二分搜索

1. 二分搜索

```
ll search_exit(const ll arr[], ll size, ll num)
{
    if (arr == nullptr || size == 0)
    {
        return -1;
    }

    ll l = 0;
    ll r = size - 1;
    while (l <= r)
    {
        ll mid = l + (r - l) / 2;
        if (arr[mid] == num)
        {
            return mid;
        }
        else if (arr[mid] < num)
        {
            l = mid + 1;
        }
        else
        {
            r = mid - 1;
        }
    }
    return -1;
}

ll search_exit_correct(const ll arr[], ll size, ll num)
{
    for (ll i = 0; i < size; ++i)
    {
        if (arr[i] == num)
        {
            return i;
        }
    }
    return -1;
}

vector<ll> Random_Array(ll n, ll v)
{
    vector<ll> arr(n);
    for (ll i = 0; i < n; ++i)
    {
        arr[i] = static_cast<ll>(static_cast<double>(rand()) / (RAND_MAX + 1.0) * v) + 1;
    }
    return arr;
}

void check_search_exit()
{
    srand(time(0));

    ll N = 10;
    ll V = 100;
    ll times = 100000;

    for (ll i = 0; i < times; ++i)
    {
        ll n = rand() % (N + 1);
        vector<ll> arr = Random_Array(n, V);
        sort(arr.begin(), arr.end());

        ll num_to_search = static_cast<ll>(static_cast<double>(rand()) / (RAND_MAX + 1.0) * V) + 1;
```

```
            ll result_exit = search_exit(arr.data(), arr.size(), num_to_search);
            ll result_correct = search_exit_correct(arr.data(), arr.size(), num_to_search);

            if (result_exit != result_correct)
            {
                cout << "Error Found on Test " << (i + 1) << "!" << endl;
                cout << "Array Size: " << arr.size() << endl;
                cout << "Array Elements: ";
                for (ll val : arr)
                {
                    cout << val << " ";
                }
                cout << endl;
                cout << "Number to Search: " << num_to_search << endl;
                cout << "Binary Search Result: " << result_exit << endl;
                cout << "Linear Search Result: " << result_correct << endl;
                return;
            }
        }
        cout << "All " << times << " tests passed successfully!" << endl;
}

// search the leftmost position of an element greater than or equal to num
ll search_greater_equal(const ll arr[], ll size, ll num)
{
    if (arr == nullptr || size == 0)
    {
        return -1;
    }

    ll l = 0;
    ll r = size - 1;
    ll ans = -1;
    while (l <= r)
    {
        ll mid = (l + r) / 2;
        // ll min = l + (r - l) / 2; // prevent overflow
        if (arr[mid] >= num)
        {
            ans = mid; // record the position
            r = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }
    return ans;
}

// search the rightmost position of an element less than or equal to num
ll search_less_equal(const ll arr[], ll size, ll num)
{
    if (arr == nullptr || size == 0)
    {
        return -1;
    }

    ll l = 0;
    ll r = size - 1;
    ll ans = -1;
    while (l <= r)
    {
        ll mid = (l + r) / 2;
        if (arr[mid] <= num)
        {
```

```
            ans = mid; // record the position
            l = mid + 1;
        }
        else
        {
            r = mid - 1;
        }
    }
    return ans;
}

// any two non-adjacent numbers are not equal, return a peak element ('small - large - small').
ll search_peak(const ll arr[], ll size)
{
    if (size == 1)
    {
        // assume arr[-1] and arr[size] are very small.
        return 0;
    }

    // size >= 2, verify if arr[0] and arr[n-1] are peak elements.
    if (arr[0] > arr[1])
    {
        return 0;
    }
    if (arr[size - 1] > arr[size - 2])
    {
        return size - 1;
    }

    // based on the function's trend, [1, size-2] must contain at least one peak.
    ll l = 1, r = size - 2, ans = -1;
    while (l <= r)
    {
        ll mid = (l + r) / 2;
        if (arr[mid] < arr[mid - 1])
        {
            r = mid - 1;
        }
        else if (arr[mid] < arr[mid + 1])
        {
            l = mid + 1;
        }
        else
        {
            ans = mid; // arr[mid] is a peak element.
            break;
        }
    }
    return ans;
}
signed main()
{
    check_search_exit();

    return 0;
}
```

2. 二分答案法

```
// array length n, partition it into k contiguous subarrays, find the partition method that minimizes the maximum
sum among all subarrays (arr[i] >= 0).
ll f(const vector<ll> &arr, ll limit)
{
    ll k = 1, sum = 0;

    for (ll num : arr)
    {
```

```
            if (num > limit)
            {
                return LLONG_MAX;
            }

            if (sum + num > limit)
            {
                k++;
                sum = num;
            }
            else
            {
                sum += num;
            }
        }

        return k;
}
ll par_min(const vector<ll> &arr, ll k)
{
        ll sum = 0, ans = 0;
        for (ll num : arr)
        {
            sum += num;
        }

        ll l = 0, r = sum, m;
        while (l <= r)
        {
            m = (l + r) / 2;
            if (f(arr, m) <= k)
            {
                ans = m;
                r = m - 1;
            }
            else
            {
                l = m + 1;
            }
        }

        return ans;
}

// the k-th smallest absolute difference between any two numbers.
ll f(const vector<ll> &arr, ll limit)
{
        ll n = arr.size(), ans = 0;
        for (ll l = 0, r = 0; l < n; l++)
        {
            while (r + 1 < n && arr[r + 1] - arr[l] <= limit)
            {
                r++;
            }
            ans += (r - l);
        }

        return ans;
}
ll sma_abs(vector<ll> &arr, ll k)
{
        ll n = arr.size(), ans;
        sort(arr.begin(), arr.end());

        ll l = 0, r = arr[n - 1] - arr[0], m;
        while (l <= r)
        {
```

```
        // arr = {1, 3, 5, 8, 10}
        // 2, 2, 2, 3, 4, 5, 5, 7, 7, 9
        m = (l + r) / 2;
        if (f(arr, m) >= k)
        {
            ans = m;
            r = m - 1;
        }
        else
        {
            l = m + 1;
        }
    }

    return ans;
}

// battery pack powers n computers, return the maximum amount of time the n computers can run simultaneously (the battery switching speed is negligible).
bool f(const vector<ll> &batt, ll n, ll minute)
{
    // Conclusion 1: If a single battery's capacity is greater than minute, it can independently power one computer.

    // Conclusion 2 : The total capacity of the remaining "fragmented" batteries only needs to be greater than or equal to the number of remaining computers multiplied by minute.

    ll sum = 0;
    for (ll x : batt)
    {
        if (x > minute)
        {
            n--;
        }
        else
        {
            sum += x;
        }

        if (sum >= n * minute)
        {
            return true;
        }
    }

    return false;
}
ll max_cpu(const vector<ll> &batt, ll n)
{
    ll sum = 0, max = 0;
    for (ll x : batt)
    {
        sum += x;
        max = std::max(x, max);
    }

    if (sum > max * n)
    {
        return sum / n;
    }

    ll l = 0, r = max, m, ans = 0;
    while (l <= r)
    {
        m = (l + r) / 2;
        if (f(batt, n, m))
        {
```

```
            ans = m;
            l = m + 1;
        }
        else
        {
            r = m - 1;
        }
    }

    return ans;
}

// group of bath attendants, each with a fixed scrubbing time, assuming m people are waiting, and everyone follows the principle of taking an available spot as soon as it's free, what is the minimum waiting time?
ll f(const vector<ll> &bath, ll tim)
{
    ll sum = 0;
    for (ll ser : bath)
    {
        sum += (tim / ser + 1);
    }

    return sum;
}
ll min_bath(const vector<ll> &bath, ll m)
{
    // can be analyzed that if a person has multiple choices, the final answer is independent of their choice, as long as they follow the principle of taking an available spot as soon as it's free.
    ll ans, min = LLONG_MAX;
    for (ll tim : bath)
    {
        min = std::min(tim, min);
    }
    ll l = 0, r = m * min, mid;
    while (l <= r)
    {
        mid = (l + r) / 2;
        if (f(bath, mid) >= m + 1)
        {
            ans = mid;
            r = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }

    return ans;
}

// monster has hp health, each turn, you can choose to either use a sword slash or apply poison.
// sword causes immediate damage in the current turn and poison causes continuous damage starting from the next turn, and it can be stacked, return the minimum number of turns to kill the monster (n turns are available for actions, if the monster is not killed after n turns, it will continue to take poison damage until it dies)?
bool f(const vector<ll> &sword, const vector<ll> &pois, ll hp, ll m){
    ll len = sword.size();
    ll n = min(len, m);
    for (ll i = 0, j = 1; i < n; i++, j++)
    {
        hp -= max(pois[i] * (m - j), sword[i]);

        if (hp <= 0)
        {
            return true;
        }
    }
```

```
    return false;
}
ll kill_mon(const vector<ll> &sword, const vector<ll> &pois, ll hp){
    ll ans = LLONG_MAX;
    ll l = 1, r = hp + 1, m;
    while (l <= r)
    {
        m = (l + r) / 2;
        if (f(sword, pois, hp, m))
        {
            ans = m;
            r = m - 1;
        }
        else
        {
            l = m + 1;
        }
    }
    return ans;
}
```

求“最大化最小值”或“最小化最大值”时，非常强烈的信号，暗示可以使用二分答案！
将数组 a 切分成 k 个连续子数组，希望让这 k 个子数组中 MEX 值最小的那个尽可能大。

```
vl a(maxn);
// mex = m, [0...m-1] ✓
bool check(ll m, ll n, ll k){
    // cout << "m: " << m << endl;
    if (m == 0)
        return true;
    vl vis(m, 0);
    for (ll i = 0, cnt = 0; i < n; i++){
        if (a[i] < m && vis[a[i]] == 0){
            vis[a[i]] = 1;
            cnt++;
            if (cnt == m){
                fill(vis.begin(), vis.end(), 0);
                cnt = 0;
                k--;
                if (k == 0)
                {
                    return true;
                }
            }
        }
    }
    return false;
}
void solve(){
    ll n, k;
    cin >> n >> k;
    for (ll i = 0; i < n; i++) cin >> a[i];
    ll ans = 0;
    ll l = 0, r = n;
    while (l <= r){
        ll m = (l + r) / 2;
        if (check(m, n, k)){
            ans = m;
            l = m + 1;
        }
        else{
            r = m - 1;
        }
    }
    cout << ans << endl;
}
```

二进制与位操作

1. 位操作实现加减乘除

```
// support addition using bitwise operations.
ll add_using_bitwise(ll a, ll b)
{
    while (b != 0)
    {
        ll carry = a & b;
        a = a ^ b; // sum without carry
        b = carry << 1;
    }
    return a;
}

// support subtraction using bitwise operations.
ll subtract_using_bitwise(ll a, ll b)
{
    ll neg_b = add_using_bitwise(~b, 1); // negation of b
    return add_using_bitwise(a, neg_b);
}

// support multiplication using bitwise operations.
ll multiply_using_bitwise(ll a, ll b)
{
    ll result = 0;
    bool negative = (a < 0) ^ (b < 0);

    if (a < 0)
        a = add_using_bitwise(~a, 1);
    if (b < 0)
        b = add_using_bitwise(~b, 1);

    while (b > 0)
    {
        if (b & 1)
            result = add_using_bitwise(result, a);
        a <<= 1;
        b >>= 1;
    }

    return negative ? add_using_bitwise(~result, 1) : result;
}

// support division using bitwise operations.
ll divide_using_bitwise(ll a, ll b)
{
    bool negative = (a < 0) ^ (b < 0);

    ll abs_a = (a < 0) ? add_using_bitwise(~a, 1) : a;
    ll abs_b = (b < 0) ? add_using_bitwise(~b, 1) : b;

    ll ans = 0;
    for (int i = 63; i >= 0; i = subtract_using_bitwise(i, 1))
    {
        if ((abs_a >> i) >= abs_b)
        {
            abs_a = subtract_using_bitwise(abs_a, abs_b << i);
            ans |= (1LL << i);
        }
    }

    return negative ? add_using_bitwise(~ans, 1) : ans;
}

signed main()
{
    ll a = 2, b = -7;
```

```
    cout << "addition: " << add_using_bitwise(a, b) << endl;
    cout << "subtraction: " << subtract_using_bitwise(a, b) << endl;
    cout << "multiplication: " << multiply_using_bitwise(a, b) << endl;
    cout << "division: " << divide_using_bitwise(a, b) << endl;
    return 0;
}

2. 二进制基础
void print_binary(ll num)
{
    for (ll i = 63; i >= 0; --i)
    {
        // 1LL means the one is treated as a long long, ensuring that the left shift operation does not overflow.
        cout << (num & (1LL << i) ? '1' : '0');
    }
    cout << endl;
}

signed main()
{
    // two's complement: positive → positive ; negative → positive → -1 → inverse
    // eg: -8 → 8 → 1000 → 0111 → 1000
    // value: positive → positive ; negative → inverse → +1 → value → sign
    // eg: 1000 → 0111 → 1000 → 8 → -8

    ll a = 0b1000; // 8
    ll b = 0b1111; // 15
    ll c = 0x7;    // 7
    ll d = 0xF;    // 15
    cout << a << " " << b << " " << c << " " << d << endl;
    cout << "Binary Representation:" << endl;
    print_binary(a);
    print_binary(b);
    print_binary(c);
    print_binary(d);

    // >>; >>>
    // Arithmetic Right Shift; Logical Right Shift
    cout << "Arithmetic Right Shift:" << endl;
    print_binary(a >> 1);
    print_binary(b >> 1);
    print_binary(c >> 1);
    print_binary(d >> 1);
    cout << "Logical Right Shift:" << endl;
    print_binary(static_cast<unsigned long long>(a) >> 1);
    print_binary(static_cast<unsigned long long>(b) >> 1);
    print_binary(static_cast<unsigned long long>(c) >> 1);
    print_binary(static_cast<unsigned long long>(d) >> 1);

    // Opposite Number(except for the smallest negative numbers)
    cout << "Opposite Number:" << endl;
    cout << "-a: " << ~a + 1 << ", -b: " << ~b + 1 << ", -c: " << ~c + 1 << ", -d: " << ~d + 1 << endl;
    print_binary(~a + 1);
    print_binary(~b + 1);
    print_binary(~c + 1);
    print_binary(~d + 1);

    // AND, OR, XOR
    cout << "AND:" << endl;
    print_binary(a & b);
    cout << "OR:" << endl;
    print_binary(a | b);
    cout << "XOR:" << endl;
    print_binary(a ^ b);

    return 0;
}
```

3. 位运算

```
bool power_of_two(ll n)
{
    return n > 0 && n == (n & -n);
}

ll min_power_of_two(ll n)
{
    // return the smallest power of two greater than or equal to n
    if (n <= 0)
        return 1;

    n--;
    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16; // for 32-bit
    n |= n >> 32; // for 64-bit
    return n + 1;
}

ll and_of_all_nums(ll left, ll right)
{
    while (left < right)
    {
        right -= (right & -right); // clear the lowest set bit of right
    }
    return right;
}

ll reversed_binary(ll n)
{
    n = (n & 0x5555555555555555LL) << 1 | (n & 0xAAAAAAAAAAAAAAAALL) >> 1; // 1 bit groups
    n = (n & 0x3333333333333333LL) << 2 | (n & 0xCCCCCCCCCCCCCCCCLL) >> 2; // 2 bit groups
    n = (n & 0x0F0F0F0F0F0F0F0FLL) << 4 | (n & 0xF0F0F0F0F0F0F0F0LL) >> 4; // 4 bit groups
    n = (n & 0x00FF00FF00FF00FFLL) << 8 | (n & 0xFF00FF00FF00FF00LL) >> 8; // 8 bit groups (for 32-bit)
    n = (n & 0x0000FFFF0000FFFFLL) << 16 | (n & 0xFFFF0000FFFF0000LL) >> 16; // 16 bit groups (for 64-bit)

    // 32 bit groups
    n = (n >> 32) | (n << 32); // n = (n >> 16) | (n << 16); // for 32-bit, n > 0

    return n;
}

ll nums_of_one(ll n)
{
    n = (n & 0x5555555555555555LL) + ((n >> 1) & 0x5555555555555555LL);
    n = (n & 0x3333333333333333LL) + ((n >> 2) & 0x3333333333333333LL);
    n = (n & 0x0F0F0F0F0F0F0F0FLL) + ((n >> 4) & 0x0F0F0F0F0F0F0F0FLL);
    n = (n & 0x00FF00FF00FF00FFLL) + ((n >> 8) & 0x00FF00FF00FF00FFLL);
    n = (n & 0x0000FFFF0000FFFFLL) + ((n >> 16) & 0x0000FFFF0000FFFFLL);
    n = (n & 0x00000000FFFFFFFFLL) + ((n >> 32) & 0x00000000FFFFFFFFLL);
    return n;
}

//  1  4 13 16
//  3  2 15 14
//  9 12  5  8
// 11 10  7  6 ...
//  ...
void solve()
{
    ll n;
    cin >> n;
    ll q;
```

```
cin >> q;
while (q--)
{
    string s;
    cin >> s;

    if (s == "->")
    {
        ll a, b;
        cin >> a >> b;

        ll row = a - 1;
        ll col = b - 1;
        ll ans = 0;

        // from the most significant level (n-1) down to 0
        // n = 2, (0, 0) to (3, 3), also (00, 00) to (11, 11)
        for (ll i = n - 1; i >= 0; --i)
        {
            // the i-th bit from row and column
            ll row_bit = (row >> i) & 1;
            ll col_bit = (col >> i) & 1;

            ll dg = 0;

            if (row_bit == 0 && col_bit == 0)
                dg = 0; // Top-Left
            else if (row_bit == 1 && col_bit == 1)
                dg = 1; // Bottom-Right
            else if (row_bit == 1 && col_bit == 0)
                dg = 2; // Bottom-Left
            else if (row_bit == 0 && col_bit == 1)
                dg = 3; // Top-Right

            // ans = 4 * ans + dg
            ans = (ans << 2) | dg;
        }
        cout << ans + 1 << endl;
    }
    else
    {
        ll d;
        cin >> d;

        ll val = d - 1;
        ll a = 0, b = 0;

        for (ll i = n - 1; i >= 0; --i)
        {
            // 010000 -> 01
            ll dg = (val >> (2 * i));
            // 010000 & 001111 -> 0000
            val &= (1LL << (2 * i)) - 1;

            ll row_bit = 0, col_bit = 0;
            if (dg == 0)
            {
                row_bit = 0;
                col_bit = 0;
            } // Top-Left
            else if (dg == 1)
            {
                row_bit = 1;
                col_bit = 1;
            } // Bottom-Right
            else if (dg == 2)
            {
```

```
                    row_bit = 1;
                    col_bit = 0;
                } // Bottom-Left
                else if (dg == 3)
                {
                    row_bit = 0;
                    col_bit = 1;
                } // Top-Right

                a = (a << 1) | row_bit;
                b = (b << 1) | col_bit;
            }
            cout << a + 1 << " " << b + 1 << endl;
        }
    }
}

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

    cout << min_power_of_two(-9) << endl;
    cout << and_of_all_nums(13, 15) << endl;
    cout << reversed_binary(13LL) << endl;
    cout << nums_of_one(13LL) << endl; // 3

    return 0;
}
```

4. 位图

```
const ll MAX_SIZE = 1000000;

struct bitmap
{
    vector<ll> set;

    void
    bitmap_set(ll n)
    {
        // set bitmap, supports "crud" operations for all numbers from 0 to n-1
        set.resize((n + 63) / 64, 0); // each element can store 64 bits
    }

    void insert(ll num)
    {
        set[num / 64] |= (1LL << (num % 64));
    }

    void erase(ll num)
    {
        set[num / 64] &= ~(1LL << (num % 64));
    }

    void reverse(ll num)
    {
        set[num / 64] ^= (1LL << (num % 64));
    }

    bool contains(ll num)
    {
        return (set[num / 64] & (1LL << (num % 64))) != 0;
    }
};

void test_bitmap()
{
```

```
    ll n = 1000, times = 10000;
    cout << "test start!" << endl;

    bitmap bm;
    bm.bitmap_set(n);
    set<ll> s;

    while (times--)
    {
        double randomdouble = static_cast<double>(rand()) / (RAND_MAX + 1.0);
        ll num = static_cast<ll>(randomdouble * n);

        if (randomdouble < 0.33)
        {
            bm.insert(num);
            s.insert(num);
        }
        else if (randomdouble < 0.66)
        {
            bm.erase(num);
            s.erase(num);
        }
        else
        {
            bm.reverse(num);
            if (s.count(num))
            {
                s.erase(num);
            }
            else
            {
                s.insert(num);
            }
        }
    }
    cout << "test end!" << endl;

    for (ll i = 0; i < n; i++)
    {
        if (bm.contains(i) != s.count(i))
        {
            cerr << "error: " << i << endl;
        }
    }
    cout << "test passed!" << endl;
}

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

    test_bitmap();

    return 0;
}
```

5. 异或

```
// Summary of XOR:
// 1. xor is essentially addition without carry.
// 2. xor satisfies commutative and associative laws, etc. (easily derived from property 1).
// 3. 0 ^ x = x (0 xor x equals x), x ^ x = 0 (x xor x equals 0).
// 4. from a ^ b = c, can be deduced that b ^ c = a and a ^ c = b (derived from properties 2 and 3).
// 5. a[i] = a[i] ^ a[i+1] <=> swap(pra[i], pra[i+1]), pra[i]=a[0]^a[1]^...^a[i]

void swap_xor(ll &a, ll &b)
{
```

```
    a = a ^ b; // a = a' ^ b', b = b'
    b = a ^ b; // b = (a' ^ b') ^ b' = a', a = a' ^ b'
    a = a ^ b; // a = (a' ^ b') ^ a' = b', b = a'
}

ll sign(ll num)
{
    return (num >> 63) ^ 1; // extract the sign bit (0 for negative, 1 for positive)
}

ll flip(ll n)
{
    return n ^ 1;
}

ll max_xor(ll &a, ll &b)
{
    ll c = a - b;
    ll returna = sign(c);
    // c < 0 → a < b, returna = 0; c > 0 → a > b, returna = 1
    ll returnb = flip(returna); // flip the sign, so 1 means a > b, 0 means a < b
    return a * returna + b * returnb;
}

ll max_xor_overflow(ll &a, ll &b)
{
    ll c = a - b;
    ll sa = sign(a);
    ll sb = sign(b);
    ll sc = sign(c);

    // diffab = 1, a and b have different signs; diffab = 0, a and b have the same sign
    ll diffab = sa ^ sb;
    ll sameab = flip(diffab);

    // diffab = 1 && sa = 1; diffab = 0, sc = 1
    ll returna = sa * diffab + sc * sameab;
    // diffab = 1 && sb = 1; diffab = 0, sc = 0
    ll returnb = flip(returna);

    return a * returna + b * returnb;
}

ll missing_num(vector<ll> &nums)
{
    ll n = nums.size();
    ll total_xor = 0;
    for (ll i = 0; i < n + 1; ++i)
    {
        total_xor ^= i; // xor from 0 to n
    }

    for (ll num : nums)
    {
        total_xor ^= num; // xor remain elements
    }

    return total_xor; // the missing number
}

ll odd_and_even(vector<ll> &nums)
{
    // only 'x' appears an odd number of times; all other numbers appear an even number of times, return x
    ll x = 0;
    for (ll num : nums)
    {
        x ^= num; // xor all numbers
```

```
    }
    return x;
}

ll brain_kernighan(ll x)
{
    return x & ((~x) + 1); // x & -x, isolate the rightmost 1 bit
}

void odd_and_even_plus(vector<ll> &nums)
{
    // only 'a' and 'b' appears an odd number of times; all other numbers appear an even number of times
    ll total_xor = 0, single_xor = 0;
    for (ll num : nums)
    {
        total_xor ^= num; // total_xor = a ^ b
    }

    ll rightmost_bit = brain_kernighan(total_xor);

    for (ll num : nums)
    {
        if (num & rightmost_bit) // group the numbers based on the rightmost bit
        {
            single_xor ^= num; // single_xor = a or b
        }
    }

    cout << single_xor << " " << (total_xor ^ single_xor); // b = total_xor ^ a
}

ll cnt_m(vector<ll> &nums, ll m)
{
    ll cnt[64] = {0}; // count of 1s of each bit position

    // only 'ans' appears less than 'm' times; all other numbers appear exactly 'm' times.
    ll ans = 0;
    for (ll num : nums)
    {
        for (ll i = 0; i < 64; ++i)
        {
            cnt[i] += (num >> i) & 1; // count the number of 1s on the i-th bit position
        }
    }

    for (ll i = 0; i < 64; ++i)
    {
        if (cnt[i] % m != 0)
        {
            ans |= (1LL << i);
        }
    }

    return ans;
}
```

比较器

1. 比较器

```
vector<ll> Random_Array(ll n, ll v)
{
    vector<ll> arr(n);
    for (ll i = 0; i < n; ++i)
    {
        // rand() generates an integer between 0 and RAND_MAX.
        // use static_cast<double>(rand()) / RAND_MAX to get a double in [0.0, 1.0)
        // static_cast<ll>(randomdouble * v) -> ll in [0, v) range, still uniformly distributed
        // static_cast<ll>(randomdouble * v) + 1 -> ll in [1, v] range, still uniformly distributed
        double randomdouble = static_cast<double>(rand()) / (RAND_MAX + 1.0);

        arr[i] = static_cast<ll>(randomdouble * v) + 1;
    }
    return arr;
}

vector<ll> Copy_Array(const vector<ll> &arr)
{
    vector<ll> copy(arr);
    return copy;
}

bool Same_Array(const vector<ll> &arr1, const vector<ll> &arr2)
{
    if (arr1.size() != arr2.size())
        return false;
    for (size_t i = 0; i < arr1.size(); ++i)
    {
        if (arr1[i] != arr2[i])
            return false;
    }
    return true;
}

void Selection_Sort(vector<ll> &arr)
{
    ll n = arr.size();
    for (ll i = 0; i < n - 1; ++i)
    {
        ll min_index = i;
        for (ll j = i + 1; j < n; ++j)
        {
            if (arr[j] < arr[min_index])
            {
                min_index = j;
            }
        }
        swap(arr[i], arr[min_index]);
    }
}
void Insertion_Sort(vector<ll> &arr)
{
    ll n = arr.size();
    for (ll i = 1; i < n; ++i)
    {
        ll key = arr[i];
        ll j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

```
signed main()
{
    ll N = 10;   // max size of the array
    ll V = 100; // range of values
    ll times = 1000;
    while (times--)
    {
        ll n = rand() % N; // random size of the array between 0 and N-1
        vector<ll> arr = Random_Array(n, V);

        // cout << "Random Array: ";
        // for (ll i = 0; i < n; ++i)
        // {
        //       cout << arr[i] << " ";
        // }
        // cout << endl;

        vector<ll> arr_copy_1 = Copy_Array(arr);
        vector<ll> arr_copy_2 = Copy_Array(arr);
        Selection_Sort(arr_copy_1);
        Insertion_Sort(arr_copy_2);
        if (!Same_Array(arr_copy_1, arr_copy_2))
        {
            cout << "Error!" << endl;
            cout << "Original Array: ";
            for (ll i = 0; i < n; ++i)
            {
                cout << arr[i] << " ";
            }
            cout << endl;

            cout << "Selection Sort Result: ";
            for (ll i = 0; i < n; ++i)
            {
                cout << arr_copy_1[i] << " ";
            }
            cout << endl;

            cout << "Insertion Sort Result: ";
            for (ll i = 0; i < n; ++i)
            {
                cout << arr_copy_2[i] << " ";
            }
            cout << endl;
        }
        else
        {
            cout << "Success!" << endl;
        }
    }

    return 0;
}
```

数据结构

1. LRU

```
// Least Recently Used
struct cache
{
    struct double_node
    {
        ll key, val;
        double_node *last, *next;
        double_node(ll k, ll v) : key(k), val(v), last(nullptr), next(nullptr) {}
    };

    struct double_list
    {
        double_node *head, *tail;
        double_list() : head(nullptr), tail(nullptr) {}

        void add_to_tail(double_node *node)
        {
            if (node == nullptr)
                return;
            if (head == nullptr)
            {
                head = node;
                tail = node;
            }
            else
            {
                tail->next = node;
                node->last = tail;
                node->next = nullptr;
                tail = node;
            }
        }

        void move_to_tail(double_node *node)
        {
            if (tail == node)
                return;
            if (head == node)
            {
                head = head->next;
                if (head)
                    head->last = nullptr;
            }
            else
            {
                node->last->next = node->next;
                if (node->next)
                    node->next->last = node->last;
            }
            add_to_tail(node);
        }

        double_node *remove_head()
        {
            if (head == nullptr)
                return nullptr;
            double_node *ans = head;
            if (head == tail)
            {
                head = nullptr;
                tail = nullptr;
            }
            else
            {
                head = ans->next;
```

```
                if (head)
                    head->last = nullptr;
                ans->next = nullptr;
            }
            return ans;
        }
    };

    ll capacity;
    map<ll, double_node *> key_to_node;
    double_list list;

    cache(ll cap) : capacity(cap) {}

    ~cache()
    {
        double_node *current = list.head;
        while (current != nullptr)
        {
            double_node *next = current->next;
            delete current;
            current = next;
        }
    }

    ll get_value(ll k)
    {
        if (key_to_node.count(k))
        {
            double_node *pos = key_to_node[k];
            list.move_to_tail(pos);
            return pos->val;
        }
        return -1;
    }

    void insert(ll k, ll v)
    {
        if (key_to_node.count(k))
        {
            double_node *pos = key_to_node[k];
            pos->val = v;
            list.move_to_tail(pos);
        }
        else
        {
            if (key_to_node.size() == capacity)
            {
                double_node *lru_node = list.remove_head();
                key_to_node.erase(lru_node->key);
                delete lru_node;
            }
            double_node *new_node = new double_node(k, v);
            key_to_node[k] = new_node;
            list.add_to_tail(new_node);
        }
    }
};

void print_cache_state(cache &lru_cache)
{
    cout << "   Cache State (LRU -> MRU): [";
    cache::double_node *current = lru_cache.list.head;
    bool first = true;
    while (current != nullptr)
    {
        if (!first)
```

```
            {
                cout << ", ";
            }
            cout << "(" << current->key << ":" << current->val << ")";
            first = false;
            current = current->next;
        }
        cout << "]" << endl;
    }

    signed main()
    {
        cache lru_cache(3);
        cout << "created a cache with capacity 3." << endl;

        cout << "\n--- test 1: basic insertion ---" << endl;
        lru_cache.insert(1, 10);
        cout << "Inserted (1, 10)." << endl;
        print_cache_state(lru_cache);

        lru_cache.insert(2, 20);
        cout << "Inserted (2, 20)." << endl;
        print_cache_state(lru_cache);

        lru_cache.insert(3, 30);
        cout << "Inserted (3, 30). cache is now full." << endl;
        print_cache_state(lru_cache);

        cout << "\n--- test 2: 'get' operation and reordering ---" << endl;
        cout << "accessing key 1... value: " << lru_cache.get_value(1) << endl;
        cout << "key 1 should now be the most recently used." << endl;
        print_cache_state(lru_cache);

        cout << "\n--- test 3: eviction ---" << endl;
        lru_cache.insert(4, 40);
        cout << "Inserted (4, 40). key 2 (the LRU) should be evicted." << endl;
        print_cache_state(lru_cache);

        cout << "\n--- test 4: updating an existing key ---" << endl;
        lru_cache.insert(3, 333);
        cout << "updated (3, 333). value should change and it should become MRU." << endl;
        print_cache_state(lru_cache);

        cout << "\n--- test 5: edge cases ---" << endl;
        cout << "getting a non-existent key 99... value: " << lru_cache.get_value(99) << endl;
        cout << "cache state should be unchanged." << endl;
        print_cache_state(lru_cache);

        return 0;
    }

    2. 最大频率栈
    struct freq_stack
    {
        ll top_times;
        map<ll, list<ll>> cnt;
        map<ll, ll> value_times;
        freq_stack() : top_times(0) {}

        void push(ll v)
        {
            ll cur_times = ++value_times[v];
            cnt[cur_times].push_back(v);
            top_times = max(value_times[v], top_times);
        }

        ll pop()
```

```
        {
            ll remove = cnt[top_times].back();
            cnt[top_times].pop_back();
            if (cnt[top_times].empty())
            {
                cnt.erase(top_times--);
            }

            if (value_times[remove] == 1)
            {
                value_times.erase(remove);
            }
            else
            {
                value_times[remove]--;
            }
            return remove;
        }
};

signed main()
{
    freq_stack fs;
    cout << "push(5), push(7), push(5), push(7), push(4), push(5)" << endl;
    fs.push(5);
    fs.push(7);
    fs.push(5);
    fs.push(7);
    fs.push(4);
    fs.push(5);

    // 5(3), 7(2), 4(1)。  top_times = 3.
    cout << "pop: " << fs.pop() << endl; // 5
    cout << "pop: " << fs.pop() << endl; // 7
    cout << "pop: " << fs.pop() << endl; // 5
    cout << "pop: " << fs.pop() << endl; // 4
    cout << "pop: " << fs.pop() << endl; // 7
    cout << "pop: " << fs.pop() << endl; // 5

    return 0;
}
```

3. 频率字符串

```
struct freq_string
{
    struct bucket
    {
        set<string> buk;
        ll freq;
        bucket *next, *last;

        bucket(ll f) : freq(f), next(nullptr), last(nullptr) {}
    };

    void after_bucket(bucket *cur, bucket *pos)
    {
        pos->next = cur->next;
        cur->next->last = pos;
        cur->next = pos;
        pos->last = cur;
    }

    void remove_bucket(bucket *cur)
    {
        cur->last->next = cur->next;
        cur->next->last = cur->last;
    }
```

```
bucket *head, *tail;
map<string, bucket *> mp;

freq_string()
{
    head = new bucket(0);
    tail = new bucket(LLONG_MAX);
    head->next = tail;
    tail->last = head;
}

~freq_string()
{
    bucket *cur = head;
    while (cur != nullptr)
    {
        bucket *next_buk = cur->next;
        delete cur;
        cur = next_buk;
    }
}

void inc(string k)
{
    bucket *cur_buk;
    ll cur_freq;

    if (mp.find(k) == mp.end())
    {
        cur_buk = head;
        cur_freq = 0;
    }
    else
    {
        cur_buk = mp[k];
        cur_freq = cur_buk->freq;
        cur_buk->buk.erase(k);
    }

    ll next_freq = cur_freq + 1;
    if (cur_buk->next->freq != next_freq)
    {
        bucket *new_buk = new bucket(next_freq);
        after_bucket(cur_buk, new_buk);
    }

    bucket *next_buk = cur_buk->next;
    next_buk->buk.insert(k);
    mp[k] = next_buk;

    if (cur_buk != head && cur_buk->buk.empty())
    {
        remove_bucket(cur_buk);
        delete cur_buk;
    }
}

void dec(string k)
{
    if (mp.find(k) == mp.end())
    {
        return;
    }

    bucket *cur_buk = mp[k];
    ll cur_freq = cur_buk->freq;
```

```
        cur_buk->buk.erase(k);

        if (cur_freq == 1)
        {
            mp.erase(k);
        }
        else
        {
            ll prev_freq = cur_freq - 1;
            if (cur_buk->last->freq != prev_freq)
            {
                bucket *new_buk = new bucket(prev_freq);
                after_bucket(cur_buk->last, new_buk);
            }
            bucket *prev_buk = cur_buk->last;
            prev_buk->buk.insert(k);
            mp[k] = prev_buk;
        }

        if (cur_buk->buk.empty())
        {
            remove_bucket(cur_buk);
            delete cur_buk;
        }
    }
};

void print_freq_state(const string &title, freq_string &fs)
{
    cout << "--- " << title << " ---" << endl;
    cout << "   list state (freq: {strings}): ";
    freq_string::bucket *cur = fs.head->next;
    while (cur != fs.tail)
    {
        cout << cur->freq << ":{";
        bool first = true;
        for (const auto &s : cur->buk)
        {
            if (!first)
                cout << ",";
            cout << s;
            first = false;
        }
        cout << "} ";
        cur = cur->next;
    }
    if (fs.head->next == fs.tail)
    {
        cout << "(empty)";
    }
    cout << endl;
    cout << "   map state: {";
    bool first = true;
    for (auto const &[key, b] : fs.mp)
    {
        if (!first)
            cout << ", ";
        cout << key << "->(freq:" << b->freq << ")";
        first = false;
    }
    cout << "}\n"
         << endl;
}

signed main()
{
```

```
    freq_string fs;
    print_freq_state("initial state", fs);

    fs.inc("a");
    print_freq_state("inc(\"a\")", fs);
    fs.inc("b");
    print_freq_state("inc(\"b\")", fs);

    fs.inc("a");
    print_freq_state("inc(\"a\") again, creates freq=2 bucket", fs);

    fs.inc("b");
    print_freq_state("inc(\"b\") again, freq=1 bucket is removed", fs);

    fs.inc("c");
    print_freq_state("inc(\"c\")", fs);

    fs.dec("a");
    print_freq_state("dec(\"a\"), creates freq=1 bucket", fs);

    fs.dec("b");
    print_freq_state("dec(\"b\"), freq=2 bucket is removed", fs);

    fs.dec("a");
    print_freq_state("dec(\"a\") again, \"a\" is removed", fs);

    fs.dec("b");
    fs.dec("c");
    print_freq_state("dec(\"b\") and dec(\"c\"), all buckets removed", fs);

    fs.dec("z");
    print_freq_state("dec(\"z\"), state is unchanged", fs);

    return 0;
}
```

4. 数据流中位数

```
// quickly find the median of a data stream.
struct median_data
{
    // the smaller half is in the max-heap, and the larger half is in the min-heap.
    priority_queue<ll> max_heap;
    priority_queue<ll, vector<ll>, greater<ll>> min_heap;

    void add(ll v)
    {
        if (max_heap.empty() || max_heap.top() >= v)
        {
            max_heap.push(v);
        }
        else
        {
            min_heap.push(v);
        }
        balance();
    }

    ll get_median()
    {
        if (max_heap.size() == min_heap.size())
        {
            return (max_heap.top() + min_heap.top()) / 2;
        }
        else if (max_heap.size() > min_heap.size())
        {
            return max_heap.top();
        }
```

```
        else
        {
            return min_heap.top();
        }
    }

    void balance()
    {
        if (max_heap.size() > min_heap.size())
        {
            if (max_heap.size() - min_heap.size() == 2)
            {
                min_heap.push(max_heap.top());
                max_heap.pop();
            }
        }
        else
        {
            if (min_heap.size() - max_heap.size() == 2)
            {
                max_heap.push(min_heap.top());
                min_heap.pop();
            }
        }
    }
};

signed main()
{
    median_data md;

    cout << "\n--- test 1: processing a stream of numbers ---" << endl;
    cout << fixed << setprecision(1);

    ll data_stream[] = {5, 1, 10, 3, 7};
    vector<ll> current_data;

    for (ll v : data_stream)
    {
        current_data.push_back(v);
        sort(current_data.begin(), current_data.end());

        cout << "added " << v << ", current stream: [";
        for (size_t i = 0; i < current_data.size(); ++i)
        {
            cout << current_data[i] << (i == current_data.size() - 1 ? "" : ", ");
        }
        cout << "]" << endl;

        md.add(v);
        cout << "    -> median: " << md.get_median() << endl;
    }

    return 0;
}
```

5. 集合随机输出

```
struct randomize_set
{
    map<ll, ll> val_to_pos; // (v, pos)
    vector<ll> arr;
    mt19937 rng;

    randomize_set()
    {
        rng.seed(chrono::high_resolution_clock::now().time_since_epoch().count());
    }
```

```
    bool insert(ll v)
    {
        if (val_to_pos.count(v))
        {
            return false;
        }
        val_to_pos[v] = arr.size();
        arr.push_back(v);
        return true;
    }

    ll get_random()
    {
        if (arr.empty())
        {
            return -1;
        }

        uniform_int_distribution<ll> dist(0, arr.size() - 1);
        return arr[dist(rng)];
    }

    bool remove(ll v)
    {
        if (!val_to_pos.count(v))
        {
            return false;
        }

        ll pos = val_to_pos[v];
        ll last_val = arr.back();

        arr[pos] = last_val;
        val_to_pos[last_val] = pos;

        val_to_pos.erase(v);
        arr.pop_back();

        return true;
    }
};

void print_set_state(randomize_set &rs)
{
    cout << "   vector state: [";
    for (size_t i = 0; i < rs.arr.size(); ++i)
    {
        cout << rs.arr[i] << (i == rs.arr.size() - 1 ? "" : ", ");
    }
    cout << "]" << endl;
    cout << "   map state: {";
    bool first = true;
    for (auto const &[val, pos] : rs.val_to_pos)
    {
        if (!first)
            cout << ", ";
        cout << val << "->" << pos;
        first = false;
    }
    cout << "}" << endl;
}

signed main()
{
    randomize_set rs;
```

```
    cout << "--- elements ---" << endl;
    rs.insert(10);
    rs.insert(20);
    rs.insert(30);
    rs.insert(40);
    print_set_state(rs);

    cout << "\n--- removing element 20 (not the last one) ---" << endl;
    rs.remove(20);
    cout << "state after removing 20:" << endl;
    print_set_state(rs);

    cout << "\n--- removing element 10 ---" << endl;
    rs.remove(10);
    cout << "state after removing 10:" << endl;
    print_set_state(rs);

    cout << "\n--- getting 5 random elements ---" << endl;
    for (ll i = 0; i < 5; ++i)
    {
        cout << "random element: " << rs.get_random() << endl;
    }

    return 0;
}
```

6. 集合随机输出（可重复）

```
struct randomize_set
{
    map<ll, set<ll>> val_to_pos; // (v, pos_set)
    vector<ll> arr;
    mt19937 rng;

    randomize_set()
    {
        rng.seed(chrono::high_resolution_clock::now().time_since_epoch().count());
    }

    ll get_random()
    {
        if (arr.empty())
        {
            return -1;
        }
        uniform_int_distribution<ll> dist(0, arr.size() - 1);
        return arr[dist(rng)];
    }

    bool insert(ll v)
    {
        val_to_pos[v].insert(arr.size());
        arr.push_back(v);
        return true;
    }

    bool remove(ll v)
    {
        if (val_to_pos.find(v) == val_to_pos.end() || val_to_pos[v].empty())
        {
            return false;
        }

        ll pos = *(val_to_pos[v].begin());
        ll last_val = arr.back();
        ll last_pos = arr.size() - 1;

        arr[pos] = last_val;
```

```
        arr.pop_back();
        val_to_pos[v].erase(pos);


        if (pos != last_pos)
        {
            val_to_pos[last_val].erase(last_pos);
            val_to_pos[last_val].insert(pos);
        }

        if (val_to_pos[v].empty())
        {
            val_to_pos.erase(v);
        }

        return true;
    }
};

void print_set_state(const string &title, randomize_set &rs)
{
    cout << title << endl;
    cout << "   vector state: [";
    for (size_t i = 0; i < rs.arr.size(); ++i)
    {
        cout << rs.arr[i] << (i == rs.arr.size() - 1 ? "" : ", ");
    }
    cout << "]" << endl;
    cout << "   map state:      {";
    bool first_val = true;
    for (auto const &[val, pos_set] : rs.val_to_pos)
    {
        if (!first_val)
            cout << ", ";
        cout << val << ":{";
        bool first_pos = true;
        for (ll pos : pos_set)
        {
            if (!first_pos)
                cout << ",";
            cout << pos;
            first_pos = false;
        }
        cout << "}";
        first_val = false;
    }
    cout << "}\n"
         << endl;
}
```

输入输出

1. 标准输出流

```
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    ll n;
    cin >> n;
    vector<ll> a(n);
    for (ll i = 0; i < n; i++) {
        cin >> a[i];
    }

    sort(a.begin(), a.end());

    for (ll i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;

    return 0;
}
```

2. 值传递与引用传递

```
// pass by value: char, int, long, float, double,  bool, string
void f(ll a)
{
    a = 100;
}

// pass by reference: class, struct, array
struct number
{
    ll a;
    number(ll val) : a(val) {}
};

void g_1(number *ptr)
{
    // this will not change the original pointer and causing the memory leak
    ptr = nullptr;
}

void g_2(number *ptr)
{
    // this will change the value of the original pointer because of the same address in heap
    ptr->a = 100;
}

void g_3(ll arr[])
{
    arr = nullptr;
}

void g_4(ll arr[])
{
    arr[0] = 100;
}

signed main()
{
    ll a = 10;
    f(a);
    // 10, pass by value, a is not changed
    cout << a << endl;
```

```
    number *ptr = new number(10);
    g_1(ptr);
    // 10, pass by reference
    cout << ptr->a << endl;
    g_2(ptr);
    cout << ptr->a << endl;

    ll arr[] = {10, 20, 30};
    g_3(arr);
    cout << arr[0] << endl;
    g_4(arr);
    cout << arr[0] << endl;

    delete ptr;
    ptr = nullptr;
    cout << "Program Finished." << endl;
    return 0;
}
```

链表

1. 链表基本操作

```
// single linked list node
struct single_node
{
    ll data;
    single_node *next;
    single_node(ll val) : data(val), next(nullptr) {}
};

// double linked list node
struct double_node
{
    ll data;
    double_node *next;
    double_node *prev;
    double_node(ll val) : data(val), next(nullptr), prev(nullptr) {}
};

// reverse a single linked list
single_node *reverse_list(single_node *head)
{
    single_node *prev = nullptr;
    single_node *next = nullptr;
    while (head != nullptr)
    {
        // prev will be linked by head, next := head
        next = head->next; // next go along the list
        head->next = prev; // head -> prev
        prev = head;       // prev go along the list
        head = next;       // head go along the list
    }
    return prev;
}

// reverse a double linked list
double_node *reverse_doubly_list(double_node *head)
{
    double_node *prev = nullptr;
    double_node *next = nullptr;
    while (head != nullptr)
    {
        // prev will be linked by head, next := head
        next = head->next; // next go along the list
        head->next = prev; // head -> prev
        head->prev = next; // head -> next
        prev = head;       // prev go along the list
        head = next;       // head go along the list
    }
    return prev;
}

// merge two sorted single linked lists
single_node *merge_sorted_lists(single_node *l1, single_node *l2)
{
    if (l1 == nullptr || l2 == nullptr)
        return l1 ? l1 : l2;

    single_node *head = l1->data <= l2->data ? l1 : l2;
    single_node *cur_1 = head->next;
    single_node *cur_2 = head == l1 ? l2 : l1;
    single_node *temp = head;

    while (cur_1 && cur_2)
    {
        if (cur_1->data <= cur_2->data)
        {
```

```
            temp->next = cur_1;
            cur_1 = cur_1->next;
        }
        else
        {
            temp->next = cur_2;
            cur_2 = cur_2->next;
        }
        temp = temp->next;
    }
    temp->next = (cur_1 ? cur_1 : cur_2);
    return head;
}

signed main()
{
    // 1 -> 3 -> 5
    single_node *l1 = new single_node(1);

    // 2 -> 4 -> 6
    single_node *l2 = new single_node(2);

