



University of Wisconsin, Madison

Model Solution

bvd top robe

2019-11-07

1 Contest

2 Mathematics

3 Data structures

4 Graph

5 Flows and Matching

6 Strings

Contest (1)

template.cpp15 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define trav(a, x) for(auto& a : x)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
    cin.sync_with_stdio(0); cin.tie(0);
    cin.exceptions(cin.failbit);
}
```

.bashrc3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <
```

.vimrc6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noe b g=dark ru cul
sy on | im jk <esc> | im kj <esc> | no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \ | tr -d '[:space:]' \
\ | md5sum \ | cut -c-6
```

hash.sh3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

troubleshoot.txt52 lines

```
Pre-submit:
Write a few simple test cases, if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.
```

```
Wrong answer:
Print your solution! Print debug output, as well.
```

```
1 Are you clearing all datastructures between test cases?
Can your algorithm handle the whole range of input?
2 Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
5 Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
8 Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
11 Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
12 Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.
```

```
Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).
```

```
Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your team mates think about your algorithm?
```

```
Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all datastructures between test cases?
```

clion.txt3 lines

```
set(CMAKE_CXX_STANDARD 17)
set(GCC_COVERAGE_COMPILE_FLAGS "-g -O2 -std=gnu++17 -static -
Wall -Werror")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${
GCC_COVERAGE_COMPILE_FLAGS}") )
```

LargerLocalStackSize.cpp33 lines

```
#include <bits/stdc++.h>
#include <inttypes.h>

static void main2(){
    freopen("jackin.txt", "r", stdin);
    freopen("jackout.txt", "w", stdout);

    scanf("%d", &T);
}

static void run_with_stack_size(void (*func)(), size_t stsize)
{
    char *stack, *send;
    stack=(char *)malloc(stsize);
    send=stack+stsize-16;
```

```
send=(char *)((uintptr_t)send/16*16);
asm volatile(
    "mov %%rsp, (%0)\n"
    "mov %0, %%rsp\n"
    :
    : "r" (send));
func();
asm volatile(
    "mov (%0), %%rsp\n"
    :
    : "r" (send));
free(stack);
}
```

```
int main() {
    run_with_stack_size(main2, 64*1024*1024);
}
```

pbdsTree.cpp47 lines

```
/**
 * Author:
 * Date:
 * License:
 * Source:
 * Description:
 */
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
tree_order_statistics_node_update
```

```
using namespace std;
typedef pair<int, int> P;
typedef long long ll;
using namespace __gnu_pbds;
```

```
typedef tree<
    int,
    null_type,
    less<>,
    rb_tree_tag,
    tree_order_statistics_node_update>
ordered_set ;
```

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

```
ordered_set X;
X.insert(1);
X.insert(2);
X.insert(4);
X.insert(8);
X.insert(16);
```

```
cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<*X.find_by_order(4)<<endl; // 16
cout<<(end(X)==X.find_by_order(6))<<endl; // true
```

```
cout<<X.order_of_key(-5)<<endl; // 0
cout<<X.order_of_key(1)<<endl; // 0
cout<<X.order_of_key(3)<<endl; // 2
cout<<X.order_of_key(4)<<endl; // 2
cout<<X.order_of_key(400)<<endl; // 5
}
```

MinorThings.cpp

```

44 lines
/**
 * Author:
 * Date:
 * License:
 * Source:
 * Description:
 */
#include <bits/stdc++.h>
using namespace std;

// Define Hash Function for hash map
struct HASH{
    size_t operator()(const P &x)const{
        return hash<ll>() ((x.first)^(x.second)<<32));
    }
};
unordered_map<P, int> ump;

// multiply numbers up to 1e18 under some modulo
ll big_mul(ll a, ll b)
{
    ll q = (ll)((ld) a * (ld) b / (ld) mod);
    ll r = a * b - q * mod;

    return (r + mod) % mod;
}

int main(){
    //random number
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().
count());
    cout << rng() % 5 << endl;
    std::shuffle(v.begin(), v.end(), rng);

    //calculating sum of floor(n/i) in O(sqrt(n))
    for (int i = 1, j = 0; i <= n; i = j + 1) j = n/(n/i), ans
+= 1ll*(j-i+1)*(n/i);

    //Better hash map
    unordered_map<int, int> mp;
    mp.reserve(32768);
    mp.max_load_factor(0.25);
}
```

Mathematics (2)

LinearSieve.cpp

```

74cf4c, 30 lines
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int M = 100050;

bool p[M];
int mu[M], pm[M], phi[M], pms;

void get_mobius_and_sieve(){
    mu[1] = 1;
    phi[1] = 1;
    fill(p, p + M, true);
    pms = 0;
    for(int i = 2; i < M; i++){
        if(p[i]){
            pm[pms++] = i;
            mu[i] = -1;
            phi[i] = i - 1;
        }
    }
}
```

```

for(int j = 0; j < pms && i * pm[j] < M; j++){
    p[i * pm[j]] = false;
    if(i % pm[j] == 0){
        mu[i * pm[j]] = 0;
        phi[i * pm[j]] = phi[i] * pm[j];
        break;
    }
    mu[i * pm[j]] = -mu[i];
    phi[i * pm[j]] = phi[i] * (pm[j] - 1);
}
}
}
```

NTT.cpp

```

0964fe, 68 lines
#include <bits/stdc++.h>
using namespace std;

// p | k | g
// 469762049 26 3
// 998244353 23 3
// 1004535809 21 3
// 2281701377 27 3

typedef long long ll;
typedef pair<int, int> P;
const int N = (1 << 16) + 50, mod = 998244353, g = 3;

int rev[N], w[2][N];

int pow(int x, int k){
    int c = 1;
    for(; k; k >>= 1, x = int(x * 1LL * x % mod)) if(k & 1) c =
int(c * 1LL * x % mod);
    return c;
}

void init(int len){
    for(int i = 0; i < len; i++){
        int y = 0, x = i;
        for(int k = 1; k < len; k *= 2, x >>= 1) (y <= 1) |= (
x & 1);
        rev[i] = y;
    }
    w[0][0] = w[1][0] = 1;
    int mp = pow(g, (mod-1)/len), ni = pow(mp, mod - 2);
    for(int i = 1; i < len; i++){
        w[0][i] = int(w[0][i-1] * 1LL * mp % mod);
        w[1][i] = int(w[1][i-1] * 1LL * ni % mod);
    }
}

void NTT(vector<int> &y, int on, int len){
    y.resize(len, 0);
    for(int i = 0; i < len; i++) if(i > rev[i]){int tmp = y[i];
y[i] = y[rev[i]], y[rev[i]] = tmp;}
    for(int h = 2; h <= len; h <= 1) {
        int wi = len / h;
        for (int j = 0; j < len; j += h) {
            int l = 0;
            for (int k = j; k < j + h / 2; k++) {
                int u = y[k];
                int t = int(1LL * w[on==1][l] * y[k + h / 2] %
mod);
                y[k] = (u + t) % mod;
                y[k + h / 2] = ((u - t) % mod + mod) % mod;
                l += wi;
            }
        }
    }
}
```

```

}

if(on == -1){
    int ni = pow(len, mod-2);
    for(int i = 0; i < len; i++) y[i] = int(1LL * y[i] * ni
% mod);
}
}

vector<int> mult_poly(vector<int> a, vector<int> b) {
    int len = 1;
    while(len < a.size() + b.size() + 1) len *= 2;
    init(len);
    NTT(a, 1, len);
    NTT(b, 1, len);
    for(int i = 0; i < len; i++) a[i] = (int)(1LL * a[i] * b[i]
% mod);
    NTT(a, -1, len);
    while(!a.empty() && a.back() == 0) a.pop_back();
    return a;
}
```

FFT.cpp

```

eb68a8, 65 lines
const double PI = acos(-1.0);

struct Complex{
    double x, y;
    Complex(double _x = 0.0, double _y = 0.0){
        x = _x, y = _y;
    }
    Complex operator - (const Complex &b) const{
        return Complex(x - b.x, y - b.y);
    }
    Complex operator + (const Complex &b) const{
        return Complex(x + b.x, y + b.y);
    }
    Complex operator * (const Complex &b) const{
        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
    }
};

void change(Complex y[], int len){
    int i, j, k;
    for(i = 1, j = len/2; i < len - 1; i++){
        if(i < j){Complex tmp = y[i]; y[i] = y[j], y[j] = tmp;}
        k = len / 2;
        while(j >= k){
            j = j - k;
            k = k / 2;
        }
        if(j < k) j += k;
    }
}

void fft(Complex y[], int len, int on){
    change(y, len);
    for(int h = 2; h <= len; h <= 1){
        Complex wn(cos(-on*2*PI/h), sin(-on*2*PI/h));
        for(int j = 0; j < len; j += h){
            Complex w(1,0);
            for(int k = j; k < j + h/2; k++){
                Complex u = y[k];
                Complex t = w * y[k + h/2];
                y[k] = u + t;
                y[k + h/2] = u - t;
                w = w * wn;
            }
        }
    }
}
```

