Data Structure

- Binary Indexed Tree
- BIT Min - Max
- Segment Tree
- PBDS
- Trie
- Sparse table
- Maximum Sub-array Sum Online Query
- MO's
- Wavelet tree
- Sliding Window Maximum

Number Theory

- Sieve/Spf
- Euler Phi Sieve
- Extended Euclid
- N! Variations
- NCR fixed/any M
- Pollard Rho

Graph

- Kruskal
- LCA
- Euler Tour
- BPM
- Articulation
- SCC
- Bridge
- HLD
- Centroid Decomposition
- Cycles

String

- Hashing
- Z-Algorithm
- KMP
- Suffix Array
- Palindromic Query
- Trie
- K-th LexigrophaicallySmallestSubString

Dynamic Programming

- LIS/LDS/LNDS/LNRS
- Kadane
- Bitmask dp
- MIM
- SOS Dp
- Digit Dp

Geometry

- Basics

Miscellaneous

- Headers

Data Structure

Binary Indexed Tree

```
ll bit[N];
ll get(int idx){
   ll res = 0;
   while(idx){
      res += bit[idx];
      idx -= (idx & -idx);
   }
   return res;
}
void update(int idx, ll val, int n){
   while(idx <= n){
      bit[idx] += val;
      idx += (idx & -idx);
   }
}
void range_update(int l, int r, ll x, int n){
   update(l, x, n);
   update(r + 1, -x, n);
}
```

Min/Max version

```
void build() {
for(int i=1; i<=n; i++)
BIT[i+n]=arr[i];
for(int i = n ; i >= 1 ; --i)
BIT[i] = min(BIT[i << 1], BIT[(i << 1) | 1]); }

void update(int k, int x,int n) {
k += n;
BIT[k] = x;
for(k >>= 1 ; k >= 1 ; k >>= 1)
BIT[k] = min(BIT[k << 1], BIT[(k << 1) | 1]); }

void update(ll l,ll r,ll val) {
for(l+=n,r+=n; l < r ; l>>=1,r>>=1){
if(l&1) BIT[l] = min(BIT[l],val), l++;
if(r&1) r--,BIT[r] = min(BIT[r],val);} }

ll query(ll idx) {
idx += n;
ll ans = inf;
while(idx) ans = min(ans,BIT[idx]), idx >>= 1;
return ans; }

int query(int a, int b) {//a = left b = right
a += n;
b += n;
int res = INT_MAX;
while(a <= b){
if(a & 1)
res = min(res, BIT[a++]);
if(!(b & 1))
res = min(res, BIT[b--]);
a >>= 1;
b >>= 1;}
return res; }
```

Segment Tree

```
int st[4*mx+10],lazy[4*mx+10];
void build_tree(int a[], int i, int low, int high){
   if(low==high) {
      st[i]=a[low]; } else{
      int mid=(low+high)/2;
      build_tree(a, 2*i, low, mid);
      build_tree(a, 2*i+1, mid+1, high);
      st[i]= st[2*i]+st[2*i+1];}}

int query(int i, int low, int high, int left, int right){
   if(lazy[i]!=0) {
      int d=lazy[i];
      lazy[i]=0;
      st[i]+=d*(high-low+1);
      if(low!=high) {
         lazy[2*i]+=d;
         lazy[2*i+1]+=d; }}

   if(left>high || right<low){
      return 0; }
   if(low>=left && high<=right){
      return st[i];}
   int mid=(low+high)/2;
   int l=query(2*i, low, mid, left, right);
   int r=query(2*i+1, mid+1, high, left, right);
    return l+r;}
```

```
void update(int i, int low, int high, int left, int right,
int qval){
   if(lazy[i]!=0) {
      int d=lazy[i];
      lazy[i]=0;
      st[i]+=d*(high-low+1);
      if(low!=high){
         lazy[2*i]+=d;
         lazy[2*i+1]+=d; }  }

  if(left>high || right<low){
      return ;}
   if(low>=left && high<=right){
      int d=(high-low+1)*qval;
      st[i]+=d;

      if(low!=high)  {
         lazy[2*i]+=qval;
         lazy[2*i+1]+=qval; }
      return; }
   int mid=(low+high)/2;
   update(2*i, low, mid, left, right,qval);
   update(2*i+1, mid+1, high, left, right,qval);

   st[i]=st[2*i]+st[2*i+1];}

int main(){
   int n,q;
   scanf("%d%d",&n,&q);
   int arr[n+10];
   for(int i=1; i<=n; i++) {
      scanf("%d",&arr[i]);}
   //cout<<"xx"<<endl;
   build_tree(arr, 1, 1,n);
   while(q--){
      int code;
      cin>>code;
      int l,r,qval;
      if(code==2) {
         scanf("%d%d%d",&l,&r,&qval);
         update(1,1,n,l,r,qval);
         continue; }

      scanf("%d%d",&l,&r);
      int ans= query(1,1,n,l,r);
      printf("%d\n",ans);  }}
```

PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <functional> // for less
using namespace __gnu_pbds;
template <typename T> using ordered_set =  tree<T,
null_type, less_equal<T>, rb_tree_tag,
      tree_order_statistics_node_update>;

void example() {
ordered_set<int> p;
   // value at 3rd index in sorted array.
   cout << "The value at 3 index :: "
        << *p.find_by_order(3) << endl;

   // index of number 6
   cout << "The index of number 6:: " <<
p.order_of_key(6)
        << endl;

   // number 7 not in the set but it will show the
   // index number if it was there in sorted array.
   cout << "The index of number seven :: "
        << p.order_of_key(7) << endl;

   ordered_set<pair<int, int>> s;
   int value;
   cin >> value;
   s.insert({value, s.size()}); // for insertion
   s.erase(s.upper_bound({value, -1})); //for deletion
   s.upper_bound({value, -1}); //for search
 }
// ekta problem er solution multiset erase pbds
ll n, m;
   cin >> n >> m;
   map<ll, pair<ll, ll>> mp;
   ordered_set<array<ll, 2>> st;
   for(int i = 1;i <= n;i++) st.insert({0, 0});
   while(m--){
      ll t, p;
      cin >> t >> p;
```

```
st.erase(st.find_by_order(st.order_of_key({mp[t].first,
mp[t].second})));
      mp[t].first += 1;
      mp[t].second -= p;
      st.insert({mp[t].first, mp[t].second});
      cout << st.order_of_key({mp[1].first,
mp[1].second}) + 1 << endl;
   }
```

TRIE

```
struct Node{
   bool isEnd = false;
   int nxt[2];
   int total =0;
   Node(){
      isEnd = false;
      memset(nxt,-1,sizeof(nxt));}};
Node trie[mx+7];

void insertIntoTrie(string s){
   int now=0;
   trie[now].total++;
   for(auto ch:s){
      int ind = ch-'0';
      if(trie[now].nxt[ind]==-1){
         trie[now].nxt[ind]=++sz;  }
      now=trie[now].nxt[ind];
      trie[now].total++;}}

void deleteFromTrie(string s){
   int now=0;
   trie[now].total--;
   for(auto ch:s){
      int ind = ch-'0';
      int temp = trie[now].nxt[ind];;
      if(trie[temp].total==1){
         trie[now].nxt[ind]=-1;}
      now=temp;
      trie[now].total--;}}

string searchMxInTrie(string s){
   int now=0;
   string ret = "";
   for(auto ch:s){
      int ind = ch-'0';
      ind^=1;
      if(trie[now].nxt[ind]==-1){
         ret+='0';
         ind^=1;}else{
         ret+='1';}
      now=trie[now].nxt[ind];}
   return ret;}
```

Sparse Table

```
void buildSparseTable(int arr[], int n) {
const int N = 1e5 + 9;
int t[N][18], a[N];
void build(int n) {
  for(int i = 1; i <= n; ++i) t[i][0] = a[i];
  for(int k = 1; k < 18; ++k) {
    for(int i = 1; i + (1 << k) - 1 <= n; ++i) {
      t[i][k] = min(t[i][k - 1], t[i + (1 << (k - 1))][k - 1]);
    }
  }
}
int query(int l, int r) {
  int k = 31 - __builtin_clz(r - l + 1);
  return min(t[l][k], t[r - (1 << k) + 1][k]);
}
```

Maximum SubArraySum in Range Query

```
struct Tree {
ll sum, pref, suff, ans;
}tree[4*mx];

