CODEBOOK RUET CSE 20

TEAM:NeverEndingHope

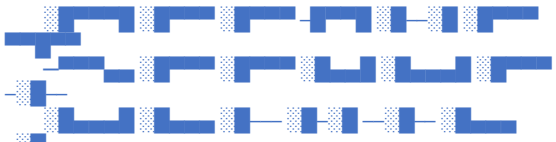
*Template:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb              push_back
#define ll              long long
#define ff first
#define ss second
#define SZ(a) (int)a.size()
#define UNIQUE(a) (a).erase(unique(all(a)),(a).end())
#define eb emplace_back
#define mp make_pair
///BIT MANIPULATION
#define Set(x, k) (x |= (1LL << k))
#define Unset(x, k) (x &= ~(1LL << k))
#define Check(x, k) (x & (1LL << k))
#define Toggle(x, k) (x ^ (1LL << k))
//LOOPS
#define scl(n)          scanf("%lld", &n)
#define fr(i,n)         for (ll i=0;i<n;i++)
#define fr1(i,n)        for(ll i=1;i<=n;i++)
#define Fo(i,k,n) for(i=k;k<n?i<n:i>n;k<n?i+=1:i-=1)
///PRINTING
#define deb(x) cout << #x << "=" << x << endl
#define deb2(x, y) cout << #x << "=" << x << "," << #y << "=" << y
<< endl
#define nn '\n'
#define pfl(x)          printf("%lld\n",x)
#define pcas(i)         printf("Case %lld: ",i)
#define Setpre(n) cout<<fixed<<setprecision(n)
#define itr(it, a) for(auto it = a.begin(); it != a.end(); it++)
#define debug           printf("I am here\n")
///SORTING AND FILLING
#define asort(a)        sort(a,a+n)
#define dsort(a)        sort(a,a+n,greater<int>())
#define vasort(v)       sort(v.begin(), v.end());
#define vdsort(v)       sort(v.begin(), v.end(),greater<ll>());
#define rev(x) reverse(all(x))
#define sortall(x) sort(all(x))
#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()
#define rev(x) reverse(all(x))
//CONSTANTS
#define md              10000007
#define PI 3.14159265358979323846264
///INLINE FUNCTIONS
inline ll GCD(ll a, ll b) { return b == 0 ? a : GCD(b, a % b); }
inline ll LCM(ll a, ll b) { return a * b / GCD(a, b); }
inline ll Ceil(ll p, ll q)  {return p < 0 ? p / q : p / q + !!(p % q);}
inline ll Floor(ll p, ll q) {return p > 0 ? p / q : p / q - !!(p % q);}
inline double logb(ll base,ll num){ return
(double)log(num)/(double)log(base);}
inline bool isPerfectSquare(long double x){ if (x >= 0) { long long sr
= sqrt(x);return (sr * sr == x); }return false; }
double euclidean_distance(ll x1,ll y1,ll x2,ll y2){double a=(x2-
x1)*(x2-x1);double b=(y2-y1)*(y2-y1);double
c=(double)sqrt(a+b);return c;}
int popcount(ll x){return __builtin_popcountll(x);};
int poplow(ll x){return __builtin_ctzll(x);};
int pophigh(ll x){return 63 - __builtin_clzll(x);};
typedef unsigned long long ull;
typedef pair<ll, ll>    pll;
typedef vector<ll>      vl;
typedef vector<pll>     vpll;
typedef vector<vl>      vvl;
template <typename T> using PQ = priority_queue<T>;
template <typename T> using QP =
priority_queue<T,vector<T>,greater<T>>;
template <typename T> using ordered_set = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename T,typename R> using ordered_map = tree<T,
R , less<T>, rb_tree_tag, tree_order_statistics_node_update>;
;
const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;
namespace io{
   template<typename First, typename Second> ostream&
operator << ( ostream &os, const pair<First, Second> &p ) { return
os << p.first << " " << p.second; }
   template<typename First, typename Second> ostream&
operator << ( ostream &os, const map<First, Second> &mp ) { for(
auto it : mp ) { os << it << endl;  } return os; }
   template<typename First> ostream& operator << ( ostream
&os, const vector<First> &v ) { bool space = false; for( First x : v ) {
if( space ) os << " "; space = true; os << x; } return os; }
   template<typename First> ostream& operator << ( ostream
&os, const set<First> &st ) { bool space = false; for( First x : st ) { if(
space ) os << " "; space = true; os << x; } return os; }
   template<typename First> ostream& operator << ( ostream
&os, const multiset<First> &st ) { bool space = false; for( First x : st
) { if( space ) os << " "; space = true; os << x; } return os; }
   template<typename First, typename Second> istream&
operator >> ( istream &is, pair<First, Second> &p ) { return is >>
p.first >> p.second; }
   template<typename First> istream& operator >> ( istream &is,
vector<First> &v ) { for( First &x : v ) { is >> x; } return is; }
   long long fastread(){ char c; long long d = 1, x = 0; do c =
getchar(); while( c == ' ' || c == '\n' ); if( c == '-' ) c = getchar(), d = -
1; while( isdigit( c ) ){ x = x * 10 + c - '0'; c = getchar(); } return d *
x; }
   static bool sep = false;
   using std::to_string;
   string to_string( bool x ){ return ( x ? "true" : "false" ); }
   string to_string( const string & s ){ return "\"" + s + "\""; }
   string to_string( const char * s ){ return "\"" + string( s ) + "\""; }
   string to_string ( const char & c ) { string s; s += c; return "\'" + s
+ "\'"; }