```
    }
    if(on == -1){
        for(int i = 0; i < len; i++){
            y[i].x /= len;
        }
    }
}

const int N = 200050;

Complex x1[N], x2[N];

int main() {
    for(int i = 0; i < len1; i++) x1[i] = Complex(1, 0);
    for(int i = 0; i < len2; i++) x2[i] = Complex(1, 0);
    fft(x1, len, 1);
    fft(x2, len, 1);
    for(int i = 0; i < len; i++) x1[i] = x1[i] * x2[i];
    fft(x1, len, -1);
}
```

FWT.cpp

<bits/stdc++.h>df14b9, 51 lines

```
using namespace std;

typedef long long ll;
const int N = (int)1e6 + (int)1e5, mod = (int)1e9 + 7, inv2 = (
    int)5e8 + 4, M = (int)1e5 + 50, INF = (int)1e9;

int add(int a, int b) {
    a = (a + b) % mod;
    if(a < 0) a += mod;
    return a;
}

int mul(int a, int b) {
    return (int)(1LL * a * b % mod);
}

int p[N];

int n, m;
string str[20];
int freq[N], cnt[N], res[N];

struct FWT {
    // Please set N!!!
    int N;
    // Sum over Subsets
    void FWTor(int *a, int opt) {
        for(int mid = 1; mid < N; mid <= 1)
            for(int R = mid << 1, j = 0; j < N; j += R)
                for(int k = 0; k < mid; k++)
                    if(opt == 1) a[j + k + mid] = add(a[j + k],
                        a[j + k + mid]);
                    else a[j + k + mid] = add(a[j + k + mid], -
                        a[j + k]);
    }
    // Sum over supersets
    void FWTand(int *a, int opt) {
        for(int mid = 1; mid < N; mid <= 1)
            for(int R = mid << 1, j = 0; j < N; j += R)
                for(int k = 0; k < mid; k++)
                    if(opt == 1) a[j + k] = add(a[j + k], a[j +
                        k + mid]);
                    else a[j + k] = add(a[j + k], -a[j + k +
                        mid]);
    }
}
```

```
void FWTxor(int *a, int opt) {
    for(int mid = 1; mid < N; mid <= 1)
        for(int R = mid << 1, j = 0; j < N; j += R)
            for(int k = 0; k < mid; k++) {
                int x = a[j + k], y = a[j + k + mid];
                if(opt == 1) a[j + k] = add(x, y), a[j + k
                    + mid] = add(x, -y);
                else a[j + k] = mul(add(x, y), inv2), a[j +
                    k + mid] = mul(add(x, -y), inv2);
            }
    }
}

} fwt;
```

FastExponentiationFactorialInverse.cpp

<bits/stdc++.h>01518e, 33 lines

```
using namespace std;

typedef long long ll;
const int N = 105;
const ll mod = 998244353;

int n, num[N];
ll fac[N], facinv[N];
ll inv[N];

ll fp(ll x, ll k){
    if(k == 0) return 1;
    ll hf = fp(x, k/2);
    return k % 2 ? hf * hf % mod * x % mod: hf * hf % mod;
}

ll comb(int n, int k){
    return fac[n] * facinv[k] % mod * facinv[n - k] % mod;
}

void init_fac() {
    inv[1] = 1;
    for(int i = 2; i < N; i++) inv[i] = (mod - (mod / i) * inv[
        mod % i] % mod) % mod;
    fac[0] = 1;
    for(int i = 1; i <= N-1; i++) fac[i] = fac[i-1] * i % mod;
    facinv[N-1] = fp(fac[N-1], mod - 2);
    for(int i = N-1 - 1; i >= 0; i--) facinv[i] = facinv[i+1] *
        (i+1) % mod;
}

int main(){
    init_fac();
}
```

GaussianElimination.cpp

<bits/stdc++.h>7ca530, 64 lines

```
using namespace std;

typedef long long ll;
const int N = 100 + 2;
int mod;

int n, m;
ll inv[N];
int num[N][N];

void calc_inv() {
    inv[1] = 1;
    for(int i = 2; i < N; i++) inv[i] = (mod - (mod / i) * inv[
        mod % i] % mod) % mod;
}
```

```

}

struct matrix {
    static const int maxn = 405;
    int n, m;
    ll mat[maxn][maxn];
    ll x[maxn];
    ll res[maxn];
    matrix() { memset(mat, 0, sizeof(mat)); }
    bool gauss() {
        for (int i = 0; i < n; i++) {
            int sid = -1;
            for (int j = i; j < n; j++)
                if (mat[j][i] > 0) {
                    sid = j;
                    break;
                }
            if (sid == -1) continue;
            if (sid != i) {
                for (int j = 0; j < n; j++) {
                    swap(mat[sid][j], mat[i][j]);
                }
                swap(res[sid], res[i]);
            }
            for (int j = i + 1; j < n; j++) {
                ll ratio = mat[j][i] * inv[mat[i][i]] % mod;
                for (int k = 0; k < n; k++) {
                    mat[j][k] -= mat[i][k] * ratio % mod;
                    mat[j][k] += mod;
                    if(mat[j][k] >= mod) mat[j][k] -= mod;
                }
                res[j] -= res[i] * ratio % mod;
                res[j] += mod;
                if(res[j] >= mod) res[j] -= mod;
            }
        }
        for(int i = m - 1; i >= 0; i--) {
            ll sum = res[i];
            for(int j = i + 1; j < m; j++) {
                sum -= x[j] * mat[i][j] % mod;
                sum %= mod;
            }
            sum = (sum + mod) % mod;
            x[i] = sum * inv[mat[i][i]] % mod;
            if(mat[i][i] == 0 && sum != 0) return false;
        }
        return true;
    }
} mat;
```

EuclidLike.cpp

Description: Calculate Sigma 0 to n, floor((a*i+b)/c) in O(log n)<bits/stdc++.h>f64150, 11 lines

```
using namespace std;

typedef long long ll;

ll fd(ll a,ll b,ll c,ll n) {
    if (a == 0) return ((b / c) * (n + 1));
    if (a >= c || b >= c) return fd(a % c, b % c, c, n) + (a /
        c) * n * (n + 1) / 2 + (b / c) * (n + 1);
    ll m = (a * n + b) / c;
    ll v = fd(c, c - b - 1, a, m - 1);
    return n * m - v;
}
```

XorBasis.cpp

```
<bits/stdc++.h> a667cf, 24 lines

using namespace std;

const int N = (int)2e5 + 50, B = 31;

int n;
int a[N];

struct Basis {
    int sz = 0;
    int bas[B];

    bool add(int x) {
        if(x == 0) return false;
        for(int i = 0; i < sz; i++) {
            if((x ^ bas[i]) < x) x ^= bas[i];
            if(x == 0) return false;
        }
        bas[sz++] = x;
        for(int i = sz - 2; i >= 0; i--) {
            if(bas[i] < bas[i+1]) swap(bas[i], bas[i+1]);
        }
        return true;
    }
};
basis;
```

BabyStepGiantStep.cpp

```
27d863, 31 lines

//Given  $g^x=y(mod\ p)$ , find  $x$  in  $O(\log(p)*sqrt(p))$ 
struct BSGS {
    const static int p = 998244353;
    const static int g = 3;
    int m;
    map<int, int> mp;
    vector<int> V;

    void pre() {
        m = (int)ceil(sqrt(p + 0.0) + 1);
        int cg = 1, revgm = 1;
        int invm = (int)(fp(fp(g, m), p - 2));
        for(int i = 0; i < m; i++) {
            mp[cg] = i;
            V.push_back(revgm);
            cg = (int)(1LL * cg * g % p);
            revgm = (int)(1LL * revgm * invm % p);
        }
    }

    int find(int y) {
        for(int i = 0; i < m; i++) {
            int cur = (int)(1LL * V[i] * y % p);
            if(mp.count(cur)){
                return i * m + mp[cur];
            }
        }
        return -1;
    }
};
```

Matrix.cpp

```
<bits/stdc++.h> 785505, 48 lines

using namespace std;

const int D = (int)101, mod = (int)1e9 + 7;
typedef long long ll;
```

```
struct matrix
{
    int size;
    ll a[D][D];
    void init(int _size = -1){
        if(_size != -1) size = _size;
        memset(a, 0, sizeof(a));
    }
    matrix(int _size = -1){
        (*this).init(_size);
    }
    void set_identity(){
        for(int i = 0; i < size; i++){
            for(int j = 0; j < size; j++) a[i][j] = (ll)(i == j);
        }
    }
    matrix operator * (const matrix &B) const
    {
        matrix C = matrix(size);

        for(int i = 0; i < size; i++){
            for(int j = 0; j < size; j++){
                for(int k = 0; k < size; k++){
                    C.a[i][j] += a[i][k] * B.a[k][j] % mod;
                    C.a[i][j] %= mod;
                }
            }
        }
        return C;
    }

    matrix operator ^ (const ll &p) const
    {
        matrix A = (*this), res = matrix(A.size);
        res.set_identity();
        ll t = p;
        while(t > 0)
        {
            if(t % 2) res = res * A;
            A = A * A;
            t /= 2;
        }
        return res;
    }
};
```

ExtendedGCD.cpp

Description: want to solve $ax + by = \gcd(a, b)$; be careful that $\gcd(a, b)$ can be negative.