Tree Combine(Tree l, Tree r) {
Tree ret; ret.sum  = l.sum+r.sum;
ret.pref = max(l.pref, l.sum+r.pref);
ret.suff = max(r.suff, r.sum+l.suff);
ret.ans  = max(l.suff+r.pref, max(l.ans, r.ans));
return ret; }

void build(ll node, ll start, ll end) {
if(start==end){
tree[node].sum = tree[node].pref = tree[node].suff =
tree[node].ans = max(-inf, (ll)arr[start]);
return; }
```

```
ll mid = (start+end)>>1;
build(node<<1, start, mid);
build(node<<1|1, mid+1, end);
tree[node] = Combine(tree[node<<1],
tree[node<<1|1]);}

Tree query(ll node, ll start, ll end, ll i, ll j) {
if(i>end || j<start) return {-1<<31, -1<<31, -1<<31,
-1<<31};
if(start>=i && end<=j) return tree[node];
ll mid = (start+end)>>1;
Tree x = query(node<<1, start, mid, i, j);
Tree y = query(node<<1|1, mid+1, end, i, j);
return Combine(x, y); }

void update(ll node, ll start, ll end, ll i, ll newvalue)
{
if(i<start || i>end) return;
if(start>=i && end<=i){
tree[node].sum = tree[node].pref = tree[node].suff =
tree[node].ans = max(-inf, newvalue);
return; }
ll mid = (start+end)>>1;
update(node<<1, start, mid, i, newvalue);
update(node<<1|1, mid+1, end, i, newvalue);
tree[node] = Combine(tree[node<<1],
tree[node<<1|1]); }

// Query
build(1,1,n);
update(1,1,n,id,val);
query(1, 1, n, 1, n).ans)
//
```

MO's

```
struct data {
int l,r,idx,k;
bool operator<(const data &b) const {
int x = l/BLOCK_SIZE, y = b.l/BLOCK_SIZE;
if(x != y)
return x < y;
return r < b.r; } };
int BLOCK_SIZE;
ll cnt, ans[MAX];
ll n, q, a[MAX], freq[MAX];
data Q[MAX];
void add(ll x) {
freq[x]++;
if(freq[x] == 1)
cnt++; }
void del(ll x) {
freq[x]--;
if(freq[x] == 0)
cnt--; }

void MO() {
BLOCK_SIZE = sqrt(n);
sort(Q,Q+q,cmp);
int st = 1, en = 0;
for(int i=0; i<q; i++){
int l = Q[i].l , r = Q[i].r , idx = Q[i].idx;
while(en < r) { en++;  add(a[en]); }
while(en > r) { del(a[en]); en--; }
while(st > l) { st--;  add(a[st]); }
while(st < l) { del(a[st]); st++; }
ans[idx] = cnt; } }
```

MO's Online

```
const int magic = 2154; const int inf = 1000000000;
vector <int> ind, val, id;
struct query {
int l, r, id;
query () {}
query (int l, int r, int id) : l(l), r(r), id(id) {}
};
vector <query> g[100][100];
int cnt[300010], ans[100010], aux[300010], l, r,
a[100010], original[100010], n;

void add(int x) {
aux[cnt[a[x]]] -= 1;
cnt[a[x]] += 1;
aux[cnt[a[x]]] += 1; }

void del(int x) {
aux[cnt[a[x]]] -= 1;
cnt[a[x]] -= 1;
aux[cnt[a[x]]] += 1; }
```

```
int get_ans() { // cout << l << " " << r << endl;
for(int i = 0; ; i++) {
if(aux[i] == 0) {
return i; } } }

void solve(int x, int y) {
for(int i = 1; i <= n; i++) {
a[i] = original[i]; }
memset(cnt, 0, sizeof cnt);
memset(aux, 0, sizeof aux);
l = r = 1;
aux[0] = inf; aux[1] = 1; cnt[a[1]] += 1;
int cur = 0;
for(auto i : g[x][y]) {
while(cur < (int) id.size() && id[cur] < i.id) {
if(l <= ind[cur] && ind[cur] <= r) del(ind[cur]);
a[ind[cur]] = val[cur];
if(l <= ind[cur] && ind[cur] <= r) add(ind[cur]);
++cur; }
while(i.l < l) add(--l);
while(r < i.r) add(++r);
while(l < i.l) del(l++);
while(i.r < r) del(r--);
ans[i.id] = get_ans(); } }

/// Query
int main(int argc, char const *argv[]) {
map <int, int> com; int idx = 0, q;
for(int i = 1; i <= n; i++) {
scanf("%d", &a[i]);
if(com.find(a[i]) == com.end()) {
com[a[i]] = ++idx; }
a[i] = com[a[i]];
original[i] = a[i]; }
memset(ans, -1, sizeof ans);
for(int i = 1; i <= q; i++) {
scanf("%d %d %d", &c, &x, &y);
if(c == 1) g[x / magic][y / magic].push_back(query(x,
y, i));
else {
if(com.find(y) == com.end()) {
com[y] = ++idx; }
y = com[y];
ind.push_back(x);
val.push_back(y);
id.push_back(i); } }

int last = n / magic;
for(int i = 0; i <= last; i++) {
for(int j = i; j <= last; j++) {
solve(i, j); } }
for(int i = 1; i <= q; i++) {
if(ans[i] != -1) {
printf("%d\n", ans[i]);} } }
///
```

Wavelet Tree

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

const int MAXN = (int)3e5 + 9;
const int MAXV = (int)1e9 + 9; //maximum value of
any element in array

//array values can be negative too, use appropriate
minimum and maximum value
struct wavelet_tree {
  int lo, hi;
  wavelet_tree *l, *r;
  int *b, *c, bsz, csz; // c holds the prefix sum of
elements

  wavelet_tree() {
    lo = 1;
    hi = 0;
    bsz = 0;
    csz = 0, l = NULL;
    r = NULL;
  }

  void init(int *from, int *to, int x, int y) {
    lo = x, hi = y;
    if(from >= to) return;
    int mid = (lo + hi) >> 1;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b = (int*)malloc((to - from + 2) * sizeof(int));
    bsz = 0;
    b[bsz++] = 0;
```

```
    c = (int*)malloc((to - from + 2) * sizeof(int));
    csz = 0;
    c[csz++] = 0;
    for(auto it = from; it != to; it++) {
      b[bsz] = (b[bsz - 1] + f(*it));
      c[csz] = (c[csz - 1] + (*it));
      bsz++;
      csz++;
    }
    if(hi == lo) return;
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree();
    l->init(from, pivot, lo, mid);
    r = new wavelet_tree();
    r->init(pivot, to, mid + 1, hi);
  }
  //kth smallest element in [l, r]
  //for array [1,2,1,3,5] 2nd smallest is 1 and 3rd smallest is 2
  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], lb = b[l - 1], 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 || 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 || k < lo || k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    int mid = (lo + hi) >> 1;
    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 sum(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->sum(lb + 1, rb, k) + this->r->sum(l - lb, r - rb, k);
  }
  ~wavelet_tree() {
    delete l;
    delete r;
  }
};
wavelet_tree t;
int a[MAXN];
int main() {
  int i, j, k, n, m, q, l, r;
  cin >> n;
  for(i = 1; i <= n; i++) cin >> a[i];
  t.init(a + 1, a + n + 1, -MAXV, MAXV);
  //beware! after the init() operation array a[] will not be same
  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.sum(l, r, k) << endl;
    }
  }
  return 0;
}
```

Sliding Window Maximum

```
vector<int> maxSlidingWindow(vector<int>& nums, int k) {
```

```
    int n = nums.size();
    deque<int> dq;
    vector<int> ans;
    for(int i = 0;i < n;i++){
        while(dq.size() > 0 && nums[dq.back()] <
nums[i]) dq.pop_back();
        dq.push_back(i);
        if(i >= k - 1){
            ans.push_back(dq.front());
            if(dq.front() < i - k + 2) dq.pop_front();
        }
    }
    for(int i = 0;i < ans.size();i++){
        ans[i] = nums[ans[i]];
    }
    return ans;
}
```

Number Theory

Linear Sieve + SPF

```
void linear_sieve() {
for(int i = 2; i < MAX; i++){
if (spf[i] == 0)
spf[i] = i, pr.push_back(i);
int sz = pr.size();
for (int j = 0; j < sz && i * pr[j] < MAX
&& pr[j] <= spf[i]; j++)
{ spf[i * pr[j]] = pr[j]; } } }
```