    // merge the two sorted linked lists
    single_node *merged = merge_sorted_lists(l1, l2);

    single_node *current = merged;
    while (current != nullptr)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;

    // free the allocated memory
    current = merged;
    while (current != nullptr)
    {
        single_node *next = current->next;
        delete current;
        current = next;
    }

    return 0;
}
```

2. 有环链表首个入环节点

```
struct node
{
    ll data;
    node *next;
    node(ll val) : data(val), next(nullptr) {}
};

// return the entrance node of a cyclic linked list.
node *cyclic_list(node *head)
{
    if (head == nullptr || head->next == nullptr || head->next->next == nullptr)
    {
        return nullptr;
    }

    node *slow = head->next, *fast = head->next->next;
    while (slow != fast)
    {
        if (fast->next == nullptr || fast->next->next == nullptr)
        {
            return nullptr;
```

```
            }
            slow = slow->next;
            fast = fast->next->next;
        }

        fast = head;
        while (slow != fast)
        {
            slow = slow->next;
            fast = fast->next;
        }

        return fast;
}

void delete_linked_list(node *head)
{
        node *current = head;
        while (current != nullptr)
        {
            node *next_node = current->next;
            delete current;
            current = next_node;
        }
}

void delete_cyclic_list(node *head)
{
        if (head == nullptr)
            return;

        node *entrance = cyclic_list(head);
        if (entrance == nullptr)
        {
            delete_linked_list(head);
            return;
        }

        node *tail_of_cycle = entrance;
        while (tail_of_cycle->next != entrance)
        {
            tail_of_cycle = tail_of_cycle->next;
        }
        tail_of_cycle->next = nullptr;

        delete_linked_list(head);
}

signed main()
{
        node *head = new node(1);
        head->next = new node(2);
        head->next->next = new node(3);
        head->next->next->next = new node(4);
        head->next->next->next->next = new node(5);
        head->next->next->next->next->next = head->next->next;

        node *entrance = cyclic_list(head);
        if (entrance != nullptr)
        {
            cout << "cycle entrance found at node with value: " << entrance->data << endl;
        }
        else
        {
            cout << "no cycle found." << endl;
        }
```

```
    delete_cyclic_list(head);
    cout << "cyclic list successfully deleted." << endl;

    return 0;
}
```

3. 链表 k 个元素一组翻转

```
struct node
{
    ll data;
    node *next;
    node(ll val) : data(val), next(nullptr) {}
};

node *goto_k(node *s, ll k)
{
    if (s == nullptr)
        return nullptr;
    node *temp = s;
    for (ll i = 0; i < k - 1 && temp != nullptr; i++)
    {
        temp = temp->next;
    }
    return temp;
}

void reverse(node *start, node *end)
{
    end = end->next;
    node *prev = nullptr, *current = start, *next = nullptr;
    while (current != end)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    start->next = end;
}

// reverse a linked list in groups of k, and leave the last group as-is if it has fewer than k elements.
node *k_reverse(node *head, ll k)
{
    node *start = head, *end = goto_k(head, k);
    if (end == nullptr)
        return head;

    head = end;
    reverse(start, end);
    node *last_team_end = start;
    while (last_team_end->next != nullptr)
    {
        start = last_team_end->next;
        end = goto_k(start, k);
        if (end == nullptr)
            return head;
        reverse(start, end);
        last_team_end->next = end;
        last_team_end = start;
    }

    return head;
}

void delete_linked_list(node *head)
{
    node *current = head;
```

```
    while (current != nullptr)
    {
        node *next_node = current->next;
        delete current;
        current = next_node;
    }
}

signed main()
{
    // 1 -> 2 -> 3 -> 4 -> 5

    ll k = 2;
    head = k_reverse(head, k);

    node *temp = head;
    while (temp != nullptr)
    {
        printf("%lld ", temp->data);
        temp = temp->next;
    }

    delete_linked_list(head);
    printf("\n");

    return 0;
}
```

4. 链表首个相交节点

```
struct node
{
    ll data;
    node *next;
    node(ll val) : data(val), next(nullptr) {}
};

// return the first intersecting node of two linked lists.
//        \      /
//         \    /
//          \ /
//           X
//           |
//           |
node *list_intersecting(node *ho, node *ht)
{
    if (ho == nullptr || ht == nullptr)
        return nullptr;

    node *po = ho;
    node *pt = ht;

    // length of both lists
    ll lo = 1, lt = 1;
    while (po->next != nullptr)
    {
        lo++;
        po = po->next;
    }
    while (pt->next != nullptr)
    {
        lt++;
        pt = pt->next;
    }

    if (po != pt)
    {
        return nullptr;
```

```
    }

    // reset pointers
    po = ho;
    pt = ht;

    // align the start of both lists
    if (lo > lt)
    {
        for (ll i = 0; i < lo - lt; i++)
            po = po->next;
    }
    else
    {
        for (ll i = 0; i < lt - lo; i++)
            pt = pt->next;
    }

    // go together
    while (po != pt)
    {
        po = po->next;
        pt = pt->next;
    }

    return po;
}

void delete_linked_list(node *head)
{
    node *current = head;
    while (current != nullptr)
    {
        node *next_node = current->next;
        delete current;
        current = next_node;
    }
}

signed main()
{
    node *ho = new node(1);
    ho->next = new node(2);
    ho->next->next = new node(3);
    ho->next->next->next = new node(4);

    node *ht = new node(5);
    ht->next = new node(6);
    ht->next->next = ho->next->next; // node 3

    node *both = list_intersecting(ho, ht);
    if (both)
        cout << "Intersecting at Node with Value: " << both->data << endl;
    else
        cout << "No Intersection Found." << endl;

    if (both != nullptr)
    {
        ht->next->next = nullptr;
    }

    delete_linked_list(ho);
    delete_linked_list(ht);

    return 0;
}
```

5. 链表实现整数相加

```
struct single_node
{
    ll data;
    single_node *next;
    single_node(ll val) : data(val), next(nullptr) {}
};

single_node *number_plus(single_node *l1, single_node *l2)
{
    single_node *ans = nullptr, *cur = nullptr;
    ll carry = 0;

    for (ll sum, val;
         // both the lists are null, then loop end
         l1 != nullptr || l2 != nullptr;
         l1 = (l1 ? l1->next : nullptr), l2 = (l2 ? l2->next : nullptr))
    {
        sum = (l1 ? l1->data : 0) + (l2 ? l2->data : 0) + carry;

        val = sum % 10;
        carry = sum / 10;

        // head node
        if (ans == nullptr)
        {
            ans = new single_node(val);
            cur = ans;
        }
        else
        {
            cur->next = new single_node(val);
            cur = cur->next;
        }
    }

    if (carry != 0)
    {
        cur->next = new single_node(1);
    }
    return ans;
}

signed main()
{
    // 1 -> 2 -> 3
    single_node *l1 = new single_node(1);
    l1->next = new single_node(2);
    l1->next->next = new single_node(3);

    // 4 -> 5 -> 6
    single_node *l2 = new single_node(4);
    l2->next = new single_node(5);
    l2->next->next = new single_node(6);

    // add the two linked lists
    single_node *result = number_plus(l1, l2);

    single_node *current = result;
    while (current != nullptr)
    {
        cout << current->data;
        current = current->next;
    }
    cout << endl;

    current = result;
```

```
    while (current != nullptr)
    {
        single_node *next = current->next;
        delete current;
        current = next;
    }

    return 0;
}
```

6. 回文链表

```
struct node
{
    ll data;
    node *next;
    node(ll val) : data(val), next(nullptr) {}
};

bool palindrome(node *head)
{
    if (head == nullptr || head->next == nullptr)
        return true;

    node *slow = head, *fast = head;

    // 1 → 2 → 2 → 1

    //       ↑
    // 1 → 2 ← 2 ← 1
    while (fast->next != nullptr && fast->next->next != nullptr)
    {
        slow = slow->next;
        fast = fast->next->next;
    }
    // reverse
    // head → ... → slow ← ... ← prev
    node *prev = slow, *cur = prev->next, *next = nullptr;
    prev->next = nullptr;
    while (cur != nullptr)
    {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }

    bool ans = true;
    node *left = head, *right = prev;
    while (left != nullptr && right != nullptr)
    {
        if (left->data != right->data)
        {
            ans = false;
            break;
        }
        left = left->next;
        right = right->next;
    }

    // restore the original list
    cur = prev->next;
    prev->next = nullptr;
    next = nullptr;
    while (cur != nullptr)
    {
        next = cur->next;
        cur->next = prev;
```

```
            prev = cur;
            cur = next;
        }

        return ans;
}

void delete_list(node *head)
{
        while (head != nullptr)
        {
            node *temp = head;
            head = head->next;
            delete temp;
        }
}

signed main()
{
        node *head = new node(1);
        head->next = new node(2);
        head->next->next = new node(2);
        head->next->next->next = new node(3);

        palindrome(head) ? cout << "YES" << endl
                         : cout << "NO" << endl;

        node *cur = head;
        while (cur != nullptr)
        {
            cout << cur->data << " ";
            cur = cur->next;
        }

        delete_list(head);
        return 0;
}
```

7. 链表按照指定数字分块

```
struct single_node
{
        ll data;
        single_node *next;
        single_node(ll val) : data(val), next(nullptr) {}
};

// divide the list into two parts, the number of one less than x and the other part is greater than or equal to x,
keeping the relative order of the nodes in each part.
single_node *partition(single_node *head, ll x)
{
        single_node *less_head = nullptr, *less_tail = nullptr;
        single_node *greater_head = nullptr, *greater_tail = nullptr;
        single_node *next = nullptr;
        while (head != nullptr)
        {
            next = head->next;      // store the next node
            head->next = nullptr; // break the link to avoid cycles
            if (head->data < x)
            {
                if (less_head == nullptr)
                {
                    less_head = head;
                    less_tail = head;
                }
                else
                {
                    less_tail->next = head;
```

```
                less_tail = less_tail->next;
            }
        }
        else
        {
            if (greater_head == nullptr)
            {
                greater_head = head;
                greater_tail = head;
            }
            else
            {
                greater_tail->next = head;
                greater_tail = greater_tail->next;
            }
        }
        head = next; // move to the next node
    }
    // there is no element less than x
    if (less_tail == nullptr)
    {
        return greater_head;
    }
    less_tail->next = greater_head; // connect the two parts
    return less_head;
}
signed main()
{
    // 6 -> 5 -> 3 -> 4 -> 2 -> 1 -> 1 -> 7
    single_node *ans = partition(head, 3);
    while (ans != nullptr)
    {
        cout << ans->data << " ";
        ans = ans->next;
    }
    return 0;
}
```

8. 链表随机指针

```
struct node
{
    ll data;
    node *next;
    node *random;
    node(ll val) : data(val), next(nullptr), random(nullptr) {}
};

// copy a linked list with random pointers
node *linked_list(node *head)
{
    if (head == nullptr)
        return nullptr;

    node *cur = head, *next = nullptr;
    // 1 → 2 → 3 → null
    // 1 → 1' → 2 → 2' → 3 → 3' → null
    while (cur != nullptr)
    {
        next = cur->next;
        cur->next = new node(cur->data);
        cur->next->next = next;
        cur = next;
    }

    // random pointers
    cur = head;
    node *copy = nullptr;
```

```
    while (cur != nullptr)
    {
        copy = cur->next;
        copy->random = (cur->random != nullptr ? cur->random->next : nullptr);
        cur = copy->next;
    }

    // 1 → 2 → 3 → null
    // 1' → 2' → 3' → null
    node *ans = head->next;
    cur = head;
    while (cur != nullptr)
    {
        next = cur->next->next;
        copy = cur->next;
        cur->next = next;
        copy->next = (next != nullptr ? next->next : nullptr);
        cur = next;
    }

    return ans;
}

void delete_linked_list(node *head)
{
    node *cur = head;
    while (cur != nullptr)
    {
        node *next_node = cur->next;
        delete cur;
        cur = next_node;
    }
}

signed main()
{
    node *head = new node(1);
    head->next = new node(2);
    head->next->next = new node(3);
    head->next->next->next = new node(4);
    head->random = head->next->next;          // random pointer to node 3
    head->next->random = head;                // random pointer to node 1
    head->next->next->random = head->next; // random pointer to node 2
    head->next->next->next->random = nullptr;

    node *copy = linked_list(head);
    cout << "original list: " << endl;
    for (node *cur = head; cur != nullptr; cur = cur->next)
    {
        cout << "Value: " << cur->data << ", Random: " << (cur->random != nullptr ? cur->random->data : -1) <<
endl;
    }

    cout << "copied list: " << endl;
    for (node *cur = copy; cur != nullptr; cur = cur->next)
    {
        cout << "Value: " << cur->data << ", Random: " << (cur->random != nullptr ? cur->random->data : -1) <<
endl;
    }

    // cleanup
    delete_linked_list(head);
    delete_linked_list(copy);
    return 0;
}
```

前缀和与差分

1. 二维前缀和

```
ll matrix[4][4] = {
    {1, 1, 0, 1},
    {0, 1, 1, 1},
    {0, 1, 0, 1},
    {1, 1, 0, 1}};
ll get_matrix(ll i, ll j)
{
    return i < 0 || j < 0 ? 0 : matrix[i][j];
}
ll sum_matrix(ll a, ll b, ll c, ll d)
{
    return a > c ? 0 : get_matrix(c, d) - get_matrix(a - 1, d) - get_matrix(c, b - 1) + get_matrix(a - 1, b - 1);
}

signed main()
{
    // 2D Prefix Sum
    // sum[i][j] = sum(arr[0][0] to arr[i][j])
    ll arr[3][3] = {
        {3, 1, -2},
        {5, -3, 4},
        {3, 6, -2}};
    ll sum[3][3] = {0};
    // 容斥 sum[i][j] = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + arr[i][j]
    for (ll i = 0; i < 3; i++)
    {
        for (ll j = 0; j < 3; j++)
        {
            sum[i][j] = arr[i][j];
            if (i > 0)
                sum[i][j] += sum[i - 1][j];
            if (j > 0)
                sum[i][j] += sum[i][j - 1];
            if (i > 0 && j > 0)
                sum[i][j] -= sum[i - 1][j - 1];
        }
    }

    // for (ll i = 0; i < 3; i++)
    // {
    //     for (ll j = 0; j < 3; j++)
    //     {
    //         cout << sum[i][j] << " ";
    //     }
    //     cout << endl;
    // }

    // sum of submatrix (x1, y1) to (x2, y2)
    // sum[x2][y2] - sum[x1-1][y2] - sum[x2][y1-1] + sum[x1-1][y1-1]
    ll x1 = 1, y1 = 1, x2 = 2, y2 = 2;
    ll result = sum[x2][y2] - sum[x1 - 1][y2] - sum[x2][y1 - 1] + sum[x1 - 1][y1 - 1];

    // verify if the borders of a sub-square in 0/1 matrix are all 1s.
    ll max_border;
    ll n = sizeof(matrix) / sizeof(matrix[0]);          // rows
    ll m = sizeof(matrix[0]) / sizeof(matrix[0][0]); // cols
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            matrix[i][j] += get_matrix(i - 1, j) + get_matrix(i, j - 1) - get_matrix(i - 1, j - 1);
        }
    }
    if (matrix[n - 1][m - 1] == 0)
    {
```

```
            max_border = 0;
        }
        else
        {
            max_border = 1;
            // check matrix[a][b] to matrix[c][d] is a sub-square
            for (ll a = 0; a < n; a++)
            {
                for (ll b = 0; b < m; b++)
                {
                    for (ll c = a + max_border, d = b + max_border, k = max_border + 1; c < n && d < m; c++, d++,
k++)
                    {

                        // sum of submatrix (a, b) to (c, d) - sum of submatrix (a+1, b+1) to (c-1, d-1)
                        if (sum_matrix(a, b, c, d) - sum_matrix(a + 1, b + 1, c - 1, d - 1) == (k - 1) << 2)
                        {
                            max_border = k;
                        }
                    }
                }
            }
        }
        cout << max_border << endl;

        return 0;
}
```

2. 二维差分

```
const ll maxn = 1001;
ll diff[maxn][maxn];

ll get_diff(ll i, ll j)
{
    return i < 0 || j < 0 ? 0 : diff[i][j];
}

ll sum_diff(ll a, ll b, ll c, ll d)
{
    return a > c ? 0 : get_diff(c, d) - get_diff(a - 1, d) - get_diff(c, b - 1) + get_diff(a - 1, b - 1);
}

// recommended to pass two-dimensional arrays using vector
void add(ll a, ll b, ll c, ll d, ll x)
{
    diff[a][b] += x;
    diff[c + 1][b] -= x;
    diff[a][d + 1] -= x;
    diff[c + 1][d + 1] += x;
}

void build_diff(ll n, ll m)
{
    for (ll i = 1; i <= n; i++)
    {
        for (ll j = 1; j <= m; j++)
        {
            ll x;
            cin >> x;
            // add the value x to the rectangle defined by (i, j) to (i, j)
            // equal to adding x to the cell (i, j)
            add(i, j, i, j, x);
        }
    }
}

void build_prefix(ll n, ll m)
```

```
{
    for (ll i = 1; i <= n; i++)
    {
        for (ll j = 1; j <= m; j++)
        {
            diff[i][j] += diff[i - 1][j] + diff[i][j - 1] - diff[i - 1][j - 1];
        }
    }
}

signed main()
{
    ll n, m, q;
    cin >> n >> m >> q;

    build_diff(n, m);
    for (ll i = 0; i < q; i++)
    {
        ll a, b, c, d, k;
        cin >> a >> b >> c >> d >> k;
        // modify the rectangle defined by (a, b) to (c, d) by k
        add(a, b, c, d, k);
    }
    // diff[n][m] padding with zeros to reduce boundary checks
    build_prefix(n, m);

    for (ll i = 1; i <= n; i++)
    {
        for (ll j = 1; j <= m; j++)
        {
            cout << diff[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

3. 离散化

```
const ll maxn = 1001;
ll field[maxn][maxn];
ll diff[maxn][maxn];

ll _sort(ll arr[], ll n)
{
    sort(arr, arr + 2 * n);
    ll size = 1;
    for (ll i = 1; i < 2 * n; i++)
    {
        if (arr[i] != arr[i - 1])
        {
            arr[size++] = arr[i];
        }
    }
    return size;
}

ll _rank(ll arr[], ll n, ll x)
{
    // return the index of the first element greater than or equal to x
    return lower_bound(arr, arr + n, x) - arr + 1;
}

void add(ll a, ll b, ll c, ll d, ll x)
{
    diff[a][b] += x;
    diff[c + 1][b] -= x;
    diff[a][d + 1] -= x;
```

```
    diff[c + 1][d + 1] += x;
}

signed main()
{
    // field[[x, y ,r], ...]
    ll n, x, y, r;
    cin >> n;
    // n rectangles, 2n coordinates
    ll fx[n << 1], fy[n << 1];
    for (ll i = 0, k = 0, p = 0; i < n; i++)
    {
        x = field[i][0];
        y = field[i][1];
        r = field[i][2];
        // x := 2x, y := 2y, r := 2r
        fx[k++] = 2 * x - r;
        fx[k++] = 2 * x + r;
        fy[p++] = 2 * y - r;
        fy[p++] = 2 * y + r;
    }

    // discretize the coordinates
    ll sizex = _sort(fx, n);
    ll sizey = _sort(fy, n);

    // 2D Differential
    ll diff_row = sizex + 2;
    ll diff_col = sizey + 2;

    for (ll i = 0, a, b, c, d; i < n; i++)
    {
        x = field[i][0];
        y = field[i][1];
        r = field[i][2];
        a = _rank(fx, sizex, 2 * x - r);
        b = _rank(fy, sizey, 2 * y - r);
        c = _rank(fx, sizex, 2 * x + r);
        d = _rank(fy, sizey, 2 * y + r);
        add(a, b, c, d, 1);
    }

    ll ans = 0;
    for (ll i = 1; i <= diff_row; i++)
    {
        for (ll j = 1; j <= diff_col; j++)
        {
            diff[i][j] += diff[i - 1][j] + diff[i][j - 1] - diff[i - 1][j - 1];
            ans = max(ans, diff[i][j]);
        }
    }
    cout << ans << endl;
    return 0;
}
```

4. 一维前缀

```
vector<ll> arr(maxn), pre(maxn);

// longest subarray whose sum equals a target value.
ll lon_sub(ll goal, ll n)
{
    ll ans = 0;
    map<ll, ll> mp;
    mp[0] = 0;
    for (ll i = 1; i <= n; i++)
    {
        pre[i] = arr[i] + pre[i - 1];
```

```
            if (!mp.count(pre[i]))
            {
                mp[pre[i]] = i;
            }
            cout << "pre[" << i << "]:" << pre[i] << endl;
        }

        for (ll i = 1; i <= n; i++)
        {
            if (mp.count(pre[i] - goal))
            {
                ans = max(ans, i - mp[pre[i] - goal]);
            }
        }

        pre.clear();
        return ans;
}

// number of subarrays whose sum equals a target value.
ll num_sub(ll goal, ll n) // goal != 0
{
        ll ans = 0;
        map<ll, ll> mp;
        mp[0] = 1;

        for (ll i = 1, sum = 0; i <= n; i++)
        {
            sum += arr[i];
            ans += mp[sum - goal];
            mp[sum]++;
        }

        pre.clear();
        return ans;
}

// longest subarray with an equal number of positive and negative numbers.
ll equ_sub(ll n)
{
        ll ans = 0;
        for (ll i = 1, sum = 0; i <= n; i++)
        {
            arr[i] > 0 ? sum += 1 : sum -= 1;
            if (!sum)
            {
                ans = max(ans, i);
            }
        }

        pre.clear();
        return ans;
}

// longest subarray with a sum greater than 0.
ll gre_sub(ll n)
{
        ll ans = 0;
        map<ll, ll> mp;
        mp[0] = 0;

        for (ll i = 1, sum = 0; i <= n; i++)
        {
            arr[i] > 0 ? sum += 1 : sum -= 1;

            if (sum > 0)
            {
```

```
            ans = i;
        }
        else
        {
            // -1 -1 -1 -1 -1   1   1 -1   1
            // -1 -2 -3 -4 -5 -4 -3 -4 -3
            if (mp.count(sum - 1))
            {
                ans = max(ans, i - mp[sum - 1]);
            }

            if (!mp.count(sum))
            {
                mp[sum] = i;
            }
        }
    }

    pre.clear();
    return ans;
}

// remove the shortest subarray so that the sum of the remaining elements can be divided by p.
ll div_sub(ll n, ll p)
{
    ll ans = LLONG_MAX;
    map<ll, ll> mp; // key means prefix sums modulo p, and value means the latest position of that remainder.
    mp[0] = 0;

    ll mod = 0;
    for (ll i = 1; i <= n; i++)
    {
        mod = (mod + arr[i]) % p;
    }
    cout << "mod=" << mod << endl;
    if (mod == 0)
    {
        return 0;
    }

    for (ll i = 1, cur = 0, find; i <= n; i++)
    {
        cur = (cur + arr[i]) % p;
        find = cur >= mod ? (cur - mod) : (cur + p - mod);

        if (mp.count(find))
        {
            ans = min(ans, i - mp[find]);
        }

        mp[cur] = i;
    }

    pre.clear();
    arr.clear();
    return ans;
}

// longest substring where the count of each vowel is even.
ll move(char ch)
{
    switch (ch)
    {
        case 'a': return 0;
        case 'e': return 1;
        case 'i': return 2;
        case 'o': return 3;
```

```
            case 'u': return 4;
            default: return -1;
        }
    }

    ll even_sub(string s)
    {
        ll n = s.length(), ans = 0;
        // u o i e a     status
        // 0 0 0 0 0
        vector<ll> mp(32, -2); // all states have not appeared yet.
        mp[0] = -1;

        for (ll i = 0, status = 0, m; i < n; i++)
        {
            m = move(s[i]);
            if (m != -1)
            {
                status ^= (1LL << m);
            }

            if (mp[status] != -2)
            {
                ans = max(ans, i - mp[status]);
            }
            else
            {
                mp[status] = i;
            }
        }
        return ans;
    }
```

5. 一维差分（等差数列差分）

```
vector<ll> arr;
ll offset = 0; // avoid negative array position
void init(ll n)
{
    arr.assign(n + 3 + offset, 0);
}

// difference of an arithmetic progression
// [l, r] + {a + (n-1)*d}
void modify(ll l, ll r, ll s, ll e, ll d)
{
    // s d-s ... -d-e e
    // s   d   ...   -e   0
    // s s+d ...     0   0
    arr[l + offset] += s;
    arr[l + 1 + offset] += d - s;
    arr[r + 1 + offset] -= d + e;
    arr[r + 2 + offset] += e;
}

void build(ll n)
{
    for (ll i = 1; i <= n + offset; i++)
    {
        arr[i] += arr[i - 1];
    }
    for (ll i = 1; i <= n + offset; i++)
    {
        arr[i] += arr[i - 1];
    }
}
```

队列与栈

1. 基本操作

```
struct circular_queue
{
    vector<ll> data;
    ll front;
    ll rear;
    ll capacity;

    circular_queue(ll size)
    {
        data.resize(size);
        front = 0;
        rear = 0;
        capacity = size;
    }

    // the circular queue can only store capacity - 1 elements to distinguish full and empty states
    bool is_full()
    {
        return (rear + 1) % capacity == front;
    }

    bool is_empty()
    {
        return front == rear;
    }

    void enqueue(ll value)
    {
        if (is_full())
        {
            cout << "full\n";
            return;
        }
        data[rear] = value;
        rear = (rear + 1) % capacity;
    }

    ll dequeue()
    {
        if (is_empty())
        {
            cout << "empty\n";
            return -1;
        }
        ll value = data[front];
        front = (front + 1) % capacity;
        return value;
    }

    ll peek()
    {
        if (is_empty())
        {
            cout << "empty\n";
            return -1;
        }
        return data[front];
    }
};

struct stack_sim_queue
{
    stack<ll> stack_in, stack_out;

    void enqueue(ll value)
```

```
    {
        stack_in.push(value);
    }

    // transfer elements from stack_in to stack_out
    void IntoOut()
    {
        // only if the stack_out is empty, can we transfer elements from stack_in to stack_out
        if (stack_out.empty())
        {
            // if we transfer elements from stack_in to stack_out, the stack_in must become empty
            while (!stack_in.empty())
            {
                stack_out.push(stack_in.top());
                stack_in.pop();
            }
        }
    }

    ll dequeue()
    {
        IntoOut();
        if (stack_out.empty())
        {
            cout << "empty\n";
            return -1;
        }
        ll value = stack_out.top();
        stack_out.pop();
        return value;
    }

    ll peek()
    {
        IntoOut();
        if (stack_out.empty())
        {
            cout << "empty\n";
            return -1;
        }
        return stack_out.top();
    }

    bool is_empty()
    {
        return stack_in.empty() && stack_out.empty();
    }
};

struct queue_sim_stack
{
    queue<ll> q;

    void push(ll value)
    {
        ll size = q.size();
        q.push(value);
        // rotate the queue to the front
        for (ll i = 0; i < size - 1; ++i)
        {
            q.push(q.front());
            q.pop();
        }
    }

    ll pop()
    {
```

```
        if (q.empty())
        {
            cout << "empty\n";
            return -1;
        }
        ll value = q.front();
        q.pop();
        return value;
    }

    ll top()
    {
        if (q.empty())
        {
            cout << "empty\n";
            return -1;
        }
        return q.front();
    }

    ll empty()
    {
        return q.empty();
    }
};

signed main()
{
    circular_queue cq(5);
    cq.enqueue(1);
    cq.enqueue(2);
    cq.enqueue(3);
    cout << "Dequeue: " << cq.dequeue() << endl; // 1
    cout << "Peek: " << cq.peek() << endl;           // 2
    cq.enqueue(4);
    cq.enqueue(5);
    cq.enqueue(6);
    cout << "Dequeue: " << cq.dequeue() << endl; // 2

    stack_sim_queue ssq;
    ssq.enqueue(10);
    ssq.enqueue(20);
    cout << "Dequeue: " << ssq.dequeue() << endl; // 10
    cout << "Peek: " << ssq.peek() << endl;           // 20

    queue_sim_stack qss;
    qss.push(100);
    qss.push(200);
    cout << "Pop: " << qss.pop() << endl; // 200
    cout << "Top: " << qss.top() << endl; // 100

    return 0;
}
```

2. 双端队列

```
struct dequeue
{
    ll *arr;
    ll size, capacity, front, back;

    dequeue(ll cap) : size(0), capacity(cap), front(0), back(-1)
    {
        arr = new ll[capacity];
    }

    ~dequeue()
    {
```

```
    delete[] arr;
}

bool is_full()
{
    return size == capacity;
}

void push_front(ll x)
{
    if (!is_full())
    {
        // (x - 1 + capacity) % capacity ensures that front wraps around when it reaches the beginning
        front = (front - 1 + capacity) % capacity;
        arr[front] = x;
        size++;
    }
    else
    {
        cout << "full\n";
    }
}

void push_back(ll x)
{
    if (!is_full())
    {
        // (x + 1) % capacity ensures that back wraps around when it reaches the end
        back = (back + 1) % capacity;
        arr[back] = x;
        size++;
    }
    else
    {
        cout << "full\n";
    }
}

void pop_front()
{
    if (size > 0)
    {
        front = (front + 1) % capacity;
        size--;
    }
    else
    {
        cout << "empty\n";
    }
}

void pop_back()
{
    if (size > 0)
    {
        back = (back - 1 + capacity) % capacity;
        size--;
    }
    else
    {
        cout << "empty\n";
    }
}

void display()
{
    for (ll i = 0; i < size; i++)
```

```
        {
            cout << arr[(front + i) % capacity] << " ";
        }
        cout << endl;
    }

    void clear()
    {
        front = 0;
        back = -1;
        size = 0;
    }

    void front_value()
    {
        if (size > 0)
            cout << arr[front] << endl;
        else
            cout << "empty\n";
    }

    void back_value()
    {
        if (size > 0)
            cout << arr[back] << endl;
        else
            cout << "empty\n";
    }

    void get_size()
    {
        cout << size << endl;
    }

    bool is_empty()
    {
        return size == 0;
    }
};
```

3. 最小栈

```
struct mystack
{
    stack<ll> min, data;
    mystack() {}

    // eg: 3 5 2 7
    // min: 3 3 2 2
    void push(ll value)
    {
        data.push(value);
        if (min.empty() || value <= min.top())
        {
            min.push(value);
        }
        else
        {
            min.push(min.top());
        }
    }

    void pop()
    {
        if (data.empty())
        {
            cout << "empty\n";
            return;
```

```
        }
        data.pop();
        min.pop();
    }

    ll get_min()
    {
        if (min.empty())
        {
            cout << "empty\n";
            return -1;
        }
        return min.top();
    }

    ll top()
    {
        if (data.empty())
        {
            cout << "empty\n";
            return -1;
        }
        return data.top();
    }
};
```

4. 单调栈

```
vector<vector<ll>> mon_stack_rep(const vector<ll> &arr)
{
    ll n = arr.size(), top = 0;
    vector<vector<ll>> ans(n, vector<ll>(2));
    ll s[n + 1];
    s[0] = -1; // top = 0, empty

    for (ll i = 0; i < n; i++)
    {
        while (top > 0 && arr[i] <= arr[s[top]])
        {
            ll t = s[top--];
            ans[t][1] = i;
            ans[t][0] = s[top];
        }
        s[++top] = i;
    }

    while (top > 0)
    {
        ll t = s[top--];
        ans[t][1] = -1;
        ans[t][0] = s[top];
    }

    for (ll i = n - 2; i >= 0; i--)
    {
        if (ans[i][1] != -1 && arr[i] == arr[ans[i][1]])
        {
            ans[i][1] = ans[ans[i][1]][1];
        }
    }

    return ans;
}

// sum of minimums of all subarrays
ll sum_min(const vector<ll> &arr)
{
    ll res = 0;
```

```
    ll n = arr.size(), top = 0;
    // this problem essentially doesn't require an ans array; only two variables are needed.
    vector<vector<ll>> ans(n, vector<ll>(2));
    ll s[n + 1];
    s[0] = -1; // top = 0, empty

    for (ll i = 0; i < n; i++)
    {
        while (top > 0 && arr[i] <= arr[s[top]])
        {
            ll t = s[top--];
            ans[t][1] = i;
            ans[t][0] = s[top];

            res = (res + (t - ans[t][0]) * (ans[t][1] - t) * arr[t]) % mod;
            // ans = max(ans, height[t] * (ans[t][1] - ans[t][0] -1))
        }
        s[++top] = i;
    }

    while (top > 0)
    {
        ll t = s[top--];
        ans[t][1] = -1;
        ans[t][0] = s[top];

        res = (res + (t - ans[t][0]) * (n - t) * arr[t]) % mod;
        // ans = max(ans, height[t] * (n - ans[t][0] -1))
    }

    // for (ll i = n - 2; i >= 0; i--)
    // {
    //     if (ans[i][1] != -1 && arr[i] == arr[ans[i][1]])
    //     {
    //         ans[i][1] = ans[ans[i][1]][1];
    //     }
    // }

    return res;
}
```

5. 最大宽度坡问题

```
// (i, j), A[i] <= A[j], then max(j - i)
ll max_ramp(const vector<ll> &arr)
{
    ll n = arr.size(), top = 0;
    ll s[n];
    s[top++] = 0;

    for (ll i = 1; i < n; i++)
    {
        if (arr[i] < arr[s[top - 1]])
        {
            s[top++] = i;
        }
    }

    ll ans = 0;
    for (ll i = n - 1; i >= 0; i--)
    {
        while (top > 0 && arr[i] >= arr[s[top - 1]])
        {
            ans = max(ans, i - s[top - 1]);
            top--;
        }
    }
```

```
    return ans;
}
```

6. 去重重复字母，且剩余字母的字典序最小

```
// remove duplicate letters to ensure the lexicographically smallest result.
string dup_lex(string str)
{
    ll cnt[256] = {};
    char s[27];
    bool vis[27];
    memset(vis, 0, sizeof(vis));
    for (char ch : str)
    {
        cnt[ch]++;
    }

    ll top = 0;
    for (char ch : str)
    {
        if (!vis[ch])
        {
            while (top > 0 && ch < s[top - 1] && cnt[s[top - 1]] > 0)
            {
                vis[s[--top]] = false;
            }

            s[top++] = ch;
            vis[ch] = true;
        }

        cnt[ch]--;
    }

    string ans = "";
    for (ll i = 0; i < top; i++)
    {
        ans += s[i];
    }
    return ans;
}
```

7. 大于吃小鱼

```
// each round, the fish will simultaneously eat to their right the first fish that is smaller than themselves, after how many rounds will the number of fish remain unchanged?
ll big_fish(const vector<ll> &arr)
{
    ll n = arr.size(), ans = 0, top = 0;
    ll s[n][2];

    for (ll i = n - 1, turns; i >= 0; i--)
    {
        turns = 0;
        while (top > 0 && arr[i] > s[top - 1][0])
        {
            turns = max(turns + 1, s[--top][1]);
        }

        s[top][0] = arr[i];
        s[top++][1] = turns;

        ans = max(ans, turns);
    }

    return ans;
}
```

8. 统计 0/1 矩阵中内部全部是 1 的子矩阵的数量

```
// 0/1 matrix, count the number of submatrices with all 1s.
ll histogram(const vector<ll> &hist)
{
    ll m = hist.size(), top = 0, ans = 0;
    ll s[m];

    for (ll i = 0, l, len, high; i < m; i++)
    {
        while (top > 0 && hist[i] <= hist[s[top - 1]])
        {
            ll t = s[--top];
            if (hist[t] > hist[i])
            {
                l = (top != 0 ? s[top - 1] : -1);
                len = i - l - 1;
                high = max(l != -1 ? hist[l] : 0, hist[i]);

                ans += (hist[t] - high) * len * (len + 1) / 2;
            }
        }
        s[top++] = i;
    }

    while (top > 0)
    {
        ll t = s[--top];
        ll l = (top != 0 ? s[top - 1] : -1);
        ll len = m - l - 1;
        ll high = l != -1 ? hist[l] : 0;

        ans += (hist[t] - high) * len * (len + 1) / 2;
    }

    return ans;
}

ll submatric(const vector<vector<ll>> &mat)
{
    ll ans = 0;
    ll n = mat.size();
    ll m = mat[0].size();
    vector<ll> hight(m, 0);

    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            hight[j] = (mat[i][j] == 0 ? 0 : hight[j] + 1);
        }
        ans += histogram(hight);
    }

    return ans;
}
```

9. 单调队列

```
// monotonic queue to maintain maximum and minimum in a sliding window.
vector<ll> max_slid(const vector<ll> &arr, ll k)
{
    ll n = arr.size(), h = 0, t = 0;
    ll dq[n]; // [h, t)

    for (ll i = 0; i < k - 1; i++)
    {
        while (h < t && arr[i] >= arr[dq[t - 1]])
        {
            t--;
```

```
        }
        dq[t++] = i;
    }

    vector<ll> ans;
    for (ll l = 0, r = k - 1; r < n; l++, r++)
    {
        while (h < t && arr[r] >= arr[dq[t - 1]])
        {
            t--;
        }
        dq[t++] = r;

        ans.push_back(arr[dq[h]]);
        // l is about to slide away, check if the head of the monotonic queue has expired.
        dq[h] != l ? h : h++;
    }

    return ans;
}

// longest subarray where the difference between its maximum and minimum elements is less than or equal to limit.
const ll maxn = 100005;
ll dq_max[maxn], dq_min[maxn];
ll hx = 0, tx = 0, hn = 0, tn = 0;
bool ok(const vector<ll> &arr, ll limit, ll num)
{
    ll max = hx < tx ? std::max(arr[dq_max[hx]], num) : num;
    ll min = hn < tn ? std::min(arr[dq_min[hn]], num) : num;
    return (max - min) > limit ? false : true;
}
ll long_sub(const vector<ll> &arr, ll limit)
{
    ll n = arr.size(), ans = 0;

    for (ll l = 0, r = 0; l < n; l++)
    {
        while (r < n && ok(arr, limit, arr[r]))
        {
            while (hx < tx && arr[r] >= arr[dq_max[tx - 1]])
            {
                tx--;
            }
            dq_max[tx++] = r;

            while (hn < tn && arr[r] <= arr[dq_min[tn - 1]])
            {
                tn--;
            }
            dq_min[tn++] = r;

            r++;
        }
        ans = std::max(ans, r - l);

        (hx < tx && dq_max[hx] == l) ? hx++ : hx;
        (hn < tn && dq_min[hn] == l) ? hn++ : hn;
    }

    return ans;
}
```

10. 单调队列应用

```
// shortest subarray with a sum greater than or equal to k.
ll sho_sub(const vector<ll> &arr, ll k)
{
    // monotonic queue maintains the possibilities for becoming a left endpoint.
```

```
    ll n = arr.size(), h = 0, t = 0, ans = LLONG_MAX;
    ll dq[n + 1], sum[n + 1];
    memset(sum, 0, sizeof(sum));

    for (ll i = 0; i < n; i++)
    {
        // sum[0] = 0
        sum[i + 1] = sum[i] + arr[i];
    }

    for (ll i = 0; i <= n; i++)
    {
        while (h < t && sum[i] - sum[dq[h]] >= k)
        {
            ans = min(ans, i - dq[h++]);
        }

        while (h < t && sum[i] <= sum[dq[t - 1]])
        {
            t--;
        }

        dq[t++] = i;
    }

    return ans != LLONG_MAX ? ans : -1;
}

// points[i] = [xi, yi]; i < j → xi < xj; xj - xi <= k → max(yi + yj + |xi - xj|)
ll poin_max(vector<vector<ll>> &points, ll k)
{
    ll n = points.size(), h = 0, t = 0, ans = LLONG_MIN;
    ll dq[n][2]; // (x, y)

    // yi + yj + |xi - xj| → (xj + yj) + (yi - xi)
    for (ll i = 0, x, y; i < n; i++)
    {
        y = points[i][1], x = points[i][0];
        while (h < t && x - dq[h][0] > k)
        {
            h++;
        }

        if (h < t)
        {
            ans = max(ans, x + y + dq[h][1] - dq[h][0]);
        }

        while (h < t && y - x >= dq[t - 1][1] - dq[t - 1][0])
        {
            t--;
        }

        dq[t][0] = x;
        dq[t++][1] = y;
    }
}

// given n tasks and m workers, each worker can complete at most one task with a value greater than or equal to
their strength. you have x pills, each providing the same strength boost,return    the maximum number of tasks that
can be completed?
bool f(vector<ll> &task, vector<ll> &ker, ll pil, ll stg, ll tl, ll tr, ll kl, ll kr)
{
    // f returns whether it's possible to complete num tasks, using a greedy approach: matching the strongest
workers with the easiest tasks.
    ll dq[10001];
    ll h = 0, t = 0, cnt = 0;
```

```
    for (ll i = kl, j = tl; i <= kr; i++)
    {
        while (j <= tr && ker[i] >= task[j])
        {
            dq[t++] = task[j++];
        }

        if (t > h && ker[i] >= dq[h])
        {
            h++;
        }
        else
        {
            cnt++;
            if (cnt > pil)
            {
                return false;
            }

            while (j <= tr && ker[i] + stg >= task[j])
            {
                dq[t++] = task[j++];
            }

            if (t > h)
            {
                t--;
            }
            else
            {
                return false;
            }
        }
    }

    return true;
}
ll task_ass(vector<ll> &task, vector<ll> &ker, ll pil, ll stg)
{
    ll n = task.size(), m = ker.size();
    ll l = 0, r = min(n, m), ans = 0;

    sort(task.begin(), task.end());
    sort(ker.begin(), ker.end());

    while (l <= r)
    {
        ll mid = (l + r) / 2;
        if (f(task, ker, pil, stg, 0, mid - 1, m - mid, m - 1))
        {
            ans = mid;
            l = mid + 1;
        }
        else
        {
            r = mid - 1;
        }
    }

    return ans;
}
```

递归

1. master 公式

```
ll arr[1000001];

ll f(ll l, ll r)
{
    if (l == r)
    {
        return arr[l];
    }
    ll mid = (l + r) / 2;

    // master: T(N) = a * T(N/b) + O(N^c), sub-processes must be of equal scale
    // log(b a) > c, then T(N) = O(N^log(b a))
    // log(b a) < c, then T(N) = O(N^c)
    // log(b a) = c, then T(N) = O(N^c * log N)
    // where a = 2, b = 2, c = 0
    // T(N) = 2 * T(N/2) + O(1)
    // T(N) = O(N^log(2 2)) = O(N^1) = O(N)

    // especially, T(N) = 2 * T(N/2) + O(N * log N)
    // T(N) = O(N * (log N)^2)

    ll lmax = f(l, mid);
    ll rmax = f(mid + 1, r);
    return max(lmax, rmax);
}

signed main()
{
    ll n;
    cin >> n;
    for (ll i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    ll ans = f(0, n - 1);
    cout << ans << endl;
    return 0;
}
```

2. 字符串的所有子序列

```
// all subsequences of a string
void f(const string &s, ll i, string path, set<string> &ans)
{
    if (i == s.length())
    {
        ans.insert(path);
        return;
    }

    f(s, i + 1, path + s[i], ans);
    f(s, i + 1, path, ans);
}

vector<string> subsequences(string s)
{
    set<string> ans;
    string path = "";

    f(s, 0, path, ans);

    vector<string> result(ans.begin(), ans.end());
    return result;
}
```

3. 数组的所有组合可能

```
vector<ll> path(maxn);

void f(const vector<ll> &arr, ll i, ll size, list<list<ll>> &ans)
{
    if (i == arr.size())
    {
        list<ll> l;
        for (ll j = 0; j < size; j++)
        {
            l.push_back(path[j]);
        }
        ans.push_back(l);
    }
    else
    {
        ll j = i + 1;
        while (j < arr.size() && arr[i] == arr[j])
        {
            j++;
        }

        f(arr, j, size, ans);
        for (ll k = i; k < j; k++)
        {
            path[size++] = arr[k];
            f(arr, j, size, ans);
        }
    }
}

list<list<ll>> subsets(vector<ll> arr)
{
    list<list<ll>> ans;
    sort(arr.begin(), arr.end());
    f(arr, 0, 0, ans);
    return ans;
}
```

4. 全排列

```
void f(vector<ll> &arr, ll i, list<list<ll>> &ans)
{
    if (i == arr.size())
    {
        list<ll> l;
        for (ll val : arr)
        {
            l.push_back(val);
        }
        ans.push_back(l);
    }
    else
    {
        for (ll j = i; j < arr.size(); j++)
        {
            swap(arr[i], arr[j]);
            f(arr, i + 1, ans);
            swap(arr[i], arr[j]);
        }
    }
}

// repeat number
void fr(vector<ll> &arr, ll i, list<list<ll>> &ans)
{
    if (i == arr.size())
    {
        list<ll> l;
```

```
            for (ll val : arr)
            {
                l.push_back(val);
            }
            ans.push_back(l);
        }
        else
        {
            set<ll> s;
            for (ll j = i; j < arr.size(); j++)
            {
                // attempt will only be made when arr[j] first arrives at position i.
                if (!s.count(arr[j]))
                {
                    s.insert(arr[j]);
                    swap(arr[i], arr[j]);
                    fr(arr, i + 1, ans);
                    swap(arr[i], arr[j]);
                }
            }
        }
}

list<list<ll>> permulations(vector<ll> arr)
{
    list<list<ll>> ans;
    // f(arr, 0, ans);
    fr(arr, 0, ans);
    return ans;
}
```

5. 递归实现逆序栈

```
// pop the bottom element of a stack while maintaining the order of the other elements.
ll bottom_pop(stack<ll> &s)
{
    ll ans = s.top();
    s.pop();
    if (s.empty())
    {
        return ans;
    }
    else
    {
        ll last = bottom_pop(s);
        s.push(ans);
        return last;
    }
}

void reverse(stack<ll> &s)
{
    if (s.empty())
    {
        return;
    }

    ll num = bottom_pop(s);
    reverse(s);
    s.push(num);
}
```

6. 递归实现栈排序

```
ll depth(stack<ll> &s)
{
    if (s.empty())
    {
        return 0;
```

```
    }

    ll num = s.top();
    s.pop();
    ll d = depth(s) + 1;
    s.push(num);

    return d;
}

ll stack_max(stack<ll> &s, ll depth)
{
    if (depth == 0)
    {
        return LLONG_MIN;
    }

    ll num = s.top();
    s.pop();
    ll m = max(num, stack_max(s, depth - 1));
    s.push(num);

    return m;
}

ll times(stack<ll> &s, ll depth, ll max)
{
    if (depth == 0)
    {
        return 0;
    }

    ll t;
    ll num = s.top();
    s.pop();
    ll ut = times(s, depth - 1, max);
    num == max
        ? t = ut + 1
        : t = ut;
    s.push(num);

    return t;
}

void down(stack<ll> &s, ll depth, ll max, ll times)
{
    if (depth == 0)
    {
        for (ll i = 0; i < times; i++)
        {
            s.push(max);
        }
    }
    else
    {
        ll num = s.top();
        s.pop();
        down(s, depth - 1, max, times);
        if (num != max)
        {
            s.push(num);
        }
    }
}

void sort(stack<ll> &s)
{
```

```
    ll d = depth(s);
    while (d > 0)
    {

        ll m = stack_max(s, d);
        ll k = times(s, d, m);
        down(s, d, m, k);
        cout << "d=" << d << ",m=" << m << ",k=" << k << endl;
        d -= k;
    }
}
```