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

   template<typename First, typename Second> string to_string(
pair<First, Second> p ){ return "{" + to_string( p.first ) + ", " +
to_string( p.second ) + "}"; }
   template<typename Type> string to_string( vector<Type> v ) {
bool sep = false; string s = "["; for( Type x: v ){ if( sep ) s += ", "; sep
= true; s += to_string( x ); } s += "]"; return s; }
   template<typename Collection> string to_string( Collection
collection ) { bool sep = false; string s = "{"; for( auto x: collection
){ if( sep ) s += ", "; sep = true; s += to_string( x ); } s += "}"; return
s; }
   void print() { cerr << endl; sep = false; }
```

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



```
//================================================
===============*/
void setIO(){
   #ifndef ONLINE_JUDGE
   freopen("input.txt", "r", stdin);
   freopen("output.txt", "w", stdout);
   #endif // ONLINE_JUDGE
}
struct custom_hash {
   static uint64_t splitmix64(uint64_t x) {
      x += 0x9e3779b97f4a7c15;
      x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
      x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
      return x ^ (x >> 31);
   }
   size_t operator()(uint64_t x) const {
      static const uint64_t FIXED_RANDOM =
chrono::steady_clock::now().time_since_epoch().count();
      return splitmix64(x + FIXED_RANDOM);
   }
};

int main()
{
   fast;
    ll t;
   //setIO();
    //ll tno=1;;
    t=1;
   //cin>>t;

   while(t--){

   }


   return 0;
}
```

INPUT_OUTPUT:

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

Clock

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

DISPLAY FUNCTION:

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

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

SUBSET GENERATION:

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

Subset Sum:

```
bitset<N> can;
cin>>n>>k;
can[0]=true;
fo(i,n){
    int x;cin>>x;
    can|=(can<<x);
}
cout<<(can[k]?"YES\n":"NO\n");
```

PREFIX SUM

2D prefix sum:

```
ll arr[N][N];
ll pfsum[N][N];
void buildPS(){
    for(int i=1;i<N;i++){
        for(int j=1;j<N;j++){
            pfsum[i][j]=arr[i][j]+pfsum[i-
1][j]+pfsum[i][j-1]-pfsum[i-1][j-1];
        }
    }
}
ll getSum(ll a,ll b,ll c,ll d){
    return pfsum[c][d]-pfsum[a-1][d]-pfsum[c][b-
1]+pfsum[a-1][b-1];
}
```

SUBARRAY RELATED:

SUBARRAY SUM:

```
ll findSubarraySum(vector<ll> &vec, ll n, ll sum)
{
   ll m=0,cnt=0;
   map<ll,ll>mp;
   for(int i=0;i<n;++i)
   {
      m+=vec[i];
      if(m==sum)cnt++;
      if(mp.count(m-sum))
      {
         cnt+=mp[m-sum];
      }
      mp[m]++;
   }
   cout<<cnt<<endl;
}
```

TWO POINTERs:

Pair such that sum of the pair=k:

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

KADANE ALGORITHM:

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

SLIDING WINDOW:

Maximum element of subarray length K:

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

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

Ordered set:

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

SUBSEQUENCE RELATED:

Length of LIS(O(nlogn):

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

***SPARSE TABLE:

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

ll query(ll l, ll r) {
   ll k = 31 - __builtin_clz(r - l + 1);
```

```
    return GCD(table[l][k], table[r - (1 << k) + 1][k]);
}
```

SPARSE TABLE 2D:

```
const int LG = 10;

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

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

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

***Hashing:

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

```
        for (int i = 1; i <= n; i++){
            pref[i] = modmul(pref[i - 1], base) +
ar[i - 1] + 997;
            if (pref[i] >= mod) pref[i] -= mod;
        }

        for (int i = n; i >= 1; i--){
            suff[i] = modmul(suff[i + 1], base) +
ar[i - 1] + 997;
            if (suff[i] >= mod) suff[i] -= mod;
        }
    }
    PolyHash(const char* str)
        : PolyHash(vector<char> (str, str +
strlen(str))) {}
    uint64_t get_hash(int l, int r){
        int64_t h = pref[r + 1] -
modmul(base_pow[r - l + 1], pref[l]);
        return h < 0 ? h + mod : h;
    }
    uint64_t rev_hash(int l, int r){
        int64_t h = suff[l + 1] -
modmul(base_pow[r - l + 1], suff[r + 2]);
        return h < 0 ? h + mod : h;
    }
};
```

Custom hash for unordered map:

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

***SUFFIX ARRAY RELATED:

O(nlogn):

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

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

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

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

void buildLCP() {
  int i, L;
  Phi[SA[0]] = -1;
  // default value
  for (i = 1; i < n; i++)
    // compute Phi in O(n)
    Phi[SA[i]] = SA[i - 1];
  // remember which suffix is behind this suffix
  for (i = L = 0; i < n; i++) {
    // compute Permuted LCP in O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }
    // special case
```

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

***BIT MANIPULATION & Binary numbers related:

Is set:

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

Print binary:

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

Toggle bit:

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

Unset finction:

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

Set bit:

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

Binary Exponentiation:

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

Binary Multiply:

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

***Binary Indexed Tree/Fenwick Tree:

```
ll BITree[100009];

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

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

***BIT(2D):

```
struct BIT2D
{
  long long M[N][N][2], A[N][N][2];
  BIT2D()
  {
    memset(M, 0, sizeof M);
    memset(A, 0, sizeof A);
  }
  void upd2(long long t[N][N][2], int x, int y, long long mul, long
long add)
  {
    for (int i = x; i < N; i += i & -i)
    {
      for (int j = y; j < N; j += j & -j)
      {
        t[i][j][0] += mul;
        t[i][j][1] += add;
      }
    }
  }
  void upd1(int x, int y1, int y2, long long mul, long long add)
  {
    upd2(M, x, y1, mul, -mul * (y1 - 1));
    upd2(M, x, y2, -mul, mul * y2);
    upd2(A, x, y1, add, -add * (y1 - 1));
    upd2(A, x, y2, -add, add * y2);
  }
```

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

Search(BIT):

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

int bit[N]; // BIT array

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

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

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

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

PRIME NO RELATED:

Sieve optimum:

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

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

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

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

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

MILLER ROBIN Primality Test (for n>10^9)

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

