	}
}

int rt, cnt_q;
persistent_dsu d;

bool cmp(query fir, query sec)
{
	if(fir.l / rt != sec.l / rt) return fir.l / rt < sec.l / rt;
	return fir.r < sec.r;
}

int answer[N];
void add(int idx) { d.unite(a[idx].u, a[idx].v); }

void solve()
{
	d.init(n);
	d.snapshot();
	rt = sqrt(ed);
	cnt_q = 0;
	int fm = m;
	for(int i = 0; i < m; i++)
	{
		int l, r;
		cin >> l >> r;

		if(r - l + 1 <= rt)
		{
			for(int k = l; k <= r; k++) add(k);
			answer[i] = d.cnt;
			d.rollback();
			continue;
		}

		q[cnt_q++] = query(l, r, i);
	}

	m = cnt_q;
	sort(q, q + m, cmp);
	int last, border, last_block = -1, block;

	for(int i = 0; i < m; i++)
	{
		block = q[i].l / rt;
		if(last_block != block)
		{
			d.init(n);
			border = rt * (block + 1);
			last = border;
		}
```

```
                last_block = block;
                for(int k = last + 1; k <= q[i].r; k++) add(k);
                d.snapshot();

                for(int k = q[i].l; k <= border; k++) add(k);
                answer[q[i].idx] = d.cnt;
                d.rollback();

                last = q[i].r;

        }

        for(int i = 0; i < fm; i++)
                cout << answer[i] << endl;
}

int main()
{
        ios_base::sync_with_stdio(false);
        int t;
        cin>>t;
        while(t--){
    read();
    solve();
        }
        return 0;
}
```

19. Sparse Table

```
///Standard RMQ problem
int t[N][21],lg2[N];
int main()
{
   fast;
   int i,j,k,n,m,l,r,q;
   lg2[1]=0;
   for(i=2;i<N;i++) lg2[i]=lg2[i/2]+1;
   cin>>n;
   for(i=1;i<=n;i++) cin>>k,t[i][0]=k;
   for(j=1;j<=20;j++) for(i=1;i+(1<<(j-1))<=n;i++) t[i][j]=min(t[i][j-
1],t[i+(1<<(j-1))][j-1]);
   cin>>q;
   while(q--){
      cin>>l>>r;
      l++,r++;
      ///int ans=INT_MAX;
      ///for(i=20;i>=0;i--) if(l+(1<<i)-1<=r)
ans=min(ans,t[l][i]),l+=(1<<i);
      k=lg2[r-l+1];
      int ans=min(t[l][k],t[r-(1<<k)+1][k]);
      cout<<ans<<nl;
   }
   return 0;
}
```

20. Merge Sort Tree

```
///number of elements greater than k in a range
```

```
vi t[4*N];
int a[N];
void build(int n,int b,int e)
{
   if(b==e){
      t[n].eb(a[b]);
      return;
   }
   int stree;
   build(l,b,mid);
   build(r,mid+1,e);
   merge(all(t[l]),all(t[r]),back_inserter(t[n]));
}
int query(int n,int b,int e,int i,int j,int k)
{
   if(b>j||e<i) return 0;
   if(b>=i&&e<=j){
      return (int)t[n].size()-(UB(all(t[n]),k)-t[n].begin());
   }
   int stree;
   return query(l,b,mid,i,j,k)+query(r,mid+1,e,i,j,k);
}
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,q;
   cin>>n;
   for(i=1;i<=n;i++) cin>>a[i];
   build(1,1,n);
   for(i=1;i<=4*n;i++){
      cout<<i<<": ";
      for(auto x:t[i]) cout<<x<<' ';
      cout<<nl;
   }
   cin>>q;
   int ans=0;
   while(q--){
      cin>>i>>j>>k;
      i^=ans;
      j^=ans;
      k^=ans;
///online query
      ans=query(1,1,n,i,j,k);
      cout<<ans<<nl;
   }
   return 0;
}
```

21. SQRT Decomposition

```
/// number of elements greater than  k in range with update
int n,idx,block=400,a[mxn],t[200][10010];
void upd(int k,int i,int v)
{
   while(i<10010) t[k][i]+=v,i+=i&-i;
}
int query(int k,int i)
{
   int ans=0;
   while(i>0) ans+=t[k][i],i-=i&-i;
   return ans;
```

```
}
void build()
{
   for(int i=1;i<=n;i++){
      if(i%block==0) idx++;
      upd(idx,a[i],1);
   }
}
void upd(int i,int v)
{
   int ind=i/block;
   upd(ind,a[i],-1);
   upd(ind,v,1);
   a[i]=v;
}
int query(int l,int r,int k)
{
   int ans=0;
   while(l<=r&&l%block!=0) ans+=(a[l]>k),l++;
   while(l+block<=r) ans+=query(l/block,10010)-
query(l/block,k),l+=block;
   while(l<=r) ans+=(a[l]>k),l++;
   return ans;
}
int main()
{
   fast;
   int i,j,k,m,q,l,r,typ,v;
   cin>>n;
   for(i=1;i<=n;i++) cin>>a[i];
   build();
   cin>>q;
   while(q--){
      cin>>typ;
      if(typ==0){
         cin>>i>>v;
         upd(i,v);
      }
      else{
         cin>>l>>r>>k;
         cout<<query(l,r,k)<<nl;
      }
   }
   return 0;
}
```

22. SQRT Tree

SQRT Tree With Update

```
///Given an array a that contains n elements and the
///operation op that satisfies associative property:
///(x op y) op z=x op (y op z) is true for any x, y, z.

///The following implementation of Sqrt Tree can perform the
following operations:
///build in O(nloglogn),
///answer queries in O(1) and update an element in O(sqrt(n)).

#define SqrtTreeItem int///change for the type you want
```

```
SqrtTreeItem op(const SqrtTreeItem &a, const SqrtTreeItem &b)
{
   return a+b;///just change this operation for different problems,no change is required inside the code
}

inline int log2Up(int n) {
   int res = 0;
   while ((1 << res) < n) {
      res++;
   }
   return res;
}
///0-indexed
struct SqrtTree {
   int n, llg, indexSz;
   vector<SqrtTreeItem> v;
   vector<int> clz, layers, onLayer;
   vector< vector<SqrtTreeItem> > pref, suf, between;

   inline void buildBlock(int layer, int l, int r) {
      pref[layer][l] = v[l];
      for (int i = l+1; i < r; i++) {
         pref[layer][i] = op(pref[layer][i-1], v[i]);
      }
      suf[layer][r-1] = v[r-1];
      for (int i = r-2; i >= l; i--) {
         suf[layer][i] = op(v[i], suf[layer][i+1]);
      }
   }

   inline void buildBetween(int layer, int lBound, int rBound, int betweenOffs) {
      int bSzLog = (layers[layer]+1) >> 1;
      int bCntLog = layers[layer] >> 1;
      int bSz = 1 << bSzLog;
      int bCnt = (rBound - lBound + bSz - 1) >> bSzLog;
      for (int i = 0; i < bCnt; i++) {
         SqrtTreeItem ans;
         for (int j = i; j < bCnt; j++) {
            SqrtTreeItem add = suf[layer][lBound + (j << bSzLog)];
            ans = (i == j) ? add : op(ans, add);
            between[layer-1][betweenOffs + lBound + (i << bCntLog) + j] = ans;
         }
      }
   }

   inline void buildBetweenZero() {
      int bSzLog = (llg+1) >> 1;
      for (int i = 0; i < indexSz; i++) {
         v[n+i] = suf[0][i << bSzLog];
      }
      build(1, n, n + indexSz, (1 << llg) - n);
   }

   inline void updateBetweenZero(int bid) {
      int bSzLog = (llg+1) >> 1;
      v[n+bid] = suf[0][bid << bSzLog];
      update(1, n, n + indexSz, (1 << llg) - n, n+bid);
```

```
    }

    void build(int layer, int lBound, int rBound, int betweenOffs) {
        if (layer >= (int)layers.size()) {
            return;
        }
        int bSz = 1 << ((layers[layer]+1) >> 1);
        for (int l = lBound; l < rBound; l += bSz) {
            int r = min(l + bSz, rBound);
            buildBlock(layer, l, r);
            build(layer+1, l, r, betweenOffs);
        }
        if (layer == 0) {
            buildBetweenZero();
        } else {
            buildBetween(layer, lBound, rBound, betweenOffs);
        }
    }

    void update(int layer, int lBound, int rBound, int betweenOffs, int
x) {
        if (layer >= (int)layers.size()) {
            return;
        }
        int bSzLog = (layers[layer]+1) >> 1;
        int bSz = 1 << bSzLog;
        int blockIdx = (x - lBound) >> bSzLog;
        int l = lBound + (blockIdx << bSzLog);
        int r = min(l + bSz, rBound);
        buildBlock(layer, l, r);
        if (layer == 0) {
            updateBetweenZero(blockIdx);
        } else {
            buildBetween(layer, lBound, rBound, betweenOffs);
        }
        update(layer+1, l, r, betweenOffs, x);
    }

    inline SqrtTreeItem query(int l, int r, int betweenOffs, int base) {
        if (l == r) {
            return v[l];
        }
        if (l + 1 == r) {
            return op(v[l], v[r]);
        }
        int layer = onLayer[clz[(l - base) ^ (r - base)]];
        int bSzLog = (layers[layer]+1) >> 1;
        int bCntLog = layers[layer] >> 1;
        int lBound = (((l - base) >> layers[layer]) << layers[layer]) + base;
        int lBlock = ((l - lBound) >> bSzLog) + 1;
        int rBlock = ((r - lBound) >> bSzLog) - 1;
        SqrtTreeItem ans = suf[layer][l];
        if (lBlock <= rBlock) {
            SqrtTreeItem add = (layer == 0) ? (
                query(n + lBlock, n + rBlock, (1 << llg) - n, n)
            ) : (
                between[layer-1][betweenOffs + lBound + (lBlock <<
bCntLog) + rBlock]
            );
            ans = op(ans, add);
```

```
    }
    ans = op(ans, pref[layer][r]);
    return ans;
}

inline SqrtTreeItem query(int l, int r) {
    return query(l, r, 0, 0);
}

inline void update(int x, const SqrtTreeItem &item) {
    v[x] = item;
    update(0, 0, n, 0, x);
}

SqrtTree(const vector<SqrtTreeItem>& a)
    : n((int)a.size()), llg(log2Up(n)), v(a), clz(1 << llg), onLayer(llg+1) {
    clz[0] = 0;
    for (int i = 1; i < (int)clz.size(); i++) {
        clz[i] = clz[i >> 1] + 1;
    }
    int tllg = llg;
    while (tllg > 1) {
        onLayer[tllg] = (int)layers.size();
        layers.push_back(tllg);
        tllg = (tllg+1) >> 1;
    }
    for (int i = llg-1; i >= 0; i--) {
        onLayer[i] = max(onLayer[i], onLayer[i+1]);
    }
    int betweenLayers = max(0, (int)layers.size() - 1);
        int bSzLog = (llg+1) >> 1;
        int bSz = 1 << bSzLog;
        indexSz = (n + bSz - 1) >> bSzLog;
        v.resize(n + indexSz);
        pref.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
        suf.assign(layers.size(), vector<SqrtTreeItem>(n + indexSz));
        between.assign(betweenLayers, vector<SqrtTreeItem>((1 << llg)
+ bSz));
        build(0, 0, n, 0);
    }
};
int main()
{
    BeatMeScanf;
    int i,j,k,n,m,q,l,r;
    cin>>n;
    vi v;
    for(i=0;i<n;i++) cin>>k,v.eb(k);
    SqrtTree t=SqrtTree(v);
    cin>>q;
    while(q--){
        cin>>l>>r;
        --l,--r;
        cout<<t.query(l,r)<<nl;
    }
}
```

SQRT Tree Without Update

///Use Same strategy as before

```
///One can use previous code for without update but in onsite contest this is faster to code
int op(int a, int b)
{

}

inline int log2Up(int n)
{
   int res = 0;
   while ((1 << res) < n)
   {
      res++;
   }
   return res;
}

struct SqrtTree
{
   int n, llg;
   vector<int> v;
   vector<int> clz;
   vector<int> layers;
   vector<int> onLayer;
   vector< vector<int> > pref;
   vector< vector<int> > suf;
   vector< vector<int> > between;

   void build(int layer, int lBound, int rBound)
   {
      if (layer >= (int)layers.size())
      {
         return;
      }
      int bSzLog = (layers[layer]+1) >> 1;
      int bCntLog = layers[layer] >> 1;
      int bSz = 1 << bSzLog;
      int bCnt = 0;
      for (int l = lBound; l < rBound; l += bSz)
      {
         bCnt++;
         int r = min(l + bSz, rBound);
         pref[layer][l] = v[l];
         for (int i = l+1; i < r; i++)
         {
            pref[layer][i] = op(pref[layer][i-1], v[i]);
         }
         suf[layer][r-1] = v[r-1];
         for (int i = r-2; i >= l; i--)
         {
            suf[layer][i] = op(v[i], suf[layer][i+1]);
         }
         build(layer+1, l, r);
      }
      for (int i = 0; i < bCnt; i++)
      {
         int ans = 0;
         for (int j = i; j < bCnt; j++)
         {
            int add = suf[layer][lBound + (j << bSzLog)];
```

```
            ans = (i == j) ? add : op(ans, add);
            between[layer][lBound + (i << bCntLog) + j] = ans;
          }
        }
    }
    inline int query(int l, int r)
    {
        if (l == r)
        {
            return v[l];
        }
        if (l + 1 == r)
        {
            return op(v[l], v[r]);
        }
        int layer = onLayer[clz[l ^ r]];
        int bSzLog = (layers[layer]+1) >> 1;
        int bCntLog = layers[layer] >> 1;
        int lBound = (l >> layers[layer]) << layers[layer];
        int lBlock = ((l - lBound) >> bSzLog) + 1;
        int rBlock = ((r - lBound) >> bSzLog) - 1;
        int ans = suf[layer][l];
        if (lBlock <= rBlock)
        {
            ans = op(ans, between[layer][lBound + (lBlock << bCntLog) +
rBlock]);
        }
        ans = op(ans, pref[layer][r]);
        return ans;
    }
    SqrtTree(const vector<int>& v)
        : n((int)v.size()), llg(log2Up(n)), v(v), clz(1 << llg), onLayer(llg+1)
    {
        clz[0] = 0;
        for (int i = 1; i < (int)clz.size(); i++)
        {
            clz[i] = clz[i >> 1] + 1;
        }
        int tllg = llg;
        while (tllg > 1)
        {
            onLayer[tllg] = (int)layers.size();
            layers.push_back(tllg);
            tllg = (tllg+1) >> 1;
        }
        for (int i = llg-1; i >= 0; i--)
        {
            onLayer[i] = max(onLayer[i], onLayer[i+1]);
        }
        pref.assign(layers.size(), vector<int>(n));
        suf.assign(layers.size(), vector<int>(n));
        between.assign(layers.size(), vector<int>(1 << llg));
        build(0, 0, n);
    }
};
```

23. DSU on tree

```
// how many node have color u in subtree of u
```

```
vll g[N];
ll ans[N],col[N],sz[N],cnt[N];
bool big[N];
void dfs(ll u,ll pre)
{
   sz[u]=1;
   for(auto v:g[u]){
      if(v==pre) continue;
      dfs(v,u);
      sz[u]+=sz[v];
   }
}
void add(ll u,ll pre,ll x)
{
   cnt[col[u]]+=x;
   for(auto v:g[u]){
      if(v==pre||big[v]==1) continue;
      add(v,u,x);
   }
}
void dsu(ll u,ll pre,bool keep)
{
   ll bigchild=-1,mx=-1;
   for(auto v:g[u]){
      if(v==pre) continue;
      if(sz[v]>mx) mx=sz[v],bigchild=v;
   }
   for(auto v:g[u]){
      if(v==pre||v==bigchild) continue;
      dsu(v,u,0);
   }
   if(bigchild!=-1) dsu(bigchild,u,1),big[bigchild]=1;
   add(u,pre,1);
   ans[u]=cnt[u];
   if(bigchild!=-1) big[bigchild]=0;
   if(keep==0) add(u,pre,-1);
}
int main()
{
   fast;
   ll i,j,k,n,m,u,v;
   cin>>n;
   for(i=1;i<=n;i++) cin>>col[i];
   for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
   dfs(1,0);
   dsu(1,0,1);
   for(i=1;i<=n;i++) cout<<ans[i]<<nl;
   return 0;
}
```

24. Centroid Decomposition

Notes

We pick the centroid as the root r and find the number of paths passing through r. Then, the other paths won't pass through r, so we can remove r and split the tree into more subtrees, and recursively solve for each subtree as well.This is the basic solution relating to all pair nodes type problems.Sample code is in problem variation 2.

When we cannot to specifically get the answer for paths passing thorugh r but can answer for all pair of paths of subtree r then we can answer those question in the way of problem variation 3.
And problems like closest to some node can be solved in the way of problem variation 1.

Problem Variation 1

```
//root node is red, find closest red node from a node,all nodes can be blue or red after every update
vll g[N];
ll cenpar[N],sz[N],subtree_sz,dep[N],par[N][20],ans[N];
bool done[N];
//preprocessing part
void dfs(ll u,ll pre)
{
   dep[u]=dep[pre]+1;
   par[u][0]=pre;
   for(auto v:g[u]){
      if(v==pre) continue;
      dfs(v,u);
   }
}
ll lca(ll u,ll v)
{
   if(dep[u]<dep[v]) swap(u,v);
   for(ll k=19;k>=0;k--) if(dep[par[u][k]]>=dep[v])  u=par[u][k];
   if(u==v) return u;
   for(ll k=19;k>=0;k--) if(par[u][k]!=par[v][k]) u=par[u][k],v=par[v][k];
   return par[u][0];
}
ll dist(ll u,ll v)
{
   return dep[u]+dep[v]-2*dep[lca(u,v)];
}
//Decomposition part
void set_subtree_size(ll u,ll pre)
{
   subtree_sz++;
   sz[u]=1;
   for(auto v:g[u]){
      if(v==pre||done[v]) continue;
      set_subtree_size(v,u);
      sz[u]+=sz[v];
   }
}
ll get_centroid(ll u,ll pre)
{
   for(auto v:g[u]){
      if(v==pre||done[v]) continue;
      else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
   }
   return u;
}
void decompose(ll u,ll pre)
{
   subtree_sz=0;
   set_subtree_size(u,pre);
   ll centroid=get_centroid(u,pre);
   cenpar[centroid]=pre;
   done[centroid]=1;
```

```
    for(auto v:g[centroid]){
        if(v==pre||done[v]) continue;
        decompose(v,centroid);
    }
}
//query part
void upd(ll x)
{
    ll u=x;
    while(x){
        ans[x]=min(ans[x],dist(u,x));
        x=cenpar[x];
    }
}
ll query(ll x)
{
    ll ret=1e9,u=x;
    while(x){
        ret=min(ret,ans[x]+dist(u,x));
        x=cenpar[x];
    }
    return ret;
}
int main()
{
    fast;
    ll i,j,k,n,m,q,x,u,v,typ;
    cin>>n>>q;
    for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
    dfs(1,0);
    for(k=1;k<20;k++) for(i=1;i<=n;i++) par[i][k]=par[par[i][k-1]][k-1];
    decompose(1,0);
    //make sure to set ans as INF
    for(i=0;i<=n;i++) ans[i]=1e9;
    upd(1);
    while(q--){
        cin>>typ>>x;
        if(typ==1) upd(x);
        else cout<<query(x)<<nl;
    }
    return 0;
}
```

Problem Variation 2