7. 汉若塔

```
// f(i, from, to, other), means 1~i move from → to
// f(i-1, from, other, to) + i move from → to + f(i-1, other, to, from)
void f(ll i, string from, string to, string other)
{
    if (i == 1)
    {
        cout << "bang " << i << " from " << from << " move to " << to << endl;
    }
    else
    {
        f(i - 1, from, other, to);
        cout << "bang " << i << " from " << from << " move to " << to << endl;
        f(i - 1, other, to, from);
    }
}

void hanoi(ll n)
{
    f(n, "left", "mid", "right");
}
```

集合与哈希表

1. 自定义排序（集合与哈希表）

```
void demonstrate_set()
{
    set<ll> s;
    // O(log N)
    s.insert(30);
    s.insert(10);
    s.insert(20);
    s.insert(10); // repeat, fail

    // traversal
    for (ll val : s)
    {
        cout << val << " ";
    }
    cout << endl; // 10 20 30

    cout << "set size: " << s.size() << endl;
    cout << "set empty? " << s.empty() << endl;

    // O(log N)
    ll x;
    cin >> x;
    // count() returns the number of occurrences of x in the set (0 or 1 for set)
    if (s.count(x))
    {
        cout << x << " exists." << endl;
    }
    // find() returns iterator to the element if found, or s.end() if not found
    auto it_found = s.find(x);
    if (it_found != s.end())
    {
        cout << "found " << x << "at (dereferenced) iterator: " << *it_found << endl;
    }

    // O(log N)
    s.erase(x);
    s.erase(--s.end());
    cout << "erasing " << x << ": ";
    for (ll val : s)
    {
        cout << val << " ";
    }
    cout << endl;

    // O(log N)
    it_found = s.find(x);
    if (it_found != s.end())
    {
        s.erase(it_found);
    }
    cout << "erasing " << x << " by iterator: ";
    for (ll val : s)
    {
        cout << val << " ";
    }
    cout << endl;

    // lower_bound/upper_bound
    // O(log N)
    s.insert(15);
    s.insert(25);
    s.insert(35);
    cout << "current set: ";
    for (ll val : s)
    {
```

```
        cout << val << " ";
    }
    cout << endl;

    auto lb_it = s.lower_bound(x); // the first element not less than x, else return s.end()
    auto ub_it = s.upper_bound(x); // the first element greater than x, else return s.end()
    cout << "lower_bound(" << x << "): " << *lb_it << endl;
    cout << "upper_bound(" << x << "): " << *ub_it << endl;

    // O(N)
    s.clear();

    // space: O(N)
}

void demonstrate_unordered_set()
{
    // average time: O(1), worst case: O(N)
    unordered_set<string> us;
    us.insert("apple");
    us.insert("banana");
    us.insert("cherry");
    us.insert("apple"); // repeat, fail

    cout << "unordered set elements (order not guaranteed): ";
    for (const string &s : us)
    {
        cout << s << " ";
    }
    cout << endl;

    // O(1)
    cout << "unordered set size: " << us.size() << endl;

    // average: O(1) ; worst case: O(N)
    if (us.count("banana"))
    {
        cout << "banana is in the set." << endl;
    }
    auto it_found = us.find("cherry");
    if (it_found != us.end())
    {
        cout << "found cherry." << endl;
    }

    // average: O(1) ; worst case: O(N)
    us.erase("banana");
    cout << "unordered set after erasing banana: ";
    for (const string &s : us)
    {
        cout << s << " ";
    }
    cout << endl;

    // space: O(N)
}

void demonstrate_map()
{
    // O(log N)
    map<string, ll> ages;
    ages["alice"] = 30;
    ages["bob"] = 25;
    // O(log N)
    ages.insert({"charlie", 35});
    ages.insert(make_pair("alice", 31)); // key exists, won't update value
    ages["bob"] = 26;                    // key exists, update value
```

```
    // O(N)
    cout << "map elements (sorted by key):" << endl;
    for (const auto &pair : ages)
    {
        cout << pair.first << ": " << pair.second << endl;
    }

    // O(log N)
    // use "[]" to access, the key does not exist, it will insert a new element.
    cout << "alice's age: " << ages["alice"] << endl;
    // use at() to access, the key does not exist, it will throw std::out_of_range exception.
    try
    {
        cout << "david's age: " << ages.at("david") << endl;
    }
    catch (const out_of_range &oor)
    {
        cerr << "error: " << oor.what() << endl;
    }

    // O(log N)
    if (ages.count("bob"))
    {
        cout << "bob is in the map with age: " << ages["bob"] << endl;
    }
    auto it_found = ages.find("charlie");
    if (it_found != ages.end())
    {
        cout << "found charlie, age: " << it_found->second << endl; // 通过 ->second 访问值
    }

    // O(log N)
    ages.erase("bob");
    cout << "map after erasing bob:" << endl;
    for (const auto &pair : ages)
    {
        cout << pair.first << ": " << pair.second << endl;
    }

    // lower_bound/upper_bound
    // O(log N)
    map<char, ll> cm = {{'a', 1}, {'c', 3}, {'e', 5}, {'g', 7}};
    auto lb_it_char = cm.lower_bound('d'); // returns the key not less than 'd', or cm.end()
    auto ub_it_char = cm.upper_bound('d'); // returns the key greater than 'd', or cm.end()
    cout << "cm lower_bound('d'): " << lb_it_char->first << endl;
    cout << "cm upper_bound('d'): " << ub_it_char->first << endl;

    // space: O(N)
}

void demonstrate_unordered_map()
{
    // average: O(1), worst case: O(N)
    unordered_map<string, ll> ages;
    ages["alice"] = 30;
    ages["bob"] = 25;
    ages.insert({"charlie", 35});
    ages.insert({"alice", 31}); // key exists, won't update value
    ages["bob"] = 26;                // key exists, update value

    cout << "unordered map elements (order not guaranteed by key):" << endl;
    for (const auto &pair : ages)
    {
        cout << pair.first << ": " << pair.second << endl;
    }
```

```
    // average: O(1), worst case: O(N)
    cout << "alice's age: " << ages["alice"] << endl;
    try
    {
        cout << "david's age (using at()): " << ages.at("david") << endl;
    }
    catch (const out_of_range &oor)
    {
        cerr << "error: " << oor.what() << endl;
    }

    // average: O(1), worst case: O(N)
    if (ages.count("bob"))
    {
        cout << "bob is in the map with age: " << ages["bob"] << endl;
    }
    auto it_found = ages.find("charlie");
    if (it_found != ages.end())
    {
        cout << "found charlie, age: " << it_found->second << endl;
    }

    // average: O(1), worst case: O(N)
    ages.erase("bob");
    cout << "unordered map after erasing bob:" << endl;
    for (const auto &pair : ages)
    {
        cout << pair.first << ": " << pair.second << endl;
    }

    // space: O(N)
}

struct person
{
    string name;
    ll age;
    double height;

    person(string n, ll a, double h) : name(n), age(a), height(h) {}

    friend ostream &operator<<(ostream &os, const person &p)
    {
        os << "[" << p.name << ", " << p.age << " yrs, " << p.height << "m]";
        return os;
    }
};

struct GreaterInt
{
    bool operator()(ll a, ll b) const
    {
        return a > b;
    }
};

struct CompareStringByLength
{
    bool operator()(const string &s1, const string &s2) const
    {
        if (s1.length() != s2.length())
        {
            return s1.length() < s2.length();
        }
        return s1 < s2;
    }
};
```

```
struct ComparepersonByAgeDesc
{
    bool operator()(const person &p1, const person &p2) const
    {
        if (p1.age != p2.age)
        {
            return p1.age > p2.age;
        }
        if (p1.name != p2.name)
        {
            return p1.name < p2.name;
        }
        return p1.height < p2.height;
    }
};

struct ComparepersonKeyByAgeDesc
{
    bool operator()(const person &p1, const person &p2) const
    {
        if (p1.age != p2.age)
        {
            return p1.age > p2.age;
        }
        if (p1.name != p2.name)
        {
            return p1.name < p2.name;
        }
        return p1.height < p2.height;
    }
};

template <typename T>
void print_set(const T &s, const string &title)
{
    cout << title << ": { ";
    for (const auto &val : s)
    {
        cout << val << " ";
    }
    cout << "}" << endl;
}

signed main()
{
    demonstrate_set();
    demonstrate_unordered_set();
    demonstrate_map();
    demonstrate_unordered_map();

    set<ll, greater<ll>> s_desc;
    s_desc.insert(10);
    s_desc.insert(50);
    s_desc.insert(20);
    s_desc.insert(5);
    print_set(s_desc, "std::set<ll> (des)");

    set<string, CompareStringByLength> s_str_len;
    s_str_len.insert("apple");   // length 5
    s_str_len.insert("cat");     // length 3
    s_str_len.insert("banana"); // length 6
    s_str_len.insert("dog");     // length 3
    s_str_len.insert("zebra");   // length 5
    print_set(s_str_len, "std::set<string> (length)");
    // { cat dog apple zebra banana }
```

```
    map<string, ll> scores = {
        {"alice", 90}, {"bob", 85}, {"charlie", 90}, {"david", 95}};

    vector<pair<string, ll>> vec_scores(scores.begin(), scores.end());

    sort(vec_scores.begin(), vec_scores.end(), [](const pair<string, ll> &p1, const pair<string, ll> &p2)
         {
             if (p1.second != p2.second)
             {
                 return p1.second > p2.second;
             }
             return p1.first < p2.first; });

    for (const auto &p : vec_scores)
    {
        cout << "   " << p.first << " : " << p.second << endl;
    }
    //    david : 95
    //    alice : 90
    //    charlie : 90
    //    bob : 85

    return 0;
}
```

2. setall 哈希表

```
// hash table with a setall function.
struct setall_hash
{
    map<ll, pair<ll, ll>> mp;
    ll setall_value;
    ll setall_time;
    ll time_stamp;

    setall_hash() : setall_value(0), setall_time(-1), time_stamp(0) {}

    void insert(ll k, ll v)
    {
        if (mp.count(k))
        {
            pair<ll, ll> &value = mp[k];
            value.first = v;
            value.second = time_stamp++;
        }
        else
        {
            mp[k] = make_pair(v, time_stamp++);
        }
    }

    void setall(ll v)
    {
        setall_time = time_stamp++;
        setall_value = v;
    }

    ll get_value(ll k)
    {
        pair<ll, ll> &value = mp[k];
        if (value.second > setall_time)
        {
            return value.first;
        }
        else
        {
            return setall_value;
        }
```

```
    }
};

signed main()
{
    setall_hash sh;
    sh.insert(1, 10);
    sh.insert(2, 20);
    sh.insert(3, 30);

    cout << "1: " << sh.get_value(1) << endl;
    cout << "2: " << sh.get_value(2) << endl;
    cout << "3: " << sh.get_value(3) << endl;

    sh.setall(100);
    cout << "\nsetall(100): " << endl;
    cout << "1: " << sh.get_value(1) << endl;
    cout << "2: " << sh.get_value(2) << endl;
    cout << "3: " << sh.get_value(3) << endl;

    sh.insert(2, 200);
    sh.insert(4, 400);
    cout << "\ninsert(2, 200): " << endl;
    cout << "2: " << sh.get_value(2) << endl; // 200
    cout << "1: " << sh.get_value(1) << endl; // 100
    cout << "4: " << sh.get_value(4) << endl; // 400

    return 0;
}
```

排序

1. 堆排序

```
ll arr[maxn];
// complete binary tree implemented using an array, the parent node of each node i is at index (i-1)/2, its left child is at 2*i+1, and its right child is at 2*i+2.
//     (i-1)/2
//          \
//           i
//          / \
// 2*i+1     2*i+2

void heapify(ll i, ll size)
{
    ll l = 2 * i + 1;
    // l < size means the left child exists
    while (l < size)
    {
        ll max_son, dex;
        l + 1 < size ? max_son = max(arr[l], arr[l + 1]) : max_son = arr[l];
        max_son == arr[l] ? dex = l : dex = l + 1;
        if (arr[i] < max_son)
        {
            swap(arr[i], arr[dex]);
            i = dex;
        }
        else
        {
            break;
        }
        l = 2 * i + 1;
    }
}

// arr[i] = x
void heap_insert(ll i)
{
    // when reach the top of the heap, 0 = (0-1)/2, stop
    while (arr[i] > arr[(i - 1) / 2])
    {
        swap(arr[i], arr[(i - 1) / 2]);
        i = (i - 1) / 2;
    }
}

void build_heap_up_down(ll n)
{
    for (ll i = 0; i < n; i++)
    {
        heap_insert(i);
    }
}

void build_heap_down_up(ll n)
{
    for (ll i = n - 1; i >= 0; i--)
    {
        heapify(i, n);
    }
}

void heap_sort(ll n)
{
    build_heap_down_up(n);
    // build_heap_up_down(n);
    ll size = n;
    while (size > 1)
    {
```

```
            swap(arr[0], arr[size - 1]);
            size--;
            heapify(0, size);
        }
    }

    signed main()
    {
        ll n;
        cin >> n;
        for (ll i = 0; i < n; i++)
        {
            cin >> arr[i];
        }
        heap_sort(n);
        for (ll i = 0; i < n; i++)
        {
            cout << arr[i] << " ";
        }
        return 0;
    }
```

3. 最多覆盖线段

```
    signed main()
    {
        vector<pair<ll, ll>> lines;
        lines.push_back({1, 3});
        lines.push_back({1, 5});
        lines.push_back({2, 4});
        lines.push_back({3, 6});
        lines.push_back({5, 7});

        sort(lines.begin(), lines.end()); // sort by starting point

        ll max_overlap = 0;
        priority_queue<ll, vector<ll>, greater<ll>> end_points; // min-heap for end points

        for (const auto &line : lines)
        {
            end_points.push(line.second);
            while (!end_points.empty() && end_points.top() <= line.first)
            {
                end_points.pop();
            }
            max_overlap = max(max_overlap, (ll)end_points.size());
        }
        cout << max_overlap << endl;

        return 0;
    }
```

4. 归并排序

```
    ll arr[maxn], help[maxn];

    void merge(ll l, ll m, ll r)
    {
        ll pl = l, pr = m + 1, ph = l;
        while (pl <= m && pr <= r)
        {
            if (arr[pl] < arr[pr])
            {
                help[ph] = arr[pl];
                ph++;
                pl++;
            }
            else
            {
```

```
            help[ph] = arr[pr];
            ph++;
            pr++;
        }
    }
    while (pl <= m)
    {
        help[ph++] = arr[pl++];
    }
    while (pr <= r)
    {
        help[ph++] = arr[pr++];
    }
    for (ll i = l; i <= r; i++)
    {
        arr[i] = help[i];
    }
}

// form top to bottom
// T(n) = 2 * T(n/2) + O(n)
// time O(n * log n)
// space O(n)
void mergesort(ll l, ll r)
{
    if (l == r)
    {
        return;
    }
    ll mid = (l + r) / 2;
    mergesort(l, mid);
    mergesort(mid + 1, r);
    merge(l, mid, r);
}

// form bottom to top
void mergesort(ll n)
{
    ll l, m, r;
    for (ll step = 1; step < n; step <<= 1)
    {
        l = 0;
        while (l < n)
        {
            m = l + step - 1;
            if (m + 1 >= n)
            {
                break;
            }
            r = min(l + (step << 1) - 1, n - 1);
            merge(l, m, r);
            l = r + 1;
        }
    }
}

signed main()
{
    ll n;
    cin >> n;
    for (ll i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    // mergesort(0, n - 1);
    // for (ll i = 0; i < n; i++)
    // {
```

```
    //        cout << arr[i] << " ";
    // }
    // cout << endl;
    mergesort(n);
    for (ll i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

5. k 段有序链表合并

```
struct node
{
    ll val;
    node *next;

    node(ll x) : val(x), next(nullptr) {}
};

struct CompareNodePointers
{
    bool operator()(const node *a, const node *b) const
    {
        return a->val > b->val; // min_heap
    }
};

node *merge_k_sorted_lists(vector<node *> &lists)
{
    priority_queue<node *, vector<node *>, CompareNodePointers> min_heap;

    for (node *list_head : lists)
    {
        if (list_head)
        {
            min_heap.push(list_head);
        }
    }

    if (min_heap.empty())
    {
        return nullptr;
    }

    node *min_node = min_heap.top();
    min_heap.pop();

    node *head = min_node;

    if (min_node->next)
    {
        min_heap.push(min_node->next);
    }

    node *current = head;
    while (!min_heap.empty())
    {
        min_node = min_heap.top();
        min_heap.pop();

        current->next = min_node;
        current = current->next;

        if (min_node->next)
        {
            min_heap.push(min_node->next);
```

```
            }
        }

        return head;
    }

    void delete_list(node *head)
    {
        while (head)
        {
            node *temp = head;
            head = head->next;
            delete temp;
        }
    }

    signed main()
    {
        vector<node *> lists;

        // 1: 1 -> 2 -> null
        node *list_1 = new node(1);
        list_1->next = new node(2);
        lists.push_back(list_1);

        // 2: 4 -> 5 -> null
        node *list_2 = new node(4);
        list_2->next = new node(5);
        lists.push_back(list_2);

        // 3: 3 -> 6 -> null
        node *list_3 = new node(3);
        list_3->next = new node(6);
        lists.push_back(list_3);

        node *merged_head = merge_k_sorted_lists(lists);
        node *current = merged_head;
        while (current)
        {
            cout << current->val << " ";
            current = current->next;
        }
        cout << endl;

        delete_list(merged_head);
        return 0;
    }

    6. 优先队列
    void demonstrate_max_heap()
    {

        priority_queue<ll> max_heap;

        // O(log N)
        max_heap.push(10);
        max_heap.push(30);
        max_heap.push(20);
        max_heap.push(50);
        max_heap.push(5);

        // O(1)
        cout << "top element: " << max_heap.top() << endl;

        // O(1)
        cout << "heap size: " << max_heap.size() << endl;
        cout << "heap empty? " << max_heap.empty() << endl;
```

```
    // O(log N)
    cout << "popping elements: ";
    while (!max_heap.empty())
    {
        cout << max_heap.top() << " ";
        max_heap.pop();
    }
    cout << endl;

    // O(N)
}

void demonstrate_min_heap()
{
    priority_queue<ll, vector<ll>, greater<ll>> min_heap;

    min_heap.push(10);
    min_heap.push(30);
    min_heap.push(20);
    min_heap.push(50);
    min_heap.push(5);

    cout << "top element: " << min_heap.top() << endl;
}

struct person
{
    string name;
    ll score;

    person(string n, ll s) : name(n), score(s) {}

    friend ostream &operator<<(ostream &os, const person &p)
    {
        os << "[" << p.name << ":" << p.score << "]";
        return os;
    }
};

// operator() returns false, p1 before p2
struct ComparePersonMaxHeap
{
    bool operator()(const person &p1, const person &p2) const
    {
        return p1.score < p2.score;
    }
};

struct ComparePersonMinHeap
{
    bool operator()(const person &p1, const person &p2) const
    {
        return p1.score > p2.score;
    }
};

void demonstrate_custom_heap()
{
    priority_queue<person, vector<person>, ComparePersonMaxHeap> custom_max_heap;
    custom_max_heap.push(person("alice", 90));
    custom_max_heap.push(person("bob", 75));
    custom_max_heap.push(person("charlie", 95));
    custom_max_heap.push(person("david", 80));

    cout << "custom Max-Heap (by score): ";
    while (!custom_max_heap.empty())
```

```
    {
        cout << custom_max_heap.top() << " ";
        custom_max_heap.pop();
    }
    cout << endl;
}

void demonstrate_lambda_heap()
{
    auto lambda_cmp_max = [](const person &p1, const person &p2)
    {
        return p1.score < p2.score;
    };

    priority_queue<person, vector<person>, function<bool(const person &, const person &)>>
lambda_max_heap(lambda_cmp_max);

    lambda_max_heap.push(person("alice", 90));
    lambda_max_heap.push(person("bob", 75));
    lambda_max_heap.push(person("charlie", 95));
    lambda_max_heap.push(person("david", 80));

    cout << "lambda Max-Heap (by score): ";
    while (!lambda_max_heap.empty())
    {
        cout << lambda_max_heap.top() << " ";
        lambda_max_heap.pop();
    }
    cout << endl;
}

signed main()
{
    demonstrate_max_heap();
    demonstrate_min_heap();
    demonstrate_custom_heap();
    demonstrate_lambda_heap();

    return 0;
}
```

7. 基数排序

```
const ll maxn = 1e6 + 5;
ll arr[maxn], BASE = 10;

// bits represent the bit-number of the maximum number (BASE-BITS)
void radix_sort(ll n, ll bits)
{
    vector<ll> cnt(BASE, 0);
    for (ll offset = 1; bits > 0; offset *= BASE, bits--)
    {
        memset(cnt.data(), 0, cnt.size() * sizeof(ll));
        for (ll i = 0; i < n; i++)
        {
            ll digit = (arr[i] / offset) % BASE;
            cnt[digit]++;
        }
        for (ll i = 1; i < BASE; i++)
        {
            cnt[i] += cnt[i - 1];
        }

        vector<ll> output(n);
        for (ll i = n - 1; i >= 0; i--)
        {
            ll digit = (arr[i] / offset) % BASE;
            output[--cnt[digit]] = arr[i];
```

```
            }

            for (ll i = 0; i < n; i++)
            {
                arr[i] = output[i];
            }
        }
    }
    signed main()
    {
        ll n, bits;
        cin >> n >> bits;
        for (ll i = 0; i < n; i++)
        {
            // (-3 -6 3 6   9) - (-6)
            //    3   0 9 12 15
            cin >> arr[i];
        }
        radix_sort(n, bits);
        for (ll i = 0; i < n; i++)
        {
            cout << arr[i] << " ";
        }
        return 0;
    }
```

8. 随机快速排序

```
ll arr[maxn];
ll bound_l, bound_r;

ll partition(ll l, ll r, ll x)
{
    ll s = l, p = l;
    ll mid = 0, base = arr[x];
    while (p <= r)
    {
        // 4 3 5 8 7 2 6     base = 4
        //       s       p
        if (arr[p] <= base)
        {
            swap(arr[s], arr[p]);
            if (arr[s] == base)
            {
                mid = s;
            }
            s++;
        }
        p++;
    }
    swap(arr[s - 1], arr[mid]);
    return s - 1;
}

void block_partition(ll l, ll r, ll x)
{
    ll s = l, b = r, p = l;
    ll mid = 0, base = arr[x];
    while (p <= b)
    {
        // 3 5 2 1 5 7 5 8 4
        //    s p       p       b
        if (arr[p] < base)
        {
            swap(arr[s], arr[p]);
            s++, p++;
        }
        else if (arr[p] == base)
```

```
            {
                p++;
            }
            else
            {
                swap(arr[p], arr[b]);
                b--;
            }
        }
        bound_l = s - 1;
        bound_r = b + 1;
}

// classic randomized quicksort
// O(n log n) on average, O(n^2) on worst case
void randomized_quicksort(ll l, ll r)
{
        if (l >= r)
        {
            return;
        }
        // rand from l to r
        ll x = l + rand() % (r - l + 1);
        ll mid = partition(l, r, x);
        randomized_quicksort(l, mid - 1);
        randomized_quicksort(mid + 1, r);
}

// blocked randomized quicksort
// O(n log n) on average, O(n^2) on worst case
void randomized_blocked_quicksort(ll l, ll r)
{
        if (l >= r)
        {
            return;
        }
        ll x = l + rand() % (r - l + 1);
        block_partition(l, r, x);

        ll left = bound_l, right = bound_r;
        randomized_blocked_quicksort(l, left);
        randomized_blocked_quicksort(right, r);
}

signed main()
{
        // randomized_quicksort(0, n - 1);
        randomized_blocked_quicksort(0, n - 1);
        for (ll i = 0; i < n; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
        return 0;
}

9. 第 k 大元素
ll arr[maxn];
ll bound_l, bound_r;

void partition(ll l, ll r, ll x)
{
        ll s = l, b = r, p = l;
        ll mid = 0, base = arr[x];
        while (p <= b)
        {
            // 3 5 2 1 5 7 5 8 4
```

```
            //    s p       p       b
            if (arr[p] < base)
            {
                swap(arr[s], arr[p]);
                s++, p++;
            }
            else if (arr[p] == base)
            {
                p++;
            }
            else
            {
                swap(arr[p], arr[b]);
                b--;
            }
        }
        bound_l = s - 1;
        bound_r = b + 1;
    }

    // select the k-th smallest element of arr
    ll randomized_selection(ll n, ll k)
    {
        ll l = 0, r = n - 1, ans = 0;
        while (l <= r)
        {
            ll x = rand() % (r - l + 1) + l;
            partition(l, r, x);
            if (k <= bound_l)
            {
                r = bound_l;
            }
            else if (k >= bound_r)
            {
                l = bound_r;
            }
            else
            {
                // found the k-th smallest element
                ans = arr[k];
                break;
            }
        }
        return ans;
    }

    signed main()
    {
        ll k = 11; // find the 5-th smallest element
        ll ans = randomized_selection(n, k - 1);
        cout << "the " << k << "-th smallest element is: " << ans << endl;
    }
```

10. 翻转对

```
ll arr[maxn], help[maxn];

// reverse pairs are pairs (i, j) such that i < j and arr[i] > 2 * arr[j]
ll merge_and_count(ll l, ll m, ll r)
{
    ll ans = 0;
    // 2 5 6    1 2 4
    //       i          j
    for (ll j = r, i = m; i >= l; i--)
    {
        while (j >= m + 1 && arr[i] <= 2 * arr[j])
        {
            j--;
```

```
            }
            ans += (j - m);
        }

        // merge
        ll pl = l, pr = m + 1, ph = l;
        while (pl <= m && pr <= r)
        {
            if (arr[pl] <= arr[pr])
            {
                help[ph++] = arr[pl++];
            }
            else
            {
                help[ph++] = arr[pr++];
            }
        }
        while (pl <= m)
        {
            help[ph++] = arr[pl++];
        }
        while (pr <= r)
        {
            help[ph++] = arr[pr++];
        }
        for (ll i = l; i <= r; i++)
        {
            arr[i] = help[i];
        }
        return ans;
}

ll reverse_pairs(ll l, ll r)
{
        if (l == r)
        {
            return 0;
        }
        ll mid = (l + r) / 2;
        ll cnt_l, cnt_r, cross_cnt;
        cnt_l = reverse_pairs(l, mid);
        cnt_r = reverse_pairs(mid + 1, r);
        cross_cnt = merge_and_count(l, mid, r);
        return cnt_l + cnt_r + cross_cnt;
}
```

11. 所有元素在数组中，前缀中比当前元素小的和，的和（分治）
eg: 3 2 4 1 7 = 15

```
ll arr[maxn], help[maxn];

// consider that:
//      1. whether the answer for a large range can be decomposed into the answer for the left part + the answer for the right part + the answer generated across the left and right parts.
//      2. furthermore, if calculating the answer generated across the left and right parts would be more convenient due to the left and right parts being individually sorted.

ll merge_and_sum(ll l, ll m, ll r)
{
        ll ans = 0;
        // 4 6 7 7    5 6 6 8
        // i          j
        for (ll j = m + 1, i = l, sum = 0; j <= r; j++)
        {
            while (i <= m && arr[i] <= arr[j])
            {
                sum += arr[i];
                i++;
```

```
            }
            ans += sum;
        }

        // merge
        ll pl = l, pr = m + 1, ph = l;
        while (pl <= m && pr <= r)
        {
            if (arr[pl] <= arr[pr])
            {
                help[ph++] = arr[pl++];
            }
            else
            {
                help[ph++] = arr[pr++];
            }
        }
        while (pl <= m)
        {
            help[ph++] = arr[pl++];
        }
        while (pr <= r)
        {
            help[ph++] = arr[pr++];
        }
        for (ll i = l; i <= r; i++)
        {
            arr[i] = help[i];
        }
        return ans;
}

ll small_sum(ll l, ll r)
{
    if (l == r)
    {
        return 0;
    }
    ll mid = (l + r) / 2;
    ll suml, sumr, sumc;

    suml = small_sum(l, mid);
    sumr = small_sum(mid + 1, r);
    sumc = merge_and_sum(l, mid, r);

    return suml + sumr + sumc;
}

signed main()
{
    ll n;
    cin >> n;
    for (ll i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    ll result = small_sum(0, n - 1);
    cout << "total small sum: " << result << endl;
    return 0;
}
```

二叉树

1. 平衡树判断

```
bool balance;

ll height(node *cur)
{
    if (!balance || cur == nullptr)
    {
        return 0;
    }

    ll lh = height(cur->left);
    ll rh = height(cur->right);

    if (abs(lh - rh) > 1)
    {
        balance = false;
    }

    return max(lh, rh) + 1;
}

bool balance_tree(node *root)
{
    balance = true;
    height(root);
    return balance;
}
```

2. 二叉树基本操作

```
struct node
{
    ll data;
    node *left;
    node *right;

    node(ll val) : data(val), left(nullptr), right(nullptr) {}

    ~node()
    {
        cout << "deleting node: " << data << endl;
        delete left;
        delete right;
    }
};

// time O(n), space O(h)
// void preorder(node *root)
// {
//     if (root == nullptr)
//         return;

//     cout << root->data << " ";
//     preorder(root->left);
//     preorder(root->right);
// }

// void inorder(node *root)
// {
//     if (root == nullptr)
//         return;

//     inorder(root->left);
//     cout << root->data << " ";
//     inorder(root->right);
// }
```

```
// void postorder(node *root)
// {
//       if (root == nullptr)
//            return;

//       postorder(root->left);
//       postorder(root->right);
//       cout << root->data << " ";
// }

// time O(n), space O(h)
void preorder(node *root)
{
     if (root != nullptr)
     {
          stack<node *> s;
          s.push(root);

          while (!s.empty())
          {
               node *curr = s.top();
               s.pop();
               cout << curr->data << " ";

               if (curr->right != nullptr)
                    s.push(curr->right);
               if (curr->left != nullptr)
                    s.push(curr->left);
          }
     }
}
```

3. 二叉搜索树

```
ll gmin, gmax;
bool bst(node *root)
{
     if (root == nullptr)
     {
          gmin = LLONG_MAX;
          gmax = LLONG_MIN;
          return true;
     }

     bool lok = bst(root->left);
     ll lmin = gmin, lmax = gmax;
     bool rok = bst(root->right);
     ll rmin = gmin, rmax = gmax;

     gmin = min(min(lmin, rmin), root->data);
     gmax = max(max(lmax, rmax), root->data);

     return lok && rok && lmax < root->data && root->data < rmin;
}

node *trim_bst(node *cur, ll low, ll high)
{
     if (cur == nullptr)
     {
          return nullptr;
     }

     if (cur->data < low)
     {
          return trim_bst(cur->right, low, high);
     }
     if (cur->data > high)
     {
```

```
        return trim_bst(cur->left, low, high);
    }

    cur->left = trim_bst(cur->left, low, high);
    cur->right = trim_bst(cur->right, low, high);
    return;
}
```

4. 统计完全二叉树节点数

```
ll most_left(node *cur, ll level)
{
    while (cur != nullptr)
    {
        level++;
        cur = cur->left;
    }
    return level - 1;
}

ll solve(node *cur, ll level, ll h)
{
    if (level == h)
    {
        return 1;
    }

    if (most_left(cur->right, level + 1) == h)
    {
        return (1 << (h - level)) + solve(cur->right, level + 1, h);
    }
    else
    {
        return (1 << (h - level - 1)) + solve(cur->left, level + 1, h);
    }
}

ll count_node(node *root)
{
    if (root == nullptr)
    {
        return 0;
    }

    return solve(root, 1, most_left(root, 1));
}
```

5. 判断完全二叉树

```
vector<node *> node_q(maxn);
ll l, r;

bool complete_tree(node *root)
{
    if (root == nullptr)
    {
        return true;
    }

    l = r = 0;
    node_q[r++] = root;

    bool leaf = false; // complete node
    while (l < r)
    {
        node *cur = node_q[l++];

        if ((cur->left == nullptr && cur->right != nullptr) || (leaf && (cur->left != nullptr || cur->right != nullptr)))
        {
```

```
            return false;
        }

        if (cur->left != nullptr)
        {
            node_q[r++] = cur->left;
        }
        if (cur->right != nullptr)
        {
            node_q[r++] = cur->right;
        }

        if (cur->left == nullptr || cur->right == nullptr)
        {
            leaf = true;
        }
    }
}
```

6. 前序和中序遍历建树

```
const ll maxn = 10001;
vector<ll> pre(maxn), mid(maxn);
map<ll, ll> mp;

// Construct the Binary Tree from Pre-order and In-order Traversals.
node *build_tree(ll pre_l, ll pre_r, ll mid_l, ll mid_r)
{
    if (pre_l > pre_r || mid_l > mid_r)
    {
        return nullptr;
    }

    node *root = new node(pre[pre_l]);
    if (pre_l == pre_r)
    {
        return root;
    }

    ll dex = mp[root->data];
    ll len = dex - mid_l;

    root->left = build_tree(pre_l + 1, pre_l + len, mid_l, dex - 1);
    root->right = build_tree(pre_l + len + 1, pre_r, dex + 1, mid_r);

    return root;
}

void preorder_traversal(node *root)
{
    if (root == nullptr)
    {
        return;
    }
    cout << root->data << " ";
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}

void inorder_traversal(node *root)
{
    if (root == nullptr)
    {
        return;
    }
    inorder_traversal(root->left);
    cout << root->data << " ";
    inorder_traversal(root->right);
```

```
}
```

7. 树的最大和最小深度

```
ll max_depth(node *root)
{
    return root != nullptr ? max(max_depth(root->left), max_depth(root->right)) + 1 : 0;
}

ll min_depth(node *root)
{
    if (root == nullptr)
    {
        return 0;
    }

    if (root->left == nullptr && root->right == nullptr)
    {
        return 1;
    }

    ll min_l = LLONG_MAX, min_r = LLONG_MAX;
    if (root->left != nullptr)
    {
        min_l = min_depth(root->left);
    }
    if (root->right != nullptr)
    {
        min_r = min_depth(root->right);
    }
    return min(min_l, min_r) + 1;
}
```

8. 最近公共祖先

```
node *lca(node *root, node *p, node *q)
{
    if (root == nullptr || root == p || root == q)
    {
        return root;
    }

    node *l = lca(root->left, p, q);
    node *r = lca(root->right, p, q);

    if (l != nullptr && r != nullptr)
    {
        return root;
    }
    if (l == nullptr && r == nullptr)
    {
        return nullptr;
    }

    return l != nullptr ? l : r;
}

node *search_tree_lca(node *root, node *p, node *q)
{
    while (root != p && root != q)
    {
        if (min(p->data, q->data) < root->data && max(p->data, q->data) > root->data)
        {
            break;
        }
        root = root->data < min(p->data, q->data) ? root->right : root->left;
    }
    return root;
}
```

9. 层序遍历序列化与反序列化

```
const ll maxn = 10001;
vector<node *> node_q(maxn);
ll l, r;

string serialize(node *root)
{
    if (root == nullptr)
    {
        return "#,";
    }

    string s = "";
    l = r = 0;

    node_q[r++] = root;
    s += (to_string(root->data) + ",");

    while (l < r)
    {
        ll size = r - l;
        for (ll i = 0; i < size; i++)
        {
            node *cur = node_q[l++];

            if (cur->left != nullptr)
            {
                node_q[r++] = cur->left;
                s += (to_string(cur->left->data) + ",");
            }
            else
            {
                s += "#,";
            }

            if (cur->right != nullptr)
            {
                node_q[r++] = cur->right;
                s += (to_string(cur->right->data) + ",");
            }
            else
            {
                s += "#,";
            }
        }
    }
    return s;
}

node *deserialize(string s)
{
    if (s == "#,")
    {
        return nullptr;
    }

    stringstream ss(s);
    string token;
    vector<string> vals;
    while (getline(ss, token, ','))
    {
        if (!token.empty())
        {
            vals.push_back(token);
        }
    }
```

```
    if (vals.empty())
    {
        return nullptr;
    }

    ll cnt = 0;
    node *root = new node(stoll(vals[cnt++]));

    l = r = 0;
    node_q[r++] = root;

    while (l < r)
    {
        node *cur = node_q[l++];

        if (cnt < vals.size() && vals[cnt] != "#")
        {
            cur->left = new node(stoll(vals[cnt]));
            node_q[r++] = cur->left;
        }
        cnt++;

        if (cnt < vals.size() && vals[cnt] != "#")
        {
            cur->right = new node(stoll(vals[cnt]));
            node_q[r++] = cur->right;
        }
        cnt++;
    }
    return root;
}

10. 先序遍历序列化与反序列化
void solve_serialize(node *root, string &s)
{
    if (root == nullptr)
    {
        s += "#,";
    }
    else
    {
        ll val = root->data;
        s += (to_string(val) + ",");
        solve_serialize(root->left, s);
        solve_serialize(root->right, s);
    }
}

string serialize(node *root)
{
    string s;
    solve_serialize(root, s);
    return s;
}

ll cnt;
node *solve_deserialize(vector<string> &vals)
{
    if (cnt >= vals.size())
    {
        return nullptr;
    }

    string cur = vals[cnt++];
    if (cur == "#")
    {
```

```
        return nullptr;
    }
    else
    {
        node *head = new node(stoll(cur));
        head->left = solve_deserialize(vals);
        head->right = solve_deserialize(vals);
        return head;
    }
}

node *deserialize(string s)
{
    // split the string with ','
    stringstream ss(s);
    string token;
    vector<string> vals;
    while (getline(ss, token, ','))
    {
        if (!token.empty())
        {
            vals.push_back(token);
        }
    }

    cnt = 0;
    return solve_deserialize(vals);
}
```

11. 层序遍历

```
ll l, r;

list<list<ll>> level_order(node *root)
{
    vector<node *> queue(maxn);
    list<list<ll>> ans;

    if (root != nullptr)
    {
        l = r = 0;
        queue[r++] = root;

        while (l < r)
        {
            ll size = r - l;
            list<ll> lt;
            for (ll i = 0; i < size; i++)
            {
                node *cur = queue[l++];
                if (cur->left != nullptr)
                {
                    queue[r++] = cur->left;
                }
                if (cur->right != nullptr)
                {
                    queue[r++] = cur->right;
                }

                lt.push_back(cur->data);
            }
            ans.push_back(lt);
        }
    }

    return ans;
}
```

12. 返回宽度最大的那层的节点数（包括中间的空节点，不包括两侧的）

```
ll l, r;

ll max_special_node(node *root)
{
    vector<node *> queue(maxn);
    vector<ll> dex(maxn);
    ll ans = 0;

    if (root != nullptr)
    {
        l = r = 0;
        queue[r] = root;
        dex[r++] = 1;

        while (l < r)
        {
            ll size = r - l;
            ans = max(ans, dex[r - 1] - dex[l] + 1);
            for (ll i = 0; i < size; i++)
            {
                node *cur = queue[l];
                ll s = dex[l++];

                if (cur->left != nullptr)
                {
                    queue[r] = cur->left;
                    dex[r++] = 2 * s;
                }
                if (cur->right != nullptr)
                {
                    queue[r] = cur->right;
                    dex[r++] = 2 * s + 1;
                }
            }
        }
    }

    return ans;
}
```

13. 路径和为 goal 的所有路径返回

```
ll goal;

void process(node *cur, ll sum, list<ll> &path, list<list<ll>> &ans)
{
    path.push_back(cur->data);
    sum += cur->data;

    if (cur->left == nullptr && cur->right == nullptr)
    {
        if (sum == goal)
        {
            ans.push_back(path);
        }
    }
    else
    {
        if (cur->left != nullptr)
        {
            process(cur->left, sum, path, ans);
        }
        if (cur->right != nullptr)
        {
            process(cur->right, sum, path, ans);
        }
    }
```

```
    path.pop_back();
}

list<list<ll>> path_sum(node *root, ll target)
{
    goal = target;
    list<list<ll>> ans;
    if (root != nullptr)
    {
        list<ll> path;
        process(root, 0, path, ans);
    }
    return ans;
}

void print_paths(const list<list<ll>> &paths)
{
    if (paths.empty())
    {
        cout << "no valid paths found." << endl;
        return;
    }
    for (const auto &path : paths)
    {
        cout << "[ ";
        for (ll val : path)
        {
            cout << val << " ";
        }
        cout << "]" << endl;
    }
}
```

14. 不可相邻节点的最大可能和（打家劫舍）

```
// for the current subtree, the maximum gain from robbing the root node, and the maximum gain from not robbing the root node.
ll yes, no;

void f(node *root)
{
    if (root == nullptr)
    {
        yes = 0;
        no = 0;
    }
    else
    {
        ll y = root->data, n = 0;
        f(root->left);
        y += no;
        n += max(yes, no);

        f(root->right);
        y += no;
        n += max(yes, no);

        yes = y;
        no = n;
    }
}

ll rob(node *root)
{
    f(root);
    return max(yes, no);
}
```

15. z 形层序遍历

```
ll l, r;

list<list<ll>> zigzag_level_order(node *root)
{
    vector<node *> queue(maxn);
    list<list<ll>> ans;

    if (root != nullptr)
    {
        l = r = 0;
        queue[r++] = root;
        // false: left → right; true: right → left
        bool reverse = false;
        while (l < r)
        {
            ll size = r - l;
            list<ll> lt;

            for (ll i = (reverse ? r - 1 : l), j = (reverse ? -1 : 1), k = 0; k < size; i += j, k++)
            {
                lt.push_back(queue[i]->data);
            }

            for (ll i = 0; i < size; i++)
            {

                node *cur = queue[l++];
                if (cur->left != nullptr)
                {
                    queue[r++] = cur->left;
                }
                if (cur->right != nullptr)
                {
                    queue[r++] = cur->right;
                }
            }

            ans.push_back(lt);
            reverse = !reverse;
        }
    }

    return ans;
}
```

字典树

1. 字典树实现

```
// struct trie
// {
//     struct node
//     {
//         ll pass, end;
//         vector<node *> next;

//         node() : pass(0), end(0), next(26, nullptr) {}
//     };

//     node *root;

//     trie() : root(new node()) {}

//     ~trie()
//     {
//         clear(root);
//     }

// private:
//     void clear(node *cur)
//     {
//         if (cur == nullptr)
//             return;
//         for (ll i = 0; i < 26; ++i)
//         {
//             if (cur->next[i] != nullptr)
//             {
//                 clear(cur->next[i]);
//             }
//         }
//         delete cur;
//     }

// public:
//     void insert(string word)
//     {
//         node *cur = root;
//         cur->pass++;
//         for (char ch : word)
//         {
//             ll orient = ch - 'a';
//             if (cur->next[orient] == nullptr)
//             {
//                 cur->next[orient] = new node();
//             }
//             cur = cur->next[orient];
//             cur->pass++;
//         }
//         cur->end++;
//     }

//     ll count(string word)
//     {
//         node *cur = root;
//         for (char ch : word)
//         {
//             ll orient = ch - 'a';
//             if (cur->next[orient] == nullptr)
//             {
//                 return 0;
//             }
//             cur = cur->next[orient];
//         }
//         return cur->end;
```

```
//      }

//      ll pre(string prefix)
//      {
//          node *cur = root;
//          for (char ch : prefix)
//          {
//              ll orient = ch - 'a';
//              if (cur->next[orient] == nullptr)
//              {
//                  return 0;
//              }
//              cur = cur->next[orient];
//          }
//          return cur->pass;
//      }

//      void erase(string word)
//      {
//          if (count(word) > 0)
//          {
//              node *cur = root;
//              cur->pass--;
//              for (char ch : word)
//              {
//                  ll orient = ch - 'a';
//                  cur = cur->next[orient];
//                  cur->pass--;
//              }
//              cur->end--;
//          }
//      }
// };

const ll maxn = 100001;
const ll S = 26;
ll trie[maxn][S], pass[maxn], tail[maxn];
ll cnt;

void build()
{
    cnt = 1;
}

void insert(string word)
{
    ll cur = 1; // root
    pass[cur]++;
    for (char ch : word)
    {
        ll orient = ch - 'a'; // path(ch)
        if (trie[cur][orient] == 0)
        {
            trie[cur][orient] = ++cnt;
        }
        cur = trie[cur][orient];
        pass[cur]++;
    }
    tail[cur]++;
}

ll count(string word)
{
    ll cur = 1;
    for (char ch : word)
    {
        ll orient = ch - 'a';
```

```
            if (trie[cur][orient] == 0)
            {
                return 0;
            }
            cur = trie[cur][orient];
        }
        return tail[cur];
}

ll pre(string prefix)
{
        ll cur = 1;
        for (char ch : prefix)
        {
            ll orient = ch - 'a';
            if (trie[cur][orient] == 0)
            {
                return 0;
            }
            cur = trie[cur][orient];
        }
        return pass[cur];
}

void erase(string word)
{
        if (count(word) != 0)
        {
            ll cur = 1;
            pass[cur]--;
            for (char ch : word)
            {
                ll orient = ch - 'a';
                if (--pass[trie[cur][orient]] == 0)
                {
                    trie[cur][orient] = 0;
                    return;
                }
                cur = trie[cur][orient];
            }
            tail[cur]--;
        }
}

void clear()
{
        for (ll i = 0; i <= cnt; i++)
        {
            memset(trie[i], 0, sizeof(trie[i]));
            tail[i] = 0;
            pass[i] = 0;
        }
}

2. 数组前缀匹配
ll path(char ch)
{
        if (ch >= '0' && ch <= '9')
            return ch - '0';
        if (ch == '-')
            return 10;
        if (ch == '#')
            return 11;
        return -1;
}