```
<bits/stdc++.h> e4650e, 25 lines

using namespace std;

typedef long long ll;

ll a1, b1, a2, b2, L, R;

ll extgcd(ll a, ll b, ll &x, ll &y) {
    ll d = a;
    if(b != 0) {
        d = extgcd(b, a % b, y, x);
        y -= (a / b) * x;
    }
    else {
        x = 1; y = 0;
    }
    return d;
}

ll mod(ll a, ll b){
```

```
return (a % b + b) % b;
}

ll dvd(ll a, ll b) {
    return (a - mod(a, b)) / b;
}
```

BigInteger.cpp

```
<iostream>, <string>, <cstdio>, <queue>, <cstring> db68dc, 79 lines

struct BigInt
{
    const static int mod = 10000;
    const static int DLEN = 4;
    int a[50], len;
    BigInt()
    {
        memset(a, 0, sizeof(a));
        len = 1;
    }
    BigInt(int v)
    {
        memset(a, 0, sizeof(a));
        len = 0;
        do
        {
            a[len++] = v%mod;
            v /= mod;
        }while(v);
    }
    BigInt(const char s[])
    {
        memset(a, 0, sizeof(a));
        int L = strlen(s);
        len = L/DLEN;
        if(L%DLEN) len++;
        int index = 0;
        for(int i = L-1; i >= 0; i -= DLEN)
        {
            int t = 0;
            int k = i - DLEN + 1;
            if(k < 0) k = 0;
            for(int j = k; j <= i; j++)
                t = t*10 + s[j] - '0';
            a[index++] = t;
        }
    }
    BigInt operator +(const BigInt &b) const
    {
        BigInt res;
        res.len = max(len, b.len);
        for(int i = 0; i <= res.len; i++)
            res.a[i] = 0;
        for(int i = 0; i < res.len; i++)
        {
            res.a[i] += ((i < len)?a[i]:0) + ((i < b.len)?b.a[i]:0);
            res.a[i+1] += res.a[i]/mod;
            res.a[i] %= mod;
        }
        if(res.a[res.len] > 0) res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b) const
    {
        BigInt res;
        for(int i = 0; i < len; i++)
        {
            int up = 0;
```

```

    for(int j = 0; j < b.len; j++)
    {
        int temp = a[i]*b.a[j] + res.a[i+j] + up;
        res.a[i+j] = temp%mod;
        up = temp/mod;
    }
    if(up != 0)
        res.a[i + b.len] = up;
}
res.len = len + b.len;
while(res.a[res.len - 1] == 0 && res.len > 1) res.len--;
return res;
}

void output()
{
    printf("%d", a[len-1]);
    for(int i = len-2; i >= 0; i--)
        printf("%04d", a[i]);
    printf("\n");
}
}dp[2][M];

```

Data structures (3)

1DBIT.cpp

e3a683, 14 lines

//BIT is 0-indexed!!!

```

int n;
int bit[N];

void add(int x, int val) {
    for(int i = x; i < n; i |= i + 1) bit[i] += val;
}

ll get(int x) {
    ll res = 0;
    for(int i = x; i >= 0; i = (i & (i + 1)) - 1) res += bit[i];
    return res;
}

```

2DBIT.cpp

a56764, 29 lines

```

vector<int> vals[N], f[N];

void addupd(int x, int y) {
    for (int i = x; i < N; i |= i + 1) vals[i].push_back(y);
}

void addget(int x, int y) {
    if (x < 0 || y < 0) return;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1) vals[i].push_back(y);
}

void upd(int x, int y, int v) {
    for (int i = x; i < N; i |= i + 1) {
        for (int j = lower_bound(vals[i].begin(), vals[i].end(), y) - vals[i].begin();
            j < (int) f[i].size(); j |= j + 1) {
            f[i][j] += v;
        }
    }
}

int get(int x, int y) {
    if (x < 0 || y < 0) return 0;
}

```

```

int res = 0;
for (int i = x; i >= 0; i = (i & (i + 1)) - 1)
    for (int j = lower_bound(vals[i].begin(), vals[i].end(), y) - vals[i].begin(); j >= 0;
        j = (j & (j + 1)) - 1)
        res += f[i][j];
return res;
}

```

DynamicConvexHullTrick.cpp

<bits/stdc++.h> 897176, 34 lines

```

using namespace std;

typedef long long ll;

struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty());
        auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
};

```

StaticConvexHullTrick.cpp

Description: Maintaining upper convex hull, querying the maximum. Need to put in lines in strictly increasing order of slope.

<bits/stdc++.h> f701ea, 50 lines

```

using namespace std;

struct Line {
    ll k, m;
    Line(ll _k, ll _m) {
        k = _k, m = _m;
    }
    Pll inter(Line o) {
        return {m - o.m, o.k - k};
    }
};

struct Hull {
    deque<Line> que;

    bool leq(Pll a, Pll b) {
        return a.first * b.second <= a.second * b.first;
    }
};

```

```

}
// k needs to be strictly increasing!
void add(ll k, ll m) {
    while(que.size() > 1) {
        int ls = que.size() - 1;
        if(leq(que[ls].inter(Line(k, m)), que[ls-1].inter(que[ls]))) que.pop_back();
        else break;
    }
    que.push_back({k, m});
}
// Arbitrary x.
ll query_bin(ll x) {
    if(que.empty()) return -INF;
    int l = 0, r = que.size() - 1;
    while(l < r) {
        int mi = (l + r) / 2;
        if(que[mi].k * x + que[mi].m < que[mi+1].k * x + que[mi+1].m) l = mi + 1;
        else r = mi;
    }
    return que[l].k * x + que[l].m;
}
// If querying increasing x.
ll query(ll x) {
    if(que.empty()) return -INF;
    while(que.size() > 1) {
        if(que[0].k * x + que[0].m < que[1].k * x + que[1].m) que.pop_front();
        else break;
    }
    return que[0].k * x + que[0].m;
}
} hull;

```

IntervalTree.cpp

c569c2, 27 lines

```

int dat[LOGN][N];

void build(int dep, int l, int r){
    if(l == r){dat[dep][l] = num[l]; return ;}
    int mid = (l + r) / 2;
    build(dep + 1, l, mid);
    build(dep + 1, mid + 1, r);
    int j = mid + 1, cur = l;
    for(int i = l; i <= mid; i++){
        while(j <= r && dat[dep+1][j] < dat[dep+1][i]){
            dat[dep][cur++] = dat[dep+1][j++];
        }
        dat[dep][cur++] = dat[dep+1][i];
    }
    while(j <= r) dat[dep][cur++] = dat[dep+1][j++];
}