```
//Given a tree and values of the nodes find the sum of all pair xor sum of nodes in the tree
vi g[N];
int cenpar[N],sz[N],subtree_sz;
bool done[N];

void set_subtree_size(int u,int pre)
{
    subtree_sz++;
    sz[u]=1;
    for(auto v:g[u]){
        if(v==pre||done[v]) continue;
        set_subtree_size(v,u);
        sz[u]+=sz[v];
    }
}
int get_centroid(int u,int pre)
```

```
{
   for(auto v:g[u]){
      if(v==pre||done[v]) continue;
      else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
   }
   return u;
}
vi vec;
int a[N];
void dfs(int u,int pre,int x)
{
   vec.eb(a[u]^x);
   for(auto v:g[u]){
      if(v==pre||done[v]) continue;
      dfs(v,u,a[u]^x);
   }
}
int one[30];
ll solve(int u,int pre)
{
   mem(one,0);
   for(int i=0;i<25;i++) if((a[u]>>i)&1) one[i]++;
   int tot=1;
   ll ans=0;
   for(auto v:g[u]){
      if(v==pre||done[v]) continue;
      vec.clear();
      dfs(v,u,0);
      for(auto x:vec){
         for(int i=0;i<25;i++){
            if((x>>i)&1) ans+=1LL*(tot-one[i])*(1<<i);
            else ans+=1LL*one[i]*(1<<i);
         }
      }
      for(auto x:vec){
         x^=a[u];
         for(int i=0;i<25;i++){
            if((x>>i)&1) one[i]++;
         }
         tot++;
      }
   }
//add answer for u to u path
   return ans+a[u];
}
ll decompose(int u,int pre)
{
   subtree_sz=0;
   set_subtree_size(u,pre);
   int centroid=get_centroid(u,pre);
   cenpar[centroid]=pre;
   done[centroid]=1;
   ll ans=solve(centroid,pre);
   for(auto v:g[centroid]){
      if(v==pre||done[v]) continue;
      ans+=decompose(v,centroid);
   }
   return ans;
}
```

```
int main()
{
   fast;
   int i,j,k,n,m,q,x,u,v,typ;
   cin>>n;
   for(i=1;i<=n;i++) cin>>a[i];
   for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
   cout<<decompose(1,0)<<nl;
   return 0;
}
```

Problem Variation 3

```
//number of paths havinf length<=mxlen and weight<=mxw
vpii g[N];
int cenpar[N],sz[N],subtree_sz;
bool done[N];

void set_subtree_size(int u,int pre)
{
   subtree_sz++;
   sz[u]=1;
   for(auto x:g[u]){
      int v=x.F;
      if(v==pre||done[v]) continue;
      set_subtree_size(v,u);
      sz[u]+=sz[v];
   }
}
int get_centroid(int u,int pre)
{
   for(auto x:g[u]){
      int v=x.F;
      if(v==pre||done[v]) continue;
      else if(sz[v]>subtree_sz/2) return get_centroid(v,u);
   }
   return u;
}
vpii vec;
void dfs(int u,int pre,int len,int w)
{
   vec.eb(w,len);
   for(auto x:g[u]){
      int v=x.F,we=x.S;
      if(v==pre||done[v]) continue;
      dfs(v,u,len+1,w+we);
   }
}
template <class T>
struct BIT
{
   ///1-indexed
        int sz;
        vector<T> t;

        void init(int n) ///max size of array
        {
                sz = n;
                t.assign(sz,0);
        }

        T query(int idx)
```

```
        {
                T ans = 0;
                for(; idx >= 1; idx -= (idx & -idx)) ans += t[idx];
                return ans;
        }

        void upd(int idx, T val)
        {
                if(idx <= 0) return;
                for(; idx <sz; idx += (idx & -idx)) t[idx] += val;
        }

        T query(int l, int r) { return query(r) - query(l - 1); }
};
BIT<int>t;
int mxlen,mxw;
ll solve(int u,int pre,int len,int w)
{
   vec.clear();
   dfs(u,pre,len,w);
   ll ans=0;
   srt(vec);
   for(auto x:vec) t.upd(x.S+1,1);
   ///how many pairs of sum <=(mxw,mxlen)
   int l=0,r=vec.size()-1;
   while(l<=r){
      t.upd(vec[l].S+1,-1);
      while(l<r&&vec[l].F+vec[r].F>mxw) t.upd(vec[r].S+1,-1),r--;
      ans+=t.query(mxlen-vec[l].S+1);
      l++;
   }
   return ans;
}
ll ans;
void decompose(int u,int pre)
{
   subtree_sz=0;
   set_subtree_size(u,pre);
   int centroid=get_centroid(u,pre);
   cenpar[centroid]=pre;
   done[centroid]=1;
   for(auto x:g[centroid]){
      int v=x.F;
      if(v==pre||done[v]) continue;
      decompose(v,centroid);
   }
   ///add answer for all pair of paths from this subtree
   ans+=solve(centroid,pre,0,0);
   for(auto x:g[centroid]){
      int v=x.F;
      if(v==pre||done[v]) continue;
      ///remove answer for all pair of paths from centroid's child's
subtree
      ans-=solve(v,centroid,1,x.S);
   }
   done[centroid]=0;
}
int main()
{
   fast;
```

```
    int i,j,k,n,m,q,x,u,v,w,typ;
    t.init(N);
    cin>>n>>mxlen>>mxw;
    for(i=2;i<=n;i++) cin>>u>>w,g[u].eb(i,w),g[i].eb(u,w);
    decompose(1,0);
    cout<<ans<<nl;
    return 0;
}
```

25. Heavy Light Decomposition

HLD Standard

```
//query on change a node value and sum of the path u to v
ll a[mxn],t[mxn*4],total_chain,ind,node[mxn],pos[mxn];
ll
par[mxn],son[mxn],chain_head[mxn],sz[mxn],dep[mxn],chain_no[m
xn];
vll g[mxn];
void build(ll n,ll b,ll e)
{
    if(b==e){
        t[n]=a[node[b]];
        return;
    }
    ll mid=(b+e)/2,l=2*n,r=2*n+1;
    build(l,b,mid);
    build(r,mid+1,e);
    t[n]=t[l]+t[r];//change when necessery
}
void upd(ll n,ll b,ll e,ll i,ll val)
{
    if(i<b||i>e) return;
    if(b==e&&b==i){
        t[n]=val;
        return;
    }
    ll mid=(b+e)/2,l=2*n,r=2*n+1;
    upd(l,b,mid,i,val);
    upd(r,mid+1,e,i,val);
    t[n]=t[l]+t[r];
}
ll query(ll n,ll b,ll e,ll i,ll j)
{
    if(b>j||i>e) return 0;
    if(b>=i&&e<=j) return t[n];
    ll mid=(b+e)/2,l=2*n,r=2*n+1;
    return query(l,b,mid,i,j)+query(r,mid+1,e,i,j);
}
void dfs(ll u,ll pre)
{
    ll mx=-1;
    sz[u]=1;
    for(auto v:g[u]){
        if(v==pre) continue;
        par[v]=u;
        dep[v]=dep[u]+1;
        dfs(v,u);
        sz[u]+=sz[v];
        if(sz[v]>mx) mx=sz[v],son[u]=v;
```

```
    }
}
void hld(ll u,ll pre)
{
    if(chain_head[total_chain]==-1) chain_head[total_chain]=u;
    pos[u]=++ind;
    node[ind]=u;
    chain_no[u]=total_chain;
    if(son[u]==-1) return;
    hld(son[u],u);
    for(auto v:g[u]){
        if(v==pre||v==son[u]) continue;
        total_chain++;
        hld(v,u);
    }
}
ll solve(ll u,ll v)
{
    ll ans=0;
    ll chain1=chain_no[u];
    ll chain2=chain_no[v];
    ll chd_u=chain_head[chain1];
    ll chd_v=chain_head[chain2];
    while(chd_u!=chd_v){
        if(dep[chd_u]<dep[chd_v]){
            swap(chd_u,chd_v);
            swap(u,v);
        }
        ans+=query(1,1,ind,pos[chd_u],pos[u]);//change when
necessery
        u=par[chd_u];
        chain1=chain_no[u];
        chd_u=chain_head[chain1];
    }
    if(dep[u]<dep[v]) swap(u,v);
    ans+=query(1,1,ind,pos[v],pos[u]);//for values given in edges
query here pos[v]+1 to pos[u] after array-fying the edges value
    return ans;
}
int main()
{
    fast;
    ll i,j,k,n,m,u,v,q,tt,w;
    cin>>n;
    for(i=1;i<=n;i++) cin>>a[i];
    for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
    mem(son,-1);
    mem(chain_head,-1);
    dfs(1,0);
    hld(1,0);
    build(1,1,ind);
    cin>>q;
    while(q--){
        cin>>tt;
        if(tt==1){
            cin>>u>>w;
            a[u]=w;
            upd(1,1,ind,pos[u],w);
        }
        else{
```

```
        cin>>u>>v;
        cout<<solve(u,v)<<nl;
      }
   }
   return 0;
}
```

HLD with Subtrees and Path Query

```
///add t value: Add value to all nodes in subtree rooted at t
///max a b: Report maximum value on the path from a to b
ll a[mxn],t[mxn*4],lazy[mxn*4],ind,node[mxn],st[mxn],en[mxn];
ll par[mxn],son[mxn],sz[mxn],dep[mxn],head[mxn];
vll g[mxn];
void build(ll n,ll b,ll e)
{
   if(b==e){
      t[n]=a[node[b]];
      return;
   }
   ll mid=(b+e)/2,l=2*n,r=2*n+1;
   build(l,b,mid);
   build(r,mid+1,e);
   t[n]=max(t[l],t[r]);
}
void propagate(ll n)
{
   if(lazy[n]==0) return;
   t[2*n]+=lazy[n];
   t[2*n+1]+=lazy[n];
   lazy[2*n]+=lazy[n];
   lazy[2*n+1]+=lazy[n];
   lazy[n]=0;
}
void upd(ll n,ll b,ll e,ll i,ll j,ll val)
{
   if(b>j||i>e) return;
   if(b>=i&&e<=j){
      t[n]+=val;
      lazy[n]+=val;
      return;
   }
   propagate(n);
   ll mid=(b+e)/2,l=2*n,r=2*n+1;
   upd(l,b,mid,i,j,val);
   upd(r,mid+1,e,i,j,val);
   t[n]=max(t[l],t[r]);
}
ll query(ll n,ll b,ll e,ll i,ll j)
{
   if(b>j||i>e) return -1e18;
   if(b>=i&&e<=j) return t[n];
   propagate(n);
   ll mid=(b+e)/2,l=2*n,r=2*n+1;
   return max(query(l,b,mid,i,j),query(r,mid+1,e,i,j));
}
void dfs(ll u,ll pre)
{
   par[u]=pre;
   dep[u]=dep[pre]+1;
   sz[u]=1;
   for(auto v:g[u]){
```

```
        if(v==pre) continue;
        dfs(v,u);
        sz[u]+=sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
}
void hld(ll u,ll pre)
{
    st[u]=++ind;
    node[st[u]]=u;
    if(son[par[u]]==u) head[u]=head[par[u]];
    else head[u]=u;
    if(son[u]) hld(son[u],u);
    for(auto v:g[u]){
        if(v==pre||v==son[u]) continue;
        hld(v,u);
    }
    en[u]=ind;
}
ll solve(ll u,ll v)
{
    ll ans=-1e18;
    while(head[u]!=head[v]){
        if(dep[head[u]]<dep[head[v]]){
            swap(u,v);
        }
        ans=max(ans,query(1,1,ind,st[head[u]],st[u]));///path query
        u=par[head[u]];
    }
    if(dep[u]>dep[v]) swap(u,v);
    ans=max(ans,query(1,1,ind,st[u],st[v]));
    return ans;
}
int main()
{
    fast;
    ll i,j,k,n,m,u,v,q,w;
    string tt;
    cin>>n;
    for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
    dfs(1,0);
    hld(1,0);
   // build(1,1,ind);
    cin>>q;
    while(q--){
        cin>>tt;
        if(tt[0]=='a'){
            cin>>u>>w;
            upd(1,1,ind,st[u],en[u],w);///subtree query
        }
        else{
            cin>>u>>v;
            cout<<solve(u,v)<<nl;
        }
    }
    return 0;
}
```

26. Treap

```
random_device rd;
```

```
mt19937 random(rd());///random generator
///If some compiler throws compilation error then
///use natural rand() function instead of mt19937

struct treap
{
   ///This is an implicit treap which investigates here on an array
        struct node
        {
                int val, sz, priority, lazy, sum,mx,mn,repl;
                bool repl_flag,rev;
                node *l, *r, *par;

                node() { lazy = 0; rev = 0; val = 0; sum=0;sz = 0;mx=0;mn=0;repl=0;repl_flag=0; priority = 0; l = NULL; r = NULL; par = NULL;}
                node(int _val)
                {
                        val = _val;
                        sum = _val;
                        mx=_val;
                        mn=_val;
                        repl=0;
                        repl_flag=0;
                        rev = 0;
                        lazy = 0;
                        sz = 1;
                        priority = random();
                        l = NULL;
                        r = NULL;
                        par = NULL;
                }
        };
        typedef node* pnode;
        pnode root;
        map<int, pnode> position;///positions of all the values
        ///clearing the treap
        void clear()
        {
                root = NULL;
                position.clear();
        }

        treap() { clear(); }

        int size(pnode t) { return t ? t->sz : 0; }
        void update_size(pnode &t) { if(t) t->sz = size(t->l) + size(t->r) + 1; }

        void update_parent(pnode &t)
        {
                if(!t) return;
                if(t->l) t->l->par = t;
                if(t->r) t->r->par = t;
        }
        ///add operation
    void lazy_sum_upd(pnode &t){
      if( !t or !t->lazy )    return;
```

```
    t->sum+=t->lazy*size(t);
    t->val += t->lazy;
    t->mx += t->lazy;
    t->mn+=t->lazy;
    if( t->l )  t->l->lazy += t->lazy;
    if( t->r )  t->r->lazy += t->lazy;
    t->lazy = 0;
}
///replace update
void lazy_repl_upd(pnode &t){
    if( !t or !t->repl_flag )   return;
    t->val = t->mx =t->mn= t->repl;
    t->sum = t->val*size(t);
    if( t->l ){
        t->l->repl = t->repl;
        t->l->repl_flag = true;
    }
    if( t->r ){
        t->r->repl = t->repl;
        t->r->repl_flag = true;
    }
    t->repl_flag = false;
    t->repl = 0;
}
///reverse update
void lazy_rev_upd(pnode &t){
    if( !t or !t->rev ) return;
    t->rev = false;
    swap(t->l, t->r);
    if( t->l )  t->l->rev ^= true;
    if( t->r )  t->r->rev ^= true;
  }
  ///reset the value of current node assuming it now
  ///represents a single element of the array
        void reset(pnode &t)
        {
           if(!t) return;
    t->sum = t->val;
    t->mx=t->val;
    t->mn=t->val;
  }
  ///combine node l and r to form t by updating corresponding
queries
        void combine(pnode &t, pnode l, pnode r)
        {
                if(!l) { t = r; return; }
                if(!r) { t = l; return; }
                ///Beware!!!Here t can be equal to l or r anytime
                ///i.e. t and (l or r) is representing same node
                ///so operation is needed to be done carefully
                ///e.g. if t and r are same then after t->sum=l->sum+r-
>sum operation,
                ///r->sum will be same as t->sum
                ///so BE CAREFUL
                t->sum = l->sum + r->sum;
                t->mx=max(l->mx,r->mx);
                t->mn=min(l->mn,r->mn);
        }
```

```
    ///perform all operations
    void operation(pnode &t)
    {
if( !t )    return;
reset(t);
lazy_rev_upd(t->l);
lazy_rev_upd(t->r);
lazy_repl_upd(t->l);
lazy_repl_upd(t->r);
lazy_sum_upd(t->l);
lazy_sum_upd(t->r);
combine(t, t->l, t);
combine(t, t, t->r);
    }
    ///split node t in l and r by key k
    ///so first k+1 elements(0,1,2,...k) of the array from node t
    ///will be splitted in left node and rest will be in right node
    void split(pnode t, pnode &l, pnode &r, int k, int add = 0)
    {
            if(t == NULL) { l = NULL; r = NULL; return; }
lazy_rev_upd(t);
lazy_repl_upd(t);
lazy_sum_upd(t);
            int idx  = add + size(t->l);
            if(idx <= k)
                    split(t->r, t->r, r, k, idx + 1), l = t;
            else
                    split(t->l, l, t->l, k, add), r = t;
            update_parent(t);
            update_size(t);
            operation(t);
    }
    ///merge node l with r in t
    void merge(pnode &t, pnode l, pnode r)
    {
lazy_rev_upd(l);
lazy_rev_upd(r);
lazy_repl_upd(l);
lazy_repl_upd(r);
lazy_sum_upd(l);
lazy_sum_upd(r);
            if(!l) { t = r; return; }
            if(!r) { t = l; return; }

            if(l->priority > r->priority)
                    merge(l->r, l->r, r), t = l;
            else
                    merge(r->l, l, r->l), t = r;

            update_parent(t);
            update_size(t);
            operation(t);
    }
    ///insert val in position a[pos]
    ///so all previous values from pos to last will be right shifted
    void insert(int pos, int val)
    {
```

```
            if(root == NULL)
            {
                pnode to_add = new node(val);
                root = to_add;
                position[val] = root;
                return;
            }

            pnode l, r, mid;
            mid = new node(val);
            position[val] = mid;

            split(root, l, r, pos - 1);
            merge(l, l, mid);
            merge(root, l, r);
        }
        ///erase from qL to qR indexes
        ///so all previous indexes from qR+1 to last will be left shifted
qR-qL+1 times
        void erase(int qL, int qR)
        {
            pnode l, r, mid;

            split(root, l, r, qL - 1);
            split(r, mid, r, qR - qL);
            merge(root, l, r);
        }
        ///returns answer for corresponding types of query
        int query(int qL, int qR)
        {
            pnode l, r, mid;

            split(root, l, r, qL - 1);
            split(r, mid, r, qR - qL);

            int answer = mid->sum;///for sum query
            ///int answer=mid->mx;///for max query
            ///int answer=mid->mn;///for min query
            merge(r, mid, r);
            merge(root, l, r);

            return answer;
        }
        ///add val in all the values from a[qL] to a[qR] positions
        void update(int qL, int qR, int val)
        {
            pnode l, r, mid;

            split(root, l, r, qL - 1);
            split(r, mid, r, qR - qL);
            lazy_repl_upd(mid);
            mid->lazy += val;

            merge(r, mid, r);
            merge(root, l, r);
        }
        ///reverse all the values from qL to qR
        void reverse(int qL, int qR)
```

```
{
        pnode l, r, mid;

        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);

        mid->rev ^= 1;
        merge(r, mid, r);
        merge(root, l, r);
}
///replace all the values from a[qL] to a[qR] by v
void replace(int qL, int qR,int v)
{
        pnode l, r, mid;