Euler Phi Sieve

$//P_1^{k1-1}(P_1-1).P_2^{k2-1}(P_2-1)....P_r^{kr-1}(P_{r-1})$

```
void computeTotient(int n) {
for(int i=1; i<=n; i++) phi[i] = i;
for(int j=2; j<=n; j++){
if(phi[j] == j){
phi[j] = j-1;
for(int i = 2*j; i<=n; i += j){
phi[i] = (phi[i]/j) * (j-1); }}}}
```

Extended GCD

```
ll extendedGCD(ll a,ll b,ll *x,ll *y) {
if(a == 0) {
*x = 0 , *y = 1;
return b; }
ll x1 , y1;
ll gcd = extendedGCD(b%a, a, &x1, &y1);
*x = y1 - (b/a)*x1;
*y = x1;
return gcd; }

ll modInverse(ll a,ll M) {
if(__gcd(a,M) > 1) return -1;
ll x , y;
ll gcd = extendedGCD(a,M,&x,&y);
return (x+M)%M; }
```

N! Problems

```
int trailingZeroes(int n) {    int c = 0 , f = 5;
while(f <= n)  {    c += n/f;    f *= 5; }    return c; }

int factDigitCnt(int n) {   if(n<=1)    return n;
double digits = 0;    for(int i=2;i<=n;i++){
digits += log10(i);}    return floor(digits)+1;    }

ll factDivisorsCnt(ll n) {   ll res = 1;
for(int i=0;primes[i]<=n;i++) { ll exp = 0;    ll p =
primes[i];    while(p <= n) {    exp += (n/p); p *=
primes[i]; }
res *= (exp+1); }    return res; }
```

NCR & NPR

```
ll Bigmod(int a,int b) {
if(b==0) return 1%MOD;
ll x=Bigmod(a,b/2);
x=(x*x)%MOD;
if(b%2==1) x=(x*a)%MOD;
return x; }

ll nCr(int x, int y) {
if(x<0 || y<0 || x<y) return 0;
return fact[x]*(inv[y]*inv[x - y] % MOD) % MOD; }


ll nPr(int x, int y) {
```

```
if(x<0 || y<0 || x<y)return 0;
return (fact[x] *inv[x - y] ) % MOD; }

void pre_cal() {
fact[0]=1;
for(int i=1; i<=MAX; i++)
fact[i]=fact[i-1]*1LL*i%MOD;
inv[MAX]=Bigmod(fact[MAX],MOD-2);
for(int i=MAX; i>0; i--){
inv[i-1]=i* 1LL*inv[i] % MOD;} }
```

nCr Modulo Any Mod

```
int power(long long n, long long k, const int mod) {
int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
while (k) {
if(k & 1) ans = (long long) ans * n % mod;
n = (long long) n * n % mod;
k >>= 1; }
return ans; }
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
if (b == 0) {
x = 1; y = 0;
return a; }
ll x1, y1;
ll d = extended_euclid(b, a % b, x1, y1);
x = y1;
y = x1 - y1 * (a / b);
return d; }
ll inverse(ll a, ll m) {
ll x, y;
ll g = extended_euclid(a, m, x, y);
if (g != 1) return -1;
return (x % m + m) % m; }

int factmod(ll n, int p, const int mod) {
vector<int> f(mod + 1);
f[0] = 1 % mod;
for(int i = 1; i <= mod; i++) {
if (i % p) f[i] = 1LL * f[i - 1] * i % mod;
else f[i] = f[i - 1]; }
int ans = 1 % mod;
while (n > 1) {
ans = 1LL * ans * f[n % mod] % mod;
ans = 1LL * ans * power(f[mod], n / mod, mod) %
mod;
n /= p; }
return ans; }
ll multiplicity(ll n, int p) {
ll ans = 0;
while (n) {
n /= p;
ans += n; }
return ans; }
int ncr(ll n, ll r, int p, int k) {
if (n < r or r < 0) return 0;
int mod = 1;
for (int i = 0; i < k; i++) {
mod *= p; }
ll t = multiplicity(n, p) - multiplicity(r, p) -
multiplicity(n - r, p);
if (t >= k) return 0;
int ans = 1LL * factmod(n, p, mod) *
inverse(factmod(r, p, mod), mod) % mod *
inverse(factmod(n - r, p, mod), mod) % mod;
ans = 1LL * ans * power(p, t, mod) % mod;
return ans; }

pair<ll, ll> CRT(ll a1, ll m1, ll a2, ll m2) {
ll p, q;
ll g = extended_euclid(m1, m2, p, q);
if (a1 % g != a2 % g) return make_pair(0, -1);
ll m = m1 / g * m2;
p = (p % m + m) % m;
q = (q % m + m) % m;
return make_pair((p * a2 % m * (m1 / g) % m + q * a1
% m * (m2 / g) % m) %  m, m); }

int spf[N];
vector<int> primes;
void sieve() {
for(int i = 2; i < N; i++) {
if (spf[i] == 0) spf[i] = i, primes.push_back(i);
int sz = primes.size();
for (int j = 0; j < sz && i * primes[j] < N &&
primes[j] <= spf[i]; j++) {
spf[i * primes[j]] = primes[j]; } } }
```

```
int ncr(ll n, ll r, int m) {
if (n < r or r < 0) return 0;
pair<ll, ll> ans({0, 1});
while (m > 1) {
int p = spf[m], k = 0, cur = 1;
while (m % p == 0) {
m /= p; cur *= p;
++k; }
ans = CRT(ans.first, ans.second, ncr(n, r, p, k), cur); }
return ans.first; }
```

Pollard Rho

```
const ll mod = 1e9+7, phi = 1e9+6;
vector<ll> factor;
ll mult(ll a, ll b, ll mod) {
ll result = 0;
while (b) {
if (b & 1) result = (result + a) % mod;
a = (a + a) % mod;
b >>= 1;    }
return result; }

ll bigmod(ll a, ll b) {
if(b == 0) return 1;
ll x = bigmod(a, b/2);
x = mult(x, x, mod);
if(b & 1) x = mult(x, a, mod);
return x;   }

ll f(ll p, ll a, ll k) {
p %= mod;
ll b = (k * (a + 1)) % phi;
ll res = (bigmod(p, b) - 1) % mod;
if(res < 0) res += mod;
ll q = bigmod(p, k % phi);
q = (q - 1) % mod;
if(q < 0) q += mod;
ll inv = bigmod(q, mod-2);
res = (res * inv) % mod;
return res; }

ll rho(ll n){
if(n%2==0)return 2;
ll x = rand()%n+1;
ll c = rand()%n+1;
ll y = x;
ll g = 1;
while(g==1){
x = (mult(x, x, n) + c)%n;
y = (mult(y, y, n) + c)%n;
y = (mult(y, y, n) + c)%n;
ll h = (x > y)? x-y : y-x;
g = __gcd(h, n);    }
return g;   }

ll binpower(ll base, ll e, ll mod) {
ll result = 1;
base %= mod;
while (e) {
if (e & 1)
result = mult(result, base, mod);
base = mult(base, base, mod);
e >>= 1;    }
return result;  }

bool check_composite(ll n, ll a, ll d, ll s) {
ll x = binpower(a, d, n);
if (x == 1 || x == n - 1)
return false;
for (ll r = 1; r < s; r++) {
x = mult(x, x, n);
if (x == n - 1)
return false;   }
return true;    }

bool MillerRabin(ll n) {
if (n < 2)
return false;
int r = 0;
ll d = n - 1;
while ((d & 1) == 0) {
d >>= 1;
r++;    }
for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
if (n == a) return true;
if (check_composite(n, a, d, r))
return false;   }
return true;    }
void factorize(ll n) {
if(n == 1) return;
```

```
if(MillerRabin(n)) {
factor.pb(n);
return; }
ll d = rho(n);
factorize(d);
factorize(n/d); }
```

Phi for a large number:

```
int phi(int n) {
double res = n;
for(int i=2;i*i<=n;i++) {
if(n%i == 0) {
while(n%i == 0)
n /= i;
res *= (1.0 - (1.0/i)) } }
if(n > 1)
res *= (1.0 - (1.0/n));
return (int)(res); }
```

SOD of n^m:

// $(P_1^{e1+1} - 1 / P_1 - 1).(P_2^{e2+1} - 1 / P_2 - 1)..... (P_r^{er+1} - 1 / P_r - 1)$

```
ll primeFact(ll n,int m) {
ll sum = 1;
for(int i=0; i<primes.size() && primes[i]<=n; i++) {
ll cnt = 0, p = primes[i];
if(n%p == 0) {
while(n%p == 0)
cnt++ , n /= p;
cnt = cnt*m+1;
ll calc = (bigMod(p,cnt,MOD)+MOD-1)%MOD;
calc *= bigMod(p-1,MOD-2,MOD);
calc %= MOD;
sum = (sum*calc)%MOD; } }
if(n > 1) {
ll calc = (bigMod(n,1+m,MOD)+MOD-1)%MOD;
calc *= bigMod( n-1, MOD-2, MOD);
calc %= MOD;
sum = (sum*calc)%MOD; }
return sum; }
```

Bits Related

```
int SET(int cur, int pos) { return cur | (1LL << pos); }
bool check(ll n,ll pos) { return n & (1ll<<pos); }
int CLEAR(int n,int pos) { return n & ~(1<<pos); }
```

Check if a subset sum exits:

```
bitset<MAX>bs;
bool check(int sum) { bs.reset(); bs[0]=1;
for(int i=1;i<=n;i++) bs |= bs << arr[i]; return bs[sum];
}
```

Xor in a range:

```
void compute(int n) {
for(int i=1;i<=n;i++) xor[i]=xor[i-1]^arr[i]; }
int query(int l,int r) { return xor[r]^xor[l-1]; }
```

Graphs

Floyed Warshall

```
for(int k=1; k<=nodes; k++){
for(int i=1; i<=nodes; i++) {
for(int j=1; j<=nodes; j++){
graph[i][j]=min(graph[i][j], graph[i][k]+graph[k][j]);
}}}
```

graph[p][q] = distance p → q

Krushkal's algorithm:

```
bool cmp(edge a, edge b){
   return a.w<b.w;}

int _find(ll src){
   if(parent[src]==src)
      return src;
   return parent[src]=_find(parent[src]);}

void initPar(ll src){
   for(ll i=0 ; i<src ; i++){
      parent[i]=i; }}

void kruskals_Algorithm(ll n){
   srt(graph,cmp);
   initPar(n);
```

```
for(ll i=0; i<graph.size() ; i++){
    ll up=_find(graph[i].u);
    ll vp=_find(graph[i].v);

    if(up!=vp) {
        if(cnt==n-1)
            break;
        output.push_back(graph[i]);
        mstValue+=graph[i].w;
        parent[up]=vp;
        cnt++;} }}
```

Lowest Common Ancestor

```
const int N = 1e5 + 10;
int tin[N], tout[N];
int up[N][32];
vector<int> adj[N];
int n, lg, timer;
void dfs(int src, int par){
    tin[src] = ++timer;
    up[src][0] = par;
    for(int i = 1;i <= lg;i++){
        up[src][i] = up[up[src][i - 1]][i - 1];
    }
    for(auto child : adj[src]){
        if(child != par)
            dfs(child, src);
    }
    tout[src] = ++timer;
}
bool is_ancestor(int u, int v){
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v){
    if(is_ancestor(u, v)) return u;
    if(is_ancestor(v, u)) return v;
    for(int i = lg;i >= 0;i--){
        if(!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
void pre_process(int root){
    timer = 0;
    lg = ceil(log2(n));
    dfs(root, root);
}
// call pre_process(1)
```

Euler Tour

```
void dfs_euler(int src,int par){
euler_path.push_back(src);
int f=0;
for(auto i:graph[src]){
if(i==par)
continue;
f=1;
dfs_euler(i,src);}
if(f)
euler_path.push_back(src);}

void init_euler_path(){
dfs_euler(1,0);
int idx=1;
for(auto i:euler_path){
if(st[i]==0)
st[i]=idx;
en[i]=idx;
idx++; } }
```

BPM

```
vector<int>graph[1000005];
bool visit[1000005];
int connection[1000005];
bool BPM(int node) {
int sz=graph[node].size();
for(int i=0; i<sz; i++){
int child=graph[node][i];
if(visit[child]==0)  {   visit[child]=1;
if(connection[child]<0 || BPM(connection[child])) {
connection[child]=node;  return true; } } }
return false;}
int maxBPM(int n)
{memset(connection,-1,sizeof(connection));
 int res=0;
for(int i=0; i<n; i++) {
memset(visit,0,sizeof(visit));
if(BPM(i))
```

```
res++;  }
return res; }
```

Articulation point

```
void dfs(int v, int p = -1) {
visited[v] = true;
tin[v] = low[v] = timer++;
int children=0;
for (int to : adj[v]) {
if (to == p) continue;
if (visited[to]) {
low[v] = min(low[v], tin[to]); }
else {  dfs(to, v);
low[v] = min(low[v], low[to]);
if (low[to] >= tin[v] && p!=-1)
IS_CUTPOINT(v);
++children; }  }
if(p == -1 && children > 1)
IS_CUTPOINT(v);//v is the point print it }

void find_cutpoints() {
timer = 0;
visited.assign(n, false);
tin.assign(n, -1);
low.assign(n, -1);
for (int i = 0; i < n; ++i) {
 if (!visited[i])
 dfs (i); } }
```

Bridge

```
void dfs(int v, int p = -1) {
visited[v] = true;
tin[v] = low[v] = timer++;
for (int to : adj[v]) {
if (to == p) continue;
if (visited[to]) {
low[v] = min(low[v], tin[to]); }
else {
dfs(to, v);
low[v] = min(low[v], low[to]);
if (low[to] > tin[v])
IS_BRIDGE(v, to);//u v is the edge } } }

void find_bridges() {
timer = 0;
visited.assign(n, false);
tin.assign(n, -1);
low.assign(n, -1);
for (int i = 0; i < n; ++i) {
if (!visited[i])
dfs(i); } }
```

SCC

```
vector < int > graph[MAX] , reverseGraph[MAX] ,
components[MAX];
bool vis[MAX];
int compCount;
stack<int>nodes;
void DFS(int src) {
vis[src] = 1;
for(auto i : graph[src]) {
if(!vis[i])
DFS(i); }
nodes.push(src); }
void DFS2(int src) {
vis[src] = 1;
for(auto i : reverseGraph[src]) {
if(!vis[i])
DFS2(i); }
components[compCount].push_back(src); }
void init() {
compCount = 1;
for(int i=1;i<MAX;i++)
graph[i].clear() , reverseGraph[i].clear() ,
components[i].clear(); }
void addEdge(int u,int v) {
graph[u].push_back(v);
reverseGraph[v].push_back(u); }
void kosaraju_SCC(int n) {
memset(vis,0,sizeof vis);
for(int i=1;i<=n;i++) {
if(!vis[i])
DFS(i); }
memset(vis,0,sizeof vis);
while(nodes.size()) {
int top = nodes.top();
nodes.pop();
if(!vis[top]) {
DFS2(top);
```

```
compCount++; } } }

void print_SCCs() {
for(int i=1;i<compCount;i++) {
cout << "Component " << i << ":\n";
for(auto j : components[i])
cout << j << " -> ";  cout << endl; } }
-> addEdge(u,v); -> print_SCCs();
-> kosaraju_SCC(n);
```

