Basic

Bitwise Operation:

```
#include<bits/stdc++.h>
using namespace std;
// Checks if kth bit of x is set (1) or
not (0)
int check_kth_bit(int x, int k) {
  return (x >> k) & 1;
}
// Prints the positions of all set (1)
bits in binary representation of x
void print_on_bits(int x) {
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      cout << k << ' '; // prints the
position of the set bit
    }
  }
  cout << '\n';
}
// Returns the count of set (1) bits in
binary representation of x
int count_on_bits(int x) {
  int ans = 0;
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      ans++;
    }
  }
  return ans;
}
// Checks if x is even or odd
bool is_even(int x) {
  if (x & 1) {
    return false;
  }
  else {
    return true;
  }
}
// Sets the kth bit of x to 1 and returns
the result
int set_kth_bit(int x, int k) {
  return x | (1 << k);
}
// Sets the kth bit of x to 0 and returns
the result
int unset_kth_bit(int x, int k) {
  return x & (~(1 << k));
}
// Toggles the kth bit of x and returns
the result
int toggle_kth_bit(int x, int k) {
  return x ^ (1 << k);
}
// Checks if x is a power of 2
bool check_power_of_2(int x) {
  return count_on_bits(x) == 1;
}
int main() {
  // Bitwise AND (&)
  int and_result = 12 & 25;  // 12
(binary 1100) & 25 (binary 11001) = 8
(binary 1000)
  cout << "AND result: " << and_result <<
'\n'; // Output: 8
  // Bitwise OR (|)
  int or_result = 12 | 25;  // 12 (binary
1100) | 25 (binary 11001) = 29 (binary
11101)
  cout << "OR result: " << or_result <<
'\n'; // Output: 29
  // Bitwise XOR (^)
  int xor_result = 12 ^ 25;  // 12
(binary 1100) ^ 25 (binary 11001) = 21
(binary 10101)
  cout << "XOR result: " << xor_result <<
'\n'; // Output: 21
  // Bitwise NOT (~)
  int not_result = ~12;
  cout << "NOT result: " << not_result <<
'\n'; // Output: -13
  // Left shift (<<)
  int left_shift_result = 3 << 2;  // 3
(binary 11) << 2 = 12 (binary 1100)
  cout << "Left shift result: " <<
left_shift_result << '\n'; // Output: 12
  // Right shift (>>)
  int right_shift_result = 12 >> 2;  //
12 (binary 1100) >> 2 = 3 (binary 11)
  cout << "Right shift result: " <<
right_shift_result << '\n'; // Output: 3
  // Difference between 1 << x and 1LL <<
x
  int x = 31;
  long long res1 = 1 << x;  // This can
lead to overflow if x is large
  long long res2 = 1LL << x;  // This
avoids overflow since we're shifting on a
long long
  cout << "1 << x result: " << res1 <<
'\n';  // Output: -2147483648 (due to
overflow)
  cout << "1LL << x result: " << res2 <<
"\n\n";  // Output: 2147483648 (correct
value)
  x = 11; // binary representation: 1011
  cout << "Check 2nd bit of 11: " <<
check_kth_bit(x, 2) << '\n'; // Output: 0
  cout << "Set bits in 11 are at
positions: ";
  print_on_bits(x); // Output: 0 1 3
  cout << "Number of set bits in 11: " <<
count_on_bits(x) << '\n'; // Output: 3
  cout << "Is 11 even? " << is_even(x) <<
'\n'; // Output: 0 (false)
  cout << "11 after setting 2nd bit: " <<
set_kth_bit(x, 2) << '\n'; // Output: 15
  cout << "15 after unsetting 2nd bit: "
<< unset_kth_bit(15, 2) << '\n'; //
Output: 11
  cout << "11 after toggling 3rd bit: "
<< toggle_kth_bit(x, 3) << '\n'; //
Output: 3
  cout << "Is 8 a power of 2: " <<
check_power_of_2(8) << '\n'; // Output: 1
(true)
  return 0;
}
Find XOR of 1 to N Efficiently :
int xor1toN(int n) {
    if (n % 4 == 0) return n;
    if (n % 4 == 1) return 1;
    if (n % 4 == 2) return n + 1;
    return 0;
}
```

Bitwise Identities:

```
a|b = a^b + a&b
a^(a&b) = (a|b)^b
b^(a&b) = (a|b)^a
(a&b)^(a|b) = a^b
a+b = a|b + a&b
a+b = a^b + 2(a&b)
a-b = (a^(a&b))-((a|b)^a)
a-b = ((a|b)^b)-((a|b)^a)
a-b = (a^(a&b))-(b^(a&b))
a-b = ((a|b)^b)-(b^(a&b))
```

Bitmask:

```
int main(){
    int n , k ;
    cin>>n >> k;
    vector < int > v(n);
    for ( int i = 0 ; i< n ; i++)cin>>
v[i];
    int cnt = 0 ;
    for ( int mask = 0 ; mask < ( 1 << n
) ; mask++ ){
        int sum = 0 ;
        for (int i = 0 ; i< n ; i++ ){
            if ( (mask >> i )&1){
                sum = sum + v[i];
            }
        }
        if ( sum == k ){
            cnt++;
        }
    }
    cout << cnt << '\n';
    return 0;
}
```

Basic Recursion:

```
// Print 1 to N
int n ;
void yo( int cur ){
    if ( cur == n ) return;
    cout << cur+1 << '\n';
    yo(cur+1);
}
// Print N to 1
int n ;
void yo( int cur ){
    if ( cur <= 0 ) return;
    if ( cur == 1 ){
        cout << cur ;
    }
    else {
        cout << cur << " ";
    }
    yo(cur-1);
}
// Palindrom of array check
vector<int> v;
int n ;
bool yo(int l , int r ) {
    if ( l >= r ) return true;
    if ( v[l] != v[r]) return false;
    return yo (l+1 , r-1);
}
// No of way to reach l to r
// in 1 , 2, 3 steps
int l , r ;
int yo( int pos ){
    if ( pos == r ) return 1 ;
    if (pos > r ) return 0 ;
    int way = 0 ;
    way = yo(pos+1) + yo(pos+2) +
yo(pos+3);
    return way;
}
// Can reach in target
int n ;
bool yo( int num ){
    if( num == n ) return true;

    bool ok = false;
    if ( num * 10 <= n ) {
        ok |= yo(num*10);
    }
    if ( num*20 <= n ){
        ok |= yo(num*20);
    }
    return ok;
}
```

2.Basic Math:

Divisors(Trial):

```
vector<int> v;
void devisor(int n){
    for (int i = 1; i * i <= n; i++){
        if (n % i == 0){
            int x = i;
            int y = n / i;
            v.push_back(x);
            if (x != y) v.push_back(y);
        }
    }
    sort(v.begin(),v.end());
}
```

NOD & SOD:

```
#define int long long
const int MX = 1e6;
int spf[MX + 1];
vector<int> primes;
void computeSPF() {
    for (int i = 2; i <= MX; i++) {
        if (!spf[i]) {
            spf[i] = i;
            primes.push_back(i);
        }
        for (int p : primes) {
            if (p > spf[i] || 1LL * i * p
> MX) break;
            spf[i * p] = p;
        }
    }
}
int  NOD(int  n) {
    int  res = 1;
    for (int  p : primes) {
        if (p * p > n) break;
        int cnt = 0;
        while (n % p == 0) {
            n /= p;
            cnt++;
        }
        res *= (cnt + 1);
    }
    if (n > 1) res *= 2;
    return res;
}
int SOD(int n) {
    int res = 1;
    for (int p : primes) {
        if (p * p > n) break;
        if (n % p == 0) {
            int term = 1;
            int pe = 1;
            while (n % p == 0) {
                n /= p;
                pe *= p;
                term += pe;
            }
            res *= term;
        }
    }
    if (n > 1) {
        res *= (1 + n);
    }
    return res;
}
signed main() {

    computeSPF();
    int n ; cin >> n ;
    cout << NOD(n) << '\n';
    cout << SOD(n) << '\n';
    return 0;
}
```

GCD,LCM:

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

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t; cin >> t;
  while (t--) {
    int a, b; cin >> a >> b;
    int GCD = __gcd(a, b);
    long long LCM = 1LL * a * b / GCD;
    cout << LCM << ' ' << GCD << '\n';
  }
  return 0;
}
```

Harmonic Number:

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6 + 9;
int d[N];
int32_t main() {
  for (int i = 1; i < N; i++) {
    for (int j = i; j < N; j += i) {
      d[j]++;
    }
  }

  int t; cin >> t;
  while (t--) {
    int n; cin >> n;
    cout << d[n] << '\n';
  }
  return 0;
}
```

3. Range Query without Update:

1D Prefix :

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;cin >> n;
    vector<int> v(n + 1),prefix(n + 1,0);
    for (int i = 1; i <= n; i++ ) cin >>
v[i];
    for (int i = 1; i <= n; ++i){
        prefix[i] = prefix[i - 1] + v[i];
    }
    for (int i = 1; i <= n; ++i){
        cout << prefix[i] << " ";
    }
    return 0;
}
```

1D Suffix:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;
    vector<int> v(n + 1), suff(n + 1);
    for (int i = 1; i <= n; ++i){
        cin >> v[i];
    }
    suff[n] = v[n];
    for (int i = n - 1; i >= 1; --i){
        suff[i] = suff[i + 1] + v[i];
    }
    for (int i = 1; i <= n; ++i){
        cout << suff[i] << " ";
    }
}
```

Prefix Xor:

```
int main(){
    int n ; cin>>n;
    int a[n];
    for ( int i = 0 ; i< n ;i++ ){
        cin>> a[i];
    }
    int pre[n];
    pre[0] = a[0];
    for ( int i = 1 ; i < n ;i++){
        pre[i] = pre[i-1] ^ a[i];
    }
    int q;  cin>>q;
    while(q--)
    {
        int l , r ;
        cin>> l >> r;
        if ( l == 0 ){
            cout << pre[r] << '\n';
        }
        else{
            cout << (pre[r] ^ pre[l-1])
<< '\n';
        }
    }
    return 0;
}
```