        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);
        lazy_sum_upd(mid);
        mid->repl_flag=1;
        mid->repl=v;
        merge(r, mid, r);
        merge(root, l, r);
}
///it will cyclic right shift the array k times
///so for k=1, a[qL]=a[qR] and all positions from ql+1 to qR will
///have values from previous a[qL] to a[qR-1]
///if you make left_shift=1 then it will to the opposite
void cyclic_shift(int qL, int qR, int k,bool left_shift=0)
{
        if(qL == qR) return;
        k %= (qR - qL + 1);

        pnode l, r, mid, fh, sh;
        split(root, l, r, qL - 1);
        split(r, mid, r, qR - qL);

        if(left_shift==0) split(mid, fh, sh, (qR - qL + 1) - k - 1);
        else split(mid,fh,sh,k-1);
        merge(mid, sh, fh);

        merge(r, mid, r);
        merge(root, l, r);
    }
    bool exist;
    ///returns index of node curr
    int get_pos(pnode curr, pnode son = nullptr)
    {
       if(exist==0) return 0;
       if(curr==NULL){
    exist=0;
    return 0;
       }
        if(!son)
        {
                if(curr == root) return size(curr->l);
                else  return  size(curr->l)  +  get_pos(curr->par,
curr);
        }
```

```
            if(curr == root)
            {
                if(son == curr->l) return 0;
                else return size(curr->l) + 1;
            }

            if(curr->l == son) return get_pos(curr->par, curr);
            else return get_pos(curr->par, curr) + size(curr->l) + 1;
        }
        ///returns index of the value
        ///if the value has multiple positions then it will
        ///return the last index where it was added last time
        ///returns -1 if it doesn't exist in the array
        int get_pos(int value)
        {
           if(position.find(value)==position.end()) return -1;
           exist=1;
    int x=get_pos(position[value]);
    if(exist==0) return -1;
    else return x;
}
///returns value of index pos
        int get_val(int pos)
        {
    return query(pos,pos);
        }
        ///returns size of the treap
        int size()
        {
           return size(root);
        }
        ///inorder traversal to get indexes chronologically
        void inorder(pnode cur)
        {
           if(cur==NULL) return;
           inorder(cur->l);
           cout<<cur->val<<' ';
           inorder(cur->r);
        }
        ///print current array values serially
        void print_array()
        {
//          for(int i=0;i<size();i++) cout<<get_val(i)<<' ';
//          cout<<nl;
      inorder(root);
      cout<<nl;
        }
        bool find(int val)
        {
           if(get_pos(val)==-1) return 0;
           else return 1;
        }
};

treap t;
///Beware!!!here treap is 0-indexed
```

```
int main()
{
        BeatMeScanf;
   int i,j,k,n,m,l,r,q;
   for(i=0;i<10;i++) t.insert(i,i*10);
   t.cyclic_shift(4,5,1);
   t.update(2,5,1);
   t.replace(2,5,100);
   t.reverse(2,9);
   t.replace(2,5,200);
   cout<<t.query(0,7)<<nl;
   t.cyclic_shift(2,3,2,1);
   cout<<t.get_pos(20)<<nl;
   t.erase(2,2);
   cout<<t.find(30)<<nl;
   t.print_array();
   return 0;
}
```

27. Wavelet Tree

```
const int MAX = 1e6;///if MAX is bigger than this compress the array
maybe
vi g[N];
int a[N];
struct wavelet_tree{
        int lo, hi;
        wavelet_tree *l, *r;
        vi b;
        vi c; /// c holds the prefix sum of elements

///array indices are [from, to)
wavelet_tree(int *from, int *to, int x, int y){
        lo = x, hi = y;
        if( from >= to) return;
        if( hi == lo ){
                b.reserve(to-from+1);
        b.pb(0);
                c.reserve(to-from+1);
        c.pb(0);
        for(auto it = from; it != to; it++){
                b.pb(b.back() + 1);
                c.pb(c.back()+*it);
        }
                return;
        }
        int mid = (lo+hi)/2;
        auto f = [mid](int x){
                return x <= mid;
        };
        b.reserve(to-from+1);
        b.pb(0);
        c.reserve(to-from+1);
        c.pb(0);
        for(auto it = from; it != to; it++){
                b.pb(b.back() + f(*it));
                c.pb(c.back() + *it);
        }
        auto pivot = stable_partition(from, to, f);
```

```
        l = new wavelet_tree(from, pivot, lo, mid);
        r = new wavelet_tree(pivot, to, mid+1, hi);
    }

    ///kth smallest element in [l, r]
    int kth(int l, int r, int k){
        if(l > r) return 0;
        if(lo == hi) return lo;
        int inLeft = b[r] - b[l-1];
        int lb = b[l-1];
        int rb = b[r];
        if(k <= inLeft) return this->l->kth(lb+1, rb , k);
        return this->r->kth(l-lb, r-rb, k-inLeft);
    }

    ///count of numbers in [l, r] Less than or equal to k
    int LTE(int l, int r, int k) {
        if(l > r or k < lo) return 0;
        if(hi <= k) return r - l + 1;
        int lb = b[l-1], rb = b[r];
        return this->l->LTE(lb+1, rb, k) + this->r->LTE(l-lb, r-rb,
k);
    }

    ///count of numbers in [l, r] equal to k
    int count(int l, int r, int k) {
        if(l > r or k < lo or k > hi) return 0;
        if(lo == hi) return r - l + 1;
        int lb = b[l-1], rb = b[r], mid = (lo+hi)/2;
        if(k <= mid) return this->l->count(lb+1, rb, k);
        return this->r->count(l-lb, r-rb, k);
    }

    ///sum of numbers in [l ,r] less than or equal to k
    int sumk(int l, int r, int k) {
        if(l > r or k < lo) return 0;
        if(hi <= k) return c[r] - c[l-1];
        int lb = b[l-1], rb = b[r];
        return this->l->sumk(lb+1, rb, k) + this->r->sumk(l-lb,
r-rb, k);
    }

    ~wavelet_tree(){
        delete l;
        delete r;
    }
};
int main()
{
    BeatMeScanf;
    int i,n,k,j,q,l,r;
    cin >> n;
    for(i=1;i<=n;i++) cin>>a[i];
    wavelet_tree t(a+1, a+n+1, 1, MAX);///here MAX is maximum
element in array
    cin >> q;
    while(q--){
        int x;
```

```
		cin>>x;
		cin >> l >> r >> k;
		if(x == 0){
			///kth smallest
			cout << t.kth(l, r, k) << endl;
		}
		else if(x == 1){
			///less than or equal to K
			cout <<t.LTE(l, r, k) << endl;
		}
		else if(x == 2){
			///count occurence of K in [l, r]
			cout << t.count(l, r, k) << endl;
		}
		if(x == 3){
			///sum of elements less than or equal to K in [l,
r]
			cout << t.sumk(l, r, k) << endl;
		}
	}
	return 0;
}
```

28. K-D tree

///Complexity: $O(\log n)$
///Average Complexity: $O(3^d \log n)$,$where\ d\ is\ dimension$
///Works for random points
///search for nearest point which has minimum euclidean distance
///from this point

```
const long long INF = 2000000000000000007;
const int d=2;///dimension

struct point {
	int p[d];
	bool operator !=(const point &a) const {
		bool ok=1;
		for(int i=0;i<d;i++) ok&=(p[i]==a.p[i]);
		return !ok;
	}
};

struct kd_node {
	int axis,value;
	point p;
	kd_node *left, *right;
};

struct cmp_points {
	int axis;
	cmp_points(){}
	cmp_points(int x): axis(x) {}
	bool operator () (const point &a, const point &b) const {
		return a.p[axis]<b.p[axis];
	}
};

typedef kd_node* node_ptr;
```

```
int tests,n;
point arr[N],pts[N];
node_ptr root;
long long ans;

long long squared_distance(point a, point b) {
	long long ans=0;
	for(int i=0;i<d;i++) ans+=(a.p[i]-b.p[i])*1ll*(a.p[i]-b.p[i]);
	return ans;
}

void build_tree(node_ptr &node, int from, int to, int axis) {
	if(from>to) {
		node=NULL;
		return;
	}

	node=new kd_node();

	if(from==to) {
		node->p=arr[from];
		node->left=NULL;
		node->right=NULL;
		return;
	}

	int mid=(from+to)/2;

	nth_element(arr+from,arr+mid,arr+to+1,cmp_points(axis));
	node->value=arr[mid].p[axis];
	node->axis=axis;
	build_tree(node->left,from,mid,(axis+1)%d);
	build_tree(node->right,mid+1,to,(axis+1)%d);
}

void nearest_neighbor(node_ptr node, point q, long long &ans) {
	if(node==NULL) return;

	if(node->left==NULL && node->right==NULL) {
		if(q!=node->p)   ans=min(ans,squared_distance(node-
>p,q));///Beware!!!need to take care here
		return;
	}

	if(q.p[node->axis]<=node->value) {
		nearest_neighbor(node->left,q,ans);
		if(q.p[node->axis]+sqrt(ans)>=node->value)
nearest_neighbor(node->right,q,ans);
	}

	else {
		nearest_neighbor(node->right,q,ans);
		if(q.p[node->axis]-sqrt(ans)<=node->value)
nearest_neighbor(node->left,q,ans);
	}
}

int main() {
```

```
    int i,j,k,m;
        scanf("%d", &tests);
        while(tests--) {
            scanf("%d", &n);
            for(i=1;i<=n;i++) {
                for(j=0;j<d;j++) scanf("%d",&arr[i].p[j]);
                pts[i]=arr[i];
            }

            build_tree(root,1,n,0);

            for(i=1;i<=n;i++) {
                ans=INF;
                nearest_neighbor(root,pts[i],ans);
                printf("%lld\n", ans);
            }
        }

        return 0;
}
```

29. Link-Cut Tree

```
random_device rd;
mt19937_64 mt(rd());

struct node
{
        int sz, prior, id, rev;
        node *par, *pp, *l, *r;
        node() { id = 0; sz = 0; rev = 0; prior = 0; par = NULL; l = NULL;
r = NULL;pp=NULL; }
        node(int v) { id = v; sz = 1; rev = 0; prior = mt(); l = NULL; r =
NULL;par=NULL;pp=NULL; }
};

typedef node* pnode;

inline int size(pnode v) { return v ? v->sz : 0; }
void push(pnode &t)
{
        if(!t) return;
        if(t->rev)
        {
            swap(t->l, t->r);
            if(t->l) t->l->rev ^= 1;
            if(t->r) t->r->rev ^= 1;
            t->rev = 0;
        }
}

void pull(pnode &v)
{
        if(!v) return;

        push(v->l);
        push(v->r);

        v->par = NULL;
```

```
	v->sz = size(v->l) + size(v->r) + 1;

	if(v->l) v->l->par = v;
	if(v->r) v->r->par = v;

	if(v->l && v->l->pp) v->pp = v->l->pp, v->l->pp = NULL;
	if(v->r && v->r->pp) v->pp = v->r->pp, v->r->pp = NULL;
}

void merge(pnode &t, pnode l, pnode r)
{
	push(l), push(r);
	if(!l) { t = r; return; }
	if(!r) { t = l; return; }

	if(l->prior > r->prior)
		merge(l->r, l->r, r), t = l;
	else
		merge(r->l, l, r->l), t = r;

	pull(t);
}

void split(pnode t, pnode &l, pnode &r, int k, int add = 0)
{
	push(t);
	if(!t) { l = NULL; r = NULL; return; }

	int idx = add + size(t->l);
	if(idx <= k)
		split(t->r, t->r, r, k, idx + 1), l = t;
	else
		split(t->l, l, t->l, k, add), r = t;

	pull(t);
}

pnode get_root(pnode t) { if(!t) return NULL; while(t->par) t = t->par;
return t; }

pnode remove_right(pnode t)
{
	pnode rt = t;

	int pos = size(rt->l);
	if(rt->rev) pos = size(rt) - pos - 1;
	while(rt->par)
	{
		if(rt->par->r == rt) pos += size(rt->par->l) + 1;
		if(rt->par->rev) pos = size(rt->par) - pos - 1;
		rt = rt->par;
	}

	pnode l, r, pp = rt->pp;
	rt->pp = NULL;
	split(rt, l, r, pos);

	l->pp = pp;
```

```
	if(r) r->pp = t;

	return l;
}

pnode remove_left(pnode t)
{
	pnode rt = t;

	int pos = size(rt->l);
	if(rt->rev) pos = size(rt) - pos - 1;
	while(rt->par)
	{
		if(rt->par->r == rt) pos += size(rt->par->l) + 1;
		if(rt->par->rev) pos = size(rt->par) - pos - 1;
		rt = rt->par;
	}

	pnode l, r, pp = rt->pp;
	rt->pp = NULL;
	split(rt, l, r, pos - 1);

	l->pp = pp;
	return r;
}

pnode merge_trees(pnode u, pnode t)
{
	u = get_root(u);
	t = get_root(t);
	t->pp = NULL;
	merge(u, u, t);
	return u;
}

struct link_cut_tree
{
	pnode ver[N];

	pnode access(pnode t)
	{
		t = remove_right(t);
		while(t->pp)
		{
			pnode u = t->pp;
			u = remove_right(u);
			t = merge_trees(u, t);
		}

		return t;
	}

	pnode find_root(pnode u)
	{
		u = access(u);
		push(u); while(u->l) u = u->l, push(u);
		access(u);
		return u;
```

```
}

void make_root(pnode u)
{
	u = access(u);
	u->rev ^= 1;
	push(u);
}

void link(pnode u, pnode w)
{
	make_root(u);
	access(w);
	merge_trees(w, u);
}

void cut(pnode p)
{
	access(p);
	remove_left(p);
}

int depth(pnode u)
{
	u = access(u);
	return size(u);
}

pnode lca(pnode u, pnode v)
	{
		if(u == v) return u;
		if(depth(u) > depth(v)) swap(u, v);

		access(v);
		access(u);

		return get_root(v)->pp;
	}

	///creating vertices of the tree
void init(int c)
{
   for(int i = 0; i <= c; i++) ver[i] = new node(i);
}
///returns lca of two vertices
inline int lca(int u, int v)
{
   return lca(ver[u], ver[v])->id;
}
///finds the root of tree which has node u
inline int root(int u)
{
   return find_root(ver[u])->id;
}
///add an edge from vertex v to u, making u a child of v,
///where initially u and v are in different trees.
inline void link(int u, int v) ///add directed edge v to u
{
```

```
        link(ver[u], ver[v]);
    }
    ///make u the root of its representative tree
    inline void make_root(int u)
    {
        make_root(ver[u]);
    }
    ///depth of vertex u in its own tree
    inline int depth(int u)
    {
        return depth(ver[u]);
    }
    ///remove edge from u to its parent, where u is a non-root vertex.
    inline void cut(int u)
    {
        cut(ver[u]);
    }
};


link_cut_tree lct;

int main()
{
    BeatMeScanf;
        int i,j,k,n,m,u,v;
        cin>>n>>m;
        lct.init(n);
        while(m--)
        {
            string type;
            cin >> type;

            if(type == "add") ///add an edge
            {
                int u, w;
                cin >> u >> w;
                lct.link(u, w);
            }
            else if(type == "conn") ///if u and v is connected
            {
                int u, v;
                cin >> u >> v;
                cout << (lct.root(u) == lct.root(v) ? "YES" :
"NO") << endl;
            }
            else if(type == "rem") ///remove edge
            {
                int u, v;
                cin >> u >> v;
                if(lct.depth(u) > lct.depth(v)) swap(u, v);
                lct.cut(v);
            }
        }
        return 0;
}
```

DYNAMIC PROGRAMMING

30. Digit DP

Count of Numbers

```
///count of numbers x such that l<=x<=r ans distinct digit in x=max digit in x
vll digit;
ll sz,dp[20][1111][2],cnt[1111],mx[1111];
ll yo(ll idx,ll mask,bool badha)
{
   if(idx==-1){
      if(cnt[mask]==mx[mask]) return 1;
      else return 0;
   }
   ll &ret=dp[idx][mask][badha];
   if(ret!=-1&&badha!=1) return ret;
   ll ans=0;
   ll mxhere=badha?digit[idx]:9;
   for(ll i=0;i<=mxhere;i++){
      bool next_badha=(i==digit[idx]?badha:0);
      ans+=yo(idx-
1,(mask==0&&i==0)?mask:mask|(1LL<<i),next_badha);
   }
   if(badha==0) ret=ans;
   return ans;
}
ll get(ll n)
{
   if(n<0) return 0;
   digit.clear();
   while(n) digit.pb(n%10),n/=10;
   sz=digit.size();
   return yo(sz-1,0,1);
}
int main()
{
   fast;
   ll i,j,k,n,m,l,r,t;
   mem(dp,-1);
   cnt[0]=1;
   for(i=0;i<1111;i++){
      for(j=0;j<=10;j++) if(i&(1<<j)) cnt[i]++,mx[i]=j;
   }
   cin>>t;
   while(t--){
      cin>>l>>r;
      cout<<get(r)-get(l-1)<<nl;
   }
   return 0;
}
```

Sum of Numbers

```
///sum of numbers x such that l<=x<=r and distinct digit in x<=k
vll digit;
bool vis[20][1111][2];
ll sz;
pll dp[20][1111][2];
```

```
ll cnt[1111],pw[30],k;
pll yo(ll idx,ll mask,bool badha)
{
   if(idx==-1){
      if(cnt[mask]<=k) return MP(0,1);
      else return MP(0,0);
   }
   pll &ret=dp[idx][mask][badha];
   bool &x=vis[idx][mask][badha];
   if(x==1&&badha!=1) return ret;
   pll ans={0,0};
   ll mxhere=badha?digit[idx]:9;
   for(ll i=0;i<=mxhere;i++){
      bool next_badha=(i==digit[idx]?badha:0);
      pll p=yo(idx-
1,(mask==0&&i==0)?mask:mask|(1LL<<i),next_badha);
      if(p.S==0) continue;
      ans.F+=(p.F+pw[idx]*i%mod*p.S%mod)%mod;
      ans.F%=mod;
      ans.S+=p.S;
      ans.S%=mod;
   }
   if(badha==0) x=1,ret=ans;
   return ans;
}
ll get(ll n)
{
   if(n<0) return 0;
   digit.clear();
   while(n) digit.pb(n%10),n/=10;
   sz=digit.size();
   return yo(sz-1,0,1).F%mod;
}
int main()
{
   fast;
   ll i,j,n,m,l,r,t;
   pw[0]=1;
   for(i=1;i<30;i++) pw[i]=10LL*pw[i-1]%mod;
   cnt[0]=1;
   for(i=0;i<1111;i++){
      for(j=0;j<=10;j++) if(i&(1<<j)) cnt[i]++;
   }
   cin>>t;
   while(t--){
      cin>>l>>r>>k;
      cout<<((get(r)-get(l-1))%mod+mod)%mod<<nl;
   }
   return 0;
}
```

CONVEX HULL TRICK

Convex Hull Trick Standard

```
ll dp[120000];
struct cline {
   ll M, C;
   cline() {}
```

```
    cline(ll m, ll c): M(m), C(c) {}
};
int last=0,pointer=0;

//pointer=0,last=0 should be made initially
cline line[N]; //y=mx+c we need only m(slope) and c(constant)