// a
```

```
// [3, 7, 8, 0],    →  4, 1, -8    →  "4#1#-8#"
// [2, 4, -1],      →  2, -5       →  "2#-5#"
// [1, 5, 8, 3]     →  4, 3, -5    →  "4#3#-5#"
// b
// [0, 4],          →  4           →  "4#"          →  pre = 2
// [1, 3],          →  2           →  "2#"          →  pre = 1
// [1, 5, 8, 2]     →  4, 3, -6    →  "4#3#-6#"     →  pre = 0
vector<ll> prefix_arr(const vector<vector<ll>> &a, const vector<vector<ll>> &b)
{
    build();
    string s;
    for (const auto &aa : a)
    {
        s = "";
        for (ll i = 1; i < aa.size(); i++)
        {
            s += (to_string(aa[i] - aa[i - 1]) + "#");
        }
        if (!s.empty())
        {
            insert(s);
        }
    }

    vector<ll> ans;
    for (const auto &bb : b)
    {
        s = "";
        for (ll i = 1; i < bb.size(); i++)
        {
            s += (to_string(bb[i] - bb[i - 1]) + "#");
        }
        if (s.empty())
        {
            ans.push_back(0);
        }
        else
        {
            ans.push_back(pre(s));
        }
    }

    clear();
    return ans;
}
```

3. 返回数组中两个数的最大异或值

```
const ll S = 2; // 0, 1
ll trie[maxn][S];
ll cnt, high;

void insert(ll num)
{
    ll cur = 1; // root
    for (ll i = high, path; i >= 0; i--)
    {
        path = (num >> i) & 1;
        if (trie[cur][path] == 0)
        {
            trie[cur][path] = ++cnt;
        }
        cur = trie[cur][path];
    }
}

void build(const vector<ll> &arr)
{
```

```
    cnt = 1;
    ll max = LLONG_MIN;
    for (ll num : arr)
    {
        max = std::max(num, max);
    }
    high = 63 - __builtin_clzll(max); // max != 0

    for (ll num : arr)
    {
        insert(num);
    }
}

void clear()
{
    for (ll i = 0; i <= cnt; i++)
    {
        memset(trie[i], 0, sizeof(trie[i]));
    }
}

ll f(ll num)
{
    ll ans = 0, cur = 1;
    for (ll i = high, path; i >= 0; i--)
    {
        path = !((num >> i) & 1);
        if (trie[cur][path] == 0)
        {
            ans |= (!path << i);
            cur = trie[cur][!path];
        }
        else
        {
            ans |= path << i;
            cur = trie[cur][path];
        }
    }
    return ans;
}

// max(arr[i] ^ arr[j])
ll max_xor(const vector<ll> &arr)
{
    build(arr);
    ll ans = 0;
    for (ll num : arr)
    {
        ans = max(ans, num ^ f(num));
    }
    clear();
    return ans;
}
```

4. 在二维数组中搜索单词

```
ll trie[maxn][S], pass[maxn];
string tail[maxn];
ll cnt;

void build(const vector<string> &words)
{
    cnt = 1;
    for (string word : words)
    {
        ll cur = 1; // root
        pass[cur]++;
```

```
            for (char ch : word)
            {
                ll orient = ch - 'a';
                if (trie[cur][orient] == 0)
                {
                    trie[cur][orient] = ++cnt;
                }
                cur = trie[cur][orient];
                pass[cur]++;
            }
            tail[cur] = word;
        }
    }

    void clear()
    {
        for (ll i = 0; i <= cnt; i++)
        {
            memset(trie[i], 0, sizeof(trie[i]));
            tail[i] = "";
            pass[i] = 0;
        }
    }

    ll dfs(vector<vector<char>> &board, ll i, ll j, ll cur, list<string> &ans)
    {
        if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() || board[i][j] == 0)
        {
            return 0;
        }

        char tmp = board[i][j];
        ll path = tmp - 'a';
        cur = trie[cur][path];
        if (pass[cur] == 0) // no rood (cur = 0); exist road but pass[cur] = 0, means the previous process has already
    collected all the words from this point onwards.
        {
            return 0;
        }

        ll sum = 0;
        if (tail[cur] != "")
        {
            sum++;
            ans.push_back(tail[cur]);
            tail[cur] = "";
        }

        board[i][j] = 0;
        sum += dfs(board, i - 1, j, cur, ans);
        sum += dfs(board, i + 1, j, cur, ans);
        sum += dfs(board, i, j - 1, cur, ans);
        sum += dfs(board, i, j + 1, cur, ans);

        pass[cur] -= sum;
        board[i][j] = tmp;
        return sum;
    }

    // b c t
    // a f k
    // e c c
    // search: "abc", "bce", "fkt"
    list<string> pair_match(vector<vector<char>> &board, const vector<string> &words)
    {
        build(words);
        list<string> ans;
```

```
    for (ll i = 0; i < board.size(); i++)
    {
        for (ll j = 0; j < board[0].size(); j++)
        {
            dfs(board, i, j, 1, ans);
        }
    }
    return ans;
}
```

5. 滑动窗口+子数组两个数的最大异或值
给定数组 a（长度 n），定义数组 b 的“美丽度” f(b) 为所有 i<j 的 b[i] xor b[j] 的最大值。子数组 a[l..r] 被称为“美丽的”当且仅当 f(a[l..r]) >= k。求最短美丽子数组的长度（若不存在，输出 -1）。

```
struct node
{
    int chl[2];
    int last; // maximum id in this subtree
    node()
    {
        chl[0] = chl[1] = -1;
        last = -1;
    }
};
using vn = vector<node>;
ll gs(vn &a)
{
    return a.size();
}

// find the max j, where val ^ a[j] >= bd
int find(vn &trie, int val, int bd)
{
    int res = -1;
    int cur = 0; // root
    bool ok = true;
    for (int pos = high; ok && pos >= 0; --pos)
    {
        int x_bit = (val >> pos) & 1;
        int k_bit = (bd >> pos) & 1;
        auto &chl = trie[cur].chl;
        if (k_bit == 1)
        {
            // must have chl[x_bit ^ 1] branch
            if (chl[x_bit ^ 1] != -1)
            {
                cur = chl[x_bit ^ 1];
            }
            else
            {
                ok = false;
            }
        }
        else
        {
            if (chl[x_bit ^ 1] != -1)
            {
                res = max(res, trie[chl[x_bit ^ 1]].last);
            }
            if (chl[x_bit] != -1)
            {
                cur = chl[x_bit];
            }
            else
            {
                ok = false;
            }
```

```
            }
        }
        if (ok)
        {
            res = max(res, trie[cur].last);
        }
        return res;
}

void add(vn &trie, int val, int id)
{
        int cur = 0;
        trie[cur].last = max(trie[cur].last, id);
        for (int pos = high; pos >= 0; --pos)
        {
            int x_bit = (val >> pos) & 1;
            if (trie[cur].chl[x_bit] == -1)
            {
                trie[cur].chl[x_bit] = gs(trie);
                trie.emplace_back();
            }
            cur = trie[cur].chl[x_bit];
            trie[cur].last = max(trie[cur].last, id);
        }
}

void solve()
{
        int n;
        ll k_ll;
        cin >> n >> k_ll;
        int k = (int)k_ll;
        vl a(n);
        for (int i = 0; i < n; ++i)
            cin >> a[i];

        if (k == 0)
        {
            cout << 1 << '\n';
            return;
        }

        int ans = n + 1;
        vn trie(1);

        for (int i = 0; i < n; ++i)
        {
            // search before add
            int y = find(trie, a[i], k);
            if (y != -1)
            {
                ans = min(ans, i - y + 1);
            }
            add(trie, a[i], i);
        }

        if (ans == n + 1)
            cout << -1 << '\n';
        else
            cout << ans << '\n';
}
```

好题
1. 嵌套类问题（递归求解）
1.1 表达式求值
```
ll where;

void push(stack<ll> &nums, stack<char> &ops, ll cur, char op)
{
    ll n = nums.size();
    if (n == 0 || ops.top() == '+' || ops.top() == '-')
    {
        nums.push(cur);
        ops.push(op);
    }
    else
    {
        ll num = nums.top();
        nums.pop();
        char top = ops.top();
        ops.pop();

        if (top == '*')
        {
            nums.push(num * cur);
        }
        else
        {
            nums.push(num / cur);
        }
        ops.push(op);
    }
}

ll calculate(stack<ll> &nums, stack<char> &ops)
{
    vector<ll> num_vec;
    vector<char> op_vec;
    ops.pop();

    while (!nums.empty())
    {
        num_vec.push_back(nums.top());
        nums.pop();
    }
    while (!ops.empty())
    {
        op_vec.push_back(ops.top());
        ops.pop();
    }
    reverse(num_vec.begin(), num_vec.end());
    reverse(op_vec.begin(), op_vec.end());

    if (num_vec.empty())
    {
        return 0;
    }

    ll ans = num_vec[0];
    for (size_t i = 0; i < op_vec.size(); ++i)
    {
        if (op_vec[i] == '+')
        {
            ans += num_vec[i + 1];
        }
        else if (op_vec[i] == '-')
        {
            ans -= num_vec[i + 1];
        }
```

```
    }
    return ans;
}

ll f(ll i, const string &s)
{
    stack<ll> nums;
    stack<char> ops;
    ll cur = 0;

    while (i < s.length() && s[i] != ')')
    {
        if (s[i] >= '0' && s[i] <= '9')
        {
            cur = cur * 10 + s[i++] - '0';
        }
        else if (s[i] != '(')
        {
            push(nums, ops, cur, s[i++]);
            cur = 0;
        }
        else
        {
            cur = f(i + 1, s);
            i = where + 1;
        }
    }
    push(nums, ops, cur, '+');
    where = i;
    return calculate(nums, ops);
}

ll evaluation(const string &exp)
{
    return f(0, exp);
}
```

1.2 字符串解码

```
ll where;

string get(ll cur, string s)
{
    string ans = "";
    while (cur--)
    {
        ans += s;
    }
    return ans;
}

string f(ll i, const string &s)
{
    string path = "";
    ll cur = 0;

    while (i < s.length() && s[i] != ']')
    {
        if ((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z'))
        {
            path += s[i++];
        }
        else if (s[i] >= '0' && s[i] <= '9')
        {
            cur = cur * 10 + s[i++] - '0';
        }
        else
        {
```

```
            path += get(cur, f(i + 1, s));
            i = where + 1;
            cur = 0;
        }
    }

    where = i;
    return path;
}

string decoder(const string &exp)
{
    return f(0, exp);
}
```

1.3 分子式求原子数

```
ll where;

void fill(map<string, ll> &atom, map<string, ll> pre, string name, ll cnt)
{
    if (name.size() > 0 || !pre.empty())
    {
        cnt = (cnt == 0 ? 1 : cnt);
        if (name.size() > 0)
        {
            atom[name] += cnt;
        }
        else
        {
            for (pair<string, ll> p : pre)
            {
                atom[p.first] += (cnt * p.second);
            }
        }
    }
}

map<string, ll> f(ll i, const string &s)
{
    map<string, ll> atom;

    string name = "";
    ll cnt = 0;
    map<string, ll> pre;

    while (i < s.length() && s[i] != ')')
    {
        if (s[i] == '(' || (s[i] >= 'A' && s[i] <= 'Z'))
        {
            fill(atom, pre, name, cnt);
            name = "";
            cnt = 0;
            pre.clear();

            if (s[i] >= 'A' && s[i] <= 'Z')
            {
                name += s[i++];
            }
            else
            {
                pre = f(i + 1, s);
                i = where + 1;
            }
        }
        else if (s[i] >= 'a' && s[i] <= 'z')
        {
            name += s[i++];
```

```
            }
            else
            {
                cnt = cnt * 10 + s[i++] - '0';
            }
        }
        fill(atom, pre, name, cnt);

        where = i;
        return atom;
}

map<string, ll> atoms(const string &exp)
{
        return f(0, exp);
}
```

1.4 N 皇后问题（位运算版本）

```
ll f_bin(ll limit, ll col, ll left, ll right)
{
        if (col == limit)
        {
                return 1;
        }

        ll ban = col | left | right;
        ll can = ~ban & limit;

        ll place, ans = 0;
        while (can != 0)
        {
                place = can & (-can);
                can ^= place;
                ans += f_bin(limit, col | place, (left | place) >> 1, (right | place) << 1);
        }
}

ll n_queens(ll n)
{
        if (n < 1)
        {
                return 0;
        }

        // return f_arr(0, n);

        // n = 5, limit = 011111
        ll limit = (1 << n) - 1;
        return f_bin(limit, 0, 0, 0);
}
```

2. 打表计数

2.1 完美装包问题

```
ll f(ll n)
{
    if (n < 0)
    {
        return LLONG_MAX;
    }
    if (n == 0)
    {
        return 0;
    }

    ll n_8 = f(n - 8);
    ll n_6 = f(n - 6);

    n_8 += (n_8 != LLONG_MAX ? 1 : 0);
    n_6 += (n_6 != LLONG_MAX ? 1 : 0);

    return min(n_6, n_8);
}

// Liu has bags of capacity 8 and 6, a single apple takes up one unit of space, return the minimum number of bags needed for n apples. But, if any bag cannot be completely filled, return -1.
ll min_bags(ll n)
{
    ll ans = f(n);
    return ans == LLONG_MAX ? -1 : ans;
}
void print_bags()
{
    for (ll i = 0; i < 100; i++)
    {
        cout << i << ": " << min_bags(i) << endl;
    }
}
```

2.2 牛吃草问题（博弈论）

```
string f(ll n, string p)
{
    string enemy = (p == "A" ? "B" : "A");

    if (n < 5)
    {
        return (n == 0 || n == 2) ? enemy : p;
    }

    ll pick = 1;
    while (pick <= n)
    {
        if (f(n - pick, enemy) == p)
        {
            return p;
        }
        pick *= 4;
    }
    return enemy;
}

// A (first) and B (second) take turns eating n portions of grass. In each turn, the player can eat 4^i portions of grass. The player who have no grass to eat loses, return the winner.
string eat_grass(ll n)
{
    return f(n, "A");
}
void print_winner()
{
```

```
    for (ll i = 0; i < 50; i++)
    {
        cout << i << ": " << eat_grass(i) << endl;
    }
}
```

2.3 判断一个数是否能拆分为小于它的连续几个数的和

```
bool is(ll num)
{
    for (ll s = 1, sum; s <= num; s++)
    {
        sum = s;
        for (ll e = s + 1; e <= num; e++)
        {
            if (sum + e > num)
            {
                break;
            }
            if (sum + e == num)
            {
                return true;
            }
            sum += e;
        }
    }
    return false;
}

// Is a number can be partitioned into a sum of several consecutive smaller integers?
bool is_sum(ll n)
{
    return is(n);
}
void print_is()
{
    for (ll i = 0; i < 200; i++)
    {
        cout << i << ": " << is_sum(i) << endl;
    }
}
```

2.4 一个字符串有且只有一个长度大于等于 2 的回文子串称为好串，统计有多少长度为 n 的且仅用 r,e,d 三个字母组成的好串

```
bool is(vector<char> &path, ll l, ll r)
{
    ll m = (l + r) / 2;
    for (; l <= m; l++, r--)
    {
        if (path[l] != path[r])
        {
            return false;
        }
    }

    return true;
}

ll f(vector<char> &path, ll i, ll n)
{
    if (i == n)
    {
        ll cnt = 0;
        for (ll l = 0; l < n; l++)
        {
            for (ll r = l + 1; r < n; r++)
            {
                if (is(path, l, r))
```

```
                {
                    cnt++;
                }
            }
        }
        return cnt == 1 ? 1 : 0;
    }
    else
    {
        ll ans = 0;
        path[i] = 'r';
        ans += f(path, i + 1, n);
        path[i] = 'e';
        ans += f(path, i + 1, n);
        path[i] = 'd';
        ans += f(path, i + 1, n);
        return ans;
    }
}

// Nice string means a string has exactly one palindromic substring of length 2 or greater. Now calculate how
many 'good strings' of length n can be formed using only the letters r, e, and d.
ll is_string(ll n)
{
    vector<char> path(n);
    return f(path, 0, n);
}
void print_string()
{
    for (ll i = 0; i < 15; i++)
    {
        cout << i << ": " << is_string(i) << endl;
    }
}
```

3. 根据数据量猜解法

3.1 超级回文数 (10^18 → 10^9 → 10^5)

```
ll even_enlarge(ll seed)
{
    ll num = seed;
    while (seed)
    {
        num = num * 10 + seed % 10;
        seed /= 10;
    }
    return num;
}

ll odd_enlarge(ll seed)
{
    ll num = seed;
    seed /= 10;
    while (seed)
    {
        num = num * 10 + seed % 10;
        seed /= 10;
    }
    return num;
}

bool palindrome(ll x)
{
    ll offset = 1;
    while (x / offset >= 10)
    {
        offset *= 10;
    }

    // x           52725
    // offset    10000
    while (x != 0)
    {
        if (x / offset != x % 10)
        {
            return false;
        }
        x = (x % offset) / 10;
        offset /= 100;
    }

    return true;
}

bool check(ll x, ll l, ll r)
{
    return x >= l && x <= r && palindrome(x);
}

// Calculate the count of super palindromic numbers between l and r. The super palindromic number is one that is a palindrome and its square root is also a palindrome.
ll super_palindrome(ll l, ll r)
{
    ll limit = (ll)sqrt((double)r);
    ll seed = 1, num = 0, ans = 0;

    do
    {
        // even palindrome
        // 123 → 123321
        num = even_enlarge(seed);
        if (check(num * num, l, r))
        {
```

```
            cout << "x: " << num * num << endl;
            ans++;
        }

        // odd palindrome
        // 123 → 12321
        num = odd_enlarge(seed);
        if (check(num * num, l, r))
        {
            ans++;
            cout << "x: " << num * num << endl;
        }

        seed++;
    } while (num < limit);

    return ans;
}
```

4. 滑动窗口

```
// shortest subarray with a sum greater than or equal to target. (arr[i] > 0)
ll min_len(ll target, ll n)
{
    ll ans = LLONG_MAX;
    for (ll l = 0, r = 0, sum = 0; r < n; r++)
    {
        sum += arr[r];
        while (sum - arr[l] >= target)
        {
            sum -= arr[l++];
        }

        if (sum >= target)
        {
            ans = min(ans, r - l + 1);
        }
    }
    return ans != LLONG_MAX ? ans : 0;
}

// longest substring without repeating characters.
ll lon_cha(string s)
{
    ll ans = 0;
    map<char, ll> mp;

    for (ll l = 0, r = 0; r < s.length(); r++)
    {
        if (mp.count(s[r]))
        {
            l = max(l, mp[s[r]] + 1);
        }
        ans = max(ans, r - l + 1);
        mp[s[r]] = r;
    }

    return ans;
}

// shortest substring that covers all characters of a target string tar (order doesn't matter).
string sho_tar(string str, string tar)
{
    ll debt = tar.length();
    unordered_map<char, ll> mp;
    for (char ch : tar)
    {
        mp[ch]--;
    }

    ll s = 0, len = LLONG_MAX;
    for (ll l = 0, r = 0; r < str.length(); r++)
    {
        mp[str[r]]++;
        if (mp[str[r]] <= 0)
        {
            debt--;
        }

        if (debt == 0)
        {
            while (mp[str[l]] > 0)
            {
                mp[str[l++]]--;
            }

            if (r - l + 1 < len)
```

```
            {
                len = r - l + 1;
                s = l;
            }
        }
    }
    return len != LLONG_MAX ? str.substr(s, len) : "";
}

// circular array of length n, find any starting point such that all prefix sums of the n-1 elements following it are
greater than 0.
ll cir_pre(ll n)
{
    for (ll l = 0, r = 0, sum; l < n; l = r + 1, r = l)
    {
        // a b c d e   a b c d e
        // 0 1 2 3 4   5 6 7 8 9
        // l       r           [l, r)
        sum = 0;
        while (sum + arr[r % n] >= 0)
        {
            if (r - l + 1 == n)
            {
                return l;
            }
            sum += arr[r % n];
            r++;
        }
    }
    return -1;
}

// string length n (n % 4 = 0), composed of 'Q', 'W', 'E', and 'R', it is considered balanced if each character appears
n/4 times.
// the minimum length of a substring that needs to be replaced to make the string balanced.
string bal_str(string str)
{
    ll n = str.length();
    // Q   W   E   R   n = 40
    // 4 12 14 10   →   the shortest substr to cover 2W and 4E

    ll debt = 0;
    unordered_map<char, ll> mp, num;
    for (char ch : str)
    {
        num[ch]++;
    }

    ll bal = n / 4;
    for (pair<char, ll> p : num)
    {
        if (p.second > bal)
        {
            debt += (p.second - bal);
            mp[p.first] = bal - p.second;
        }
    }

    if (!debt)
    {
        return "";
    }

    ll s = 0, len = LLONG_MAX;
    for (ll l = 0, r = 0; r < str.length(); r++)
    {
        mp[str[r]]++;
```

```
            if (mp[str[r]] <= 0)
            {
                debt--;
            }

            if (debt == 0)
            {
                while (mp[str[l]] > 0)
                {
                    mp[str[l++]]--;
                }

                if (r - l + 1 < len)
                {
                    len = r - l + 1;
                    s = l;
                }
            }
        }
        return len != LLONG_MAX ? str.substr(s, len) : "";
}

// the number of subarrays with at most k distinct numbers.
ll f(ll k, ll n)
{
    if (k == 0)
    {
        return 0;
    }

    ll ans = 0;
    unordered_map<ll, ll> mp;
    for (ll l = 0, r = 0, dist = 0; r < n; r++)
    {
        if (++mp[arr[r]] == 1)
        {
            dist++;
        }

        while (dist > k)
        {
            if (--mp[arr[l++]] == 0)
            {
                dist--;
            }
        }

        ans += (r - l + 1);
    }

    return ans;
}
// the number of subarrays where the count of distinct numbers equals k → dis_k = f(k) - f(k-1).
ll dis_k(ll k, ll n)
{
    return f(k, n) - f(k - 1, n);
}

// longest substring where every character appears at least k times (a~z).
ll cha_k(string str, ll k)
{
    unordered_map<char, ll> mp;
    ll ans = 0;
    // under the premise that the substring has exactly require distinct characters, find the longest substring
where every character appears at least k times.
    for (ll requ = 1; requ <= 26; requ++)
    {
```

```
            mp.clear();
            for (ll l = 0, r = 0, dis = 0, sati = 0; r < str.length(); r++)
            {
                if (++mp[str[r]] == 1)
                {
                    dis++;
                }
                if (mp[str[r]] == k)
                {
                    sati++;
                }

                while (dis > requ)
                {
                    if (mp[str[l]] == 1)
                    {
                        dis--;
                    }
                    if (mp[str[l]] == k)
                    {
                        sati--;
                    }

                    mp[str[l++]]--;
                }

                if (sati == requ)
                {
                    ans = max(ans, r - l + 1);
                }
            }
        }
        return ans;
}

// I've found the secret to the sliding window algorithm. It's the relationship between the subarray and monotonicity. As long as there is a monotonic relationship between the length of the subarray and achieving the desired result, you can consider using a sliding window.

// 对顶堆求中位数+滑动窗口
multiset<ll> sml, big;
ll ss = 0, sb = 0;
void add(ll val)
{
    if (!sml.empty() && val <= *sml.rbegin())
    {
        sml.insert(val);
        ss += val;
    }
    else
    {
        big.insert(val);
        sb += val;
    }
}

void remove(ll val){
    auto it_big = big.find(val);
    if (it_big != big.end())
    {
        big.erase(it_big);
        sb -= val;
    }
    else
    {
        auto it_sml = sml.find(val);
        if (it_sml != sml.end())
```

```
        {
            sml.erase(it_sml);
            ss -= val;
        }
    }
}

void rebalance(){
    while (sml.size() > big.size() + 1)
    {
        ll val = *sml.rbegin();
        sml.erase(prev(sml.end()));
        big.insert(val);
        ss -= val;
        sb += val;
    }
    while (sml.size() < big.size())
    {
        ll val = *big.begin();
        big.erase(big.begin());
        sml.insert(val);
        sb -= val;
        ss += val;
    }
}

ll get_cost(){
    if (sml.empty() && big.empty())
        return 0;

    ll median = *sml.rbegin();
    return (median * (ll)sml.size() - ss) + (sb - median * (ll)big.size());
}

void solve(){
    int n;
    ll k;
    cin >> n >> k;
    vl b(n);
    for (int i = 0; i < n; ++i)
    {
        ll val;
        cin >> val;
        // a[l..r], a[i] - a[l] = i - l, then a[i] - i = a[l] - l
        b[i] = val - (i + 1);
    }

    sml.clear();
    big.clear();
    ss = 0, sb = 0;

    int l = 0, ans = 0;
    for (int r = 0; r < n; ++r){
        add(b[r]);
        rebalance();

        while (get_cost() > k)
        {
            remove(b[l]);
            rebalance();
            l++;
        }

        ans = max(ans, r - l + 1);
    }
    cout << ans << "\n";
}
```

5. 双指针

```
// rearrange an array such that odd numbers are placed at odd indices and even numbers are placed at even indices.
void rearrange(vector<ll> &arr)
{
    ll n = arr.size(), even = 0, odd = 1;

    while (even < n && odd < n)
    {
        if (arr[n - 1] % 2 != 0)
        {
            swap(arr[odd], arr[n - 1]);
            odd += 2;
        }
        else
        {
            swap(arr[even], arr[n - 1]);
            even += 2;
        }
    }
}

// arr length n+1, range 1 to n, where only one number is repeated, return the repeated number.
ll rep_num(vector<ll> &arr)
{
    if (arr.size() < 2)
    {
        return -1;
    }
    ll fast = arr[arr[0]], slow = arr[0];

    while (fast != slow)
    {
        fast = arr[arr[fast]];
        slow = arr[slow];
    }

    fast = 0;
    while (fast != slow)
    {
        fast = arr[fast];
        slow = arr[slow];
    }

    return fast;
}

// histogram water collection problem, how much water can be trapped at most?
// ll trap(vector<ll> &height)
// {
//     ll ans = 0, lm, rm, n = height.size();
//     ll left[n], right[n];
//     left[0] = height[0];
//     for (ll i = 1; i < n; i++)
//     {
//         left[i] = max(height[i], left[i - 1]);
//     }
//     right[n - 1] = height[n - 1];
//     for (ll i = n - 2; i >= 0; i--)
//     {
//         right[i] = max(height[i], right[i + 1]);
//     }

//     for (ll i = 1; i < n - 1; i++)
//     {
//         lm = left[i - 1], rm = right[i + 1];
//         ans += max(min(lm, rm) - height[i], (ll)0);
```

```
//      }

//      return ans;
// }
ll trap(vector<ll> &height)
{
    ll ans = 0, n = height.size();
    ll lm = height[0], rm = height[n - 1], l = 1, r = n - 2;

    while (l <= r)
    {
        if (lm >= rm)
        {
            ans += max(rm - height[r], (ll)0);
            rm = max(height[r--], rm);
        }
        else
        {
            ans += max(lm - height[l], (ll)0);
            lm = max(height[l++], lm);
        }
    }

    return ans;
}

// n people crossing a river (all with weights not exceeding limit), each boat can hold at most two people and has
a weight limit, return the minimum number of boats needed?
ll rescue(vector<ll> &peo, ll limit)
{
    ll n = peo.size(), ans = 0;
    ll l = 0, r = n - 1;
    sort(peo.begin(), peo.end());

    while (l <= r)
    {
        peo[r--] + peo[l] > limit ?: l++;
        ans++;
    }

    return ans;
}

// among lots of lines, choose two lines that can trap the most water.
ll area(vector<ll> &high)
{
    ll n = high.size(), ans = 0;
    ll l = 0, r = n - 1;

    while (l < r)
    {
        ans = max(ans, (r - l) * min(high[l], high[r]));
        // Ingeniously Superb
        high[l] < high[r] ? l++ : r--;
    }

    return ans;
}

// the minimum heating radius required to cover all houses.
ll radius(vector<ll> &hou, vector<ll> &heat)
{
    // each house find its nearest heater, then find the maximum of all these distances.
    sort(hou.begin(), hou.end());
    sort(heat.begin(), heat.end());
    ll pu = 0, pt = 0, ans = 0;
    ll n = hou.size(), m = heat.size();
```

```
    while (pu < n)
    {
        while (pt < m - 1 && abs(hou[pu] - heat[pt]) >= abs(hou[pu] - heat[pt + 1]))
        {
            pt++;
        }
        ans = max(ans, (ll)abs(hou[pu++] - heat[pt]));
    }

    return ans;
}

// the first missing positive integer in an unsorted array.
ll mis_pos(vector<ll> &arr)
{
    // right side of r is the garbage area (including r), and the left side of l is the completed area (not including
l).
    ll n = arr.size();
    ll l = 0, r = n;

    while (l < r)
    {
        if (arr[l] == l + 1)
        {
            l++;
            continue;
        }

        if (arr[l] < l + 1 || arr[l] > r || arr[arr[l] - 1] == arr[l])
        {
            swap(arr[l], arr[--r]);
            continue;
        }

        swap(arr[l], arr[arr[l] - 1]);
    }

    return l + 1;
}
```

6. 洪水填充

```
// calculate the number of islands in the grid.
void dfs(vector<vector<char>> &grid, ll i, ll j)
{
    ll n = grid.size(), m = grid[0].size();
    if (i < 0 || i > n - 1 || j < 0 || j > m - 1 || grid[i][j] != '1')
    {
        return;
    }
    grid[i][j] = '#';
    dfs(grid, i - 1, j);
    dfs(grid, i + 1, j);
    dfs(grid, i, j - 1);
    dfs(grid, i, j + 1);
}
ll lands(vector<vector<char>> &grid)
{
    ll ans = 0;
    ll n = grid.size(), m = grid[0].size();
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] == '1')
            {
                ans++;
                dfs(grid, i, j);
            }
        }
    }
    return ans;
}

// x x x x          x x x x
// x o o x    →      x x x x
// o x o x    →      o x x x
// x o x x          x o x x
void dfs(vector<vector<char>> &board, ll i, ll j)
{
    ll n = board.size(), m = board[0].size();
    if (i < 0 || i > n - 1 || j < 0 || j > m - 1 || board[i][j] != 'O')
    {
        return;
    }
    board[i][j] = 'F';
    dfs(board, i - 1, j);
    dfs(board, i + 1, j);
    dfs(board, i, j - 1);
    dfs(board, i, j + 1);
}
void solve(vector<vector<char>> &board)
{
    ll n = board.size(), m = board[0].size();
    // infect from the four borders.
    for (ll i = 0; i < n; i++)
    {
        dfs(board, i, 0);
        dfs(board, i, m - 1);
    }

    for (ll j = 0; j < m; j++)
    {
        dfs(board, 0, j);
        dfs(board, n - 1, j);
    }

    for (int i = 0; i < n; i++)
```

```
    {
        for (int j = 0; j < m; j++)
        {
            board[i][j] = (board[i][j] != 'F' ? 'X' : 'O');
        }
        cout << endl;
    }
}

// 0 1    →    0 1
// 1 0    →    1 1
// change one position to maximize the area of the largest island.
void dfs(vector<vector<ll>> &grid, ll i, ll j, ll color)
{
    ll n = grid.size(), m = grid[0].size();
    if (i < 0 || i > n - 1 || j < 0 || j > m - 1 || grid[i][j] != 1)
    {
        return;
    }
    grid[i][j] = color;
    dfs(grid, i - 1, j, color);
    dfs(grid, i + 1, j, color);
    dfs(grid, i, j - 1, color);
    dfs(grid, i, j + 1, color);
}
ll larg_land(vector<vector<ll>> &grid)
{
    ll ans = 0, color = 1;
    ll n = grid.size(), m = grid[0].size();
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] == 1)
            {
                dfs(grid, i, j, ++color);
            }
        }
    }

    vector<ll> size(++color, 0);
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] > 1)
            {
                ans = max(ans, ++size[grid[i][j]]);
            }
        }
    }

    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] == 0)
            {
                ll area = 1;
                set<ll> neigh;

                if (i > 0 && grid[i - 1][j] > 1)
                    neigh.insert(grid[i - 1][j]);
                if (i < n - 1 && grid[i + 1][j] > 1)
                    neigh.insert(grid[i + 1][j]);
                if (j > 0 && grid[i][j - 1] > 1)
                    neigh.insert(grid[i][j - 1]);
```

```
                if (j < m - 1 && grid[i][j + 1] > 1)
                    neigh.insert(grid[i][j + 1]);

                for (ll x : neigh)
                {
                    area += size[x];
                }

                ans = max(ans, area);
            }
        }
    }

    return ans;
}

// m x n grid of bricks (1) and empty spaces (0), bricks are stable if they are connected to the top row or to other
stable bricks.
// given a list of hits, remove bricks sequentially, after each removal, count how many bricks fall, return an array
of the counts of fallen bricks for each hit.
ll n, m;
ll dfs(vector<vector<ll>> &grid, ll i, ll j){
    if (i < 0 || i > n - 1 || j < 0 || j > m - 1 || grid[i][j] != 1)
    {
        return 0;
    }
    grid[i][j] = 2;
    return 1 + dfs(grid, i - 1, j) + dfs(grid, i + 1, j) + dfs(grid, i, j - 1) + dfs(grid, i, j + 1);
}
bool worth(vector<vector<ll>> &grid, ll i, ll j){
    return (grid[i][j] == 1 && ((i == 0) ||
                                (i > 0 && grid[i - 1][j] == 2) ||
                                (i < n - 1 && grid[i + 1][j] == 2) ||
                                (j > 0 && grid[i][j - 1] == 2) ||
                                (j < m - 1 && grid[i][j + 1] == 2)));
}
vector<ll> hit(vector<vector<ll>> &grid, vector<vector<ll>> &hits){
    n = grid.size(), m = grid[0].size();
    ll cnt = hits.size();
    vector<ll> ans(cnt, 0);

    if (n <= 1){
        return ans;
    }

    for (auto &h : hits)
    {
        grid[h[0]][h[1]]--;
    }

    for (ll j = 0; j < m; j++)
    {
        dfs(grid, 0, j);
    }

    for (ll i = cnt - 1; i >= 0; i--)
    {
        ll row = hits[i][0], col = hits[i][1];
        grid[row][col]++;
        if (worth(grid, row, col))
        {
            ans[i] = dfs(grid, row, col) - 1;
        }
    }

    return ans;
}
```

7. 贡献法

判断是否可以修改 [l, r] 区间，使得 s 中 "01" 子序列的数量等于 "10" 子序列的数量。

```
void solve()
{
    ll n, l, r;
    cin >> n >> l >> r;
    string s;
    cin >> s;
    l--;
    r--;

    // Σ{i<j, s[i]=0, s[j]=1} ==  Σ{i<j, s[i]=1, s[j]=0}
    // ai = 1, if s[i] = '1', ai = 0, if s[i] = '0'
    // Σ{i<j} (1 - ai) * aj == Σ{i<j} ai * (1 - aj)
    // Σ{i<j} (aj - ai)
    vl a(n);
    for (ll i = 0; i < n; ++i)
    {
        a[i] = s[i] - '0';
    }

    // for k, (k - (n - 1 - k)) * ak = (2k - n + 1) * ak, k = 0...n-1
    // Σ(ak * (2k - n + 1)) = 0, k = 0...n-1
    vl d(n);
    for (ll k = 0; k < n; ++k)
    {
        d[k] = 2 * k - n + 1;
    }

    vl pad(n + 1, 0); // pre of a[k] * d[k]
    vl pdn(n + 1, 0); // pre of negative number of d[k]
    vl pdp(n + 1, 0); // pre of positive number of d[k]

    for (ll k = 0; k < n; ++k)
    {
        pad[k + 1] = pad[k] + a[k] * d[k];
        pdn[k + 1] = pdn[k] + (d[k] < 0 ? d[k] : 0);
        pdp[k + 1] = pdp[k] + (d[k] > 0 ? d[k] : 0);
    }

    // FixedSum
    ll fs = pad[l] + pad[n] - pad[r + 1]; // [0...l-1] + [r+1...n-1]
    // Target
    ll tar = -fs;

    // [min_sum, max_sum]
    // -7 -5 -3 -1 1 3 5
    ll mns = pdn[r + 1] - pdn[l];
    ll mxs = pdp[r + 1] - pdp[l];

    bool ok = false;
    if (tar >= mns && tar <= mxs)
    {
        // (Target - min_sum) % 2 == 0
        // get -15, -13, -11, -9, -7, -5, -3, -1, 1, 3, 5, 7, 9
        if ((tar - mns) % 2 == 0)
        {
            ok = true;
        }
    }
}
```

贡献法求任意对差值绝对值

$$\sum_{i<j} |a_i - a_j| = \sum_{i=1}^{k} a_i \times (2i - k - 1)$$

```
ll calc(vl &v)
{
	sort(v.begin(), v.end());
	ll res = 0;
	ll k = v.size();
	for (ll i = 0; i < k; i++)
	{
		res += v[i] * (2 * i - k + 1);
	}
	return res;
}
```

数论

1. 最大公约数与容斥原理

```
ll gcd(ll a, ll b)
{
    return b == 0 ? a : gcd(b, a % b);
}

ll lcm(ll a, ll b)
{
    return (a / gcd(a, b)) * b;
}

ll f(ll x, ll a, ll b)
{
    return x / a + x / b - x / lcm(a, b);
}

// the n-th number divisible by a or b.
ll n_magic_num(ll n, ll a, ll b)
{
    ll ans = 0;
    ll l = 0, r = min(a, b) * n, m;
    while (l <= r)
    {
        m = (l + r) / 2;
        if (f(m, a, b) >= n)
        {
            ans = m;
            r = m - 1;
        }
        else
        {
            l = m + 1;
        }
    }
    return ans % mod;
}
```

2. 同余原理

```
// ((a+b) + (c+d)) % mod = ((a+b) % mod + (c+d) % mod) % mod
// ((a*b) * (c*d)) % mod = ((a*b) % mod * (c*d) % mod) % mod
// (a-b) % mod = (a % mod - b % mod + mod) % mod
```

每次可以选择 S 中的一个元素 x，使其变为 x+k，或者|x-k|，可操作无限次，问最终能否变为 T。

```
void solve()
{
    int n;
    ll k;
    cin >> n >> k;
    map<ll, int> cnt;

    // (x + k) % k === x % k
    // if x >= k, |x - k| = x - k, then (x - k) % k === x % k
    // if x < k, |x - k| = k - x, then (k - x) % k === -x % k
    for (int i = 0; i < n; ++i)
    {
        ll s;
        cin >> s;
        ll r = s % k;
        ll key = min(r, (k - r) % k);
        cnt[key]++;
    }

    for (int i = 0; i < n; ++i)
    {
        ll t;
        cin >> t;
```

```
            ll r = t % k;
            ll key = min(r, (k - r) % k);
            cnt[key]--;
        }

        bool flag = true;
        for (auto const &[key, val] : cnt)
        {
            if (val != 0)
            {
                flag = false;
                break;
            }
        }

        cout << (flag ? "YES" : "NO") << endl;
}
```

3. 博弈问题

3.1 巴什博奕

```
// n stones, two people take turns picking up 1 to m stones, the person who takes the last stone wins.
// n % (m+1) != 0 ? fir : sec
const ll maxn = 1001;
string dp[maxn][maxn];
string bao(ll n, ll m)
{
    if (n < 0)
    {
        return "fir";
    }
    if (n == 0)
    {
        return "sec";
    }
    if (!dp[n][m].empty())
    {
        return dp[n][m];
    }
    string ans = "sec";
    for (ll pick = 1; pick < m + 1; pick++)
    {
        if (bao(n - pick, m) == "sec")
        {
            ans = "fir";
            break;
        }
    }
    dp[n][m] = ans;
    return ans;
}

string bea(ll n, ll m)
{
    return n % (m + 1) != 0 ? "fir" : "sec";
}

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    random_device rd;
    mt19937 gen(rd());

    ll V = 500;
    ll times = 5000;
    cout << "Test Start!" << endl;
    for (ll i = 0; i < times; i++)
```

```
    {
        uniform_int_distribution<> dist(0, V - 1);
        ll n = dist(gen);
        uniform_int_distribution<> m_dist(1, V);
        ll m = m_dist(gen);

        for (ll j = 0; j <= n; ++j)
        {
            for (ll k = 0; k <= m; ++k)
            {
                dp[j][k] = "";
            }
        }

        if (bao(n, m) != bea(n, m))
        {
            cout << "error!" << endl;
            cout << "n = " << n << ", m = " << m << endl;
            cout << "bao(n, m) = " << bao(n, m) << ", bea(n, m) = " << bea(n, m) << endl;
            // break;
        }
    }
    cout << "Test Over!" << endl;
    return 0;
}

// n stones, two players take turns, each turn, player can take p^k stones, p prime and k non-negative, the person
who takes the last stone wins.
// n % 6 != 0 ? fir : sec
string bea(ll n)
{
    return n % 6 != 0 ? "fir" : "sec";
}
```

3.2 尼姆博弈

```
// n piles of stones, each turn, one player chooses a non-empty pile and removes a positive number of stones from
it, the player who takes the last stone wins.
// ^ != 0 ? fir : sec
ll st[1001];
string nim(ll n)
{
    ll ans = 0;
    // a a 0
    // b b 0
    // c c 0
    // 1 0 1   →   subtract positive
    // x x x
    // x x x
    // x x x
    for (ll i = 0; i < n; i++)
    {
        ans ^= st[i];
    }
    return ans != 0 ? "fir" : "sec";
}

// nim game, but the player who takes the last stone loes.
// if all the piles only have one stone, !(n & 1) ? fir : sec;
// else ^ != 0 ? fir : sec
```

3.3 斐波那契博弈

```
// n stones, the first player takes any amount, subsequent players take 1 to 2x stones (x is the previous amount),
find the minimum first move to win.
// if n equals fibo (1 1 2 3 5 8...), take all the stones;
// else n = f_1 + f_2 + ... (f_1, f_2, ... are fiboes), take f_1 stones;
const ll maxn = 1000000000000001L, maxm = 101;
ll f[maxm];
```

```
ll size;
void build()
{
    f[0] = 1;
    f[1] = 2;
    size = 1;
    while (f[size] < maxn)
    {
        f[size + 1] = f[size] + f[size - 1];
        size++;
    }
}
ll bs(ll n)
{
    ll l = 0, r = size;
    ll ans = -1;
    while (l <= r)
    {
        ll m = l + (r - l) / 2;
        if (f[m] <= n)
        {
            ans = f[m];
            l = m + 1;
        }
        else
        {
            r = m - 1;
        }
    }
    return ans;
}
ll fibo(ll n)
{
    ll ans = -1, find;
    while (n != 1 && n != 2)
    {
        find = bs(n);
        if (n == find)
        {
            ans = find;
            break;
        }
        else
        {
            n -= find;
        }
    }
    if (ans != -1)
    {
        return ans;
    }
    else
    {
        return n;
    }
}
```

3.4 威尔佐夫博弈

```
// two piles of stones, two players take turns, two types of moves:
// take any number of stones from one pile.
// take the same number of stones from both piles.
// the player who takes the last stone wins.

// a < b, a != floor((b - a) * phi) ? fir : sec; phi = 1.618
```

3.5 SG 函数与 SG 定理

```
// mex operation:
```

```
// mex(S) refers to the smallest non-negative integer that does not belong to the set S; mex({0, 1, 3}) = 2, mex({1,
2, 4}) = 0.

// terminal state SG(S_0) = 0; for any non-terminal state S, SG(S) =    mex{SG(S') | S' is a successor states of S}.
// SG(S) != 0 ? win : lose;

// complex game composed of multiple independent subgames, its total SG value equals to the xor sum of the SG
values of all the subgames.

string bash(ll n, ll m)
{
    ll sg[n + 1];
    memset(sg, 0, sizeof(sg));
    bool appear[m + 1];

    for (ll i = 1; i < n + 1; i++)
    {
        memset(appear, false, sizeof(appear));
        for (ll j = 1; j < m + 1 && i - j >= 0; j++)
        {
            appear[sg[i - j]] = true;
        }

        ll mex = 0;
        while (m >= mex && appear[mex])
        {
            mex++;
        }
        sg[i] = mex;
    }

    return sg[n] != 0 ? "fir" : "sec";
}

// bash game, but have two piles of stones, a b
string bash(ll a, ll b, ll m)
{
    // 0 1 2 3 4 5 6 7 8 9   m = 3
    // 0 1 2 3 0 1 2 3 0 1
    // sg[i] = i % (m + 1)
    return a % (m + 1) != b % (m + 1) ? "fir" : "sec";
}

ll arr[10001];
string nim(ll n)
{
    if (n == 0)
    {
        return "sec";
    }

    ll max = 0;
    for (ll i = 0; i < n; i++)
    {
        max = std::max(max, arr[i]);
    }

    ll sg[max + 1];
    memset(sg, 0, sizeof(sg));
    bool appear[max + 1];

    for (ll i = 1; i < max + 1; i++)
    {
        memset(appear, false, sizeof(appear));
        for (ll j = 0; j < i; j++)
        {
            appear[j] = true;
```

```
            }

            for (ll s = 0; s < max + 1; s++)
            {
                if (!appear[s])
                {
                    sg[i] = s;
                    break;
                }
            }
        }

        ll eor = 0;
        for (ll i = 0; i < n; i++)
        {
            eor ^= sg[arr[i]];
        }

        return eor != 0 ? "fir" : "sec";
}

// three piles of stones with quantities a, b, and c, each turn, player can choose one pile and remove a number of stones equal to a fiboes, the player who takes the last stone wins.
ll f[] = {1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
ll sg[201];
bool appear[201];
void sg_cnt()
{
        sg[0] = 0;
        for (ll i = 1; i < 201; i++)
        {
            memset(appear, false, sizeof(appear));
            for (ll j = 0; j < 11 && i - f[j] >= 0; j++)
            {
                appear[sg[i - f[j]]] = true;
            }
            for (ll s = 0; s < 201; s++)
            {
                if (!appear[s])
                {
                    sg[i] = s;
                    break;
                }
            }
        }
}
string fiboes(ll a, ll b, ll c)
{
        return (sg[a] ^ sg[b] ^ sg[c]) != 0 ? "fir" : "sec";
}
signed main()
{
        sg_cnt();
        for (ll i = 0; i < 201; i++)
        {
            cout << sg[i] << " ";
        }
        cout << fiboes(1, 2, 3) << endl;
}

// two piles of stones with quantities a and b, move consists of the following steps:
// chooses one pile and removes it, then splits the other pile into two new piles, x and y, where x and y are positive integers.