HLD

```
const int D = 19; const int S = (1<<D);
vector<int> adj[N];
int sz[N], p[N][D], dep[N], seg_t[S], v[N], id[N],
tp[N], cnt = 1, n, q;
void update(int idx, int val, int pos=1, int l=1, int
r=n){
if(l == r){
seg_t[pos] = val;return;}
int m = (l+r)/2;
if(idx <= m) update(idx, val, pos*2, l, m);
else update(idx,  val, pos*2+1, m+1, r);
seg_t[pos] = max(seg_t[pos*2], seg_t[pos*2+1]); }

int query(int lo, int hi, int pos=1, int l=1, int r=n){
if(lo > r || hi < l) return 0;
if(lo <= l && r <= hi) return seg_t[pos];
int m = (l+r)/2;
return max(query(lo, hi, pos*2, l, m),query(lo, hi,
pos*2+1, m+1, r)); }

int dfs_sz(int src, int par){
sz[src] = 1;
for(int child : adj[src]){
if(child == par) continue;
dep[child] = dep[src] + 1;
p[child][0] = src;
sz[src] += dfs_sz(child, src); }
return sz[src]; }

void init_lca(){
for(int d=1; d<18; d++)
for(int i=1; i<=n; i++)
p[i][d] = p[p[i][d-1]][d-1]; }

void dfs_hld(int cur, int par, int top){
id[cur] = cnt++; tp[cur] = top;
update(id[cur], v[cur]);
int h_chi = -1, h_sz = -1;
for(int chi : adj[cur]) {
if(chi == par) continue;
if(sz[chi] > h_sz) {
h_sz = sz[chi];
h_chi = chi; } }
if(h_chi == -1) return;
dfs_hld(h_chi, cur, top);
for(int chi : adj[cur]) {
if(chi == par || chi == h_chi) continue;
dfs_hld(chi, cur, chi); } }

int lca_query(int a, int b) {
if(dep[a] < dep[b]) swap(a, b);
for(int d=D-1; d>=0; d--) {
if(dep[a] - (1<<d) >= dep[b]) {
a = p[a][d]; } }
for(int d=D-1; d>=0; d--) {
if(p[a][d] != p[b][d]){
a = p[a][d]; b = p[b][d]; } }
if(a != b) {
a = p[a][0]; b = p[b][0]; }
return a; }
int path(int chi, int par) {
int ret = 0;
while(chi != par) {
if(tp[chi] == chi) {
ret = max(ret, v[chi]); chi = p[chi][0]; }
else if(dep[tp[chi]] > dep[par]) {
ret = max(ret, query(id[tp[chi]], id[chi]));
chi = p[tp[chi]][0]; }
else {
ret = max(ret, query(id[par]+1, id[chi]));
break; } }
return ret; }

int main() {
dfs_sz(1, 1); init_lca();
memset(seg_t, 0, sizeof seg_t);
dfs_hld(1, 1, 1);
```

```
while(q--) {
scanf("%d", &t);
if(t == 1) {
scanf("%d%d", &s, &x);
v[s] = x;
update(id[s], v[s]); }
else {
scanf("%d%d", &a, &b);
int c = lca_query(a, b);
int res = max(max(path(a,c), path(b,c)), v[c]);
printf("%d ", res); } } }
```

Centroid Decomposition

```
vector<int> graph[200001];
int subtree[200001], mx_depth; ll ans = 0,
bit[200001];
bool processed[200001];

int get_subtree_sizes(int node, int parent = 0) {
subtree[node] = 1;
for(int i : graph[node])
if(!processed[i] && i != parent)
subtree[node] += get_subtree_sizes(i, node);
return subtree[node]; }

int get_centroid(int desired, int node, int parent =
0) {
for(int i : graph[node])
if(!processed[i] && i != parent && subtree[i] >=
desired)
return get_centroid(desired, i, node);
return node; }

void update(int pos, ll val,int n){
for(pos++; pos <= n; pos += pos & -pos) bit[pos] +=
val; }

ll query(int l, int r) {
ll ans = 0;
for (r++; r; r -= r & -r) ans += bit[r];
for (; l; l -= l & -l) ans -= bit[l];
return ans; }

void get_cnt(int a,int b,int n,int node, int parent,
bool filling, int depth = 1) {
if (depth > b) return;
mx_depth = max(mx_depth, depth);
if (filling) update(depth, 1,n);
else ans += query(max(0, a - depth), b - depth);
for (int i : graph[node]) if (!processed[i] && i !=
parent)
get_cnt(a,b,n,i, node, filling, depth + 1); }

void centroid_decomp(int a,int b,int n,int node =
1){
int centroid = get_centroid(get_subtree_sizes(node)
>> 1, node);
processed[centroid] = true;
mx_depth = 0;
for (int i : graph[centroid]) if (!processed[i]) {
get_cnt(a,b,n,i, centroid, false);
get_cnt(a,b,n,i, centroid, true); }
for (int i = 1; i <= mx_depth; i++) update(i, -query(i,
i),n);
for (int i : graph[centroid]) if (!processed[i])
centroid_decomp(a,b,n,i); }
int main() {
int n,a,b;
cin >> n >> a; b=a;
update(0, 1,n);
centroid_decomp(a,b,n);
cout << ans;
return 0; }
```

Cycle

```
bool dfs(int v) {
color[v] = 1;
for (int u : adj[v]) {
if (color[u] == 0) {
parent[u] = v;
if (dfs(u))
return true; }
else if (color[u] == 1) {
cycle_end = v;
cycle_start = u;
return true; } }
color[v] = 2;
return false;}
```

```
void find_cycle() {
color.assign(n, 0);
parent.assign(n, -1);
cycle_start = -1;
for (int v = 0; v < n; v++) {
if (color[v] == 0 && dfs(v))
break; }
if (cycle_start == -1) {
cout << "Acyclic" << endl; }
else { vector<int> cycle;
cycle.push_back(cycle_start);
for (int v = cycle_end; v != cycle_start; v = parent[v])
cycle.push_back(v);
cycle.push_back(cycle_start);
reverse(cycle.begin(), cycle.end());
cout << "Cycle found: ";
for (int v : cycle)
cout << v << " ";  cout << endl; }}
```

Dijkstra K shortest

```
const ll mx =1e6;
ll n,m,k;
map<pair<ll,ll>,ll>mp;
void dijkstra(ll src, vector<pair<ll,ll>>adj[]){
   vector<vector<ll>> dist(n+10,
vector<ll>(k,LONG_LONG_MAX));
   priority_queue< pair<ll,ll>, vector<pair<ll,ll>>,
greater<pair<ll,ll>> > pq;
   pq.push({0,src});
   while(pq.size()>0){
      pair<ll, ll> cur=pq.top();
      pq.pop();
      ll u=cur.second;
      ll d=cur.first;

      if(d>dist[u][k-1]){
         continue; }
      for(ll i=0; i<adj[u].size(); i++ ){
         ll v =adj[u][i].first;
         ll d1=adj[u][i].second;
         if(d+d1<dist[v][k-1] ){
            dist[v][k-1]=d+d1;
            pq.push({dist[v][k-1],v});
            srt(dist[v]); }  }}
   for(ll i=0; i<k; i++) {
      cout<<dist[n][i]<<" ";}
   cout<<endl;}
```

Topological Sort

```
vector<int> adj[N];
vector<int> longestDistance(N, INF);
void topo_Sort(int node, vector<bool>& visited,
stack<int>& st) {
   visited[node] = true;
   for (int child : adj[node]) {
      if (!visited[child]) {
         dfs(child, visited, st);
      }
   }
   st.push(node);
}
void longestPathDAG(int src, int n) {
   vector<bool> visited(n, false);
   stack<int> st;
   for (int i = 1; i < n; ++i) {
      if (!visited[i]) {
         topo_Sort(i, visited, st);
      }
   }
   longestDistance[src] = 0;
   while (!st.empty()) {
      int node = st.top();
      st.pop();
      if (longestDistance[node] != INF) {
         for (int child : adj[node]) {
    longestDistance[child] =
max(longestDistance[child], longestDistance[node] +
1);}}}}
```

String

Hashing

```
const ll MOD = 999999999999999989,
MOD1=1000001137;
ll base = 7919,base1= 2551, base2 = 6091;

void pre_power(){
```

```
pw[0] = 1;
for(int i = 1; i < 300015; i++)
pw[i] = (pw[i - 1] * base) % MOD; }

ll get_hashval(string str){
int len=str.length();
ll hash_val=0;
for(int i = 0; i < len; i++){
hash_val=((hash_val*base)+str[i])%MOD;
HASH[i+1]=hash_val; }
return hash_val; }

ll SubstringHash(int l, int r){
return (HASH[r]-(HASH[l-1]*pw[r-l+1]) %
MOD + MOD) % MOD; }
```

Z-Algorithm:

```
string S;
int z[MAX];
void zFunction() {
int left , right;
left = right = z[0] = 0;
for(int i=1;i<S.size();i++) {
if(i <= right)
z[i] = min(z[i-left],right-i+1);
while(i+z[i] < S.size() && S[i+z[i]] == S[z[i]]) z[i]++;
if(i+z[i]-1 > right)
left = i , right = i+z[i]-1; } }

bool isSubstr(string t,string p) {
S = p + "#" + t;
zFunction();
for(int i=p.size()+1;i<S.size();i++) {
if(z[i] == p.size())
return true; }
return false; }

int countSubstr(string t,string p) {
S = p + "#" + t;
memset(z,0,sizeof z);
zFunction();
int cnt = 0;
for(int i=p.size()+1;i<S.size();i++) {
if(z[i] == p.size())
cnt++; }
return cnt; }
```