//Returns true if either line l1 or line l3 is always better than line l2
bool bad(const cline & l1,const cline & l2,const cline & l3) {
    /*
    intersection(l1,l2) has x-coordinate (c1-c2)/(m2-m1)
    intersection(l1,l3) has x-coordinate (c1-c3)/(m3-m1)
    set the former greater than the latter, and cross-multiply to
    eliminate division
    */
    //if the query x values is non-decreasing (reverse(> sign) for vice
verse)
    return       (double)(l3.C-l1.C)*(double)(l1.M-l2.M)<=(double)(l2.C-
l1.C)*(double)(l1.M-l3.M);
}

//Adding should be done serially
//If we want minimum y coordinate(value) then maximum valued m
should be inserted first
//If we want maximum y coordinate(value) then minimum valued m
should be inserted first
void add(cline l) {
    //First, let's add it to the end
    line[last++]=l;
    //If the penultimate is now made irrelevant between the
antepenultimate
    //and the ultimate, remove it. Repeat as many times as necessary
    //in short convex hull main convex hull tecnique is applied here
    while(last>=3&&bad(line[last-3],line[last-2],line[last-1])) {
        line[last-2]=line[last-1];
        last--;
    }
}

//Returns the minimum y-coordinate of any intersection between a
given vertical
//line(x) and the lower/upper envelope(pointer)
//This can only be applied if the query of vertical line(x) is already
sorted
//works better if number of query is huge
long long query(long long x) {
    //If we removed what was the best line for the previous query, then
the
    //newly inserted line is now the best for that query
    if (pointer>=last)
        pointer=last-1;
    //Any better line must be to the right, since query values are
    //non-decreasing
    // Min Value wanted... (reverse(> sign) for max value)
    while                (pointer<last-1                &&
line[pointer+1].M*x+line[pointer+1].C<=line[pointer].M*x+line[poin
ter].C)
        pointer++;
```

```
    return line[pointer].M*x+line[pointer].C;
}

//for any kind of query(sorted or not) it can be used
//it works because of the hill property
//works better if number of query is few
long long bs(int st,int end,long long x,int last) {
    int mid=(st+end)/2;
    // Min Value wanted... (reverse(> sign) for max value)
    if(mid+1<last &&
line[mid+1].M*x+line[mid+1].C<line[mid].M*x+line[mid].C) return
bs(mid+1,end,x,last);
    // Min Value wanted... (reverse(> sign) for max value)
    if(mid-1>=0 && line[mid-1].M*x+line[mid-
1].C<line[mid].M*x+line[mid].C) return bs(st,mid-1,x,last);
    return line[mid].M*x+line[mid].C;
}
int b[120000],ara[1200000];
int main() {
    int i,j,k,l,m,n;
    scanf("%d",&n);
    for(i=0;i<n;i++){
        scanf("%d",&ara[i]);
    }
    for(i=0;i<n;i++)cin>>b[i];
    cline gr;
    ll ans=0;
    gr.M=b[0];
    gr.C=0;
    add(gr);
    for(int i=1;i<n;i++){
        ans=query(ara[i]);
        gr.M=b[i];
        gr.C=ans;
        add(gr);
    }
    cout<<ans<<endl;
    return 0;
}
```

Dynamic Convex Hull Trick

```
/// Keeps upper hull for maximums.
/// add lines with -m and -b and return -ans to
/// make this code working for minimums.
const ll is_query = -(1LL<<62);
struct line {
    ll m, b;
    mutable function<const line*()> succ;
    bool operator<(const line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
///Dynamic Convex Hull Trick
```

```
struct CHT : public multiset<line> { /// will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return 1.0*(x->b - y->b)*(z->m - y->m) >= 1.0*(y->b - z->b)*(y->m - x->m);
    }
    void add(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll query(ll x) {
        auto l = *lower_bound((line) { x, is_query });
        return l.m * x + l.b;
    }
};
ll a[N],b[N];
CHT* x;
int main()
{
    fast;
    ll i,j,k,n,m,ans=0;
    cin>>n;
    for(i=0;i<n;i++) cin>>a[i];
    for(i=0;i<n;i++) cin>>b[i];
    x=new CHT();
    x->add(-b[0],0);
    for(i=1;i<n;i++){
        ans=-x->query(a[i]);
        x->add(-b[i],-ans);
    }
    cout<<ans<<nl;
    return 0;
}
```

Persistent Convex Hull Trick

```
///You can only remove last added line with this code
const ll nsz=5e4+9;//maximum number of lines
ll msz;//make it 0 for restarting the CHT
ll outside = nsz-1;
ll M[nsz], B[nsz]; // y = M*X + B formatted lines, must be sorted in advanced by M //clear M, B for test cases, make qptr = 0
bool bad(int l1, int l2, int l3, bool lowerPart = 1) // returns true if l1-l3 line is better than l2
{
    /*
    intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
    intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
```

```
   */
 //   cout << (B[l3]-B[l1])*(M[l1]-M[l2]) << " " << (B[l2]-B[l1])*(M[l1]-
M[l3]) << endl;
   if (lowerPart == 1)
      return 1.00*(B[l3]-B[l1])*(M[l1]-M[l2]) <= 1.00*(B[l2]-
B[l1])*(M[l1]-M[l3]);
   else return 1.00*(B[l3]-B[l1])*(M[l1]-M[l2]) >= 1.00*(B[l2]-
B[l1])*(M[l1]-M[l3]);
}
struct data //information to undo change in CHT
{
   ll m, b, pos;
   data(ll _m = 0, ll _b = 0, ll _pos = 0)
   {
      m = _m, b = _b, pos = _pos;
   }
};
data add(ll m, ll b, bool lowerPart = 1)
{
// lowerPart is called upper hull. For m decreasing, this creates lower
part, but if m increasing, it does reverse
// lower part is needed for finding minimum, upper part for maximum
   M[outside] = m, B[outside] = b;
   while (msz >= 2 && bad(msz-2, msz-1, outside, lowerPart))
   {
      msz--;
   }
   data temp(M[msz], B[msz], msz);
   M[msz] = m;
   B[msz] = b;
   msz++;
   return temp;
}
ll query(ll x, bool findMin = 1) //online query
{
   int lo = 0, hi = msz - 1;
   ll ans = LLONG_MAX;
   if (findMin)
      ans = -LLONG_MAX;
   while(lo <= hi)
   {
      int diff = (hi-lo)/3;
      int mid1 = lo + diff;
      int mid2 = hi - diff;
      ll y1 = M[mid1]*x + B[mid1], y2 = M[mid2]*x + B[mid2];
      if(y1 <= y2)
      {
         ans = y1;
         if (findMin)
            hi = mid2 - 1;
         else lo = mid1 + 1;
      }
      else
      {
         ans = y2;
         if (findMin)
            lo = mid1 + 1;
         else hi = mid2 - 1;
```

```
        }
    }
    return ans;
}
ll sum[N],val[N],a,b,ans;
vll g[N];
void dfs(ll u,ll pre=0)
{
    sum[u]=val[u];
    for(auto v:g[u]){
        if(v==pre) continue;
        dfs(v,u);
        sum[u]+=sum[v];
    }
}
void yo(ll u,ll pre=0)
{
    bool leaf=1;
    for(auto v:g[u]){
        if(v==pre) continue;
        leaf=0;
        ll res=query(sum[v])+a*sum[v]*sum[v]+b;
        ans=min(ans,res+a*sum[v]*sum[v]+b);
        ll prvsz=msz;
        data undo=add(-2*a*sum[v],a*sum[v]*sum[v]+res,0);
        yo(v,u);
        msz=prvsz;
        M[undo.pos]=undo.m;
        B[undo.pos]=undo.b;
    }
    //if(leaf) ans=min(ans,query(0)+b);
}
int main()
{
    fast;
    ll i,j,k,n,m,t,u,v;
    cin>>t;
    while(t--){
        cin>>n>>a>>b;
        for(i=1;i<=n;i++) cin>>val[i];
        for(i=1;i<n;i++){
            cin>>u>>v;
            g[u].eb(v);
            g[v].eb(u);
        }
        dfs(1);
        ans=a*sum[1]*sum[1]+b;
        msz=0;
        add(-2*a*sum[1],a*sum[1]*sum[1]);
        yo(1);
        cout<<ans<<nl;
        mem(sum,0);
        for(i=1;i<=n;i++) g[i].clear();
    }
    return 0;
}
```

31. Divide and Conquer Optimization

```
///Divide 1,2,3...n people in k consecutive parts so that sum of cost of each individual part is minimum
int a[N][N],c[N][N],dp[810][N];///dp[i][j]=minimum cost for dividing [1...j] in i parts
int cost(int i,int j)
{
   if(i>j) return 0;
   return c[j][j]-c[i-1][j]-c[j][i-1]+c[i-1][i-1];
}
void yo(int i,int l,int r,int optl,int optr)
{
   if(l>r) return;
   int mid=(l+r)/2;
   dp[i][mid]=2e9;
   int opt=-1;
   for(int k=optl;k<=min(mid,optr);k++){
      int c=dp[i-1][k]+cost(k+1,mid);
      if(c<dp[i][mid]){
         dp[i][mid]=c;
         opt=k;
      }
   }
   ///for opt[1..j]<=opt[1...j+1]
   yo(i,l,mid-1,optl,opt);
   yo(i,mid+1,r,opt,optr);
   ///for opt[1...j]>=opt[1...j+1]
   ///yo(i,l,mid-1,opt,optr);
   ///yo(i,mid+1,r,optl,opt);

}
int main()
{
   BeatMeScanf;
   int i,j,k,n,m;
   n=sc();
   k=sc();
   for(i=1;i<=n;i++) for(j=1;j<=n;j++) a[i][j]=sc();
   for(i=1;i<=n;i++){
      for(j=1;j<=n;j++){
         c[i][j]=a[i][j]+c[i-1][j]+c[i][j-1]-c[i-1][j-1];
      }
   }
   for(i=1;i<=n;i++) dp[1][i]=cost(1,i);
   for(i=2;i<=k;i++) yo(i,1,n,1,n);
   cout<<dp[k][n]/2<<nl;
   return 0;
}
```

STRING THEORY

32. Trie max/min

```
struct node
{
   node* next[2];
```

```
    node()
    {
        next[0]=next[1]=NULL;
    }
}*root;
int sum[mxn];
void insert_num(int x)
{
    node* cur=root;
    int b;
    for(int i=31;i>=0;i--){
        b=(x>>i)&1;
        if(cur->next[b]==NULL) cur->next[b]=new node();
        cur=cur->next[b];
    }
}
int get_max(int x)
{
    node* cur=root;
    int k,ans=0;
    for(int i=31;i>=0;i--){
        k=(x>>i)&1;
        if(cur->next[!k]) cur=cur->next[!k],ans<<=1,ans++;
        else cur=cur->next[k],ans<<=1;
    }
    return ans;
}
int get_min(int x)
{
    node* cur=root;
    int k,ans=0;
    for(int i=31;i>=0;i--){
        k=(x>>i)&1;
        if(cur->next[k]) cur=cur->next[k],ans<<=1;
        else cur=cur->next[!k],ans<<=1,ans++;
    }
    return ans;
}
void del(node* cur)
{
    for(int i=0;i<2;i++) if(cur->next[i]) del(cur->next[i]);
    delete(cur);
}
int main()
{
    //fast;
    int i,j,k,n,m,t,cs=0;
    sf(t);
    while(t--){
        root= new node();
        int mx=-inf,mn=inf;
        sf(n);
        sum[0]=0;
        for(i=1;i<=n;i++) sf(k),sum[i]=sum[i-1]^k;
        insert_num(0);
        for(i=1;i<=n;i++){
            mx=max(mx,get_max(sum[i]));
            mn=min(mn,get_min(sum[i]));
```

```
            insert_num(sum[i]);
        }
        _ccase;
        pf("%d %d\n",mx,mn);
        for(i=1;i<=n;i++) sum[i]=0;
        del(root);
    }
    return 0;
}
```

33. String Matching

Knuth-Morris-Pratt

```
///returns the longest proper prefix array of pattern p
///where lps[i]=longest proper prefix which is also suffix of p[0...i]
vector<int> build_lps(string p)
{
    int sz = p.size();
    vector<int> lps;
    lps.assign(sz + 1, 0);
    int j = 0;
    lps[0] = 0;
    for(int i = 1; i < sz; i++)
    {
        while(j >= 0 && p[i] != p[j])
        {
            if(j >= 1) j = lps[j - 1];
            else j = -1;
        }
        j++;
        lps[i] = j;
    }

    return lps;
}
vector<int>ans;
///returns matches in vector ans in 0-indexed
void kmp(vector<int> lps, string s, string p)
{
        int psz = p.size(), sz = s.size();
        int j = 0;
        for(int i = 0; i < sz; i++)
        {
                while(j >= 0 && p[j] != s[i])
                        if(j >= 1) j = lps[j - 1];
                        else j = -1;

                j++;
                if(j == psz)
                {
                        j = lps[j - 1];
                        ///pattern found in string s at position i-psz+1
        ans.eb(i-psz+1);
                }
                ///after each loop we have j=longest common suffix of
s[0..i] which is also prefix of p
        }
}
```

```
int main()
{
          int i,j,k,n,m,t;
          cin>>t;
          while(t--){
      string s,p;
      cin>>s>>p;
      vector<int>lps = build_lps(p);
      kmp(lps, s, p);
      if(ans.empty()) cout<<"Not Found\n";
      else{
         cout<<ans.size()<<nl;
         for(auto x:ans) cout<<x<<' ';
         cout<<nl;
      }
      ans.clear();
      cout<<nl;
          }
          return 0;
}
```

Bitset

///Complexity: $\frac{n^2}{64}$

```
vi v;
bitset<N>bs[26],oc;
int main()
{
   fast;
   int i,j,k,n,q,l,r;
   string s,p;
   cin>>s;
   for(i=0;s[i];i++) bs[s[i]-'a'][i]=1;
   cin>>q;
   while(q--){
      cin>>p;
      oc.set();
      for(i=0;p[i];i++) oc&=(bs[p[i]-'a']>>i);
      cout<<oc.count()<<nl;///number of occurences
      int ans=N,sz=p.size();
      int pos=oc._Find_first();
      v.pb(pos);
      pos=oc._Find_next(pos);
      while(pos<N){
         v.pb(pos);
         pos=oc._Find_next(pos);
      }
      for(auto x:v) cout<<x<<' ';///position of occurences
      cout<<nl;
      v.clear();
      cin>>l>>r;///number of occurences from l to r,where l and r is 1-
indexed
      if(sz>r-l+1) cout<<0<<nl;
      else cout<<(oc>>(l-1)).count()-(oc>>(r-sz+1)).count()<<nl;
   }
   return 0;
}
```

Z-Algorithm

```
///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;
}
///for pattern searching use P$T version (P=pattern,T=text).
///for text indexes if (z[i]==pattern length) then pattern is found from ///that index
int main()
{
    fast;
    ll i,j,k,n,m;
    string s;
    cin>>s;
    vi ans=z_function(s);
    for(auto x:ans) cout<<x<<' ';
    return 0;
}
```

Aho Corasick

Time Complexity:$O(n)$

Space Complexity for index of all occurrences:$O(m\sqrt{m}), where\ m = sum\ of\ all\ patterns$

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

```
        {
                for(int i = 0; i < psz; i++)
                        is_end[i] = 0,cnt[i]=0,path[i]=0, ind[i]=0,link[i]
= 0, to[i].clear(),ending[i].clear(),oc[i].clear();

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

        aho_corasick() { psz = N - 2; clear(); }

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

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

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

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

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

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

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

                                while(j != -1 && !to[j].count(c)) j =
link[j];
                                if(j != -1) link[v] = to[j][c];
```

```
                                else link[v] = 0;

                                q.push(v);
                                populate(v);
                                path[len++]=v;
                        }
                }
        }
void populate(vi &en, int cur)
{
   /// occurrences of patterns in the given string
   for(auto idx: en)
   {
      oc[idx].eb(cur);
   }
}
        void traverse(string s)
{
   int n=s.size();
   int cur=0;///root

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

aho_corasick t;
string p[N];
int main()
{
        BeatMeScanf;
        int i,j,k,n,m;
        string s;
        cin>>s;
        cin>>m;
   for(i=0;i<m;i++){
      cin>>p[i];
      t.add_word(p[i],i);
   }
   t.push_links();
   t.traverse(s);
   ///print all occurrences
   for(i=0;i<m;i++){
      cout<<t.oc[i].size()<<nl;
      for(auto x:t.oc[i]) cout<<x-p[i].size()+1<<' ';
      cout<<nl;
   }
   ///print number of occurrences
   for(i=0;i<m;i++) cout<<t.cnt[t.ind[i]]<<' ';
   cout<<nl;
        return 0;
```

```
}
```

34. String Hashing

Notes

The probability that collision happens is $\approx \frac{1}{mod}$

If we compare a string with N different strings then the probability of collision is $\approx \frac{N}{mod}$

Hashing

```
ll qpow(ll n,ll k,ll mod) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1) ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}

const int MOD1=127657753,MOD2=987654319;
const int p1=137,p2=277;
int invp1,invp2;
pii pw[N],invpw[N];
void pre()
{
   pw[0]={1,1};
   for(int i=1;i<N;i++){
      pw[i].F=1LL*pw[i-1].F*p1%MOD1;
      pw[i].S=1LL*pw[i-1].S*p2%MOD2;
   }
   invp1=qpow(p1,MOD1-2,MOD1);
   invp2=qpow(p2,MOD2-2,MOD2);
   invpw[0]={1,1};
   for(int i=1;i<N;i++){
      invpw[i].F=1LL*invpw[i-1].F*invp1%MOD1;
      invpw[i].S=1LL*invpw[i-1].S*invp2%MOD2;
   }

}
///returns hash of string s
pii get_hash(string s)
{
   int n=s.size();
   pii ans={0,0};
   for(int i=0;i<n;i++){
      ans.F=(ans.F+1LL*pw[i].F*s[i]%MOD1)%MOD1;
      ans.S=(ans.S+1LL*pw[i].S*s[i]%MOD2)%MOD2;
   }
   return ans;
}
struct RollingHash
{
   int n;
   string s;///0-indexed
   vector<pii>hs;///1-indexed
   void init(string _s)
   {
      n=_s.size();
      s=_s;
      hs.eb(0,0);
      for(int i=0;i<n;i++){
         pii p;
         p.F=(hs[i].F+1LL*pw[i].F*s[i]%MOD1)%MOD1;
```

```
            p.S=(hs[i].S+1LL*pw[i].S*s[i]%MOD2)%MOD2;
            hs.pb(p);
        }
    }
    ///returns hash of substring [l....r],1-indexed
    pii get_hash(int l,int r)
    {
        pii ans;
        ans.F=(hs[r].F-hs[l-1].F+MOD1)%MOD1*1LL*invpw[l-
1].F%MOD1;
        ans.S=(hs[r].S-hs[l-1].S+MOD2)%MOD2*1LL*invpw[l-
1].S%MOD2;
        return ans;
    }
    ///returns hash of total string
    pii get()
    {
        return get_hash(1,n);
    }
};
RollingHash h;
int main()
{
    BeatMeScanf;
    int i,j,k,n,m;
    ///never forget to initialize pre()
    pre();
    string s;
    cin>>s;
    h.init(s);
    while(1){
        ll a,b,c,d;
        cin>>a>>b>>c>>d;
        cout<<(h.get_hash(a,b)==h.get_hash(c,d))<<nl;
    }
    return 0;
}
```

35. String Suffix Structures

Suffix Array O(n)