```
    }
    if(mod!=p-1 && temp%2==0){
    return false;
    }
  }
  return true;
}
```

EULER TOTIENT:

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

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

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

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

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

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

void computeTotient(){
  etf[1] = 1;
  for(ll i = 2; i <= sz; i++){
    if(!prime[i]){
      etf[i] = i - 1;
      for(ll j = 1; j * i <= sz; j++)
        if(!prime[j*i])prime[j*i] = i;
    }
    else{
      etf[i] = etf[prime[i]] * etf[ i/prime[i] ];
      ll g = 1;
      if(i % (prime[i]*prime[i]) == 0) g = prime[i];
      etf[i] *= g;
      etf[i] /= etf[g];
    }
  }
}
```

EULER TOTIENT (1-n) 2:

```
int phi[N];

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

Extended gcd:

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

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

MODULAR ARITHMATICS

POWER MOD:

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

NCR MOD:

```
ll FM[N];
int is_initialized = 0;
ll factorialMod(ll n, ll x){
  if (!is_initialized){
    FM[0] = 1 % x;
    for (int i = 1; i < N; i++)
      FM[i] = (FM[i - 1] * i) % x;
    is_initialized = 1;
  }
  return FM[n];
}
ll powerMod(ll x, ll y, ll p){
  ll res = 1 % p;
  x = x % p;
  while (y > 0){
    if (y & 1) res = (res * x) % p;
```

```
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}
ll inverseMod(ll a, ll x){
    return powerMod(a, x - 2, x);
}
ll nCrMod(ll n, ll r, ll x){
    if (r == 0) return 1;
    if (r > n) return 0;
    ll res = factorialMod(n, x);
    ll fr = factorialMod(r, x);
    ll zr = factorialMod(n - r, x);
    res = (res * inverseMod((fr * zr) % x, x)) % x;
    return res;
}
```

GCD:

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

POWER:

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

JOSEPHUS:

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

MIN_25_Sieve:

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

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

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

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

    vector<unsigned char> pre(prod, 0xFF);
    for (size_t pi = 0; pi < pbeg; ++pi)
    {
        auto pp = sprimes[pi];
        const int p = pp.p;
        for (int t = 0; t < 8; ++t)
        {
            const unsigned char m = ~(1 << t);
            for (int i = pp.pos[t]; i < prod; i += p)
                pre[i] &= m;
        }
    }

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

    for (int beg = 0; beg < M; beg += block_size, pblock -=
block_size)
    {
        int end = min(M, beg + block_size);
        for (int i = beg; i < end; i += prod)
        {
            copy(pre.begin(), pre.end(), pblock + i);
        }
        if (beg == 0)
```

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

FLOOR sum of n/1+n/2+…:

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

****Combinatorics***

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

****GEOMETRY****

Check if there exists a point that all ranges cover:

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

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

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

Check if there exists a point that all ranges cover:

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

// Function that returns true if any k
// segments overlap at any point
bool kOverlap(vector<pair<ll, ll> > pairs, ll k)
{
```

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

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

Count the number of rectangle with given points:

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

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

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

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

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

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

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

Maximum possible rectangles:

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

Minimum number of straight lines to connect all points:

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

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

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

    int numoflines = 1;

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

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

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

*****STRINGS***

Pattern matching:KMP

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

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

Lexicographically compare of two strings:

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

Custom comparator sort numeric strings

```
bool myCmp(string s1, string s2)

{

    if (s1.size() == s2.size()) {

        return s1 < s2;

    }

     else {

        return s1.size() < s2.size();

    }

}


// in main

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

Z-FUNCTION:

```
// An element Z[i] of Z array stores length of the longest substring
// starting from str[i] which is also a prefix of str[0..n-1].
// The first entry of Z array is meaning less as complete string is always
prefix of itself.
// Here Z[0]=0.
vector<int> z_function(string s) {
 int n = (int) s.length();
 vector<int> z(n);
 for (int i = 1, l = 0, r = 0; i < n; ++i) {
  if (i <= r)
   z[i] = min (r - i + 1, z[i - l]);
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
   ++z[i];
  if (i + z[i] - 1 > r)
   l = i, r = i + z[i] - 1;
 }
 return z;
}
```

COUNT unique substrings:

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

    const int p = 31;
    const int m = 1e9 + 9;
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow[i]) % m;
    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * p_pow[n-i-1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```

String operations:

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

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

```
    ans=cut_leading_zero(ans);
    return ans;
}

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

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

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

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

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

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

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

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

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

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

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

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

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

string cut_leading_zero(string a){
```

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


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

Binary to Roman:

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

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

BINARY Search:

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

TERNARY SEARCH(Int):

```
long double func(ll pos){

}

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

Comparators:

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

SORT check:

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

Count sort:

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

TOPOLOGICAL SORT:

```
vector<ll> topoSort(ll n){
    queue<ll> q;
    vector<ll> indegree(n,0);
    for(int i=0;i<n;i++){
```

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

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

Reset function:

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

DFS:

On graph:

```
bool vis[N];
int subtree_sum[N];
void dfs(ll vertex){
  vis[vertex]=true;
  for(ll child: g[vertex]){
    if(vis[child]) continue;
    dfs(child);
    subtree_sum[vertex]+=subtree_sum[child];
  }
}
```

DFS:

On grid:

```
ll n,m,t=0;
vector<string> g;
vpll Move={ {1,0},{-1,0},{0,1},{0,-1} };
bool vis[N][N];
bool isValid(ll x,ll y){
  return
(x>=0&&y>=0&&x<n&&y<m&&vis[x][y]==0&&g[x][y]=='.');
}
void dfs(pll vertex){
  vis[vertex.first][vertex.second]=true;
  for(pll &child: Move){
    ll x=child.first+vertex.first;
    ll y=child.second+vertex.second;
    if(!isValid(x,y)) continue;
    dfs({x,y});
  }
}
```

DIAMETER OF A WEIGHTED TREE:

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

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

LCA:

```
vector<int> path(int vertex){
  vector<int> ans;
  while(vertex!=-1){
    ans.push_back(vertex);
    vertex=parent[vertex];
  }
  reverse(ans.begin(),ans.end());
  return ans;
}
int LCA(int n){
  int i;
  dfs(1);
  int x,y;
  cin>>x>>y;
  vector<int> path_x=path(x);
  vector<int> path_y=path(y);
  int mn_ln=min(path_x.size(),path_y.size());

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

BFS:

```
bool vis[N];
ll level[N];
void bfs(ll source){
  queue<ll> q;
```

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

BFS on grid:

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

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

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

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

Multisource bfs:

```
const ll maxN=1e3+10;//for graph
const ll INF=1e9+10;
#define M 10000
//when edges dont have same weight...0 and 1
weights..use 0-1 bfs
 ll n,m;
 ll val[maxN][maxN];
 ll vis[maxN][maxN];
 ll lev[maxN][maxN];
 void reset(){
    for(ll i=0;i<n;i++){
        for(ll j=0;j<m;j++){
            vis[i][j]=0;
            lev[i][j]=INF;
        }
    }
}
bool isvalid(ll i,ll j){
   return i>=0 &&  j>=0 && i< n && j<m;

}
 vector<pair<ll,ll> >movements={
{0,1},{0,-1},{1,0},{-1,0},
{1,1},{1,-1},{-1,1},{-1,-1}

};
ll bfs(){
    ll mx=0;
    for(ll i=0;i<n;i++){
        for(ll j=0;j<m;j++){
          mx=max(mx,val[i][j]);
        }
    }
    queue< pair<ll,ll> >q;

    for(ll i=0;i<n;i++){
        for(ll j=0;j<m;j++){
        if(mx==val[i][j]){
            q.push({i,j});
            lev[i][j]=0;
            vis[i][j]=1;
        }
        }
    }
    ll ans=0;
    while(!q.empty()){
        auto v=q.front();
        ll v_x=v.first;
        ll v_y=v.second;
        q.pop();
        for(auto movement : movements){
            ll child_x=movement.first+v_x;
            ll child_y=movement.second+v_y;
            if(!isvalid(child_x,child_y))
continue;
            if(vis[child_x][child_y]) continue;
```

```
            q.push({child_x,child_y});

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

    }
    return ans;
 }
```

0-1 Bfs:

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



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

ll bfs(){

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

Dijkstra(+find parent):

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

BELLMAN_FORD:

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

   for(ll i=0; i<n; i++){
      x=-1;
      for(ll node=1; node<=n; node++){
         //if(dist[node]==INF) continue;
         for(pair<ll,ll> a : g[node]){
            if(dist[a.first]>dist[node]+a.second){
               dist[a.first]=dist[node]+a.second;
               par[a.first]=node;
               x=a.first;
            }
         }
      }
      //if(!x) break;
   }
   if(x==-1){
      cout<<"NO"<<endl;
   }
   else{
```

```
    //x can be on any cycle or reachable from some cycle
    vl path;
    for (ll i=0; i<n; i++) x = par[x];

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

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

}
```

FLOYD WARSHALL:

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

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

Disjoint set union:

```
int par[N];
int sz[N];
multiset<int> sizes;
void make(int v){
  par[v]=v;
  sz[v]=1;
  sizes.insert(1);
}

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

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

DSU ON TREES:

```
#define maxn 100009

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

void szdfs(ll u, ll p)

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

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

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

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

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

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

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

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

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

```
const double EPS = 1e-9;
const int N = 2e5+10;
ll T=0;
ll tre[3*N];
ll lazy[3*N];
ll merge(ll x,ll y){
```

```
    return x+y;
}
void buildSegTree(vector<ll>& arr, ll treeIndex, ll lo, ll hi){
    if (lo == hi) {              // leaf node. store value in node.
        tre[treeIndex] = arr[lo];
        return;
    }
    ll mid = lo + (hi - lo) / 2;   // recurse deeper for children.
    buildSegTree(arr, 2 * treeIndex + 1, lo, mid);
    buildSegTree(arr, 2 * treeIndex + 2, mid + 1, hi);
    // merge build results
    tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex
+ 2]);
}
// call this method as buildSegTree(arr, 0, 0, n-1);
// Here arr[] is input array and n is its size.
ll querySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j){
    // query for arr[i..j]
    if (lo > j || hi < i)            // segment completely outside range
        return 0;                   // represents a null node
    if (i <= lo && j >= hi)          // segment completely inside range
        return tre[treeIndex];

