DSU:

```
int parent[N+5],sz[N+5];
void make_set(int v) {
  parent[v] = v;
  sz[v] = 1;
}
int find_set(int v) {
  if (v == parent[v])
    return v;
  return parent[v] =
find_set(parent[v]);
}
void union_sets(int a, int b) {
  a = find_set(a);
  b = find_set(b);
  if (a != b) {
    if (sz[a] < sz[b])
      swap(a, b);
    parent[b] = a;
    sz[a] += sz[b];
  }
}
```

DSU on Tree(Sack):

```
#define ll long long
#define maxn 100009
vector <ll> graph[maxn];
ll col[maxn], sz[maxn], cnt[maxn],
ans[maxn];
bool big[maxn];
void szdfs(ll u, ll p)
{
  sz[u] = 1;
  for(ll i = 0; i < graph[u].size(); i++) {
    ll nd = graph[u][i];
    if(nd == p)
      continue;
    szdfs(nd, u);
    sz[u] += sz[nd];
  }
}
void add(ll u, ll p, ll x)
{
  cnt[ col[u] ] += x;
  for(auto v: graph[u])
    if(v != p && !big[v])
      add(v, u, x);
}
void dfs(ll u, ll p, bool keep)
{
  ll mx = -1, bigChild = -1;
  for(auto v : graph[u])
    if(v != p && sz[v] > mx)
      mx = sz[v], bigChild = v;
  for(auto v : graph[u])
    if(v != p && v != bigChild)
      dfs(v, u, 0);  /// run a dfs on
small childs and clear them from cnt
  if(bigChild != -1) {
    dfs(bigChild, u, 1);
    big[bigChild] = 1;  /// bigChild
marked as big and not cleared from cnt
  }
  add(u, p, 1);
  ///now cnt[c] is the number of
vertices in subtree of vertex v that has
color c. You can answer the queries
easily.
  if(bigChild != -1)
    big[bigChild] = 0;
  if(keep == 0)
    add(u, p, -1);
}
//szdfs(1,-1); dfs(1,-1,0);
```

nCr, factorial, inverse mod, extended euclidean, gcd:

```
ll fact[N+5],inv[N+5];
void gen_factoruial(ll n){
  fact[0]=1;
  for(ll i=1;i<=n;i++){
    fact[i]=(i*fact[i-1])%mod;
  }
  return;
}
ll extended_euclidean(ll a, ll b, ll& x, ll&
y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclidean(b, a % b,
x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll ModularInverse(ll a,ll m){
  ll x, y;
  ll g = extended_euclidean(a, m, x, y);
  if (g != 1) {
    return -1;
  }
  else {
    x = (x % m + m) % m;
    return x;
  }
}
void gen_InverseMod(ll n, ll m){
  inv[0]=1;
  inv[1]=1;
  for(ll i=2;i<=N;i++){
    inv[i]=ModularInverse(i,m);
  }
  return;
}
void gen_Factorial_InverseMod(ll n, ll
m){
  gen_factoruial(n);
  gen_InverseMod(n,m);
  for(ll i=1;i<=N;i++){
    inv[i]*=inv[i-1];
    inv[i]%=m;
  }
  return;
}
ll ncr(ll n, ll r){
  ll ret=fact[n]*inv[r];
  ret%=mod;
  ret*=inv[n-r];
  ret%=mod;
  return ret;
}
```

Hashing A to Z:

```
typedef long long LL;
typedef pair<LL, LL> PLL;
const PLL M=mp(1e9+7, 1e9+9);
///Should be large primes
const LL base=347; ///Should be a
prime larger than highest value
///const int N = 1e6+7; ///Highest
length of string
ostream& operator<<(ostream& os,
PLL hash) {
  return os<<"("<<hash.ff<<",
"<<hash.ss<<")";
}
PLL operator+ (PLL a, LL x) {return
mp(a.ff + x, a.ss + x);}
PLL operator- (PLL a, LL x) {return
mp(a.ff - x, a.ss - x);}
PLL operator* (PLL a, LL x) {return
mp(a.ff * x, a.ss * x);}
PLL operator+ (PLL a, PLL x) {return
mp(a.ff + x.ff, a.ss + x.ss);}
```

```
PLL operator- (PLL a, PLL x) {return mp(a.ff - x.ff, a.ss - x.ss);}
PLL operator* (PLL a, PLL x) {return mp(a.ff * x.ff, a.ss * x.ss);}
PLL operator% (PLL a, PLL m) {return mp(a.ff % m.ff, a.ss % m.ss);}
PLL power (PLL a, LL p) {
  if (p==0) return mp(1,1);
  PLL ans = power(a, p/2);
  ans = (ans * ans)%M;
  if (p%2) ans = (ans*a)%M;
  return ans;
}
///Magic!!!!!!!
PLL inverse(PLL a) {
  return power(a, (M.ff-1)*(M.ss-1)-1);
}
PLL pb[N]; ///powers of base mod M
PLL invb;
///Call pre before everything
void hashPre() {
  pb[0] = mp(1,1);
  for (int i=1; i<N; i++) pb[i] = (pb[i-1] * base)%M;
  invb = inverse(pb[1]);
}
///Calculates Hash of a string
PLL Hash (string s) {
  PLL ans = mp(0,0);
  for (int i=0; i<s.size(); i++) ans=(ans*base + s[i])%M;
  return ans;
}
///appends c to string
PLL append(PLL cur, char c) {
  return (cur*base + c)%M;
}
///prepends c to string with size k
PLL prepend(PLL cur, int k, char c) {
  return (pb[k]*c + cur)%M;
}
///replaces the i-th (0-indexed) character from right from a to b;
PLL replace(PLL cur, int i, char a, char b) {
  cur = (cur + pb[i] * (b-a))%M;
  return (cur + M)%M;
}
///Erases c from the back of the string
PLL pop_back(PLL hash, char c) {
  return (((hash-c)*invb)%M+M)%M;
}
///Erases c from front of the string with size len
PLL pop_front(PLL hash, int len, char c)
{
  return ((hash - pb[len-1]*c)%M+M)%M;
}
///concatenates two strings where length of the right is k
PLL concat(PLL left, PLL right, int k) {
  return (left*pb[k] + right)%M;
}
///Calculates hash of string with size len repeated cnt times
///This is O(log n). For O(1), pre-calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
  PLL mul = (pb[len*cnt] - 1) *inverse(pb[len]-1);
  mul = (mul%M+M)%M;
  PLL ans = (hash*mul)%M;
  if (pb[len].ff == 1) ans.ff = hash.ff*cnt;
  if (pb[len].ss == 1) ans.ss = hash.ss*cnt;
  return ans;
}
 ///Calculates hashes of all prefixes of s including empty prefix
vector<PLL> hashList(string s) {
  int n = s.size();
  vector<PLL> ans(n+1);
  ans[0] = mp(0,0);
  for (int i=1; i<=n; i++) ans[i] = (ans[i-1] * base + s[i-1])%M;
  return ans;
}
///Calculates hash of substring s[l..r] (1 indexed)
PLL substringHash(const vector<PLL> &hashlist,int l, int r) {
  int len = (r-l+1);
  return ((hashlist[r] - hashlist[l-1]*pb[len])%M+M)%M;
}
BIT:
ll BITree[100009];
///do this for range: getSum(r) - getSum(l - 1)
ll getSum(ll index)
{
  ll sum = 0; // Iniialize result
  // Traverse ancestors of BITree[index]
  while (index>0)
  {
    // Add current element of BITree to sum
    sum += BITree[index];
    // Move index to parent node in getSum View
    index -= index & (-index);
  }
  return sum;
}
void updateBIT(ll n, ll index, ll val)
{
  // Traverse all ancestors and add 'val'
  while (index <= n)
  {
    // Add 'val' to current node of BI Tree
    BITree[index] += val;
    // Update index to that of parent in update View
    index += index & (-index);
  }
}
```

