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;

}

1. 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;

}

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;

}

1. 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;

}

1. 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;

}

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;

}

1. 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;

}

1. 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).

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;

}

signed main()

{

ll a = 5, b = 10;

cout << "before swap: a = " << a << ", b = " << b << endl;

swap\_xor(a, b);

cout << "after swap: a = " << a << ", b = " << b << endl;

max\_xor(a, b);

cout << "max\_xor: " << max\_xor(a, b) << endl

<< "max\_xor\_overflow: " << max\_xor\_overflow(a, b) << endl;

vector<ll> nums = {0, 1, 2, 4, 5}; // missing 3

cout << "missing\_num: " << missing\_num(nums) << endl;

vector<ll> odd\_even\_nums = {1, 2, 3, 2, 1, 3, 4};

cout << "odd\_and\_even: " << odd\_and\_even(odd\_even\_nums) << endl;

vector<ll> odd\_even\_plus\_nums = {1, 2, 3, 2, 1, 3, 4, 5};

cout << "odd\_and\_even\_plus: ";

odd\_and\_even\_plus(odd\_even\_plus\_nums);

vector<ll> cnt\_m\_nums = {1, 2, 3, 1, 2, 3, 8, 1, 2, 3};

ll m = 3;

cout << "\ncnt\_m: " << cnt\_m(cnt\_m\_nums, m) << endl;

}

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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

};

1. 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();

}

};

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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);

}

1. 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;

}

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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);

}

}

}

1. 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;

}

1. 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));

}

1. 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;

}

}

}

1. 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);

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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);

}

1. 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;

}

1. 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;

}

1. 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;

}

}

1. 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);

}

1. 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;

}

}

1. 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;

}

1. 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;

}

1. 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;

}

好题

1. 嵌套类问题（递归求解）
   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. 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. 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. 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);

}

1. 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;

}

}

1. 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;

}

1. 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.

1. 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;

}

1. 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;

}

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;

}

1. 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

1. 博弈问题
   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";

}

* 1. 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

* 1. 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;

}

}

* 1. 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

* 1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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。

1. 矩阵构造性质
   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。

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;

}

1. 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;

}

1. 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.

1. BFS
   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;

}

* 1. 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;

}

* 1. 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;

}

1. 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;

}

1. 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;

}

1. 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;

}

1. 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)

{

// cell(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;

}

1. 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;

}

1. 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];

}

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 wrapround(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;

}

1. 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;

}

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;

}

1. 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;

}