    ll mid = lo + (hi - lo) / 2;      // partial overlap of current segment
and queried range. Recurse deeper.
    if (i > mid)
        return querySegTree(2 * treeIndex + 2, mid + 1, hi, i, j);
    else if (j <= mid)
        return querySegTree(2 * treeIndex + 1, lo, mid, i, j);
    ll leftQuery = querySegTree(2 * treeIndex + 1, lo, mid, i, mid);
    ll rightQuery = querySegTree(2 * treeIndex + 2, mid + 1, hi, mid
+ 1, j);
    // merge query results
    return merge(leftQuery, rightQuery);
}
// call this method as querySegTree(0, 0, n-1, i, j);
// Here [i,j] is the range/interval you are querying.
// This method relies on "null" nodes being equivalent to storing
zero.
void updateValSegTree(ll treeIndex, ll lo, ll hi, ll arrIndex, ll val)
{
    if (lo == hi) {              // leaf node. update element.
        tre[treeIndex] = val;
        return;
    }
    ll mid = lo + (hi - lo) / 2;   // recurse deeper for appropriate child
    if (arrIndex > mid)
        updateValSegTree(2 * treeIndex + 2, mid + 1, hi, arrIndex,
val);
    else if (arrIndex <= mid)
        updateValSegTree(2 * treeIndex + 1, lo, mid, arrIndex, val);
    // merge updates
    tre[treeIndex] = merge(tre[2 * treeIndex + 1], tre[2 * treeIndex
+ 2]);
}
// call this method as updateValSegTree(0, 0, n-1, i, val);
// Here you want to update the value at index i with value val.

void updateLazySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j, ll val){
    if (lazy[treeIndex] != 0) {                    // this node is lazy
        tre[treeIndex] += (hi - lo + 1) * lazy[treeIndex]; // normalize
current node by removing laziness
        if (lo != hi) {                            // update lazy[] for children
nodes
            lazy[2 * treeIndex + 1] += lazy[treeIndex];
            lazy[2 * treeIndex + 2] += lazy[treeIndex];
        }
        lazy[treeIndex] = 0;                       // current node
processed. No longer lazy
    }
    if (lo > hi || lo > j || hi < i)
        return;                                    // out of range. escape.
    if (i <= lo && hi <= j) {                      // segment is fully within
update range
        tre[treeIndex] += (hi - lo + 1) * val;          // update segment
        if (lo != hi) {                            // update lazy[] for children
            lazy[2 * treeIndex + 1] += val;
            lazy[2 * treeIndex + 2] += val;
        }
        return;
    }
    ll mid = lo + (hi - lo) / 2;                   // recurse deeper for
appropriate child
    updateLazySegTree(2 * treeIndex + 1, lo, mid, i, j, val);
    updateLazySegTree(2 * treeIndex + 2, mid + 1, hi, i, j, val);
    // merge updates
    tre[treeIndex] = tre[2 * treeIndex + 1] + tre[2 * treeIndex + 2];
}
// call this method as updateLazySegTree(0, 0, n-1, i, j, val);
// Here you want to update the range [i, j] with value val.
ll queryLazySegTree(ll treeIndex, ll lo, ll hi, ll i, ll j){
    // query for arr[i..j]
    if (lo > j || hi < i)                          // segment completely
outside range
        return 0;                                  // represents a null node
    if (lazy[treeIndex] != 0) {                    // this node is lazy
        tre[treeIndex] += (hi - lo + 1) * lazy[treeIndex]; // normalize
current node by removing laziness
        if (lo != hi) {                            // update lazy[] for children
nodes
            lazy[2 * treeIndex + 1] += lazy[treeIndex];
            lazy[2 * treeIndex + 2] += lazy[treeIndex];
        }

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

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

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

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

SEGMENTED SIEVE:

```
vector<int> smallest_factor;
```

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

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

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

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

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

TRIE TO NUMBER OF DISTICT SUBSTR:

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

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

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

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

```
        root->insertSuffix(s.substr(i));
    }

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

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

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

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

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

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

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

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

    // Return count of nodes in Trie of Suffixes
    return sTrie.countNodesInTrie();
```

DYNAMIC PROGRAMMING***

0-1 knapsack:

```
int knapSack(int W, int wt[], int val[], int n)
{
    // making and initializing dp array
    int dp[W + 1];
    memset(dp, 0, sizeof(dp));

    for (int i = 1; i < n + 1; i++) {
        for (int w = W; w >= 0; w--) {

            if (wt[i - 1] <= w)
                // finding the maximum value
                dp[w] = max(dp[w],
                            dp[w - wt[i - 1]] + val[i - 1]);
        }
    }
    return dp[W]; // returning the maximum value of knapsack
}
```

COIN change:

```
ll dp[][];
vl coins;
ll func(ll ind,ll amount){
    if(amount==0) return 1;
    if(ind<0) return 0;
    if(dp[ind][amount] !=-1) return dp[ind][amount];
    ll ways=0;
    for(ll
coin_amount=0;coin_amount<=amount;coin_amount+=coins[ind]
){
        ways+=func(ind-1,amount-coin_amount);
    }
    return dp[ind][amount]=ways;
}
ll coinChange(ll amount){
    memset(dp,-1,sizeof(dp));
    return func(coins.size()-1,amount);
}
```

Iterative solution of coin change:

```
ll t=0,n;
ll dp[10005];
ll coins[105];
ll func(ll amount){
    dp[0]=1;
    ll i,j;
    fo(i,n){
        for(j=coins[i];j<=amount;j++){
            dp[j]=(dp[j]+dp[j-coins[i]])%M;
        }
    }
    return dp[amount];
}
```

LCS:

```
string s1,s2;
int dp[1005][1005];
int lcs(int i,int j){
    if(i<0 || j<0) return 0;
    if(dp[i][j]!=-1) return dp[i][j];
    //remove 1 char from s1
    int ans=lcs(i-1,j);
    //remove 1 char from s2
    ans=max(ans,lcs(i,j-1));
    //remove 1 char from s1 and s2
    ans=max(ans,lcs(i-1,j-1))+(s1[i]==s2[j]);
    return dp[i][j]=ans;
}
```