KMP:

```
vector<int> prefix_function(string s) {
   int n = (int)s.length();
   vector<int> pi(n);
   for (int i = 1; i < n; i++) {
      int j = pi[i-1];
      while (j > 0 && s[i] != s[j])
         j = pi[j-1];
      if (s[i] == s[j])
         j++;
      pi[i] = j;}
   return pi;}

void solve(int cs){
   string s,s1;
   cin>>s;

   s1=s;
   reverse(s1.begin(),s1.end());
   s = s + "#" + s1;
   vector<int> v = prefix_function(s);
   int mx=0;
   //cout<<s<<endl;
   for(int i=(s.length()/2); i<v.size(); i++){
      mx=max(mx,v[i]);
      //cout<<i<<" "<<v[i]<<endl; }

   for(int i=mx-1; i>=0; i--){
      cout<<s[i]; }
   cout<<endl;}
```

Suffix Array

```
const int mxN = 1e+5,K = 20;
int sa[mxN], pos[mxN],
tmp[mxN],st[mxN][K+1],lcp[mxN],pre[mxN], gap,
N;
bool comp(int x, int y) {
if(pos[x] != pos[y])return pos[x] < pos[y];
x += gap;
```

```
y += gap;
return (x < N && y < N) ? pos[x] < pos[y] : x > y; }
void suffix(string &s) {
for (int i = 0; i < N; i++) sa[i] = i, pos[i] = s[i];
for (gap = 1;; gap <<= 1) {
sort(sa, sa+N, comp);
for (int i = 0; i < N-1; i++)
tmp[i+1] = tmp[i] + comp(sa[i], sa[i+1]);
for (int i = 0; i < N; i++)
pos[sa[i]] = tmp[i];
if (tmp[N - 1] == N - 1) break; } }

int check(int m, string &s, string &x) {
int f = -1, k = x.size(), j = sa[m];
if (N - j >= k)f = 0;
for (int i = 0; i < min(N - j, k); i++) {
if (s[j+i] < x[i]) return -1;
if (s[j+i] > x[i]) return 1; }
return f; }

int patternExistsLB(int l,int r, string &s, string &x)
{
int ans=-1;
while (l <= r) {
int m = l + (r-l)/2;
int v = check(m, s, x);
if (v == 0){
ans = m; r = m - 1; }
else if (v == 1) r = m - 1;
else l = m + 1; }
return ans; }

int patternExistsRB(int l,int r, string &s, string &x)
{
int ans=-1;
while (l <= r) {
int m = l + (r-l)/2;
int v = check(m,s, x);
if (v == 0) {
ans = m; l = m + 1;}
else if (v == -1) l = m + 1;
else r = m - 1; }
return ans; }

int patternCount(int l,int r, string s,string &x) {
int L=patternExistsLB(l,r,s,x);
if(L==-1)return 0;
int R=patternExistsRB(L,r,s,x);
return (R-L)+1; }

int query(int l, int r) {
int j = log2(r-l+1);
return min(st[l][j], st[r-(1<<j)+1][j]) + 1; }

void buildIdx() {
for (int i = 0; i < N; i++)
st[i][0] = sa[i];

for (int j = 1; j <= K; j++) {
for (int i = 0; i + (1<<j) <= N; i++) {
st[i][j] = min(st[i][j-1], st[i + (1<<j-1)][j-1]); } } }

int findFirstIdx(int l,int r, string s,string &x) {
int L=patternExistsLB(l,r,s,x);
if(L==-1)return -1;
int R=patternExistsRB(L,r,s,x);
return query(L,R); }

void build_lcp(string &s){
for (int i = 0, k = 0; i < N; i++) if (pos[i] != N-1) {
int j = sa[pos[i] + 1];
while (s[i + k] == s[j + k]) k++;
lcp[pos[i]] = k;
if (k) k--; } }

long long int getUniqueSubCnt() {
long long int sm = accumulate(lcp, lcp+N, 0LL);
long long int tot = ((long long)1*(N)*(N+1))/2;
tot-=sm;
return tot; }
void printEveryUnqiueSubStirngCnt() {
int prev = 0;
for (int i = 0; i < N; i++) {
pre[prev + 1]++;
pre[N - sa[i] + 1]--;
prev = lcp[i]; }
for (int i = 1; i <= N; i++) {
cout << pre[i] << ' ' ;
pre[i+1] += pre[i];} }
```

```
string kthDistinctSubstring(string s,long long k){
long long prev = 0;
long long cur = 0;
for (int i = 0; i < N; i++){
if (cur + (N-sa[i]) - prev >= k) {
long long j = prev;
string ans = s.substr(sa[i], j);
while (cur < k){
ans += s[sa[i]+j];
cur++; j++; }
return ans; }
cur += (N-sa[i]) - prev;
prev = lcp[i]; }
return ""; }
```

Palindromic Query

```
char s[MAX+500], ori[MAX+50];
vector<int>adj[MAX];
ull pw[MAX];
int n,q, cnt=0;

struct BIT{
ull t[MAX];
#define lowb (i&-i)
void modify(int i,ull k) {
while(i<=n)t[i]+=k,i+=lowb; }

ull Qsum(int i) {
ull res=0;
while(i)res+=t[i],i-=lowb;
return res; }
} t1,t2;

void build(){
for(int i=0; i<n; i++)ori[i]=s[i];
pw[0]=1;
for(int i=1; i<=n; ++i)pw[i]=pw[i-1]*131;
for(int i=1; i<=n; ++i) {
t1.modify(i,(s[i-1]-'a'+1)*pw[n-i+1]);
t2.modify(i,(s[i-1]-'a'+1)*pw[i]); } }

void update(char c, int pos) {
t1.modify(pos,(c-s[pos-1])*pw[n-pos+1]);
t2.modify(pos,(c-s[pos-1])*pw[pos]); }

bool isPalindrome(int l,int r) {
ull f1=t1.Qsum(r)-t1.Qsum(l-1);
ull f2=t2.Qsum(r)-t2.Qsum(l-1);
if(n-r-l+1>=0) return (f1==f2*pw[n-r-l+1]);
else return (f1*pw[r+l-1-n]==f2); }

void reset() {
cnt=0;
for(int i=1; i<=n; i++)
adj[i].clear();
for(int i=0; i<=n; i++)
t1.t[i]=0,t2.t[i]=0; }

// Query
isPlaindome(l,r)
update(c,idx);
str[idx-1]=c;
///
```

K-th LexigrophaicallySmallestSubString

```
const int N = 1e5 + 5;
struct state {
int len, link, next[26];
ll cnt = 0, cnt2 = -1;
};
string s; state st[2 * N];
int n, k, sz, last;
vector<pair<int, int>> order;
void st_init() {
st[0].len = 0;
st[0].link = -1;
sz++; last = 0; }

void dfs(int u) {
if (st[u].cnt2 != -1) return;
st[u].cnt2 = st[u].cnt;
for (int i = 0; i < 26; ++i) {
if(st[u].next[i]) {
dfs(st[u].next[i]);
st[u].cnt += st[st[u].next[i]].cnt; } } }

void st_extend(int c) {
int cur = sz++;
```

```
st[cur].len = st[last].len + 1;
st[cur].cnt = 1;
order.emplace_back(st[cur].len, cur);
int p = last;
while (p != -1 && !st[p].next[c]) {
st[p].next[c] = cur;
p = st[p].link; }
if (p == -1) st[cur].link = 0;
else {
int q = st[p].next[c];
if (st[p].len + 1 == st[q].len) {
st[cur].link = q;}
else {
int clone = sz++;
st[clone].len = st[p].len + 1;
st[clone].link = st[q].link;
memcpy(st[clone].next, st[q].next, sizeof(st[q].next));
order.emplace_back(st[clone].len, clone);
while (p != -1 && st[p].next[c] == q) {
st[p].next[c] = clone;
p = st[p].link; }
st[cur].link = st[q].link = clone; } }
last = cur; }

int main() {
cin >> s >> k;
n = s.length(); k += n;
st_init();
for (int i = 0; i < n; ++i) st_extend(s[i] - 'a');
sort(order.begin(), order.end());
reverse(order.begin(), order.end());
for (auto &p: order) {
st[st[p.second].link].cnt += st[p.second].cnt; }

dfs(0);
if (st[0].cnt < k) {
cout << "No such line.";
return 0; }

int cur = 0;
while (k > st[cur].cnt2) {
k -= st[cur].cnt2;
for (int i = 0; i < 26; ++i) {
if (st[cur].next[i]) {
int j = st[cur].next[i];
if (st[j].cnt < k) k -= st[j].cnt;
else {
cout << (char)(i + 'a');
cur = j;
break; } } } }
```