int query(int ql, int qr, int dep, int l, int r, int val){
    if(r < ql || l > qr) return 0;
    if(ql <= l && r <= qr){
        return int(upper_bound(dat[dep] + l, dat[dep] + r + 1, val) - (dat[dep] + l));
    }
    int mid = (l + r) / 2;
    int LHS = query(ql, qr, dep + 1, l, mid, val);
    int RHS = query(ql, qr, dep + 1, mid + 1, r, val);
    return LHS + RHS;
}

```

MediumDivideTree.cpp

Description: I unofficially named it medium divide tree. HDU 2665 / POJ 2104 O(logn) query for kth number in certain interval

<iostream>, <cstdio>, <algorithm>a307e1, 64 lines

using namespace std;

const int N = (int)1e5 + 500, LOGN = 20;

int n,q;
int num[N], sorted[N];
int tree[LOGN][N], toleft[LOGN][N];

void build(int dep, int l, int r){
 if(l == r) return ;
 int mid = (l + r) / 2;
 int same = mid - l + 1;
 for(int i = l; i <= r; i++) if(tree[dep][i] < sorted[mid]) same--;
 int lpos = l, rpos = mid + 1;
 for(int i = l; i <= r; i++){
 if(tree[dep][i] < sorted[mid]){
 tree[dep+1][lpos++] = tree[dep][i];
 }
 else if(tree[dep][i] == sorted[mid] && same > 0){
 tree[dep+1][lpos++] = tree[dep][i]; same--;
 }
 else tree[dep+1][rpos++] = tree[dep][i];
 toleft[dep][i] = toleft[dep][l-1] + lpos - 1;
 }
 build(dep + 1, l, mid);
 build(dep + 1, mid + 1, r);
}

void init(){
 for(int i = 1; i <= n; i++) sorted[i] = tree[0][i] = num[i]
 1;
 sort(sorted + 1, sorted + n + 1);
 build(0, 1, n);
}

int query(int dep, int l, int r, int ql, int qr, int k){
 if(ql == qr) return tree[dep][ql];
 int mid = (l + r) / 2;
 int cnt = toleft[dep][qr] - toleft[dep][ql - 1];
 if(cnt >= k){
 int newl = 1 + toleft[dep][ql-1] - toleft[dep][l-1];
 int newr = newl + cnt - 1;
 return query(dep + 1, l, mid, newl, newr, k);
 }
 else{
 int newr = qr + toleft[dep][r] - toleft[dep][qr];
 int newl = newr - (qr - ql - cnt);
 return query(dep + 1, mid + 1, r, newl, newr, k - cnt);
 }
}

int main(){
 int T;
 scanf("%d", &T);
 while(T--){
 scanf("%d%d", &n, &q);
 for(int i = 1; i <= n; i++) scanf("%d", &num[i]);
 init();
 while(q--){
 int a, b, k;
 scanf("%d%d%d", &a, &b, &k);
 printf("%d\n", query(0, 1, n, a, b, k));
 }
 }
}

MonotonousDeque.cpp

<bits/stdc++.h>e8856b, 29 lines

using namespace std;

typedef pair<int, int> P;
typedef long long ll;
const int N = 3005, INF = (int)1e9 + 50;

int n, m, a, b;
ll g0, x, y, z;

int num[N][N];
ll mn[N];

// Monotonous Interval Min Queries
struct MonQueue {
 deque<P> que;

void clear() {
 que.clear();
}
void add(P p) {
 while(!que.empty() && que.back().first >= p.first) que.
 pop_back();
 que.push_back(p);
}

int get(int l, int r) {
 while(!que.empty() && (que.front().second < l || que.
 front().second > r)) que.pop_front();
 return que.empty() ? -INF : que.front().first;
}
} mque[N], cque;

PersistentSegmentTreePointUpdate.cpp

<iostream>, <cstdio>, <vector>, <algorithm>, <assert.h>a04195, 51 lines

using namespace std;

const int N = (int)1e5 + 50;
typedef pair<int, int> P;

//Need to initialize before every test case!
int ncnt = 1;

struct node{
 int ls, rs, sum;
} ns[N * 30];

int newnode(int val){
 ns[ncnt].ls = ns[ncnt].rs = 0;
 ns[ncnt].sum = val;
 return ncnt++;
}

int newnode(int ls, int rs){
 ns[ncnt].ls = ls;
 ns[ncnt].rs = rs;
 ns[ncnt].sum = (ls ? ns[ls].sum : 0) + (rs ? ns[rs].sum :
 0);
 return ncnt++;
}

int n, q;
int num[N];
int x[N], zeros[N];

PersistentSegmentTreeRangeUpdate.cpp

<iostream>, <cstdio>, <vector>, <algorithm>, <assert.h>0a9451, 77 lines

using namespace std;

const int N = (int)1e5 + 50;
typedef pair<int, int> P;
typedef long long ll;

//Need to initialize before every test case!
int ncnt = 1;

struct node{
 int ls, rs, lazy;
 ll sum;
} ns[N * 100];

int newnode(int val){
 ns[ncnt].ls = ns[ncnt].rs = 0;
 ns[ncnt].sum = val;
 ns[ncnt].lazy = 0;
 return ncnt++;
}

int newnode(int ls, int rs){
 ns[ncnt].ls = ls;
 ns[ncnt].rs = rs;
 ns[ncnt].sum = (ls ? ns[ls].sum : 0) + (rs ? ns[rs].sum :
 0);
 ns[ncnt].lazy = 0;
 return ncnt++;
}

int n, q;
int num[N];
int vs[N];
int tim = 0;

int newlazynode(int v, int val, int l, int r){
 ns[ncnt].ls = ns[v].ls;
 ns[ncnt].rs = ns[v].rs;
 ns[ncnt].lazy = ns[v].lazy + val;
 ns[ncnt].sum = ns[v].sum + (r - l + 1) * val;

```

    return ncnt++;
}

void push_down(int v, int tl, int tr){
    if(ns[v].lazy){
        if(tl != tr){
            int mid = (tl + tr) / 2;
            ns[v].ls = newlazynode(ns[v].ls, ns[v].lazy, tl, mid);
            ns[v].rs = newlazynode(ns[v].rs, ns[v].lazy, mid + 1, tr);
        }
        ns[v].lazy = 0;
    }
}

int build(int a[], int tl = 0, int tr = n-1){
    if(tl == tr) return newnode(a[tl]);
    int mid = (tl + tr) / 2;
    return newnode(build(a, tl, mid), build(a, mid + 1, tr));
}

ll get_sum(int v, int l, int r, int tl = 0, int tr = n-1){
    if(tr < l || tl > r) return 0;
    if(l <= tl && tr <= r) return ns[v].sum;
    push_down(v, tl, tr);
    int tm = (tl + tr) / 2;

    return get_sum(ns[v].ls, l, r, tl, tm)
        + get_sum(ns[v].rs, l, r, tm + 1, tr);
}

int update(int v, int l, int r, int val, int tl = 0, int tr = n-1){
    if(tr < l || tl > r) return v;
    if(l <= tl && tr <= r) return newlazynode(v, val, tl, tr);
    push_down(v, tl, tr);
    int tm = (tl + tr) / 2;
    return newnode(update(ns[v].ls, l, r, val, tl, tm), update(
        ns[v].rs, l, r, val, tm+1, tr));
}

```

SegmentTreePointUpdate.cpp

<bits/stdc++.h> 05495a, 61 lines

using namespace std;

```

#define lson(x) 2*x+1
#define rson(x) 2*x+2

```

```

typedef long long ll;
typedef pair<int, int> P;
const int N = (int)2e5 + 500, mod = (int)1e9 + 7;

```

```

int n;
P p[N];
int rs[N];

```

```

struct node {
    int mn;
    int cnt;

    void merge(node &LHS, node &RHS) {
        mn = min(LHS.mn, RHS.mn);
        cnt = (LHS.mn == mn ? LHS.cnt : 0) + (RHS.mn == mn ?
            RHS.cnt : 0);
        cnt %= mod;
    }
};

```

```

struct Tree {
    node dat[N * 4];

    void init_dat(int l, int r, int x){
        if(l == r){dat[x].mn = p[l].first; dat[x].cnt = 1;
            return ;}

        int mid = (l + r) / 2;
        init_dat(l, mid, lson(x));
        init_dat(mid+1, r, rson(x));
        dat[x].merge(dat[lson(x)], dat[rson(x)]);
    }

    void update(int pos, int x, int l, int r, int val, int cnt)
    {
        int mid = (l + r) / 2;
        if(l == r) {
            dat[x].mn = val;
            dat[x].cnt = cnt;
            return ;
        }
        if(pos <= mid) update(pos, lson(x), l, mid, val, cnt);
        else update(pos, rson(x), mid+1, r, val, cnt);
        dat[x].merge(dat[lson(x)], dat[rson(x)]);
    }

    node query(int a, int b, int x, int l, int r){
        if(r < a || b < l) return {mod + 5, 0};

        int mid = (l + r) / 2;
        if(a <= l && r <= b) return dat[x];

        node res;
        node LHS = query(a, b, lson(x), l, mid);
        node RHS = query(a, b, rson(x), mid+1, r);
        res.merge(LHS, RHS);
        return res;
    }
} tree;

```

SegmentTreeRangeUpdate.cpp

<bits/stdc++.h> 826cd9, 103 lines

```

#define ls(x) x * 2 + 1
#define rs(x) x * 2 + 2

```

using namespace std;

```

typedef long long ll;
const int N = (int)1e6 + 50;
int INF = (int)1e9 + 50;

```

```

int n,m,q;
int a[N], b[N];
int num[N];

```

```

struct node {
    int mn, add;

    void add_val(int x) {
        mn += x;
        add += x;
    }

    void merge(node &ls, node &rs) {
        mn = min(ls.mn, rs.mn);
    }
};

```

```

struct Tree {
    node dat[4 * N];

    void push_down(int x, int l, int r) {
        if(dat[x].add) {
            if(l < r) {
                dat[ls(x)].add_val(dat[x].add);
                dat[rs(x)].add_val(dat[x].add);
            }
            dat[x].add = 0;
        }
    }

    void init(int x = 0, int l = 0, int r = n-1) {
        if(l == r) {
            dat[x].mn = num[l];
            dat[x].add = 0;
            return ;
        }
        int mid = (l + r) / 2;
        init(ls(x), l, mid);
        init(rs(x), mid + 1, r);
        dat[x].add = 0;
        dat[x].merge(dat[ls(x)], dat[rs(x)]);
    }

    node query(int a, int b, int x = 0, int l = 0, int r = N-1)
    {
        int mid = (l + r) / 2;
        if(r < a || l > b) return {INF, 0};

        push_down(x, l, r);

        if(l >= a && r <= b) return dat[x];

        node LHS = query(a, b, ls(x), l, mid);
        node RHS = query(a, b, rs(x), mid+1, r);
        node res;
        res.merge(LHS, RHS);
        return res;
    }

    void update(int a, int b, int x, int l, int r, int delta) {
        int mid = (l + r) / 2;
        if(r < a || l > b) return ;

        push_down(x, l, r);

        if(l >= a && r <= b) {
            dat[x].add_val(delta);
            return ;
        }

        update(a, b, ls(x), l, mid, delta);
        update(a, b, rs(x), mid+1, r, delta);

        dat[x].merge(dat[ls(x)], dat[rs(x)]);
    }

    void update(int a, int b, int delta) {
        update(a, b, 0, 0, N - 1, delta);
    }

    int find(int x, int l, int r) {
        if(l == r) return l;
        int mid = (l + r) / 2;
        push_down(x, l, r);
    }
}

```