```
const int kinds = 128;///maximum ASCII value of any character of the
string
char str[N];
int K, buc[N], r[N], sa[N], X[N], Y[N], high[N];
bool cmp(int *r, int a, int b, int x)
{
    return (r[a] == r[b] && r[a+x] == r[b+x]);
}

void suffix_array_DA(int n, int m)
{
    int *x = X, *y = Y, i, j, k = 0, l;
    memset(buc, 0, sizeof(buc));
    for(i = 0; i < n; i++) buc[ x[i]=str[i] ]++;
    for(i = 1; i < m; i++) buc[i] += buc[i-1];
    for(i = n-1; i >= 0; i--) sa[--buc[x[i]]] = i;
    for(l = 1, j = 1; j < n; m = j, l <<= 1)
```

```
    {
        j = 0;
        for(i = n-l; i < n; i++) y[j++] = i;
        for(i = 0; i < n; i++) if(sa[i] >= l) y[j++] = sa[i]-l;
        for(i = 0; i < m; i++) buc[i] = 0;
        for(i = 0; i < n; i++) buc[ x[y[i]] ]++;
        for(i = 1; i < m; i++) buc[i] += buc[i-1];
        for(i = n-1; i >= 0; i--) sa[ --buc[ x[y[i]] ]] = y[i];
        for(swap(x, y), x[sa[0]] = 0, i = 1, j = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i-1], sa[i], l) ? j-1 : j++;
    }
    for(i = 1; i < n; i++) r[sa[i]] = i;
    for(i = 0; i < n-1; high[r[i++]] = k)
        for(k ? k--: 0, j = sa[r[i]-1]; str[i+k] == str[j+k]; k++);
}
vector<int> suffix_array_construction(string s)
{
    int n=s.size();
    for(int i=0;i<n;i++) str[i]=s[i];
    str[n]='\0';
    suffix_array_DA(n+1,kinds);
    vector<int>saa;
    for(int i=1;i<=n;i++) saa.eb(sa[i]);
    return saa;
}
vector<int> lcp_construction(string const& s, vector<int> const& p)
{
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);

    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])  k++;
        lcp[rank[i]] = k;
        if (k)  k--;
    }
    return lcp;
}
const int MX = 18;
int st[N][MX];
int lg[N];

void pre()
{
    lg[1] = 0;
    for (int i=2; i<N; i++)
        lg[i] = lg[i/2]+1;
}
```

```
void build(vector<int> &lcp)
{
   int n = lcp.size();
   for (int i=0; i<n; i++)
      st[i][0] = lcp[i];

   for (int k=1; k<MX; k++)
      for (int i=0; i<n; i++)
      {
         st[i][k] = st[i][k-1];
         int nxt = i + (1<<(k-1));
         if (nxt >= n)    continue;
         st[i][k] = min(st[i][k], st[nxt][k-1]);
      }
}
///minimum of lcp[l.....r]
int get(int l, int r)
{
   int k = lg[r-l+1];
   return min(st[l][k], st[r-(1<<k)+1][k]);
}
int ra[N],sz;
///lcp of suffix starting from i and j
int lcp_(int i,int j)
{
   if(i==j) return sz-i;
   int l=ra[i];
   int r=ra[j];
   if(l>r) swap(l,r);
   return get(l,r-1);
}
string ss;
///lower bound of string t
int lb(string &t,vi &sa){
   int l=0,r=sz-1;
   int k=t.size();
   int ans=sz;
   while(l<=r){
      int mid = (l+r)/2;
      if(ss.substr(sa[mid],min(sz-sa[mid],k)) >= t) ans=mid,r=mid-1;
      else l = mid+1;
   }
   return ans;
}
///upper bound of string t
int ub(string &t,vi &sa){
   int l=0,r=sz-1;
   int k=t.size();
   int ans=sz;
   while(l<=r){
      int mid = (l+r)/2;
      if(ss.substr(sa[mid],min(sz-sa[mid],k)) > t) ans=mid,r=mid-1;
      else l = mid+1;
   }
   return ans;
}
int main()
{
```

```
fast;
int i,j,k,n,m,q;
string s;
cin>>s;
n = s.size();
vector<int> sa = suffix_array_construction(s);
vector<int> lcp = lcp_construction(s, sa);
sz=n;
ss=s;
for(i=0;i<n;i++) ra[sa[i]]=i;
pre();
build(lcp);
for(i=0;i<n;i++) cout<<sa[i]<<' ';
cout<<nl;
for(i=0;i<n-1;i++) cout<<lcp[i]<<' ';
cout<<nl;
cin>>q;
///lcp of suffixes
while(q--){
   cin>>i>>j;
   cout<<lcp_(i,j)<<nl;
}
cin>>q;
///number of occurrences of a pattern, not in sorted order
while(q--){
   string t;
   cin>>t;
   int l=lb(t,sa);
   int r=ub(t,sa);
      debug(l,r);
      for(i=l;i<r;i++) cout<<sa[i]<<' ';
      cout<<nl;
   }
   return 0;
}
```

Suffix Array O(nlogn)

```
/// Equivalence Class of every k-th step
vector<vector<int> >c;
vector<int>sort_cyclic_shifts(string const& s)
{
   int n = s.size();
   const int alphabet = 256;
   vector<int> p(n), cnt(alphabet, 0);

   c.clear();
   c.emplace_back();
   c[0].resize(n);

   for (int i = 0; i < n; i++)        cnt[s[i]]++;
   for (int i = 1; i < alphabet; i++)  cnt[i] += cnt[i-1];
   for (int i = 0; i < n; i++)        p[--cnt[s[i]]] = i;

   c[0][p[0]] = 0;
   int classes = 1;

   for (int i = 1; i < n; i++) {
      if (s[p[i]] != s[p[i-1]])   classes++;
```

```
    c[0][p[i]] = classes - 1;
}

vector<int> pn(n), cn(n);
cnt.resize(n);

for (int h = 0; (1<<h) < n; h++) {
    for (int i = 0; i < n; i++) {
        pn[i] = p[i] - (1<<h);
        if (pn[i] < 0)  pn[i] += n;
    }
    fill(cnt.begin(), cnt.end(), 0);

    /// radix sort
    for (int i = 0; i < n; i++)         cnt[c[h][pn[i]]]++;
    for (int i = 1; i < classes; i++)   cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; i--)      p[--cnt[c[h][pn[i]]]] = pn[i];

    cn[p[0]] = 0;
    classes = 1;
    for (int i = 1; i < n; i++) {
        pii cur = {c[h][p[i]]   , c[h][(p[i] + (1<<h))%n]};
        pii prev = {c[h][p[i-1]], c[h][(p[i-1] + (1<<h))%n]};
        if (cur != prev)    ++classes;
        cn[p[i]] = classes - 1;
    }
    c.push_back(cn);
}
return p;
}

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

/// compare two suffixes starting at i and j with length 2^k
int compare(int i, int j, int n, int k)
{
    pii a = {c[k][i], c[k][(i+1-(1<<k))%n]};
    pii b = {c[k][j], c[k][(j+1-(1<<k))%n]};
    return a == b ? 0 : a < b ? -1 : 1;
}

int lcp_(int i, int j)
{
    int log_n = c.size()-1;
    int ans = 0;

    for (int k = log_n; k >= 0; k--) {
        if (c[k][i] == c[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
```

```
    }
    return ans;
}

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

    for (int i = 0; i < n; i++) rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);

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

const int K = 18;
int st[N][K];
int lg[N];

void pre()
{
    lg[1] = 0;
    for (int i=2; i<N; i++)
        lg[i] = lg[i/2]+1;
}

void build(vector<int> &lcp)
{
    int n = lcp.size();
    for (int i=0; i<n; i++)
        st[i][0] = lcp[i];

    for (int k=1; k<K; k++)
        for (int i=0; i<n; i++)
        {
            st[i][k] = st[i][k-1];
            int nxt = i + (1<<(k-1));
            if (nxt >= n)    continue;
            st[i][k] = min(st[i][k], st[nxt][k-1]);
        }
}
///minimum of lcp[l.....r]
int get(int l, int r)
{
    int k = lg[r-l+1];
    return min(st[l][k], st[r-(1<<k)+1][k]);
```

```
}

int main()
{
   int i,j,k,n,m;
   pre();
   string s;
   cin>>s;
   n = s.size();

   vector<int> sa = suffix_array_construction(s);
   vector<int> lcp = lcp_construction(s, sa);
   build(lcp);
   for(i=0;i<n;i++) cout<<sa[i]<<nl;
   return 0;
}
```

Suffix Automaton

```
///number of states or nodes  in a suffix automaton is equal to the
/// number of equivalence classes i.e. endpos-equivalent classes among all substrings
struct node
{
   int len;          ///largest string length of the corresponding endpos-equivalent class
   int link;         ///leads to the state that corresponds to the longest suffix of w
                     ///that is another endpos-equivalent class.
   int firstpos;     ///1-indexed end position of the first occurrence of the largest string length of the
                     ///corresponding endpos-equivalent class
   map<char,int>nxt;
};
///all suffix links of the last node are terminal nodes including the last node
const int MX=mxn*2;
node t[MX];
int sz,last;
void init()
{
   sz=last=0;
   t[0].len=0;
   t[0].firstpos=0;
   t[0].link=-1;
   sz++;
}
ll cnt[MX];///number of times i-th node occurs in the string
vpii v;
set<pii>nodes;
void add_letter(char ch)
{
   int cur=sz++;
   t[cur].len=t[last].len+1;
   t[cur].firstpos=t[cur].len;
   cnt[cur]=1;
   nodes.insert({t[cur].len,cur});
   int p;
```

```
    for(p=last;p!=-1&&!t[p].nxt.count(ch);p=t[p].link) t[p].nxt[ch]=cur;
    if(p==-1) t[cur].link=0;
    else{
        int q=t[p].nxt[ch];
        if(t[p].len+1==t[q].len) t[cur].link=q;
        else{
            int clone=sz++;
            t[clone].len=t[p].len+1;
            t[clone].nxt=t[q].nxt;
            t[clone].link=t[q].link;
            t[clone].firstpos=t[q].firstpos;
            cnt[clone]=0;
            nodes.insert({t[clone].len,clone});
            for(;p!=-1&&t[p].nxt[ch]==q;p=t[p].link) t[p].nxt[ch]=clone;
            t[q].link=t[cur].link=clone;
        }
    }
    last=cur;
}
ll dcnt[MX];            ///number of distinct substrings in the subtree of
node i
ll dist_sub(int u)      ///number of distinct substrings of the string
{
    ll ans=1;
    if(dcnt[u]) return dcnt[u];
    for(auto x:t[u].nxt){
        char ch=x.F;
        ans+=1LL*dist_sub(t[u].nxt[ch]);
    }
    return dcnt[u]=ans;
}
///returns the lexicographically k-th substring pos in 1-indexed
///O(n)
pii kth_Path(int k)
{
    int len = 0;
    int cur = 0;
    int pos = -1;
    for(; k; --k){
        int s = 0, p = cur;
        for(auto it:t[cur].nxt){
            if(dcnt[it.S] + s < k) s += dcnt[it.S];
            else{
                len ++, cur = it.second, pos = t[it.second].firstpos;
                break;
            }
        }
        if(cur == p) break;
        k -= s;
    }
    if(k == 0) return MP(pos - len + 1,pos);
    else return MP(-1, -1);
}
int lcs(string s)
{
    int cur=0,ans=0,len=0,pos=0;
    for(int i=0;i<s.size();i++){
        while(cur&&!t[cur].nxt.count(s[i])){
```

```
            cur=t[cur].link;
            len=t[cur].len;
        }
        if(t[cur].nxt.count(s[i])){
            cur=t[cur].nxt[s[i]];
            len++;
        }
        if(len>ans) ans=len,pos=i;
    }
    string sub=s.substr(pos-ans+1,ans);
    return ans;
}
int main()
{
    fast;
    int i,j,n,m,k,q;
    string s;
    cin>>s;
    n=s.size();
    init();
    for(i=0;i<n;i++) add_letter(s[i]);
    for(auto it=nodes.rbegin();it!=nodes.rend();++it)
cnt[t[(*it).S].link]+=cnt[(*it).S];
    dist_sub(0);
    cout<<dcnt[0]-1<<nl;
    k=2;
    pii pos=kth_Path(k);
    if(pos.F==-1) cout<<"no such string\n";
    else cout<<s.substr(pos.F-1,pos.S-pos.F+1)<<nl;
    cout<<lcs("abc")<<nl;
    for(i=1;i<sz;i++) cout<<t[i].firstpos<<' '<<t[i].len<<' '<<cnt[i]<<nl;
    ///longest repeated substring
    ///for bababa ans is 4(baba)
    int ans=0;
    for(i=1;i<sz;i++) if(cnt[i]>1) ans=max(ans,t[i].len);
    cout<<ans<<nl;
    return 0;
}
```

Suffix Tree

```
string s;
int n;

struct node {
        int l, r, par, link;
        map<char,int> next;

        node (int l=0, int r=0, int par=-1)
                : l(l), r(r), par(par), link(-1) {}
        int len()  {  return r - l;  }
        int &get (char c) {
                if (!next.count(c))  next[c] = -1;
                return next[c];
        }
};
node t[N];
int sz;
```

```
struct state {
	int v, pos;
	state (int v, int pos) : v(v), pos(pos)  {}
};
state ptr (0, 0);

state go (state st, int l, int r) {
	while (l < r)
		if (st.pos == t[st.v].len()) {
			st = state (t[st.v].get( s[l] ), 0);
			if (st.v == -1)  return st;
		}
		else {
			if (s[ t[st.v].l + st.pos ] != s[l])
				return state (-1, -1);
			if (r-l < t[st.v].len() - st.pos)
				return state (st.v, st.pos + r-l);
			l += t[st.v].len() - st.pos;
			st.pos = t[st.v].len();
		}
	return st;
}

int split (state st) {
	if (st.pos == t[st.v].len())
		return st.v;
	if (st.pos == 0)
		return t[st.v].par;
	node v = t[st.v];
	int id = sz++;
	t[id] = node (v.l, v.l+st.pos, v.par);
	t[v.par].get( s[v.l] ) = id;
	t[id].get( s[v.l+st.pos] ) = st.v;
	t[st.v].par = id;
	t[st.v].l += st.pos;
	return id;
}

int get_link (int v) {
	if (t[v].link != -1)  return t[v].link;
	if (t[v].par == -1)  return 0;
	int to = get_link (t[v].par);
	return t[v].link = split (go (state(to,t[to].len()), t[v].l +
(t[v].par==0), t[v].r));
}

void tree_extend (int pos) {
	for(;;) {
		state nptr = go (ptr, pos, pos+1);
		if (nptr.v != -1) {
			ptr = nptr;
			return;
		}

		int mid = split (ptr);
		int leaf = sz++;
		t[leaf] = node (pos, n, mid);
		t[mid].get( s[pos] ) = leaf;
```

```
                ptr.v = get_link (mid);
                ptr.pos = t[ptr.v].len();
                if (!mid)  break;
        }
}

void build_tree() {
        sz = 1;
        for (int i=0; i<n; ++i)
                tree_extend (i);
}
void dfs(int i)
{
   cout<<i<<' '<<t[i].l<<' '<<t[i].r<<nl;
   for(auto x:t[i].next) dfs(x.S);
}
int main()
{
   BeatMeScanf;
   int i,j,k,m;
   s="banana";
   s+="$";
   n=s.size();
   build_tree();
   dfs(0);
   return 0;
}
```

36. Palindromes

Palindromic Tree

```
///There can be at most n unique palindromes for a string of size n
struct node          ///a node is a palindromic substring of the string
{
   int nxt[26];      ///link to the palindrome which is formed by adding
                     ///next[i] in both side of this palindrome
   int len;          ///length of the palindrome
   int st,en;        ///starting and ending index of the node
   int suflink;      ///link to the maximum proper suffix palindrome of
the node
   int cnt;          ///stores the length of the suffix link chain from it
(including this node)
                     ///i.e. the number of palindromic suffix of this node
   int oc;           ///stores the number of occurrence of the node
};
string s;
node t[N];
int n;               ///size of string
int sz;              ///indicates size of the tree
int suf;             ///index of maximum suffix palindrome
void init()
{
   sz=2,suf=2;
   t[1].len=-1,t[1].suflink=1;  ///node 1- root with length -1
   t[2].len=0,t[2].suflink=1;   ///node 2- root with length 0 i.e null
palindrome
}
```

```
/// return if creates a new palindrome
int add_letter(int pos)
{
   ///find the maximum suffix of the prefix+s[pos]
   int cur=suf,curlen=0;
   int ch=s[pos]-'a';
   while(1){
      curlen=t[cur].len;
      if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]) break;
      cur=t[cur].suflink;
   }
   ///if the node is not created yet then create the new node
   if(t[cur].nxt[ch]){
      suf=t[cur].nxt[ch];
      t[suf].oc++;
      return 0;
   }
   sz++;
   suf=sz;
   t[sz].oc=1;
   t[sz].len=t[cur].len+2;
   t[cur].nxt[ch]=sz;
   t[sz].en=pos;
   t[sz].st=pos-t[sz].len+1;
   if(t[sz].len==1){
      t[sz].suflink=2;
      t[sz].cnt=1;
      return 1;
   }
   while(1){
      cur=t[cur].suflink;
      curlen=t[cur].len;
      if(pos-1-curlen>=0&&s[pos-1-curlen]==s[pos]){
            t[sz].suflink=t[cur].nxt[ch];
            break;
      }
   }
   t[sz].cnt=1+t[t[sz].suflink].cnt;
   return 1;
}
int main()
{
   fast;
   int i,j,k,n,m;
   ll ans=0;///number of palindromic substrings of a string (not unique
palindrome)
   cin>>s;
   n=s.size();
   init();
   for(i=0;i<n;i++){
      add_letter(i);
      ans+=1LL*t[suf].cnt;
   }
   cout<<ans<<nl;
   for(i=sz;i>=3;i--) t[t[i].suflink].oc+=t[i].oc;
   for(i=3;i<=sz;i++) cout<<t[i].st<<' '<<t[i].en<<' '<<t[i].oc<<nl;
///for multiple input clear all the (0..sz) nodes nxt array
   return 0;
```