4. STL:

```
// min_element
int result = *min_element(arr + 0, arr +
5);
// fill
fill(v.begin(), v.end(), -1);
fill(v.begin() + x, v.begin() + y, -1);
// fill_n
fill_n(v.begin(), n, -1);
// rotate
rotate(v.begin(), v.begin() + 2,
v.end());
// accumulate
int sum = accumulate(v.begin(), v.end(),
0);
// next_permutation
vector<int> v = {1,2,3};
sort(v.begin(), v.end());
do {
    for(int x: v) cout << x << " ";
    cout << "\n";
} while(next_permutation(v.begin(),
v.end()));
// 2. C++ STL Array
#include <array>
array<int, n> arr;
arr.fill(-1);
// 3. C++ STL String
#include <string>
string str1, str2;
// full compare
bool ok = str1.compare(str2);
// compare in range
// str1.substr(2,5) vs str2.substr(1,5)
bool ok2 = str1.compare(2, 5, str2, 1,
5);
// existence check
if(s.find('0') != string :: npos ){
   return false ;
}
// 4. C++ STL Set
#include <set>
set<int> st;
// erase
st.erase(x);
// empty
if(st.empty()) cout << "It's empty\n";
// clear
st.clear();
// lower_bound
auto it = st.lower_bound(6);
cout << *it;
// find
it = st.find(5);
```

```
if(it != st.end()) cout << *it;
// 5. C++ STL Vector
vector<vector<int>> vec2d(n,
vector<int>(m));
// 6. C++ STL Multiset
#include <set>
multiset<int> ms;
// erase all
ms.erase(x);
// erase 1 instance
ms.erase(ms.find(x));
// 7. String Stream
string s , k ;
getline(cin , s);
stringstream name(s);
while ( name >> k ){
    cout << k << " ";
}
```

5. Monotonocity

Bs on Ans:

```
#include<bits/stdc++.h>
using namespace std ;
int n ,k ;
bool check(int dis , vector<int>&v ){
    int cow = 1 ;
    int l = 0 ;
    for ( int r = 0 ; r < n ; r++ ){
        if ( v[r] - v[l] >= dis ){
            cow++;
            l = r;
        }
    }
    return cow >= k ;
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int tc ; cin>> tc ;
    for ( int i = 1 ; i <= tc ; i++ ){
        cin >> n >> k ;
        vector < int > v(n);
        for ( int i = 0 ; i < n ; i++ ){
            cin >> v[i];
        }
        sort ( v.begin() , v.end());
        int l = 1 ,r = v[n-1] - v[0] ;
        int ans = -1 ;
        while ( l <= r ){
            int mid = (l+r)/2;
           // cout << l << " " << r << "
" << mid << '\n';
            if ( check(mid , v )){
                ans = mid ;
                l = mid+1;
            }
            else{
                r = mid-1;
            }
        }
        cout << ans << '\n';
    }


    return 0 ;
}
```

Bs on Decimal:

```
#include<bits/stdc++.h>
using namespace std ;
bool good( long double x , long double n
){
   long double a = ( x * x ) + sqrt(x);
   return a >= n ;
}
int main(){
    long double n ; cin>> n ;
    long double l = 0 , r = 1e18 ;
    for ( int i = 0 ; i < 100 ; i++ ){
        long double mid = l + (r-l)/2;
        if ( good(mid , n )){
            r = mid ;
        }
        else{
            l = mid ;
        }
    }
    cout << setprecision(16) << r <<
 '\n';
    return 0 ;
 }
```

Ternary Search:

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

int check ( int val , vector<int>v , int n
){
    int maxi = 0;
    for (int i = 1; i < n; i++)
    {
        if (v[i] == -1)
            v[i] = val ;
        if (v[i - 1] == -1)
            v[i - 1] = val ;
        maxi = max(maxi, abs(v[i - 1] -
v[i]));
    }
    return maxi;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int tc ; cin>> tc ;
    for ( int i = 1 ; i <= tc ; i++ ){
        int n ; cin>> n ;
        vector < int > v(n);
        for ( int i = 0 ; i < n ; i++ ){
            cin >> v[i];
        }

        int l = 0 , r = 1e9;
        int ans = 0 ;
        while ( l <= r ){
            int m1 = l+(r-l)/3;
            int m2 = r-(r-l)/3;
            if(check(m1 , v , n ) >
check(m2 , v , n )){
               ans = m2;
                l = m1+1;
            }
            else {
               // ans = m1; can work also
                r = m2-1;
            }
        }
        cout << check(ans , v , n ) << " "
<< ans << '\n';
    }

    return 0 ;
}
```

6.Divide and Conquer

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

const int mx = 1e5+123;
vector< int > v ;
int n ;
void merge_( int lo, int mid, int hi )
{
    int l = lo ;
    int r = mid+1;
    vector < int > tmp ;
    while ( l <= mid && r <= hi ){
        if ( v[l] <= v[r]){
            tmp.push_back(v[l]);
            l++;
        }
        else{
            tmp.push_back(v[r]);
            r++;
        }
    }
    while ( l <= mid ){
        tmp.push_back(v[l]);
        l++;
    }
    while ( r <= hi ){
        tmp.push_back(v[r]);
        r++;
    }
    for ( int i = lo ; i <= hi ; i++ ){
        v[i] = tmp[i-lo];
    }
}

void merge_sort( int lo, int hi )
{
    if ( lo >= hi ) return ;
    int mid = (lo+hi)/2;
    merge_sort( lo, mid );
    merge_sort( mid+1, hi );
    merge_( lo, mid, hi );
}
int main()
{
    cin>> n ;
    v.resize(n);
    for ( int i = 0 ; i < n ; i++ ){
        cin>> v[i];
    }
    merge_sort( 0, n-1 );
    for ( auto it : v ){
        cout << it << " ";
    }
    cout << '\n';
    return 0 ;
}
```

7. 2D Prefix Sum

2D Prefix(Trivial):

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
signed main() {
    int n, m, q;
    cin >> n >> m >> q;
    vector<vector<int>> a(n+1,
vector<int>(m+1));
    vector<vector<int>> pref(n+1,
vector<int>(m+1));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            pref[i][j] = a[i][j] +
pref[i-1][j] + pref[i][j-1] -
pref[i-1][j-1];
        }
    }
    while (q--) {
        int x1, y1, x2, y2;
```

```
        cin >> x1 >> y1 >> x2 >> y2;
        int ans = pref[x2][y2] -
pref[x1-1][y2] - pref[x2][y1-1] +
pref[x1-1][y1-1];
        cout << ans << '\n';
    }
    return 0;
}
```

2D Prefix(Interval):

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int tc; cin >> tc;
    for (int i = 1; i <= tc; i++) {
        int n, m, k;
        cin >> n >> m >> k;
        char v[n + 5][m + 5];
        int pref[n + 5][m + 5];
        memset(pref, 0, sizeof(pref));
        int totG = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                cin >> v[i][j];
                if (v[i][j] == 'g') {
                    pref[i][j] = 1 +
pref[i - 1][j] + pref[i][j - 1] - pref[i -
1][j - 1];
                    totG++;
                }
                else {
                    pref[i][j] = pref[i -
1][j] + pref[i][j - 1] - pref[i - 1][j -
1];
                }
            }
        }
        int mn = INT_MAX;
        for( int i = 1 ; i <= n ; i++ ){
         for ( int j = 1 ; j <= m ; j++ ){
           if( v[i][j] == '.'){
          // interval
          int x1 = max(1 , i-k+1);
          int y1 = max(1 , j-k+1);
          int x2 = min( n , i+k-1);
          int y2 = min( m , j+k-1);
          int val =  pref[x2][y2] -
pref[x1 - 1][y2] - pref[x2][y1 - 1] +
pref[x1 - 1][y1 - 1];
                    mn = min( mn , val );
                }
            }
        }
        if (mn == INT_MAX){
            cout << 0 << '\n';
        }
        else {
            cout << totG - mn << '\n';
        }
    }

    return 0;
}
```

8. Difference Array

1D Difference:

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
signed main() {
    int n, q;
    cin >> n >> q;
    vector<int> diff(n + 2, 0);
    while (q--) {
        int l, r;
        int val;
        cin >> l >> r >> val;
        diff[l] += val;
        diff[r + 1] -= val;
    }
    vector<int> pre(n + 1);
    for (int i = 1; i <= n; i++) {
        pre[i] = pre[i - 1] + diff[i];
    }
    for (int i = 1; i <= n; i++) {
        cout << pre[i] << " ";
    }
}
```

2D Difference:

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

const int mx = 1000 + 12;
#define int long long
signed main() {
    int n, m;
    cin >> n >> m;
    vector<vector<char>> v(n + 2,
vector<char>(m + 2, 0));
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> v[i][j];
        }
    }
    vector<vector<int>> diff(n + 3,
vector<int>(m + 3, 0));
    int q;
    cin >> q;
    while (q--) {
        int x1 , y1 , x2 , y2 ;
        cin >> x1 >> y1 >> x2 >> y2;

        diff[x1][y1] += 1;
        diff[x2 + 1][y1] -= 1;
        diff[x1][y2 + 1] -= 1;
        diff[x2 + 1][y2 + 1] += 1;
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            diff[i][j] += diff[i - 1][j] +
diff[i][j - 1] - diff[i - 1][j - 1];
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (diff[i][j] % 2){
                if (v[i][j] == '1') {
                    v[i][j] = '0';
                }
                else {
                    v[i][j] = '1';
                }
            }

        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cout << v[i][j];
        }
        cout << '\n';
    }

    return 0;
}
```

9. Sliding Window

maxSumSubarrayOfSizeK:

```
#define int long long
int maxSumSubarrayOfSizeK(const
vector<int>& a, int k) {
    int sum = 0 , mx = LLONG_MIN;
    int n = a.size() ;

    for ( int i = 0 ; i < k ; i++ ){
        sum += a[i];
    }
    mx = max( mx , sum );
    for ( int i = k ; i < n ; i++ ){
        int out = a[i-k];
        int in = a[i];
        sum -= out;
        sum += in;
        mx = max(mx , sum);
    }
    return mx;
}
```

Longest Subarray with Sum ≤ K

```
// for pos
int longestSubarraySumLEK(const
vector<int>& a, long long k) {
    int n = a.size();
    int left = 0, ans = 0;
    long long sum = 0;

    for (int right = 0; right < n;
right++) {
        sum += a[right];
        while (sum > k) {
            sum -= a[left];
            left++;
        }
        ans = max(ans, right - left + 1);
    }
    return ans;
}
// pos + neg
int getLongestSubarray(vector<int>& a, int
k) {
    int n = a.size();

    map<int, int> preSumMap;
    int sum = 0;
    int maxLen = 0;
    for (int i = 0; i < n; i++) {
        sum += a[i];

        if (sum == k) {
            maxLen = max(maxLen, i + 1);
        }

        int rem = sum - k;

        if (preSumMap.find(rem) !=
preSumMap.end()) {
            int len = i - preSumMap[rem];
            maxLen = max(maxLen, len);
        }

        if (preSumMap.find(sum) ==
preSumMap.end()) {
            preSumMap[sum] = i;
        }
    }
    return maxLen;
}
```