```

        if(dat[rs(x)].mn < 0) return find(rs(x), mid + 1, r);
        else return find(ls(x), l, mid);
    }

    int find() {
        if(dat[0].mn >= 0) return -1;
        else return find(0, 0, N-1);
    }
} tree;

```

SparseTable.cpp

<bits/stdc++.h>

7c6adf, 90 lines

using namespace std;

```

typedef long long ll;
const int N = (int)3e5 + 50, LOGN = 19;

```

```

struct SA {
    int n;
    // ht[i] = lcp(suffix[sa[i]], suffix[sa[i-1]])
    int rk[N], sa[N], ht[N];
    int st[LOGN + 1][N], mm[N];

    void build(string str) {
        n = str.length();
        str = " " + str;
        static int set[N], a[N];
        for(int i = 1; i <= n; i++) set[i] = str[i];
        sort(set + 1, set + n + 1);
        int *end = unique(set + 1, set + n + 1);
        for(int i = 1; i <= n; i++) a[i] = (int)(lower_bound(
            set + 1, end, str[i]) - set);

        static int fir[N], sec[N], tmp[N], buc[N];
        fill(buc, buc + n + 1, 0);
        for(int i = 1; i <= n; i++) buc[a[i]]++;
        for(int i = 1; i <= n; i++) buc[i] += buc[i-1];
        for(int i = 1; i <= n; i++) rk[i] = buc[a[i]-1] + 1;

        for(int t = 1; t <= n; t *= 2) {
            for(int i = 1; i <= n; i++) fir[i] = rk[i];
            for(int i = 1; i <= n; i++) sec[i] = i + t > n ? 0
                : rk[i + t];

            fill(buc, buc + n + 1, 0);
            for(int i = 1; i <= n; i++) buc[sec[i]]++;
            for(int i = 1; i <= n; i++) buc[i] += buc[i-1];
            for(int i = 1; i <= n; i++) tmp[n - (--buc[sec[i]])
                ] = i;

            fill(buc, buc + n + 1, 0);
            for(int i = 1; i <= n; i++) buc[fir[i]]++;
            for(int i = 1; i <= n; i++) buc[i] += buc[i-1];
            for(int j = 1, i; j <= n; j++) i = tmp[j], sa[buc[
                fir[i]]--] = i;

            bool unique = true;
            for (int j = 1, i, last = 0; j <= n; j++)
            {
                i = sa[j];
                if (!last) rk[i] = 1;
                else if (fir[i] == fir[last] && sec[i] == sec[
                    last]) rk[i] = rk[last], unique = false;
                else rk[i] = rk[last] + 1;

                last = i;
            }
}

```

```

        if (unique) break;
    }

    for(int i = 1, k = 0; i <= n; i++) {
        if(rk[i] == 1) k = 0;
        else {
            if(k > 0) k--;
            int j = sa[rk[i]-1];
            while(i + k <= n && j + k <= n && a[i + k] == a
                [j + k]) k++;
        }
        ht[rk[i]] = k;
    }

    mm[0] = -1;
    for(int i = 1; i <= n; i++) mm[i] = (i & (i-1)) == 0 ?
        mm[i-1] + 1 : mm[i-1];
    for(int i = 0; i <= n; i++){
        st[0][i] = ht[i];
    }
    for(int lg = 1; lg < LOGN; lg++){
        for(int j = 0; j + (1 << lg) - 1 <= n; j++){
            st[lg][j] = min(st[lg-1][j], st[lg-1][j+(1<<(lg
                -1))]);
        }
    }

    int rmq(int l, int r){
        int k = mm[r - l + 1];
        return min(st[k][l], st[k][r-(1<<k)+1]);
    }

    int lcp(int l, int r) {
        if(l == r) return -1;
        int li = rk[l], ri = rk[r];
        if(li > ri) swap(li, ri);
        li++;
        return rmq(li, ri);
    }
} sa;

```

Graph (4)

VertexBiconnectedComponent.cpp

<iostream>, <vector>, <set>, <stack>

5b4cdd, 78 lines

using namespace std;

typedef pair<int, int> P;

const int N = (int)1e4 + 50, M = (int)1e5 + 50;

```

struct edge {
    int to, id;
};

```

```

int n, m;
vector<edge> G[N];
P p[M];
int low[N], pre[N];
int cnt = 0;
vector<int> bcc[N];
int cut[N];
stack<int> S;
int cnt = 0;

```

void init(int n) {

```

    for(int i = 0; i < n; i++) G[i].clear();
    for(int i = 0; i < n; i++) bcc[i].clear();
    cnt = cnt = 0;
    fill(cut, cut + n, 0);
    fill(pre, pre + n, 0);
    fill(low, low + n, 0);
    while(!S.empty()) S.pop();
}

void dfs(int v, int par) {
    low[v] = pre[v] = ++cnt;

    int childcnt = 0;
    for(int j = 0; j < G[v].size(); j++) {
        edge e = G[v][j];
        if(e.to == par) continue;
        if(!pre[e.to]) {
            S.push(e.id);
            childcnt++;
            dfs(e.to, v);
            low[v] = min(low[v], low[e.to]);
            if(low[e.to] >= pre[v]) {
                cut[v] = true;
                int cur;
                do {
                    cur = S.top();
                    S.pop();
                    bcc[cnt].push_back(cur);
                } while(cur != e.id);
                cnt++;
            }
            else if(pre[e.to] < pre[v]){
                S.push(e.id);
                low[v] = min(low[v], pre[e.to]);
            }
        }

        if(childcnt < 2 && par == -1) cut[v] = false;
    }
}

int main() {
    cin >> n >> m;
    init(n);
    for(int i = 0; i < m; i++) {
        int a, b; cin >> a >> b; a--, b--;
        p[i].first = a, p[i].second = b;
        G[a].push_back({b, i});
        G[b].push_back({a, i});
    }
    for(int i = 0; i < n; i++) {
        if(!pre[i]) {
            dfs(i, -1);
        }
    }
}

```

EdgeBiconnectedComponent.cpp

<iostream>, <vector>, <set>, <stack>

014e74, 39 lines

using namespace std;

const int N = (int)3e5 + 50;

```

struct edge {
    int to, id;
};

```

```
int n, m;
vector<edge> G[N];
set<int> nG[N];
int low[N], pre[N];
int cnt = 0;
int cmp[N];
stack<int> S;
int cnt = 0;

void dfs(int v, int id) {
    pre[v] = low[v] = ++cnt;
    S.push(v);
    for(int j = 0; j < G[v].size(); j++) {
        edge e = G[v][j];
        if(e.id == id) continue;
        if(pre[e.to] == 0) {
            dfs(e.to, e.id);
            low[v] = min(low[v], low[e.to]);
        }
        else low[v] = min(low[v], pre[e.to]);
    }

    if(pre[v] == low[v]) {
        int cur;
        do {
            cur = S.top(); S.pop();
            cmp[cur] = cnt;
        } while(cur != v);
        cnt++;
    }
}
```

CentroidDecomposition.cpp

<bits/stdc++.h>e20dbb, 107 lines

using namespace std;

typedef long long ll;
typedef pair<int, int> P;

const int N = 100050, INF = (int)1e9;
const int LOG_N = 17;
int root = 0;
int n,m,a,b;

set<int> G[N];
int parent[LOG_N][N];
int depth[N], par[N], sub[N], dis[N];

void dfs(int v, int p, int d){
 parent[0][v] = p;
 depth[v] = d;
 for(int nxt : G[v]){
 if(nxt != p) dfs(nxt, v, d+1);
 }
}

void init(int V){
 dfs(root, -1, 0);

 for(int k = 0; k+1 < LOG_N; k++){
 for(int v = 0; v < V; v++){
 if(parent[k][v] < 0) parent[k+1][v] = -1;
 else parent[k+1][v] = parent[k][parent[k][v]];
 }
 }
}