// both new piles must have a quantity greater than 0, if can't, that player loses.
ll dp[1001][1001];
void build()
```

```
{
    memset(dp, -1, sizeof(dp));
}
ll loz(ll status)
{
    ll cnt = 0;
    while ((status & 1) != 0)
    {
        status >>= 1;
        cnt++;
    }
    return cnt;
}
ll sg(ll a, ll b)
{
    if (a == 1 && b == 1)
    {
        return 0;
    }
    if (dp[a][b] != -1)
    {
        return dp[a][b];
    }
    vector<bool> appear(max(a, b) + 1, false);
    if (a > 1)
    {
        for (ll l = 1, r = a - 1; l < a; l++, r--)
        {
            appear[sg(l, r)] = true;
        }
    }
    if (b > 1)
    {
        for (ll l = 1, r = b - 1; l < b; l++, r--)
        {
            appear[sg(l, r)] = true;
        }
    }
    ll ans = 0;
    for (ll s = 0; s <= max(a, b); s++)
    {
        if (!appear[s])
        {
            ans = s;
            break;
        }
    }
    dp[a][b] = ans;
    return ans;
}
void prove()
{
    cout << "Test Start!" << endl;
    for (ll a = 1; a < 1001; a++)
    {
        for (ll b = 1; b < 1001; b++)
        {
            if (sg(a, b) != loz((a - 1) | (b - 1)))
            {
                cout << "error!" << endl;
                return;
            }
        }
    }
    cout << "Test Over!" << endl;
}
```

4. 质数合集

```
// typedef __int128 ll;

bool is_prime(ll x)
{
    if (x <= 1)
    {
        return false;
    }
    for (ll i = 2; i * i <= x; i++)
    {
        if (x % i == 0)
        {
            return false;
        }
    }
    return true;
}

// 大质数判断
// template <typename ll>
inline ll read()
{
    ll x = 0, f = 1;
    char ch = 0;
    for (; !isdigit(ch); ch = getchar())
        if (ch == '-')
            f = -1;
    for (; isdigit(ch); ch = getchar())
        x = (x << 3) + (x << 1) + (ch - '0');
    return x * f;
}

// template <typename ll>
inline void write(ll x)
{
    if (x < 0)
        putchar('-'), x = -x;
    if (x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}

// template <typename ll>
inline void print(ll x, char ed = '\n')
{
    write(x), putchar(ed);
}

ll t, n;

ll qpow(ll a, ll b, ll mod)
{
    ll ret = 1;
    while (b)
    {
        if (b & 1)
            ret = (ret * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ret % mod;
}

vector<ll> p = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

// s * (logn)^3
```

```
bool miller_rabin(ll n)
{
    if (n < 3 || n % 2 == 0)
        return n == 2;
    ll u = n - 1, t = 0;
    while (u % 2 == 0)
        u /= 2, ++t;
    for (auto a : p)
    {
        if (n == a)
            return 1;
        if (n % a == 0)
            return 0;
        ll v = qpow(a, u, n);
        if (v == 1)
            continue;
        ll s = 1;
        for (; s <= t; ++s)
        {
            if (v == n - 1)
                break;
            v = v * v % n;
        }
        if (s > t)
            return 0;
    }
    return 1;
}

signed main()
{
    t = read(); // t = read<ll>();
    while (t--)
    {
        n = read(); // n = read<ll>();
        if (miller_rabin(n))
            puts("Yes");
        else
            puts("No");
    }
    return 0;
}

// prime factorize
void factor(ll x)
{
    for (ll i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            cout << i << " ";
            while (n % i == 0)
            {
                n /= i;
            }
        }
    }
    if (n > 1)
    {
        cout << n;
    }
}

// arr has distinct positive integers, consider the following graph:
// n nodes, edge exists between arr[i] and arr[j] if and only if arr[i] and arr[j] share a common factor greater than
1, return the largest connected component.
ll factors[100001];
```

```
ll root[20001], size[20001];
ll n;
void init()
{
    for (ll i = 0; i < n; i++)
    {
        root[i] = i, size[i] = 1;
    }
    memset(factors, -1, sizeof(factors));
}
ll find(ll x)
{
    return root[x] != x ? root[x] = find(root[x]) : x;
}
void un(ll x, ll y)
{
    ll fx = find(x), fy = find(y);
    if (fx != fy)
    {
        root[fy] = fx;
        size[fx] += size[fy];
    }
}
ll max_size()
{
    ll ans = 0;
    for (ll i = 0; i < n; i++)
    {
        ans = max(ans, size[i]);
    }
    return ans;
}

ll larg_compon(vector<ll> &arr)
{
    n = arr.size();
    init();
    for (ll i = 0; i < n; i++)
    {
        ll x = arr[i];
        for (ll j = 2; j * j <= x; j++)
        {
            if (x % j == 0)
            {
                if (factors[j] == -1)
                {
                    factors[j] = i;
                }
                else
                {
                    un(i, factors[j]);
                }

                while (x % j == 0)
                {
                    x /= j;
                }
            }
        }
        if (x > 1)
        {
            if (factors[x] == -1)
            {
                factors[x] = i;
            }
            else
            {
```

```
                un(i, factors[x]);
            }
        }
    }
    return max_size();
}

// n * log(logn)
bool prime[10000001];
ll ehrlich(ll n)
{
    memset(prime, false, sizeof(prime)); // all prime
    for (ll i = 2; i * i <= n; i++)
    {
        if (!prime[i])
        {
            for (ll j = i * i; j <= n; j += i)
            {
                prime[j] = true;
            }
        }
    }
    ll cnt = 0;
    for (ll i = 2; i <= n; i++)
    {
        if (!prime[i])
        {
            cnt++;
        }
    }
    return cnt;
}

// 欧拉筛
bool vis[1001];
ll prime[501];
ll euler(ll n)
{
    memset(vis, false, sizeof(vis));
    ll cnt = 0;
    for (ll i = 2; i <= n; i++)
    {
        if (!vis[i])
        {
            prime[cnt++] = i;
        }
        for (ll j = 0; j < cnt; j++)
        {
            if (i * prime[j] > n)
            {
                break;
            }
            vis[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                break;
            }
        }
    }
    return cnt;
}
```

5. 快速幂

```
// safe qpow
ll mul(ll a, ll b, ll mod)
{
    ll ans = 0;
```

```
    a %= mod;
    while (b > 0)
    {
        if (b & 1)
        {
            ans = (ans + a) % mod;
        }
        a = (a + a) % mod;
        b >>= 1;
    }
    return ans;
}
ll qpow(ll x, ll n, ll mod)
{
    ll ans = 1;
    while (n > 0)
    {
        if (n & 1)
        {
            ans = mul(ans, x, mod);
        }
        x = mul(x, x, mod);
        n >>= 1;
    }
    return ans;
}

vvl mmul(vvl &a, vvl &b)
{
    ll n = a.size(), m = b[0].size();
    ll p = a[0].size();
    vvl res(n, vl(m, 0));
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            for (ll k = 0; k < p; k++)
            {
                res[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return res;
}

// logp * n^3
vvl mpow(vvl &a, ll p)
{
    ll n = a.size();
    vvl res(n, vl(n, 0));
    for (ll i = 0; i < n; i++)
    {
        res[i][i] = 1;
    }

    for (; p != 0; p >>= 1)
    {
        if ((p & 1) != 0)
        {
            res = mmul(res, a);
        }
        a = mmul(a, a);
    }
    return res;
}

// logn * k^3
```

```
// matrix qpow solve "1d k-th order", for example, the fibonacci sequence, f(n) = f(n-1) + f(n-2).
ll fibo(ll n)
{
    if (n < 2)
        return n;
    vvl r = {{1, 1}, {1, 0}};
    vvl rn = mpow(r, n - 1);
    vvl f = {{1, 0}};
    vvl fn = mmul(f, rn);
    return fn[0][0];
}
// f(n) = af(n-1) + bf(n-2) + cf(n-3) + ...
//          [a x x]
// r   →    [b x x]
//          [c x x]

// two types of tiles, how many ways to tile a 2 x n board.
// . . x    . o x    x x o
// o o x    . o x    x o o
ll f(ll n, ll r)
{
    if (n == 0)
        return r != 0 ? 0 : 1;
    if (n == 1)
        return 1;

    // n > 1
    if (r == 1)
        return f(n - 1, 1) + f(n - 1, 0);
    else
        return f(n - 1, 0) + f(n - 2, 0) + 2 * f(n - 2, 1);
}
// f(n) = 2f(n-1) + f(n-3)
ll tilings(ll n)
{
    if (n < 2)
        return n;
    vvl r = {{2, 1, 0}, {0, 0, 1}, {1, 0, 0}};
    vvl rn = mpow(r, n - 1);
    vvl f = {{5, 2, 1}};
    vvl fn = mmul(f, rn);
    return fn[0][0];
}

// logn * k^3
// matrix qpow solve "kd 1-th order", for example, dp[i][3] = dp[i-1][2] + dp[i-1][3] - dp[i-1][4].

// count the number of strings of length n that can be formed according to the following rules:
// only a e i o u
// u → a → e → a/i
// i !→ i, o → i/u

// dp[i][j] valid strings of length i that end with character j.
// dp[i][a] = dp[i-1][e/i/u]
// dp[i][e] = dp[i-1][a/i]
// dp[i][i] = dp[i-1][e/o]
// dp[i][o] = dp[i-1][i]
// dp[i][u] = dp[i-1][i/o]
ll vowel(ll n)
{
    if (n < 2)
        return 5;
    vvl r = {{0, 1, 0, 0, 0},
             {1, 0, 1, 0, 0},
             {1, 1, 0, 1, 1},
             {0, 0, 1, 0, 1},
             {1, 0, 0, 0, 0}};
```

```
    vvl rn = mpow(r, n - 1);
    vvl f = {{1, 1, 1, 1, 1}};
    vvl fn = mmul(f, rn);

    ll ans = 0;
    for (ll x : fn[0])
    {
        ans = (x + ans) % mod;
    }
    return ans;
}

// string length n, A: absent, L: late, P: present, consider it valid if the total count of 'A's is less than 2, and there is no consecutive sequence of 'L's of length 3 or more, how many valid strings.

// dp[i][a][l] the first i days, exists a absences and exactly l consecutive days of being late at the end.
// dp[i][0][0] → dp[i][0]
// dp[i][0][1] → dp[i][1]
// dp[i][0][2] → dp[i][2]
// dp[i][1][0] → dp[i][3]
// dp[i][1][1] → dp[i][4]
// dp[i][1][2] → dp[i][5]
// dp[i][a][l] → dp[i][3a+l]

// dp[i][0] = dp[i-1][0/1/2]
// dp[i][1] = dp[i-1][0]
// dp[i][2] = dp[i-1][1]
// dp[i][3] = dp[i-1][0/1/2/3/4/5]
// dp[i][4] = dp[i-1][3]
// dp[i][5] = dp[i-1][4]
ll record(ll n)
{
    if (n < 2)
        return 3;
    vvl r = {{1, 1, 0, 1, 0, 0},
             {1, 0, 1, 1, 0, 0},
             {1, 0, 0, 1, 0, 0},
             {0, 0, 0, 1, 1, 0},
             {0, 0, 0, 1, 0, 1},
             {0, 0, 0, 1, 0, 0}};
    vvl rn = mpow(r, n - 1);
    vvl f = {{1, 1, 0, 1, 0, 0}};
    vvl fn = mmul(f, rn);

    ll ans = 0;
    for (ll x : fn[0])
    {
        ans = (x + ans) % mod;
    }
    return ans;
}
```

6. 逆元

```
ll inv(ll x)
{
    return qpow(x, mod - 2, mod);
}
ll div(ll a, ll b, ll mod)
{
    return ((a % mod) * inv(b)) % mod;
}

ll invs[10001];
void lin_inv(ll n, ll p)
{
    invs[1] = 1;
    for (ll i = 2; i < n; i++)
```

```
    {
        invs[i] = (p - (invs[p % i] * (p / i)) % p) % p;
    }
}

ll fac[10001], finvs[10001];
void lin_finvs(ll n, ll p)
{
    fac[0] = 1;
    fac[1] = 1;
    for (ll i = 2; i < n + 1; i++)
    {
        fac[i] = (i * fac[i - 1]) % p;
    }

    // for (ll i = 1; i < n; i++)
    // {
    //      finvs[i] = inv(fac[i], p) % p;
    // }
    finvs[n] = inv(fac[n], p);
    for (ll i = n - 1; i >= 0; i--)
    {
        finvs[i] = (finvs[i + 1] * (i + 1)) % p;
    }
}

ll comb(ll n, ll m, ll p)
{
    if (m < 0 || m > n)
        return 0;

    lin_finvs(n, p);

    ll ans = fac[n];
    ans = (ans * finvs[m]) % p;
    ans = (ans * finvs[n - m]) % p;
    return ans;
}
```

7. 容斥原理

```
// X1 ∪ X2 ∪ ... ∪ Xk
// C(k, 1) - C(k, 2) + ... - C(k, k-1) + C(k, k)

// array, return the number of subsequences whose gcd equals 1.
vl cnt(maxn, 0), tp(maxn, 0), dp(maxn);
void build()
{
    tp[0] = 1;
    for (ll i = 1; i < maxn; i++)
    {
        tp[i] = (2 * tp[i - 1]) % mod;
    }
}
ll sub_gcd()
{
    build();
    for (ll i = maxn - 1; i > 0; i--)
    {
        ll ct = 0;
        for (ll j = i; j < maxn; j += i)
        {
            ct += cnt[j];
        }
        dp[i] = (tp[ct] - 1 + mod) % mod;

        for (ll j = 2 * i; j < maxn; j += i)
        {
```

```
                dp[i] = (dp[i] - dp[j] + mod) % mod;
            }
        }
        return dp[1];
}

// return the number of subsequences whose gcd equals k: keeping only k and its multiples, the problem is transformed into the base problem of gcd equals 1.

// 4 types of coins with fixed denominations, for each query, give the quantity of each coin type and a target amount, return the methods.
vl dp(maxn, 0), val(4), cnt(4);
void f(){
    // make a target amount with an unlimited supply of coins.
    dp[0] = 1;
    for (ll i = 0; i < 4; i++)
    {
        for (ll j = val[i]; j < maxn; j++)
        {
            dp[j] += dp[j - val[i]];
        }
    }
}
ll coins(ll tar){
    ll ill = 0;
    // 0001 -> 1111, using the exceeding state for val[0], val[1], val[2], and val[3] yuan.
    for (ll s = 1; s < 16; s++)
    {
        ll t = tar;
        ll sign = -1;
        for (ll bit = 0; bit < 4; bit++)
        {
            if ((s >> bit) & 1)
            {
                t -= val[bit] * (cnt[bit] + 1);
                sign = -sign;
            }
        }
        if (t >= 0)
        {
            ill += sign * dp[t];
        }
    }
    return dp[tar] - ill;
}

// you have n different songs, and your wife wanna to listen to l songs, songs can be repeated, but each song must be played at least once, a song can only be played again after k songs, return the play methods.

// without the first limitation
// A(n, k + 1) * (n - k)^(l - k - 1)
// = A(n, k) * (n - k)^(l - k)
ll playlists(ll n, ll l, ll k){
    // i missing songs.
    // (-1)^i * C(n, i) * f(n-i, l, k), i = 0, 1, 2, 3, ..., n-k-1
    // (-1)^i * n! * (n-i-k)^(l-k) / (i! * (n-i-k)!)
    lin_finvs(100, mod);
    ll tmp, ans = 0, sign = 1;
    for (ll i = 0; i < n - k; i++, sign = (sign != 1 ? 1 : (mod - 1))){
        tmp = (sign * qpow(n - i - k, l - k, mod)) % mod;
        tmp = (tmp * fac[n]) % mod;
        tmp = (tmp * finvs[i]) % mod;
        tmp = (tmp * finvs[n - i - k]) % mod;
        ans = (ans + tmp) % mod;
    }

    return ans;
```

```
}

// sum(arr[l...r]) = s, max(arr[l..r]) = x
vl a(maxn);
// count subsegments with sum equals s
ll seg(vl &arr, ll s)
{
    if (arr.empty())
    {
        return 0;
    }
    map<ll, ll> pcnt;
    pcnt[0] = 1;

    ll sum = 0, cnt = 0;
    for (ll val : arr)
    {
        sum += val;
        if (pcnt.count(sum - s))
        {
            cnt += pcnt[sum - s];
        }
        pcnt[sum]++;
    }
    return cnt;
}

ll f(vl &arr, ll s, ll x)
{
    // (seg with sum euqals s) - (seg with sum euqals s and without x)
    ll cnt_all = seg(arr, s);

    ll cnt_nox = 0;
    vl mb;
    for (ll val : arr)
    {
        if (val == x)
        {
            if (!mb.empty())
            {
                cnt_nox += seg(mb, s);
            }
            mb.clear();
        }
        else
        {
            mb.push_back(val);
        }
    }
    if (!mb.empty())
    {
        cnt_nox += seg(mb, s);
    }
    return cnt_all - cnt_nox;
}
void solve()
{
    ll n, s, x;
    cin >> n >> s >> x;
    for (ll i = 0; i < n; ++i)
        cin >> a[i];

    ll ans = 0;
    vl cb;

    for (ll i = 0; i < n; ++i)
    {
```

```
            if (a[i] > x)
            {
                if (!cb.empty())
                {
                    ans += f(cb, s, x);
                    cb.clear();
                }
            }
            else
            {
                cb.push_back(a[i]);
            }
        }

        // the last block
        if (!cb.empty())
        {
            ans += f(cb, s, x);
        }

        cout << ans << endl;
}
```

有 t 次询问，每次询问给出 l，r，k，问 l 到 r 范围内有多少个子段使得不同元素的个数刚好为 k（刚好为 k 的题目利用容斥原理通常可以转化为小于等于 k）

```
vl a(maxn);
// <= k, <= len
ll most(ll k, ll n, ll len)
{
    if (len == 0)
        return 0;

    ll ans = 0;
    map<ll, ll> freq;
    ll l = 0;
    for (ll r = 0, cnt = 0; r < n; ++r)
    {
        if (++freq[a[r]] == 1)
        {
            cnt++;
        }

        while (cnt > k)
        {
            freq[a[l]]--;
            if (freq[a[l]] == 0)
            {
                freq.erase(a[l]);
                cnt--;
            }
            l++;
        }

        // subarr end with r
        ans += min(r - l + 1, len);
    }
    return ans;
}
ll solve(ll n)
{
    ll k, l, r;
    cin >> k >> l >> r;
    for (ll i = 0; i < n; i++)
        cin >> a[i];
    return (most(k, n, r) - most(k - 1, n, r)) - (most(k, n, l - 1) - most(k - 1, n, l - 1));
}
```

8. 排列组合定理

8.1 排列的奇偶性

一个排列的奇偶性由其逆序数的奇偶性决定。逆序数为奇数，则为奇排列；逆序数为偶数，则为偶排列。任意一次交换，都会改变排列的奇偶性；一个排列无法通过偶数次交换变成另一个奇偶性不同的排列。

8.2 轮换（环）分解

任何一个排列都可以唯一地分解成若干个不相交的轮换的乘积。
一个长度为 n、包含 c 个轮换的排列，可以由 n-c 次交换（不一定是相邻交换）从初始排列 (1, 2, ..., n) 得到。
每一个轮换 (c1, c2, ..., ck) 可以由 k-1 次交换得到。例如 (1,2,3) 可以通过先换 (1,2) 再换 (1,3) 得到。所有轮换需要的总交换次数就是 Σ(ki - 1) = (Σki) - c = n - c。

8.3 逆序数与轮换数奇偶性

一个排列的逆序数 I(P) 的奇偶性，与 n - c 的奇偶性相同。

8.4 错位排列

一个长度为 n 的排列，满足所有元素 p[i] != i，被称为错位排列。
其数量 D(n) 满足递推式：D(n) = (n-1) * (D(n-1) + D(n-2))，且 D(1)=0, D(2)=1。

8.5 第 k 个排列（康托展开）

可以快速地计算出任意一个排列在所有 n! 个排列中的字典序排名，也可以根据排名反推出对应的排列。
基于“康托展开”的阶乘进制系统，X = a[n-1]*(n-1)! + a[n-2]*(n-2)! + ... + a[0]*0!，其中 a[i] 是当前位右边比它小的数的个数。

8.6 排列复合的奇偶性

两个排列 A 和 B 复合后的新排列 P = A * B，其逆序数的奇偶性等于 A 和 B 的逆序数奇偶性之和。
I(A*B) % 2 == (I(A) + I(B)) % 2，允许我们将一个复杂的相关状态（A 和 B 之间的不和谐对）分解为两个独立状态（A 的逆序数，B 的逆序数）的简单组合。

8.7 排列与子序列 (LIS / LDS)

对于任意一个 1 到 n 的排列，其最长上升子序列 (LIS) 的长度，与最长下降子序列 (LDS) 的长度之间存在深刻联系。一个著名的结果是，将排列分割成的最少下降子序列数量等于其 LIS 的长度。

8.8 将 n 分解为两个互质数的乘积，例如 12=1*12 或者 3*4，注意不可以是 2*6，共有多少种分法，结果就等于 2^(k-1)种，其中 k 为 n 的质因数个数。

9. 矩阵构造性质

9.1 邻接矩阵的幂

对于一个图（有向或无向）的邻接矩阵 A，其 k 次幂 A^k 中的元素 A^k[i][j]，表示从顶点 i 到顶点 j 恰好经过 k 条边的路径数量。如果 A 是 0/1 矩阵，这个和式计算的就是：是否存在一个中间点 p，使得 i->p 和 p->j 的路径都存在，这正是长度为 2 的路径。
这个思想可以推广到任意长度 k。当题目要求“求 k 步之内从 i 到 j 的方案数”时，可以计算 S=I+A+A^2+...+A^k。如果 k 很大，可以用二分法求等比矩阵和，或者在状态矩阵中增加一个“吸收点”来一次性计算。

10. 高斯消元

10.1 高斯消元解加法方程组

```
const ll mod = 1e9 + 7;
const double sml = 1e-7;

vvd mat(101, vd(101));
// n^3
void gauss(ll n)
{
    // n x n+1
    for (ll i = 1; i < n + 1; i++)
    {
        ll max = i;
        for (ll j = 1; j < n + 1; j++)
        {
            if (j < i && abs(mat[j][j]) >= sml)
            {
                continue;
            }
            if (abs(mat[j][i]) > abs(mat[max][i]))
            {
                max = j;
            }
        }
```

```
            swap(mat[i], mat[max]);

            if (abs(mat[i][i]) >= sml)
            {
                double t = mat[i][i];
                for (ll j = i; j < n + 2; j++)
                {
                    mat[i][j] /= t;
                }
                for (ll j = 1; j < n + 1; j++)
                {
                    if (i != j)
                    {
                        double rate = mat[j][i] / mat[i][i];
                        for (ll k = i; k < n + 2; k++)
                        {
                            mat[j][k] -= mat[i][k] * rate;
                        }
                    }
                }
            }
        }
}
// absolutely no dependency within the pivot variables or the free variables; free variables do not depend on pivot
variables, but the values of the pivot variables may depend on the values of some free variables.

void print(ll n)
{
    ll sign = 1; // 1: unique solution, 0: many solutions, -1: no solution

    for (ll i = 1; i < n + 1; i++)
    {
        if (abs(mat[i][i]) < sml && abs(mat[i][n + 1]) >= sml)
        {
            sign = -1;
            break;
        }
        if (abs(mat[i][i]) < sml)
        {
            sign = 0;
        }
    }

    if (sign == 1)
    {
        for (ll i = 1; i < n + 1; i++)
        {
            cout << "x_" << i << "="
                 << fixed << setprecision(2) << mat[i][n + 1] << "\n";
        }
    }
    else
    {
        cout << sign << "\n";
    }
}

// signed main()
// {
//     ll n;
//     cin >> n;
//     for (ll i = 1; i < n + 1; i++)
//     {
//         for (ll j = 1; j < n + 2; j++)
//         {
//             cin >> mat[i][j];
//         }
```

```
//      }
//      gauss(n);
//      print(n);
// }

// n+1 points in n-dimensional space, each with n coordinates, assuming these points all lie on the surface of a sphere, find the coordinates of the center of the sphere.
vvd data(101, vd(101));
void solve(ll n)
{
    for (ll i = 1; i < n + 2; i++)
    {
        for (ll j = 1; j < n + 1; j++)
        {
            cin >> data[i][j];
        }
    }

    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < n + 1; j++)
        {
            mat[i][j] = 2 * (data[i][j] - data[i + 1][j]);
            mat[i][n + 1] += data[i][j] * data[i][j] - data[i + 1][j] * data[i + 1][j];
        }
    }
    gauss(n);
    for (ll i = 1; i < n + 1; i++)
    {
        cout << fixed << setprecision(3) << mat[i][n + 1] << "\n";
    }
}

// n^4
// n items with positive integer weights, where there is exactly one heaviest item, have n+1 weighing data points, of which exactly one is incorrect, find the number of the heaviest item, if the data is invalid, return "illegal".

10.2 高斯消元解异或方程组
const ll maxv = 2001;

// vvl mat(101, vl(101));
void gauss(ll n)
{
    // n x n+1
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < n + 1; j++)
        {
            if (j < i && mat[j][j] == 1)
            {
                continue;
            }
            if (mat[j][i] == 1)
            {
                swap(mat[i], mat[j]);
                break;
            }
        }

        if (mat[i][i] == 1)
        {
            for (ll j = 1; j < n + 1; j++)
            {
                if (i != j && mat[j][i] == 1)
                {
                    for (ll k = i; k < n + 2; k++)
                    {
```

```
                            mat[j][k] ^= mat[i][k];
                        }
                    }
                }
            }
        }
    }
}

// the prime factors of each number do not exceed 2000, find the number of non-empty subsets of arr whose
elements' product is a perfect square.
vvl mat(maxv, vl(maxv));
vl arr(305), prime(305), pt(305);
vb vis(maxv, false);
ll cnt;
ll build()
{
    for (ll i = 2; i * i < maxv; i++)
    {
        if (!vis[i])
        {
            for (ll j = i * i; j < maxv; j += i)
            {
                vis[j] = true;
            }
        }
    }
    cnt = 0;
    for (ll i = 2; i < maxv; i++)
    {
        if (!vis[i])
        {
            prime[++cnt] = i; // 303
        }
    }

    pt[0] = 1; // 2^i % mod
    for (ll i = 1; i < 305; i++)
    {
        pt[i] = (2 * pt[i - 1]) % mod;
    }
}
ll square(ll n)
{
    // arr[] = {12, 8, 6, 60, 21}
    //       2 3 5 7
    // 12: 0 1 0 0
    //  8: 1 0 0 0
    //  6: 1 1 0 0
    // 60: 0 1 1 0
    // 21: 0 1 0 1
    build();
    for (auto &row : mat)
    {
        fill(row.begin(), row.end(), 0);
    }
    ll cur;
    for (ll i = 1; i < n + 1; i++)
    {
        cur = arr[i];
        for (ll j = 1; j < cnt + 1 && cur != 0; j++)
        {
            while (cur % prime[j] == 0)
            {
                mat[j][i] ^= 1;
                cur /= prime[j];
            }
        }
```

```
    }

    gauss(cnt);
    ll pivot = 0;
    for (ll i = 1; i < cnt + 1; i++)
    {
        if (mat[i][i] == 1)
        {
            pivot++;
        }
    }

    return pt[n - pivot] - 1;
}
signed main()
{
    ll t;
    cin >> t;
    while (t--)
    {
        ll n;
        cin >> n;
        for (ll i = 1; i < n + 1; i++)
        {
            cin >> arr[i];
        }
        cout << square(n) << endl;
    }
}

// n nodes and m undirected edges, the state of every node is 0, perform an operation on any node to flip its state
and the state of all its adjacent nodes, the minimum step to make all nodes become 1.
vl op(maxv);
ll ans, n;
void dfs(ll i, ll num)
{
    if (num > ans - 1)
    {
        return;
    }
    // 1 ~ n, points have been determined.
    if (i == 0)
    {
        ans = num;
    }
    else
    {
        if (mat[i][i] == 0)
        {
            op[i] = 0;
            dfs(i - 1, num);
            op[i] = 1;
            dfs(i - 1, num + 1);
        }
        else
        {
            ll cur = mat[i][n + 1];
            for (ll j = i + 1; j < n + 1; j++)
            {
                if (mat[i][j] == 1)
                {
                    cur ^= op[j];
                }
            }
            dfs(i - 1, num + cur);
        }
    }
```

```
}
void build(ll n)
{
	for (ll i = 1; i < n + 1; i++)
	{
		for (ll j = 1; j < n + 1; j++)
		{
			mat[i][j] = 0;
		}
		mat[i][i] = 1;
		mat[i][n + 1] = 1;
		op[i] = 0;
	}
}

void flip(ll n, ll m)
{
	build(n);
	for (ll i = 0; i < m; i++)
	{
		ll u, v;
		cin >> u >> v;
		mat[u][v] = 1;
		mat[v][u] = 1;
	}
	gauss(n);

	ll sign = 1;
	for (ll i = 1; i < n + 1; i++)
	{
		if (mat[i][i] == 0)
		{
			sign = 0;
			break;
		}
	}
	ans = 0;
	if (sign)
	{
		for (ll i = 1; i < n + 1; i++)
		{
			ans += mat[i][n + 1];
		}
	}
	else
	{
		ans = n;
		dfs(n, 0);
	}
	cout << ans << endl;
}

// given n insects and m consistent measurement records, each record consists of a set of insects and the parity of their total number of legs, find the minimum number of records k required to uniquely determine the parity of each insect's legs.
const ll bit = 64;
const ll maxn = maxv / bit + 1;
vvl mat(maxn, vl(maxn));
ll n, m, s, k;
void eor(ll ro, ll rt, ll bits)
{
	for (ll k = 0; k <= bits / bit; k++)
	{
		mat[rt][k] ^= mat[ro][k];
	}
}
void gauss(ll n)
```

```
{
    k = 0;
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = i; j < n + 1; j++)
        {
            if (get(j, i) == 1)
            {
                swap(mat[i], mat[j]);
                k = max(k, j);
                break;
            }
        }
        if (get(i, i) == 0)
        {
            return;
        }
        for (ll j = 1; j < n + 1; j++)
        {
            if (i != j && get(j, i) == 1)
            {
                eor(i, j, n + 2);
            }
        }
    }
}
void setb(ll row, ll col, ll v)
{
    if (v == 0)
    {
        mat[row][col / bit] &= ~(1LL << (col % bit));
    }
    else
    {
        mat[row][col / bit] |= 1LL << (col % bit);
    }
}
ll get(ll row, ll col)
{
    return ((mat[row][col / bit] >> (col % bit)) & 1) == 1 ? 1 : 0;
}
ll legs()
{
    cin >> n >> m;
    s = max(n, m);

    for (ll i = 1; i < m + 1; i++)
    {
        ll lg;
        string sta;
        cin >> sta >> lg;
        for (ll j = 1; j < n + 1; j++)
        {
            setb(i, j, sta[j - 1] - '0');
        }
        setb(i, s + 1, lg);
    }
    gauss(s);

    ll sign = 1;
    for (ll i = 1; i < n + 1; i++)
    {
        if (get(i, i) == 0)
        {
            sign = 0;
            break;
        }
```

```
    }
    if (sign == 0)
    {
        cout << "Cannot Determine" << endl;
    }
    else
    {
        cout << k << endl;
        for (ll i = 1; i < n + 1; i++)
        {
            if (get(i, s + 1) == 1)
            {
                cout << "Moon" << endl;
            }
            else
            {
                cout << "Earth" << endl;
            }
        }
    }
}
```

10.3 高斯消元解同余方程组

```
vl invs(mod + 1);
void inv()
{
    invs[1] = 1;
    for (int i = 2; i < mod; i++)
    {
        invs[i] = mod - invs[mod % i] * (mod / i) % mod;
    }
}
vvl mat(1001, vl(1001));
void gauss(ll n)
{
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < n + 1; j++)
        {
            if (j < i && mat[j][j] != 0)
            {
                continue;
            }
            if (mat[j][i] != 0)
            {
                swap(mat[i], mat[j]);
                break;
            }
        }
        if (mat[i][i] != 0)
        {
            for (ll j = 1; j < n + 1; j++)
            {
                if (i != j && mat[j][i] != 0)
                {
                    ll g = gcd(mat[j][i], mat[i][i]);
                    ll a = mat[i][i] / g;
                    ll b = mat[j][i] / g;
                    if (j < i && mat[j][j] != 0)
                    {
                        for (ll k = j; k < i; k++)
                        {
                            mat[j][k] = (mat[j][k] * a) % mod;
                        }
                        // mat[j][j] = (mat[j][j] * a) % mod; assume all the free variables are 0
                    }
                    for (ll k = i; k < n + 2; k++)
```

```
                    {
                        mat[j][k] = ((mat[j][k] * a - mat[i][k] * b) % mod + mod) % mod;
                    }
                }
            }
        }
    }
    for (ll i = 1; i < n + 1; i++)
    {
        if (mat[i][i] != 0)
        {
            bool flag = false;
            for (ll j = i + 1; j <= n; j++)
            {
                if (mat[i][j] != 0)
                {
                    flag = true;
                    break;
                }
            }
            if (!flag)
            {
                mat[i][n + 1] = (mat[i][n + 1] * invs[mat[i][i]]) % mod;
                mat[i][i] = 1;
            }
        }
    }
}

// n x m grid with initial values of 0, 1, or 2, brushing a cell (i, j) adds +2 to its value and +1 to its neighbors, all
modulo 3, return one of the methods to make all cells have a final value of 0.
const ll mod = 3;
ll n, m, s;
void build(vvl &grid)
{
    ll dir[] = {0, -1, 0, 1, 0};
    for (ll i = 1; i < s + 1; i++)
    {
        for (ll j = 1; j < s + 2; j++)
        {
            mat[i][j] = 0;
        }
    }
    ll cur, row, col;
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            cur = i * m + j + 1;
            mat[cur][cur] = 2;
            for (ll d = 0; d < 4; d++)
            {
                row = i + dir[d];
                col = j + dir[d + 1];
                if (row > -1 && row < n && col > -1 && col < m)
                {
                    mat[cur][row * m + col + 1] = 1;
                }
            }
            mat[cur][s + 1] = (3 - grid[i][j]) % mod;
        }
    }
}

ll brush(vvl &grid)
{
    cin >> n >> m;
```

```
    s = n * m;
    build(grid);
    gauss(s);

    // simply assume all free variables are not operated on.
    // for (ll i = 1; i < s + 1; i++)
    // {
    //     if (mat[i][i] != 0)
    //     {
    //         mat[i][s + 1] = (mat[i][s + 1] * invs[mat[i][i]]) % mod;
    //     }
    // }
    // ans += mat[i][s + 1];
}

// given n tools and m records, each record contains a set of tools and information about their total manufacturing days, the manufacturing time for each tool is a fixed integer between 3 and 9 days, determine if the records are contradictory, have multiple solutions, or a unique solution, and print the corresponding result.
const ll mod = 7;
ll n, m;
void build()
{
    for (ll i = 1; i < s + 1; i++)
    {
        for (ll j = 1; j < s + 2; j++)
        {
            mat[i][j] = 0;
        }
    }
}

ll tool()
{
    cin >> n >> m;
    s = max(n, m);
    build();
    for (ll i = 1; i < m + 1; i++)
    {
        ll k, sta, end, x;
        cin >> k >> sta >> end;
        while (k--)
        {
            cin >> x;
            mat[i][x] = (mat[i][x] + 1) % mod;
        }
        mat[i][s + 1] = ((end - sta + 1) % mod + mod) % mod;
    }
    gauss(s);

    ll sign = 1;
    for (ll i = 1; i < s + 1; i++)
    {
        if (mat[i][i] == 0 && mat[i][s + 1] != 0)
        {
            sign = -1;
            break;
        }
        if (i < n + 1 && mat[i][i] == 0)
        {
            sign = 0;
        }
    }

    if (sign == 1)
    {
        for (ll i = 1; i < n + 1; i++)
        {
```

```
                if (mat[i][s + 1] < 3)
                {
                    cout << mat[i][s + 1] + 7 << " ";
                }
                else
                {
                    cout << mat[i][s + 1] << " ";
                }
            }
        }
        else
        {
            cout << sign << endl;
        }
}
```

11. 线性基
11.1 异或空间线性基
对于一个给定的整数集合 S，它的异或空间线性基是一个最小的集合 B。通过对 B 中的元素进行异或运算，可以
得到所有能由 S 中元素异或得到的数。
其不在乎每种异或值的个数，只关心去重的非零异或值能否全部得到。

性质：
1) 一堆数字中，任意的 a 和 b，用 a^b 的结果代替 a 或者 b，不会影响非零异或集合的组成。
2) 一堆数字中，任意的 a 和 b，如果有 a^b=0，去掉 a 或者 b，不会影响非零异或集合的组成。
3) 原始数组中，如果可以异或出 0，求出异或空间线性基后需要单独标记一下。

朴素消元法：

```
const ll maxn = 101;
const ll bit = 63;
vl arr(maxn), basis(bit + 1, 0);
ll n, flag;
bool insert(ll x)
{
    for (ll i = bit; i > -1; i--)
    {
        if ((x >> i) & 1)
        {
            if (basis[i] == 0)
            {
                basis[i] = x;
                return true;
            }
            x ^= basis[i];
        }
    }
    return false;
}
void build()
{
    flag = 0;
    for (ll i = 1; i < n + 1; i++)
    {
        if (!insert(arr[i]))
        {
            flag = 1;
        }
    }
}
```

朴素消元法得到的线性基个数 n 可以遍历 basis 的非零值轻松得到，而总的非零异或值的个数等于 2^n-1 个，最
大异或值只需从最高位线性基依次向下遍历比较 max = std::max(max, max^cur) 即可。

高斯消元法：

```
// gauss elimination to solve the linear basis
ll len;
vl arr(maxn), basis(maxn, 0);
void gbuild()
```

```
{
    len = 1;
    for (ll i = bit; i > -1; i--)
    {
        for (ll j = len; j < n + 1; j++)
        {
            if ((basis[j] & (1L << i)) != 0)
            {
                swap(basis[j], basis[len]);
                break;
            }
        }
        if ((basis[len] & (1L << i)) != 0)
        {
            for (ll j = 1; j < n + 1; j++)
            {
                if (j != len && (basis[j] & (1L << i)) != 0)
                {
                    basis[j] ^= basis[len];
                }
            }
            len++;
        }
    }
    len--;
    flag = (len != n ? 1 : 0);
}
```

高斯线性消元得到的标准线性基不仅可以得到朴素消元得到的性质，还可以求原数组所有非零异或集合中的第 k 小的异或值，即将 k 转化为二进制，然后将二进制中为 1 的位对应的基异或起来即可，例如 101001，则将第 0，3，5 个线性基（从小到大）异或起来。

```
ll kx(ll k)
{
    vl cb;
    for (int i = 1; i < len + 1; ++i)
    {
        if (basis[i] > 0)
        {
            cb.push_back(basis[i]);
        }
    }
    sort(cb.begin(), cb.end());

    if (k > (1LL << len) - 1)
    {
        return -1;
    }

    ll ans = 0;
    for (ll i = 0; i < len; ++i)
    {
        if ((k >> i) & 1)
        {
            ans ^= cb[i];
        }
    }
    return ans;
}
```

问：给定 n 个魔法矿石，每个矿石有状态和魔力值。一个矿石组合是有效的，如果其中没有任何非空子集的矿石状态异或和为 0。求所有有效组合中，最大的魔力总和。

解：将所有矿石按魔力值从大到小排序，创建一个空的线性基，对于每个矿石，尝试将其状态 s 插入到线性基中；如果 s 能够成功插入，这说明它与当前线性基是线性无关的，如果成功插入，就选择这个矿石，并将它的魔力值加到总和中，如果不能成功插入，说明 s 可以由线性基中的其他矿石异或得到，那么选择它会导致无效组合，所以放弃这个矿石。

可以说，线性基就是保证子集合没有异或值为 0 的最大集合，同时又是可以表示原数组所有异或值的最小集合。

问：给定 n 个初始熄灭的灯泡和 m 个开关。每个开关能改变一组灯泡的状态。可以随意使用开关，求最终有多少种不同的亮灯组合。
解：等价于求原数组的所有异或值的个数，使用高斯消元法，得到基底的大小 k，异或空间中的向量总数是 2^k，即所有可能亮灯组合的总数。

```
// 2024 CCPC Online Contest
ll n;
vl a(maxn), b(maxn), p(bit, 0);
void insert(ll x){
    for (ll i = bit - 1; i > -1; i--){
        if ((x >> i) & 1){
            if (!p[i]){
                p[i] = x;
                break;
            }
            x ^= p[i];
        }
    }
}
void solve(){
    cin >> n;
    ll sa = 0, sb = 0;
    for (ll i = 1; i < n + 1; i++){
        cin >> a[i];
        sa ^= a[i];
    }
    for (ll i = 1; i < n + 1; i++){
        cin >> b[i];
        sb ^= b[i];
        insert(a[i] ^ b[i]);
    }
    for (ll i = bit - 1; i > -1; i--)
    {
        ll xa = sa ^ p[i];
        ll xb = sb ^ p[i];
        // 由于 sa^sb = xa^xb，为此可以理解为保持总体异或不变的情况下使得两者“差值”减小，详细证明略
        if (max(xa, xb) < max(sa, sb)){
            sa = xa, sb = xb;
        }
    }
    cout << max(sa, sb) << '\n';
}
```

向量空间线性基

```
ll n, m;
vvl mat(maxn, vl(maxn));
bool insert(ll i)
{
    for (ll j = 1; j < m + 1; j++)
    {
        if (abs(mat[i][j]) >= sml)
        {
            if (basis[j] == 0)
            {
                basis[j] = i;
                return true;
            }
            double rate = mat[i][j] / mat[basis[j]][j];
            for (ll k = j; k < m + 1; k++)
            {
                mat[i][k] -= rate * mat[basis[j]][k];
            }
        }
    }
    return false;
}
```

12. 01 分数规划

```
// n points (a, b) and discard k points, find the maximum value of the ratio sum(a) / sum(b) for the remaining n-k points, the final result should be (ratio * 100) rounded to the nearest integer.
vvd arr(maxn, vd(3)); // arr[i][0] = ai, arr[i][1] = bi, arr[i][2] = ai - xbi
ll n, k;
bool check(double m)
{
    for (ll i = 0; i < n; i++)
    {
        arr[i][2] = arr[i][0] - m * arr[i][1];
    }
    sort(arr.begin(), arr.begin() + n, [](const vd &a, const vd &b)
         { return a[2] > b[2]; });

    double sum = 0;
    for (ll i = 0; i < k; i++)
    {
        sum += arr[i][2];
    }

    return sum >= 0;
}
void solve()
{
    // (a1 + a2 + a3)/(b1 + b2 + b3) = x
    // (a1 - xb1) + (a2 - xb2) + (a3 - xb3) = 0
    // binary check x for (a1 - xb1) + (a2 - xb2) + (a3 - xb3) >= 0
    while (n != 0 || k != 0)
    {
        k = n - k;
        double l = 0, r = 0, m;
        for (ll i = 0; i < n; i++)
        {
            cin >> arr[i][0];
            r += arr[i][0];
        }
        for (ll i = 0; i < n; i++)
        {
            cin >> arr[i][1];
        }

        double ans = 0;
        while (l < r && r - l >= sml)
        {
            m = (l + r) / 2;
            if (check(m))
            {
                ans = m;
                l = m + sml;
            }
            else
            {
                r = m - sml;
            }
        }
        cout << (ll)(100 * (ans + 0.005));
    }
}
```

13. 裴蜀定理——扩展欧几里得算法

13.1 基础性质

对于任意两个不全为零的整数 a 和 b，一定存在整数 x 和 y，使得 ax+by = gcd(a, b)，且有无穷多组解。

推论 1：gcd(a, b) = ax + by，其中 x 和 y 是所有可能配对中，使 ax + by 达到最小正数的组合。

推论 2：a, b 不全为 0，如果 a 和 b 互质，那么一定存在整数 x 和 y，使得 ax+by=1。

推论 3：a, b 不全为 0，则 ax+by=c 有解等价于 c 一定是 gcd(a, b) 的整数倍（不一定为正数）

推论 4：ax+by+cz = gcd(a, b, c)，且前三条推论同样可以推广

扩展欧几里得算法

```
ll d, x, y, px, py;
// ax + by = gcd(a, b)
// log min(a, b)^3
void exgcd(ll a, ll b) // a >= 0, b >= 0
{
    if (b == 0)
    {
        d = a;
        x = 1;
        y = 0;
    }
    else
    {
        exgcd(b, a % b);
        px = x;
        py = y;
        x = py;
        y = px - py * (a / b);
    }
}

// x = a^(-1) → (xa) % mod = 1 → xa - ?mod = 1
// exgcd(a, mod) = 1 → x = x, -? = y
```

扩展欧几里得算法还可以用于求解逆元：inv(a, mod) = exgcd(a, mod) → a^(-1) = (x % mod + mod) % mod
费马小定理求逆元时要求模数必须为质数，而扩展欧几里得算法不要求模数为质数，只要求 a 和模数互质即可。

问：给定长度为 n 的一组整数值 [a1, a2, a3...]，找到一组数值 [x1, x2, x3...]，要让 a1*x1 + a2*x2 + a3*x3...得到的结果为最小正整数。
解：由推论 1 的多元推广可知，求全体元素的总最大公因数即可。

问：一共有编号 1~n 的宝塔，其中 a 号和 b 号宝塔已经修好，Yuwgna 和 Iaka 两个人轮流修塔，Yuwgna 先手，Iaka 后手，谁先修完所有的塔谁赢，每次可以选择 j+k 号或者 j-k 号塔进行修理，其中 j 和 k 是任意两个已经修好的塔，输入 n、a、b，如果先手赢打印"Yuwgna"，后手赢打印"Iaka"。
解：分析可知可修的塔的编号为 a + ?b，其中“？”为整数即可，同理 ?a + b 也可修到，其余塔均不可修，为此，任意的 xa + yb 的塔都可以修到，假设总的可修塔数为 sum = xa + yb，则 sum 有解等价于 sum % gcd(a, b) == 0，所以 1~n 范围内，gcd(a, b) 的最大倍数即为总的修塔数，奇数先手赢，偶数后手赢。

问：如果有两个数字 step 和 mod，那么可以由以下方式生成很多数字，seed(1) = 0，seed(i+1) = (seed(i) + step) % mod。比如，step = 3、mod = 5，则 seed(1) = 0，seed(2) = 3，seed(3) = 1，seed(4) = 4，seed(5) = 2，如果能产生 0 ~ mod-1 所有数字，step 和 mod 的组合叫 "Good Choice"，否则"Bad Choice"。
解：原题等价于 k*step % mod 能否得到 1~mod-1，对于 1~mod-1 中的任意数 i，k*step % mod == i 成立等价于 k*step - ?mod == i 有解，等价于 i 是 gcd(step, mod) 的整数倍，所以 gcd(step, mod) == 1。

问：给定 a, b，求关于 x 的同余方程 $ax \equiv 1 \pmod b$ 的最小正整数解，题目保证有解。
解：题目等价于 xa % b = 1，即求 a 的逆元 (x % b + b) % b。

问：一共有 n 张牌，n 一定是偶数，每张牌的牌面从 1 到 n，洗牌规则如下：比如 n = 6，牌面最初排列为 1 2 3 4 5 6，先分成左堆 1 2 3，右堆 4 5 6，然后按照右堆第 i 张在前，左堆第 i 张在后的方式依次放置，所以洗一次后，得到 4 1 5 2 6 3，如果再洗一次，得到 2 4 6 1 3 5，如果再洗一次，得到 1 2 3 4 5 6，想知道 n 张牌洗 m 次的之后，第 L 张牌，是什么牌面。
解：观察每张牌在洗 m 轮后的位置变化：1: 1 2 4 1，2: 2 4 1 2，3: 3 6 5 3，4: 4 1 2 4 ...，注意到，每张牌洗完后位置乘 2，且到达 7 之后回到 1，所以猜测 (x*(2^m)) % 7 即为初始第 x 张牌最后的位置，先已知洗完 m 轮后，反推 x 的位置，即 (x*(2^m)) % 7 == L，将 n 代入式子，等价于 x*(2^m) - ?(n+1) == L，由奇偶性可知，2^m 与 n+1 必定互质，所以扩展欧几里得求得 X_0*(2^m) + Y_0(n+1) == 1，先将 X_0 转化为正数（此处也可以不转，那么得到的 x 将为负数，在那时转也行，不影响答案），X_0 = (X_0 % (n+1) + (n+1)) % (n+1)，x = (X_0 * L) % (n + 1)。

13.2 二元一次不定方程
ax + by = d，其中 d 为 a 和 b 的最大公因数，解得 x_0, y_0 为不定方程的特解（扩展欧几里得解得），则有：

$$\begin{cases} x = x_0 + \frac{b}{d}n \\ y = y_0 - \frac{a}{d}n \end{cases}$$

其中 n 为任意整数，若 ax + by = c，其中 c 为 d 的整数倍，代入该方程特解同样符合上式，由上述式子也可以看出，x 和 y 是此消彼长的，所以虽然解得数量无穷，但是两者都为正整数的解有限，甚至可能没有。