Longest Subarraywith At Most K Distinct Elements:

```
int longestSubarrayAtMostKDistinct(const
vector<int>& a, int k) {
    map<int, int> freq;
    int left = 0, ans = 0;

    for (int right = 0; right <
(int)a.size(); right++) {
        freq[a[right]]++;
```

```
        while ((int)freq.size() > k) {
            freq[a[left]]--;
            if (freq[a[left]] == 0){
                freq.erase(a[left]);
            }
            left++;
        }
        ans = max(ans, right - left + 1);
    }
    return ans;
}
```

10. Bit Manupulation

Range Bit Check:

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

// Counts the number of integers in [0, n]
that have the k-th bit set.
ll count_bit(ll n, ll k) {
    if (n < 0) return 0;
    ll cycle = 1LL << (k+1);
    ll full_cycles = (n + 1) / cycle;
    ll count = full_cycles * (1LL << k);
    ll remainder = (n + 1) % cycle;
    if (remainder > (1LL << k)) {
        count += remainder - (1LL << k);
    }
    return count;
}

// Computes the bitwise AND of all
integers from l to r.
ll range_and(ll l, ll r) {
    ll ans = 0;
    ll total = r - l + 1;
    for (ll k = 0; k < 60; k++) {
        ll cnt_r = count_bit(r, k);
        ll cnt_l_minus = (l == 0) ? 0 :
count_bit(l-1, k);
        if (cnt_r - cnt_l_minus == total)
{
            ans |= (1LL << k);
        }
    }
    return ans;
}

// Computes the bitwise OR of all integers
from l to r.
ll range_or(ll l, ll r) {
    ll ans = 0;
    for (ll k = 0; k < 60; k++) {
        ll cnt_r = count_bit(r, k);
        ll cnt_l_minus = (l == 0) ? 0 :
count_bit(l-1, k);
        if (cnt_r - cnt_l_minus > 0) {
            ans |= (1LL << k);
        }
    }
    return ans;
}

// Computes the bitwise XOR of all
integers from l to r.
ll range_xor(ll l, ll r) {
    ll ans = 0;
    for (ll k = 0; k < 60; k++) {
        ll cnt_r = count_bit(r, k);
        ll cnt_l_minus = (l == 0) ? 0 :
count_bit(l-1, k);
        if ((cnt_r - cnt_l_minus) % 2 ==
1) {
            ans |= (1LL << k);
        }
    }
    return ans;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int t;
    cin >> t;
    while (t--) {
        ll l, r;
        cin >> l >> r;
        ll A = range_and(l, r);
        ll O = range_or(l, r);
        ll X = range_xor(l, r);
        cout << A << " " << O << " " << X
<< "\n";
    }

    return 0;
}
```

11. Modular Arithmetic

Modular Multiplicative Inverse:

```
//Topic : Modular Multiplicative Inverse
(log(n))
#include<bits/stdc++.h>
using namespace std ;

#define LL long long int
// (a^n % mod)
int bin_power ( int a , int n , int mod)
{
    int ans = 1 ;
    while ( n ){
        if ( n & 1){
            ans = ( ans * 1LL * a ) % mod
;
        }
        a = ( a *1LL * a )% mod ;
        n >>= 1 ;
    }
    return ans ;
}

int inverse ( int a , int n ){
    return ( bin_power ( a , n-2 , n ) );
}
int main()
{
    int a = 5 , n = 7;
    // is this calculating (a⁻¹ % n)
    cout << inverse ( a , n );
    return 0;
}
```

12. nCr

nCr(Prime):

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

const int N = 1e6 + 9, mod = 1e9 + 7;

int f[N], inv[N], finv[N];
void prec() {
  f[0] = 1;
  for (int i = 1; i < N; i++) {
    f[i] = 1LL * i * f[i - 1] % mod;
  }
  inv[1] = 1;
  for (int i = 2; i < N; i++ ) {
    inv[i] = (-(1LL * mod / i) * inv[mod %
i] ) % mod;
    inv[i] = (inv[i] + mod) % mod;
  }
  finv[0] = 1;
  for (int i = 1; i < N; i++) {
    finv[i] = 1LL * inv[i] * finv[i - 1] %
mod;
  }
}
int ncr(int n, int r) {
  if (n < r || n < 0 || r < 0) return 0;
  return 1LL * f[n] * finv[n - r] % mod *
finv[r] % mod;
}

signed main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  prec();
  int tc ; cin>> tc ;
  for ( int i = 1 ; i <= tc ; i++ ){
    int n, r; cin >> n >> r;
    cout << ncr(n, r) << '\n';
  }
  return 0;
}
```

nCr(Pascal):

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

const int N = 1005;
const int mod = 1e9 + 7;
int C[N][N];

// Precompute nCr using Pascal's Triangle
void preCal() {
    for (int i = 0; i < N; i++) {
        C[i][0] = 1; // nC0 = 1
        for (int j = 1; j <= i; j++) {
            C[i][j] = (C[i - 1][j - 1] +
C[i - 1][j]) % mod;
        }
    }
}

int ncr(int n, int r) {
    if (r < 0 || r > n) return 0;
    return C[n][r];
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    preCal();
    int tc ; cin>> tc ;
    for ( int i = 1 ; i <= tc ; i++ ){
        int n, r;
        cin >> n >> r;
        cout << ncr(n, r) << '\n';
    }

    return 0;
}
```

Number Theory

1. Primality

Primality Check(Trial):

```
bool isPrime = true ;
    if ( n == 0 || n == 1 ){
        isPrime = false ;
    }
    for ( int i = 2 ; i * i <= n ; i++ ){
        if ( n % i == 0 ){
            isPrime = false ;
            break ;
        }
```

```
    }
```

Primality(Miller Robin):

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

using ll = long long;
namespace MillerRabin {
  mt19937
rnd(chrono::steady_clock::now().time_since
_epoch().count());
  const int P = 1e6 + 9;
  int primes[P], spf[P];
  inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
    // ll res = x * y - (ll)((long
double)x * y / m + 0.5) * m;
    // return res < 0 ? res + m : res;
  }
  inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
      if (n & 1) res = mul_mod(res, x, m);
      x = mul_mod(x, x, m);
    }
    return res;
  }
  // O(it * (logn)^3), it = number of
rounds performed (but faster in practice)
  inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n
== 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n &&
primes[i] < 32; i++) {
      c = pow_mod(primes[i], r, n);
      for (int j = 0; j < s; j++) {
        d = mul_mod(c, c, n);
        if (d == 1 && c != 1 && c != (n -
1)) return false;
        c = d;
      }
      if (c != 1) return false;
    }
    return true;
  }
  void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
      if (!spf[i]) primes[cnt++] = spf[i]
= i;
      for (int j = 0, k; (k = i *
primes[j]) < P; j++) {
        spf[k] = primes[j];
        if (spf[i] == spf[k]) break;
      }
    }
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  MillerRabin::init();
  int t; cin >> t;
  while (t--) {
    ll n; cin >> n;
    if (MillerRabin::miller_rabin(n)) {
      cout << "Yes\n";
    } else {
      cout << "No\n";
    }
  }
  return 0;
}
```

2. Prime Factorization

Prime factorization(Trial):

```
int n ;cin>>n ;
    vector < int > pf ;
    for ( int i = 2 ; i * i <= n ; i++ ){
        while ( n%i == 0 ){
            pf.push_back(i);
            n = n/i;
        }
    }
    if ( n > 1 ){
        pf.push_back(n);
    }
```

Factorization using SPF:

```
const int mx = 1e6;
int spf[mx + 1];
vector<int> primes;
vector<int> g[mx + 1];
void computeSPF() {
    for (int i = 2; i <= mx; ++i) {
        if (spf[i] == 0) {
            spf[i] = i;
            primes.push_back(i);
        }
        for (int p : primes) {
            if (p > spf[i] || i * p > mx)
break;
            spf[i * p] = p;
        }
    }
}
vector<int> getPFactor(int val) {
    vector<int> tmp;
    while (val > 1) {
        int p = spf[val];
        tmp.push_back(p);
        // calculate freq
        while (val % p == 0) {
            val /= p;
        }
    }
    return tmp;
}
int main() {
    computeSPF();
    return 0;
}
```

Prime Factor of (n!):

```
long long fact_pow(int n, int p) {
    long long res = 0;
    while (n) {
        n /= p;
        res += n;
    }
    return res;
}
```

3. Prime number generate

Linear Scieve:

```
const int mx = 1e7+123;
vector<int> v (mx + 1);
vector<int> prime ;
int main()
{
    for (int i = 2; i <= mx; i++ ){
        if (v[i] == 0){
            v[i] = i;
            prime.push_back(i);
        }
        for (int j = 0; j < prime.size()
&& i * prime[j] <= mx; j++){
            v[i * prime[j]] = prime[j];
            if (prime[j] == v[i]){
                break;
            }
        }
    }
}
```

Bitwise Scieve[O(N/8)]:

```
const int mx = 1e8;
int status[mx/32 + 5];
vector<int> prime;
bool check(int n, int pos) {
    return (bool)(n & (1 << pos));
}
int mark(int n, int pos) {
    return n = (n | (1 << pos));
}
void bitwiseSieve() {
    for (int i = 3; i*i <= mx; i += 2) {
        if (check(status[i/32], i % 32) ==
0) {
            for (int j = i * i; j <= mx; j
+= 2 * i) {
                status[j/32] =
mark(status[j/32], j % 32);
            }
        }
    }
    prime.push_back(2);
    for (int i = 3; i <= mx; i += 2) {
        if (check(status[i/32], i % 32) ==
0) {
            prime.push_back(i);
        }
    }
}
int main() {
    bitwiseSieve();
}
```

Prime using SPF:

```
const int mx =100000000 ;
int spf[mx + 1];
vector<int> primes;
void SPF() {
    for (int i = 2; i <= mx; ++i) {
        if (spf[i] == 0) {
            spf[i] = i;
            primes.push_back(i);
        }
        for (int p : primes) {
            if (p > spf[i] || i * p > mx)
break;
            spf[i * p] = p;
        }
    }
}
int main(){
    SPF();
}
```