```
}
```

Manacher's Algorithm

```
int main()
{
   BeatMeScanf;
   int i,j,k,n,m;
   string s;
   cin>>s;
   n=s.size();
   vector<int> d1(n);  ///maximum odd length palindrome centered at i
                  ///here d1[i]=the palindrome has d1[i]-1 right characters from i
                  ///e.g. for aba, d1[1]=2;
   for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
      while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
         k++;
      }
      d1[i] = k--;
      if (i + k > r) {
         l = i - k;
         r = i + k;
      }
   }
   vector<int> d2(n);  ///maximum even length palindrome centered at i
                  ///here d2[i]=the palindrome has d2[i]-1 right characters from i
                  ///e.g. for abba, d2[2]=2;
   for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
      while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
         k++;
      }
      d2[i] = k--;
      if (i + k > r) {
         l = i - k - 1;
         r = i + k ;
      }
   }
   for(i=0;i<n;i++) cout<<d1[i]<<' ';
   cout<<nl;
   for(i=0;i<n;i++) cout<<d2[i]<<' ';
   cout<<nl;
   ///number of palindromes
   ll ans=0;
   for(i=0;i<n;i++){
      ans+=1LL*d1[i];
      ans+=1LL*d2[i];
   }
   cout<<ans<<nl;
   ///
   return 0;
}
```

37. Lyndon Factorization

///Complexity: $O(n)$
///A string is called simple (or a Lyndon word), if it is strictly smaller than
///any of its own nontrivial suffixes. Examples of simple strings are: a, b, ab, aab, abb, ababb,
///abcd. It can be shown that a string is simple, if and only if it is strictly smaller
///than all its nontrivial cyclic shifts.
///Next,let there be a given string s. The Lyndon factorization of the
///string s is a factorization s=w1w2…wk, where all strings wi are simple,
///and they are in non-increasing order w1>=w2>=....>=wk.
///It can be shown, that for any string such a factorization exists and that it is unique.
///Here we use Duval algorithm for finding lyndon factorization

```
vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}
string min_cyclic_string(string s) {
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
            i += j - k;
    }
    return s.substr(ans, n / 2);
}
int main()
```

```
{
   BeatMeScanf;
   ll i,j,k,n,m;
   string s;
   cin>>s;
   cout<<min_cyclic_string(s)<<nl;
   return 0;
}
```

38. Expression Parsing

```
///returns precedence of operators
int precedence(char symbol)
{

   switch(symbol)
   {
   case '+':
   case '-':
      return 2;
      break;
   case '*':
   case '/':
      return 3;
      break;
   case '^':
      return 4;
      break;
   case '(':
   case ')':
   case '#':
      return 1;
      break;
   }
}

///check whether the symbol is operator?
int isOperator(char symbol)
{

   switch(symbol)
   {
   case '+':
   case '-':
   case '*':
   case '/':
   case '^':
   case '(':
   case ')':
      return 1;
      break;
   default:
      return 0;
   }
}

///converts infix expression to postfix
string convert(string infix)
{
```

```
stack<char>st;
string postfix="";
st.push('#');
for(auto ch:infix){
   if(isOperator(ch) == 0) postfix+=ch;
   else
   {
      if(ch == '(') st.push(ch);
      else
      {
         if(ch == ')')
         {
            while(st.top() != '(')
            {
               postfix+= st.top();
               st.pop();
            }
            st.pop();
         }
         else
         {
            if(precedence(ch)>precedence(st.top())) st.push(ch);
            else
            {
               while(precedence(ch)<=precedence(st.top()))
               {
                  postfix+= st.top();
                  st.pop();
               }
                  st.push(ch);
               }
            }
         }
      }
   }
   while(st.top() != '#')
   {
      postfix+=st.top();
      st.pop();
   }
   return postfix;
}

///evaluates postfix expression
int evaluate(string postfix)
{
   int op1,op2;
   stack<int>st;
   for(auto ch:postfix){
      if(isdigit(ch)) st.push(ch-'0');
      else
      {
         ///Operator,pop two  operands
         op2 = st.top();
         st.pop();
         op1 = st.top();
         st.pop();
         switch(ch)
```

```
        {
        case '+':
           st.push(op1+op2);
           break;
        case '-':
           st.push(op1-op2);
           break;
        case '*':
           st.push(op1*op2);
           break;
        case '/':
           st.push(op1/op2);
           break;
        default:
           st.push((int)(pow(1.0*op1,1.0*op2)+eps));
        }
      }
   }
   return st.top();
}
int main()
{
   BeatMeScanf;
   int i,j,k,n,m;
   string infix= "(9/3)*1*(2+3)+7-9^2";
   string postfix=convert(infix);
   cout<<infix<<nl<<postfix<<nl<<evaluate(postfix)<<nl;
   return 0;
}
```

GRAPH THEORY

39. Strongly Connected Components

Condensation Graph

////// A condensation graph is a graph containing every strongly connected component as one vertex

///this code also describes- given undirected graph return minimum number of edges to be added so the whole graph become a SCC

```
bool vis[N];
vi g[N],r[N],cn[N],vec;
void dfs1(int u)
{
   vis[u]=1;
   for(auto v:g[u]) if(!vis[v]) dfs1(v);
   vec.eb(u);
}
vi comp;
void dfs2(int u)
{
   comp.eb(u);
   vis[u]=1;
   for(auto v:r[u]) if(!vis[v]) dfs2(v);
}
int idx[N],in[N],out[N];
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,u,v;
```

```
cin>>n>>m;
for(i=1;i<=m;i++){
   cin>>u>>v;
   g[u].eb(v);
   r[v].eb(u);
}
for(i=1;i<=n;i++) if(!vis[i]) dfs1(i);
rev(vec);
mem(vis,0);
int scc=0;///number of SCC
for(auto u:vec){
   if(!vis[u]){
      comp.clear();
      dfs2(u);
      ///here we have all the nodes in this component
      scc++;
      for(auto x:comp) idx[x]=scc;
   }
}
for(u=1;u<=n;u++){
   for(auto v:g[u]){
      if(idx[u]!=idx[v]){
         in[idx[v]]++,out[idx[u]]++;
         cn[idx[u]].eb(idx[v]);
      }
   }
}
int needed_in=0,needed_out=0;
for(i=1;i<=scc;i++){
   if(!in[i]) needed_in++;
   if(!out[i]) needed_out++;
}
int ans=max(needed_in,needed_out);
if(scc==1) ans=0;///corner case
///answer for the corresponding problem;
cout<<ans<<nl;
///output the condensation graph
for(u=1;u<=scc;u++){
   cout<<u<<": ";
   for(auto v:cn[u]) cout<<v<<' ';
   cout<<nl;
}
return 0;
}
```

40. Articulation Points

```
bool vis[N];
int art[N];
vi g[N];
int dis[N],low[N],T;
void dfs(int u,bool isroot)
{
   dis[u]=low[u]=++T;
   vis[u]=1;
   int child=0;
   for(auto v:g[u]){
      if(!vis[v]){
         dfs(v,0);
```

```
            if(dis[u]<=low[v]&&!isroot) art[u]=1;
            low[u]=min(low[u],low[v]);
            child++;
        }
        else low[u]=min(low[u],dis[v]);
    }
    if(isroot&&child>1) art[u]=1;
}
int main()
{
    BeatMeScanf;
    int i,j,k,n,m,u,v;
    cin>>n>>m;
    for(i=1;i<=m;i++){
        cin>>u>>v;
        g[u].eb(v);
        g[v].eb(u);
    }
    for(i=1;i<=n;i++) if(!vis[i]) dfs(i,1);
    int ans=0;
    for(i=1;i<=n;i++) if(art[i]) ans++;
    cout<<ans<<nl;
    return 0;
}
```

41. Articulation Bridges

Articulation Bridges Standard

```
bool vis[N];
vi g[N];
int dis[N],low[N],T;
set<pii>bridge;
void dfs(int u,int pre)
{
    low[u]=dis[u]=++T;
    vis[u]=1;
    for(auto v:g[u]){
        if(!vis[v]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dis[u]) bridge.insert({min(u,v),max(u,v)});
        }
        else{
            if(v!=pre) low[u]=min(low[u],dis[v]);
        }
    }
}
int main()
{
    BeatMeScanf;
    int i,j,k,n,m,u,v;
    cin>>n>>m;
    for(i=1;i<=m;i++){
        cin>>u>>v;
        g[u].eb(v);
        g[v].eb(u);
    }
    dfs(1,0);
```

```
    int ans=bridge.size();
    cout<<ans<<nl;
    return 0;
}
```

Articulation Bridges Online

```
///Given number of nodes n and q queries
///add edge (u,v)
///output the bridges in current graph
int n, bridges, par[N], bl[N], comp[N], sz[N];


void init() {
        for (int i=0; i<n; ++i) {
                bl[i] = comp[i] = i;
                sz[i] = 1;
                par[i] = -1;
        }
        bridges = 0;
}


int get (int v) {
        if (v==-1)  return -1;
        return bl[v]==v ? v : bl[v]=get(bl[v]);
}

int get_comp (int v) {
        v = get(v);
        return comp[v]==v ? v : comp[v]=get_comp(comp[v]);
}

void make_root (int v) {
        v = get(v);
        int root = v,
                child = -1;
        while (v != -1) {
                int p = get(par[v]);
                par[v] = child;
                comp[v] = root;
                child=v;  v=p;
        }
        sz[root] = sz[child];
}


int cu, u[N];

void merge_path (int a, int b) {
        ++cu;

        vector<int> va, vb;
        int lca = -1;
        for(;;) {
                if (a != -1) {
                        a = get(a);
                        va.pb (a);
```

```
                if (u[a] == cu) {
                        lca = a;
                        break;
                }
                u[a] = cu;

                a = par[a];
        }

        if (b != -1) {
                b = get(b);
                vb.pb (b);

                if (u[b] == cu) {
                        lca = b;
                        break;
                }
                u[b] = cu;

                b = par[b];
        }
}

for (int i=0; i<va.size(); ++i) {
        bl[va[i]] = lca;
        if (va[i] == lca)  break;
        --bridges;
}
for (int i=0; i<vb.size(); ++i) {
                bl[vb[i]] = lca;
                if (vb[i] == lca)  break;
                --bridges;
        }
}


void add_edge (int a, int b) {
        a = get(a);   b = get(b);
        if (a == b)  return;

        int ca = get_comp(a), cb = get_comp(b);
        if (ca != cb) {
                ++bridges;
                if (sz[ca] > sz[cb]) {
                        swap (a, b);
                        swap (ca, cb);
                }
                make_root (a);
                par[a] = comp[a] = b;
                sz[cb] += sz[a];
        }
        else merge_path (a, b);
}
///1-indexed
int main()
{
   BeatMeScanf;
   int i,j,k,m,q,u,v;
```

```
    cin>>n>>q;
    init();
    while(q--){
        cin>>u>>v;
        add_edge(u,v);
        cout<<bridges<<nl;
    }
    return 0;
}
```

Articulation Bridge Tree

```
///diameter of the Articulation Bridge Tree
vi g[N],gr[N];
bool vis[N];
int T,low[N],dis[N],d[N],par[N];
set<pii>bridge;
void dfs(int u,int pre)
{
    low[u]=dis[u]=++T;
    vis[u]=1;
    for(auto v:g[u]){
        if(!vis[v]){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dis[u]) bridge.insert({min(u,v),max(u,v)});
        }
        else{
            if(v!=pre) low[u]=min(low[u],dis[v]);
        }
    }
}
int find_(int x)
{
    if(par[x]==x) return x;
    return par[x]=find_(par[x]);
}
void merge_(int x,int y)
{
    int u=find_(x);
    int v=find_(y);
    if(u!=v){
        if(rand()%2) par[u]=v;
        else par[v]=u;
    }
}
int bfs(int s)
{
    mem(d,-1);
    queue<int>q;
    q.push(s);
    d[s]=0;
    int u=s;
    while(!q.empty()){
        u=q.front();
        q.pop();
        for(auto v:gr[u]){
            if(d[v]==-1){
                q.push(v);
```

```
            d[v]=d[u]+1;
          }
        }
    }
    return u;
}
int main()
{
    fast;
    int i,j,k,n,m,u,v,ans=0;
    cin>>n>>m;
    for(i=1;i<=m;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
    dfs(1,0);
    for(i=1;i<=n;i++) par[i]=i;
    for(u=1;u<=n;u++){
        for(auto v:g[u]){
            if(bridge.find({min(u,v),max(u,v)})==bridge.end())
merge_(u,v);
        }
    }
    for(auto p:bridge){
        u=p.F;
        v=p.S;
        int x=find_(u);
        int y=find_(v);
        gr[x].eb(y);
        gr[y].eb(x);
    }
    u=bfs(find_(1));
    v=bfs(find_(u));
    cout<<d[v]<<nl;
    return 0;
}
```

42. Biconnected Components

```
/// biconnected component of a given graph is the maximal
connected subgraph
///which does not contain any articulation vertices.

///1 Based,no problem in multiple edge and self loop
int dis[N],low[N];
int T,n;
vector<int> g[N]; ///only g should be cleared
stack<pii >st;
vector<set<pii>>e;///biconnected components
void calc_bcc(int u, int v)
{
    int i, j, uu, vv, cur;
    pii now;
    set<pii>se;
    while(!st.empty())
    {
        now = st.top();
        st.pop();
        uu = now.first, vv = now.second;
        se.insert({uu, vv});
        if(u==uu && v==vv)
            break;
```

```
        if(u==vv && v==uu)
            break;
    }
    e.eb(se);
    return;
}
void bcc(int u,int pre)    /// pre=-1 dhore call dite hobe(root ar parent
nai)
{
    dis[u] = low[u] = ++T;
    for(int i = 0 ; i<g[u].size() ; i++)
    {
        int v = g[u][i];
        if(v==pre) continue;
        if(dis[v]==0)
        {
            st.push(make_pair(u, v));
            bcc(v,u);
            low[u] = min(low[u],low[v]);
            if(low[v]>=dis[u])
            {
                calc_bcc(u, v);
            }
        }
        else if(dis[v] < dis[u])
        {
            low[u] = min(low[u],dis[v]);
            st.push(make_pair(u, v));
        }
    }
    return;
}
int main()
{
    BeatMeScanf;
    int i,j,k,m,u,v;
    cin>>n>>m;
    while(m--)
    {
        cin>>u>>v;
        g[u].eb(v);
        g[v].eb(u);
    }
    T=0;
    memset(dis,0,sizeof dis);
    for(i = 1; i <= n; i++)if(!dis[i]) bcc(i,-1);
    for(auto se:e){
        for(auto edge:se) cout<<edge.F<<' '<<edge.S<<' ';
        cout<<nl;
    }
    ///if two nodes u and v have at least two vertex disjoint path i.e. if
two paths have
    ///no common vertices except u and v
    ///check if they are in same biconnected components
}
```

43. Block Cut Tree

```
///Any connected graph decomposes into a tree of biconnected
///components called the block-cut tree of the graph
///1 Based,no problem in multiple edge and self loop
int dis[N],low[N];
int T,n;
vector<int> g[N]; ///only g should be cleared
stack<pii >st;
vector<set<pii>>e;///biconnected components
void calc_bcc(int u, int v)
{
   int i, j, uu, vv, cur;
   pii now;
   set<pii>se;
   while(!st.empty())
   {
      now = st.top();
      st.pop();
      uu = now.first, vv = now.second;
      se.insert({uu, vv});
      if(u==uu && v==vv)
         break;
      if(u==vv && v==uu)
         break;
   }
   //if(vec.size()<=1) return;
   e.eb(se);
   return;
}
int art[N];
void bcc(int u,int pre)    /// pre=-1 dhore call dite hobe(root ar parent nai)
{
   dis[u] = low[u] = ++T;
   int child=0;
   for(int i = 0 ; i<g[u].size() ; i++)
   {
      int v = g[u][i];
      if(v==pre) continue;
      if(dis[v]==0)
      {
         st.push(make_pair(u, v));
         bcc(v,u);
         low[u] = min(low[u],low[v]);
         if(low[v]>=dis[u])
         {
            if(pre!=-1) art[u]=1;
            calc_bcc(u, v);
         }
         child++;
      }
      else if(dis[v] < dis[u])
      {
         low[u] = min(low[u],dis[v]);
         st.push(make_pair(u, v));
      }
```

```
    }
    if(pre==-1&&child>1) art[u]=1;
    return;
}
set<int> bt[N];///block cut tree
int id[N];
int main()
{
    BeatMeScanf;
    int i,j,k,m,u,v;
    cin>>n>>m;
    while(m--)
    {
        cin>>u>>v;
        g[u].eb(v);
        g[v].eb(u);
    }
    for(i = 1; i <= n; i++)if(!dis[i]) bcc(i,-1);
    int sz=0;
    for(i=1;i<=n;i++){
        if(art[i]){
            id[i]=++sz;
            cout<<i<<' ';
        }
    }
    cout<<nl;
    for(auto se:e){
        bool nonart=0;
        for(auto edge:se){
            u=edge.F,v=edge.S;
            if(!art[u]||!art[v]){
                nonart=1;
                break;
            }
        }
        if(nonart) ++sz;
        for(auto edge:se){
            u=edge.F,v=edge.S;
            if(art[u]&&art[v]){
                bt[id[u]].insert(id[v]);
                bt[id[v]].insert(id[u]);
                continue;
            }
            int dummy;
            if(!art[u]) id[u]=sz;
            else bt[id[u]].insert(sz),bt[sz].insert(id[u]);
            int dumm;
            if(!art[v]) id[v]=sz;
            else bt[id[v]].insert(sz),bt[sz].insert(id[v]);
        }
    }
    for(i=1;i<=n;i++) cout<<id[i]<<' ';
    cout<<nl;
    for(i=1;i<=sz;i++){
        for(auto x:bt[i]) cout<<x<<' ';
        cout<<nl;
    }
}
```

44. Spanning Tree

Prim's Minimum Spanning Tree

```
int g[N][N];
struct edge {
   int w = 1e9, to = -1;
};
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,u,v,w;
   cin>>n>>m;
   for(i=1;i<=n;i++) for(j=1;j<=n;j++) g[i][j]=1e9;
   for(i=1;i<=m;i++){
      cin>>u>>v>>w;
      g[u][v]=w;
      g[v][u]=w;
   }
   int ans = 0;
   vector<bool> selected(n+1);
   vector<edge> e(n+1);
   e[1].w = 0;
   vector<pii>edges;
   for (i=1;i<=n;++i) {
      u=-1;
      for (j=1;j<=n;++j) {
         if (!selected[j] && (u == -1 || e[j].w < e[u].w))
            u = j;
      }
      if (e[u].w == 1e9) {
         cout<<"No MST!"<< endl;
         exit(0);
      }
      selected[u] = true;
      ans += e[u].w;
      if (e[u].to != -1) edges.eb(u,e[u].to);

      for (int to = 1; to <= n; ++to) {
         if (g[u][to] < e[to].w) e[to] = {g[u][to], u};
      }
   }
   cout<<ans<<endl;
   for(auto x:edges) cout<<x.F<<' '<<x.S<<nl;
   return 0;
}
```

Directed Minimum Spanning Tree

///Complexity:$O(nlogn)$

///Directed MST is finding a spanning arborescence of minimum weight

///An arborescence is a directed graph in which,a vertex u

///called the root and for any other vertex v, there is exactly one

///directed path from u to v. An arborescence is thus

///the directed-graph form of a rooted tree,