Longest Increasing Subsequence:

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

int lis(int n){
    if(dp[n]!=-1) return dp[n];
```

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

DP right capital small partition:

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

DP Rod cutting:

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

DP SUBSET SUM:

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

DP SUBSET Count:

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

Find minimum number operations to convert str1 to str2:

```
int editDistDP(string str1, string str2, int m, int n)
{
    int dp[m + 1][n + 1];
        for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0)
                dp[i][j] = j; // Min. operations = j
            else if (j == 0)
                dp[i][j] = i; // Min. operations = i
            else if (str1[i - 1] == str2[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            else dp[i][j]= 1 + min(dp[i][j - 1], dp[i - 1][j ], dp[i - 1][j - 1]);
        }
    }
    return dp[m][n];
}
```

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

Paranthesis:

```
unordered_map<char,int> symbols ={{'(',-1},{'{',-2},{'[',-3},{')',1},{'}',2},{']',3}};
bool isBalanced(string s){
    stack<char> st;
    for(char bracket : s){
        if(symbols[bracket]<0){
            st.push(bracket);
        }else{
            if(st.empty()) return 0;
            char top=st.top();
            st.pop();
            if(symbols[top] + symbols[bracket] !=0){
                return 0;
            }
        }
    }
    if(st.empty()) return 1;
    return 0;
}
```

generate balanced paranthesis:

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