```
// ax + by = c, no solution, return -1; no positive integer solutions, return the minimum positive values of x and y; positive integer solutions, return the number of positive integer solutions, as well as the minimum and maximum positive values of x and y.
void solve(ll a, ll b, ll c)
{
    exgcd(a, b);
    if (c % d != 0)
    {
        cout << -1 << endl;
        return;
    }

    x *= c / d;
    y *= c / d;

    ll xd = b / d, yd = a / d;
    if (x < 0)
    {
        // (1 - x)/xd ↑
        ll tims = (1 - x + xd - 1) / xd;
        x += xd * tims;
        y -= yd * tims;
    }
    else
    {
        // (x - 1)/xd ↓
        ll tims = (x - 1) / xd;
        x -= xd * tims;
        y += yd * tims;
    }

    if (y < 1)
    {
        ll miy = y + yd * ((1 - y + yd - 1) / yd);
        cout << x << " " << miy << endl;
    }
    else
    {
        ll cnt = (y - 1) / yd + 1;
        cout << cnt << " ";

        ll miy = y - yd * ((y - 1) / yd);
        cout << x << " " << miy << " ";

        ll max = x + xd * ((y - 1) / yd);
        cout << max << " " << y << endl;
    }
}

// circle length l, frog A is at position x1 and jumps m units per second, while frog B is at x2 and jumps n units per second, both move clockwise, the minimum time required for them to meet.
ll meet(ll l, ll sa, ll sb, ll va, ll vb)
{
    // sa < sb, va > vb
    // (va - vb)t = sb - sa + kl

    // sa > sb, va < vb
    // (vb - va)t = sa - sb + kl

    //  sa < sb, va < vb
    // (vb - va)t = l - (sb - sa) + kl = sa - sb + kl

    // sb < sa, va > vb
```

```
    // (va - vb)t = l - (sa - sb) + kl = sb - sa + kl

    ll a, b = l, c;
    if (sa < sb)
    {
        a = va - vb;
        c = sb - sa;
    }
    else
    {
        a = vb - va;
        c = sa - sb;
    }

    if (a < 0)
    {
        a = -a;
        c = l - c;
    }
    exgcd(a, l);

    if (c % d != 0)
    {
        cout << "Impossible";
    }
    else
    {
        x *= c / d;
        ll xd = l / d;
        if (x < 0)
        {
            x += (1 - x + xd - 1) / xd * xd;
        }
        else
        {
            x -= (x - 1) / xd * xd;
        }
        cout << x;
    }
}

// points A(xa, ya) and B(xb, yb), lattice point is a point with integer coordinates, find the number of lattice points on the line segment AB, including A and B.
ll points(ll xa, ll xb, ll ya, ll yb)
{
    // (yb - ya)x + (xa - xb)y = yb.xa - ya.xb
    // x = xa + ((xa -xb)/gcd) * n
    ll g = gcd(abs(xa - xb), abs(ya - yb));
    return g + 1;
}
```

鞋带公式，顺时针求解凸多边形面积：
S = ((x2y1 - x1y2) + (x3y2 - x2y3) + (x4y3 - x3y4) + (x1y4 - x4y1)) / 2，更多点以此类推
逆时针鞋带公式：S = ((x1y2 - x2y1) + (x2y3 - x3y2) + (x3y4 - x4y3) + (x4y1 - x1y4)) / 2

皮克定理：
如果多边形的顶点都是格点，则有以下等式：面积 = 内点数 + 边点数/2 + 1

```
// robot starts at (0,0) and moves according to n given displacements (dx, dy), forming a counter-clockwise simple polygon, find the number of lattice points inside the polygon, the number of lattice points on its boundary, and the area of the polygon.
void robot()
{
    ll n, egs = 0, area = 0;
    for (ll i = 0, x = 0, y = 0, dx, dy; i < n + 1; i++)
    {
        cin >> dx >> dy;
```

```
        egs += gcd(abs(dx), abs(dy));
        area += x * (y + dy) - (x + dx) * y;

        x += dx;
        y += dy;
    }

    area /= 2;
    ll inn = area - egs / 2 + 1;
    cout << egs << " " << area << " " << inn << endl;
}
```

问：给定两种互质的硬币面值 a 和 b，每种硬币数量无限。求无法由这两种硬币组成的钱数中的最大值。
解：由于 a 和 b 互质，所以 ax+by=1 一定有解，所以 ax+by=c 一定有解，题目等价于求最大的 c，使得 x,y 没有非负数解，观察到要么 x=-1，要么 y=-1，因为如果取其他负数，c 将会变小，假设 x=-1，y 最大取得 a-1，因为一旦大于等于 a，原式等于 -a+ab=(b-1)a=c，此时存在 x=b-1, y=0 的解不符合题意，y=-1 同理，所以最终答案等于 ab-a-b。

14. 中国剩余定理
14.1 求解一组线性同余方程：

$$\begin{aligned} x \% m_1 &= r_1 \\ x \% m_2 &= r_2 \\ x \% m_3 &= r_3 \end{aligned}$$

尝试将 x 分为 n 部分（n 为方程组的数量），上述例子中 $x = C_1 + C_2 + C_3$，更有构造：

$$\begin{aligned} C_1 &= r_1 * m_2 * m_3 * (m_2 * m_3)^{-1} (\% m_1) \\ C_2 &= r_2 * m_1 * m_3 * (m_1 * m_3)^{-1} (\% m_2) \\ C_3 &= r_3 * m_1 * m_2 * (m_1 * m_2)^{-1} (\% m_3) \end{aligned}$$

使得 $C_i \% m_i = r_i$, $C_i \% m_k = 0$, $k != i$。由于涉及扩展欧几里得求解逆元，所以要求所有的模数两两互质，此时得到的解只是无穷多组解中的一个，其中最小的正数解 $x_0 = (C_1 + C_2 + C_3) \% \operatorname{lcm}(m_1, m_2, m_3)$，所以 x 的通解为 $k * lcm + x_0$。

```
ll mul(ll a, ll b, ll mod)
{
    a = (a % mod + mod) % mod;
    b = (b % mod + mod) % mod;
    ll ans = 0;
    while (b != 0)
    {
        if ((b & 1) != 0)
        {
            ans = (ans + a) % mod;
        }
        a = (a + a) % mod;
        b >>= 1;
    }
    return ans;
}
vl m(maxn, 0), r(maxn, 0);
ll crt(ll n)
{
    ll lcm = 1;
    for (ll i = 1; i < n + 1; i++)
    {
        lcm = lcm * m[i];
    }
    ll ai, ci, ans = 0;
    for (ll i = 1; i < n + 1; i++)
    {
        // ai = lcm / m[i]
        ai = lcm / m[i];
        // ai^(-1)
        exgcd(ai, m[i]);
        // ci = (ri * ai * ai^(-1)) % lcm
        ci = mul(r[i], mul(ai, x, lcm), lcm);
        ans = (ans + ci) % lcm;
    }
    return ans;
```

```
}
```

14.2 扩展中国剩余定理

首先由扩展欧几里得算法推广出一个结论：

$ax + by = c$ 的通解 x 的最小非负解等于 $(x_0 * c/d) \% (b/d)$ 再取非负余数，其中 x_0 为 $ax + by = gcd(a, b)$ 由扩展欧几里得求出的特解，$d = gcd(a, b)$。

扩展中国剩余定理不再要求所有的模数之间互质，具体证明略：

```
ll excrt(ll n)
{
    ll tail = 0, lcm = 1, tmp, b, c, x0;
    // ans = lcm * x + tail
    for (ll i = 1; i < n + 1; i++)
    {
        // ans = m[i] * y + ri
        // lcm * x + m[i] * y = ri - tail
        // a = lcm
        // b = m[i]
        // c = ri - tail
        b = m[i];
        c = ((r[i] - tail) % b + b) % b;
        exgcd(lcm, b);
        // 无解返回 -1
        if (c % d != 0)
        {
            return -1;
        }
        // x0 = (x * (c/d)) % (b/d)
        // x = x0 + (b/d) * n
        x0 = mul(x, c / d, b / d);
        // ans = lcm * x + tail
        // ans = lcm * (x0 + (b/d) * n) + tail
        // ans = lcm * (b/d) * n + lcm * x0 + tail
        // tail' = tail' % lcm'
        tmp = lcm * (b / d);
        tail = (tail + mul(x0, lcm, tmp)) % tmp;
        lcm = tmp;
    }
    return tail;
}
```

问：一共 n 只巨龙，每只巨龙都有初始血量 hp[i]，每只巨龙都有恢复能力 recovery[i]；

// 每只巨龙都会在攻击结束后开始恢复，初始一共 m 把剑，每把剑攻击力 init[i]；

// 每只巨龙只有当血量恰好为 0 时，才能被杀死。面对某只具体的龙，只能用固定的剑来攻击，规定如下：

// 攻击力不高于当前巨龙的血量，并且攻击力最大的一把剑，如果没有这样的剑，就选择攻击力最低的一把剑；

// 需要按 1~n 的顺序依次讨伐巨龙，i 号巨龙被杀后，那把攻击的剑会消失，同时奖励攻击力 reward[i]的剑；

// 勇士制定的策略如下，不管面对什么巨龙，攻击过程只打击 ans 下，让当前巨龙的血量小于等于 0；

// 然后在当前巨龙恢复的过程中，如果血量恰好为 0，那么当前巨龙被杀死，勇士继续讨伐下一只；

// 你的任务是算出最小的 ans，让勇士可以在该策略下杀死所有巨龙；

// 如果在固定打击次数的策略下，就是无法杀死所有巨龙，返回 -1。

解：

对于第 i 只龙，使用攻击力为 a_i 的剑打击，h_i 为血量，r_i 为单次恢复量，ans 为固定打击次数，所以对于每只龙可以列出如下同余方程：

$$(a_i * ans) \% r_i = (h_i \% r_i)$$

求解最小正整数解 ans，由于 $a_i * ans$ 不固定，所以本题是扩展中国剩余定理的再扩展，并且必须保证每只龙的血量降至 0 以下，所以是求 $ans \geq max$ 的最小正整数解，其中 $max = std::max((h_i + a_i - 1)/a_i)$

```
ll hp[maxn], rc[maxn], rd[maxn], init[maxn], at[maxn];
map<ll, ll> sorted;
ll allocate(ll n, ll m){
    sorted.clear();
    for (ll i = 1; i < m + 1; i++){
        sorted[init[i]]++;
    }

    ll max_hits = 0;
    for (ll i = 1; i < n + 1; i++){
        auto it = sorted.upper_bound(hp[i]);
```

```
            ll ai;
            if (it == sorted.begin()){
                ai = sorted.begin()->first;
            }
            else
            {
                ai = (--it)->first;
            }

            at[i] = ai;
            sorted[ai]--;
            if (sorted[ai] == 0){
                sorted.erase(ai);
            }
            sorted[rd[i]]++;

            max_hits = max(max_hits, (hp[i] + at[i] - 1) / at[i]);
            hp[i] %= rc[i];
        }
        return max_hits;
}

ll compute(ll n, ll m){
        ll max_val = allocate(n, m);
        ll tail = 0, lcm = 1, tmp, a, b, c, x0;

        for (ll i = 1; i <= n; i++){
            // at[i] * ans ≡ hp[i] (mod rc[i])
            // ans = k * lcm + tail
            // -> at[i] * (k*lcm + tail) ≡ hp[i] (mod rc[i])
            // -> (at[i]*lcm) * k ≡ hp[i] - at[i]*tail (mod rc[i])
            a = mul(at[i], lcm, rc[i]);
            b = rc[i];
            c = ((hp[i] - mul(at[i], tail, b)) % b + b) % b;

            exgcd(a, b);
            if (c % d != 0){
                return -1;
            }

            ll b_d = b / d;
            x0 = mul(x, c / d, b_d);

            tmp = lcm * b_d;
            tail = (tail + mul(x0, lcm, tmp)) % tmp;
            lcm = tmp;
        }

        ll ans;
        if (tail >= max_val){
            ans = tail;
        }
        else{
            // ans = k * lcm + tail >= max_val
            // k >= (max_val - tail) / lcm
            // k_min = ceil((max_val - tail) / lcm)
            ll k = (max_val - tail + lcm - 1) / lcm;
            ans = k * lcm + tail;
        }
        return ans;
}
```

15. 二项式定理

15.1 二项式定理

$$(a+b)^n = \sum_{k=0}^{n} C_n^k * a^{n-k} * b^k$$

特别地，当 $a = 1, b = 1$ 时，得到杨辉三角，且 $2^n = C_n^0 + C_n^1 + \ldots + C_n^n$ 以及组合恒等式：

$$C_i^j = C_i^{j-1} + C_{i-1}^{j-1}$$

```
ll fac[10001], finvs[10001];
void lin_finvs(ll n, ll p)
{
    fac[0] = 1;
    fac[1] = 1;
    for (ll i = 2; i < n + 1; i++)
    {
        fac[i] = (i * fac[i - 1]) % p;
    }

    finvs[n] = inv(fac[n], p);
    for (ll i = n - 1; i >= 0; i--)
    {
        finvs[i] = (finvs[i + 1] * (i + 1)) % p;
    }
}

ll comb(ll n, ll m, ll p)
{
    if (m < 0 || m > n)
        return 0;

    lin_finvs(n, p);

    ll ans = (((fac[n] * finvs[m]) % p) * finvs[n - m]) % p;
    return ans;
}

// (i, j) is valid if the C(i, j) % k = 0, find the total number of valid pairs within the ranges i ~ [0, n] and j ~ [0, min(i,
m)], all test cases share the same value of k.
vvl c(maxn, vl(maxn, 0));
vvl f(maxn, vl(maxn, 0));
vvl sum(maxn, vl(maxn, 0));
ll k;
void build()
{
    for (ll i = 0; i < maxn; i++)
    {
        c[i][0] = 1; // C(i, 0) = 1
        for (ll j = 1; j < i + 1; j++)
        {
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % k;
        }
    }

    // C(i, j) % k == 0
    for (ll i = 1; i < maxn; i++)
    {
        for (ll j = 1; j < i + 1; j++)
        {
            f[i][j] = (c[i][j] == 0) ? 1 : 0;
        }
    }

    // (0,0) to (i,j)
    for (ll i = 1; i < maxn; i++)
    {
        for (ll j = 1; j < i + 1; j++)
        {
            sum[i][j] = sum[i][j - 1] + sum[i - 1][j] - sum[i - 1][j - 1] + f[i][j];
        }
        if (i + 1 < maxn)
        {
```

```
            sum[i][i + 1] = sum[i][i];
        }
    }
}
```

问：给定一个长度为 n 的数组 A，将其分割成数组 B 和数组 C，满足 A[i] = B[i] + C[i]，也就是一个数字分成两份，然后各自进入 B 和 C，要求 B[i], C[i] >= 1。同时要求，B 数组从左到右不能降序，C 数组从左到右不能升序。比如，A = { 5, 4, 5 }，一种有效的划分，B = { 2, 2, 3 }，C = { 3, 2, 2 }，返回有多少种有效的划分方式。
解：假设数组中只有一种数值 V，则其有 k=V-1 种分裂方式，可以观察到总的方法数为 C(n+k-1, n)，现在讨论不同数字的情况，首先第一个数依旧有 $k_1 = A[1] - 1$ 种分裂方式，来到第 i 个数时，如果前一个数比当前值小，维持 $k_i = k_{i-1}$ 种分裂方式，反之 $k_i = k_{i-1} - (A[i-1] - A[i])$，如果最终 k 小于等于 0，最终方案数为 0，否则为 C(n+k-1, n)。

15.2 二项式反演

$$g(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f(i) \Longleftrightarrow f(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g(i)$$

$$g(n) = \sum_{i=0}^{n} \binom{n}{i} f(i) \Longleftrightarrow f(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} g(i)$$

$$g(n) = \sum_{i=n}^{N} (-1)^i \binom{i}{n} f(i) \Longleftrightarrow f(n) = \sum_{i=n}^{N} (-1)^i \binom{i}{n} g(i)$$

$$g(n) = \sum_{i=n}^{N} \binom{i}{n} f(i) \Longleftrightarrow f(n) = \sum_{i=n}^{N} (-1)^{i-n} \binom{i}{n} g(i)$$

错排问题（反演公式 2）：
假设 g(n) 表示 n 个人全排列的方法数，f(n) 表示指定具体 n 个人错排的方法数，则有：

$$g(n) = \sum_{i=0}^{n} \binom{n}{i} f(i) = n!$$

为此，反演得到：

$$f(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} i! = \sum_{i=0}^{n} (-1)^i \frac{n!}{i!}$$

```
ll derg(ll n)
{
    ll facn = 1; // n!
    for (ll i = 1; i < n + 1; i++)
    {
        facn *= i;
    }
    ll ans = facn; // i = 0
    ll faci = 1;    // i!
    for (ll i = 1; i < n + 1; i++)
    {
        // i = 1...n
        // (-1)^i * (n! / i!)
        faci = faci * i;
        if ((i & 1) == 0)
        {
            ans += facn / faci;
        }
        else
        {
            ans -= facn / faci;
        }
    }
    return ans;
}
```

具体二项式反演的证明略，证明过程中会用到二项式系数恒等式：

$$C_n^i \bullet C_i^j = C_n^j \bullet C_{n-j}^{n-i}$$

集合计数：有 n 个不同的数，能构成 2^n 个不同集合，在 2^n 个集合中挑出若干个集合，至少挑一个，希望这若干

个集合的交集，正好有 k 个数（反演公式 4）。
定义 g(i)，表示任选 i 个元素作为钦定，钦定不同就认为方案不同！挑选集合后，形成的交集至少包含钦定的 i 个元素，总的方案数，根据定义，可以得到如下式子：

$$g(i) = \binom{n}{i}(2^{2^{n-i}} - 1)$$

注意在求解的过程中次方的同余较为复杂 $a^b \% mod \neq a^{b \% mod} \% mod$，所以先单独处理次方项（倒推）：

$2^{2^{n-i}}$: ... 256 16 4 2

观察到前一项为后一项的平方，所以乘法同余：

```
ll tmp = 2;
for (ll i = n; i >= 0; i--){
    g[i] = tmp;
    tmp = tmp * tmp % mod;
}
```

再将其他部分处理好即可：

```
for (ll i = 0; i <= n; i++){
    g[i] = (g[i] + mod - 1) * c(n, i) % mod;
}
```

$$g(n) = \sum_{i=n}^{N} \binom{i}{n} f(i) \Longleftrightarrow f(n) = \sum_{i=n}^{N} (-1)^{i-n} \binom{i}{n} g(i)$$

```
long ans = 0;
for (ll i = k; i <= n; i++){
    if (((i - k) & 1) == 0)
    {
        ans = (ans + c(i, k) * g[i] % mod) % mod;
    }
    else
    {
            ans = (ans + c(i, k) * g[i] % mod * (mod - 1) % mod) % mod;
    }
}
```

分特产：有 m 种特产，arr[i]表示 i 种特产有几个，有 n 个同学，每个同学至少要得到一个特产，返回分配特产的方法数（反演公式 4）。
定义 g(i)，表示一定要分光所有特产，任选 i 人作为钦定，不获得任何特产，钦定不同就认为方案不同！最终所有无特产的人中，至少包含钦定的 i 个人，总方案数；定义 f(i)，表示一定要分光所有特产，最终有 i 个人无特产，总方案数，最终答案就是 f(0)。
求解 g(i) 需要用到隔板法，即 m 个相同的物品分给 n 个不同的人，总的方案数为 $C(m+n-1, n-1)$，所以：

$$g(i) = C_n^i \prod_{j=1}^{m} C_{arr[j]+n-i-1}^{n-i-1}$$

16. 康托展开
16.1 计算一个给定排列在所有可能的全排列中的字典序排名。

```
ll arr[maxn], fac[maxn], tree[maxn];
ll n;
ll lowbit(ll i) {
    return i & -i;
}
ll sum(ll i) {
    ll ans = 0;
    while (i > 0) {
        ans = (ans + tree[i]) % mod;
        i -= lowbit(i);
    }
    return ans;
}
void add(ll i, ll v) {
    while (i <= n) {
        tree[i] += v;
        i += lowbit(i);
    }
}

ll compute() {
    fac[0] = 1;
```

```
    for (ll i = 1; i < n; i++) {
        fac[i] = (fac[i - 1] * i) % mod;
    }

    for (ll i = 1; i <= n; i++) {
        tree[i] = 0;
    }
    for (ll i = 1; i <= n; i++) {
        add(i, 1);
    }

    ll ans = 0;
    for (ll i = 1; i <= n; i++) {
        ans = (ans + (sum(arr[i] - 1) * fac[n - i]) % mod) % mod;
        add(arr[i], -1);
    }

    ans = (ans + 1) % mod; // 从 1 开始
    return ans;
}
```

16.2 逆康托展开

当 n 很大时，长度为 n 的某个排列在全排列中的位置可能很大并溢出，所以需要引入阶乘进制，阶乘进制可以表示所有长度为 n 的排列的排位——对于十进制 7 来说，采用阶乘进制表示如下：

$$\text{rank}\,[1 \quad 0 \quad 1 \quad 0]$$

即 $7 = 1 * 3! + 0 * 2! + 1 * 1! + 0 * 0!$，本质上与十进制以及二进制没有区别，而阶乘进制加法的算法如下：

```
ll arr[maxn], sum[maxn << 2];
ll n, m;

void build(ll l, ll r, ll i){
    if (l == r){
        sum[i] = 1;
    }
    else{
        ll mid = (l + r) >> 1;
        build(l, mid, i << 1);
        build(mid + 1, r, i << 1 | 1);
        sum[i] = sum[i << 1] + sum[i << 1 | 1];
    }
}

void add(ll jobi, ll jobv, ll l, ll r, ll i){
    if (l == r){
        sum[i] += jobv;
    }
    else{
        ll mid = (l + r) >> 1;
        if (jobi <= mid)
        {
            add(jobi, jobv, l, mid, i << 1);
        }
        else
        {
            add(jobi, jobv, mid + 1, r, i << 1 | 1);
        }
        sum[i] = sum[i << 1] + sum[i << 1 | 1];
    }
}

ll qsum(ll jobl, ll jobr, ll l, ll r, ll i){
    if (jobl <= l && r <= jobr)
    {
        return sum[i];
    }
    ll mid = (l + r) >> 1;
    ll ans = 0;
    if (jobl <= mid){
```

```
        ans += qsum(jobl, jobr, l, mid, i << 1);
    }
    if (jobr > mid){
        ans += qsum(jobl, jobr, mid + 1, r, i << 1 | 1);
    }
    return ans;
}
// 线段树上找到第 k 名的是什么，找到后删掉词频，返回的过程修改累加和
ll gd(ll k, ll l, ll r, ll i){
    ll ans;
    if (l == r){
        sum[i]--;
        ans = l;
    }
    else{
        ll mid = (l + r) >> 1;
        if (sum[i << 1] >= k){
            ans = gd(k, l, mid, i << 1);
        }
        else{
            ans = gd(k - sum[i << 1], mid + 1, r, i << 1 | 1);
        }
        sum[i] = sum[i << 1] + sum[i << 1 | 1];
    }
    return ans;
}

void compute(){
    // 线段树
    build(1, n, 1);
    // 将自然排位转化为阶乘排位
    for (ll i = 1; i < n + 1; i++){
        ll x = arr[i];
        if (x > 1){
            arr[i] = qsum(1, x - 1, 1, n, 1);
        }
        else{
            arr[i] = 0;
        }
        add(x, -1, 1, n, 1);
    }

    // 阶乘排位加上 m 后得到新的阶乘排位
    arr[n] += m;
    for (ll i = n; i > 1; i--){
        arr[i - 1] += arr[i] / (n - i + 1);
        arr[i] %= (n - i + 1);
    }

    build(1, n, 1);
    // 将更新后的阶乘排位还原为自然排位
    for (ll i = 1; i < n + 1; i++){

        arr[i] = gd(arr[i] + 1, 1, n, 1); // 1-base
    }
}
```

17. 约瑟夫环问题

有 n 个人围成一圈，从某个人开始按顺序报数。每当数到 k 的人就会被淘汰出局，然后从他身后的人开始重新报数，直到最后只剩下一个人。

```
ll joseph(ll n, ll k)
{
    ll ans = 1;
    for (ll c = 2; c < n + 1; c++)
    {
        ans = (ans + k - 1) % c + 1;
    }
```

```
    return ans;
}
```

约瑟夫环问题加强版，有 1~n 点，组成首尾相接的环，游戏一共有 n-1 轮，每轮给定一个数字 arr[i]，第一轮游戏中，1 号点从数字 1 开始报数，哪个节点报到数字 arr[1]，就删除该节点，然后下一个节点从数字 1 开始重新报数，游戏进入第二轮，第 i 轮游戏中，哪个节点报到数字 arr[i]，就删除该节点，然后下一个节点从数字 1 开始重新报数，游戏进入第 i+1 轮，最终环上会剩下一个节点，返回该节点的编号。

```
// k → arr[i]
ll joseph(ll n, vl &arr)
{
    if (n == 1)
    {
        return 1;
    }
    ll ans = 1;
    for (ll c = 2, i = n - 1; c < n + 1; c++, i--)
    {
        ans = (ans + arr[i] - 1) % c + 1;
    }
    return ans;
}
```

18. 完美洗牌

给定数组 arr，给定某个范围 arr[l..r]，该范围长度为 n，n 是偶数，将给定的范围分成左右两个部分，arr[l1, l2, ...lk, r1, r2, ...rk]，请把 arr[l..r]范围上的数字调整成 arr[r1, l1, r2, l2, ... rk, lk]，其他位置的数字不变。

完美洗牌应用：将数组按照如下规则排序：

$$arr[0] < arr[1] > arr[2] < arr[3] > \dots$$

解：利用随机选择排序算法选择一个中心数将原数组大致分成左右等分，然后使用完美洗牌算法将数组调整，最后根据数组长度奇偶性加入数组逆序逻辑即可。

```
ll first, last;
ll sele(vl &arr, ll n, ll i)
{
    ll ans = 0;
    for (ll l = 0, r = n - 1; l < r + 1;)
    {
        pt(arr, l, r, arr[l + (ll)(rand() * (r - l + 1))]);
        if (i < first)
        {
            r = first - 1;
        }
        else if (i > last)
        {
            l = last + 1;
        }
        else
        {
            ans = arr[i];
            break;
        }
    }
    return ans;
}
void pt(vl &arr, ll l, ll r, ll x)
{
    first = l;
    last = r;
    ll i = l;
    while (i < last + 1)
    {
        if (arr[i] == x)
        {
            i++;
        }
        else if (arr[i] < x)
        {
            swap(arr[first++], arr[i++]);
```

```
        }
        else
        {
            swap(arr[i], arr[last--]);
        }
    }
}
void gsort(vl &arr, ll n)
{
    sele(arr, n, n / 2);
    if ((n & 1) == 0)
    {
        shuffle(arr, 0, n - 1);
        reverse(arr, 0, n - 1);
    }
    else
    {
        shuffle(arr, 1, n - 1);
    }
}
```

19. 卡特兰数

$$[1, 1, 2, 5, 14, 42, 132, 429, \ldots]$$

$$f(n) = \left(\frac{2n}{n}\right) - \left(\frac{2n}{n-1}\right)$$

$$f(n) = \left(\frac{2n}{n}\right)/(n+1)$$

$$f(n) = f(n-1)_{n-1} \cdot (4n-2)/(n+1)$$

$$f(n) = \sum_{i=0} f\ (i) \cdot f(n-1-i)$$

以上四个式子均可得到卡特兰数，其中前三个式子为线性时间，最后一个为平方时间。

19.1 出栈顺序模型（公式 1）：进栈顺序为 1、2、3 ... n，求有多少种不同的出栈顺序。

解：将入栈等价为左括号，出栈等价为右括号，任意前缀上左括号的数量大于等于右括号，求合法的括号串数量。

而由 n 个左括号和 n 个右括号，任意组合拼成的括号串共有 $C(2n, n)$ 种，而违规的括号串的规模等于由 n-1 个左括号和 n+1 个右括号任意组合拼成的括号串的规模，即 $C(2n, n-1)$，证明略。

```
vl fac(maxn), finv(maxn), inv(maxn);
void build(ll n)
{
    fac[0] = 1;
    fac[1] = 1;
    for (ll i = 2; i < n + 1; i++)
    {
        fac[i] = (i * fac[i - 1]) % mod;
    }

    finv[0] = 1;
    finv[n] = qpow(fac[n], mod - 2);
    for (ll i = n - 1; i > 0; i--) // i!^(-1) % p
    {
        finv[i] = ((i + 1) * finv[i + 1]) % mod;
    }

    inv[1] = 1;
    for (ll i = 2; i < n + 2; i++) // i^(-1) % p
    {
        inv[i] = (mod - (inv[mod % i] * (mod / i)) % mod) % mod;
    }
}

ll c(ll n, ll k)
{
    if (k < 0 || k > n)
        return 0;
```

```
    return (((fac[n] * finv[k]) % mod) * finv[n - k]) % mod;
}

// C(n) = C(2n, n) - C(2n, n-1)
ll f1(ll n)
{
    if (n == 0)
        return 1;
    build(2 * n);
    return (c(2 * n, n) - c(2 * n, n - 1) + mod) % mod;
}

// C(n) = C(2n, n) / (n+1)
ll f2(ll n)
{
    if (n == 0)
        return 1;
    build(2 * n);
    return (c(2 * n, n) * qpow(n + 1, mod - 2)) % mod;
}

// C(n) = (4n-2)/(n+1) * C(n-1)
ll f3(ll n)
{
    build(n);
    vl f(n + 1);
    f[0] = 1;
    if (n >= 1)
        f[1] = 1;
    for (ll i = 2; i < n + 1; i++)
    {
        f[i] = f[i - 1] * (4 * i - 2) % mod * inv[i + 1] % mod;
    }
    return f[n];
}

// C(n) = sum_{i=0..n-1} C(i) * C(n-1-i)
ll f4(ll n)
{
    vl f(n + 1);
    f[0] = 1;
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll l = 0, r = i - 1; l < i; l++, r--)
        {
            f[i] = (f[i] + f[l] * f[r]) % mod;
        }
    }
    return f[n];
}
```

注：除上述两种模型是卡特兰数外，还有一些经典模型：
1）门票 5 元，卖票人没钱，n 个人有 5 元，n 个人有 10 元，确保顺利找零，返回排队方式数。
2）圆上有 2n 个点，将这些点成对连线，形成 n 条线，任意两条线不可相交，返回连接方法数。

19.2 路径计数模型（公式 1）：正方形中，从 (0, 0) 到 (n, n)，每次只能向上或向右，可以碰对角线，但不能走出右下半区，返回路径总数。不考虑违规总的路径数为 $C(2n, n)$ 种，而违规路径按照 $y = x + 1$ 翻转等价于 $C(2n, n-1)$，所以总的路径数依旧是卡特兰数：

$$\left(\frac{2n}{n}\right) - \left(\frac{2n}{n-1}\right)$$

变式：正方形中，从 (0, 0) 到 (n, n)，每次只能向上或向右，不可以碰对角线，返回路径总数。

$$2\left(\frac{2n-2}{n-1}\right) - 2\left(\frac{2n-2}{n-2}\right)$$

19.3 划分左右相乘模型（公式 4）：有 n+2 条边的凸多边形，通过顶点连线将其完全划分为三角形，连线不相交，返回划分方式；n 个节点，节点无差别，返回可以形成多少种不同结构的二叉树（这类题多用递归分析）。

模数总结：
1）公式 4 虽然时间复杂度最高，但是无需使用逆元，所以在模数不是质数时，无法使用费马小定理，同时由于要求多次逆元也没法使用扩展欧几里得算法，所以 $n \le 10^3$ 时优先考虑公式 4。
2）倘若 $n = 10^6$，且模数为合数，此时只能使用公式 2 + 因子计数法：

```
vl minpf(maxn), prime(maxn), ct(maxn);
ll cnt;
void euler(ll n)
{
    fill(minpf.begin(), minpf.begin() + n + 1, 0);
    cnt = 0;
    for (ll i = 2; i < n + 1; i++)
    {
        if (minpf[i] == 0)
        {
            prime[cnt++] = i;
        }
        for (ll j = 0; j < cnt; j++)
        {
            if (i * prime[j] > n)
            {
                break;
            }
            minpf[i * prime[j]] = prime[j];
            if (i * prime[j] == 0)
            {
                break;
            }
        }
    }
}

ll compute(ll n, ll mod)
{
    euler(2 * n);

    fill(ct.begin(), ct.begin() + 2 * n + 2, 0);
    for (ll i = 2; i < n + 1; ++i)
        ct[i] = -1;

    for (ll i = n + 2; i < 2 * n + 1; ++i)
        ct[i] = 1;

    for (ll i = 2 * n; i > 1; i--)
    {
        if (minpf[i] != 0)
        {
            ct[minpf[i]] += ct[i];
            ct[i / minpf[i]] += ct[i];
            ct[i] = 0;
        }
    }

    ll ans = 1;
    for (ll i = 2; i < 2 * n + 1; i++)
    {
        if (ct[i] != 0)
        {
            ans = ans * qpow(i, ct[i]) % mod;
        }
    }
    return ans;
}
```

3）倘若 $n = 10^3$，且要求打印真实结果，需要高精度

20.其他

20.1 倍增循环构造

```
vb comp(1e5 + 1);
vl prim;
void sieve()
{
    for (ll i = 2; i * i <= 1e5; i++)
        if (!comp[i])
            for (ll j = i * i; j <= 1e5; j += i)
                comp[j] = true;
    for (ll i = 2; i <= 1e5; i++)
        if (!comp[i])
            prim.push_back(i);
}

void solve()
{
    ll n, cnt;
    cin >> n;
    vl p(n + 1);
    for (auto it = prim.rbegin(); it != prim.rend(); ++it)
    {
        vl cycle;
        cnt = 0;
        // n = 10
        // i = 7 => cycle = {7}
        // i = 5 => cycle = {5, 10}
        // i = 3 => cycle = {3, 6, 9}
        // i = 2 => cycle = {2, 4, 8}
        for (ll i = *it; i <= n; i += *it)
            if (!p[i])
            {
                cycle.push_back(i);
                cnt++;
            }

        // p[7] = 7
        // p[5] = 10, p[10] = 5
        // p[3] = 6, p[6] = 9, p[9] = 3
        // p[2] = 4, p[4] = 6, p[8] = 2
        for (ll i = 0; i < cnt; i++)
            p[cycle[i]] = cycle[(i + 1) % cnt];
    }

    // p[1] = 1,...
    for (ll i = 1; i <= n; i++)
        if (!p[i])
            p[i] = i;
    for (ll i = 1; i <= n; i++)
        cout << p[i] << (i != n ? ' ' : '\n');
}
```

20.2 统计 1-n 所有数字的各个位的和

```
vl pows(18);
unordered_map<ll, ll> memo;

// sum_digits from 1 to n
// (logn)^2
ll sd(ll n){
    if (n < 0)
        return 0;
    if (n < 10)
        return n * (n + 1) / 2;

    if (memo.count(n)){
        return memo[n];
    }
```

```
    string s = to_string(n);
    ll d = s.length();
    ll p10 = pows[d - 1];
    ll fd = s[0] - '0';
    ll rem = n % p10;

    ll res = 0;

    res += (d - 1) * 45 * pows[d - 2];
    res += (fd - 1) * fd / 2 * p10;
    res += (fd - 1) * sd(p10 - 1);
    res += fd * (rem + 1);
    res += sd(rem);

    return memo[n] = res;
}

void solve(){
    ll k;
    cin >> k;

    ll ans = 0, d = 1;
    while (true){
        ll num = 9 * pows[d - 1];
        num *= d;

        if (k > num){
            ll st = pows[d - 1];
            ll ed = pows[d] - 1;
            ans += sd(ed) - sd(st - 1);
            k -= num;
            d++;
        }
        else{
            ll con = (k - 1) / d;
            ll st = pows[d - 1];
            ll ed = st + con - 1;

            if (con > 0)
            {
                ans += sd(ed) - sd(st - 1);
            }

            ll rem = (k - 1) % d + 1;
            ll fin = ed + 1;
            string s = to_string(fin);
            for (ll i = 0; i < rem; ++i)
            {
                ans += s[i] - '0';
            }
            break;
        }
    }
    cout << ans << endl;
}
```

20.3 蒙特卡洛模拟

```
// a[l, r], cnt(a[i]) > |len/3|+ 1
// cnt(i) <= 2, i satisfies the above condition.
vl g[maxn];
mt19937 rd(chrono::high_resolution_clock::now().time_since_epoch().count());
void solve(){
    ll n, q;
    cin >> n >> q;

    vl a(n + 1);
    vl div;
```

```
    for (ll i = 0; i <= n; i++){
        g[i].clear();
    }

    for (ll i = 1; i <= n; i++){
        cin >> a[i];
        div.pb(a[i]);
    }

    sort(all(div));
    div.erase(unique(all(div)), div.end());

    for (ll i = 1; i <= n; i++){
        // compressed position
        ll cp = lower_bound(all(div), a[i]) - div.begin() + 1;
        g[cp].pb(i);
    }

    while (q--){
        ll l, r;
        cin >> l >> r;
        ll k = 40;
        const ll th = (r - l + 1) / 3;
        uniform_int_distribution<ll> grp(l, r); // [l, r]
        set<ll> res;

        while (k--)
        {
            ll radx = grp(rd);
            ll val = a[radx];
            ll cp = lower_bound(all(div), val) - div.begin() + 1;
            auto &pos = g[cp];
            // count times of val of range [l, r]
            ll cnt = upper_bound(all(pos), r) - lower_bound(all(pos), l);

            if (cnt > th)
            {
                res.insert(val);
            }
            if (res.size() >= 2)
            {
                break;
            }
        }

        if (res.empty())
        {
            cout << -1 << endl;
        }
        else
        {
            bool flag = true;
            for (auto const &val : res)
            {
                if (!flag)
                {
                    cout << " ";
                }
                cout << val;
                flag = false;
            }
            cout << endl;
        }
    }
}
```

图论

1. 并查集

```
ll root[maxn], size[maxn], st[maxn];
ll n, sets;

void init()
{
    for (ll i = 0; i < n; i++)
    {
        root[i] = i, size[i] = 1;
    }
}

ll find(ll x)
{
    // return root[x] != x ? root[x] = find(root[x]) : x;

    ll size = 0;
    while (x != root[x])
    {
        st[size++] = x;
        x = root[x];
    }

    while (size > 0)
    {
        root[st[--size]] = x;
    }

    return x;
}

void un(ll x, ll y)
{
    ll fx = find(x), fy = find(y);
    if (fx != fy)
    {
        if (size[fy] > size[fx])
        {
            root[fx] = fy;
            size[fy] += size[fx];
        }
        else
        {
            root[fy] = fx;
            size[fx] += size[fy];
        }
        sets--;
    }
}

// n couples, a total of 2n people (0 and 1 are a couple, 2 and 3 are a couple, and so on), sitting in seats numbered
0 to 2n-1, return minimum number of swaps required to make all couples sit side by side.
ll couples(vector<ll> &row)
{
    ll ans = 0;
    n = row.size() / 2;
    init();

    for (ll i = 0; i / 2 < n; i += 2)
    {
        un(row[i] / 2, row[i + 1] / 2);
    }

    for (ll x = 0; x < n; x++)
    {
        ll fx = root[x];
```

```
            ans += (fx != x ? 0 : (size[x] - 1));
        }

        return ans;
}

// 2d grid composed of '1's (land) and '0's (water), calculate the number of islands in the grid.
void build(ll n, ll m, vector<vector<char>> &grid)
{
        sets = 0;
        for (ll i = 0; i < n; i++)
        {
                for (ll j = 0; j < m; j++)
                {
                        if (grid[i][j] == '1')
                        {
                                ll x = i * m + j;
                                root[x] = x;
                                sets++;
                        }
                }
        }
}
ll lands(vector<vector<char>> &grid)
{
        ll n = grid.size(), m = grid[0].size();
        build(n, m, grid);

        for (ll i = 0; i < n; i++)
        {
                for (ll j = 0; j < m; j++)
                {
                        if (grid[i][j] == '1')
                        {
                                if (j > 0 && grid[i][j - 1] == '1')
                                {
                                        ll x = i * m + j;
                                        ll y = i * m + (j - 1);
                                        un(x, y);
                                }
                                if (i > 0 && grid[i - 1][j] == '1')
                                {
                                        ll x = i * m + j;
                                        ll y = (i - 1) * m + j;
                                        un(x, y);
                                }
                        }
                }
        }

        return sets;
}

ll root[maxn];
ll n, sets;

ll find(ll x)
{
        return root[x] != x ? root[x] = find(root[x]) : x;
}

void un(ll x, ll y)
{
        ll fx = find(x), fy = find(y);
        if (fx != fy)
        {
                root[fy] = fx;
```

```
            sets--;
        }
    }

    // n stones in a 2d plane, a stone can be removed if there is another stone in the same row or column.
    map<ll, ll> col, row;
    void build(ll n)
    {
        col.clear(), row.clear();
        sets = n;
        for (ll i = 0; i < n; i++)
        {
            root[i] = i;
        }
    }
    ll rem_ston(vector<vector<ll>> &stones)
    {
        //  can be proven that if stones in the same row or column are considered a set, the final answer is the total number of stones minus the number of sets.
        ll n = stones.size();
        build(n);

        for (ll i = 0; i < n; i++)
        {
            ll x = stones[i][0], y = stones[i][1];
            if (!row.count(x))
            {
                row[x] = i;
            }
            else
            {
                un(i, row[x]);
            }

            if (!col.count(y))
            {
                col[y] = i;
            }
            else
            {
                un(i, col[y]);
            }
        }

        return n - sets;
    }

    // n experts, expert 0 and fp initially know a secret at time 0, the meetings are given as a list meetings[i] = [xi, yi, timei]; the secret spreads through these meetings: if one expert in a meeting knows the secret at timei, the other expert will also learn it.
    bool secret[maxn];
    void build(ll n, ll fp)
    {
        for (ll i = 0; i < n; i++)
        {
            root[i] = i;
            secret[i] = false;
        }

        root[fp] = 0;
        secret[0] = true;
    }
    void un(ll x, ll y)
    {
        ll fx = find(x), fy = find(y);
        if (fx != fy)
        {
```

```
            root[fy] = fx;
            secret[fx] |= secret[fy];
        }
}
vector<ll> fin_peo(ll n, vector<vector<ll>> &meet, ll fp)
{
    build(n, fp);
    sort(meet.begin(), meet.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[2] < b[2]; });

    ll m = meet.size();
    for (ll l = 0, r; l < m;)
    {
        r = l;
        while (r + 1 < m && meet[l][2] == meet[r + 1][2])
        {
            r++;
        }

        // the same time
        for (ll i = l; i <= r; i++)
        {
            un(meet[i][0], meet[i][1]);
        }

        for (ll i = l; i <= r; i++)
        {
            ll f = find(meet[i][0]);
            if (!secret[f])
            {
                root[meet[i][0]] = meet[i][0];
                root[meet[i][1]] = meet[i][1];
            }
        }

        l = r + 1;
    }

    vector<ll> ans;
    for (ll i = 0; i < n; i++)
    {
        if (secret[find(i)])
        {
            ans.push_back(i);
        }
    }
    return ans;
}

// given a tree, n nodes, where each node has a value, find the number of "good paths.", good path is defined as a
path where :
// 1. the start and end nodes have the same value.
// 2. all intermediate nodes have a value less than or equal to the start / end node's value (path of a single node
is also considered a good path).
ll maxcnt[maxn];
void build(ll n)
{
    for (ll i = 0; i < n; i++)
    {
        root[i] = i;
        maxcnt[i] = 1;
    }
}
ll un(ll x, ll y, vector<ll> &vals)
{
    ll fx = find(x), fy = find(y), path = 0;
    if (vals[fx] > vals[fy])
```

```
    {
        root[fy] = fx;
    }
    else if (vals[fx] < vals[fy])
    {
        root[fx] = fy;
    }
    else
    {
        path = maxcnt[fx] * maxcnt[fy];
        root[fy] = fx;
        maxcnt[fx] += maxcnt[fy];
    }
    return path;
}
ll paths(vector<ll> &vals, vector<vector<ll>> &edges)
{
    ll n = vals.size();
    build(n);
    ll ans = n;
    sort(edges.begin(), edges.end(), [&vals](const vector<ll> &a, const vector<ll> &b)
         { return max(vals[a[0]], vals[a[1]]) < max(vals[b[0]], vals[b[1]]); });

    for (const vector<ll> &e : edges)
    {
        ans += un(e[0], e[1], vals);
    }
    return ans;
}

// given some initially infected nodes, an infection spreads along connections, remove one initially infected node
to minimize the final number of infected nodes, if there is a tie, return the node with the smallest index.
// cnts records how many nodes can be saved by deleting each source virus; infect[x] equals -1, means no virus;
greater than or equal to 0, represents the virus index; equals - 2, means two or more source viruses.
bool virus[maxn];
ll cnts[maxn], infect[maxn], size[maxn];
void un(ll x, ll y)
{
    ll fx = find(x), fy = find(y);
    if (fx != fy)
    {
        root[fy] = fx;
        size[fx] += size[fy];
    }
}
void build(ll n, vector<ll> &init)
{
    for (ll i = 0; i < n; i++)
    {
        virus[i] = false;
        cnts[i] = 0;
        infect[i] = -1;
        size[i] = 1;
        root[i] = i;
    }

    for (ll x : init)
    {
        virus[x] = true;
    }
}
ll spread(vector<vector<ll>> &graph, vector<ll> &init)
{
    ll n = graph.size();
    build(n, init);

    for (ll i = 0; i < n; i++)
```

```
    {
        for (ll j = 0; j < n; j++)
        {
            if (graph[i][j] && !virus[i] && !virus[j])
            {
                un(i, j);
            }
        }
    }

    for (ll sick : init)
    {
        ll neigh;
        for (ll i = 0; i < n; i++)
        {
            if (graph[sick][i] && sick != i && !virus[i])
            {
                neigh = i;
                ll fn = find(neigh);
                if (infect[fn] == -1)
                {
                    infect[fn] = sick;
                }
                else if (infect[fn] != -2 && infect[fn] != sick)
                {
                    infect[fn] = -2;
                }
            }
        }
    }

    for (ll i = 0; i < n; i++)
    {
        if (find(i) == i && !virus[i] && infect[i] >= 0)
        {
            cnts[infect[i]] += size[i];
        }
    }

    sort(init.begin(), init.end());
    ll max = cnts[init[0]], pos = init[0];

    for (ll v : init)
    {
        if (cnts[v] > max)
        {
            max = cnts[v];
            pos = v;
        }
    }

    return pos;
}
```

2. 图论基础

2.1 建图

```
const ll maxn = 11; // n + 1
const ll MAXM = 21; // m + 1 or 2m + 1
ll graph[maxn][maxn];
vector<vector<pair<ll, ll>>> adj(maxn); // vector<vector<ll>> adj, unweighted graph

void build(ll n, vector<vector<ll>> &edges)
{
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < n; j++)
        {
            graph[i][j] = 0;
        }
        adj[i].clear();
    }

    for (vector<ll> &e : edges)
    {
        graph[e[0]][e[1]] = e[2];
        adj[e[0]].push_back(make_pair(e[1], e[2]));
    }
}