Segment scieve:

```
#define ll long long
// Generate all primes
vector<int> sieve(int limit) {
    vector<bool> is_prime(limit + 1,
true);
    is_prime[0] = is_prime[1] = false;
    for (int p = 2; p * p <= limit; ++p) {
        if (is_prime[p]) {
            for (int i = p * p; i <=
limit; i += p) {
                is_prime[i] = false;
            }
        }
    }
    vector<int> primes;
    for (int p = 2; p <= limit; ++p) {
        if (is_prime[p]) {
            primes.push_back(p);
```

```
        }
    }
    return primes;
}
// O((r - l) log (r) + sqrt(r))
vector<ll> segmented_sieve(ll l, ll r) {
    if (l == 1) {
        l++;
    }
    int limit = sqrtl(r);
    while ((ll) limit * limit <= r)
limit++;
    while ((ll) limit * limit > r)
limit--;
    auto primes = sieve(limit);
    vector<bool> is_prime(r - l + 1,
true);
    for (ll p : primes) {
        ll start = max((ll)p * p, (ll)(l +
p - 1) / p * p);
        for (ll j = start; j <= r; j += p)
{
            is_prime[j - l] = false;
        }
    }
    vector<ll> vec;
    for (ll i = l; i <= r; ++i) {
        if (is_prime[i - l]) {
            vec.push_back(i);
        }
    }
    return vec;
}

int main(){
    ll l, r;
    cin >> l >> r;
    auto primes = segmented_sieve(l, r);
}
```

Phi Function(Co-prime count):

```
i ( int n )
{
    int ans = n ;
    int cnt = 0 ;
    for ( int i = 2 ; i * i <= n ; i++ ){
        if ( n % i == 0 ){
            while ( n % i == 0 ){
                cnt++;
                n = n/i;
            }
            // formula of phi function
            ans = ans * ( i - 1 );
            ans = ans / i ;
        }
    }
    if ( n > 1 ){
        ans = ans * ( n - 1 );
        ans = ans / n ;
    }
    cout << "NOD" << cnt << '\n';
    cout << ans << '\n';

}

int main(){
    int n ; cin>> n;
    phi(n);
}
```

Graph Theory

2D Grid Trversal, Path Generate:

```
int n, m;
vector<vector<int>> g;
vector<vector<int>> vis;
vector<vector<pair<int,int>>> par;
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

bool valid(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y
< m && !vis[x][y] && g[x][y] == 0;
}

void bfs(int sx, int sy, int tx, int ty) {
    queue<pair<int,int>> q;
    q.push({sx, sy});
    vis[sx][sy] = 1;
    par[sx][sy] = {-1, -1};

    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        if (x == tx && y == ty) break;
        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            if (valid(nx, ny)) {
                vis[nx][ny] = 1;
                par[nx][ny] = {x, y};
                q.push({nx, ny});
            }
        }
    }

    vector<pair<int,int>> path;
    if (!vis[tx][ty]) {
        cout << "No path found\n";
        return;
    }

    for (pair<int,int> cur = {tx, ty};
cur.first != -1; cur =
par[cur.first][cur.second])
        path.push_back(cur);

    reverse(path.begin(), path.end());
    for (auto [x, y] : path)
        cout << "(" << x << "," << y << ")
";
    cout << "\n";
}

int main() {
    cin >> n >> m;
    g.assign(n, vector<int>(m));
    vis.assign(n, vector<int>(m, 0));
    par.assign(n, vector<pair<int,int>>(m,
{-1, -1}));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> g[i][j];

    int sx, sy, tx, ty;
    cin >> sx >> sy >> tx >> ty;
    bfs(sx, sy, tx, ty);
}
```

Knight Traversal:

```
int n, m;
vector<vector<int>> vis;
int dx[] = {-2, -2, -1, -1, 1, 1, 2, 2};
int dy[] = {-1, 1, -2, 2, -2, 2, -1, 1};
bool valid(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y
< m && !vis[x][y];
}
void bfs(int sx, int sy) {
    queue<pair<int,int>> q;
    q.push({sx, sy});
    vis[sx][sy] = 1;
    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        for (int i = 0; i < 8; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            if (valid(nx, ny)) {
                vis[nx][ny] = 1;
                q.push({nx, ny});
            }
        }
    }
}
int main() {
    cin >> n >> m;
    vis.assign(n, vector<int>(m, 0));
    bfs(0, 0);
}
```

Componen(dfs):

```
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  vector<int> com;
  for (int u = 1; u <= n; u++) {
    if (!vis[u]) {
      com.push_back(u);
      dfs(u);
    }
  }
  cout << (int)com.size() - 1 << '\n';
  for (int i = 0; i + 1 < (int)com.size();
i++) {
    cout << com[i] << ' ' << com[i + 1] <<
'\n';
  }
}
```

Cycle Detect(Directed):

```
const int N = 2e5 + 5;
vector<int> g[N];
vector<int> vis, inStack;
int n, m;
bool dfs(int u) {
    vis[u] = 1;
    inStack[u] = 1;
    for (int v : g[u]) {
        if (!vis[v]) {
            if (dfs(v)) return true;
        }
        else if (inStack[v]) {
            return true;
        }
    }
    inStack[u] = 0;
    return false;
}
bool hasCycleDirected(int n) {
    vis.assign(n, 0);
    inStack.assign(n, 0);
    for (int i = 0; i < n; i++) {
        if (!vis[i] && dfs(i))
```

```
            return true;
    }
    return false;
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    if (hasCycleDirected(n)){
        cout << "Cycle Found" << '\n';
    }
}
```

Cycle Detect(Undirected):

```
const int N = 2e5 + 5;
vector<int> g[N];
vector<int> vis;
int n, m;
bool dfs(int u, int parent) {
    vis[u] = 1;
    for (int v : g[u]) {
        if (!vis[v]) {
            if (dfs(v, u)) return true;
        }
        else if (v != parent) {
            return true;
        }
    }
    return false;
}
bool hasCycleUndirected(int n) {
    vis.assign(n, 0);
    for (int i = 0; i < n; i++) {
        if (!vis[i] && dfs(i, -1))
            return true;
    }
    return false;
}
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    if (hasCycleUndirected(n)){
        cout << "Cycle Found" << '\n';
    }
}
```

Bipartitte Check:

```
const int N = 1e5 ;
vector< int > g[N];
bool vis[N] , ok;
int col[N];
void dfs( int u ){
    vis[u] = true ;
    for ( auto v : g[u] ){
        if ( !vis[v] ){
           col[v] = col[u] ^ 1 ;
           dfs(v);
        }
        else{
            if ( col[v] == col[u] ){
                ok = false ;
            }
        }
    }
}
int main()
{
    int n , m ;
    cin>> n >> m ;
    while ( m-- )
    {
        int u , v ;
        cin>> u >> v ;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    ok = true ;
    for ( int i = 1 ; i <= n ; i++ ){
        if ( !vis[i] ){
           dfs(i);
        }
    }
    if (ok) cout << "Bipartitr" << '\n';
}
```

Path generate(bfs):

```
int n, m;
vector<vector<int>> grid;
vector<vector<int>> vis;
vector<vector<pair<int,int>>> parent;
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};
bool valid(int x, int y) {
    return x >= 0 && x < n && y >= 0 && y
< m && !vis[x][y] && grid[x][y] == 0;
}
void bfs(int sx, int sy, int tx, int ty) {
    queue<pair<int,int>> q;
    q.push({sx, sy});
    vis[sx][sy] = 1;
    parent[sx][sy] = {-1, -1};

    while (!q.empty()) {
        auto [x, y] = q.front();
        q.pop();
        if (x == tx && y == ty) break;
        for (int i = 0; i < 4; i++) {
            int nx = x + dx[i];
            int ny = y + dy[i];
            if (valid(nx, ny)) {
                vis[nx][ny] = 1;
                parent[nx][ny] = {x, y};
                q.push({nx, ny});
            }
        }
    }
    vector<pair<int,int>> path;
    if (!vis[tx][ty]) {
        cout << "No path found\n";
        return;
    }
    for (pair<int,int> cur = {tx, ty};
cur.first != -1; cur =
parent[cur.first][cur.second])
        path.push_back(cur);

    reverse(path.begin(), path.end());
    for (auto [x, y] : path)
        cout << "(" << x << "," << y << ")
";
    cout << "\n";
}

int main() {
    cin >> n >> m;
    grid.assign(n, vector<int>(m));
    vis.assign(n, vector<int>(m, 0));
    parent.assign(n,
vector<pair<int,int>>(m, {-1, -1}));

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> grid[i][j];
    int sx, sy, tx, ty;
    cin >> sx >> sy >> tx >> ty;
    bfs(sx, sy, tx, ty);
}
```

Topological Sort:

```
const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
vector<int> ord;
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
  ord.push_back(u);
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i);
    }
  }
  reverse(ord.begin(), ord.end());

  vector<int> pos(n + 1);
  for (int i = 0; i < (int) ord.size();
i++) {
    pos[ord[i]] = i;
  }
  for (int u = 1; u <= n; u++) {
    for (auto v: g[u]) {
      if (pos[u] >= pos[v]) {
        cout << "IMPOSSIBLE\n";
        return 0;
      }
    }
  }
  for (auto u: ord) cout << u << ' ';
  cout << '\n';;
}
```

LCA:

```
const int N = 3e5 + 9, LG = 18;
vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i]
= par[par[u][i - 1]][i - 1];
  for (auto v: g[u]) if (v != p) {
    dfs(v, u);
    sz[u] += sz[v];
  }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if
(dep[par[u][k]] >= dep[v]) u = par[u][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if
(par[u][k] != par[v][k]) u = par[u][k], v
= par[v][k];
  return par[u][0];
}
```

```
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1
<< i)) u = par[u][i];
  return u;
}
int dist(int u, int v) {
  int l = lca(u, v);
  return dep[u] + dep[v] - (dep[l] << 1);
}
//kth node from u to v, 0th node is u
int go(int u, int v, int k) {
  int l = lca(u, v);
  int d = dep[u] + dep[v] - (dep[l] << 1);
  assert(k <= d);
  if (dep[l] + k <= dep[u]) return kth(u,
k);
  k -= dep[u] - dep[l];
  return kth(v, dep[v] - dep[l] - k);
}
int32_t main() {
  int n; cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs(1);
  int q; cin >> q;
  while (q--) {
    int u, v; cin >> u >> v;
    cout << dist(u, v) << '\n';
  }
}
```

MST(Prim):

```
const int N = 2020;
int g[N][N], w[N], to[N], selected[N];
long long Prims(int n, vector< pair<int,
int> > &edges) {
  long long ans = 0;
  for(int i = 1; i <= n; i++) w[i] = 1e9,
selected[i] = 0, to[i] = -1;
  w[1] = 0;
  for(int i = 1; i <= n; i++) {
    int u = -1;
    for(int j = 1; j <= n; j++)
if(!selected[j] && (u == -1 || w[j] <
w[u])) u = j;
    if (w[u] == 1e9) return - 1;
    selected[u] = 1;
    ans += w[u];
    if(to[u] != -1) edges.emplace_back(u,
to[u]);
    for(int v = 1; v <= n; v++) if(g[u][v]
< w[v]) w[v] = g[u][v], to[v] = u;
  }
  return ans;
}
string s[N];
int main() {
  int n, m; cin >> n >> m;
  for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) g[i][j] =
1e9;
  }
  for(int i = 1; i <= n; i++) cin >> s[i];
  for(int i = 1; i <= n; i++){
    for(int j = i + 1; j <= n; j++){
      int w = 0;
      for(int k = 0; k < m; k++) w =
max(w, (int)abs(s[i][k] - s[j][k]));
      g[i][j] = min(g[i][j], w);
      g[j][i] = min(g[j][i], w);
    }
  }
  vector< pair<int, int> > ed;
  long long ans = Prims(n, ed);
  int res = 0; for(auto e: ed){
     res = max(res, g[e.first][e.second]);
  }
  cout << res << '\n';
  return 0;
}
```