```
}
```

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

ALPHA BETA PRUNING:

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

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

   if (maximizingPlayer)
   {
      int best = MIN;

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

         int val = minimax(depth + 1, nodeIndex * 2 + i,false, values,
alpha, beta);
         best = max(best, val);
         alpha = max(alpha, best);

         // Alpha Beta Pruning
         if (beta <= alpha)
            break;
      }
      return best;
   }
   else
   {
      int best = MAX;

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

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

CHESS MOVES:

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

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

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

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

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

***ALL MATHEMATICAL FORMULAE:
(SS below are taken from Shahjalal Shohag vai's blog...
https://blog.shahjalalshohag.com/equation-list/
)

The Equation List

Combinatorics

General

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. $1^2\binom{n}{1}+2^2\binom{n}{2}+3^2\binom{n}{3}+\ldots+n^2\binom{n}{n}=(n+n^2)2^{n-2}$
13. Vandermonde's Identify: $\sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}=\binom{m+n}{r}$
14. Hockey-Stick Identify: $n,r\in N, n>r, \sum_{i=r}^{n}\binom{i}{r}=\binom{n+1}{r+1}$
15. $\sum_{i=0}^{k}\binom{k}{i}^2=\binom{2k}{k}$
16. $\sum_{k=0}^{n}\binom{n}{k}\binom{n}{n-k}=\binom{2n}{n}$
17. $\sum_{k=q}^{n}\binom{n}{k}\binom{k}{q}=2^{n-q}\binom{n}{q}$
18. $\sum_{i=0}^{n}k^i\binom{n}{i}=(k+1)^n$
19. $\sum_{i=0}^{n}\binom{2n}{i}=2^{2n-1}+\frac{1}{2}\binom{2n}{n}$
20. $\sum_{i=1}^{n}\binom{n}{i}\binom{n-1}{i-1}=\binom{2n-1}{n-1}$
21. $\sum_{i=0}^{n}\binom{2n}{i}^2=\frac{1}{2}\left(\binom{4n}{2n}+\binom{2n}{n}^2\right)$
22. **Highest Power of 2 that divides $^{2n}C_n$:** Let z be the number of 1s in the binary representation. Then the number of odd terms will be 2^z.Let it form a sequence. The n-th value in the sequence (starting from $n=0$) gives the highest power of 2 that divides $^{2n}C_n$.

23. **Pascal Triangle**
    - In a row p where p is a prime number, all the terms in that row except the 1s are multiples of p.
    - Parity: To count odd terms in row n, convert n to binary. Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x.
    - Every entry in row $2^n - 1, n \geq 0$, is odd.
24. An integer $n \geq 2$ is prime if and only if all the intermediate binomial coefficients $\binom{n}{1}, \binom{n}{2}, \ldots, \binom{n}{n-1}$ are divisible by n.
25. **Kummer's Theorem**: For given integers $n \geq m \geq 0$ and a prime number p, the largest power of p dividing $\binom{n}{m}$ is equal to the number of carries when m is added to $n-m$ in base p. For implementation take inspiration from lucas theorem.
26. Number of different binary sequences of length n such that no two 0's are adjacent$=Fib_{n+1}$
27. **Combination with repetition**: Let's say we choose k elements from an n-element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is: $\binom{n+k-1}{k}$
28. Number of ways to divide n persons in $\frac{n}{k}$ equal groups i.e. each having size k is

$$\frac{n!}{k!^{\frac{n}{k}}(\frac{n}{k})!} = \prod_{n \geq k}^{n-k} \binom{n-1}{k-1}$$

29. The number non-negative solution of the equation: $x_1 + x_2 + x_3 + \ldots + x_k = n$ is $\binom{n+k-1}{n}$
30. Number of ways to choose n ids from 1 to b such that every id has distance at least k $= \binom{b-(n-1)(k-1)}{n}$
31. $\sum_{i=1,3,5,\ldots}^{i \leq n} \binom{n}{i} a^{n-i} b^i = \frac{1}{2}((a+b)^n - (a-b)^n)$
32. $\sum_{i=0}^{n} \frac{\binom{k}{i}}{\binom{n}{i}} = \frac{\binom{n+1}{n-k+1}}{\binom{n}{k}}$
33. **Derangement**: a permutation of the elements of a set, such that no element appears in its original position. Let $d(n)$ be the number of derangements of the identity permutation fo size n.

33. **Derangement**: a permutation of the elements of a set, such that no element appears in its original position. Let $d(n)$ be the number of derangements of the identity permutation fo size n.

$$d(n) = (n-1) \cdot (d(n-1) + d(n-2)) \text{ where } d(0) = 1, d(1) = 0$$

34. **Involutions:** permutations such that $p^2 =$ identity permutation. $a_0 = a_1 = 1$ and $a_n = a_{n-1} + (n-1)a_{n-2}$ for $n > 1$.
35. Let $T(n,k)$ be the number of permutations of size n for which all cycles have length $\leq k$.

$$T(n,k) = \begin{cases} n! & ; n \leq k \\ n \cdot T(n-1,k) - F(n-1,k) \cdot T(n-k-1,k) & ; n > k \end{cases}$$

Here $F(n,k) = n \cdot (n-1) \cdot \ldots \cdot (n-k+1)$

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

$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} (mod\ p)$, where,
$m = m_k p^k + m_{k-1} p^{k-1} + \ldots + m_1 p + m_0$,
and
$n = n_k p^k + n_{k-1} p^{k-1} + \ldots + n_1 p + n_0$
are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$,when $m < n$.

37. $\sum_{i=0}^{n} \binom{n}{i} \cdot i^k$
$= \sum_{i=0}^{n} \binom{n}{i} \cdot \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot i^{\underline{j}}$
$= \sum_{i=0}^{n} \binom{n}{i} \cdot \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot j! \binom{n}{i}$
$= \sum_{i=0}^{n} \frac{n!}{(n-i)!} \cdot \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot \frac{1}{(i-j)!}$
$= \sum_{i=0}^{n} \sum_{j=0}^{k} \frac{n!}{(n-i)!} \cdot \left\{ {k \atop j} \right\} \cdot \frac{1}{(i-j)!}$
$= n! \sum_{i=0}^{n} \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot \frac{1}{(n-i)!} \cdot \frac{1}{(i-j)!}$
$= n! \sum_{i=0}^{n} \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot \binom{n-j}{n-i} \cdot \frac{1}{(n-j)!}$
$= n! \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot \frac{1}{(n-j)!} \sum_{i=0}^{n} \binom{n-j}{n-i}$
$= \sum_{j=0}^{k} \left\{ {k \atop j} \right\} \cdot n^{\underline{j}} \cdot 2^{n-j}$

Here $n^{\underline{j}} = P(n,j) = \frac{n!}{(n-j)!}$ and $\left\{ {k \atop j} \right\}$ is stirling number of the second kind.

So, instead of $O(n)$, now you can calculate the original equation in $O(k^2)$ or even in $O(k \log^2 n)$ using NTT.

38. $\sum_{i=0}^{n-1} \binom{i}{j} x^i = x^j (1-x)^{-j-1} \left(1 - x^n \sum_{i=0}^{j} \binom{n}{i} x^{j-i} (1-x)^i\right)$
39. $x_0, x_1, x_2, x_3, \ldots, x_n$
$x_0 + x_1, x_1 + x_2, x_2 + x_3, \ldots x_n$
...
If we continuously do this n times then the polynomial of the first column of the n-th row will be

39. $x_0, x_1, x_2, x_3, \ldots, x_n$
$x_0 + x_1, x_1 + x_2, x_2 + x_3, \ldots x_n$
...