```
int lca(int u, int v){
    if(depth[u] > depth[v]){int tmp = u; u = v; v = tmp;}
    for(int k = 0; k < LOG_N; k++){
        if((depth[v] - depth[u]) >> k & 1){
            v = parent[k][v];
        }
    }
    if(u == v) return u;
    for(int k = LOG_N - 1; k >= 0; k--){
        if(parent[k][u] != parent[k][v]){
            u = parent[k][u];
            v = parent[k][v];
        }
    }
    return parent[0][u];
}

int min_dis(int u, int v){
    int ca = lca(u, v);
    return depth[u] + depth[v] - 2 * depth[ca];
}

int dfs1(int v, int p){
    sub[v] = 1;
    for(int nxt : G[v])
        if(nxt != p)
            sub[v] += dfs1(nxt, v);
    return sub[v];
}

int dfs2(int v, int p, int nn){
    for(int nxt : G[v]){
        if(nxt != p && sub[nxt] > nn/2) return dfs2(nxt, v, nn)
        ;
    }
    return v;
}

void decompose(int v, int p){
    dfs1(v, -1);
    int centroid = dfs2(v, -1, sub[v]);
    par[centroid] = p;
    for(int nxt : G[centroid]){
        G[nxt].erase(centroid);
        decompose(nxt, centroid);
    }
    G[centroid].clear();
}

void update(int v){
    int cur = v;
    while(1){
        dis[cur] = min(dis[cur], min_dis(v, cur));
        if(par[cur] == -1) break;
        cur = par[cur];
    }
}

int query(int v){
    int res = (int)1e9;
    int cur = v;
    while(1){
        res = min(dis[cur] + min_dis(v, cur), res);
        if(par[cur] == -1) break;
        cur = par[cur];
    }
    return res;
}
```

```
int main(){
    init(n);
    decompose(0, -1);
    update(0);
}

TreeDiameter.cpp
<bits/stdc++.h>14c68b, 46 lines

using namespace std;

const int N = 123456 + 50;
const int INF = (int)1e9;
typedef pair<int, int> P;

int n, m;
set<int> G[N];
int dis[N];

P bfs(int v) {
    fill(dis, dis + n, INF);
    queue<int> que;
    dis[v] = 0;
    que.push(v);
    while(!que.empty()) {
        int u = que.front(); que.pop();
        for(int nxt : G[u]) {
            if(dis[nxt] > dis[u] + 1) {
                dis[nxt] = dis[u] + 1;
                que.push(nxt);
            }
        }
    }
    int res = -1, u = -1;
    for(int i = 0; i < n; i++) {
        if(dis[i] != INF) {
            if(dis[i] > res){
                res = dis[i];
                u = i;
            }
            else if(dis[i] == res) {
                if(i < u) u = i;
            }
        }
    }
    return {res, u};
}

int get_diameter() {
    int u = -1;
    for(int i = 0; i < n; i++) if(in[i]) {u = i; break;}
    P p = bfs(u);
    P p2 = bfs(p.second);
    return p2.first;
}

Dijkstra.cpp
<bits/stdc++.h>87862d, 39 lines

using namespace std;

typedef long long ll;
typedef pair<ll, int> P;
const int N = (int)1e5 + 500;
const ll INF = (ll)1e18;

struct edge{
    int to, cost;
    edge(int _to, int _cost){
```

```

        to = _to, cost = _cost;
    }
};

int n,m,k,a,b,y;
vector<edge> G[N];
ll dis[N];

void dijkstra(){
    fill(dis, dis+N, INF);
    dis[0] = 0;
    priority_queue<P, vector<P>, greater<P> > pqque;
    pqque.push({0, 0});

    while(!pqque.empty()){
        P p = pqque.top(); pqque.pop();
        int i = p.second;
        ll dist = p.first;
        if(dist > dis[i]) continue;

        for(edge e : G[i]){
            if(e.cost + dis[i] < dis[e.to]){
                dis[e.to] = e.cost + dis[i];
                pqque.push({dis[e.to], e.to});
            }
        }
    }
}

```

DominatorTree.cpp

Description: Dominator tree in $O(M \log(N)/\log(2+M/N))$ time, $O(M+N)$ space Algorithm by T.Lengauer and R.E.Tarjan This is essentially the directed version of articulation points

```

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

typedef long long ll;
const int N = (int)1e5 + 50;

struct Dominator{
    struct min_DSU{
        vector<int> par, val;
        vector<int> const&semi;
        min_DSU(int N, vector<int> const&semi):par(N, -1),val(N), semi(semi){
            iota(val.begin(), val.end(), 0);
        }
        void comp(int x){
            if(par[par[x]]!=-1){
                comp(par[x]);
                if(semi[val[par[x]]]<semi[val[x]])
                    val[x] = val[par[x]];
                par[x]=par[par[x]];
            }
        }
        int f(int x){
            if(par[x]==-1) return x;
            comp(x);
            return val[x];
        }
        void link(int x, int p){
            par[x] = p;
        }
    };
    int N;
    vector<vector<int> > G, rG;
    vector<int> idom, order;

```

```

    Dominator(int _N):N(_N), G(N), rG(N){}
    void add_edge(int a, int b){
        G[a].emplace_back(b);
        rG[b].emplace_back(a);
    }
    vector<int> calc_dominators(int S){
        idom.assign(N, -1);
        vector<int> par(N, -1), semi(N, -1);
        vector<vector<int> > bu(N);
        stack<int> s;
        s.emplace(S);
        while(!s.empty()){
            int a=s.top();s.pop();
            if(semi[a]==-1){
                semi[a] = order.size();
                order.emplace_back(a);
                for(int i=0;i<(int)G[a].size();++i){
                    if(semi[G[a][i]]!=-1){
                        par[G[a][i]]=a;
                        s.push(G[a][i]);
                    }
                }
            }
        }
        min_DSU uni(N, semi);
        for(int i=(int)order.size()-1;i>0;--i){
            int w=order[i];
            for(int f:rG[w]){
                int oval = semi[uni.f(f)];
                if(oval>=0 && semi[w]>oval) semi[w] = oval;
            }
            bu[order[semi[w]]].push_back(w);
            uni.link(w, par[w]);
            while(!bu[par[w]].empty()){
                int v = bu[par[w]].back(); bu[par[w]].pop_back();
            }
            int u=uni.f(v);
            idom[v] = semi[u] < semi[v] ? u : par[w];
        }
        for(int i=1;i<(int)order.size();++i){
            int w=order[i];
            if(idom[w] != order[semi[w]])
                idom[w] = idom[idom[w]];
        }
        idom[S]=-1;
        return idom;
    }
};

int n, m;
vector<int> par;
int sz[N];

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

    cin >> n >> m;
    Dominator D(n);
    for(int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b; a--, b--;
        D.add_edge(a, b);
    }
    // Calc the immediate dominators for each node.
    par = D.calc_dominators(0);

```

```

    // D.order stores the topological order the the dominator
    // tree. 0 is the root.
    // Only reachable nodes from the source will appear here.
    for(int i = 0; i < n; i++) sz[i] = 1;
    for(int i = (int)D.order.size() - 1; i > 0; i--) {
        int e = D.order[i];
        sz[par[e]] += sz[e];
    }
}

```

HLD.cpp

<bits/stdc++.h> ca9278, 59 lines

```

using namespace std;
const int N = 100050;

int n,m,q;
vector<int> G[N];
int chainNo, chainHead[N], chainInd[N], totPos[N], totsize,
    veridx[N];
int subsize[N], par[N];

```

```

void dfs(int v, int p, int d){
    par[v] = parent[0][v] = p;
    subsize[v] = 1;
    for(int &nxt : G[v]){
        if(nxt == p) continue;
        dfs(nxt, v, d+1);
        subsize[v] += subsize[nxt];
        if(subsize[nxt] > subsize[G[v][0]])
            swap(nxt, G[v][0]);
    }
}

```

```

void hld(int v, int p){
    if(chainHead[chainNo] == -1) chainHead[chainNo] = v;
    chainInd[v] = chainNo;
    in[v] = totPos[v] = totsize;
    veridx[totsize++] = v;
    // v.cost[totsize++] = cost;

    for(auto nxt : G[v]){
        if(nxt == p) continue;
        if(nxt != G[v][0]) chainNo ++;
        hld(nxt, v);
    }
    out[v] = totsize;
}

```

```

void init_hld(int V){
    chainNo = totsize = 0;
    fill(chainHead, chainHead + n, -1);
    dfs(0, -1, 0);
    hld(0, -1);
    fill(sum, sum + 4 * n, 0);
    fill(add, add + 4 * n, 0);
}

```

```

int query(int pos, int x, int l, int r) {
    // Segment Tree Query
}

```

```

void update(int a, int b, int x, int l, int r, int val) {
    // Segment Tree Update
}

```

```

void update_to_root(int v, int val) {
    while(chainInd[v] != chainInd[0]){

```

```
        update(totPos[chainHead[chainInd[v]]], totPos[v], 0, 0,
               n-1, val);
        v = par[chainHead[chainInd[v]]];
    }
    update(totPos[0], totPos[v], 0, 0, n-1, val);
}
```

LCA.cpp

<bits/stdc++.h>761d7e, 54 lines