Digit DP:

```
///How many zeros in the numbers' digits. Range of the numbers is (l, r)
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
ll dp[2][2][12][12];
vector <ll> num;
ll solve(ll isStart, ll isSmall, ll pos, ll val)
{
  if(pos == num.size())
    return val;
  ll &ret = dp[isStart][isSmall][pos][val];
  if(ret != -1)
    return ret;
  ll lim;
  if(isSmall)
    lim = 9;
  else
    lim = num[pos];
ll rt = 0;
  if(!isStart) {
    for(ll i = 0; i <= lim; i++)
```

```
        rt += solve(0, isSmall | i < num[pos], pos + 1, (i == 0) + val);
    }
    else {
        for(ll i = 1; i <= lim; i++)
            rt += solve(0, isSmall | i < num[pos], pos + 1, val);
        rt += solve(1, 1, pos + 1, 0);
    }
    return ret = rt;
}
ll calc(ll n)
{
    if(n < 0)
        return 0;
    if(n < 10)
        return 1;
    ll tmp = n;
    num.clear();
    while(tmp) {
        num.pb(tmp % 10);
        tmp /= 10;
    }
    reverse(num.begin(), num.end());
    return solve(1, 0, 0, 0) + 1;      /// + 1 is for the number "0". We are not calculating this number in solve function.
}
int main()
{
    ll t, caseno = 0;
    cin >> t;
    while(t--) {
        memset(dp, -1, sizeof(dp));
        ll l, r;
        scanf("%lld %lld", &l, &r);
        ll ans = calc(r);
        memset(dp, -1, sizeof(dp));
        ans -= calc(l - 1);
        printf("Case %lld: %lld\n", ++caseno, ans);
    }
    return 0;
}
```

Digit DP (One time memset):

```
///How many zeros in the numbers' digits. Range of the numbers is (l, r)
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
ll dp[2][2][12][12];
vector <ll> num;
ll solve(ll isStart, ll isSmall, ll pos, ll val)
{
    if(pos == 0)
        return val;
    ll &ret = dp[isStart][isSmall][pos][val];
    if(ret != -1 && isSmall)
        return ret;
    ll lim, pos2 = num.size() - pos;
    if(isSmall)
        lim = 9;
    else
        lim = num[pos2];
    ll rt = 0;
    if(!isStart) {
        for(ll i = 0; i <= lim; i++)
            rt += solve(0, isSmall | i < num[pos2], pos - 1, (i == 0) + val);
    }
    else {
        for(ll i = 1; i <= lim; i++)
            rt += solve(0, isSmall | i < num[pos2], pos - 1, val);
        rt += solve(1, 1, pos - 1, 0);
    }
    return ret = rt;
}
ll calc(ll n)
{
    if(n < 0)
        return 0;
    if(n < 10)
        return 1;
    ll tmp = n;
    num.clear();
    while(tmp) {
        num.pb(tmp % 10);
        tmp /= 10;
    }
    reverse(num.begin(), num.end());
    return solve(1, 0, num.size(), 0) + 1;
/// + 1 is for the number "0". We are not calculating this number in solve function.
}
int main()
{
    ll t, caseno = 0;
    memset(dp, -1, sizeof(dp));
    cin >> t;
    while(t--) {
        ll l, r;
        scanf("%lld %lld", &l, &r);
        ll ans = calc(r);
        ans -= calc(l - 1);
        printf("Case %lld: %lld\n", ++caseno, ans);
    }
    return 0;
}
```

CHT:

```
vector<ll> m,c;
int p;
bool useless_inc(int f1,int f2,int f3){
    return (c[f3]-c[f1])*1.0L*(m[f1]-m[f2]) > (c[f2]-c[f1])*1.0L*(m[f1]-m[f3]);
}
bool useless_dec(int f1,int f2,int f3){
    return (c[f3]-c[f1])*1.0L*(m[f1]-m[f2]) < (c[f2]-c[f1])*1.0L*(m[f1]-m[f3]);
}
void add_line_inc(ll mm, ll cc){
    m.pb(mm);
    c.pb(cc);
    int sz=m.size();
    while(sz>=3){
        if(useless_inc(sz-3,sz-2,sz-1)){
            m.erase(m.end()-2);
            c.erase(c.end()-2);
            sz--;
        }
        else break;
    }
}
void add_line_dec(ll mm, ll cc){
    m.pb(mm);
    c.pb(cc);
    int sz=m.size();
    while(sz>=3){
        if(useless_dec(sz-3,sz-2,sz-1)){
            m.erase(m.end()-2);
            c.erase(c.end()-2);
            sz--;
        }
        else break;
    }
}
void add_dec_min(ll mm, ll cc){
    add_line_dec(mm,cc);
}
void add_dec_max(ll mm, ll cc){
```

```
   add_line_inc(-mm,-cc);
}
void add_inc_min(ll mm, ll cc){
   add_line_inc(mm,cc);
}
void add_inc_max(ll mm, ll cc){
   add_line_dec(-mm,-cc);
}
ll query_linear(ll x,bool mn,bool dec){
   if(p>=m.size()) p=m.size()-1;
   while(p<(m.size()-1)){

if((m[p]*x+c[p])<(m[p+1]*x+c[p+1]))
break;
      p++;
   }
   if(!dec)
   while(p>0){
      if((m[p-1]*x+c[p-
1])>(m[p]*x+c[p])) break;;
      p--;
   }
   //cout<<"->"<<p<<"\n";
   if(mn) return (m[p]*x+c[p]);
   else return -(m[p]*x+c[p]);
}
ll query_binsearch(ll x,int mn){
   int l=-1,r=m.size()-1;
   while(r-l>1){
      int mid=(l+r)/2;

if((m[mid]*x+c[mid])>=(m[mid+1]*x+c[
mid+1])) l=mid;
      else r=mid;
   }
   if(mn) return (m[r]*x+c[r]);
   else return -(m[r]*x+c[r]);
}
int main()
{
   int n,t,q;scn(n),scn(t);
   p=0;
   ll mm,cc,x,ans;
   for(int i=1;i<=n;i++){
      scn(q);
      if(q==1){
         scnll(mm),scnll(cc);
         if(t==1){
            add_dec_min(mm,cc);
         }
         else if(t==2){
            add_dec_max(mm,cc);
         }
         else if(t==3){
           add_inc_min(mm,cc);
         }
         else{
            add_inc_max(mm,cc);
         }
      }
      else{
         scnll(x);
         bool op=false,dec=false;
         if(t==1||t==3){
            op=true;
         }
         if(t==1||t==4){
            dec=true;
         }
         ans=query_linear(x,op,dec);
         printf("%lld\n",ans);
      }
   }
}
```

Dynamic CHT:

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you
can add lines of the form mx+c, and
query max values at points x.
          For min query, add line in (-m,
-c) format. You will get -ans.
 *  Useful for dynamic programming
(``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */
struct Line {
   mutable ll m, c, p;
   bool isQuery;
   bool operator<(const Line& o) const
{
      if(o.isQuery)
         return p < o.p;
      return m < o.m;
   }
};
struct LineContainer : multiset<Line> {
   // (for doubles, use inf = 1/.0,
div(a,b) = a/b)
   const ll inf = LLONG_MAX;
   ll div(ll a, ll b) { // floored division
      return a / b - ((a ^ b) < 0 && a %
b); }
   bool isect(iterator x, iterator y) {
      if (y == end()) { x->p = inf; return
false; }
      if (x->m == y->m) x->p = x->c > y->c
? inf : -inf;
      else x->p = div(y->c - x->c, x->m - y-
>m);
      return x->p >= y->p;
   }
   void add(ll m, ll c) {
auto z = insert({m, c, 0, 0}), y = z++, x =
y;
      while (isect(y, z)) z = erase(z);
      if (x != begin() && isect(--x, y))
isect(x, y = erase(y));
      while ((y = x) != begin() && (--x)->p
>= y->p)
         isect(x, erase(y));
   }
   ll query(ll x) {
      if(empty()) return inf;
      Line q; q.p = x, q.isQuery = 1;
      auto l = *lower_bound(q);
      return l.m * x + l.c;
   }
};
```

FFT:

```
/***
  * Multiply (7x^2 + 8x^1 + 9x^0) with
(6x^1 + 5x^0)
  * ans = 42x^3 + 83x^2 + 94x^1 +
45x^0
  * A = {9, 8, 7}
  * B = {5, 6}
  * V = multiply(A,B)
  * V = {45, 94, 83, 42}
***/
/*** Tricks
  * Use vector < bool > if you need to
check only the status of the sum
  * Use bigmod if the power is over
same polynomial && power is big
  * Use long double if you need more
precision
  * Use long long for overflow
***/
typedef vector <int> vi;
const double PI = 2.0 * acos(0.0);
using cd = complex<double>;
```

```
void fft(vector<cd> & a, bool invert =
0)
{
   int n = a.size();
   for (int i = 1, j = 0; i < n; i++) {
      int bit = n >> 1;
      for (; j & bit; bit >>= 1)
         j ^= bit;
      j ^= bit;
      if (i < j)
         swap(a[i], a[j]);
   }
   for (int len = 2; len <= n; len <<= 1) {
      double ang = 2 * PI / len * (invert ?
-1 : 1);
      cd wlen(cos(ang), sin(ang));
      for (int i = 0; i < n; i += len) {
         cd w(1);
         for (int j = 0; j < len / 2; j++) {
         cd u = a[i+j], v = a[i+j+len/2] * w;
            a[i+j] = u + v;
            a[i+j+len/2] = u - v;
            w *= wlen;
         }
      }
   }
   if (invert) {
      for (cd & x : a)
         x /= n;
   }
}
void ifft(vector <cd> & p)
{
   fft(p, 1);
}
vi multiply(vi const& a, vi const& b)
{
   vector<cd> fa(a.begin(), a.end()),
fb(b.begin(), b.end());
   int n = 1;
   while (n < a.size() + b.size())
      n <<= 1;
   fa.resize(n);
   fb.resize(n);
   fft(fa);
   fft(fb);
   for (int i = 0; i < n; i++)
      fa[i] *= fb[i];
   ifft(fa);
   vi result(n);
   for (int i = 0; i < n; i++)
      result[i] = round(fa[i].real());
   return result;
}
```

Palindromic Tree:

```
#define ll long long
const ll N = 1e5+10;
int tree[N][26], idx;
ll len[N], link[N], cnt[N], t;
char s[N]; // 1-indexed
void add(ll p) {
   while(s[p - len[t] - 1] != s[p]) t =
link[t];   // searching node for creating
pTp type palindrome.
   ll x = link[t], c = s[p] - 'a';
   while(s[p - len[x] - 1] != s[p]) x =
link[x];   // searching node to link pXp
type palindrome, where pXp is a
proper suffix.
   if(!tree[t][c]) {
      tree[t][c] = ++idx;
      len[idx] = len[t] + 2;
      link[idx] = len[idx] == 1 ? 2 :
tree[x][c];
   }
   t = tree[t][c];
   cnt[t]++;
}
/* node 1 and node 2 are the two
roots.
 * idx-2 is the number of total distinct
palindromes in the string s.
 * Let, a node is i,
 * len[i] represents the length of the
palindrome represented by node i.
 * link[i] represents the node
containing the palindrome which is the
largest proper suffix
 * of the palindrome of node i.
*/
int main()
{
   len[1] = -1, link[1] = 1;
   len[2] = 0, link[2] = 1;
   idx = t = 2;
   memset(tree, 0, sizeof(tree));
   memset(cnt, 0, sizeof(cnt));
   scanf("%s", s+1);
   ll len = strlen(s+1);
   for(ll i = 1; i <= len; i++) add(i);    //
adding each index in pal tree one by
one. O(len).
   for(ll i = idx; i > 2; i--) cnt[ link[i] ] +=
cnt[i];   // adding count to the suffix
link.
   // cnt[i] now holds the count of the
palindrome represented by node i in
the string s.
   return 0;
}
```

SOS DP:

```
/// Given a fixed array A of 2^N
integers, we need to calculate
/// the function F(x) = Sum of all A[i]
such that x&i = i, i.e., i is a subset of x.
/// It means i is the subset bitmask of
the bitmask of x.
/// Suboptimal Bruteforce Method
O(3^n):
// iterate over all the masks
for (int mask = 0; mask < (1<<n);
mask++) {
   F[mask] = A[0];
   // iterate over all the subsets of the
mask
   for(int i = mask; i > 0; i = (i-1) &
mask){
      F[mask] += A[i];
   }
}
/// Two DP methods O(n*2^n):
/// iterative version
for(int mask = 0; mask < (1<<N);
mask++){
   dp[mask][0] = A[mask];
         //handle base case separately
(leaf states)
   for(int i = 0;i < N; i++){
      if(mask & (1<<i))
         dp[mask][i + 1] = dp[mask][i] +
dp[mask^(1<<i)][i];
      else
         dp[mask][i + 1] = dp[mask][i];
   }
   F[mask] = dp[mask][N];
}
/// memory optimized, super easy to
code.
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)];
   }
}
```

Matrix Exponentiation:

```
ll f[102];
struct matrix{
  ll e[102][102];
};
matrix A;
matrix solve(ll n,ll sz){
  matrix x,z;
  if(n==1LL) return A;
  matrix y=solve(n/2LL,sz);
  for(ll i=1;i<=sz;i++){
    for(ll j=1;j<=sz;j++){
      ll sum=0LL;
      for(ll k=1;k<=sz;k++){
        sum+=(y.e[i][k]*y.e[k][j]);
        sum%=mod;
      }
      x.e[i][j]=sum;
    }
  }
  if(n%2){
    for(ll i=1;i<=sz;i++){
      for(ll j=1;j<=sz;j++){
        ll sum=0LL;
        for(ll k=1;k<=sz;k++){
          sum+=(x.e[i][k]*A.e[k][j]);
          sum%=mod;
        }
        z.e[i][j]=sum;
      }
    }
  }
  else z=x;
  return z;
}
int main()
{
  ll
n,m,k,ans=0,u,v;scnll(n),scnll(m),scnll(k
);
  for(ll i=1;i<=m;i++){
    scnll(u),scnll(v);
    A.e[v][u]=1;
    f[v]++;
  }
  if(k>1){
    matrix res=solve(k-1,n);
    for(ll i=1;i<=n;i++){
      for(ll j=1;j<=n;j++){
        ans+=(res.e[i][j]*f[j]);
        ans%=mod;
      }
    }
  }
  else{
    for(ll i=1;i<=n;i++){
      ans+=f[i];
      ans%=mod;
    }
  }
  printf("%lld\n",ans);
}
```

Suffix Array:

```
// O(n log n) Suffix Array
#define MAX_N 1000020
int n, t;
char s[MAX_N];
int SA[MAX_N], LCP[MAX_N];
int RA[MAX_N], tempRA[MAX_N];
int tempSA[MAX_N];
int c[MAX_N];
int Phi[MAX_N], PLCP[MAX_N];
void countingSort(int k) {    // O(n)
  int i, sum, maxi = max(300, n);
  // up to 255 ASCII chars or length of
n
  memset(c, 0, sizeof c);
  // clear frequency table
  for (i = 0; i < n; i++)
  // count the frequency of each
integer rank
  c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++)
    // shuffle the suffix array if
necessary
    tempSA[c[SA[i] + k < n ? RA[SA[i] +
k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)
    // update the suffix array SA
    SA[i] = tempSA[i];
}
void buildSA() {
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = s[i];
  // initial rankings
  for (i = 0; i < n; i++) SA[i] = i;
    if(v != p && v != bigChild)
      dfs(v, u, 0);  /// run a dfs on
small childs and clear them from cnt
  if(bigChild != -1) {
    dfs(bigChild, u, 1);
    big[bigChild] = 1;  /// bigChild
marked as big and not cleared from cnt
  }
  add(u, p, 1);
  ///now cnt[c] is the number of
vertices in subtree of vertex v that has
color c. You can answer the queries
easily.
  if(bigChild != -1)
    big[bigChild] = 0;
  if(keep == 0)
    add(u, p, -1);
}
//szdfs(1,-1); dfs(1,-1,0);
```

Miller Rabin Primality Test:

```
/* Miller Rabin Primality Test for <=
10^18 */
ll mulmod(ll a, ll b, ll c)
{
  ll x = 0, y = a % c;
  while (b)
  {
    if (b & 1) x = (x + y) % c;
    y = (y << 1) % c;
    b >>= 1;
  }
  return x % c;
}
ll fastPow(ll x, ll n, ll MOD)
{
  ll ret = 1;
  while (n)
  {
    if (n & 1) ret = mulmod(ret, x,
MOD);
    x = mulmod(x, x, MOD);
    n >>= 1;
  }
  return ret % MOD;
}
const int a[9] = { 2, 3, 5, 7, 11, 13, 17,
19, 23 };
bool isPrime(ll n)
{
  if(n == 2 || n == 3) return true;
  if(n == 1 || !(n & 1)) return false;
  ll d = n - 1;
  int s = 0;
  while (d % 2 == 0)
  {
    s++;
    d /= 2;
```

```
    }
    for(int i = 0; i < 9; i++)
    {
        if(n == a[i]) return true;
        bool comp = fastPow(a[i], d, n) !=
1;
        if(comp) for(int j = 0; j < s; j++)
            {
                ll fp = fastPow(a[i], (1LL <<
(ll)j)*d, n);
                if (fp == n - 1)
                {
                    comp = false;
                    break;
                }
            }
        if(comp) return false;
    }
    return true;
}
```

Trie:

```
int nxt[100005][10], cnt[100005];
int cnt_node=1;
void add_string(char s[]){
            int len=strlen(s);
            int node=1,ch;
            for(int i=0;i<len;i++){
                        ch=s[i]-'0';
                        cnt[node]++;
                        if(nxt[node][ch]==0) {
          cnt_node++;
          nxt[node][ch]=cnt_node;
       }
                        node=nxt[node][ch];
            }
            cnt[node]++;
}
int query(char s[]){
            int len=strlen(s);
            int node=1,ch;
            for(int i=0;i<len;i++){
                        int ch=s[i]-'0';
                        if(nxt[node][ch]==0)
return 0;
                        node=nxt[node][ch];
            }
            return cnt[node];
}
char s[N][12];
int main()
{
    int t,ts=1;scn(t);
    while(t--){
        int n;scn(n);
        for(int i=0;i<n;i++){
            scanf(" %s",s[i]);
            add_string(s[i]);
        }
        int f=0;
        for(int i=0;i<n;i++){
            int x=query(s[i]);
            if(x>1){
                f=1;
                break;
            }
        }
        if(f) NO;
        else YES;
        for(int i=0;i<=cnt_node;i++){
            cnt[i]=0;
            for(int j=0;j<10;j++) nxt[i][j]=0;
        }
        cnt_node=1;
    }
}
```

Segment Tree:

```
int seg[N];
int a[N];
void build_seg(int node,int l,int r){
    if(l==r){
        seg[node]=a[l];
        return;
    }
    int mid=(l+r)>>1;
    build_seg(node<<1,l,mid);
    build_seg((node<<1)+1,mid+1,r);

seg[node]=seg[node<<1]+seg[(node<<
1)+1];
    return;
}
void update(int node,int l,int r,int i,int
j,int val){
    if(i>r||j<l) return;
    if(i<=l&&j>=r){
        if(val<0) seg[node]=0;
        else seg[node]+=val;
        return;
    }
    int mid=(l+r)>>1;
    update(node<<1,l,mid,i,j,val);
    update((node<<1)+1,mid+1,r,i,j,val);
seg[node]=seg[node<<1]+seg[(node<<
1)+1];
    return;
}
int query(int node,int l,int r,int i,int j){
    if(i>r||j<l) return 0;
    if(i<=l&&j>=r){
        return seg[node];
    }
    int mid=(l+r)>>1;
    int res1=query(node<<1,l,mid,i,j);
    int
res2=query((node<<1)+1,mid+1,r,i,j);
    int ret=res1+res2;
    return ret;
}
```

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define l1(i, n) for (ll i = 1; i <= n; i++)
#define l0(i, n) for (ll i = 0; i < n; i++)
#define pb push_back
#define sorted(x) sort(x.begin(), x.end())
#define reversed(x) reverse(x.begin(), x.end())
#define all(x) x.begin(), x.end()
#define ms(a, b) memset(a, b, sizeof(a));
#define cases(tc) cout<<"Case #"<<tc<<": "
#define nl cout<<"\n";
#define pi acos(-1)
#define mod 1000000007
#define inf 999999999999999999
#define maxn 100001
#define xx first
#define yy second

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0), cout.tie(0);
  ll t;
  cin>>t;
  while(t--){
    ll n;
    cin>>n;
  }
  return 0;
}
```

DEBUG:

```
#define Gene template< class
#define Rics printer& operator,
Gene c > struct rge {c b, e;};
Gene c > rge<c> range(c i, c j) { return {i, j};}
struct printer {
  ~printer() {cerr << endl;}
  Gene c > Rics(c x) { cerr << boolalpha << x; return *this;}
  Rics(string x) {cerr << x; return *this;}
  Gene c, class d > Rics(pair<c, d> x) { return *this, "(", x.first, ", ", x.second, ")";}
  Gene ... d, Gene ... > class c > Rics(c<d...> x) { return *this, range(begin(x), end(x));}
  Gene c > Rics(rge<c> x) {*this, "["; for (auto it = x.b; it != x.e; ++it)*this, (it == x.b ? "" : ", "), *it; return *this, "]";
  }
};
#define debug() cerr<<"LINE "<<__LINE__<<" >> ", printer()
#define dbg(x) "[",#x,": ",(x),"] "
```

UNIFORM RANDOM:

```
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int my_rand(int l, int r) {
  return uniform_int_distribution<int>(l, r)(rng);
}
```

PBDS:

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
 #define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_order_statistics_node_update>
//declare ordered_set setname
//setname.find_by_order(k): It returns to an iterator to the kth element (counting from zero) in the set in O(logn)
//setname.order_of_key(k) : It returns to the number of items that are strictly smaller than our item k in O(logn) time
```

POWER:

```
ll power(ll a, ll b){
  if(b==0) return 1;
  ll temp=power(a, b/2);
  if(b & 1) return a*temp*temp;
  else return temp*temp; }
```

MODULAR EXPONENTATION:

```
ll pmod(ll a, ll b, ll mod){
  if(b==0) return 1;
  ll temp=pmod(a, b/2, mod);
  if(b & 1) return (((a*temp)%mod)*temp)%mod;
  else return (temp*temp)%mod; }
```

SIEVE:

```
vector <ll> primes;
ll chk[maxn];
void sieve(){
  for(ll i=2; i*i<maxn; i++){
    if(!chk[i]){
      for(ll j=i*i; j<maxn; j+=i) if(!chk[j]) chk[j]=i;
    }
  }
  for(ll i=2; i<maxn; i++) if(!chk[i]) chk[i]=i;
  for(ll i=2; i<maxn; i++) if(chk[i]==i) primes.pb(i);
}
```

PRIME FACTORIZE:

```
//SIEVE
vector <ll> fac;
void factors(ll a){
  fac.clear();
  ll val=a;
  for(ll i=0; i<primes.size() && primes[i]*primes[i]<=a; i++){
    if(!(val%primes[i])){
      fac.pb(primes[i]);
      while(!(val%primes[i])){
        val/=primes[i];
      }
    }
  }
  if(val!=1) fac.pb(val); }
```

BFS:

```
vector <ll> adj[maxn];
ll dis[maxn];
void bfs(ll n, ll st){
  l1(i, n) dis[i]=-1;
  queue <ll> q;
  q.push(st);
  dis[st]=0;
  while(!q.empty()){
    ll a=q.front();
    q.pop();
    for(ll i=0; i<adj[a].size(); i++){
      if(dis[adj[a][i]]==-1){
        dis[adj[a][i]]=dis[a]+1;
        q.push(adj[a][i]);
      } } } }
```

DFS:

```
vector <ll> adj[maxn];
bool visited[maxn];
void dfs(ll a){
  if(!visited[a]){
    visited[a]=true;
    for(ll i=0; i<adj[a].size(); i++){
      dfs(adj[a][i]);
    }
```

```
  }
  return; }
```
DJIKSTRA:
```
vector <pair<ll, ll>> adj[maxn];
vector <ll> dist(maxn, inf);
vector <bool> visited(maxn);
void dijkstra(ll s){
  priority_queue<pair<ll, ll>> pq;
  pq.push({0, s});
  dist[s]=0;
  while(!pq.empty()){
    ll u=pq.top().second;
    pq.pop();
    if(visited[u]) continue;
    visited[u]=true;
    for(ll i=0; i<adj[u].size(); i++){e
      ll v=adj[u][i].first;
      ll w=adj[u][i].second;
      if(dis[v]>dis[u]+w){
        dis[v]=dis[u]+w;
        pq.push({-dis[v], v});
      } } } }
```
DSU:
```
ll cnt;
ll par[maxn];
ll rnk[maxn];
ll sz[maxn];
void make_set(ll a){
  par[a]=a;
  rnk[a]=1;
  sz[a]=1; }
ll find(ll a){
  if(a==par[a]) return a;
  return par[a]=find(par[a]); }
void merge(ll a, ll b){
  ll p1=find(a);
  ll p2=find(b);
  if(p1==p2){
    return;
  }
  if(p1>p2) swap(p1, p2);
  par[p1]=p2;
  sz[p2]+=sz[p1];
  if(rnk[p1]==rnk[p2]) rnk[p2]++;
  cnt--; }
bool same(ll a, ll b){
  return(par[a]==par[b]); }
ll count(){
  return cnt; }
ll get_size(ll a){
  return sz[par[a]];}
```
MSTKRUSKAL
```
//dsu
int main()
{
  ll t;
  t=1;
  while(t--){
    cnt=0;
    ll n, m;
    cin>>n>>m;
    vector<vector <ll>> v(m, vector
<ll> (3));
    ll ans=0;
    l0(i, n) make_set(i);
    l0(i, m){
      ll a, b, c;
      cin>>a>>b>>c;
      v[i]={c, a, b};
    }
    sorted(v);
    l0(i, m){
      if(find(v[i][1])!=find(v[i][2])){
        merge(v[i][1], v[i][2]);
        ans+=v[i][0];
      }
    }
    cout<<ans;
    nl
  }
  return 0; }
```
EULERTOUR:
```
vector <ll> adj[maxn];
ll dt[2*maxn];
ll st[maxn];
ll en[maxn];
ll cnt=0;
void dfs(ll a, ll p){
  dt[++cnt]=a;
  st[a]=cnt;
  for(ll i=0; i<adj[a].size(); i++){
    if(adj[a][i]!=p) dfs(adj[a][i], a);
  }
  dt[++cnt]=-a;
  en[a]=cnt;
  return;}
```
CENTROID DECOMPOSITION
```
vector <ll> adj[maxn];
ll del[maxn], sz[maxn], par[maxn];
ll cursz;
void dfs(ll a, ll p){
  sz[a]=1;
  for(ll i=0; i<adj[a].size(); i++){
    ll nd=adj[a][i];
    if(nd!=p && !del[nd]){
      dfs(adj[a][i], a);
      sz[a]+=sz[adj[a][i]];
    } } }
ll findcen(ll a, ll p){
  for(ll i=0; i<adj[a].size(); i++){
    ll nd=adj[a][i];
    if(nd!=p && !del[nd] &&
sz[nd]>cursz/2){
      return findcen(nd, a);
    }
  }
  return a;  }
void decomp(ll a, ll p){
  dfs(a, -1);
  cursz=sz[a];
  ll cen=findcen(a, -1);
  if(p==-1) p=cen;
  par[cen]=p, del[cen]=1;

  for(ll i=0; i<adj[cen].size(); i++){
    ll nd=adj[cen][i];
    if(!del[nd]) decomp(nd, cen);
  } }
```
LCA:
```
vector <ll> adj[maxn];
ll par[maxn][19];
ll lev[maxn];
void dfs(ll a, ll p){
  par[a][0]=p;
  lev[a]=lev[p]+1;
  for(ll i=1; i<=18; i++){
    par[a][i]=par[par[a][i-1]][i-1];
  }
  for(ll i=0; i<adj[a].size(); i++){
    if(adj[a][i]!=p) dfs(adj[a][i], a);
  }
  return; }
ll lca(ll u, ll v) {
  if (lev[u]<lev[v]) swap(u, v);
  for(ll k=18; k>=0; k--) if (lev[par[u][k]]
>= lev[v]) u = par[u][k];
  if (u == v) return u;
  for(ll k=18; k>=0; k--) if (par[u][k] !=
par[v][k]) u = par[u][k], v = par[v][k];
  return par[u][0]; }
```
SEGMENT TREE:
```
ll seg[4*maxn];
ll lazy[4*maxn];
ll dt[maxn];
void build(ll st, ll en, ll nd){
  if(st==en){
    seg[nd]=dt[st];
    return;
  }
```

```
   ll mid=(st+en)/2;
   build(st, mid, 2*nd);
   build(mid+1, en, 2*nd+1);
   seg[nd]=seg[2*nd]+seg[2*nd+1]; }
void update(ll st, ll en, ll nd, ll l, ll r, ll
val){
   if(lazy[nd]!=0){
      ll temp=lazy[nd];
      lazy[nd]=0;
      seg[nd]+=temp*(en-st+1);
      if(st!=en){
         lazy[2*nd]+=temp;
         lazy[2*nd+1]+=temp;
      }
   }
   if(st>r || en<l){
      return;
   }
   if(st>=l && en<=r){
      seg[nd]+=val*(en-st+1);
      if(st!=en){
         lazy[2*nd]+=val;
         lazy[2*nd+1]+=val;
      }
      return; }
   ll mid=(st+en)/2;
   update(st, mid, 2*nd, l, r, val);
   update(mid+1, en, 2*nd+1, l, r, val);
   seg[nd]=seg[2*nd]+seg[2*nd+1]; }
ll query(ll st, ll en, ll nd, ll l, ll r){
   if(lazy[nd]!=0){
      ll temp=lazy[nd];
      lazy[nd]=0;
      seg[nd]+=temp*(en-st+1);
      if(st!=en){
         lazy[2*nd]+=temp;
         lazy[2*nd+1]+=temp;
      }
   }
   if(st>r || en<l){
      return 0;
   }
   if(st>=l && en<=r){
      return seg[nd];
}
   ll mid=(st+en)/2;
   return query(st, mid, 2*nd, l, r) +
query(mid+1, en, 2*nd+1, l, r); }
```

PERSISTANT SEGMENT TREE:

```
ll dt[maxn];
ll seg[120*maxn];
ll lazy[120*maxn];
ll lef[120*maxn];
ll rig[120*maxn];
ll ver[maxn];
ll nf=1;
void build(ll st, ll en, ll nd){
   if(st==en){
      seg[nd]=dt[st];
      return;
   }
   lef[nd]=++nf;
   rig[nd]=++nf;
   ll mid=(st+en)/2;
   build(st, mid, lef[nd]);
   build(mid+1, en, rig[nd]);
   seg[nd]=seg[lef[nd]]+seg[rig[nd]]; }
ll propogate(ll st, ll en, ll nd, ll val){
   ll nnd=++nf;
   lef[nnd]=lef[nd];
   rig[nnd]=rig[nd];
   lazy[nnd]=lazy[nd];
   lazy[nnd]+=val;
   seg[nnd]=seg[nd]+(en-st+1)*val;
   return nnd; }
ll update(ll st, ll en, ll nd, ll l, ll r, ll val){
   if(lazy[nd]!=0){
      ll temp=lazy[nd];
      lazy[nd]=0;
      if(st!=en){
         ll mid=(st+en)/2;
         lef[nd]=propogate(st, mid,
lef[nd], temp);
         rig[nd]=propogate(mid+1, en,
rig[nd], temp);
      }
   }
   if(st>r || en<l){
      return nd;
   }
   ll nnd=++nf;
   if(st>=l && en<=r){
      seg[nnd]=seg[nd]+(en-st+1)*val;
      lazy[nnd]=val;
      lef[nnd]=lef[nd];
      rig[nnd]=rig[nd];
      return nnd;
   }
   ll mid=(st+en)/2;
   lef[nnd]=update(st, mid, lef[nd], l, r,
val);
   rig[nnd]=update(mid+1, en, rig[nd],
l, r, val);

seg[nnd]=seg[lef[nnd]]+seg[rig[nnd]];
   return nnd; }
ll query(ll st, ll en, ll nd, ll l, ll r){
   if(lazy[nd]!=0){
      ll temp=lazy[nd];
      lazy[nd]=0;
      if(st!=en){
         ll mid=(st+en)/2;
         lef[nd]=propogate(st, mid,
lef[nd], temp);
         rig[nd]=propogate(mid+1, en,
rig[nd], temp);
      }
   }
   if(st>r || en<l){
      return 0;
   }
   if(st>=l && en<=r){
      return seg[nd];
   }
   ll mid=(st+en)/2;
   return query(st, mid, lef[nd], l, r) +
query(mid+1, en, rig[nd], l, r); }
```

PERSISTANT SEGMENT TREE LAZY:

```
seg[120*maxn];
ll lazy[120*maxn];
ll lef[120*maxn];
ll rig[120*maxn];
ll ver[maxn];
ll nf=1;
// Take build() from above
ll propogate(ll st, ll en, ll nd, ll val){
   ll nnd=++nf;
   lef[nnd]=lef[nd];
   rig[nnd]=rig[nd];
   lazy[nnd]=lazy[nd];
   lazy[nnd]+=val;
   seg[nnd]=seg[nd]+(en-st+1)*val;
   return nnd; }
ll update(ll st, ll en, ll nd, ll l, ll r, ll val){
   if(lazy[nd]!=0){
      ll temp=lazy[nd];
      lazy[nd]=0;
      if(st!=en){
         ll mid=(st+en)/2;
         lef[nd]=propogate(st, mid,
lef[nd], temp);
         rig[nd]=propogate(mid+1, en,
rig[nd], temp);
      }
   }
   if(st>r || en<l){
      return nd;
   }
   ll nnd=++nf;
```

```
  if(st>=l && en<=r){
    seg[nnd]=seg[nd]+(en-st+1)*val;
    lazy[nnd]=val;
    lef[nnd]=lef[nd];
    rig[nnd]=rig[nd];
    return nnd;
  }
  ll mid=(st+en)/2;
  lef[nnd]=update(st, mid, lef[nd], l, r,
val);
  rig[nnd]=update(mid+1, en, rig[nd],
l, r, val);

seg[nnd]=seg[lef[nnd]]+seg[rig[nnd]];
  return nnd; }
ll query(ll st, ll en, ll nd, ll l, ll r){
  if(lazy[nd]!=0){
    ll temp=lazy[nd];
    lazy[nd]=0;
    if(st!=en){
      ll mid=(st+en)/2;
      lef[nd]=propogate(st, mid,
lef[nd], temp);
      rig[nd]=propogate(mid+1, en,
rig[nd], temp);
    }
  }
  if(st>r || en<l){
    return 0;
  }
  if(st>=l && en<=r){
    return seg[nd];
  }
  ll mid=(st+en)/2;
  return query(st, mid, lef[nd], l, r) +
query(mid+1, en, rig[nd], l, r); }
```

MO'S ALGORITHM:

```
ll sq;
bool compare(pair<ll, ll> p1, pair<ll, ll>
p2){
  if(p1==p2) return false;
  if(p1.first/sq!=p2.first/sq) return
p1.first/sq<p2.first/sq;
  return p1.second<p2.second;
}
int main()
{
  ll t;
  t=1;
  while(t--){
    ll n, q;
    cin>>n>>q;
    vector <ll> x(n+1);
    l1(i, n) cin>>x[i];
    vector <pair<ll, ll>> v;
    l0(i, q){
      ll a, b;
      cin>>a>>b;
      v.pb({a, b});
    }
    sq=sqrt(n);
    sort(v.begin(), v.end(), compare);
    vector <ll> freq(n+1, 0);
    ll l=0, r=-1;
    ll ans=0;
    l0(i, q){
      if(l<v[i].first){
        while(l!=v[i].first){
          freq[x[l]]--;
          if(!freq[x[l]]) ans--;
          l++;
        }
      }
      else if(l>v[i].first){
        while(l!=v[i].first){
          freq[x[l-1]]++;
          if(freq[x[l-1]]==1) ans++;
          l--;
        }
      }
      if(r<v[i].second){
        while(r!=v[i].second){
          freq[x[r+1]]++;
          if(freq[x[r+1]]==1) ans++;
          r++;
        }
      }
      else if(r>v[i].second){
        while(r!=v[i].second){
          freq[x[r]]--;
          if(!freq[x[r]]) ans--;
          r--;
        }
      }
      cout<<ans;
      nl
    }
  }
  return 0; }
```

SPARSE TABLE:

```
ll st[maxn][32];
void buildSparseTable(ll dt[], ll n){
  for (ll i=0; i<n; i++) st[i][0]=dt[i];
  for (ll j=1; (1<<j)<=n; j++){
    for(ll i=0; (i+(1<<j)-1)<n; i++)
st[i][j]=min(st[i][j-1], st[i+(1<<(j-1))][j-
1]);
  } }
ll query(ll l, ll r)
{
  ll j=(ll)log2(r-l+1);
  if (st[l][j]<=st[r-(1<<j)+1][j]) return
st[l][j];
  else return st[r-(1<<j)+1][j]; }
```

DOUBLE HASH:

```
ll base1 = 1e9+21, base2 = 1e9+181,
mod=2000000063;
string s;
ll pw1[maxn], pw2[maxn], len;
void pw_calc() {
  pw1[0] = pw2[0] = 1;
  for(int i = 1; i < maxn; i++) {
    pw1[i] = (pw1[i-1] * base1) % mod;
    pw2[i] = (pw2[i-1] * base2) % mod;
  } }
struct hash {
  ll h1[maxn], h2[maxn];
  void init() {
    h1[0] = h2[0] = 0;
    for(int i = 1; i <= len; i++) {
      h1[i] = (h1[i-1] * base1 + s[i]) %
mod;
      h2[i] = (h2[i-1] * base2 + s[i]) %
mod;
    }
  }
  inline ll hashval(int l, int r) {
    ll hsh1 = (h1[r] - h1[l-1] * pw1[r-
l+1]) % mod;
    if(hsh1 < 0) hsh1 += mod;
    ll hsh2 = (h2[r] - h2[l-1] * pw2[r-
l+1]) % mod;
    if(hsh2 < 0) hsh2 += mod;
    return (hsh1 << 32) | hsh2;
  }
  inline ll hashone(int l, int r) {
    ll hsh1 = (h1[r] - h1[l-1] * pw1[r-
l+1]) % mod;
    if(hsh1 < 0) hsh1 += mod;
    return hsh1;
  }
  inline ll hashtwo(int l, int r) {
    ll hsh2 = (h2[r] - h2[l-1] * pw2[r-
l+1]) % mod;
    if(hsh2 < 0) hsh2 += mod;
    return hsh2;
  } } h;
```

KMP:

```
ll f[maxn];
vector <ll> ans;
void failure(string &y){
   f[0]=0;
   ll i=1;
   ll len=0;
   ll m=y.size();
   while(i<m){
      if(y[len]==y[i]){
         len++;
         f[i]=len;
         i++;
      }
      else{
         if(len!=0){
            len=f[len-1];
         }
         else{
            f[i]=0;
            i++;
         } } } }
void kmp(string &x, string &y){
   failure(y);
   ll n=x.size();
   ll m=y.size();
   ll i=0;
   ll j=0;
   while(i<n){
      if(x[i]==y[j]){
         i++;
         j++;
         if(j==m){
            ans.pb(i-j);
            j=f[j-1];
         }
      }
      else{
         if(j!=0){
            j=f[j-1];
         }
         else{
            i++;
         } } } }
```

TRIE:

```
struct node{
   bool end;
   node* next[26];
   node(){
      end=false;
      for(ll i=0; i<26; i++){
         next[i]=NULL;
      }
   }
}*root;
void insert(string str, ll len){
   node* curr=root;
   for(ll i=0; i<len; i++){
      ll ch=str[i]-'a';
      if(curr->next[ch]==NULL){
         curr->next[ch]=new node();
      }
      curr=curr->next[ch];
   }
   curr->end=true; }
bool search(string str, ll len){
   node* curr=root;
   for(ll i=0; i<len; i++){
      ll ch=str[i]-'a';
      if(curr->next[ch]==NULL){
         return false;
      }
      curr=curr->next[ch];
   }
   return curr->end; }
void del(node* curr){
   for(ll i=0; i<26; i++){
      if(curr->next[i]) del(curr->next[i]);
   }
   delete(curr); }
```

SUFFIX ARRAY:

```
string s;
ll n;
ll ra[maxn], tempra[maxn];
ll sa[maxn], tempsa[maxn];
ll lcp[maxn], plcp[maxn];
ll phi[maxn];
ll cnt[maxn];
void countingsort(ll k){
   memset(cnt, 0, sizeof cnt);
   for(ll i=0; i<n; i++){
      if(i+k<n) cnt[ra[i+k]]++;
      else cnt[0]++;
   }
   ll mx=max(n, 300ll);
   ll sum=0;
   for(ll i=0; i<mx; i++){
      ll temp=cnt[i];
      cnt[i]=sum;
      sum+=temp;
   }
   for(ll i=0; i<n; i++){
      if(sa[i]+k<n)
tempsa[cnt[ra[sa[i]+k]]++]=sa[i];
      else tempsa[cnt[0]++]=sa[i];
   }
   for(ll i=0; i<n; i++) sa[i]=tempsa[i]; }
void suffixarray(){
   for(ll i=0; i<n; i++) ra[i]=s[i];
   for(ll i=0; i<n; i++) sa[i]=i;
   for(ll k=1; k<n; k<<=1){
      countingsort(k);
      countingsort(0);
      ll r=0;
      tempra[sa[0]]=r;
      for(ll i=1; i<n; i++){
         if(ra[sa[i]]==ra[sa[i-1]] &&
ra[sa[i]+k]==ra[sa[i-1]+k])
tempra[sa[i]]=r;
         else tempra[sa[i]]=++r;
      }
      for(ll i=0; i<n; i++) ra[i]=tempra[i];
      if(ra[sa[n-1]]==n-1) break;
   } }
void lcprefix(){
   phi[sa[0]]=-1;
   for(ll i=1; i<n; i++) phi[sa[i]]=sa[i-1];
   for(ll i=0, len=0; i<n; i++){
      if(phi[i]==-1){
         plcp[i]=0;
         continue;
      }
      while(s[i+len]==s[phi[i]+len])
len++;
      plcp[i]=len;
      len=max(len-1, 0ll);
   }
   for(ll i=0; i<n; i++) lcp[i]=plcp[sa[i]]; }
int main()
{
   ll t;
   t=1;
   while(t--){
      cin>>s;
      s+='$';
      n=s.size();
      suffixarray();
      lcprefix();
   }
   return 0; }
```

LI CIAO TREE

```
/* * Can be used to solve Convex Hull
Trick problems
 * Adding line (y = mx+c) : O(logn)
 * Query : O(logn)
 * To find minimum, use f1(x) < f2(x),
min in update & query.
 * To find maximum, use f1(x) > f2(x),
max in update & query.
```