If we continuously do this n times then the polynomial of the first column of the n-th row will be

$$p(n) = \sum_{k=0}^{n} \binom{n}{k} \cdot x(k)$$

40. If $P(n) = \sum_{k=0}^{n} \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} \cdot P(k)$$

41. If $P(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot Q(k)$, then,

$$Q(n) = \sum_{k=0}^{n} (-1)^k \binom{n}{k} \cdot P(k)$$

Catalan Numbers

42. $C_n = \frac{1}{n+1} \binom{2n}{n}$
43. $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
44. Number of correct bracket sequence consisting of n opening and n closing brackets.
45. The number of ways to completely parenthesize $n+1$ factors.
46. The number of triangulations of a convex polygon with $n+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
47. The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
48. The number of monotonic lattice paths from point $(0,0)$ to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0,0)$ to (n,n)).
49. The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
50. Number of permutations of $1, \ldots, n$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing sub-sequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321.\displaystyle For $n = 4$, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321.
51. **Balanced Parentheses count with prefix:** The count of balanced parentheses sequences consisting of $n + k$ pairs of parentheses where the first k symbols are open brackets. Let the number be $C_n^{(k)}$, then

$$C_n^{(k)} = \frac{k+1}{n+k+1} \binom{2n+k}{n}$$

Narayana numbers

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

Stirling numbers of the first kind

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

GCD and LCM

154. $\gcd(a,0) = a$

155. $\gcd(a,b) = \gcd(b, a \mod b)$

156. Every common divisor of a and b is a divisor of $\gcd(a,b)$.

157. If m is any integer, then $\gcd(a+mb,b) = \gcd(a,b)$

158. The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1,b) \cdot \gcd(a_2,b)$.

159. $\gcd(a,b) \cdot \operatorname{lcm}(a,b) = |ab|$

160. $\gcd(a,\operatorname{lcm}(b,c)) = \operatorname{lcm}(\gcd(a,b),\gcd(a,c))$.

161. $\operatorname{lcm}(a,\gcd(b,c)) = \gcd(\operatorname{lcm}(a,b),\operatorname{lcm}(a,c))$.

162. For non-negative integers a and b, where a and b are not both zero, $\gcd(n^a-1, n^b-1) = n^{\gcd(a,b)}-1$

163. $\gcd(a,b) = \sum_{k|a \text{ and } k|b} \phi(k)$

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

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

166. $\sum_{k=1}^{n} x^{\gcd(k,n)} = \sum_{d|n} x^d \cdot \phi\left(\frac{n}{d}\right)$

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

168. $\sum_{k=1}^{n} \frac{k}{\gcd(k,n)} = \frac{n}{2} \cdot \sum_{d|n} \frac{1}{d} \cdot \phi\left(\frac{n}{d}\right) = \frac{n}{2} \cdot \frac{1}{n} \cdot \sum_{d|n} d \cdot \phi(d)$

169. $\sum_{k=1}^{n} \frac{n}{\gcd(k,n)} = 2 * \sum_{k=1}^{n} \frac{k}{\gcd(k,n)} - 1$, for $n > 1$

170. $\sum_{i=1}^{n}\sum_{j=1}^{n} |\gcd(i,j) = 1| = \sum_{d=1}^{n} \mu(d) \left\lfloor \frac{n}{d} \right\rfloor^2$

171. $\sum_{i=1}^{n}\sum_{j=1}^{n} \gcd(i,j) = \sum_{d=1}^{n} \phi(d) \left\lfloor \frac{n}{d} \right\rfloor^2$

172. $\sum_{i=1}^{n}\sum_{j=1}^{n} i \cdot j |\gcd(i,j) = 1| = \sum_{i=1}^{n} \phi(i) i^2$

173. $F(n) = \sum_{i=1}^{n}\sum_{j=1}^{n} \operatorname{lcm}(i,j) = \sum_{i=1}^{n} \left(\frac{\left(1 + \left\lfloor \frac{n}{i} \right\rfloor\right)\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{2} \right)^2 \sum_{d|i} \mu(d) i d$

174. $\gcd(\operatorname{lcm}(a,b),\operatorname{lcm}(b,c),\operatorname{lcm}(a,c)) = \operatorname{lcm}(\gcd(a,b),\gcd(b,c),\gcd(a,c))$

175. $\gcd(A_L, A_{L+1}, \ldots, A_R) = \gcd(A_L, A_{L+1} - A_L, \ldots, A_R - A_{R-1})$.[7]

176. Given n, If $SUM = LCM(1,n) + LCM(2,n) + \ldots + LCM(n,n)$
then $SUM = \frac{n}{2}(\sum_{d|n} (\phi(d) \times d) + 1$

Legendre Symbol

177. Let p be an odd prime number. An integer a is a quadratic residue modulo p if it is congruent to a perfect square modulo p and is a quadratic nonresidue r function of a and p defined as

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \text{ and } a \not\equiv 0 \pmod p, \\ -1 & \text{if } a \text{ is a non-quadaratic residue modulo } p, \\ 0 & \text{if } a \equiv 0 \pmod p \end{cases}$$

178. Legenres's original definition was by means of explicit formula

$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p$ *and* $\left(\frac{a}{p}\right) \in -1, 0, 1.$

179. The Legendre symbol is periodic in its first (or top) argument: if $a \equiv b \pmod p$, then

$\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

180. The Legendre symbol is a completely multiplicative function of its top argument:

$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$

181. The Fibonacci numbers $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ are defined by the recurrence $F_1 = F_2 = 1, F_{n+1} = F_n + F_{n-1}$. If p is a prime number then

$F_{p-\left(\frac{p}{5}\right)} \equiv 0 \pmod p$, $F_p \equiv \left(\frac{p}{5}\right) \pmod p$.

For example,

$\left(\frac{2}{5}\right) = -1$, $F_3 = 2$, $F_3 = 1$,

$\left(\frac{3}{5}\right) = -1$, $F_4 = 3$, $F_3 = 2$,

$\left(\frac{5}{5}\right) = 0$, $F_5 = 5$,

$\left(\frac{7}{5}\right) = -1$, $F_8 = 21$, $F_7 = 13$,

$\left(\frac{11}{5}\right) = 1$, $F_{10} = 55$, $F_{11} = 89$,

182. Continuing from previous point, $\left(\frac{p}{5}\right)$ = infinite concatenation of the sequence $(1, -1, -1, 1, 0)$ from $p \geq 1$.

183. If $n = k^2$ is perfect square then $\left(\frac{n}{p}\right) = 1$ for every odd prime except $\left(\frac{n}{k}\right) = 0$ if k is an odd prime.

Miscellaneous

184. $a + b = a \oplus b + 2(a \& b)$.

185. $a + b = a \mid b + a \& b$

186. $a \oplus b = a \mid b - a \& b$

187. k_{th} bit is set in x iff $x \mod 2^{k-1} \geq 2^k$. It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.

188. k_{th} bit is set in x iff $x \mod 2^{k-1} - x \mod 2^k \neq 0$ (= 2^k to be exact). It comes handy when you need to look at the bits of the numbers which are pair sums or subset sums etc.

189. $n \mod 2^x = n \& (2^x - 1)$

190. $1 \oplus 2 \oplus 3 \oplus \cdots \oplus (4k-1) = 0$ for any $k \geq 0$

191. **Erdos Gallai Theorem:** The degree sequence of an undirected graph is the non-increasing sequence of its vertex degrees
A sequence of non-negative integers $d_1 \geq d_2 \geq \cdots \geq d_n$ can be represented as the degree sequence of finite simple graph on n vertices if and only if $d_1 + d_2 + \cdots + d_n$ is even and

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

holds for every k in $1 \leq k \leq n$.

Credits:
Author: Khandoker Sefayet Alam…
Thanks to: Nafis Ahmed for the code snippets