// Linked Forward Star
// head array is indexed by vertex numbers, starting from 1 whose values are the indices of the head edges.
// nexe, to, and w arrays are all indexed by edge numbers whose values are respectively the index of the next edge
the endpoint of the current edge, and the weight of the current edge.
ll head[maxn], nexe[MAXM], to[MAXM], wg[MAXM];
ll cnt;
void add_edge(ll u, ll v, ll w)
{
    nexe[cnt] = head[u];
    to[cnt] = v;
    wg[cnt] = w;
    head[u] = cnt++;
}
void build(ll n, vector<vector<ll>> &edges)
{
    cnt = 1;
    memset(head, 0, sizeof(head));

    for (vector<ll> &e : edges)
    {
        ll u = e[0], v = e[1], w = e[2];
        add_edge(u, v, w);
    }
}
void print(ll n)
{
    for (ll i = 1; i <= n; i++)
    {
        for (ll eid = head[i]; eid != 0; eid = nexe[eid])
        {
            cout << i << "→" << to[eid] << ": " << wg[eid] << " | ";
        }
        cout << endl;
    }
}
```

2.2 拓扑排序

```
// topo-sort
vector<ll> topo(ll n, vector<vector<ll>> &pre)
{
    vector<ll> deg(n, 0), q(n);
    vector<vector<ll>> adj(n);
```

```
    for (vector<ll> &e : pre)
    {
        // e[1] → e[0]
        adj[e[1]].push_back(e[0]);
        deg[e[0]]++;
    }

    ll l = 0, r = 0;
    for (ll i = 0; i < n; i++)
    {
        deg[i] != 0 ?: q[r++] = i;
    }

    ll cnt = 0;
    while (l < r)
    {
        ll cur = q[l++];
        cnt++;
        for (ll next : adj[cur])
        {
            --deg[next] != 0 ?: q[r++] = next;
        }
    }

    vector<ll> null;
    return cnt != n ? null : q;
}

// lexicographically smallest topological sort.
ll head[maxn], nexe[MAXM], to[MAXM];
ll cnt;
void add_edge(ll u, ll v)
{
    nexe[cnt] = head[u];
    to[cnt] = v;
    head[u] = cnt++;
}
priority_queue<ll, vector<ll>, greater<ll>> pq;
vector<ll> deg(maxn, 0);
void build(vector<vector<ll>> &edges)
{
    cnt = 1;
    memset(head, 0, sizeof(head));

    for (vector<ll> &e : edges)
    {
        ll u = e[0], v = e[1];
        add_edge(u, v);
        deg[v]++;
    }
}
void topo_lex(ll n, vector<vector<ll>> &pre)
{
    build(pre);

    for (ll i = 1; i < n + 1; i++)
    {
        if (deg[i] == 0)
        {
            pq.push(i);
        }
    }

    while (!pq.empty())
    {
        ll cur = pq.top();
```

```
            pq.pop();
            cout << cur << " ";
            for (ll eid = head[cur]; eid != 0; eid = nexe[eid])
            {
                if (--deg[to[eid]] == 0)
                {
                    pq.push(to[eid]);
                }
            }
        }
    }
```

2.3 拓扑排序扩展

```
ll head[maxn], nexe[MAXM], to[MAXM];
ll q[maxn], deg[maxn], lines[maxn];
ll cnt;

void add_edge(ll u, ll v)
{
    nexe[cnt] = head[u];
    to[cnt] = v;
    head[u] = cnt++;
}

vector<vector<ll>> adj;
void build()
{
    cnt = 1;
    memset(head, 0, sizeof(head));
    memset(deg, 0, sizeof(deg));
    memset(lines, 0, sizeof(lines));

    for (vector<ll> &e : adj)
    {
        ll u = e[0], v = e[1];
        add_edge(u, v);
        deg[v]++;
    }
}
// count the number of food chains of a food web
ll prey(ll n, ll m){
    build();
    ll l = 0, r = 0;
    for (ll i = 1; i < n + 1; i++){
        if (deg[i] != 0){}
        else{
            q[r++] = i, lines[i] = 1;
        }
    }
    ll ans = 0;
    while (l < r){
        ll u = q[l++];
        if (head[u] != 0){
            for (ll eid = head[u]; eid != 0; eid = nexe[eid]){
                ll v = to[eid];
                --deg[v] != 0 ?: q[r++] = v;
                lines[v] = (lines[v] + lines[u]) % mod;
            }
        }
        else
        {
            ans = (ans + lines[u]) % mod;
        }
    }

    return ans;
}
```

3. 最小生成树

3.1 Kruskal

```
ll root[maxn];
ll n, m;
void init(){
    for (ll i = 1; i < n + 1; i++)
    {
        root[i] = i;
    }
}

ll find(ll x){
    return root[x] != x ? root[x] = find(root[x]) : x;
}

bool un(ll x, ll y)
{
    ll fx = find(x), fy = find(y);
    if (fx != fy)
    {
        root[fy] = fx;
        return true;
    }
    else
    {
        return false;
    }
}

vector<vector<ll>> adj(MAXM, vector<ll>(3));
// O (m * log m + n + m)
void kruskal()
{
    init();
    for (ll i = 0; i < m; i++)
    {
        cin >> adj[i][0] >> adj[i][1] >> adj[i][2];
    }
    sort(adj.begin(), adj.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[2] < b[2]; });

    ll ans = 0, cnt = 0;
    for (vector<ll> &edge : adj)
    {
        ll u = edge[0], v = edge[1], w = edge[2];
        if (un(u, v))
        {
            cnt++, ans += w;
        }
    }
}
```

3.2 Prim

```
void prim(){
    vector<vector<pair<ll, ll>>> graph(n + 1);

    for (ll i = 0; i < m; ++i){
        ll u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }

    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<pair<ll, ll>>> pq;
    vector<bool> vis(n + 1, false);

    ll cnt = 1;
```

```
    vis[1] = true;
    ll ans = 0;

    for (const auto &edge : graph[1])
    {
        ll v = edge.first;
        ll w = edge.second;
        pq.push({w, v});
    }

    while (!pq.empty())
    {
        pair<ll, ll> edge = pq.top();
        pq.pop();

        ll cost = edge.first;
        ll next = edge.second;

        if (!vis[next])
        {
            cnt++;
            vis[next] = true;
            ans += cost;

            for (const auto &ne : graph[next])
            {
                ll v = ne.first;
                ll w = ne.second;
                pq.push({w, v});
            }
        }
    }
}
```

3.3 最小生成树拓展

```
const ll maxn = 5001;
ll root[maxn];
ll n, cnt;
vector<vector<ll>> adj(maxn << 1, vector<ll>(3));
void init(){
    // include virtual source
    for (ll i = 0; i <= n; i++)
    {
        root[i] = i;
    }
    cnt = 0;
}
bool un(ll x, ll y){
    ll fx = find(x), fy = find(y);
    if (fx != fy){
        root[fy] = fx;
        return true;
    }
    else{
        return false;
    }
}
bool same_set(ll x, ll y){
    return find(x) != find(y) ? false : true;
}

// among several villages, there is a cost to build roads connecting them, and a cost to drill a well in each village, how should we plan the drilling of wells and the building of roads to ensure all villages have access to water at the minimum total cost?
ll cost(ll n, vector<ll> &wells, vector<vector<ll>> &pipes){
    init();
    for (ll i = 0; i < pipes.size(); i++, cnt++){
```

```
            adj[cnt][0] = pipes[i][0];
            adj[cnt][1] = pipes[i][1];
            adj[cnt][2] = pipes[i][2];
        }
        // suppose that there is an edge between the virtual source and each village v, and its weight is the cost to
drill a well in village v.
        for (ll i = 0; i < n; i++, cnt++){
            adj[cnt][0] = 0;
            adj[cnt][1] = i + 1;
            adj[cnt][2] = wells[i];
        }

        sort(adj.begin(), adj.begin() + cnt, [](const vector<ll> &a, const vector<ll> &b)
             { return a[2] < b[2]; });

        ll ans = 0;
        for (ll i = 0; i < cnt; i++){
            vector<ll> &edge = adj[i];
            ll u = edge[0], v = edge[1], w = edge[2];
            if (un(u, v))
            {
                ans += w;
            }
        }
        return ans;
}

// some queries [u, v, w] on an undirected graph, return whether there exists a path from u to v where all edge
weights on the path are less than w.
vector<bool> dist(ll n, vector<vector<ll>> &adj, vector<vector<ll>> &queries){
        ll m = adj.size(), k = queries.size();
        vector<vector<ll>> q(k, vector<ll>(4));
        for (ll i = 0; i < k; i++){
            q[i][0] = queries[i][0];
            q[i][1] = queries[i][1];
            q[i][2] = queries[i][2];
            q[i][3] = i;
        }
        sort(q.begin(), q.end(), [](const vector<ll> &a, const vector<ll> &b)
             { return a[2] < b[2]; });

        sort(adj.begin(), adj.end(), [](const vector<ll> &a, const vector<ll> &b)
             { return a[2] < b[2]; });

        vector<bool> ans(k);
        init();

        for (ll i = 0, j = 0; i < k; i++){
            while (j < m && adj[j][2] < q[i][2]){
                vector<ll> &edge = adj[j++];
                ll u = edge[0], v = edge[1], w = edge[2];
                un(u, v);
            }

            ans[q[i][3]] = same_set(q[i][0], q[i][1]);
        }

        return ans;
}

// connected, undirected, weighted graph, select the minimum number of roads to keep all intersections connected,
you must also minimize the maximum weight among the selected roads.
// proven that mst is the mbst that this problem asks for, but the reverse is not true.
```

4. BFS
5.1 BFS 基础

```
// distance of the farthest ocean cell ('0') from land ('1').
ll dist(vector<vector<ll>> &grid)
{
    ll n = grid.size(), m = grid[0].size();
    vector<vector<bool>> vis(n, vector<bool>(m, false));
    vector<pair<ll, ll>> q(n * m);
    ll move[] = {-1, 0, 1, 0, -1};

    ll l = 0, r = 0, seas = 0;
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] != 1)
            {
                seas++;
            }
            else
            {
                vis[i][j] = true;
                q[r].first = i;
                q[r++].second = j;
            }
        }
    }

    ll level = 0, size;
    while (l < r)
    {
        level++;
        size = r - l;
        for (ll i = 0, tx, ty; i < size; i++)
        {
            pair<ll, ll> cur = q[l++];
            for (ll j = 0; j < 4; j++)
            {
                tx = cur.first + move[j];
                ty = cur.second + move[j + 1];
                if (tx >= 0 && tx < n && ty >= 0 && ty < m && !vis[tx][ty])
                {
                    vis[tx][ty] = true;
                    q[r].first = tx;
                    q[r++].second = ty;
                }
            }
        }
    }

    return level - 1;
}

// target string and a list of sticker words, find the minimum number of stickers required to spell the target string,
you can use letters from the stickers, and each sticker is available in unlimited supply.
string str_sub(const string &cur, const string &s)
{
    unordered_map<char, ll> s_cnt;
    for (char c : s)
    {
        s_cnt[c]++;
    }

    string result = "";

    for (char c : cur)
    {
```

```
            if (s_cnt.count(c) && s_cnt[c] > 0)
            {
                s_cnt[c]--;
            }
            else
            {
                result += c;
            }
        }

        return result;
}
ll stickers(vector<string> &st, string tar)
{
        // branch pruning
        vector<vector<string>> graph(26);
        for (string &s : st)
        {
            sort(s.begin(), s.end());
            for (ll i = 0; i < s.length(); i++)
            {
                if (i < 1 || s[i] != s[i - 1])
                {
                    graph[s[i] - 'a'].push_back(s);
                }
            }
        }

        set<string> vis;
        sort(tar.begin(), tar.end());
        vector<string> q(1001);
        ll l = 0, r = 0, level = 1, size;
        q[r++] = tar;
        vis.insert(tar);
        while (l < r)
        {
            size = r - l;
            for (ll i = 0; i < size; i++)
            {
                string cur = q[l++];
                for (string &s : graph[cur[0] - 'a'])
                {
                    string cur_s = str_sub(cur, s);
                    if (cur_s.empty())
                    {
                        return level;
                    }
                    else if (!vis.count(cur_s))
                    {
                        vis.insert(cur_s);
                        q[r++] = cur_s;
                    }
                }
            }
            level++;
        }

        return -1;
}
```

5.2 0-1 BFS

```
const ll inf = 99999;
// m x n grid of empty cells (0) and obstacles (1), you can move to adjacent empty cells, return the minimum number
of obstacles to remove to clear a path from the top-left to the bottom-right corner.
ll obstacles(vector<vector<ll>> &grid)
{
        //   converting moves to obstacles into edges 1, and empty spaces into edges 0, making it a 0-1 bfs problem.
```

```
    ll move[] = {-1, 0, 1, 0, -1};
    ll n = grid.size(), m = grid[0].size();
    vector<vector<ll>> dist(n, vector<ll>(m, inf));

    deque<pair<ll, ll>> dq;
    dq.push_back({0, 0});
    dist[0][0] = 0;

    while (!dq.empty())
    {
        pair<ll, ll> cur = dq.front();
        dq.pop_front();
        ll x = cur.first, y = cur.second;

        if (x != n - 1 || y != m - 1)
        {
            for (ll i = 0; i < 4; i++)
            {
                ll tx = x + move[i], ty = y + move[i + 1];
                if (tx >= 0 && tx < n && ty >= 0 && ty < m)
                {
                    if (dist[tx][ty] > dist[x][y] + grid[tx][ty])
                    {
                        dist[tx][ty] = dist[x][y] + grid[tx][ty];
                        if (grid[tx][ty] != 0)
                        {
                            dq.push_back({tx, ty});
                        }
                        else
                        {
                            dq.push_front({tx, ty});
                        }
                    }
                }
            }
        }
        else
        {
            return dist[x][y];
        }
    }

    return -1;
}

// 2d trapping rain water problem
struct comp
{
    bool operator()(const vector<ll> &a, const vector<ll> &b) const
    {
        return a[2] > b[2];
    }
};
ll trap_rain(vector<vector<ll>> &h)
{
    ll move[] = {-1, 0, 1, 0, -1};
    ll n = h.size(), m = h[0].size();
    priority_queue<vector<ll>, vector<vector<ll>>, comp> pq;
    vector<vector<bool>> vis(n, vector<bool>(m, false));

    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (i != 0 && i != n - 1 && j != 0 && j != m - 1)
            {
                vis[i][j] = false;
```

```
                }
                else
                {
                    pq.push({i, j, h[i][j]});
                    vis[i][j] = true;
                }
            }
        }

        ll ans = 0;
        while (!pq.empty())
        {
            vector<ll> cur = pq.top();
            pq.pop();
            ll x = cur[0], y = cur[1], line = cur[2];

            for (ll i = 0; i < 4; i++)
            {
                ll tx = x + move[i], ty = y + move[i + 1];
                if (tx >= 0 && tx < n && ty >= 0 && ty < m && !vis[tx][ty])
                {
                    vis[tx][ty] = true;
                    pq.push({tx, ty, max(line, h[tx][ty])});
                }
            }

            ans += (line - h[x][y]);
        }
        return ans;
}

// using words from a word list as bridges, starting from ben, change one character at a time to become end, and
find all shortest paths.
unordered_set<string> cur_level, next_level;
unordered_map<string, vector<string>> g;
vector<string> path;
vector<vector<string>> ans;

bool bfs(unordered_set<string> &dict, string beg, string end)
{
    bool flag = false;
    cur_level.insert(beg);

    while (!cur_level.empty())
    {
        // branch pruning
        for (const string &word : cur_level)
        {
            dict.erase(word);
        }
        for (string s : cur_level)
        {
            for (ll i = 0; i < s.length(); i++)
            {
                char old = s[i];
                for (char ch = 'a'; ch <= 'z'; ch++)
                {
                    string str = s;
                    str[i] = ch;
                    if (dict.count(str) && str != s)
                    {
                        if (str == end)
                        {
                            flag = true;
                        }
                        g[str].push_back(s);
                        next_level.insert(str);
```

```
                    }
                }
            }
        }

        if (flag)
        {
            return true;
        }
        else
        {
            cur_level = next_level;
            next_level.clear();
        }
    }

    return false;
}
void dfs(unordered_set<string> &dict, string cur, string tar)
{
    path.push_back(cur);
    if (cur == tar)
    {
        vector<string> tmp(path);
        reverse(tmp.begin(), tmp.end());
        ans.push_back(tmp);
    }
    else if (g.count(cur))
    {
        for (string s : g[cur])
        {
            dfs(dict, s, tar);
        }
    }
    path.pop_back();
}
vector<vector<string>> ladders(string beg, string end, vector<string> &list)
{
    unordered_set<string> dict(list.begin(), list.end());

    if (!dict.count(end))
    {
        return ans;
    }

    if (bfs(dict, beg, end))
    {
        dfs(dict, end, beg);
    }

    return ans;
}
```

5.3 双向广搜

```
// using words from a word list as bridges, starting from ben, change one character at a time to become end,
return the length of one shortest path.
unordered_set<string> lar_level, sma_level, next_level;
unordered_map<string, vector<string>> g;

ll bfs(unordered_set<string> &dict, string beg, string end)
{
    sma_level.insert(beg);
    lar_level.insert(end);

    ll len = 2;
    while (!sma_level.empty())
    {
```

```
        for (string s : sma_level)
        {
            for (ll i = 0; i < s.length(); i++)
            {
                char old = s[i];
                for (char ch = 'a'; ch <= 'z'; ch++)
                {
                    string str = s;
                    str[i] = ch;
                    if (str != s)
                    {
                        if (lar_level.count(str))
                        {
                            return len;
                        }
                        if (dict.count(str))
                        {
                            dict.erase(str);
                            next_level.insert(str);
                        }
                    }
                }
            }
        }
        len++;
        if (lar_level.size() < next_level.size())
        {
            sma_level = lar_level;
            lar_level = next_level;
        }
        else
        {
            sma_level = next_level;
        }
        next_level.clear();
    }

    return 0;
}
ll ladders(string beg, string end, vector<string> &list)
{
    unordered_set<string> dict(list.begin(), list.end());

    if (!dict.count(end))
    {
        return 0;
    }

    return bfs(dict, beg, end);
}

// n items with sizes v[i] and a backpack with capacity w, return the number of subsets of items whose total size
does not exceed w (including the subset with a total size of 0)
ll n, w;
vector<ll> arr(40), lsum(1 << 20), rsum(1 << 20);

ll f(ll s, ll e, ll pak, ll w, vector<ll> &ans, ll pos)
{
    if (pak > w)
    {
        return pos;
    }

    if (s != e)
    {
        pos = f(s + 1, e, pak, w, ans, pos);
        pos = f(s + 1, e, pak + arr[s], w, ans, pos);
```

```
    }
    else
    {
        ans[pos++] = pak;
    }

    return pos;
}

ll compute()
{
    // n = 30, and v[i] = 10^9 is so large that dynamic programming is not feasible, we can split the items llo two
halves, brute-force all subsets for each half, and then combine the results.
    ll lsize = f(0, n >> 1, 0, w, lsum, 0); // [0, n/2)
    ll rsize = f(n >> 1, n, 0, w, rsum, 0); // [n/2, n)

    sort(lsum.begin(), lsum.begin() + lsize);
    sort(rsum.begin(), rsum.begin() + rsize);

    ll ans = 0;
    for (ll i = lsize - 1, j = 0; i >= 0; i--)
    {
        while (j < rsize && lsum[i] + rsum[j] - 1 < w)
        {
            j++;
        }
        ans += j;
    }
    return ans;
}

// get a subsequence whose sum minimizes abs(sum - goal), return this minimum absolute difference.
ll pos;
vector<ll> lsum, rsum;
void f(ll s, ll e, ll sum, vector<ll> &ans, vector<ll> &nums)
{
    if (s != e)
    {
        ll ns = s + 1;
        while (ns < e && nums[s] == nums[ns])
        {
            ns++;
        }

        for (ll k = 0; k < ns - s + 1; k++)
        {
            f(ns, e, sum + k * nums[s], ans, nums);
        }
    }
    else
    {
        ans[pos++] = sum;
    }
}
ll min_abs(vector<ll> &nums, ll goal)
{
    ll n = nums.size();

    ll min_sum = 0, max_sum = 0;
    for (ll x : nums)
    {
        x > 0 ? max_sum += x : min_sum += x;
    }
    if (max_sum < goal)
        return (ll)(goal - max_sum);
    if (min_sum > goal)
        return (ll)(min_sum - goal);
```

```
        ll left_n = n / 2;
        ll right_n = n - left_n;
        lsum.resize(1 << left_n);
        rsum.resize(1 << right_n);

        sort(nums.begin(), nums.end());

        pos = 0;
        f(0, left_n, 0, lsum, nums);
        ll lsize = pos;

        pos = 0;
        f(left_n, n, 0, rsum, nums);
        ll rsize = pos;

        sort(lsum.begin(), lsum.begin() + lsize);
        sort(rsum.begin(), rsum.begin() + rsize);

        ll ans = abs(goal);
        for (ll i = 0, j = rsize - 1; i < lsize; i++)
        {
            while (j > 0 && abs(goal - lsum[i] - rsum[j]) >= abs(goal - lsum[i] - rsum[j - 1]))
            {
                j--;
            }
            ans = std::min(ans, abs(goal - lsum[i] - rsum[j]));
        }
        return ans;
    }
```

5. 最短路问题
5.1 Dijkstra 算法

```
vector<vector<pair<ll, ll>>> adj(101);
// without negative edge weights.
// leetcode.cn/problems/network-delay-time/submissions/655555774/
struct comp
{
    bool operator()(const pair<ll, ll> &a, const pair<ll, ll> &b) const
    {
        return a.second > b.second;
    }
};
ll network(vector<vector<ll>> &times, ll n, ll s)
{
    for (vector<ll> &eg : times)
    {
        adj[eg[0]].push_back({eg[1], eg[2]});
    }
    vector<ll> dist(n + 1, inf), vis(n + 1, false);
    dist[s] = 0;

    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, comp> pq;
    pq.push({s, 0});

    while (!pq.empty())
    {
        pair<ll, ll> cur = pq.top();
        ll u = cur.first;
        pq.pop();
        if (vis[u])
        {
            continue;
        }
        vis[u] = true;

        for (pair<ll, ll> &eg : adj[u])
        {
            ll v = eg.first, w = eg.second;
            if (!vis[v] && dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                pq.push({v, dist[v]});
            }
        }
    }

    ll ans = LLONG_MIN;
    for (ll i = 1; i < n + 1; i++)
    {
        if (dist[i] != inf)
        {
            ans = min(ans, dist[i]);
        }
        else
        {
            return -1;
        }
    }
    return ans;
}

// m * logm → m * logn
const ll maxn = 100001;
const ll maxm = 200001;
ll head[maxn], nexe[maxm], to[maxm], weight[maxm];
ll cnt;
```

```
// the reverse index table stores the indices of nodes, -1 means never entered the heap, -2 means has already been popped from the heap.
ll heap[maxn], where[maxn];
ll hsize;

ll dist[maxn];
ll n, m, s;

void build()
{
    cnt = 1;
    hsize = 0;
    for (ll i = 1; i < n + 1; ++i)
    {
        head[i] = 0;
        where[i] = -1;
        dist[i] = LLONG_MAX;
    }
}

void add_edge(ll u, ll v, ll w)
{
    nexe[cnt] = head[u];
    to[cnt] = v;
    weight[cnt] = w;
    head[u] = cnt++;
}

void swap(ll i, ll j)
{
    ll tmp_node = heap[i];
    heap[i] = heap[j];
    heap[j] = tmp_node;
    where[heap[i]] = i;
    where[heap[j]] = j;
}

void hinsert(ll i)
{
    while (dist[heap[i]] < dist[heap[(i - 1) / 2]])
    {
        swap(i, (i - 1) / 2);
        i = (i - 1) / 2;
    }
}

// the shortest path problem, the value of a node in the min-heap will only be adjusted to a smaller value. Therefore, at this time, only a 'heapify' operation is needed.
void heapify(ll i)
{
    ll l = i * 2 + 1;
    while (l < hsize)
    {
        ll best = l + 1 < hsize && dist[heap[l + 1]] < dist[heap[l]] ? l + 1 : l;
        best = dist[heap[best]] < dist[heap[i]] ? best : i;
        if (best == i)
        {
            break;
        }
        swap(best, i);
        i = best;
        l = i * 2 + 1;
    }
}

ll pop()
{
```

```
    ll ans = heap[0];
    swap(0, --hsize);
    heapify(0);
    where[ans] = -2;
    return ans;
}

void update(ll v, ll w)
{
    if (where[v] > -1)
    {
        dist[v] = std::min(dist[v], w);
        hinsert(where[v]);
    }
    else if (where[v] > -2)
    {
        heap[hsize] = v;
        where[v] = hsize++;
        dist[v] = w;
        hinsert(where[v]);
    }
}

void dijkstra()
{
    update(s, 0);
    while (hsize != 0)
    {
        ll v = pop();
        for (ll eid = head[v]; eid > 0; eid = nexe[eid])
        {
            update(to[eid], dist[v] + weight[eid]);
        }
    }
}

// https://leetcode.cn/problems/path-with-minimum-effort/
// At its core, this is a shortest path problem; it's just that the calculation of the path's distance needs to be
modified.
struct comp
{
    bool operator()(const vector<ll> &a, const vector<ll> &b) const
    {
        return a[2] > b[2];
    }
};
// nm * lognm
ll min_eff(vector<vector<ll>> &h)
{
    ll n = h.size(), m = h[0].size();
    priority_queue<vector<ll>, vector<vector<ll>>, comp> pq;
    vector<vector<ll>> dist(n, vector<ll>(m, inf));
    vector<vector<bool>> vis(n, vector<bool>(m, false));

    dist[0][0] = 0;
    pq.push({0, 0, 0});

    while (!pq.empty())
    {
        vector<ll> cur = pq.top();
        ll x = cur[0], y = cur[1], d = cur[2];
        pq.pop();

        if (vis[x][y])
        {
            continue;
        }
```

```
            vis[x][y] = true;

            if (x != n - 1 || y != m - 1)
            {
                ll move[] = {-1, 0, 1, 0, -1};
                for (ll i = 0; i < 4; i++)
                {
                    ll tx = x + move[i], ty = y + move[i + 1];
                    if (tx >= 0 && tx < n && ty >= 0 && ty < m && !vis[tx][ty])
                    {
                        ll nd = max(d, abs(h[x][y] - h[tx][ty]));
                        if (dist[tx][ty] > nd)
                        {
                            dist[tx][ty] = nd;
                            pq.push({tx, ty, nd});
                        }
                    }
                }
            }
            else
            {
                return d;
            }
        }

        return -1;
}
```

5.2 分层最短路问题

// grid containing empty rooms, walls, a start point '@', lowercase letter keys, and their corresponding uppercase locks, find the minimum number of moves to collect all keys starting from the initial position; you can pick up keys, but you can only pass through a lock if you have the matching key, return -1 if it is impossible to collect all keys.

```
const ll maxn = 31, maxm = 31, maxk = 1 << 6;
bool vis[maxn][maxm][maxk];
ll q[maxn * maxm * maxk][3];
ll l, r, n, m, key;
void build(vector<string> &grid)
{
    memset(vis, 0, sizeof(vis));
    l = 0, r = 0, key = 0;
    n = grid.size(), m = grid[0].size();

    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (grid[i][j] == '@')
            {
                q[r][0] = i;
                q[r][1] = j;
                q[r++][2] = 0; // 000000
            }
            if (grid[i][j] >= 'a' && grid[i][j] < 'g')
            {
                key |= 1 << (grid[i][j] - 'a');
            }
        }
    }
}
ll sho_keys(vector<string> &grid)
{
    // . . @ #
    // b B . a
    build(grid);
    ll move[] = {-1, 0, 1, 0, -1};
    ll level = 1;
    while (l < r)
```

```
    {
        ll size = r - l;
        for (ll k = 0; k < size; k++)
        {
            ll x = q[l][0], y = q[l][1];
            ll sta = q[l++][2];
            for (ll i = 0; i < 4; i++)
            {
                ll tx = x + move[i], ty = y + move[i + 1], nsta = sta;
                if (tx < 0 || tx >= n || ty < 0 || ty >= m || grid[tx][ty] == '#')
                {
                    continue;
                }
                // check key
                if ((grid[tx][ty] >= 'A' && grid[tx][ty] < 'G') && ((nsta & (1 << (grid[tx][ty] - 'A'))) == 0))
                {
                    continue;
                }

                if (grid[tx][ty] >= 'a' && grid[tx][ty] < 'g')
                {
                    nsta |= 1 << (grid[tx][ty] - 'a');
                }

                if (nsta != key)
                {
                    if (!vis[tx][ty][nsta])
                    {
                        vis[tx][ty][nsta] = true;
                        q[r][0] = tx;
                        q[r][1] = ty;
                        q[r++][2] = nsta;
                    }
                }
                else
                {
                    return level;
                }
            }
        }
        level++;
    }
    return -1;
}

// electric scooter has a max power of cnt, there are n cities on a map with bidirectional roads, starting with 0
power, each city has a different time cost to charge, find the minimum total time required to travel from a starting
city to an ending city, including both travel and charging time.
struct comp
{
    bool operator()(const vector<ll> &a, const vector<ll> &b) const
    {
        return a[2] > b[2];
    }
};
// O((n+m)·cnt·log(n·cnt))
ll electric(vector<vector<ll>> &paths, ll cnt, ll start, ll end, vector<ll> &charge)
{
    ll n = charge.size();
    vector<vector<pair<ll, ll>>> adj(101);
    for (vector<ll> &path : paths)
    {
        ll u = path[0], v = path[1], w = path[2];
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }
```

```
    ll dist[101][101]; // dist[n][cnt+1]
    bool vis[101][101];
    memset(vis, 0, sizeof(vis));
    memset(dist, inf, sizeof(dist));
    dist[start][0] = 0;

    priority_queue<vector<ll>, vector<vector<ll>>, comp> pq;
    pq.push({start, 0, 0});

    while (!pq.empty())
    {
        vector<ll> ct = pq.top();
        ll pos = ct[0], pow = ct[1], cost = ct[2];
        pq.pop();

        if (vis[pos][pow])
        {
            continue;
        }
        vis[pos][pow] = true;

        if (pos != end)
        {
            if (pow < cnt)
            {
                if (!vis[pos][pow + 1] && dist[pos][pow + 1] > cost + charge[pos])
                {
                    dist[pos][pow + 1] = cost + charge[pos];
                    pq.push({pos, pow + 1, dist[pos][pow + 1]});
                }
            }
            for (pair<ll, ll> &eg : adj[pos])
            {
                ll npos = eg.first, dis = eg.second;
                ll rpow = pow - dis;
                if (rpow >= 0 && !vis[npos][rpow] && dist[npos][rpow] > cost + dis)
                {
                    dist[npos][rpow] = cost + dis;
                    pq.push({npos, rpow, dist[npos][rpow]});
                }
            }
        }
        else
        {
            return cost;
        }
    }
    return -1;
}
```

5.3 floyd （多源最短距离）

```
ll dist[101][101];
void floyd()
{
    for (ll k = 0; k < n; ++k)
    {
        for (ll i = 0; i < n; ++i)
        {
            for (ll j = 0; 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]);
                }
            }
        }
    }
```

```
}
```

5.4 bellman 和 spfa 算法

```
// Bellman-Ford O(m*n)
ll price(ll n, vector<vector<ll>> &flights, ll src, ll dst, ll k)
{
    vector<ll> cur(n, inf);
    cur[src] = 0;
    for (ll i = 0; i < k + 1; i++)
    {
        vector<ll> next(cur);
        for (vector<ll> &eg : flights)
        {
            ll u = eg[0], v = eg[1], w = eg[2];
            if (cur[u] != inf)
            {
                next[v] = min(next[v], cur[u] + w);
            }
        }
        cur = next;
    }
    return cur[dst] != inf ? cur[dst] : -1;
}

// SPFA
const ll maxn = 2001, maxm = 6001, maxq = 4000001;
ll head[maxn], nexe[maxm], to[maxm], wg[maxm];
ll cnt;
void add_edge(ll u, ll v, ll w)
{
    nexe[cnt] = head[u];
    to[cnt] = v;
    wg[cnt] = w;
    head[u] = cnt++;
}

ll dist[maxn], u_cnt[maxn], q[maxq];
bool enter[maxn];
ll l, r;
void build(ll n)
{
    cnt = 1;
    memset(head, 0, sizeof(head));

    l = 0, r = 0;
    for (int i = 0; i < n + 1; ++i)
    {
        dist[i] = inf;
    }
    memset(u_cnt, 0, sizeof(u_cnt));
    memset(enter, 0, sizeof(enter));
}

bool spfa(ll s, ll n)
{
    dist[s] = 0;
    u_cnt[s]++;
    q[r++] = s;
    enter[s] = true;
    while (l < r)
    {
        ll u = q[l++];
        enter[u] = false;
        for (ll eid = head[u], v, w; eid > 0; eid = nexe[eid])
        {
            v = to[eid], w = wg[eid];
            if (dist[u] != inf && dist[v] > dist[u] + w)
```

```
                {
                    dist[v] = dist[u] + w;
                    if (!enter[v])
                    {
                        if (++u_cnt[v] >= n)
                        {
                            return true;
                        }
                        q[r++] = v;
                        enter[v] = true;
                    }
                }
            }
        }
        return false;
}
```

贪心算法

1. 贪心-01

```
void swap(vector<string> &strs, ll i, ll j)
{
    string tmp = strs[i];
    strs[i] = strs[j];
    strs[j] = tmp;
}

void f(vector<string> &strs, ll i, vector<string> &ans)
{
    if (i != strs.size())
    {
        for (ll j = i; j < strs.size(); j++)
        {
            swap(strs, i, j);
            f(strs, i + 1, ans);
            swap(strs, i, j);
        }
    }
    else
    {
        stringstream ss;
        for (const string &s : strs)
        {
            ss << s;
        }
        ans.push_back(ss.str());
    }
}

string bao(vector<string> strs)
{
    vector<string> ans;
    f(strs, 0, ans);
    sort(ans.begin(), ans.end());
    return ans[0];
}

string bea(vector<string> &strs)
{
    sort(strs.begin(), strs.end(), [](const string &a, const string &b)
         { return a + b < b + a; });
    stringstream ss;
    for (ll i = 0; i < strs.size(); i++)
    {
        ss << strs[i];
    }
    return ss.str();
}

// random string
string rand_str(ll m, ll v)
{
    ll len = (rand() % m) + 1;
    string ans(len, ' ');
    for (ll i = 0; i < len; i++)
    {
        ans[i] = (char)('a' + (rand() % v));
    }
    return ans;
}

vector<string> rand_str_arr(ll n, ll m, ll v)
{
    ll size = (rand() % n) + 1;
    vector<string> ans(size);
```

```
    for (ll i = 0; i < ans.size(); i++)
    {
        ans[i] = rand_str(m, v);
    }
    return ans;
}

signed main()
{
    srand(time(0));
    ll n = 8; // number
    ll m = 5; // length
    ll v = 4; // kind
    ll times = 2001;
    cout << "Test Start!" << endl;
    for (ll i = 1; i < times; i++)
    {
        vector<string> strs = rand_str_arr(n, m, v);
        vector<string> sbao = strs;
        vector<string> sbea = strs;
        string rbao = bao(sbao);
        string rbea = bea(sbea);
        if (rbao != rbea)
        {
            cout << "error!" << endl;
            cout << "bao:" << rbao << ", ele:" << rbea << endl;
        }
    }
    cout << "Test Over!" << endl;

    return 0;
}

// array of non-negative integers, rearrange the numbers to form the largest possible integer, return the result as
a string.
string larg_num(vector<ll> &nums)
{
    ll n = nums.size();
    vector<string> strs(n);
    for (ll i = 0; i < n; i++)
    {
        strs[i] = to_string(nums[i]);
    }
    sort(strs.begin(), strs.end(), [](const string &a, const string &b)
         { return b + a < a + b; });

    if (strs[0] == "0")
    {
        return "0";
    }
    stringstream ss;
    for (const string &s : strs)
    {
        ss << s;
    }
    return ss.str();
}

// company plans to interview 2n people, costs[i] = [ac, bc] represents the costs for i to fly to city a/b, return the
minimum cost to send n people to a and n people to b.
ll city_cost(vector<vector<ll>> &costs)
{
    ll n = costs.size();
    ll diff[n];

    ll sum = 0;
    for (ll i = 0; i < n; i++)
```

```
    {
        diff[i] = costs[i][1] - costs[i][0];
        sum += costs[i][0];
    }

    sort(diff, diff + n);
    ll m = n / 2;
    for (ll i = 0; i < m; i++)
    {
        sum += diff[i];
    }

    return sum;
}

// n oranges, each day you can eat 1, n/2 (n % 2 = 0), or 2n/3 (n % 3) oranges,  find the minimum number of days
to eat all the oranges.
unordered_map<ll, ll> dp;
ll days(ll n)
{
    if (n < 2)
    {
        return n;
    }
    if (dp.count(n))
    {
        return dp[n];
    }

    ll ans = min(n % 2 + 1 + days(n / 2), n % 3 + 1 + days(n / 3));
    dp[n] = ans;
    return ans;
}

// maximum line segment overlap problem
ll minMeetingRooms(vector<vector<ll>> &mt)
{
    if (mt.empty())
    {
        return 0;
    }

    sort(mt.begin(), mt.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[0] < b[0]; });

    priority_queue<ll, vector<ll>, greater<ll>> etime;

    ll max = 0;
    for (const auto &meeting : mt)
    {
        ll st = meeting[0];
        ll et = meeting[1];

        while (!etime.empty() && etime.top() <= st)
        {
            etime.pop();
        }

        etime.push(et);
        max = std::max(max, (ll)etime.size());
    }

    return max;
}

// courses, each with a duration and a deadline, you cannot take multiple courses at the same time, find the
maximum number of courses you can take.
```

```
ll schedule(vector<vector<ll>> &courses)
{
    sort(courses.begin(), courses.end(), [](const vector<ll> &a, const vector<ll> &b)
        { return a[1] < b[1]; });
    priority_queue<ll> pq;
    ll time = 0;
    for (vector<ll> &cour : courses)
    {
        if (time + cour[0] > cour[1])
        {
            // cost substitution, one course for one course.
            if (!pq.empty() && pq.top() > cour[0])
            {
                time -= (pq.top() - cour[0]);
                pq.pop();
                pq.push(cour[0]);
            }
        }
        else
        {
            time += cour[0];
            pq.push(cour[0]);
        }
    }
    return pq.size();
}

// give some sticks with different lengths, the cost to connect two sticks of lengths x and y equals x + y, find the
minimum total cost to connect all sticks into a single stick.
ll pop_top(priority_queue<ll, vector<ll>, greater<ll>> &pq)
{
    if (pq.empty())
    {
        return 0;
    }
    ll top_val = pq.top();
    pq.pop();
    return top_val;
}
ll hafuman(vector<ll> &sticks)
{
    priority_queue<ll, vector<ll>, greater<ll>> pq;
    for (ll s : sticks)
    {
        pq.push(s);
    }
    ll sum = 0, cur = 0;
    while (pq.size() > 1)
    {
        cur = pop_top(pq) + pop_top(pq);
        sum += cur;
        pq.push(cur);
    }
    return sum;
}
```

2. 贪心-02

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

```
        a = (a + a) % mod;
        b >>= 1;
    }
    return ans;
}
ll qpow(ll x, ll n, ll mod)
{
    ll ans = 1;
    while (n > 0)
    {
        if (n & 1)
        {
            ans = mul(ans, x, mod);
        }
        x = mul(x, x, mod);
        n >>= 1;
    }
    return ans;
}

// length bamboo_len, cut it into several pieces of positive integer length, find the maximum product of the lengths of these pieces.
ll bamboo(ll len)
{
    if (len < 4)
    {
        return len - 1;
    }
    // n = 4    →  2 2
    // n = 5    →  3 2
    // n = 6    →  3 3
    // n = 7    →  3 2 2
    // n = 8    →  3 3 2
    // n = 9    →  3 3 3
    // n = 10   →  3 3 2 2
    // n = 11   →  3 3 3 2
    ll tail = (len % 3 != 0 ? (len % 3 != 1 ? 2 : 4) : 1);
    ll p = (tail != 1 ? (len - tail) : len) / 3;

    return (qpow(3, p, mod) * tail % mod);
}

// n must be partitioned into k parts, find the maximum possible product of these parts; n, k up to 10^12.
ll f(ll rest, ll k)
{
    if (k == 1)
    {
        return rest;
    }
    ll ans = LLONG_MIN;
    for (ll cur = 1; cur - 1 < rest && (rest - cur) >= (k - 1); cur++)
    {
        ll curAns = cur * f(rest - cur, k - 1);
        ans = max(ans, curAns);
    }
    return ans;
}

ll bao(ll n, ll k)
{
    return f(n, k);
}

ll bea(ll n, ll k)
{
    ll a = n / k;
    ll b = n % k;
```

```
    return (ll)((qpow(a + 1, b, mod) * qpow(a, k - b, mod)) % mod);
}

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    random_device rd;
    mt19937 gen(rd());

    ll N = 30;
    ll times = 2001;
    cout << "Test Start!" << endl;
    for (int i = 1; i < times; i++)
    {
        uniform_int_distribution<> n_dist(1, N);
        ll n = n_dist(gen);
        uniform_int_distribution<> k_dist(1, n);
        ll k = k_dist(gen);

        if (bao(n, k) != bea(n, k))
        {
            cout << "error!" << endl;
            // cout << "n = " << n << ", k = " << k << endl;
            // cout << "bao(n, k) = " << bao(n, k) << ", bea(n, k) = " << bea(n, k) << endl;
            break;
        }
    }
    cout << "Test Over!" << endl;

    return 0;
}

// maximum activity selection problem, sorting activities by their end times.
ll f(vector<vector<ll>> &mt, ll n, ll i)
{
    ll ans = 0;
    if (i != n)
    {
        for (ll j = i; j < n; j++)
        {
            swap(mt[i], mt[j]);
            ans = max(ans, f(mt, n, i + 1));
            swap(mt[i], mt[j]);
        }
    }
    else
    {
        ll curAns = 0;
        ll curTime = -1;
        for (ll j = 0; j < n; j++)
        {
            if (curTime <= mt[j][0])
            {
                curAns++;
                curTime = mt[j][1];
            }
        }
        ans = curAns;
    }
    return ans;
}
ll bao(vector<vector<ll>> &mt)
{
    return f(mt, mt.size(), 0);
}
```

```
ll meeting(vector<vector<ll>> &mt)
{
    sort(mt.begin(), mt.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[1] < b[1]; });

    ll ans = 0, n = mt.size();
    for (ll i = 0, time = -1; i < n; i++)
    {
        if (time <= mt[i][0])
        {
            ans++;
            time = mt[i][1];
        }
    }
    return ans;
}

// maximum event selection problem, but each event requires one day to attend, and you can only attend one event
per day.
ll events(vector<vector<ll>> &events)
{
    sort(events.begin(), events.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[0] < b[0]; });
    priority_queue<ll, vector<ll>, greater<ll>> pq;

    ll min = events[0][0];
    ll max = 0;
    for (vector<ll> &eve : events)
    {
        max = std::max(max, eve[1]);
    }

    ll ans = 0, pos = 0;
    ll n = events.size();
    for (ll i = min; i <= max; i++)
    {
        while (pos < n && i == events[pos][0])
        {
            pq.push(events[pos++][1]);
        }
        while (!pq.empty() && pq.top() < i)
        {
            pq.pop();
        }

        if (!pq.empty())
        {
            ans++;
            pq.pop();
        }
    }

    return ans;
}

// non-negative arr, repeatedly calculate the absolute difference of any two numbers and add any new differences
to arr, return the final length of arr when its size no longer changes.
ll gcd(ll a, ll b)
{
    return b != 0 ? gcd(b, a % b) : a;
}
ll abs_diff(vector<ll> &arr)
{
    ll max = 0, all_gcd = 0;
    for (ll num : arr)
    {
        if (num != 0)
```

```
        {
            all_gcd = num;
        }
        max = std::max(max, num);
    }

    if (!all_gcd)
    {
        return arr.size();
    }

    unordered_map<ll, ll> mp;
    for (ll num : arr)
    {
        mp[num]++;
        if (num != 0)
        {
            all_gcd = gcd(all_gcd, num);
        }
    }
    ll ans = max / all_gcd;
    bool rep = false;
    for (auto itr : mp)
    {
        if (itr.first != 0)
        {
            ans += (itr.second - 1);
            if (itr.second > 1)
            {
                rep = true;
            }
        }
    }

    if (mp[0] > 0)
    {
        ans += mp[0];
    }
    else if (rep)
    {
        ans++;
    }

    return ans;
}
```

3. 贪心-03

```
// array, you can sort a contiguous subarray to make the entire array becomes sorted in ascending order, find the shortest subarray length.
ll unsorted(vector<ll> &nums)
{
    // 1 2 5 4 3 6 7
    // ✓ ✓ ✓ × × ✓ ✓
    // ✓ ✓ × × ✓ ✓ ✓
    ll n = nums.size();
    ll max = LLONG_MIN, min = LLONG_MAX;
    ll r = -1;
    for (ll i = 0; i < n; i++)
    {
        max = std::max(max, nums[i]);
        if (nums[i] < max)
        {
            r = i;
        }
    }
    ll l = n;
    for (ll i = n - 1; i >= 0; i--)
```

```
    {
        min = std::min(min, nums[i]);
        if (nums[i] > min)
        {
            l = i;
        }
    }

    return std::max(r - l + 1, (ll)0);
}

// k non-decreasingly sorted lists of integers, find the minimum interval that includes at least one number from
each of the k lists.
struct node
{
    ll v;
    ll k;
    ll pos;
    node(ll val, ll i, ll j) : v(val), k(i), pos(j) {}
    bool operator<(const node &other) const
    {
        if (v != other.v)
        {
            return v < other.v;
        }
        return k < other.k;
    }
};

vector<ll> smallest(vector<vector<ll>> &nums)
{
    ll n = nums.size();
    set<node> sma;
    for (ll i = 0; i < n; i++)
    {
        sma.insert(node(nums[i][0], i, 0));
    }

    ll ans = LLONG_MAX;
    ll l = 0, r = 0;
    while (sma.size() > n - 1)
    {
        node min = *sma.begin(), max = *sma.rbegin();
        sma.erase(sma.begin());

        if (max.v - min.v < ans)
        {
            ans = max.v - min.v;
            l = min.v, r = max.v;
        }

        ll i = min.k, j = min.pos;
        if (j + 1 < nums[i].size())
        {
            sma.insert(node(nums[i][j + 1], i, j + 1));
        }
    }

    vector<ll> res = {l, r};
    return res;
}

// n people can choose from m activities or play nothing at all,  the total cost for activity i is C_i  (X) = max(X (B_i
— K_iX),0), where X is the number of participants, find the maximum possible total cost when all n people make
their choices.
struct park
{
```