Dynamic Programming

LIS

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;
    for (int i = 0; i < n; i++) {
        int l = upper_bound(d.begin(), d.end(), a[i]) -
d.begin();
        if (d[l-1] < a[i] && a[i] < d[l])
            d[l] = a[i];
    }
    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;}
```

LNDS

```
int lnds[MAX];
int LNDS(int n) {
lnds[1]=arr[1]; //1 base index
int len = 1 ;
for(int i = 2; i<=n;i++)  {
if(arr[i]>=lnds[len])
lnds [++ len] = arr [i];
else{
int j=upper_bound(lnds+1,lnds+len+ 1,arr[i])-lnds;
lnds [j] = arr [i];  }   }
return len; }
```

SOS DP

```
for(int i = 0; i < (1 << N); ++i) {
F[i] = A[i];  }
for(int i = 0; i < N; ++i) {
for(int mask=0;mask<(1<<N); ++mask) {
if(mask & (1 << i)) {
F[mask] += F[mask ^ (1 << i)]; } }}
```

Meet in the middle

```
void calcsubarray(ll a[], ll x[],
int n, int c) {
for (int i=0; i<(1<<n); i++) {
ll s = 0;
for (int j=0; j<n; j++)
if (i & (1<<j))
s += a[j+c];
x[i] = s;  }  }
ll SSsum(ll a[],int n,ll S) {
calcsubarray(a, X, n/2, 0);
calcsubarray(a, Y, n-n/2, n/2);
int size_X = 1<<(n/2);
int size_Y = 1<<(n-n/2);
sort(Y, Y+size_Y);
ll mx = 0;
for (int i=0; i<size_X; i++) {
if (X[i] <= S) {
int p =lower_bound(Y,Y+size_Y,
S-X[i])-Y;
if(p==size_Y || Y[p]!=(S-X[i]))
        p--;
 if ((Y[p]+X[i]) > mx)
        mx = Y[p]+X[i];  }  }
        return mx; }
```

Digit Dp

```
vector<int>num;
int sz,k,n,m;
int dp[10][2][100][100];
int digitdp(int pos,int issmall,int sum,int val) {
if(pos==sz)  {
if(!sum && !val)
return 1;
return 0;  }
if(dp[pos][issmall][sum][val]!=-1)
return dp[pos][issmall][sum][val];
int lim;
if(issmall==0)
lim=num[pos];
else
lim=9;
int ans=0;
for(int digit=0; digit<=lim; digit++) {
int cur_issmall=issmall;
if(issmall==0 && digit<lim)
cur_issmall=1;
int cur_sum=(sum+digit)%k;
int cur_val=((val*10)+digit)%k;
 ans+=digitdp(pos+1,cur_issmall,cur_sum,cur_val); }
 return dp[pos][issmall][sum][val]=ans;}

int solve(int n) {
num.clear();
while(n>0)  {
num.push_back(n%10);
n/=10;  }
sz=num.size();
reverse(num.begin(),num.end());
memset(dp,-1,sizeof(dp));
return digitdp(0,0,0,0); }
```

Kadane

```
int kadane(int sz) {
int max_so_far=arr[0];
int max_ending_here=0;
int st=0,en=0,point=0;
for(int i=0; i<sz; i++) {
max_ending_here=max_ending_here+arr[i];
if(max_so_far<max_ending_here) {
max_so_far=max_ending_here;
st=point,en=i;  }
if(max_ending_here<0) {
max_ending_here=0;
point=i+1;  }  }
return max_so_far; }
```

Bit Mask Dp

```
ll arr[20][20];
ll n;
```

```
ll dp[20][(1<<16)+10];

ll bitmaskdp(ll i, ll mask){
   if(mask==((1<<n)-1))return 0;
   if(~dp[i][mask]) return dp[i][mask];

   ll ans=0;

   for(int j=0; j<n; j++){
      if((mask&(1<<j))==0){

ans=max(ans,arr[i][j]+bitmaskdp(i+1,mask|(1<<j)));
      } }
return dp[i][mask]=ans;}
```

GEOMETRY

Triangle:

- To form: a+b>c, b+c>a, c+a>b
- Check if 3 points form triangle: |(x2-x1)(y3-y1)-(y2-y1)(x3-x1)| >0

Perimeter: p = a+b+c
Area: 1) (a*b)/2
2) (abSinC)/2
3) sqrt(s(s-a)(s-b)(s-c)); ///**s=(p/2)**
4) (sqrt(3)/4)*a*a; ///equi triangle
5) (b*sqrt(4*a*a-b*b))/4; ///isosceles
Pythagoras: a*a+b*b = c*c

SineRule: a/SinA=b/SinB=c/SinC
CosineRule: CosA = (b*b+c*c-a*a)/2bc
Centre: x=(x1+x2+x3)/3, y=(y1+y2+y3)/3
Median: AD=sqrt((2*b*b+2*c*c-a*a)/4)
Centroid: AG=sqrt(2*b*b-2*c*c-a*a)/3
Inradius: A/s
Circumradius: a/(2*sinA)
r=abc/sq((a+b+c)(a+b-c)(a+c-b)(b+c-a))

Circle:

Distance: sqrt((x2-x1)^2 + (y2-y1)^2)
Check if 3 points are in same line:
x1*(y2-y3)-x2(y1-y3)+x3(y1-y2) = 0
Find a circle that covers 2 given:
x3 = (x1+x2)/2 , y3 = (y1+y2)/2
r = dist(x1,y1,x2,y2)
Lattice Points:1+gcd(|x1-x2|,|y1-y2|)
Slope formed by 2 points:(y2-y1)/(x2-x1)
Area of sector of circle: ½ r^2*θ
Arc Length: r*θ

Parallelogram:

Given 3 points find 4th point:
Dx = Ax + (Cx-Bx)
Dy = Ay + (Cy-By)
Area: |½((Ax*By+Bx*Cy+Cx*Dy+Dx*Ay)
-(Ay*Bx + By*Cx + Cx*Dx + Dy*Ax))|

Trapezium:

Area:(a+b)/(a-b) * sqrt((s-a)(s-b) (s-b-c)(s-b-d))
-> s = (a+b+c+d)/2
-> a = long parallel side
-> b = short parallel side
-> c,d = non-parallel side
Area:h*((b1+b2)/2)
H:sq(b^2-(b^2-d^2+(a-c)^2)/2(a-c))^2)

Right Circular Cone:

Volume:(pie*h/3)*(R^2+R*r+r^2)
Lateral surface Area:pie(r+R)*S
Area of the base:pie*r^2
Lateral area:pie*r*L
Total Surface A:pie*r^2+pie*r*s
Volume:⅓*pie*r^2*h
s=sq(r^2+h^2)

Polygon:

The sum of the interior angles: (2n – 4) × 90°
*Area of the largest square inside
a pentagon->
s*(sin(108)/(sin(18)+sin(36)))
Area:(s^2*n)/4*tan(180/4)
Area:(r^2*n*sin(360/n))/2
Area:Apo^2*n*tan(180/n)
Area:¼*sq(5*(5+2sq(5)))*s^2
Area:(Apo*s)/2

Area:pr/2
Area:½*n*sin(360/n)*s^2
Area:n*apo^2*tan(180/n)
Area:(n*r*sin(2*(180/n)))/2
Area:(¼*n*s^2)/tan(180/n)
Perimeter:5*s
Diagonal:(s*(1+sq(5)))/2
Height:(s*sq(5+2sq(5)))/2
an:2*R*sin(180/n) ..here:an=side of regular inscribed polygon,R=radius of circumscribed circle.
Sum of interior angles of a Convex polygon:180(n-2)
Exterior taken one at each vertex:360
measurement of Exterior Ang:360/n
Measure Interior An:((n-2)*180)/n
No. Of Dia:(n*(n-3))/2
No. Of Tri:N-2
Side:2*R*sin(180/n)
Apo:R*cos(180/n)
Side:2*apo*tan(180/n)
Area of smallest tri:½*apo*(s/2) = ½*Apo^2*tan(180/n)
Intersection points of diagonals of n(odd) sided regular polygon = nC4

Truncated Cone:

z=(H*r2^2)(r1^2-r2^2)
R=(½*r1^2(z+h))/(H+z)
Volume:⅓*pie*h*(R^2+(R*r2)+r2^2)
Volume of a cylinder: pi*r*r*h
Volume of a triangular prism: .5*b*h*H

Combinatorics:

Summation of squares of n natural numbers: (n*(n+1)*(2n+1))/6
(n,r): n! / (r! * (n-r)!)
C(n,r): (n*(n-1)*..*(n-r+1)) / r!
P(n,k): n! / (n-k)!
-> nCk = nCn-k
-> Ways to go from (0,0) to (r,c): (r+c)Cr or (r+c)Cc
-> Ways to go from (0,0,0) to (x,y,z): (x+y+z)Cx * (y+z)Cy
-> a1+a2+.+an = k , ai>=0: C(k+n-1,n-1)
-> Catalan Numbers: C(n) = (2n)! / ((n+1)! * r!)

Others:
Decider for a point located left or right of a line: d=(x3-x2)*(y2-y1)-(y3-y2)*(x2-x1)
Number of digits: log10(n)+1
Depth of road water: (s^2-h^2)/2h
//sum of series n/1+n/2+n/3+....n/n
ll root=sqrt(n);
for(int i=1; i<=root; i++)
sum+=n/i;
sum=(2*sum)-(root*root);

count the numbers that are divisible by given number in a certain range:a=2,b=3,c=7;
low=(a+b-1)/a;
high=c/a;
total=high-low+1;
Euler Constant: γ≈0.5772156649
#Number of squares in a n*n grid:
S=(n*(n+ 1)*(2*n + 1))/6;
#Number of rectangle in a n*n grid:
R=(n+1)*n/2*(n+1)*n/2 - S;
#Total number of rectangle and square in a n*n grid:
ans=[(n^2 + n)^2]/4
#Number of squares in a n*m grid exp:6*4
S=6*4+5*3+4*2+3*1=50
#Number of rectangles in n*m grid
R=m(m+1)n(n+1)/4
#Number of cubes in a n*n*n grid formula:n^k-(n-2)^k
C=n*(n+1)/2*n*(n+1)/2;
#Number of boxes in a n*n*n grid:
B=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2-C;
#Number of hypercube in a n^4grid:
start a loop from 1 to <=n;
HC=0;
for(i=1;i<=n;i++)