MST(Krushkals):

```
const int N = 3e5 + 9, mod = 1e9;
struct dsu {
  vector<int> par, rnk, size; int c;
  dsu(int n) : par(n+1), rnk(n+1,0),
size(n+1,1), c(n) {
    for (int i = 1; i <= n; ++i) par[i] =
i;
  }
  int find(int i) { return (par[i] == i ?
i : (par[i] = find(par[i]))); }
  bool same(int i, int j) { return find(i)
== find(j); }
  int get_size(int i) { return
size[find(i)]; }
  int count() { return c; } //connected
components
  int merge(int i, int j) {
    if ((i = find(i)) == (j = find(j)))
return -1; else --c;
    if (rnk[i] > rnk[j]) swap(i, j);
    par[i] = j; size[j] += size[i];
    if (rnk[i] == rnk[j]) rnk[j]++;
    return j;
  }
};

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  vector<array<int, 3>> ed;
  for(int i = 1; i <= m; i++){
    int u, v, w; cin >> u >> v >> w;
    ed.push_back({w, u , v});
  }
  sort(ed.begin(), ed.end());
  long long ans = 0;
  dsu d(n);
  for (auto e: ed){
    int u = e[1], v = e[2], w = e[0];
    if (d.same(u, v)) continue;
    ans += w;
    d.merge(u, v);
  }
  cout << ans << '\n';
  return 0;
}
```

Dijastra:

```
const int N = 2e5 + 5;
const long long INF = 1e18;
vector<pair<int, int>> g[N];
vector<long long> dist;
int n, m;

void dijkstra(int src) {
    dist.assign(n + 1, INF);
    dist[src] = 0;
    priority_queue<pair<long long, int>,
vector<pair<long long, int>>, greater<>>
pq;
    pq.push({0, src});

    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if (d != dist[u]) continue;
        for (auto [v, w] : g[u]) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }
    int src;
    cin >> src;
    dijkstra(src);
    for (int i = 1; i <= n; i++) {
        if (dist[i] == INF) cout << "INF
";
        else cout << dist[i] << " ";
    }
}
```

Bellmen-ford:

```
const int N = 2e5 + 5;
const long long INF = 1e18;
struct Edge {
    int u, v;
    long long w;
};
int n, m;
vector<Edge> edges;
vector<long long> dist;
bool bellmanFord(int src) {
    dist.assign(n + 1, INF);
    dist[src] = 0;
    // Relax all edges (n-1) times
    for (int i = 1; i < n; i++) {
        for (auto &e : edges) {
            if (dist[e.u] != INF &&
dist[e.u] + e.w < dist[e.v]) {
                dist[e.v] = dist[e.u] +
e.w;
            }
        }
    }
    for (auto &e : edges) {
        if (dist[e.u] != INF && dist[e.u]
+ e.w < dist[e.v]) {
            return false;
        }
    }
    return true;
}
int main() {
    cin >> n >> m;
    edges.resize(m);
    for (int i = 0; i < m; i++) {
        cin >> edges[i].u >> edges[i].v >>
edges[i].w;
    }
    int src; cin >> src;
    if (!bellmanFord(src))
        cout << "Negative Cycle
Detected\n";
    else {
        for (int i = 1; i <= n; i++) {
            if (dist[i] == INF) cout <<
"INF ";
            else cout << dist[i] << " ";
        }
        cout << "\n";
    }
```

```
}
```

Floyd-Warshall:

```
const int N = 505;
const long long INF = 1e15;
long long dist[N][N];
int n, m;
void floydWarshall() {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (dist[i][k] < INF &&
dist[k][j] < INF)
                    dist[i][j] =
min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            dist[i][j] = (i == j ? 0 :
INF);

    for (int i = 0; i < m; i++) {
        int u, v;
        long long w;
        cin >> u >> v >> w;
        dist[u][v] = min(dist[u][v], w);
    }
    floydWarshall();
    for (int i = 1; i <= n; i++) {
        if (dist[i][i] < 0) {
            cout << "Negative Cycle
Detected\n";
            return 0;
        }
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dist[i][j] == INF) cout <<
"INF ";
            else cout << dist[i][j] << "
";
        }
        cout << "\n";
    }
}
```

Tree

Depth and Parent:

```
const int mx = 1e3+12;
vector<int>g[mx];
int parent[mx];
int depth[mx];
void dfs( int u , int p ){
    parent[u] = p ;
    depth[u] = depth[p]+1 ;
    if (p == -1 ){
        depth[u] = 0 ;
    }
    for ( auto v : g[u]){
        if ( v == p ) continue;
        dfs(v,u);
    }
}
int main()
{
    int n ; cin >> n ;
    for ( int i = 0 ; i < n-1 ; i++ ){
        int u , v ;
        cin >> u >> v ;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    int root = 1; // assume root
    dfs(root , -1);
    cout << parent[1] << " " << depth[1]
<< '\n';
}
```

SubtreeSize,ChildCount,Leaf:

```
const int mx = 1e3+12;
vector<int> g[mx];
int subtree_size[mx];
int child_count[mx];
bool isLeaf[mx];
void dfs( int u , int p ){
    subtree_size[u] = 1 ;
    child_count[u] = 0 ;
    for ( auto v : g[u]){
        if ( v == p ) continue;
        dfs(v , u);
        subtree_size[u] +=
subtree_size[v];
        child_count[u]++;
    }
    if (child_count[u] == 0 ){
        isLeaf[u] = true;
    }
}

int main()
{
    int n ; cin >> n ;
    for ( int i = 0 ; i < n-1 ; i++ ){
        int u , v ;
        cin >> u >> v ;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    int root = 1; // assume root
    dfs(root , -1);
    return 0 ;
}
```

Diameter,print&countDiameter:

```
#define int long long
const int mx = 1e5+12;
vector<int> g[mx];
int farthest(int s, int n ,vector<int>
&dist, vector<int> &parent) {
    dist.assign(n+1, -1);
    parent.assign(n+1, -1);
    queue<int> q;
    q.push(s);
    dist[s] = 0;
    int last = s;

    while(!q.empty()) {
        int u = q.front(); q.pop();
        last = u;
        for(int v: g[u]) {
            if(dist[v] == -1) {
                dist[v] = dist[u]+1;
                parent[v] = u;
                q.push(v);
            }
        }
    }
    return last;
}
// Returns diameter path
vector<int> get_diameter(int n ) {
    vector<int> dist, parent;
    int u = farthest(1, n, dist, parent);
    int v = farthest(u, n, dist, parent);

    vector<int> path;
    int cur = v;
    while(cur != -1) {
        path.push_back(cur);
        cur = parent[cur];
    }
    reverse(path.begin(), path.end());
    return path;
}
int count_diameters( vector<int>&
diameter) {
    int len = diameter.size();
    int res = 1;

    for(int i = 1; i < len-1; i++) {
        int u = diameter[i];
        int cnt = 0;
        for(int v : g[u]) {
            if(v != diameter[i-1] && v !=
diameter[i+1])
                cnt++;
        }
        res *= max(1ll, cnt);
    }

    return res;
}

signed main() {
    int n ; cin >> n ;
    for ( int i = 0 ; i < n-1 ; i++ ){
        int u , v ;
        cin >> u >> v ;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    auto diameter = get_diameter(n);
    cout << diameter.size()-1 << '\n';
    for ( auto it : diameter){
        cout << it << " ";
    }
    cout << '\n';
    int num_diameters =
count_diameters(diameter) ;
    cout << num_diameters << '\n';
}
```

Center of Tree:

```
// Find center(s) of tree
vector<int> get_centers( vector<int>&
diameter) {
    int len = diameter.size();
    vector<int> centers;
    if(len % 2 == 1) {

centers.push_back(diameter[len/2]);
// odd length → 1 center
    } else {
        centers.push_back(diameter[len/2 -
1]);   // even length → 2 centers

centers.push_back(diameter[len/2]);
    }
    return centers;
}
```

Centroid of Tree:

```
const int mx = 1e3+12;
vector<int> g[mx];
int subtree_size[mx];
int n ;
void dfs( int u , int p ){
    subtree_size[u] = 1 ;
    for ( auto v : g[u]){
        if ( v == p ) continue;
        dfs(v , u);
        subtree_size[u] +=
subtree_size[v];
    }
}
int centroid( int u , int p){
    for ( auto v : g[u]){
        if( v == p ) continue;
```

```
        if ( subtree_size[v] > n/2 ){
            return centroid( v , u);
        }
    }
    return u ;
}
int main()
{
    cin >> n ;
    for ( int i = 0 ; i < n-1 ; i++ ){
        int u , v ;
        cin >> u >> v ;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    int root = 1; // assume root
    dfs(root , -1);
    int node = centroid(root , -1);
    cout << node << '\n';
}
```

Data Structure

1. Trie:

BitTrie:

```
struct BitTrie {
    static const int B = 31;
    struct node {
        node* nxt[2];
        int sz;
        node() {
            nxt[0] = nxt[1] = nullptr;
            sz = 0;
        }
    } *root;

    BitTrie() {
        root = new node();
    }

    // Insert integer into trie
    void insert(int val) {
        node* cur = root;
        cur->sz++;
        for (int i = B - 1; i >= 0; i--) {
            int b = (val >> i) & 1;
            if (!cur->nxt[b]) cur->nxt[b]
= new node();
            cur = cur->nxt[b];
            cur->sz++;
        }
    }

    // Count numbers such that val ^ x < k
    int query(int x, int k) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            if (!cur) break;
            int b1 = (x >> i) & 1;
            int b2 = (k >> i) & 1;

            if (b2) {
                if (cur->nxt[b1]) ans +=
cur->nxt[b1]->sz;
                cur = cur->nxt[!b1];
            } else {
                cur = cur->nxt[b1];
            }
        }
        return ans;
    }

    // Get maximum XOR with x
    int get_max(int x) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            int k = (x >> i) & 1;
            if (cur->nxt[!k]) {
                cur = cur->nxt[!k];
                ans = (ans << 1) | 1;
            } else {
                cur = cur->nxt[k];
                ans = ans << 1;
            }
        }
        return ans;
    }
    // Get minimum XOR with x
    int get_min(int x) {
        node* cur = root;
        int ans = 0;
        for (int i = B - 1; i >= 0; i--) {
            int k = (x >> i) & 1;
            if (cur->nxt[k]) {
                cur = cur->nxt[k];
                ans <<= 1;
            } else {
                cur = cur->nxt[!k];
                ans = (ans << 1) | 1;
            }
        }
        return ans;
    }
    void del(node* cur) {
        for (int i = 0; i < 2; i++)
            if (cur->nxt[i])
del(cur->nxt[i]);
        delete cur;
    }

    ~BitTrie() {
        del(root);
    }
};

int32_t main() {
    BitTrie t;
    int n, k;
    cin >> n >> k;
    int cur = 0;
    long long ans = 1LL * n * (n + 1) / 2;
    t.insert(cur);
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        cur ^= x;
        ans -= t.query(cur, k);
        t.insert(cur);
    }
    cout << ans << "\n";
}
```