```
 * The line of tree[node] in range [lo,
hi] represents
   that this line gives the best result for
point [x, y] range, where x <= mid <= y.
Here, mid = (lo+hi) / 2. */
const ll sz = 1e5 + 10;
ll pnt[sz]; // 1 based indexing
// This array stores the points (in
ascending order) needed to be
queried.
struct Line{
   ll m, c;    // y = m*x + c
   inline ll f(ll x) {return m*x + c;}
} tree[4*sz];
bool exist[4*sz];
// This will track the nodes updated.
// Thus query'll be optimized by not
traversing
// the non-updated nodes.
void add(ll lo, ll hi, Line line, ll node){
   // To initialize the tree, add this line:
   // if(!exist[node]) tree[node] = {m,
c};
   // replace m, c with your desired
values.
   exist[node] = 1;
   if(lo == hi){
      if(line.f(pnt[lo]) <
tree[node].f(pnt[lo]))
         tree[node] = line;
      return;}
   ll mid = lo+hi >> 1;
   bool left = line.f(pnt[lo]) <
tree[node].f(pnt[lo]);
   bool m = line.f(pnt[mid]) <
tree[node].f(pnt[mid]);
   if(m) swap(tree[node], line);
   // if m == true, new line gives the
best answer in point [x,y] range,
   // where x <= mid <= y.
   // left != m means line intersection
between new line and
   // tree[node]'s line is occurring in
the left side of mid point
   if(left != m) add(lo, mid, line,
node<<1);
   else add(mid+1, hi, line,
node<<1|1);}
ll query(ll lo, ll hi, ll idx, ll node){
   if(lo == hi)
      return tree[node].f(pnt[idx]);
   ll mid = lo+hi >> 1, ret =
tree[node].f(pnt[idx]);
   // We are not traversing the non-
updated nodes. Thus query is
optimized.
   if(idx <= mid && exist[node<<1]) ret
= min(ret, query(lo, mid, idx,
node<<1));
   else if(idx > mid &&
exist[node<<1|1]) ret = min(ret,
query(mid+1, hi, idx, node<<1|1));
   return ret;}
/* * To add a line: Call add(1, n, {m, c},
1)
 * To query for a point x:
   Let, x is in i index of pnt array.
   Call query(1, n, i, 1)
 * [N.B: Query points are in the pnt
array from
   index 1 to index n] */
```

1. if n=(a^p)*(b^q)*(c^r), Sum of Divisors, S.O.D = (a^(p+1)-1)/(a-1)*(b^(q+1)-1)/(b-1)*(c^(r+1)-1)/(c-1)
2. Arithmetic Progression: n_{th} number=a+(n-1)d, sum=(n{2a+(n-1)d})/2
3. Geometric Progression: n_{th} number =ar^(n-1), sum=(a(r^n-1))/(r-1)
4. Catalan Numbers: 1, 1, 2, 5, 14, 42, 132….. $C_n = (2n)!/((n+1)!n!)$; n >= 0
5. Suppose, there are n unlabeled objects to be placed into k bins, ways= (n-1)C(k-1)
6. Statement of 5no. and empty bins are valid, ways= (n+k-1)C(k-1)
7. Sine Rule of a Triangle: a/sinA=b/sinB=c/sinC
8. Cosine Rule of a Triangle: cosA=(b^2+c^2-a^2)/2bc
9. Surface Area & Volumes:
   Sphere: SA =4πr^2, V=4/3 πr^3
   Cone: SA=πr^2+πrs, V=1/3 πr^2 h, [side,s= √(h^2+r^2)]
   Cylinder: SA=2πr^2+2πrh, V=πr^2 h
   Cuboid: SA=2(wh+lw+lh), V=lwh
   Trapezoid: Area=1/2 (b1+b2)h
10. Area of a Circle Sector=θ/360 πr^2 (in degree)
11. Number of permutations of n elements with k disjoint cycles
= Str1(n,k) = (n-1) * Str1(n-1,k) + Str1(n-1,k-1)
12. n! = Sum(Str1(n,k)) (for all 0 <= k <= n).
13. Ways to partition n labelled objects into k unlabeled subsets = Str2(n,k) = k * Str2(n-1,k) + Str2(n-1,k-1)
14. Parity of Str2(n,k) : ((n-k) & Floor((k-1)/2))==0)
15. Ways to partition n labelled objects into k unlabelled subsets, with each subset containing at least r elements:
SR(n,k) = k * SR(n-1,k) + C(n-1,r-1) * SR(n-r,k-1)
16. Number of ways to partition n labelled objects 1,2,3, ... n into k non-empty subsets so that for any integers i and j in a given subset |i-j| >= d: Str2(n-d+1, k-d+1), n >= k >= d
17. Total number of paths from point P(x1, y1) to point Q(x2, y2) where x2 >= x1 and y2 >= y1 :
Let x = x2 - x1 and y = y2 - y1. Then ans = C(x+y, x).
18. Total number of paths from point P(x1, y1) to point Q(x2, y2)
where x2 >= x1 and y2 >= y1 without crossing the line X = Y + c :
Let x = x2 - x1 and y = y2 - y1. Then ans = C(x+y, x) - C(x+y, x+c-1).
Special Case: x = n, y = n, c = 0, then ans = C(2n, n) - C(2n, n-1) [Catalan Number]
19. Catalan triangle: Total number of permutations having n X and k Y so that Count(X)-Count(Y)>=0 in any prefix (Non-negative Partial Sum): ans = C(n+k,k) - C(n+k, k-1)
20. Catalan trapezoid: Total number of permutation having n X and k Y so that Count(Y) - Count(X) < m in any prefix, then :
when 0 <= k < m, ans = C(n+k,k)
when m <= k <= n+m-1, ans = C(n+k,k) - C(n+k,k-m)
when k > n+m-1, ans = 0
21. Eulerian number of the first kind:
A1(n,k) is the number of permutations of 1 to n in which exactly k elements are greater than their previous element.
Then : A1(n,k) = (n-k) * A1(n-1,k-1) + (k+1) * A1(n-1,k).

22. Eulerian number of the second kind :
Number of permutations of the multiset {1,1,2,2,..,n,n} such that for each k, all the numbers appearing between the two occurrences of k are greater than k = (2n - 1)!
A2(n,m) is the number of such permutations with m ascents.
Then: A2(n,m) = (2n-m-1) * A2(n-1, m-1) + (m+1) * A2(n-1,m)
[ex: 332211: 0 ascent, 233211: 1 ascent, 112233: 2 ascents]
23. In 2-SAT: A, A' mustn't be in the same SCC.
(A | B)=TRUE is eqv to (A' => B)&(B' => A).
24. #pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx,avx2,fma") //gcc optimization
25. checker.sh: run "bash checker.sh" (For comparing outputs of two codes on the generated cases)

```
for((i = 1; ; ++i)); do
        echo $i
        ./gen.exe $i > int
        diff -w <(./a.exe < int) <(./brute.exe < int) || break
done
```

26. Pick's Theorem: A = I + (B/2) -1

A = Area of Polygon, B = Number of integral points on edges of polygon, I = Number of integral points strictly inside the polygon
27. Sum of i^2 from 1 to n = n*(n+1)*(2n+1)/6. // 1+4+16+...+n^2
28. Sum of i^3 from 1 to n = n^2 * (n+1)^2 / 4. // 1+8+27+...+n^3
29. (a+b)^n = Sum of nCk*(a^n)*(b^(n-k)) for k in range [0, n]

1. $\sum_{0 \le k \le n} \binom{n-k}{k} = Fib_{n+1}$
2. $\binom{n}{k} = \binom{n}{n-k}$
3. $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
4. $k\binom{n}{k} = n\binom{n-1}{k-1}$
5. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
6. $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
7. $\sum_{i \ge 0} \binom{n}{2i} = 2^{n-1}$
8. $\sum_{i \ge 0} \binom{n}{2i+1} = 2^{n-1}$
9. $\sum_{i=0}^{k} (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$
10. $\sum_{i=0}^{k} \binom{n+i}{i} = \sum_{i=0}^{k} \binom{n+i}{n} = \binom{n+k+1}{k}$
11. $1\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \ldots + n\binom{n}{n} = n2^{n-1}$
12. $1^2\binom{n}{1} + 2^2\binom{n}{2} + 3^2\binom{n}{3} + \ldots + n^2\binom{n}{n} = (n+n^2)2^{n-2}$

Regular polygon
A polygon is said to be a regular polygon if it has all the interior angles and the sides are of the same measure.
Irregular polygon
A polygon is said to be a regular polygon if it has all the interior angles and the sides have different values.
Concave polygon
A concave polygon is a polygon that has at least one interior angle greater than 180 degrees, i.e., a reflex angle.
Convex polygon
A convex polygon is a polygon that has all the interior angles of a polygon less than 180 degrees.
Equilateral Polygon
An equilateral polygon is a polygon whose all sides measure the same.
Equiangular Polygon
An equiangular polygon is a polygon whose all angles measure the same.
Equiangular Polygon
Below are some types of polygons based on the number of sides of a polygon

Polygon Formulae

The sum of interior angles of a polygon is given as,
Sum of interior angles of a polygon = $(n-2)\times180°$
where “n” is the number of sides of a regular polygon.

The formula for the measurement of each interior angle of a polygon is given as,
Each interior angle of a regular polygon = $(n-2)\times180°/n$
where “n” is the number of sides of a regular polygon.

The measure of the exterior angles of a regular n-sided polygon is,
Each exterior angle of a regular polygon = $360°/n$
where “n” is the number of sides of a regular polygon.

The formula for the perimeter of an n-sided regular polygon is given as,
Perimeter = $n \times s$
where “n” is the number of sides of a regular polygon and
“s” is the length of each side.
The formula for the area of an n-sided regular polygon is given as,
Area of a regular polygon = (number of sides × length of one side × apothem)/2.
Area = (Perimeter × apothem)/2
Area = $l/2\tan(180°/n)$
where “n” is the number of sides of a regular polygon and
“l” is the apothem length.