```
HC+=i*i*i*i;
```
#Number of hyper box in a n^4 grid:
```
HB=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2*(n+1)*n/2 - HC;
```

rectangle sum in ranges:

```
#define  ms0(s) memset(s,0,sizeof s)
ll bit[SZ][SZ],data[SZ][SZ],R, C;
void Update(ll r, ll c, ll val)
{for(ll i=r;i<=R;i+=i&-i)
for(ll j=c;j<=C;j+=j&-j)
  bit[i][j] += val;  }
ll Sum(ll r,ll c)
{  ll i,j,s = 0;
for (i = r; i > 0; i &= i - 1)
for (j = c; j > 0; j &= j - 1)
    s += bit[i][j];
   return s; }
int main()
{   R = C = n;
   ms0(bit);
   ms0(data);
if(!strcmp(s,"SET"))
 {   int r,c,val;
 scanf("%d %d %d",&r,&c,&val);
       r++,c++;
Update(r, c, -data[r][c] + val);
       data[r][c] = val;  }
else if(!strcmp(s,"SUM"))
{  int r1,c1,r2,c2;
  TAKEINPUT
  r1++,c1++,r2++,c2++;
  int res =0;
  res+=Sum(r2, c2) ;
  res-=Sum(r1 - 1, c2);
  res-=Sum(r2, c1 - 1);
  res+=Sum(r1 - 1, c1 - 1);
  printf("%d\n",res);}
```

Physics Formuas

motion

v = u + at

s = ut +(1/2) at*t ,

v*v − u*u = 2*a*s

Projectile motion

x = utcosθ

y = utsinθ −(1/2) gt*t

y = x tanθ − g*x*x/(2u*u*cosθ*cosθ)

T = 2u sinθ/g

R = u*u*sin2θ/g

H = u*u*sinθ*sinθ/2g

others:

p=mv

v*v/r*g = tanθ(**Banking angle)**

W = F S cos θ

K = (½)mv*v = p*p/ 2m

T = 2*π*sqrt(l/g)

Trigonometry

sin2θ=2sinθcosθ

cos2θ=cosθ*cosθ-sinθ*sinθ

sin3θ=3sinθ-4*sinθ*sinθ*sinθ

cos3θ=4*cosθ*cosθ*cosθ-3cosθ

For triangle:

a=bcosC+ccosB

b=acosC+ccosA

c=bcosA+acosB

sin(A+–B)=sinAcosB +– cosAsinB

cos(A+–B)=cosAcosB –+ sinAsinB

Circle Line intersection

```
double r, a, b, c; // given as input
//ax+by+c=0//EPS=1e-9
double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
if (c*c > r*r*(a*a+b*b)+EPS)
   puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < EPS){
puts ("1 point");
cout << x0 << ' ' << y0 << '\n';}
else {
double d = r*r - c*c/(a*a+b*b);
double mult = sqrt (d / (a*a+b*b));
double ax, ay, bx, by;
ax = x0 + b * mult;
bx = x0 - b * mult;
ay = y0 - a * mult;
by = y0 + a * mult;
puts ("2 points");
```

```
cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';}
```

Miscellaneous

Fast I/O:

```
#define pc putchar
#define fastread() (ios_base::
    sync_with_stdio(false),cin.tie(NULL));
void Cin(ll &num) {
   num = 0;
   char ch = gc();
   ll flag = 0;
   while(!((ch >= '0' & ch <= '9') || ch == '-')) {
      ch = gc();}
   if(ch == '-') {
      flag = 1;
      ch= gc();}
   while(ch >= '0' && ch <= '9') {
      num = (num << 1) + (num << 3) + ch - '0';
      ch = gc();}
   if(flag == 1) {
      num = 0 - num;}}
void Cout(ll n)
{ ll num=n,rev=n,cnt=0;
   char ch;
   if(n==0)
   {  pc('0');
      return ;}
   while(rev%10==0)
   { cnt++;rev/=10;
   }
   rev=0;
   while(num>0)
   { rev= (rev<<3) + (rev<<1) + num%10;
      num/=10;}
   while(rev>0)
   {ch=(rev%10)+'0';
      pc(ch);
      rev/=10;}
   while(cnt--)pc('0');}
```

Miscellaneous

```
#include<bits/stdc++.h>
using namespace std;
#define bug(var) cout<<#var<<" "<<var<<endl;
#define FastRead    ios::sync_with_stdio(0); cin.tie(0);
cout.tie(0);
#define pi acos(-1)
```

Generate Random

```
mt19937_64//
rng(chrono::steady_clock::now().time_since_epoch().c
ount());
inline ll gen_random(ll l, ll r) {
   return uniform_int_distribution<ll>(l, r)(rng);
}
```

Stress Test

```
for((i = 1; ; ++i)); do
echo $i
./gen.exe $i > int
diff -w <(./bad_sol.exe < int) <(./good_sol.exe < int) ||
break
done
//good_sol.cpp for bruteforce
//bad_sol.cpp for original solution
```

Sublime-build(linux)

```
{
"cmd" : ["g++ -std=c++14 $file_name -o
$file_base_name && timeout 4s
./$file_base_name<inputf.in>outputf.in"],
"selector" : "source.c",
"shell": true,
"working_dir" : "$file_path"
}
```

Sublime-build(Windows)

```
{
"cmd": ["g++.exe","-std=c++14", "${file}", "-o",
"${file_base_name}.exe", "&&" ,
"${file_base_name}.exe<inputf.in>outputf.in"],
"selector":"source.cpp",
"shell":true,
"working_dir":"$file_path"
}
```

i/o Text

```
#ifndef ONLINE_JUDGE
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
#endif
```