String Trie:

```
struct StringTrie {
    static const int ALPHA = 26;
    struct node {
        node* nxt[ALPHA];
        int sz;
        bool isEnd;
        node() {
            fill(nxt, nxt + ALPHA,
nullptr);
            sz = 0;
            isEnd = false;
        }
    }* root;
    StringTrie() {
        root = new node();
    }
    // Insert a word into the trie
    void insert(const string &s) {
        node* cur = root;
        cur->sz++;
        for (char c : s) {
            int idx = c - 'a';
            if (!cur->nxt[idx])
cur->nxt[idx] = new node();
            cur = cur->nxt[idx];
            cur->sz++;
        }
        cur->isEnd = true;
    }
    // Count how many words in the trie
are equal to s
    int countWord(const string &s) {
        node* cur = root;
        for (char c : s) {
            int idx = c - 'a';
            if (!cur->nxt[idx]) return 0;
            cur = cur->nxt[idx];
        }
        return cur->isEnd ? 1 : 0;
    }
    // Count words that start with prefix
    int countPrefix(const string &prefix)
{
        node* cur = root;
        for (char c : prefix) {
            int idx = c - 'a';
            if (!cur->nxt[idx]) return 0;
            cur = cur->nxt[idx];
        }
        return cur->sz;
    }
    // Delete the trie to free memory
    void del(node* cur) {
        for (int i = 0; i < ALPHA; i++) if
(cur->nxt[i]) del(cur->nxt[i]);
        delete cur;
    }

    ~StringTrie() {
        del(root);
    }
};
int32_t main() {
    StringTrie t;
    int n;
    cin >> n;
    vector<string> words(n);
    for (int i = 0; i < n; i++) {
        cin >> words[i];
        t.insert(words[i]);
    }
    string queryWord;
    cin >> queryWord;
    cout << "Exact word count: " <<
t.countWord(queryWord) << "\n";
    cout << "Prefix count: " <<
t.countPrefix(queryWord) << "\n";
}
```

2. Monotonic:

Monotonic Queue:

```
// Maintain max/min in every k-length
window :
#define int long long
vector<int> slidingWindowMax(const
vector<int>& a, int k) {
    int n = a.size();
    deque<int> dq;
    vector<int> res
    for (int i = 0; i < k; i++) {
        // pushing decresing order (i.e, 5
4 3..)
        while (!dq.empty() && a[dq.back()]
< a[i]) {
            dq.pop_back();
        }
```

```
        dq.push_back(i);
    }
    res.push_back(a[dq.front()]);
    for (int i = k; i < n; i++) {
        // out
        while (!dq.empty() && dq.front()
<= i - k) {
            dq.pop_front();
        }
        // in
        while (!dq.empty() && a[dq.back()]
< a[i]) {
            dq.pop_back();
        }
        dq.push_back(i);
        res.push_back(a[dq.front()]);
    }
    return res;
}
// Longest Subarray / Window with Absolute
Difference ≤ K (max - min ≤ K)
#define int long long
int longestSubarrayAbsDiffLEK(const
vector<int>& a, int k) {
    deque<int> mx, mn;
    int l = 0, ans = 0;
    int n = a.size();
    for (int r = 0; r < n; r++) {

        // inserting decresing order
        while (!mx.empty() && a[mx.back()]
< a[r]) {
            mx.pop_back();
        }
        mx.push_back(r);
        // inserting incresing order
        while (!mn.empty() && a[mn.back()]
> a[r]) {
            mn.pop_back();
        }
        mn.push_back(r);

        while (a[mx.front()] -
a[mn.front()] > k) {
            if (mx.front() == l)
mx.pop_front();
            if (mn.front() == l)
mn.pop_front();
            l++;
        }
        ans = max(ans, r - l + 1);
    }
    return ans;
}
```

Monotonic Stack:

```
// Previous Smaller / Greater
vector<int> prevSmaller(const vector<int>&
a) {
    int n = a.size();
    vector<int> res(n, -1); // -1 if no
previous smaller
    stack<int> st;
    for (int i = 0; i < n; i++) {
        while (!st.empty() && a[st.top()]
>= a[i]) {
            st.pop(); // change >= to <=
for prev greater
        }
        if (!st.empty()) {
            res[i] = st.top();
        }
        st.push(i);
    }
    return res;
}
// Next Smaller / Greater
vector<int> nextSmaller(const vector<int>&
a) {
    int n = a.size();
    vector<int> res(n, n);
    stack<int> st;
    for (int i = 0; i < n; i++) {
        while (!st.empty() && a[i] <
a[st.top()]) {
            res[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }
    return res;
}
```

3. DSU:

```
const int mx = 2e5+123;
struct Disjoint_set_union{
    int Parent[mx], Size[mx];
    void Init(int n){
        for (int i = 0; i <= n; i++){
            Parent[i] = i;
            Size[i] = 1;
        }
    }
    int Find(int v){
        if (v == Parent[v]) return v;
        // Path compression
        return Parent[v] =
Find(Parent[v]);
    }
    bool Union(int u, int v){
        int root_u = Find(u);
        int root_v = Find(v);
        if (root_u != root_v){
            if (Size[root_u] <
Size[root_v]){
                swap(root_u, root_v);
            }
            Parent[root_v] = root_u;
            Size[root_u] += Size[root_v];
            return true;
        }
        // 'u' and 'v' are already in the
same set
        return false;
    }
} DSU;

int main(){
    int n , m ;
    cin>> n >> m ;
    DSU.Init(n);
    while ( m-- ){
        int u , v ;
        cin>> u >> v ;
        DSU.Union( u , v );
    }
    int conCmp = 0 ;
    for ( int i = 1 ; i <= n ; i++ ){
        if ( DSU.Find(i) == i ){
            conCmp++;
        }
    }
    cout << conCmp << '\n';
}
```

4. GP Hash Table

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
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_epo
ch().count();
    return splitmix64(x + FIXED_RANDOM);
  }
};

gp_hash_table<int, int, custom_hash> mp;
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, x; cin >> n >> x;
  int a[n + 1];
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  for (int i = 1; i <= n; i++) {
    if (mp[x - a[i]]) {
      cout << mp[x - a[i]] << ' ' << i <<
'\n';
      return 0;
    }
    mp[a[i]] = i;
  }
  cout << "IMPOSSIBLE\n";
  return 0;
}
```

5. Order Set

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

template <typename T> using o_set =
tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
template <typename T, typename R> using
o_map = tree<T, R, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
int main() {
  int i, j, k, n, m;
  o_set<int>se;
  se.insert(1);
  se.insert(2);
  ///k th element
  cout << *se.find_by_order(0) << endl;
  ///number of elements less than k
  cout << se.order_of_key(2) << endl;
  o_map<int, int>mp;
  mp.insert({1, 10});
  mp.insert({2, 20});
  ///k th element
  cout << mp.find_by_order(0)->second <<
endl;
  ///number of first elements less than k
  cout << mp.order_of_key(2) << endl;
  return 0;
}
```

6. Segment Tree (Lazy)

```
const int mx = 1e5+123;
int v[mx] , t[4*mx] , lazy[4*mx];
void propagate(int node , int b , int e ){
    if (lazy[node] != 0 ){
        t[node] += (e-b+1)*lazy[node];
        int left = node*2;
        int right = node*2+1;
        if (left != right ){
            lazy[left] = lazy[node];
            lazy[right] = lazy[node];
```

```
        }
        lazy[node] = 0 ;
    }
}
int merge ( int leftNode , int rightNode
){
    return leftNode + rightNode;
}
void build ( int node , int b , int e ){
    lazy[node] = 0 ;
    if ( b == e ){
        t[node] = v[b];
        return ;
    }
    int left = node*2;
    int right = node*2+1;
    int mid = (b+e)/2;
    build(left , b , mid);
    build(right,mid+1,e);
    t[node] = merge(t[left] , t[right]);
}

void upd( int node , int b , int e , int i
, int j , int val ){
    propagate(node , b , e );
    if (j < b || i > e) return;
    if (i <= b && e <= j){
        lazy[node] = val ;
        propagate(node , b , e );
        return ;
    }
    int left = node*2;
    int right = node*2+1;
    int mid = (b+e)/2;
    upd(left,b,mid,i,j,val);
    upd(right,mid+1,e,i,j,val);
    t[node] = merge(t[left] , t[right]);
}

int query(int node, int b, int e, int i,
int j){
    propagate(node,b,e);
    if (j < b || i > e) return 0 ;
    if (i <= b && e <= j){
        return t[node];
    }
    int left = node*2;
    int right = node*2+1;
    int mid = (b+e)/2;
    int p1 = query(left , b , mid , i , j
) ;
    int p2 = query(right , mid+1 , e , i ,
j );
    return merge(p1,p2);
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> v[i];
    }
    build(1, 1, n);
    cout << query(1, 1, n, 2, 5) << '\n';
    upd(1, 1, n, 2, 5, 10);
    cout << query(1, 1, n, 2, 5) << '\n';
    return 0 ;
}
```

7. Sparse Table

```
const int N = 1e5 + 9;

int t[N][18], a[N];

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

int32_t main() {
  int n;
  cin >> n;
  for(int i = 1; i <= n; i++) cin >> a[i];
  build(n);
  int q; cin >> q;
  while(q--) {
    int l, r;
    cin >> l >> r;
    ++l; ++r;
    cout << query(l, r) << '\n';
  }
  return 0;
}
```