```
#define INF 100100000
struct edge
{
```

```
    int u,v,w,idx;
    edge(int u=0,int v=0,int w=0)
    {
        this->u = u;
        this->v = v;
        this->w = w;
    }
    bool operator < (const edge &b)     const
    {
        return w<b.w;
    }
};
///here value of N should be maximum number of vertices
int n,m; ///n = number of vertex,m=number of edges
vector<edge> e[N]; ///edge u->v inserted into list of v.
vector<edge> edges; ///all edges ,needed if used edges required.
vector<int>g[N]; /// to check the graph connectivity.
int par[N],color[N];
int weight[N],touse[N];
bool used[N+100];
vector<int>choosed;
int directed_mst(int root)
{
    int i,j,t,u,v;
    e[root].clear();
    for(i=0; i<n; i++)
    {
        par[i] = i;
        sort(e[i].begin(),e[i].end());
    }
    bool cycle_found = true;
    while(cycle_found)
    {
        cycle_found = false;
        mem(color,0);
        color[root] = -1;
        for(i=0,t=1; i<n; i++,t++)
        {
            u = par[i];
            if(color[u]) continue;
            for(v=u; !color[v]; v=par[e[v][0].u])
            {
                color[v] = t;
                choosed.push_back(e[v][0].idx);
            }
            if(color[v] != t) continue;
            cycle_found = true;
            int sum = 0, super = v;
            for( ; color[v]==t; v=par[e[v][0].u])
            {
                color[v]++;
                sum+= e[v][0].w;
            }
            for(j=0; j<n; j++) weight[j] = INF;
            for(; color[v]==t+1; v=par[e[v][0].u])
            {
                color[v]--;
                for(j = 1; j<e[v].size(); j++)
```

```
        {
            int w = e[v][j].w+sum-e[v][0].w;
            if(w<weight[e[v][j].u])
            {
                weight[e[v][j].u] = w;
                touse[e[v][j].u]=e[v][j].idx;
            }
        }
        par[v] = super;
    }
    e[super].clear();
    for(j=0; j<n; j++) if(par[j] != par[par[j]]) par[j] = par[par[j]];
    for(j=0; j<n; j++){
        if(weight[j]<INF && par[j]!= super)
        {
            edge ed = edge(j,super,weight[j]);
            ed.idx = touse[j];
            e[super].push_back(ed);
        }
    }
    sort(e[super].begin(),e[super].end());
    for(j=0; j<e[super].size(); j++)
    {
        edge ed=e[super][j];
    }
  }
}
int sum = 0;
for(i=0; i<n; i++){
    if(i!=root && par[i]==i)
    {
        sum += e[i][0].w; /// i'th node's zero'th edge contains the minimum cost after directed_mst algo.
    }
  }
  return sum;
}
int ispossible(int root)
{
  int i,j,u,v;
  for(i=0; i<n; i++)
  {
    for(j=0; j<e[i].size(); j++)
    {
      g[e[i][j].u].push_back(e[i][j].v);
    }
  }
  queue<int>q;
  q.push(root);
  mem(color,0);
  color[root] = 1;
  while(!q.empty())        ///BFS to check graph connectivity.
  {
    u = q.front();
    q.pop();
    for(i=0; i<g[u].size(); i++)
    {
      v = g[u][i];
```

```
            if(color[v]) continue;
            color[v] = 1;
            q.push(v);
        }
    }
    for(i=0; i<n; i++) if(!color[i]) return -1;
    return directed_mst(root);
}
///Beware!!!0-indexed
int main()
{
    int i,j,k;
    edge ed;
    int root;
    cin>>n>>m;
    for(i=0; i<m; i++)
    {
        cin>>ed.u>>ed.v>>ed.w;
        ed.u--;
        ed.v--;
        ed.idx = i;
        e[ed.v].push_back(ed);
        edges.push_back(ed);
    }
    root=0;///select the root of the rooted minimum spanning tree
    int res = ispossible(root);
    if(res == -1) cout<<"impossible\n";
    else
    {
        mem(used,0);
        mem(color,0);
        for(i=(int)choosed.size()-1; i>=0; i--)
        {
            edge ed = edges[choosed[i]];
            if(color[ed.v]) continue;
            color[ed.v] = 1;
            used[choosed[i]] = true;
        }
        cout<<res<<nl;
        for(i=0; i<m; i++) if(used[i]) cout<<i+1<<' ';
        cout<<nl;
    }
    return 0;
}
```

45. Down Trick On Tree

Notes

When we need to do path queries without any update and if path query has following charactaristics,
path(u,v)=path(u,lca)+path(v,lca)-path(root,lca)-path(root,par[lca])
or similar formula, Then these types of problems can be solved using down trick on tree.
Again all pair path sum or similar problems can be solved with this technique.

Down Trick Standard

```
//There's a tree, with each vertex assigned a number. For each query (a, b, c), you are asked whether there is a vertex on the path from a to b, which is assigned number c?
vi g[N];
int dep[N],par[N][20],a[N];
void dfs(int u,int pre=0)
{
   dep[u]=dep[pre]+1;
   par[u][0]=pre;
   for(int i=1;i<18;i++) par[u][i]=par[par[u][i-1]][i-1];
   for(auto v:g[u]){
      if(v==pre) continue;
      dfs(v,u);
   }
}
int lca(int u,int v)
{
   if(dep[u]<dep[v]) swap(u,v);
   for(int i=17;i>=0;i--) if(dep[par[u][i]]>=dep[v]) u=par[u][i];
   if(u==v) return u;
   for(int i=17;i>=0;i--) if(par[u][i]!=par[v][i]) u=par[u][i],v=par[v][i];
   return par[u][0];
}
int cnt[N],ans[N];
vpii q[N];
void down(int u,int pre=0)
{
   cnt[a[u]]++;
//now here in cnt array we have all information from the path root to u
   for(auto x:q[u]){
      int idx=x.F,val=x.S;
      if(idx<0) ans[-idx]-=cnt[val];
      else ans[idx]+=cnt[val];
   }
   for(auto v:g[u]){
      if(v==pre) continue;
      down(v,u);///Don't make any silly mistake by typing here dfs(v,u)
   }
   cnt[a[u]]--;
}
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,u,v,c,que;
   while(cin>>n>>m){
      for(i=1;i<=n;i++) cin>>a[i];
      for(i=1;i<n;i++) cin>>u>>v,g[u].eb(v),g[v].eb(u);
      dfs(1);
      for(i=1;i<=m;i++){
         cin>>u>>v>>c;
         q[u].eb(i,c);
         q[v].eb(i,c);
         int lc=lca(u,v);
         q[lc].eb(-i,c);
         q[par[lc][0]].eb(-i,c);
      }
```

```
    down(1);
    for(i=1;i<=m;i++){
      if(ans[i]) cout<<"Find\n";
      else cout<<"NotFind\n";
    }
    cout<<nl;
    for(i=1;i<=n;i++) g[i].clear();
    for(i=1;i<=m;i++) q[i].clear(),ans[i]=0;
  }
  return 0;
}
```

46. Lowest Common Ancestor

```
vi g[N];
int par[N][20],dep[N],sz[N];
void dfs(int u,int pre)
{
  par[u][0]=pre;
  dep[u]=dep[pre]+1;
  sz[u]=1;
  for(int i=1;i<=18;i++) par[u][i]=par[par[u][i-1]][i-1];
  for(auto v:g[u]){
    if(v==pre) continue;
    dfs(v,u);
    sz[u]+=sz[v];
  }
}
int lca(int u,int v)
{
  if(dep[u]<dep[v]) swap(u,v);
  for(int k=18;k>=0;k--) if(dep[par[u][k]]>=dep[v]) u=par[u][k];
  if(u==v) return u;
  for(int k=18;k>=0;k--) if(par[u][k]!=par[v][k])
u=par[u][k],v=par[v][k];
  return par[u][0];
}
int kth(int u,int k)
{
  for(int i=0;i<=18;i++) if(k&(1<<i)) u=par[u][i];
  return u;
}
int dist(int u,int v)
{
  int lc=lca(u,v);
  return dep[u]+dep[v]-2*dep[lc];
}
int main()
{
  BeatMeScanf;
  int i,j,k,n,m,u,v,q;
  cin>>n;
  for(i=1;i<n;i++) cin>>u>>v,g[u].pb(v),g[v].pb(u);
  dfs(1,0);
  cin>>q;
  while(q--){
    cin>>u>>v;
    cout<<dist(u,v)<<nl;
  }
```

```
    return 0;
}
```

47. Shortest Paths

0-1 BFS

```
///minimum weight from (0,0) to (n-1,m-1) where weight (x1,y1) to (x2,y2)=(s[x1][y1]!=s[x2][y2])
int ans[N][N];
string s[N];
int main()
{
    BeatMeScanf;
    int i,j,k,n,m,t,x,y;
    cin>>t;
    while(t--){
        cin>>n>>m;
        for(i=0;i<n;i++) cin>>s[i];
        deque<pii>d;
        for(i=0;i<n;i++) for(j=0;j<m;j++) ans[i][j]=1e9;
        ans[0][0]=0;
        d.push_front({0,0});
        while(!d.empty()){
            tie(x,y)=d.front();
            d.pop_front();
            for(i=0;i<4;i++){
                int nx=x+dx[i];
                int ny=y+dy[i];
                if(valid(nx,ny)){
                    int w=(s[nx][ny]!=s[x][y]);
                    if(ans[x][y]+w<ans[nx][ny]){
                        ans[nx][ny]=ans[x][y]+w;
                        if(w==0) d.push_front({nx,ny});
                        else d.push_back({nx,ny});
                    }
                }
            }
        }
        cout<<ans[n-1][m-1]<<nl;
    }
    return 0;
}
```

Dijkstra's Algorithm

```
vpll g[N];
int main()
{
    BeatMeScanf;
    ll i,j,k,n,m,u,v,w;
    cin>>n>>m;
    for(i=0;i<m;i++) cin>>u>>v>>w,g[u].eb(v,w),g[v].eb(u,w);
    PQ<pll,vpll,greater<pll>>q;
    vll d(n+1,inf);
    vll par(n+1,0);
    q.push({0,1});
    d[1]=0;
    while(!q.empty()){
        tie(w,u)=q.top();
```

```
      q.pop();
      for(auto x:g[u]){
         v=x.F,w=x.S;
         if(d[u]+w<d[v]){
            par[v]=u;
            d[v]=d[u]+w;
            q.push({d[v],v});
         }
      }
   }
   if(d[n]==inf) return cout<<-1<<nl,0;
   vll path;
   for(ll nw=n;nw!=0;nw=par[nw]) path.eb(nw);
   rev(path);
   for(auto x:path) cout<<x<<' ';
   return 0;
}
```

Bellman-Ford Algorithm

```
#define INF 2e9
struct st
{
   int u,v,w;
}e[N];
///1-based
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,s,t;
   cin>>n>>m;
   for(i=1;i<=m;i++){
      cin>>e[i].u>>e[i].v>>e[i].w;
   }
   cin>>s>>t;
   vector<int> d (n+1, INF);
   d[s] = 0;
   vector<int> p (n+1,-1);
   int cnt=0;
   for (;;)
   {
      bool any = false;
      for (i = 1; i <=m; ++i){
         if (d[e[i].u] < INF){
            if (d[e[i].v] > d[e[i].u] + e[i].w)
            {
               d[e[i].v] = d[e[i].u] + e[i].w;
               p[e[i].v] = e[i].u;
               any = true;
            }
         }
      }
      if (!any)  break;
      ++cnt;
      if(cnt>=n){
         cout<<"Negative cycle detected\n";
         return 0;
      }
   }
```

```
    if (d[t] == INF) cout << "No path from " << s << " to " << t << ".";
    else
    {
        cout<<d[t]<<nl;
        vector<int> path;
        for (int cur = t; cur != -1; cur = p[cur]) path.push_back (cur);
        reverse (path.begin(), path.end());
        cout << "Path from " << s << " to " << t << ": ";
        for (i=0; i<path.size(); ++i) cout << path[i] << ' ';
    }
    return 0;
}
```

Shortest Path Faster Algorithm

Complexity:

Average:$O(m)$

Worst:$O(nm)$

```
///it works for graph with negative edges
#define INF 2e9
vpii g[N];
///0-based
///directed graph
int main()
{
    BeatMeScanf;
    int i,j,k,n,m,u,v,w;
    cin>>n>>m;
    for(i=0;i<m;i++){
        cin>>u>>v>>w;
        --u;
        --v;
        g[u].eb(v,w);
    }
    vector<int>d;
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;
    int s=0;
    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : g[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n){
```

```
                cout<<"Negative cycle detected\n";
                return 0;
            }
        }
    }
  }
  for(i=0;i<n;i++) cout<<d[i]<<' ';
  return 0;
}
```

Floyd-Warshall Algorithm

```
#define INF 2e9
int d[N][N];
///0-based
int main()
{
  BeatMeScanf;
  int i,j,k,n,m,u,v,w;
  cin>>n>>m;
  for(i=0;i<n;i++) for(j=0;j<n;j++) d[i][j]=INF;
  for(i=0;i<n;i++) d[i][i]=0;
  for(i=0;i<m;i++){
    cin>>u>>v>>w;
    --u;
    --v;
    d[u][v]=w;
    d[v][u]=w;
  }
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        if (d[i][k] < INF && d[k][j] < INF) d[i][j] = min(d[i][j], d[i][k] +
d[k][j]);
      }
    }
  }
  ///For a undirected graph
  /// The graph has a negative cycle if at the end of the algorithm,
  ///the distance from a vertex v to itself is negative.
  ///for undirected graph d[u][v]=w hobe only
  ///not d[u][v]=w and d[v][u]=w eksathe
  return 0;
}
```

48. Dominator Tree

```
const int N = int(1e5)+10;
const int M = int(5e5)+10;
vi g[N];
vi t[N],rg[N],bucket[N];
int sdom[N],par[N],dom[N],dsu[N],label[N];
int arr[N],rev[N],T;
int find_(int u,int x=0)
{
        if(u==dsu[u])return x?-1:u;
        int v = find_(dsu[u],x+1);
        if(v<0)return u;
        if(sdom[label[dsu[u]]] < sdom[label[u]])
```

```
                label[u] = label[dsu[u]];
        dsu[u] = v;
        return x?v:label[u];
}
void union_(int u,int v) ///Add an edge u-->v
{
        dsu[v]=u;
}
void dfs(int u)
{
        T++;arr[u]=T;rev[T]=u;
        label[T]=T;sdom[T]=T;dsu[T]=T;
        for(int i=0;i<g[u].size();i++)
        {
                int w = g[u][i];
                if(!arr[w])dfs(w),par[arr[w]]=arr[u];
                rg[arr[w]].eb(arr[u]);
        }
}
int main()
{
        int i,j,k,n,m,u,v,w;
        cin>>n>>m;
        for(int i=0;i<m;i++)
        {
                cin>>u>>v;
                g[u].eb(v);
        }
        ///Build Dominator tree
        dfs(1);
        n=T;
        for(i=n;i>=1;i--)
        {
                for(j=0;j<rg[i].size();j++)
                        sdom[i] = min(sdom[i],sdom[find_(rg[i][j])]);
                if(i>1)bucket[sdom[i]].eb(i);
                for(j=0;j<bucket[i].size();j++)
                {
                        w = bucket[i][j];
                        v = find_(w);
                        if(sdom[v]==sdom[w])dom[w]=sdom[w];
                        else dom[w] = v;
                }
                if(i>1)union_(par[i],i);
        }
        for(int i=2;i<=n;i++)
        {
                if(dom[i]!=sdom[i])
                        dom[i]=dom[dom[i]];
                t[rev[i]].eb(rev[dom[i]]);
                t[rev[dom[i]]].eb(rev[i]);
        }
        ///make sure to use t[] for the the dominator tree
        return 0;
}
```

49. 2-SAT

```
///every door has exactly 2 switches to control it, is there any combination of switches if the door
///can be unlocked at the same time
///you are also given the initial state of the doors
bool vis[N];
vi g[N],r[N],vec;
int idx[N];
void dfs1(int u)
{
   vis[u]=1;
   for(auto v:g[u]) if(!vis[v]) dfs1(v);
   vec.eb(u);
}
vi comp;
void dfs2(int u)
{
   comp.eb(u);
   vis[u]=1;
   for(auto v:r[u]) if(!vis[v]) dfs2(v);
}
vi d[N];
int a[N];
int main()
{
   BeatMeScanf;
   int i,j,k,n,m,u,v;
   cin>>n>>m;
   for(i=1;i<=n;i++) cin>>a[i];
   for(i=1;i<=m;i++){
      cin>>k;
      while(k--){
         cin>>u;
         d[u].eb(i);
      }
   }
   for(i=1;i<=n;i++){
      u=d[i][0];
      v=d[i][1];
      if(a[i]==0){
         g[u+m].eb(v);
         g[v+m].eb(u);
         g[u].eb(v+m);
         g[v].eb(u+m);

         r[v].eb(u+m);
         r[u].eb(v+m);
         r[v+m].eb(u);
         r[u+m].eb(v);
      }
      else{
         g[u+m].eb(v+m);
         g[v+m].eb(u+m);
         g[u].eb(v);
         g[v].eb(u);

         r[v+m].eb(u+m);
```

```
            r[u+m].eb(v+m);
            r[v].eb(u);
            r[u].eb(v);
        }
    }
    for(i=1;i<=2*m;i++) if(!vis[i]) dfs1(i);
    rev(vec);
    mem(vis,0);
    int scc=0;
    for(auto u:vec){
        if(!vis[u]){
            comp.clear();
            dfs2(u);
            scc++;
            for(auto x:comp) idx[x]=scc;
        }
    }
    for(i=1;i<=m;i++) if(idx[i]==idx[i+m]) no();
    cout<<"YES\n";
    for(i=1;i<=m;i++) if(idx[i]>idx[i+m]) cout<<i<<' ';
    return 0;
}
```

GEOMETRY

50. Geometry 2D

```
const int mod=1e9+7;
const int mxn=3e5+9;
const double eps=1e-9;
const double PI=acos(-1.0);
const int mxp=2100;
//ll gcd(ll a,ll b){while(b){ll x=a%b;a=b;b=x;}return a;}
//ll lcm(ll a,ll b){return a/gcd(a,b)*b;}
//ll qpow(ll n,ll k) {ll ans=1;assert(k>=0);n%=mod;while(k>0){if(k&1)
ans=(ans*n)%mod;n=(n*n)%mod;k>>=1;}return ans%mod;}
int sign(double d)
{
    if (fabs(d)<eps)return 0;
    return d>eps?1:-1;
}
inline double sqr(double x){return x*x;}
struct PT
{
    double x,y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    void in() {
        sf("%lf %lf",&x,&y);
    }
    void out() {
        pf("%.10f %.10f\n",x,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 sign(a.x-x)==0?sign(y-a.y)<0:x<a.x;
}
bool operator>(PT a)const
{
    return sign(a.x-x)==0?sign(y-a.y)>0:x>a.x;
}
double val()
{
    return sqrt(x*x+y*y);
}
double val2()
    {
        return (x*x+y*y);
    }
    double arg()
    {
        return atan2(y,x);
    }
    //return point that is truncated the distance from center to r
    PT trunc(double r){
        double l=val();
        if (!sign(l)) return *this;
        r/=l;
        return PT(x*r,y*r);
    }
};
istream& operator >> (istream& is,PT &a)
{
    return is>>a.x>>a.y;
}
ostream& operator << (ostream& os,const PT &a)
{
    return os<<fixed<<setprecision(10)<<a.x<<' '<<a.y;
}
double dist(PT a,PT b)
{
    return sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));
}
double dist2(PT a,PT b)
{
```