```
using namespace std;

typedef long long ll;
typedef pair<int, int> P;

const int N = 100050, INF = (int)1e9;
const int LOG_N = 17;
int root = 0;
int n,m,a,b;

vector<int> G[N];
int parent[LOG_N][N];
int depth[N], par[N];

void dfs(int v, int p, int d){
    parent[0][v] = p;
    depth[v] = d;
    for(int nxt : G[v]){
        if(nxt != p) dfs(nxt, v, d+1);
    }
}

void init(int V){
    dfs(root, -1, 0);

    for(int k = 0; k+1 < LOG_N; k++){
        for(int v = 0; v < V; v++){
            if(parent[k][v] < 0) parent[k+1][v] = -1;
            else parent[k+1][v] = parent[k][parent[k][v]];
        }
    }
}

int lca(int u, int v){
    if(depth[u] > depth[v]){int tmp = u; u = v; v = tmp;}
    for(int k = 0; k < LOG_N; k++){
        if((depth[v] - depth[u]) >> k & 1){
            v = parent[k][v];
        }
    }
    if(u == v) return u;
    for(int k = LOG_N - 1; k >= 0; k--){
        if(parent[k][u] != parent[k][v]){
            u = parent[k][u];
            v = parent[k][v];
        }
    }
    return parent[0][u];
}

int min_dis(int u, int v){
    int ca = lca(u, v);
    return depth[u] + depth[v] - 2 * depth[ca];
}
```

SCC.cpp

<bits/stdc++.h>c86624, 46 lines

using namespace std;

```
const int N = (int)2e5 + 500, INF = (int)1e9;
typedef pair<int, int> P;
typedef long long ll;

int n, m;
vector<int> G[N], rG[N], vs;
bool used[N];
int cmp[N];

void add_edge(int from, int to){
    G[from].push_back(to);
    rG[to].push_back(from);
}

void dfs(int v){
    used[v] = true;
    for(int nxt : G[v]){
        if(!used[nxt]) dfs(nxt);
    }
    vs.push_back(v);
}

void rdfs(int v, int k){
    used[v] = true;
    cmp[v] = k;
    for(int nxt : rG[v]){
        if(!used[nxt]) rdfs(nxt, k);
    }
}

int scc(){
    memset(used, 0, sizeof(used));
    vs.clear();
    for(int v = 0; v < n; v++){
        if(!used[v]) dfs(v);
    }
    memset(used, 0, sizeof(used));
    int k = 0;
    reverse(vs.begin(), vs.end());
    for(int v : vs){
        if(!used[v]) rdfs(v, k++);
    }
    return k;
}
```

Flows and Matching (5)

Dinic.cpp

<bits/stdc++.h>ec1c8c, 72 lines

```
using namespace std;

const int N = 100050;
const int INF = (int)1e9;

struct edge{
    int to, cap, rev;
    edge(int _to, int _cap, int _rev){
        to = _to, cap = _cap, rev = _rev;
    }
};

// Finding max flow in O(V^2 * E)
struct Dinic {
    vector<edge> G[N];
    int level[N], iter[N];

```

```
void add_edge(int from, int to, int cap){
    G[from].push_back(edge(to, cap, G[to].size()));
    G[to].push_back(edge(from, 0, G[from].size() - 1));
}

void bfs(int s){
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while(!que.empty()){
        int v = que.front(); que.pop();
        for(int i = 0; i < G[v].size(); i++){
            edge &e = G[v][i];
            if(e.cap > 0 && level[e.to] < 0){
                level[e.to] = level[v] + 1;
                que.push(e.to);
            }
        }
    }

int dfs(int v, int t, int f){
    if(v == t) return f;

    for(int &i = iter[v]; i < G[v].size(); i++){
        edge &e = G[v][i];
        if(e.cap > 0 && level[v] < level[e.to]){
            int d = dfs(e.to, t, min(e.cap, f));
            if(d > 0){
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t){
    int flow = 0;
    for(;;){
        bfs(s);
        if(level[t] < 0) return flow;
        memset(iter, 0, sizeof(iter));
        int f;
        while((f = dfs(s, t, INF)) > 0){
            flow += f;
        }
    }
}
} dinic;
```

FordFulkerson.cpp

<cstdio>, <iostream>, <vector>2e195b, 47 lines

```
using namespace std;

const int N = 3100, INF = (int)1e9;

struct edge{
    int to, cap, rev;
    edge(int _to, int _cap, int _rev){
        to = _to, cap = _cap, rev = _rev;
    }
};

int n,m;
vector<edge> G[N];

```

```

bool used[N];

void add_edge(int from, int to, int cap){
    G[from].push_back(edge(to, cap, G[to].size()));
    G[to].push_back(edge(from, 0, G[from].size() - 1));
}

int dfs(int v, int t, int f){
    if(v == t) return f;
    used[v] = true;

    for(int i = 0; i < G[v].size(); i++){
        edge &e = G[v][i];
        if(!used[e.to] && e.cap > 0){
            int d = dfs(e.to, t, min(e.cap, f));
            if(d > 0){
                e.cap -= d;
                G[e.to][e.rev].cap += d;
                return d;
            }
        }
    }
    return 0;
}

int max_flow(int s, int t){
    int f = 0;
    for(;;){
        fill(used, used + N, false);
        int d = dfs(s, t, INF);
        if(d == 0) return f;
        f += d;
    }
}

```

MaximumWeightMatching.cpp

<bits/stdc++.h> 47d906, 48 lines

```

using namespace std;
const int MAXN = 2010;
const int oo = 1000000007;

int dist[MAXN][MAXN];
// Finding the minimum weight prefect matching (of size n) in O(N^3)
// The dist matrix is 1-indexed.
int hungarian(int n, int m){
    vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
    for(int i = 1; i <= n; i++){
        p[0] = i;
        int j0 = 0;
        vector<int> minv(m + 1, oo);
        vector<char> used(m + 1, false);
        do{
            used[j0] = true;
            int i0 = p[j0], delta = oo, j1;
            for(int j = 1; j <= m; j++){
                if(!used[j]){
                    int cur = dist[i0][j] - u[i0] - v[j];
                    if(cur < minv[j]){
                        minv[j] = cur;
                        way[j] = j0;
                    }
                }
                if(minv[j] < delta){
                    delta = minv[j];
                    j1 = j;
                }
            }
        } while(j1 > 0);
        for(int j = 1; j <= m; j++){
            if(way[j] > 0){
                p[j] = p[way[j]];
                v[j] += delta;
                u[p[j]] -= delta;
            }
        }
        j0 = j1;
    }
}

```

```

for(int j = 0; j <= m; j++){
    if(used[j]){
        u[p[j]] += delta;
        v[j] -= delta;
    } else {
        minv[j] -= delta;
    }
}
j0 = j1;
} while (p[j0] != 0);
do{
    int j1 = way[j0];
    p[j0] = p[j1];
    j0 = j1;
} while(j0);
}
return -v[0];
}

```

MinimumCostMaxFlow.cpp

<iostream>, <cstdio>, <vector>, <queue> dfc98e, 61 lines

```

using namespace std;

const int N = 55, MAX_V = 105, INF = (int)1e9;

typedef pair<int, int> P;
struct edge{int to, cap, cost, rev;;};

// Finding Min Cost Max Flow in min(O(F * E * log(V)), O(F * V * V));
struct MincostFlow {
    int V; //Please set V!!!
    vector<edge> G[MAX_V];
    int h[MAX_V];
    int dist[MAX_V];
    int prevv[MAX_V], preve[MAX_V];

    void add_edge(int from, int to, int cap, int cost){
        G[from].push_back((edge){to, cap, cost, (int)G[to].size()});
        G[to].push_back((edge){from, 0, -cost, (int)G[from].size() - 1});
    }

    int min_cost_flow(int s, int t, int f){
        int res = 0;
        fill(h, h + V, 0);
        while(f > 0){
            priority_queue<P, vector<P>, greater<P> > que;
            fill(dist, dist + V, INF);
            dist[s] = 0;
            que.push(P(0, s));
            while(!que.empty()){
                P p = que.top(); que.pop();
                int v = p.second;
                if(dist[v] < p.first) continue;
                for(int i = 0; i < G[v].size(); i++){
                    edge &e = G[v][i];
                    if(e.cap > 0 && dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]){
                        dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
                        prevv[e.to] = v;
                        preve[e.to] = i;
                        que.push(P(dist[e.to], e.to));
                    }
                }
            }
        }
        return res;
    }
}

```

```

if(dist[t] == INF) return -1;
for(int v = 0; v < V; v++) h[v] += dist[v];

int d = f;
for(int v = t; v != s; v = prevv[v]){
    d = min(d, G[prevv[v]][preve[v]].cap);
}
f -= d;
res += d * h[t];
for(int v = t; v != s; v = prevv[v]){
    edge &e = G[prevv[v]][preve[v]];
    e.cap -= d;
    G[v][e.rev].cap += d;
}
}
return res;
}

} mf;

```

Strings (6)

AhoCorasick.cpp

Description: Init, and insert strings, and then build.

<bits/stdc++.h> 263a36, 66 lines