String

<u>String Hashing:</u>

```
const int N = 1e6 + 9;
int power(long long n, long long k, const
int mod){
    int ans = 1 % mod;
    n %= mod;
    if (n < 0) n += mod;
    while (k){
        if (k & 1) ans = (long long) ans *
n % mod;
        n = (long long) n * n % mod;
        k >>= 1;
    }
    return ans;
}
const int MOD1 = 127657753, MOD2 =
987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec(){
    pw[0] =  {1, 1};
    for (int i = 1; i < N; i++){
        pw[i].first = 1LL * pw[i -
1].first * p1 % MOD1;
        pw[i].second = 1LL * pw[i -
1].second * p2 % MOD2;
    }
    ip1 = power(p1, MOD1 - 2, MOD1);
    ip2 = power(p2, MOD2 - 2, MOD2);
    ipw[0] =  {1, 1};
    for (int i = 1; i < N; i++){
        ipw[i].first = 1LL * ipw[i -
1].first * ip1 % MOD1;
        ipw[i].second = 1LL * ipw[i -
1].second * ip2 % MOD2;
    }
}
struct Hashing{
    int n;
    string s; // 0 - indexed
    vector<pair<int, int>> hs; // 1 -
indexed
    Hashing() {}
    Hashing(string _s){
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (int i = 0; i < n; i++){
            pair<int, int> p;
            p.first = (hs[i].first + 1LL *
pw[i].first * s[i] % MOD1) % MOD1;
            p.second = (hs[i].second + 1LL
* pw[i].second * s[i] % MOD2) % MOD2;
            hs.push_back(p);
        }
    }
    pair<int, int> get_hash(int l, int r){
// 1 - indexed
        assert(1 <= l && l <= r && r <=
n);
        pair<int, int> ans;
        ans.first = (hs[r].first - hs[l -
1].first + MOD1) * 1LL * ipw[l - 1].first
% MOD1;
        ans.second = (hs[r].second - hs[l
- 1].second + MOD2) * 1LL * ipw[l -
1].second % MOD2;
        return ans;
    }
    pair<int, int> get_hash(){
        return get_hash(1, n);
    }
};
int32_t main()
{
    prec();
    int n;
    while (cin >> n){
        string s, p;
        cin >> p >> s;
        Hashing h(s);
        auto hs = Hashing(p).get_hash();
        for(int i = 1; i + n - 1 <=
s.size(); i++){
            if (h.get_hash(i, i + n - 1)
== hs) cout << i - 1 << '\n';
        }
        cout << '\n';
    }
    return 0;
}
```

Dynamic Programming

1. Basic / 1D DP

<u>Min/Max Cost to Reach End:</u>

a array given c[i] is the cost to step on the i-th stair.You can climb either 1 or 2 stairs. Find the minimum total cost to reach the top.

```
const int mx = 1e3+12;
int dp[mx] ;
int n ;
vector<int>c;
int yo(int ind) {
    if(ind == n-1) {
        return c[ind];
    }
    if(ind >= n) {
        return 0;
    }
    if(dp[ind] != -1) {
        return dp[ind];
    }
    int way1 = c[ind] + yo(ind+1);
    int way2 = c[ind] + yo(ind+2);
    return dp[ind] = min(way1, way2);
}
```

```
int main(){
    cin >> n ;
    c.resize(n);
    for ( int i = 0 ; i < n ; i++ ){
      cin >> c[i];
    }
    memset(dp , -1 , sizeof(dp));
    int mnCost = yo(0);
}
```

Maximum Sum of Non-Adjacent Elements:

```
int yo(int i) {
    if (i < 0) return 0;
    if (dp[i] != -1) return dp[i];
    int take = a[i] + yo(i-2);
    int notTake = yo(i-1);
    return dp[i] = max(take, notTake);
}
int main() {
    vector<int> a = {3, 2, 5, 10, 7};
    cout << f(a.size()-1);
}
```

Jump Game II:

Given an array where a[i] is the maximum jump length from that index, find the minimum number of jumps to reach the end.

```
int yo(int i, vector<int>& a, vector<int>& dp) {
    int n = a.size();
    if (i >= n-1) return 0;
    if (dp[i] != -1) return dp[i];
    int ans = 1e9;
    for (int j = 1; j <= a[i]; j++)
        ans = min(ans, 1 + yo(i+j, a, dp));
    return dp[i] = ans;
}
int main() {
    vector<int> a = {2,3,1,1,4};
    vector<int> dp(a.size(), -1);
    cout << yo(0, a, dp);
}
```

Coin Change (Number of Ways):

You are given coins and a target sum. Count the number of ways to make the sum using unlimited coins.

```
#define int long long
const int mx = 1e4+12;
int dp[100][mx];
vector<int>coins;
int n , target;

int yo(int i , int sum ) {
    if (sum == 0) return 1;
    if (i == 0) return (sum % coins[0] == 0);
    if (dp[i][sum] != -1) return dp[i][sum];

    int notTake = yo(i-1, sum);
    int take = 0;
    if (coins[i] <= sum) take = yo(i, sum - coins[i]);
    return dp[i][sum] = take + notTake;
}
int main() {
    cin >> n ;
    coins.resize(n);
    for ( int i = 0 ; i < n ; i++ ){
      cin >> coins[i];
    }
    memset(dp , -1 , sizeof(dp));
    cout << yo(n-1 , target);
}
```

Partition Equal Subset Sum:

Given an array, determine if it can be partitioned into two subsets with equal sum.

```
#define int long long
const int mx = 1e5 + 5;
int dp[105][mx];
vector<int> a;
int n, target;
bool f(int i, int target) {
    if (target == 0) return true;
    if (i == 0) return a[0] == target;
    if (dp[i][target] != -1) return dp[i][target];
    bool notTake = f(i - 1, target);
    bool take = false;
    if (a[i] <= target) {
      take = f(i - 1, target - a[i]);
    }
    return dp[i][target] = take or notTake;
}
int32_t main() {
    a = {1, 5, 11, 5};
    n = a.size();
    int sum = accumulate(a.begin(), a.end(), 0);
    if (sum % 2) cout << "No";
    else {
        target = sum / 2;
        memset(dp, -1, sizeof(dp));
        cout << (f(n - 1, target) ? "Yes" : "No");
    }
}
```

Count Ways to Reach Nth Stair:

You can climb 1 or 2 steps at a time. Find the number of distinct ways to reach the n-th stair.

```
#include <bits/stdc++.h>
using namespace std;
int f(int n, vector<int>& dp) {
    if (n == 0 || n == 1) return 1;
    if (dp[n] != -1) return dp[n];
    return dp[n] = f(n-1, dp) + f(n-2, dp);
}
int main() {
    int n = 5;
    vector<int> dp(n+1, -1);
    cout << f(n, dp);
}
```

2. Knapsack Variants

0/1 Knapsack:

You are given n items with values and weights. Each item can be taken at most once.
Find the maximum value you can achieve with total weight ≤ W.

```
#define int long long
const int mx = 1e3 + 5;
int dp[105][mx];
vector<int> wt, val;
int n, W;
int yo(int i, int bag) {
    if (i == n) return 0;
    if (dp[i][bag] != -1) return dp[i][bag];
    int notTake = yo(i + 1, bag);
    int take = 0;
    if (wt[i] <= bag) {
      take = val[i] + yo(i + 1, bag - wt[i]);
    }
    return dp[i][bag] = max(take, notTake);
}
int32_t main() {
    cin >> n >> W;
    wt.resize(n); val.resize(n);
    for (int i = 0; i < n; i++) cin >> wt[i];
    for (int i = 0; i < n; i++) cin >> val[i];
    memset(dp, -1, sizeof(dp));
    cout << yo(0, W);
}
```

Unbounded Knapsack:

You have n items; each item can be taken any number of times.
Find the maximum value achievable within weight W.

```
#define int long long
const int mx = 1e3 + 5;
int dp[105][mx];
vector<int> wt, val;
int n, W;
int yo(int i, int bag) {
    if (i == n) return 0;
    if (dp[i][bag] != -1) return dp[i][bag];
    int notTake = yo(i + 1, bag);
    int take = 0;
    if (wt[i] <= bag) take = val[i] + yo(i, bag - wt[i]);
    return dp[i][bag] = max(take, notTake);
}
int32_t main() {
    cin >> n >> W;
    wt.resize(n); val.resize(n);
    for (int i = 0; i < n; i++) cin >> wt[i];
    for (int i = 0; i < n; i++) cin >> val[i];
    memset(dp, -1, sizeof(dp));
    cout << yo(0, W);
}
```

Count number of Subsets with Given Sum:

```
#define int long long
const int mx = 1e5 + 5;
int dp[105][mx];
vector<int> a;
int n, target;
int yo(int i, int sum) {
    if (sum == 0) return 1;
    if (i == n) return 0;
    if (dp[i][sum] != -1) return dp[i][sum];
    int notTake = yo(i + 1, sum);
    int take = 0;
    if (a[i] <= sum) take = yo(i + 1, sum - a[i]);
    return dp[i][sum] = take + notTake;
}
int32_t main() {
    cin >> n >> target;
    a.resize(n);
    for (int i = 0; i < n; i++) cin >> a[i];
    memset(dp, -1, sizeof(dp));
    cout << yo(0, target);
}
```

3. DP on Strings

Longest Common Subsequence (LCS):

```
#define int long long
const int mx = 1005;
int dp[mx][mx];
```

```
string s1, s2;
int n, m;
int yo(int i, int j) {
    if (i == n || j == m) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    if (s1[i] == s2[j]) return dp[i][j] =
1 + yo(i + 1, j + 1);
    return dp[i][j] = max(yo(i + 1, j),
yo(i, j + 1));
}
int32_t main() {
    cin >> s1 >> s2;
    n = s1.size(); m = s2.size();
    memset(dp, -1, sizeof(dp));
    cout << yo(0, 0);
}
```

Longest Palindromic Subsequence:

```
#define int long long
const int mx = 1005;
int dp[mx][mx];
string s;
int n;
int yo(int i, int j) {
    if (i > j) return 0;
    if (i == j) return 1;
    if (dp[i][j] != -1) return dp[i][j];
    if (s[i] == s[j]) return dp[i][j] = 2
+ yo(i + 1, j - 1);
    return dp[i][j] = max(yo(i + 1, j),
yo(i, j - 1));
}
int32_t main() {
    cin >> s;
    n = s.size();
    memset(dp, -1, sizeof(dp));
    cout << yo(0, n - 1);
}
```

Edit Distance (Levenshtein Distance):