```
   return sqr(a.x-b.x)+sqr(a.y-b.y);
}
double dot(PT a,PT b)
{
   return a.x*b.x+a.y*b.y;
}
double cross(PT a,PT b)
{
   return a.x*b.y-a.y*b.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 th)
{
   return PT(a.x*cos(th)-a.y*sin(th),a.x*sin(th)+a.y*cos(th));
}
PT rotatecw(PT a,double th)
{
   return PT(a.x*cos(th)+a.y*sin(th),-a.x*sin(th)+a.y*cos(th));
}
double angle_between_vectors(PT a, PT b)
{
   double costheta=dot(a,b)/a.val()/b.val();
   return acos(fmax(-1.0,fmin(1.0,costheta)));
}
double rad_to_deg(double r) {
 return (r * 180.0 / PI);
}

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



// a line is defined by two points
struct line
{
   PT a,b;
   line(){}
   line(PT _a,PT _b)
   {
      a=_a;
      b=_b;
   }
   //ax+by+c=0
   line(double _a,double _b,double _c)
   {
      if (sign(_a)==0)
      {
         a=PT(0,-_c/_b);
         b=PT(1,-_c/_b);
```

```
    }
    else if (sign(_b)==0)
    {
        a=PT(-_c/_a,0);
        b=PT(-_c/_a,1);
    }
    else
    {
        a=PT(0,-_c/_b);
        b=PT(1,(-_c-_a)/_b);
    }
}
void in()
{
    a.in();
    b.in();
}
PT vec() const
{
    return (b-a);
}
bool operator==(line v)
{
    return (a==v.a)&&(b==v.b);
}
PT cross_point(line v){
    double a1=cross(v.b-v.a,a-v.a);
    double a2=cross(v.b-v.a,b-v.a);
    return PT((a.x*a2-b.x*a1)/(a2-a1),(a.y*a2-b.y*a1)/(a2-a1));
    }
};
istream &operator>>(istream &is, line &a) {
    return is >> a.a >> a.b;
}
ostream &operator<<(ostream &os, line &p) {
    return os << p.a << " to " << p.b;
}



// 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).val()) * 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).val2();
}
// 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*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).val());
}
```

```
//return 1 if  point c is on line segment ab
bool is_point_on_seg(PT a,PT b,PT c)
{
   double d1=dist(a,c)+dist(c,b);
   double d2=dist(a,b);
   if(fabs(d1-d2)<eps) return 1;
   else return 0;
}
//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 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));
}
//returns a parallel line of line ab in counterclockwise direction with d distance from ab
line get_parallel_line(PT a,PT b,double d)
{
   return line(point_along_line(a,rotateccw90(b-a)+a,d),point_along_line(b,rotatecw90(a-b)+b,d));
}
//Return a tangent line of line ab which intersects
//with it at point c in counterclockwise direction
line get_perpendicular_line(PT a,PT b,PT c)
{
   return line(c+rotateccw90(a-c),c+rotateccw90(b-c));
}
//relation of point p with line ab
//return
//1 if point is ccw with line
//2 if point is cw with 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;
}
//return
//0 if not parallel
//1 if parallel
//2 if collinear
bool 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;
}
double area_of_triangle(PT a,PT b,PT c)
{
    return fabs(cross(b-a,c-a)/2.0);
}
//radian angle of <bac
double angle(PT b,PT a,PT c)
{
    return angle_between_vectors(b-a,c-a);
}

//orientation of point a,b,c
double orient(PT a,PT b,PT c)
{
    return cross(b-a,c-a);
}
//is point p within angle <bac
bool is_point_in_angle(PT b,PT a,PT c,PT p)
{
    assert(fabs(orient(a,b,c)-0.0)>0);
    if(orient(a,c,b)<0) swap(b,c);
    return orient(a,c,p)>=0&&orient(a,b,p)<=0;
}
//equation of bisector line of <bac
line bisector(PT b,PT a,PT c)
{
    PT unit_ab=(b-a)/(b-a).val();
    PT unit_ac=(c-a)/(c-a).val();
    PT l=unit_ab+unit_ac;
    return line(l.y,-l.x,l.x*a.y-l.y*a.x);
}
//sort points in counterclockwise;
bool half(PT p)
{
    return p.y>0.0||(p.y==0.0&&p.x<0.0);
}
void polar_sort(vector<PT>&v)
{
    sort(all(v),[](PT a,PT b){return
make_tuple(half(a),0.0)<make_tuple(half(b),cross(a,b));});
}
//intersection point between ab and cd
//assuming unique intersection exists
bool line_line_intersection(PT a,PT b,PT c,PT d,PT &out)
{
    double a1=a.y-b.y;
    double b1=b.x-a.x;
    double c1=cross(a,b);
    double a2=c.y-d.y;
    double b2=d.x-c.x;
    double c2=cross(c,d);
    double det=a1*b2-a2*b1;
    if(det==0) return 0;
    out=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 &out)
{
   double oa=orient(c,d,a);
   double ob=orient(c,d,b);
   double oc=orient(a,b,c);
   double od=orient(a,b,d);
   //proper intersection exists iff opposite tmps
   if(oa*ob<0&&oc*od<0){
      out=(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 out;
   if (seg_seg_intersection(a,b,c,d,out)) return {out};
   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
//return
//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=orient(c,d,a);
   double q=orient(c,d,b);
   if(sign(p)==0&&sign(q)==0) return 2;
   //proper intersection exists iff opposite tmps
   else if(p*q<0) return 1;
   else return 0;
}
//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)});
}
```

```
//a circle is defined by a center and radius
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){};
   //compute circle given three points i.e. circumcircle of a triangle
   circle(PT a,PT b,PT c){
      b=(a+b)/2.0;
      c=(a+c)/2.0;
      line_line_intersection(b,b+rotatecw90(a-b),c,c+rotatecw90(a-
c),p);
      r=dist(a,p);
   }
   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);
   }
   void in(){
      p.in();scanf("%lf",&r);
   }
   void out(){
      printf("%.10f %.10f %.10f\n",p.x,p.y,r);
   }
   bool operator==(circle v){
      return ((p==v.p)&&sign(r-v.r)==0);
   }
   bool operator<(circle v)const{
      return ((p<v.p)||(p==v.p)&&sign(r-v.r)<0);
   }
   double area(){return PI*sqr(r);}
   double circumference(){return 2.0*PI*r;}
};
istream &operator>>(istream &is, circle &a) {
   return is >> a.p >> a.r;
}
ostream &operator<<(ostream &os, circle &a) {
   return os << a.p <<" "<< a.r;
}
//if point is inside circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_point_relation(PT p,double r,PT b)
{
   double dst=dist(p,b);
```

```
    if (sign(dst-r)<0)return 2;
    if (sign(dst-r)==0)return 1;
    return 0;
}
//if segment is inside circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_seg_relation(PT p,double r,PT a,PT b)
{
    double dst=dist_from_point_to_seg(a,b,p);
    if (sign(dst-r)<0)return 2;
    if (sign(dst-r)==0)return 1;
    return 0;
}
//if line cross circle
//return
//0 outside
//1 on circumference
//2 inside circle
int circle_line_relation(PT p,double r,PT a,PT b)
{
    double dst=dist_from_point_to_line(a,b,p);
    if (sign(dst-r)<0)return 2;
    if (sign(dst-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);
    double B = dot(a, b);
    double C = dot(a, a) - r*r;
    double D = B*B - A*C;
    if (D < -eps) return ret;
    ret.pb(c+a+b*(-B+sqrt(D+eps))/A);
    if (D > eps) ret.pb(c+a+b*(-B-sqrt(D))/A);
    return ret;
}
//return
//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;
}
// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
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.pb(a+v*x + rotateccw90(v)*y);
    if (y > 0) ret.pb(a+v*x - rotateccw90(v)*y);
    return ret;
}
// returns two circle c1,c2 through points a,b of radius r
// returns 0 if there is no such circle
// 1 if one circle
// 2 if two circle
int getcircle(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
// returns 0 for no circle ,1 if c1=c2 ,2 if c1!=c2
int getcircle(line u,PT q,double r1,circle &c1,circle &c2)
{
    double dis=dist_from_point_to_line(u.a,u.b,q);
    if (sign(dis-r1*2)>0) return 0;
    if (sign(dis)==0)
    {
        c1.p=(q+rotateccw90(u.vec())).trunc(r1);
        c2.p=(q+rotatecw90(u.vec())).trunc(r1);
        c1.r=c2.r=r1;
        return 2;
    }
    line
u1=line((u.a+rotateccw90(u.vec())).trunc(r1),(u.b+rotateccw90(u.ve
c())).trunc(r1));
    line
u2=line((u.a+rotatecw90(u.vec())).trunc(r1),(u.b+rotatecw90(u.vec()
)).trunc(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.eb(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)
{
    circle u(a,r1),v(b,r2);
    int rel=circle_circle_relation(a,r1,b,r2);
    if (rel>=4) return 0.0;
    if (rel<=2) return min(u.area(),v.area());
    double d=dist(u.p,v.p);
    double hf=(u.r+v.r+d)/2.0;
    double ss=2*sqrt(hf*(hf-u.r)*(hf-v.r)*(hf-d));
    double a1=acos((u.r*u.r+d*d-v.r*v.r)/(2.0*u.r*d));
    a1=a1*u.r*u.r;
    double a2=acos((v.r*v.r+d*d-u.r*u.r)/(2.0*v.r*d));
    a2=a2*v.r*v.r;
    return a1+a2-ss;
}
//tangent lines i.e. sporshoks  from point q to circle with center p and
radius r
int tangent_lines_from_point(PT p,double r,PT q,line &u,line &v)
{
    int x=circle_point_relation(p,r,q);
    if (x==2) return 0;
    if (x==1)
    {
        u=line(q,q+rotateccw90(q-p));
        v=u;
        return 1;
    }
    double d=dist(p,q);
    double l=sqr(r)/d;
    double h=sqrt(sqr(r)-sqr(l));
    u=line(q,p+((q-p).trunc(l)+(rotateccw90(q-p).trunc(h))));
    v=line(q,p+((q-p).trunc(l)+(rotatecw90(q-p).trunc(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.val(), 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.pb({c1 + v*r1, c2 + v*r2});
    }
    u=line(out[0].F,out[0].S);
    if(out.size()==2) v=line(out[1].F,out[1].S);
    return 1 + (h2 > 0);
}


//a polygon is defined by n points
//here l[] array stores lines of the polygon
struct polygon
{
    int n;
    PT p[mxp];
    line l[mxp];
    void in(int _n){
        n=_n;
        for (int i=0;i<n;i++)   p[i].in();
    }
    void add(PT q){p[n++]=q;}
    void getline(){
        for (int i=0;i<n;i++)
            l[i]=line(p[i],p[(i+1)%n]);
    }
    double getcircumference()
    {
        double sum=0;
        int i;
        for (i=0;i<n;i++)
        {
            sum+=dist(p[i],p[(i+1)%n]);
        }
        return sum;
    }
    double getarea()
    {
        double sum=0;
        int i;
        for (i=0;i<n;i++)
        {
            sum+=cross(p[i],p[(i+1)%n]);
        }
        return fabs(sum)/2;
    }
    //returns 0 for cw, 1 for ccw
    bool getdir()
    {
        double sum=0;
        int i;
        for (i=0;i<n;i++)
        {
            sum+=cross(p[i],p[(i+1)%n]);
        }
        if (sign(sum)>0)return 1;
        return 0;
    }
```

```
struct cmp{
   PT p;
   cmp(const PT &p0){p=p0;}
   bool operator()(const PT &aa,const PT &bb){
      PT a=aa,b=bb;
      int d=sign(cross(a-p,b-p));
      if (d==0) return sign(dist(a,p)-dist(b,p))<0;
      return d>0;
   }
};
//sorting in convex_hull order
void norm(){
   PT mi=p[0];
   for (int i=1;i<n;i++)mi=min(mi,p[i]);
   sort(p,p+n,cmp(mi));
}
//returns convex hull in convex (monotone chain)
void getconvex(polygon &convex)
{
   int i,j,k;
   sort(p,p+n);
   convex.n=n;
   for (i=0;i<min(n,2);i++)
   {
      convex.p[i]=p[i];
   }
   if (n<=2)return;
   int &top=convex.n;
   top=1;
   for (i=2;i<n;i++)
   {
      while (top&&cross(convex.p[top]-p[i],convex.p[top-1]-
p[i])<=0)
         top--;
      convex.p[++top]=p[i];
   }
   int temp=top;
   convex.p[++top]=p[n-2];
   for (i=n-3;i>=0;i--)
   {
      while (top!=temp&&cross(convex.p[top]-p[i],convex.p[top-1]-
p[i])<=0)
         top--;
      convex.p[++top]=p[i];
   }
}
//checks if convex
bool isconvex()
{
   bool s[3];
   memset(s,0,sizeof(s));
   int i,j,k;
   for (i=0;i<n;i++)
   {
      j=(i+1)%n;
      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;
}
//used for later function
double xmult(PT a, PT b, PT c)
{
    return cross(b - a,c - a);
}
// returns
// 3 - if q is a vertex
        // 2 - if on a side
        // 1 - if inside
        // 0 - if outside
int relation_point(PT q){
    int i,j;
    for (i=0;i<n;i++){
        if (p[i]==q)return 3;
    }
    getline();
    for (i=0;i<n;i++){
        if (is_point_on_seg(l[i].a,l[i].b,q))return 2;
    }
    int cnt=0;
    for (i=0;i<n;i++){
        j=(i+1)%n;
        int k=sign(cross(q-p[j],p[i]-p[j]));
        int u=sign(p[i].y-q.y);
        int v=sign(p[j].y-q.y);
        if (k>0&&u<0&&v>=0)cnt++;
        if (k<0&&v<0&&u>=0)cnt--;
    }
    return cnt!=0;
}
//returns minimum enclosing circle
void find_(int st,PT tri[],circle &c){
    if (!st) c=circle(PT(0,0),-2);
    if (st==1) c=circle(tri[0],0);
    if (st==2) c=circle((tri[0]+tri[1])/2.0,dist(tri[0],tri[1])/2.0);
    if (st==3) c=circle(tri[0],tri[1],tri[2]);
}
void solve(int cur,int st,PT tri[],circle &c){
    find_(st,tri,c);
    if (st==3)return;
    int i;
    for (i=0;i<cur;i++){
        if (sign(dist(p[i],c.p)-c.r)>0){
            tri[st]=p[i];
            solve(i,st+1,tri,c);
        }
    }
}
circle minimum_enclosing_circle(){
    random_shuffle(p,p+n);
    PT tri[4];
    circle c;
    solve(n,0,tri,c);
    return c;
}
```

```
};


//stores some polygons
struct polygons
{
   vector<polygon>p;
   polygons(){p.clear();}
   void clear(){p.clear();}
   void push(polygon q){if (sign(q.getarea()))p.pb(q);}
   vector<pair<double,int> >e;
   //used for later use
   void ins(PT s,PT t,PT X,int i){
      double r=fabs(t.x-s.x)>eps?(X.x-s.x)/(t.x-s.x):(X.y-s.y)/(t.y-s.y);
      r=fmin(r,1.0);r=fmax(r,0.0);
      e.pb(MP(r,i));
   }
   double polyareaunion(){
      double ans=0.0;
      int c0,c1,c2,i,j,k,w;
      for (i=0;i<p.size();i++)
         if (p[i].getdir()==0)reverse(p[i].p,p[i].p+p[i].n);
      for (i=0;i<p.size();i++){
         for (k=0;k<p[i].n;k++){
            PT &s=p[i].p[k],&t=p[i].p[(k+1)%p[i].n];
            if (!sign(cross(s,t)))continue;
            e.clear();
            e.pb(MP(0.0,1));
            e.pb(MP(1.0,-1));
            for (j=0;j<p.size();j++)if (i!=j){
               for (w=0;w<p[j].n;w++){
                  PT          a=p[j].p[w],b=p[j].p[(w+1)%p[j].n],c=p[j].p[(w-
1+p[j].n)%p[j].n];
                  c0=sign(cross(t-s,c-s));
                  c1=sign(cross(t-s,a-s));
                  c2=sign(cross(t-s,b-s));
                  if (c1*c2<0)ins(s,t,line(s,t).cross_point(line(a,b)),-c2);
                  else if (!c1&&c0*c2<0)ins(s,t,a,-c2);
                  else if (!c1&&!c2){
                     int c3=sign(cross(t-s,p[j].p[(w+2)%p[j].n]-s));
                     int dp=sign(dot(t-s,b-a));
                     if (dp&&c0)ins(s,t,a,dp>0?c0*((j>i)^(c0<0)):-(c0<0));
                     if (dp&&c3)ins(s,t,b,dp>0?-c3*((j>i)^(c3<0)):c3<0);
                  }
               }
            }
            sort(e.begin(),e.end());
            int ct=0;
            double tot=0.0,last;
            for (j=0;j<e.size();j++){
               if (ct==2)tot+=e[j].first-last;
               ct+=e[j].second;
               last=e[j].first;
            }
            ans+=cross(s,t)*tot;
         }
      }
      return fabs(ans)*0.5;
```

```
    }
};
int main()
{
    fast;
    PT a(1,0),b(2,0),c(1,2),d(3,4);
    circle x(a,b,c,0);
    x.out();
    return 0;
}
```

51. Convex Hull

```
int sign(double d)
{
    if (fabs(d)<eps)return 0;
    return d>eps?1:-1;
}
struct PT
{
    double x,y;
    PT() {}
    PT(double x, double y) : x(x), y(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);
    }
    bool operator==(PT a)const
    {
        return sign(a.x-x)==0&&sign(a.y-y)==0;
    }
    bool operator<(PT a)const
    {
        return sign(a.x-x)==0?sign(y-a.y)<0:x<a.x;
    }
    bool operator>(PT a)const
    {
        return sign(a.x-x)==0?sign(y-a.y)>0:x>a.x;
    }
    double val()
    {
        return sqrt(x*x+y*y);
    }
};
double cross(PT a,PT b)
{
    return a.x*b.y-a.y*b.x;
}
bool cmp(PT p,PT q)
{
    return mt(p.x,p.y)<mt(q.x,q.y);
}
vector<PT> hull(vector<PT> a) {
        sort(all(a),cmp);
```

```
        a.resize(unique(a.begin(), a.end()) - a.begin());
        if((int)a.size()==1) return a;
        vector<PT> res;
        int l = 0;
        for(int i=0;i<2;i++) {
                for(auto & C : a) {
                        while((int) res.size() >= l + 2) {
                                PT A = res[(int) res.size() - 2];
                                PT B = res.back();
                                if(cross((C-A),(B-A)) >= 0) break;
                                res.pop_back();
                        }
                        res.pb(C);
                }
                res.pop_back();
                reverse(a.begin(), a.end());
                l = (int) res.size();
        }
        return res;
}
int main()
{
   fast;
   ll i,j,k,n,m;
   cin>>n;
   vector<PT> v(n);
   for(i=0;i<n;i++) cin>>v[i].x>>v[i].y;
   vector<PT>a=hull(v);
   ll sz=a.size();
   double ans=0;
   for(i=0;i<sz;i++) ans+=(a[(i+1)%sz]-a[i]).val();
   cout<<fout(10)<<ans<<nl;
   return 0;
}
```

52. Pick's Theorem

Given a certain lattice polygon with non-zero area.

We denote its area by S, the number of points with integer coordinates lying strictly inside the polygon by I and the number of points lying on polygon sides by B.

Then, the Pick's formula states:

$$S = I + \frac{B}{2} - 1$$

In particular, if the values of I and B for a polygon are given, the area can be calculated in O(1) without even knowing the vertices.