```
    ll ki;
    ll bi;
    ll peo;

    park(ll k, ll b) : ki(k), bi(b), peo(0) {}
    ll earn() const
    {
        // one more people
        // (peo+1).(bi-(peo+1).ki) - peo.(bi-peo.ki)
        return bi - (2 * peo + 1) * ki;
    }
};
struct comp
{
    bool operator()(const park &a, const park &b) const
    {
        return a.earn() < b.earn();
    }
};

ll cost_max(ll n, vector<vector<ll>> &active)
{
    ll m = active.size();
    priority_queue<park, vector<park>, comp> pq;

    for (vector<ll> &act : active)
    {
        pq.push(park(act[0], act[1]));
    }

    ll ans = 0;
    for (ll i = 0; i < n; i++)
    {
        park cur = pq.top();
        pq.pop();
        if (cur.earn() >= 0)
        {
            ans += cur.earn();
            cur.peo++;
            pq.push(cur);
        }
        else
        {
            break;
        }
    }

    return ans;
}

// arr length n, you must partition arr into exactly k subsets, return the minimum possible sum of the average
values of each subset.
ll ave_sum(vector<ll> &arr, ll k)
{
    sort(arr.begin(), arr.end());
    ll ans = 0;
    for (ll i = 0; i < k - 1; i++)
    {
        ans += arr[i];
    }
    ll aver = 0, n = arr.size();
    for (ll i = k - 1; i < n; i++)
    {
        aver += arr[i];
    }
    aver /= n - k + 1;
    return ans + aver;
```

```
}

// tasks, each with an energy cost and a minimum starting energy, you can complete the tasks in any order, find the minimum initial energy required to complete all tasks.
ll effort(vector<vector<ll>> &tasks)
{
    // {cost - starting}
    sort(tasks.begin(), tasks.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[0] - a[1] > b[0] - b[1]; });

    ll ans = 0;
    for (vector<ll> &tk : tasks)
    {
        ans = max(ans + tk[0], tk[1]);
    }
    return ans;
}

4. 贪心-04
// arr positive integers, you can divide even numbers by 2 or multiply odd numbers by 2, find the minimum possible maximum difference between any two elements after some operations.
ll deviation(vector<ll> &nums)
{
    set<ll> s;
    for (ll num : nums)
    {
        s.insert(num % 2 != 0 ? 2 * num : num);
    }

    ll ans = *s.rbegin() - *s.begin();
    while (ans > 0 && *s.rbegin() % 2 == 0)
    {
        ll max = *s.rbegin();
        s.erase(max);
        max /= 2;
        s.insert(max);
        ans = min(ans, *s.rbegin() - *s.begin());
    }

    return ans;
}

// array answers containing replies from several rabbits to the question, "how many other rabbits are the same color as you?", find the minimum number of rabbits in the forest.
ll rabbits(vector<ll> &answers)
{
    unordered_map<ll, ll> mp;
    ll ans = 0;
    for (ll aw : answers)
    {
        mp[aw]++;
    }
    for (auto &rb : mp)
    {
        // ceil(a/b) = (a + b - 1)/b
        ans += (rb.second + rb.first) / (rb.first + 1) * (rb.first + 1);
    }
    return ans;
}

// arrays nums and target, you can perform an operation: choose two distinct i and j in nums and set nums[i] += 2, nums[j] -= 2, find the minimum number of operations to make nums and target similar (i.e., have the same element frequencies).
ll split(vector<ll> &arr, ll n)
{
    ll osize = 0;
    for (ll i = 0; i < n; i++)
```

```
    {
        if (arr[i] & 1)
        {
            swap(arr[i], arr[osize++]);
        }
    }
    return osize;
}
ll similar(vector<ll> &nums, vector<ll> &tar)
{
    ll n = tar.size();
    ll osize = split(tar, n);
    sort(tar.begin(), tar.begin() + osize);
    sort(tar.begin() + osize, tar.end());
    split(nums, n);
    sort(nums.begin(), nums.begin() + osize);
    sort(nums.begin() + osize, nums.end());

    ll ans = 0;
    for (ll i = 0; i < n; i++)
    {
        ans += abs(nums[i] - tar[i]);
    }
    return ans / 4;
}

// n employees, each with two ability scores a_i and b_i, when two employees i, j form a team, their team abilities
are x = (a_i+a_j)/2 and y = (b_i+b_j)/2, maximize(min(X, Y)).
double quiz(vector<vector<ll>> &abt)
{
    sort(abt.begin(), abt.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return abs(a[0] - a[1]) < abs(b[0] - b[1]); });

    ll n = abt.size();
    ll amax = abt[0][0], bmax = abt[0][1];
    ll ans = 0;
    for (ll i = 1; i < n; i++)
    {
        if (abt[i][0] > abt[i][1])
        {
            ans = max(ans, abt[i][1] + bmax);
        }
        else
        {
            ans = max(ans, abt[i][0] + amax);
        }
        amax = max(amax, abt[i][0]), bmax = max(bmax, abt[i][1]);
    }
    return ans / 2.0;
}

// sorted array, determine if arr can be partitioned into one or more non-overlapping, increasing subsequences,
each of length at least k.
bool partition(vector<ll> &arr, ll k)
{
    // approach: the more subsequences you have, the harder it is to reach the goal, therefore, using the
frequency of the most frequent number as the number of subsequences is optimal.
    // 1 1 2 2 3 3 3 4 4 5 6 7
    // 1 2 3 5
    //   2 3 4 7
    //   1 3 4 6
    ll cnt = 1, max = 1;
    ll n = arr.size();
    for (ll i = 1; i < n; i++)
    {
        if (arr[i] != arr[i - 1])
        {
```

```
            max = std::max(max, cnt);
            cnt = 1;
        }
        else
        {
            cnt++;
        }
    }
    max = std::max(max, cnt);
    return n / max >= k;
}
```

5. 贪心-05

```
// arr[i] is the maximum jump distance from index i, starting from index 0, find the minimum number of jumps required to reach the last index n-1.
ll jump(vector<ll> &arr)
{
    ll n = arr.size();
    ll cur = 0, next = 0, ans = 0;
    for (ll i = 0; i < n; i++)
    {
        if (cur < i)
        {
            ans++;
            cur = next;
        }
        // next records the maximum potential reach of the positions reachable in the previous ans jumps.
        next = max(next, i + arr[i]);
    }
    return ans;
}

// garden length n and n+1 sprinklers at positions [0, 1, ..., n], each sprinkler i has a watering range of [i - ranges[i], i + ranges[i]], find the minimum number of sprinklers required to water the entire garden.
ll taps(ll n, vector<ll> &rg)
{
    ll r[10001] = {0};
    for (ll i = 0, l; i < n + 1; i++)
    {
        l = (i - rg[i]) > 0 ? (i - rg[i]) : 0;
        r[l] = max(rg[l], i + rg[i]);
    }

    ll cur = 0, next = 0, ans = 0;
    for (ll i = 0; i < n + 1; i++)
    {
        if (cur < i)
        {
            if (next < i)
            {
                return -1;
            }
            ans++;
            cur = next;
        }
        next = max(next, r[i]);
    }
    return ans;
}

// n people cross a river, each with a specific crossing time, the boat can hold at most two people, total time for two people to cross the river equals max(t_i, t_j), find the minimum total time for everyone to cross.
ll cross(vector<ll> &t)
{
    // 1 2 3
    // 2 → 1 3
    // 1 2 ← 3
```

```
    // → 1 2 3
    sort(t.begin(), t.end());
    ll dp[100001];
    ll n = t.size();
    if (n < 2)
    {
        return t[0];
    }
    else if (n < 3)
    {
        return t[1];
    }
    else if (n < 4)
    {
        return t[0] + t[1] + t[2];
    }
    else
    {
        dp[0] = t[0], dp[1] = t[1], dp[2] = t[0] + t[1] + t[2];
        for (ll i = 3; i < n; i++)
        {
            dp[i] = min(t[i] + t[0] + dp[i - 1], t[i] + 2 * t[1] + t[0] + dp[i - 2]);
        }
    }
    return dp[n - 1];
}

// some clothes in n machines, you can simultaneously move one piece of clothing from several machines to an adjacent machine, find the minimum number of steps to make the number of clothes in all machines equal.
ll moves(vector<ll> &mh)
{
    // 0 3 0
    // 1 2 0
    // 1 1 1 ✓
    ll sum = 0, n = mh.size();
    for (ll clo : mh)
    {
        sum += clo;
    }
    if (sum % n != 0)
    {
        return -1;
    }

    ll ln, rn, avg = sum / n, ans = 0;
    sum = 0;
    for (ll i = 0; i < n; i++)
    {
        ln = i * avg - sum;
        sum += mh[i];
        // (n - i - 1) * avg - sum - (n * avg - sum)
        rn = sum - (i + 1) * avg;
        if (ln > 0 && rn > 0)
        {
            ans = max(ans, ln + rn);
        }
        else
        {
            ans = max(ans, max(abs(ln), abs(rn)));
        }
    }

    return ans;
}
```

6. 贪心-06

```
// 73414494957  →  7449447
```

```
// 0090 → 9
string palindromic(string num)
{
    ll cnt[10] = {};
    for (char s : num)
    {
        cnt[s - '0']++;
    }
    ll ans[10001];
    ll ls = 0, mid = 0;

    for (ll i = 9; i > 0; i--)
    {
        if ((cnt[i] & 1) && !mid)
        {
            mid = i;
        }
        for (ll j = cnt[i] / 2; j > 0; j--)
        {
            ans[ls++] = i;
        }
    }

    if (!ls)
    {
        if (mid)
        {
            return to_string(mid);
        }
        else
        {
            return "0";
        }
    }

    for (ll j = cnt[0] / 2; j > 0; j--)
    {
        ans[ls++] = 0;
    }

    bool has_middle = (mid != 0 || (cnt[0] & 1));
    ll len = ls * 2 + (has_middle ? 1 : 0);

    string result(len, ' ');

    for (ll i = 0; i < ls; i++)
    {
        char ch = (char)('0' + ans[i]);
        result[i] = ch;
        result[len - 1 - i] = ch;
    }

    if (has_middle)
    {
        result[ls] = (char)('0' + mid);
    }

    return result;
}

// n workers with their quality and minimum expected wage, hire k workers, where each worker's wage must be
proportional to their quality and not less than their minimum expected wage, find the minimum total cost to hire k
workers.
struct ly
{
    ll qul;
    ll wg;
```

```
    double rate;

    ly(ll q, ll w) : qul(q), wg(w), rate((double)w / q) {}
    bool operator<(const ly &other) const
    {
        return qul < other.qul;
    }
};

double hire(vector<ll> &qul, vector<ll> &wg, ll k)
{
    // align with the lyee who has the highest unit price, otherwise, the requirements will inevitably be violated.
    ll n = qul.size();
    vector<ly> emp;
    for (ll i = 0; i < n; i++)
    {
        emp.push_back(ly(qul[i], wg[i]));
    }
    sort(emp.begin(), emp.end(), [](const ly &a, const ly &b)
         { return a.rate < b.rate; });

    priority_queue<ly> pq;
    ll qsum = 0;
    double ans = 1e12;
    for (ll i = 0; i < n; i++)
    {
        if (pq.size() < k)
        {
            qsum += emp[i].qul;
            pq.push(ly(emp[i].qul, emp[i].wg));
            if (pq.size() >= k)
            {
                ans = min(ans, qsum * emp[i].rate);
            }
        }
        else
        {
            if (emp[i].qul < pq.top().qul)
            {
                qsum += emp[i].qul - pq.top().qul;
                pq.pop();
                pq.push(ly(emp[i].qul, emp[i].wg));
                ans = min(ans, qsum * emp[i].rate);
            }
        }
    }

    return ans;
}

// n trees, each with an initial weight and a daily growth rate, you can cut at most one tree per day for m days,
the value of a tree is its initial weight plus its total growth up to that day, find the maximum total value you can
obtain over m days.
ll cutting(vector<vector<ll>> &tree, ll n, ll m){
    ll dp[251][251];
    memset(dp, 0, sizeof(dp));
    sort(tree.begin() + 1, tree.end(), [](const vector<ll> &a, const vector<ll> &b)
         { return a[1] < b[1]; });

    for (ll i = 1; i < n + 1; i++){
        for (ll j = 1; j < m + 1; j++){
            dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - 1] + tree[i][0] + tree[i][1] * (j - 1));
        }
    }

    return dp[n][m];
}
```

初始数组 c 长度为 n，每个位置都是 *（空白）。有 m 个神经网络，第 i 个网络给出一个长度为 n 的字符串数组 b_i。可以做两类操作：
1. 选择某个网络 i：它会在当前所有空白位置中随机选一个位置 j，并把 c_j 设为 b_{i,j}。
2. 选择位置 j，把 c_j 设回空白 *。
贪心思路：
先使用在匹配 a 的位置上数量最多（记作 x）的网络。对该网络做 n 次操作（最坏情况）会把所有 n 个位置都填上，其中 x 个位置是正确的，n-x 个位置错误。后续每修正一个错误位置需要恰好 2 次操作。

```
void solve()
{
    int n, m;
    cin >> n >> m;
    vs a(n);
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    vvs b(m, vs(n));
    for (int i = 0; i < m; ++i)
    {
        for (int j = 0; j < n; ++j)
            cin >> b[i][j];
    }

    vb vis(n, false);
    int best = 0;
    for (int i = 0; i < m; ++i)
    {
        int cnt = 0;
        for (int j = 0; j < n; ++j)
        {
            if (b[i][j] == a[j])
            {
                ++cnt;
                vis[j] = true;
            }
        }
        best = max(best, cnt);
    }

    if (!all_of(vis.begin(), vis.end(), [](bool x)
                { return x; }))
    {
        cout << -1 << '\n';
    }
    else
    {
        cout << (n + 2 * (n - best)) << '\n';
    }
}
```

动态规划

1. 一维动态规划

```
// array of travel days and the costs of three types of passes with different durations (1, 7, 30), find the minimum
total cost to cover all the specified travel days.
vl dur = {1, 7, 30};
vl dp(366, -1);
ll f(vl &days, vl &costs, ll i, ll n)
{
    if (i == n)
    {
        return 0;
    }

    if (dp[i] != -1)
    {
        return dp[i];
    }

    ll ans = LLONG_MAX;
    for (ll k = 0, j = i; k < 3; k++)
    {
        while (j < n && dur[k] + days[i] > days[j])
        {
            j++;
        }
        ans = min(ans, costs[k] + f(days, costs, j, n));
    }

    dp[i] = ans;
    return ans;
}
ll tickets(vl &days, vl &costs)
{
    ll n = days.size();
    return f(days, costs, 0, n);
}

// string s of digits, where letters A-Z are mapped to numbers 1-26, find the total number of ways to decode s into
a sequence of letters.
// ll f(string &s, ll i)
// {
//      ll n = s.length();
//      if (i == n)
//      {
//          return 1;
//      }

//      ll ans;
//      if (s[i] == '0')
//      {
//          ans = 0;
//      }
//      else
//      {
//          ans = f(s, i + 1);
//          if (i + 1 < n && 10 * (s[i] - '0') + (s[i + 1] - '0') < 27)
//          {
//              ans += f(s, i + 2);
//          }
//      }

//      return ans;
// }
ll decodings(string s)
{
    // f(s, 0);
    ll n = s.length();
```

```
    vl dp(n + 1);
    dp[n] = 1;

    for (ll i = n - 1; i >= 0; i--)
    {
        if (s[i] == '0')
        {
            dp[i] = 0;
        }
        else
        {
            dp[i] = dp[i + 1];
            if (i + 1 < n && 10 * (s[i] - '0') + (s[i + 1] - '0') < 27)
            {
                dp[i] += dp[i + 2];
            }
        }
    }

    return dp[0];
}
// decodings-2.0, string s of digits and *, * can represent any digit from 1 to 9.

// return the n-th ugly number, which is a positive integer whose only prime factors are 2, 3, or 5.
ll ugly(ll n)
{
    vl dp(n + 1);
    dp[1] = 1;

    for (ll i = 2, pto = 1, pte = 1, pfv = 1, to, te, fv; i < n + 1; i++)
    {
        to = 2 * dp[pto];
        te = 3 * dp[pte];
        fv = 5 * dp[pfv];
        ll min = std::min(to, std::min(te, fv));

        min != to ?: pto++;
        min != te ?: pte++;
        min != fv ?: pfv++;

        dp[i] = min;
    }

    return dp[n];
}

// string containing only '(' and ')', find the length of the longest valid parenthesis substring.
ll parentheses(string s)
{
    ll ans = 0, n = s.length();
    vl dp(n, 0);

    for (ll i = 1, p; i < n; i++)
    {
        if (s[i] != '(')
        {
            p = i - dp[i - 1] - 1;
            if (p >= 0 && s[p] == '(')
            {
                dp[i] = dp[i - 1] + 2 + (p > 0 ? dp[p - 1] : 0);
            }
        }
        ans = max(ans, dp[i]);
    }

    return ans;
}
```

```
// string s, count and return how many different non-empty substrings of s also appear in an infinitely repeating
string base of the alphabet.
// base: ...abcdefg...xyzabcdefg...
ll wraproud(string s)
{
    ll ans = 0, n = s.length();
    vl dp(26, 0);
    dp[s[0] - 'a'] = 1;

    for (ll i = 1, len = 1; i < n; i++)
    {
        ll ch = s[i] - 'a';
        (s[i] - s[i - 1] + 26) % 26 != 1 ? len = 1 : len++;
        dp[ch] = max(dp[ch], len);
    }

    for (ll num : dp)
    {
        ans += num;
    }
    return ans;
}

// string s, count the number of different non-empty subsequences.
// abc → {a}, {b}, {c}, {ab}, {ac}, {bc}
const ll mod = 1e9 + 7;
ll distinct(string s)
{
    vl cnt(26, 0);
    ll all = 1, nadd;
    for (char x : s)
    {
        nadd = (all - cnt[x - 'a'] + mod) % mod;
        all = (all + nadd) % mod;
        cnt[x - 'a'] = (cnt[x - 'a'] + nadd) % mod;
    }

    return (all - 1 + mod) % mod;
}

卡登算法：求解子段最大和问题
问：数组中两个不重叠子段的最大和是多少？
ll solve(ll n)
{
    if (n < 2)
    {
        return 0;
    }

    vl a(n);
    for (ll i = 0; i < n; ++i)
    {
        cin >> a[i];
    }

    // maximum subarray sum of a[0...i]
    vl l(n);

    // Kadane's Algorithm
    ll cml = 0;
    ll gml = LLONG_MIN;

    for (ll i = 0; i < n; ++i)
    {
        cml = max(a[i], cml + a[i]);
        gml = max(gml, cml);
```

```
        l[i] = gml;
    }

    // maximum subarray sum of a[i...n-1]
    vl r(n);

    ll cmr = 0;
    ll gmr = LLONG_MIN;

    for (ll i = n - 1; i > -1; --i)
    {
        cmr = max(a[i], cmr + a[i]);
        gmr = max(gmr, cmr);
        r[i] = gmr;
    }

    // l[i] + r[i+1]
    ll res = LLONG_MIN;
    for (ll i = 0; i < n - 1; ++i)
    {
        res = max(res, l[i] + r[i + 1]);
    }
    return res;
}
```

2. 二维动态规划

```
// m x n mat of non-negative llegers, find a path from the top-left to the bottom-right that minimizes the sum of the numbers along the path, you can only move down or right at each step.
ll path(vvl &mat)
{
    ll n = mat.size(), m = mat[0].size();
    vvl dp(n, vl(m, inf));

    dp[0][0] = mat[0][0];
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            if (!i && !j)
                continue;
            dp[i][j] = mat[i][j] + min(i > 0 ? dp[i - 1][j] : inf, j > 0 ? dp[i][j - 1] : inf);
        }
    }
    return dp[n - 1][m - 1];
}

// longest common subsequence
ll lcs(string a, string b)
{
    ll n = a.length(), m = b.length();
    vvl dp(n + 1, vl(m + 1, 0));

    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < m + 1; j++)
        {
            dp[i][j] = a[i - 1] != b[j - 1] ? max(dp[i - 1][j], dp[i][j - 1]) : dp[i - 1][j - 1] + 1;
        }
    }

    return dp[n][m];
}

// longest palindrome subsequence (not substring)
// ll lps(string s)
// {
//      string rs(s.rbegin(), s.rend());
```

```
//       return lcs(s, rs);
// }
ll lps(string s)
{
     ll n = s.length();
     vvl dp(n, vl(n, 0));

     for (ll l = n - 1; l > -1; l--)
     {
          dp[l][l] = 1;
          if (l + 1 < n)
          {
               dp[l][l + 1] = (s[l] != s[l + 1] ? 1 : 2);
          }
          for (ll r = l + 2; r < n; r++)
          {
               dp[l][r] = s[l] != s[r] ? max(dp[l + 1][r], dp[l][r - 1]) : dp[l + 1][r - 1] + 2;
          }
     }

     return dp[0][n - 1];
}

// count the number of binary trees with n nodes and a height of at most m.
ll tren(ll n, ll m)
{
     vvl dp(n + 1, vl(m + 1, 0));
     for (ll j = 0; j < m; j++)
     {
          dp[0][j] = 1;
     }

     for (ll i = 1; i < n + 1; i++)
     {
          for (ll j = 1; j < m + 1; j++)
          {
               for (ll k = 0; k < i; k++)
               {
                    dp[i][j] = (dp[i][j] + dp[k][j - 1] * dp[i - k - 1][j - 1] % mod) % mod;
               }
          }
     }

     return dp[n][m];
}

// m x n lleger matrix, find the length of the longest increasing path.
ll path(vvl &mat)
{
     ll n = mat.size(), m = mat[0].size();
     vvl dp(n, vl(m, 0));
     ll ans = 0;
     for (ll i = 0; i < n; i++)
     {
          for (ll j = 0; j < m; j++)
          {
               ans = max(ans, f(mat, i, j, dp, n, m));
          }
     }
     return ans;
}

ll f(vvl &mat, ll i, ll j, vvl &dp, ll n, ll m)
{
     if (dp[i][j] != 0)
     {
          return dp[i][j];
```

```
    }

    ll next = 0;
    if (i > 0 && mat[i][j] < mat[i - 1][j])
    {
        next = max(next, f(mat, i - 1, j, dp, n, m));
    }
    if (i + 1 < n && mat[i][j] < mat[i + 1][j])
    {
        next = max(next, f(mat, i + 1, j, dp, n, m));
    }
    if (j > 0 && mat[i][j] < mat[i][j - 1])
    {
        next = max(next, f(mat, i, j - 1, dp, n, m));
    }
    if (j + 1 < m && mat[i][j] < mat[i][j + 1])
    {
        next = max(next, f(mat, i, j + 1, dp, n, m));
    }

    next++;
    dp[i][j] = next;
    return next;
}

// s and t, count the number of times t appears as a subsequence in s.
ll distinct(string s, string t)
{
    ll n = s.length(), m = t.length();
    vvl dp(n + 1, vl(m + 1, 0)); // dp[i][j] → s.substr(0, i) & t.substr(0, j)

    for (ll i = 0; i < n + 1; i++)
    {
        dp[i][0] = 1;
    }
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < m + 1; j++)
        {
            dp[i][j] = dp[i - 1][j];
            dp[i][j] += (s[i - 1] != t[j - 1] ? 0 : dp[i - 1][j - 1]);
        }
    }

    return dp[n][m];
}

// transform s into t by inserting, deleting, or replacing characters, with each operation having a different cost,
find the minimum total cost required.
ll distance(string s, string t, ll a, ll b, ll c)
{
    ll n = s.length(), m = t.length();
    vvl dp(n + 1, vl(m + 1, 0));

    for (ll i = 0; i < n + 1; i++)
    {
        dp[i][0] = i * b;
    }
    for (ll j = 0; j < m + 1; j++)
    {
        dp[0][j] = j * a;
    }

    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < m + 1; j++)
        {
```

```
                dp[i][j] = (s[i - 1] != t[j - 1] ? min(min(dp[i - 1][j] + b, dp[i][j - 1] + a), dp[i - 1][j - 1] + c) : dp[i - 1][j
- 1]);
            }
        }

        return dp[n][m];
}

// verify if c is an interleaving of a and b.
bool leave(string a, string b, string c)
{
        if (a.length() + b.length() != c.length())
        {
            return false;
        }

        ll n = a.length(), m = b.length();
        vvl dp(n + 1, vl(m + 1, 0)); // dp[i][j] → a.substr(0, i) & b.substr(0, j) → c.substr(0, i+j)

        dp[0][0] = 1;
        for (ll i = 1; i < n + 1; i++)
        {
            if (a[i - 1] != c[i - 1])
            {
                break;
            }
            dp[i][0] = 1;
        }
        for (ll j = 1; j < m + 1; j++)
        {
            if (b[j - 1] != c[j - 1])
            {
                break;
            }
            dp[0][j] = 1;
        }

        for (ll i = 1; i < n + 1; i++)
        {
            for (ll j = 1; j < m + 1; j++)
            {
                dp[i][j] =
                    (a[i - 1] == c[i + j - 1] && dp[i - 1][j]) ||
                    (b[j - 1] == c[i + j - 1] && dp[i][j - 1]);
            }
        }

        return dp[n][m];
}

// n cells and m colors, color the n cells such that each of the m colors is used at least once, find the total number
of valid ways.
ll color(ll n, ll m)
{
        vvl dp(n + 1, vl(m + 1, 0)); // dp[i][j] → i cells, j colors
        for (ll i = 1; i < n + 1; i++)
        {
            dp[i][1] = m;
        }

        for (ll i = 2; i < n + 1; i++)
        {
            for (ll j = 2; j < m + 1; j++)
            {
                dp[i][j] = j * dp[i - 1][j] + (m - j + 1) * dp[i - 1][j - 1];
            }
        }
```

```
    return dp[n][m];
}

// the minimum number of characters that must be deleted from s to make it a substring of t.
ll remove(string s, string t)
{
    ll n = s.length(), m = t.length();
    vvl dp(n + 1, vl(m + 1, 0)); // dp[i][j] → s.substr(0, i) be suffix of t.substr(0, j)

    for (ll i = 0; i < n + 1; i++)
    {
        dp[i][0] = i;
    }
    for (ll i = 1; i < n + 1; i++)
    {
        for (ll j = 1; j < m + 1; j++)
        {
            dp[i][j] = (s[i - 1] != t[j - 1] ? dp[i - 1][j] + 1 : dp[i - 1][j - 1]);
        }
    }
    ll ans = LLONG_MAX;
    for (ll j = 0; j < m + 1; j++)
    {
        ans = min(ans, dp[n][j]);
    }
    return ans;
}
```

3. 三维动态规划

```
// binary strings strs, find the maximum size of a subset of strs such that the total number of '0's is at most m and
the total number of '1's is at most n.
ll o, t;
void count(string str)
{
    o = 0;
    t = 0;
    for (ll i = 0; i < str.length(); i++)
    {
        str[i] != '0' ? t++ : o++;
    }
}
ll form(vs &strs, ll m, ll n)
{
    // ll dp[len + 1][m + 1][n + 1]; // strs[k,...], i 0, j 1
    vvl dp(m + 1, vl(n + 1, 0));
    for (string str : strs)
    {
        count(str);
        for (ll i = m; i + 1 > o; i--)
        {
            for (ll j = n; j + 1 > t; j--)
            {
                dp[i][j] = max(dp[i][j], 1 + dp[i - o][j - t]);
            }
        }
    }

    return dp[m][n];
}

// m jobs, each with a profit and a required members, the total profit is at least minProfit and the total number
of participants is at most n, find the number of different profitable plans.
ll profitable(ll n, ll pt, vl &g, vl &p)
{
    vvl dp(n + 1, vl(pt + 1));
    for (ll r = 0; r < n + 1; r++)
```

```
    {
        dp[r][0] = 1;
    }
    ll m = g.size();

    for (ll i = m - 1; i > -1; i--)
    {
        for (ll r = n; r > -1; r--)
        {
            for (ll s = pt; s > -1; s--)
            {
                ll pp = g[i] < r + 1 ? dp[r - g[i]][max((ll)0, s - p[i])] : 0;
                dp[r][s] = (pp + dp[r][s]) % mod;
            }
        }
    }
    return dp[n][pt];
}

// n x n chessboard, a knight starts at (row, col) and makes k moves, each move, the knight randomly chooses one
of the 8 possible moves, find the probability that the knight remains on the board after k moves.
double f(ll n, ll i, ll j, ll k, vvvd &dp)
{
    if (i < 0 || i > n - 1 || j < 0 || j > n - 1)
    {
        return 0;
    }
    if (dp[i][j][k] != -1)
    {
        return dp[i][j][k];
    }
    double ans = 0;
    if (k == 0)
    {
        ans = 1;
    }
    else
    {
        ans += (f(n, i - 2, j + 1, k - 1, dp) / 8);
        ans += (f(n, i - 1, j + 2, k - 1, dp) / 8);
        ans += (f(n, i + 1, j + 2, k - 1, dp) / 8);
        ans += (f(n, i + 2, j + 1, k - 1, dp) / 8);
        ans += (f(n, i + 2, j - 1, k - 1, dp) / 8);
        ans += (f(n, i + 1, j - 2, k - 1, dp) / 8);
        ans += (f(n, i - 1, j - 2, k - 1, dp) / 8);
        ans += (f(n, i - 2, j - 1, k - 1, dp) / 8);
    }
    dp[i][j][k] = ans;
    return ans;
}
double knight(ll n, ll k, ll row, ll col)
{
    vvvd dp(n, vvd(n, vd(k + 1, -1)));
    return f(n, row, col, k, dp);
}

// n x m grid and k, you can only move down or right from the top-left to the bottom-right, find the number of paths
where the sum of the numbers along the path is divisible by k.
ll paths(vvl &grid, ll k, ll n, ll m){
    vvvl dp(n, vvl(m, vl(k, 0)));
    dp[n - 1][m - 1][grid[n - 1][m - 1] % k] = 1;

    for (ll i = n - 2; i > -1; i--){
        for (ll r = 0; r < k; r++){
            dp[i][m - 1][r] = dp[i + 1][m - 1][(k + r - grid[i][m - 1] % k) % k];
        }
    }
```

```
    for (ll j = m - 2; j > -1; j--){
        for (ll r = 0; r < k; r++){
            dp[n - 1][j][r] = dp[n - 1][j + 1][(k + r - grid[n - 1][j] % k) % k];
        }
    }
    for (ll i = n - 2, need; i > -1; i--){
        for (ll j = m - 2; j > -1; j--){
            for (ll r = 0; r < k; r++)
            {
                need = (k + r - grid[i][j] % k) % k;
                dp[i][j][r] = dp[i + 1][j][need];
                dp[i][j][r] = (dp[i][j][r] + dp[i][j + 1][need]) % mod;
            }
        }
    }
    return dp[0][0][0];
}

// https://leetcode.cn/problems/scramble-string/
bool scramble(string s, string t){
    ll n = s.length();
    vvvl dp(n, vvl(n, vl(n + 1, 0)));

    for (ll l1 = 0; l1 < n; l1++){
        for (ll l2 = 0; l2 < n; l2++){
            dp[l1][l2][1] = (s[l1] != t[l2] ? 0 : 1);
        }
    }
    for (ll len = 2; len < n + 1; len++){
        for (ll l1 = 0; l1 < n + 1 - len; l1++){
            for (ll l2 = 0; l2 < n + 1 - len; l2++){
                for (ll k = 1; k < len; k++){
                    if (dp[l1][l2][k] && dp[l1 + k][l2 + k][len - k])
                    {
                        dp[l1][l2][len] = 1;
                        break;
                    }
                }
                if (!dp[l1][l2][len]){
                    for (ll i = l1 + 1, j = l2 + len - 1, k = 1; k < len; i++, j--, k++){
                        if (dp[l1][j][k] && dp[i][l2][len - k]){
                            dp[l1][l2][len] = 1;
                            break;
                        }
                    }
                }
            }
        }
    }
    return dp[0][0][n];
}
```

字符串

1. KMP

```
// nexp array represents the length of the longest prefix of the string up to the current position (not including the current character) that is also a suffix of that same string (not including the whole string).
vl nexp(100001);
void f(string s, ll m)
{
    nexp[0] = -1, nexp[1] = 0;
    // s != ""
    if (m < 2)
        return;
    ll i = 2, cn = 0;
    while (i < m + 1)
    {
        if (s[i - 1] == s[cn])
        {
```

```
                nexp[i++] = ++cn;
            }
            else if (cn > 0)
            {
                cn = nexp[cn];
            }
            else
            {
                nexp[i++] = 0;
            }
        }
}
// n + m
ll kmp(string s, string tar)
{
    ll n = s.length(), m = tar.length();
    ll x = 0, y = 0;
    while (x < n && y < m)
    {
        if (s[x] == tar[y])
        {
            x++, y++;
        }
        else if (y == 0)
        {
            x++;
        }
        else
        {
            y = nexp[y];
        }
    }

    return y != m ? -1 : x - y;
}

// binary trees, root and subRoot, check if root contains a subtree that has the same structure and node values
as subRoot.
struct node
{
    node *left;
    node *right;
    ll val;
};

void f(node *root, string &s)
{
    if (root == nullptr)
    {
        s += "#,";
    }
    else
    {
        ll v = root->val;
        s += (to_string(v) + ",");
        f(root->left, s);
        f(root->right, s);
    }
}
string serialize(node *root)
{
    string s;
    f(root, s);
    return s;
}
bool subtree(node *root, node *sub)
{
```

```
    if (root != nullptr && sub != nullptr)
    {
        string r = ',' + serialize(root);
        string s = ',' + serialize(sub);
        return kmp(r, s) != -1;
    }

    return sub == nullptr;
}

// string s which is formed by repeating a core unit, at least two repetitions, but the last one might be incomplete, find the length of the shortest repeating unit of s.
// U U U U l
// max(nexp) = (k-1)U + l
ll unit(string s)
{
    ll n = s.length();
    f(s, n);
    return n - nexp[n];
}

// string s and a pattern string tar, repeatedly remove the leftmost occurrence of tar from s until s no longer contains tar, return the final string.
void remove(string s, string tar)
{
    ll n = s.length(), m = tar.length(), x = 0, y = 0;
    vpll st(n);
    ll top = 0;
    f(tar, m);

    while (x < n)
    {
        if (s[x] == tar[y])
        {
            st[top].first = x++;
            st[top++].second = y++;
        }
        else if (y == 0)
        {
            st[top].first = x++;
            st[top++].second = -1;
        }
        else
        {
            y = nexp[y];
        }

        // x a b c b c c y a b c a
        // a b c
        //-1 0 0
        if (y == m)
        {
            top -= m;
            y = top > 0 ? (st[top - 1].second + 1) : 0;
        }
    }

    for (ll i = 0; i < top; i++)
    {
        cout << s[st[i].first];
    }
}

// binary tree and a linked list, determine if there is a downward path in the tree whose sequence of values is identical to that of the linked list.
struct ln
{
```

```
    ll val;
    ln *next;
};
void f(vl &s, ll m)
{
    nexp[0] = -1, nexp[1] = 0;
    // s != ""
    if (m < 2)
        return;
    ll i = 2, cn = 0;
    while (i < m + 1)
    {
        if (s[i - 1] == s[cn])
        {
            nexp[i++] = ++cn;
        }
        else if (cn > 0)
        {
            cn = nexp[cn];
        }
        else
        {
            nexp[i++] = 0;
        }
    }
}
bool find(vl &tar, vl &nexp, node *cur, ll i, ll m)
{
    if (i == m)
        return true;
    if (cur == nullptr)
        return false;

    while (i > -1 && cur->val != tar[i])
    {
        i = nexp[i];
    }
    return find(tar, nexp, cur->left, i + 1, m) || find(tar, nexp, cur->right, i + 1, m);
}
// n + m
bool subpath(ln *head, node *root)
{
    ll m = 0;
    ln *t = head;
    while (t != nullptr)
    {
        m++;
        t = t->next;
    }

    vl tar(m);
    m = 0;
    while (head != nullptr)
    {
        tar[m++] = head->val;
        head = head->next;
    }

    f(tar, m);
    return find(tar, nexp, root, 0, m);
}

// st and se of length n, and a string evil, "good string" is a string of length n that is lexicographically greater than
or equal to st, less than or equal to se, and does not contain evil as a substring, return the number of good strings.
ll jump(char pick, string evil, ll j)
{
    while (j > -1 && pick != evil[j])
```

```
    {
        j = nexp[j];
    }
    return j;
}
ll solve(ll n, ll m, string s, string evil, ll i, ll j, ll free)
{
    if (j == m)
        return 0;
    if (i == n)
        return 1;

    if (dp[i][j][free] != -1)
    {
        return dp[i][j][free];
    }

    char ch = s[i];
    ll ans = 0;
    if (!free)
    {
        for (char pick = 'a'; pick < ch; pick++)
        {
            ans = (ans + solve(n, m, s, evil, i + 1, jump(pick, evil, j) + 1, 1)) % mod;
        }
        ans = (ans + solve(n, m, s, evil, i + 1, jump(ch, evil, j) + 1, 0)) % mod;
    }
    else
    {
        for (char pick = 'a'; pick - 1 < 'z'; pick++)
        {
            ans = (ans + solve(n, m, s, evil, i + 1, jump(pick, evil, j) + 1, 1)) % mod;
        }
    }

    dp[i][j][free] = ans;
    return ans;
}
ll dp[501][501][2];
void clear(ll n, ll m)
{
    for (ll i = 0; i < n; i++)
    {
        for (ll j = 0; j < m; j++)
        {
            dp[i][j][0] = -1;
            dp[i][j][1] = -1;
        }
    }
}
const ll mod = 1e9 + 7;
ll evil(ll n, string st, string se, string evil)
{
    ll m = evil.length();
    f(evil, m);
    clear(n, m);
    ll ans = solve(n, m, se, evil, 0, 0, 0);
    clear(n, m);
    ans = (ans - solve(n, m, st, evil, 0, 0, 0) + mod) % mod;

    if (kmp(st, evil) == -1)
    {
        ans = (ans + 1) % mod;
    }
    return ans;
}
```

2. 字符串哈希

```
// 赌狗的胜利
// '0' → 1
// '1' → 2
// ...
// '9' → 10
// 'A' → 11
// 'B' → 12
// ...
// 'a' → 37
// ...
ll v(char c)
{
    if (c > 47 && c < 58)
    {
        return c - '0' + 1;
    }
    else if (c > 64 && c < 91)
    {
        return c - 'A' + 11;
    }
    else
    {
        return c - 'a' + 37;
    }
}
ll value(string s)
{
    ll ans = v(s[0]);
    for (ll i = 1; i < s.length(); i++)
    {
        ans = ans * base + v(s[i]);
    }
    return ans;
}

// s of digits, return the number of unique non-empty substrings where each digit appears with the same
frequency.
ll frequency(string &str)
{
    ll n = str.length();
    unordered_set<ll> s;
    vl cnt(10);

    for (ll i = 0; i < n; i++)
    {
        fill(cnt.begin(), cnt.end(), 0);
        ll hashCode = 0;
        ll curVal = 0, maxCnt = 0, maxCntKinds = 0, allKinds = 0;

        for (ll j = i; j < n; j++)
        {
            curVal = str[j] - '0';
            hashCode = hashCode * base + curVal + 1;

            if (cnt[curVal] == 0)
            {
                allKinds++;
            }
            cnt[curVal]++;

            if (cnt[curVal] > maxCnt)
            {
                maxCnt = cnt[curVal];
                maxCntKinds = 1;
            }
            else if (cnt[curVal] == maxCnt)
```

```
                {
                    maxCntKinds++;
                }

                if (maxCntKinds == allKinds)
                {
                    s.insert(hashCode);
                }
            }
        }

        return s.size();
}

// substring hash
vl bp(maxn), ph(maxn);
void build(string s, ll n)
{
        bp[0] = 1;
        for (ll i = 1; i < n; i++)
        {
            bp[i] = base * bp[i - 1];
        }
        ph[0] = s[0] - 'a' + 1;
        for (ll i = 1; i < n; i++)
        {
            ph[i] = base * ph[i - 1] + s[i] - 'a' + 1;
        }
}
ll subh(ll l, ll r)
{
        ll ans = ph[r];
        if (l > 0)
        {
            ans -= ph[l - 1] * bp[r - l + 1];
        }
        return ans;
}

// the minimum number of times to repeat a so that b becomes a substring of the repeated string.
ull repeated(string a, string b)
{
        ull n = a.length(), m = b.length();
        ull k = (m + n - 1) / n;

        // k+1
        ull len = 0;
        for (ull i = 0; i < k + 1; i++)
        {
            for (ull j = 0; j < n; j++)
            {
                s[len++] = a[j];
            }
        }

        // build(len);
        ull hb = b[0] - 'a' + 1;
        for (ull i = 1; i < m; i++)
        {
            hb = base * hb + b[i] - 'a' + 1;
        }

        for (ull l = 0, r = m - 1; r < len; l++, r++)
        {
            if (hb == subh(l, r))
            {
                return r < n * k ? k : k + 1;
```

```
        }
    }
    return -1;
}

// array of words where all words have the same length, concatenated substring is a substring formed by concatenating all the words from the array in any order, find all starting indices of such concatenated substrings in s.
vl substring(string s, vs &rds)
{
    vl ans;
    if (s.empty() || rds.empty() || rds[0].empty())
    {
        return ans;
    }

    unordered_map<ull, ll> mp;
    for (string &rd : rds)
    {
        ull hd = value(rd);
        mp[hd]++;
    }

    ll n = s.length();
    build(s, n);

    ll len = rds[0].size();
    ll num = rds.size();
    ll allen = num * len;

    for (ll init = 0; init < len && init + allen < n + 1; init++)
    {
        unordered_map<ull, ll> slid;
        ll debt = num;

        for (ll i = 0; i < num; i++)
        {
            ull ch = subh(init + i * len, init + (i + 1) * len);
            slid[ch]++;
            if (mp.count(ch) && slid[ch] < mp[ch] + 1)
            {
                debt--;
            }
        }

        if (debt == 0)
        {
            ans.push_back(init);
        }

        for (ll l1 = init; l1 + allen + len < n + 1; l1 += len)
        {
            ll r1 = l1 + len;
            ll l2 = l1 + allen;
            ll r2 = l2 + len;

            ull oh = subh(l1, r1);
            if (mp.count(oh) && slid[oh] < mp[oh] + 1)
            {
                debt++;
            }
            slid[oh]--;

            ull ih = subh(l2, r2);
            slid[ih]++;
            if (mp.count(ih) && slid[ih] < mp[ih] + 1)
            {
```

```
                debt--;
            }

            if (debt == 0)
            {
                ans.push_back(r1);
            }
        }
    }
    return ans;
}

// substring of s matches p if they have the same length and their characters differ by at most k positions, find
the number of substrings in s that match p.
vl bp(maxn), phs(maxn), php(maxn);
void build(string s, ull n, string p, ll m)
{
    bp[0] = 1;
    for (ull i = 1; i < n; i++)
    {
        bp[i] = base * bp[i - 1];
    }

    phs[0] = s[0] - 'a' + 1;
    for (ull i = 1; i < n; i++)
    {
        phs[i] = base * phs[i - 1] + s[i] - 'a' + 1;
    }
    php[0] = p[0] - 'a' + 1;
    for (ull i = 1; i < m; i++)
    {
        php[i] = base * php[i - 1] + p[i] - 'a' + 1;
    }
}
ull subh(vl &ph, ull l, ull r)
{
    ull ans = ph[r];
    if (l > 0)
    {
        ans -= ph[l - 1] * bp[r - l + 1];
    }
    return ans;
}
bool same(ll ls, ll lp, ll len)
{
    return subh(phs, ls, ls + len - 1) == subh(php, lp, lp + len - 1);
}
// s[l,r], p[0,m-1]
bool check(ll ls, ll rs, ll k)
{
    ll diff = 0;
    ll lp = 0;

    while (ls < rs + 1 && diff < k + 1)
    {
        ll l = 1, r = rs - ls + 1;
        ll m, len = 0;
        // search max length
        while (l < r + 1)
        {
            m = (l + r) / 2;
            if (same(ls, lp, m))
            {
                len = m;
                l = m + 1;
            }
            else
```

```
                {
                    r = m - 1;
                }
            }
            if (ls + len < rs + 1)
            {
                diff++;
            }
            ls += len + 1;
            lp += len + 1;
        }

        return diff < k + 1;
}
ll match(string s, string p, ll k)
{
        ll n = s.length(), m = p.length();
        if (n < m)
        {
            return 0;
        }
        build(s, n, p, m);
        ll ans = 0;
        for (ll i = 0; i < n - m + 1; i++)
        {
            if (check(i, i + m - 1, k))
            {
                ans++;
            }
        }
        return ans;
}
```

交互题

刷新缓冲区

```
// 换行 + 刷新
std::cout << "? " << l << " " << r << std::endl;

// 或者，手动刷新（不换行）
std::cout << "! " << answer;
std::cout.flush();
```

交互题的本质是信息论——如何用**最少的问题获取最多的信息**。因此，很多高效的**搜索算法**是其核心。
问："猜数字"，裁判心中有一个 1 到 n 之间的数，你最多可以猜 logN 次。每次猜测 x，裁判会告诉你 x 是大了、小了还是对了。

```
void guess(ll n) {
    ll left = 1, right = n;
    while (left <= right) {
        ll mid = left + (right - left) / 2;
```

```
        // 1. 提问
        cout << "? " << mid << endl;

        // 2. 读回答
        string response;
        cin >> response;

        // 3. 根据回答调整策略
        if (response == "correct") {
            cout << "! " << mid << endl;
            return;
        } else if (response == "smaller") {
            right = mid - 1;
        } else { // response == "larger"
            left = mid + 1;
        }
    }
}
```

问：有一个长度为 n 的隐藏数组，你不能直接看到它的值。但你可以做一种查询 ？i j，裁判会返回 A[i] 和 A[j] 中值较大的那个元素的索引。要求在有限次数内找到最大值的索引。

```
ll findex(ll n) {
    ll max_idx = 0; // 假设 0 号是当前最大
    for (ll i = 1; i < n; ++i) {
        // 1. 提问：比较当前最大和下一个元素
        cout << "? " << max_idx + 1 << " " << i + 1 << endl; // 假设题目是 1-based

        // 2. 读回答
        ll winner_idx;
        cin >> winner_idx;

        // 3. 更新
        max_idx = winner_idx - 1; // 更新为胜者的索引
    }

    // 最终答案
    cout << "! " << max_idx + 1 << endl;
    return max_idx;
}
```

有些题目中，如果你问了一个不合法的问题，裁判会返回 -1 或其他特殊值。你的程序需要能处理这种情况，而不是直接崩溃。