Given strings s1 and s2, find the minimum number of insertions, deletions, and replacements to convert s1 to s2.

```
#define int long long
const int mx = 1005;
int dp[mx][mx];
string s1, s2;
int n, m;
int yo(int i, int j) {
    if (i == n) return m - j; // insert
remaining chars of s2
    if (j == m) return n - i; // delete
remaining chars of s1
    if (dp[i][j] != -1) return dp[i][j];
    if (s1[i] == s2[j]) return dp[i][j] =
f(i + 1, j + 1);
    int insertOp = 1 + yo(i, j + 1);
    int deleteOp = 1 + yo(i + 1, j);
    int replaceOp = 1 + yo(i + 1, j + 1);
    return dp[i][j] = min({insertOp,
deleteOp, replaceOp});
}
int32_t main() {
    cin >> s1 >> s2;
    n = s1.size(); m = s2.size();
    memset(dp, -1, sizeof(dp));
    cout << yo(0, 0);
}
```

Count of Distinct Subsequences:

Given strings s and t, count the number of distinct subsequences of s equal to t.

```
#define int long long
const int mx = 1005;
int dp[mx][mx];
string s, t;
int n, m;
int yo(int i, int j) {
    if (j == m) return 1; // t completely
matched
    if (i == n) return 0; // s exhausted
    if (dp[i][j] != -1) return dp[i][j];
    int notTake = yo(i + 1, j);
    int take = 0;
    if (s[i] == t[j]) take = yo(i + 1, j +
1);
    return dp[i][j] = take + notTake;
}
int32_t main() {
    cin >> s >> t;
    n = s.size(); m = t.size();
    memset(dp, -1, sizeof(dp));
    cout << yo(0, 0);
}
```

4. DP on Subsequences / Subarrays

Longest Increasing Subsequence (LIS):

```
#define int long long
const int mx = 1005;
int dp[mx][mx];
vector<int> a;
int n;
int yo(int i, int prevInd) {
    if (i == n) return 0; // reached end
    if (dp[i][prevInd + 1] != -1) return
dp[i][prevInd + 1];
    int notTake = yo(i + 1, prevInd);
    int take = 0;
    if (prevInd == -1 || a[i] >
a[prevInd]) take = 1 + yo(i + 1, i);
    return dp[i][prevInd + 1] = max(take,
notTake);
}
int32_t main() {
    cin >> n;
    a.resize(n);
    for (int i = 0; i < n; i++) cin >>
a[i];
    memset(dp, -1, sizeof(dp));
    cout << yo(0, -1);
}
```

Maximum Sum Subarray (Kadane's Algorithm):

```
#define int long long
const int mx = 1005;
int dp[mx];
vector<int> a;
int n;
int yo(int i) {
    if (i < 0) return 0;
    if (dp[i] != -1) return dp[i];
    int take = a[i] + max(0LL, yo(i - 1));
    return dp[i] = take;
}
int32_t main() {
    cin >> n;
    a.resize(n);
    for (int i = 0; i < n; i++) cin >>
a[i];
    memset(dp, -1, sizeof(dp));
    int ans = LLONG_MIN;
    for (int i = 0; i < n; i++) {
      ans = max(ans, yo(i));
    }
    cout << ans;
}
```

5. Dp on Grid/Path

Unique Paths II (with obstacles):

Given an n x m grid with obstacles (1 = blocked, 0 = free), count the number of unique paths from top-left (0,0) to bottom-right (n-1,m-1) moving only down or right.

```
#define int long long
const int mx = 105;
int dp[mx][mx];
int n, m;
vector<vector<int>> grid;
int yo(int i, int j) {
    if (i < 0 || j < 0 || grid[i][j] == 1)
return 0;
    if (i == 0 && j == 0) return 1;
    if (dp[i][j] != -1) return dp[i][j];
    return dp[i][j] = yo(i-1, j) + yo(i,
j-1);
}
int32_t main() {
    cin >> n >> m;
    grid.resize(n, vector<int>(m));
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            cin >> grid[i][j];
    memset(dp, -1, sizeof(dp));
    cout << yo(n-1, m-1);
}
```

Path in Matrix with Maximum Gold:

In a grid, each cell has gold. Start from any row in first column and move right, right-up, or right-down. Maximize gold collected reaching the last column.

```
#define int long long
const int mx = 105;
int dp[mx][mx];
int n, m;
vector<vector<int>> grid;
int f(int i, int j) {
    if (i < 0 || i >= n) return LLONG_MIN;
    if (j == m-1) return grid[i][j];
    if (dp[i][j] != -1) return dp[i][j];
    int right = grid[i][j] + f(i, j+1);
    int rightUp = grid[i][j] + f(i-1,
j+1);
    int rightDown = grid[i][j] + f(i+1,
j+1);
    return dp[i][j] = max({right, rightUp,
rightDown});
}
int32_t main() {
    cin >> n >> m;
    grid.resize(n, vector<int>(m));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> grid[i][j];
        }
    }
    memset(dp, -1, sizeof(dp));
    int ans = 0;
    for(int i = 0; i < n; i++) {
        ans = max(ans, f(i, 0));
    }
    cout << ans;
}
```

6. Bitmask Dp

-There are N ranks and M students.
-Each student i has a happiness value happy[i][j] for getting rank j.
-Each rank can be assigned to at most one student; some students may get no rank (0 happiness).
-Find the maximum total happiness by assigning students to ranks.

-Constraints: 1 ≤ N ≤ 10, 1 ≤ M ≤ 50, 0 ≤ happy[i][j] ≤ 10^5.

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

#define int long long
const int MAXM = 50;
const int MAXN = 10;
int m, n;
int arr[MAXM][MAXN];
int dp[MAXM][1 << MAXN];

int rec(int level, int mask) {
    if (level == m) {
        return 0;
    }
    if (dp[level][mask] != -1) return dp[level][mask];
    int ans = 0;
    ans = max(ans, rec(level + 1, mask));
    for (int i = 0; i < n; i++) {
        if ((mask & (1 << i)) == 0) {
            ans = max(ans, arr[level][i] + rec(level + 1, mask | (1 << i)));
        }
    }
    return dp[level][mask] = ans;
}

signed main() {
    int t ; cin >> t ;
    while ( t-- ){
        cin >> n >> m ;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                cin >> arr[i][j];
            }
        }
        memset(dp , -1 , sizeof(dp));
        int answer = rec(0, 0);
        cout << answer << "\n";
    }

    return 0;}
```

Miscellaneous

Cmp:

```
bool cmp(pair<int, int> a, pair<int, int> b){
    return a.second > b.second;
}
```

Interactive:

```
#define int long long
int ask(int i, int j) {
    cout << "? " << i << ' ' << j << endl;// ask
    int x;
    cin >> x; // ans
    return x;
}
signed main() {
    int a = ask(1, 2), b = ask(2, 3), c = ask(3, 4), d = ask(4, 5);
    vector<int> ans = {a, b, c, d};
    vector<int> p = {4, 8, 15, 16, 23, 42};

    do {
        bool f = true;
        for (int i = 0; i < 4; i++) {
            if (p[i] * p[i + 1] != ans[i]) {
                f = false;
                break;
            }
        }
        if (f) {
            cout << "! ";
            for (auto x : p) {
                cout << x << ' ';
            }
            cout << endl;
            break;
        }
    } while (next_permutation(p.begin(), p.end()));

    return 0;
}
```

Big Int__128:

```
void printInt128(__int128 num)
{
    if (num == 0){
        cout << 0;
        return;
    }
    if (num < 0){
        cout << '-';
        num = -num;
    }
    string res;
    while (num > 0){
        res += (char)('0' + (num % 10));
        num /= 10;
    }
    reverse(res.begin(), res.end());
    cout << res;
}

__int128 readInt128(){
    string s;
    cin >> s;
    __int128 num = 0;
    bool neg = false;
    int i = 0;
    if (s[0] == '-')
    {
        neg = true;
        i = 1;
    }
    for (; i < s.size(); i++){
        num = num * 10 + (s[i] - '0');
    }
    return neg ? -num : num;
}
int main()
{
    __int128 a = readInt128();
    __int128 b = readInt128();

    printInt128(__gcd( a , b ));
    cout << '\n';
    return 0;
}
```

Intersect Checked on point:

```
vector< pair < int, int >> pr ;
int mx_l = -1, mn_r = INT_MAX;
for ( int i = 0 ; i < n ; i++ )
{
 int l, r ;
    cin>> l >>  r ;
    pr.push_back({ l, r });
    mx_l = max( mx_l, l );
    mn_r = min( mn_r, r );
}
intersect : mx_l <= mn_r
else no intersect
```

MEX:

```
const int mx = 2e5 + 5;
vector<int> A;
map<int, int> freq;
set<int> missing;
void build(const vector<int>& v) { // nlogn (pre compute)
    A = v;
    freq.clear();
    missing.clear();
    int n = A.size();
    for (int i = 0; i <= n; i++) {
        missing.insert(i);
    }
    for (int x : A) {
        freq[x]++;
        missing.erase(x);
    }
}
int get() { // O(1)
    return *missing.begin();
}
void update(int idx, int new_val) { // logn
    int old_val = A[idx];

    freq[old_val]--;
    if (freq[old_val] == 0) {
        missing.insert(old_val);
    }
    A[idx] = new_val;
    freq[new_val]++;
    missing.erase(new_val);
}

int main() {
    int n, q;
    cin >> n >> q;
    vector<int> v(n);
    for (int i = 0; i < n; i++) {
        cin >> v[i];
    }
    build(v);
    while (q--) {
        int type;
        cin >> type;
        if (type == 1) {
            cout << get() << '\n';
        } else if (type == 2) {
            int idx, val;
            cin >> idx >> val;
            update(idx, val);
        }
    }
    return 0;
}
```

Stress Testing:

```
void generate_test(vector<int> &a, int &n)
{
    n = rand() % 10 + 1;
    a.resize(n);
    for (int i = 0; i < n; i++) {
        a[i] = rand() % 20;
    }
}
long long brute(const vector<int> &a, int n) {
    long long sum = 0;
    for (int i = 0; i < n; i++) sum += a[i];
    return sum;
}
long long fast(const vector<int> &a, int n) {
    return accumulate(a.begin(), a.end(), 0LL);
}
int main() {
    srand(time(0));
```

```
    for (int tc = 1; tc <= 1000000; tc++)
{
        int n;
        vector<int> a;
        generate_test(a, n);
        long long ans1 = brute(a, n);


        long long ans2 = fast(a, n);
        if (ans1 != ans2) {
            cout << "Mismatch at test " <<
tc << ":\n";
            cout << "n = " << n << "\n";
            cout << "a = ";
            for (int x : a) cout << x << "
";
            cout << "\n";
            cout << "brute = " << ans1 <<
"\n";
            cout << "fast  = " << ans2 <<
"\n";
            return 0;
        }
        if (tc % 10000 == 0)
            cout << "Passed " << tc << "
tests\n";
    }
    cout << "All tests passed!\n";
    return 0;
}
```