```

using namespace std;

const int M = (int)5e5 + 500;

struct Trie{
    static const int B = 26;

    int next[M][B], fail[M], end[M];
    int root, L;
    int newnode(){
        for(int i = 0; i < B; i++) next[L][i] = -1;
        end[L++] = 0;
        return L - 1;
    }
    // Please do initialize it !!!
    void init(){
        L = 0;
        root = newnode();
    }
    void insert(const string &buf){
        int len = buf.length();
        int now = root;
        for(int i = 0; i < len; i++){
            if(next[now][buf[i] - 'a'] == -1) next[now][buf[i] - 'a'] = newnode();
            now = next[now][buf[i] - 'a'];
        }
        end[now] ++;
    }
    void build(){
        queue<int> Q;
        fail[root] = root;
        for(int i = 0; i < B; i++){
            if(next[root][i] == -1) next[root][i] = root;
            else{
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        }
        while(!Q.empty()){
            int now = Q.front();
            Q.pop();
        }
    }
}

```

```
        for(int i = 0; i < B; i++){
            if(next[now][i] == -1) next[now][i] = next[fail
                [now]][i];
            else{
                fail[next[now][i]] = next[fail[now]][i];
                Q.push(next[now][i]);
            }
        }
    }
}

int query(const string &buf){
    int len = buf.length();
    int now = root;
    int res = 0;
    for(int i = 0; i < len; i++){
        now = next[now][buf[i]-'a'];
        int temp = now;
        while(temp != root){
            res += end[temp];
            end[temp] = 0;
            temp = fail[temp];
        }
    }
    return res;
}

}
} Aho;
```

KMP.cpp

<bits/stdc++.h>3a7af0, 28 lines

```
using namespace std;

const int N = (int)1e6 + 500, M = (int)1e4 + 500;

string s, t;
int f[M];

void getnext(){
    int m = t.length();
    f[0] = 0; f[1] = 0;
    for(int i = 1; i < m; i++){
        int j = f[i];
        while(j && t[i] != t[j]) j = f[j];
        f[i+1] = t[i] == t[j] ? j + 1 : 0;
    }
}

int find(){
    int n = s.length(), m = t.length();
    int res = 0;
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j && t[j] != s[i]) j = f[j];
        if(t[j] == s[i]) j++;
        if(j == m) res ++, j = f[j];
    }
    return res;
}

}
```

Manacher.cpp

<iostream>, <cstdio>, <algorithm>, <cstring>2065b8, 50 lines

```
struct Manacher {
    string s, sn;
    int p[2*N];

    int Init()
    {
        int len = s.length();
```

KMP Manacher PolynomialHashing SAM

```
        sn = "$#";
        int j = 2;

        for (int i = 0; i < len; i++)
        {
            sn.push_back(s[i]);
            sn.push_back('#');
        }

        sn.push_back('\0');

        return sn.length();
    }

    int run()
    {
        int len = Init();
        int max_len = -1;

        int id = 0;
        int mx = 0;

        for (int i = 1; i < len; i++)
        {
            if (i < mx)
                p[i] = min(p[2 * id - i], mx - i);
            else
                p[i] = 1;

            while (sn[i - p[i]] == sn[i + p[i]])
                p[i]++;

            if (mx < i + p[i])
            {
                id = i;
                mx = i + p[i];
            }

            max_len = max(max_len, p[i] - 1);
        }
        return max_len;
    }
} mnc;
```

PolynomialHashing.cpp

<bits/stdc++.h>3a4d96, 64 lines

```
using namespace std;

typedef long long ll;
typedef pair<int, int> P;
const int mods[4] = {(int)1e9 + 7, (int)1e9 + 9, (int)1e9 + 21,
    (int)1e9 + 33};
const int N = (int)2e5 + 50;

string s, t;
int p = 37;
ll pw[4][N];

struct hs {
    ll val[4];

    hs() { fill(val, val + 4, 0); }

    hs(ll a, ll b, ll c, ll d) {
        val[0] = a, val[1] = b, val[2] = c, val[3] = d;
    }

    bool operator<(const hs &other) const {
```

```
        for (int i = 0; i < 4; i++) if (val[i] != other.val[i])
            return val[i] < other.val[i];
        return false;
    }

    bool operator==(const hs &other) const {
        for (int i = 0; i < 4; i++) if (val[i] != other.val[i])
            return false;
        return true;
    }

    hs operator + (const hs &other) const{
        hs res;
        for(int i = 0; i < 4; i++) res.val[i] = ( val[i] +
            other.val[i]) % mods[i];
        return res;
    }

    hs operator - (const hs &other) const{
        hs res;
        for(int i = 0; i < 4; i++) res.val[i] = (val[i] - other
            .val[i] + mods[i]) % mods[i];
        return res;
    }

    hs operator ^ (const int pwi) const {
        hs res;
        for(int i = 0; i < 4; i++){
            res.val[i] = (val[i] * pw[i][pwi]) % mods[i];
        }
        return res;
    }

    void add(int x, int pwi){
        for(int i = 0; i < 4; i++) {
            val[i] = (val[i] + x * pw[i][pwi]) % mods[i];
            if(val[i] < 0) val[i] += mods[i];
        }
    }

};

void init() {
    for(int t = 0; t < 4; t++){
        pw[t][0] = 1;
        for(int i = 1; i < N; i++) pw[t][i] = pw[t][i-1] * p %
            mods[t];
    }
}
```

SAM.cpp

<bits/stdc++.h>b6e7ac, 60 lines

```
using namespace std;

const int N = (int)1e5 + 50, B = 256;

struct state {
    int len, link;
    int next[B];
};

struct SAM {
    const static int MAXLEN = (int)1005;
    state st[MAXLEN * 2];
    int sz, last;

    void sam_init() {
        st[0].len = 0;
        st[0].link = -1;
```

```

memset(st[0].next, -1, sizeof(st[0].next));
sz = 1;
last = 0;
}

void sam_extend(int c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    memset(st[cur].next, -1, sizeof(st[cur].next));
    int p = last;
    while(p != -1 && st[p].next[c] == -1) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if(p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if(st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            memcpy(st[clone].next, st[q].next, sizeof(st[q].next));
            st[clone].link = st[q].link;
            while(p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

int calc() {
    int res = 0;
    for(int v = 0; v < sz; v++) {
        if(st[v].link != -1) res += st[v].len - st[st[v].link].len;
    }
    return res;
}
} sam;

```

SuffixArray.cpp

<bits/stdc++.h> 24a7e4, 68 lines

using namespace std;

```

typedef long long ll;
const int N = (int)1e5 + 50;

```

int n;

```

struct SA {
    int n;
    // ht[i] = lcp(suffix[sa[i]], suffix[sa[i-1]])
    int rk[N], sa[N], ht[N];

    void build(string str) {
        n = str.length();
        str = " " + str;
        static int set[N], a[N];
        for(int i = 1; i <= n; i++) set[i] = str[i];
        sort(set + 1, set + n + 1);
        int *end = unique(set + 1, set + n + 1);
    }
};

```

```

for(int i = 1; i <= n; i++) a[i] = (int)(lower_bound(
    set + 1, end, str[i]) - set);

static int fir[N], sec[N], tmp[N], buc[N];
fill(buc, buc + n + 1, 0);
for(int i = 1; i <= n; i++) buc[a[i]]++;
for(int i = 1; i <= n; i++) buc[i] += buc[i-1];
for(int i = 1; i <= n; i++) rk[i] = buc[a[i]-1] + 1;

for(int t = 1; t <= n; t *= 2) {
    for(int i = 1; i <= n; i++) fir[i] = rk[i];
    for(int i = 1; i <= n; i++) sec[i] = i + t > n ? 0
        : rk[i + t];

    fill(buc, buc + n + 1, 0);
    for(int i = 1; i <= n; i++) buc[sec[i]]++;
    for(int i = 1; i <= n; i++) buc[i] += buc[i - 1];
    for(int i = 1; i <= n; i++) tmp[n - (--buc[sec[i]])] = i;

    bool unique = true;
    for (int j = 1, i, last = 0; j <= n; j++) {
        i = sa[j];
        if (!last) rk[i] = 1;
        else if (fir[i] == fir[last] && sec[i] == sec[
            last]) rk[i] = rk[last], unique = false;
        else rk[i] = rk[last] + 1;

        last = i;
    }

    if (unique) break;
}

for(int i = 1, k = 0; i <= n; i++) {
    if(rk[i] == 1) k = 0;
    else {
        if(k > 0) k--;
        int j = sa[rk[i]-1];
        while(i + k <= n && j + k <= n && a[i + k] == a
            [j + k]) k++;
    }
    ht[rk[i]] = k;
}
}
} sa;

```

ZAlgorithm.cpp

<bits/stdc++.h> 06f29b, 27 lines

using namespace std;

const int MAXN = (int)1e6 + 500;

```

string s;
int z[MAXN], cnt[MAXN];

```

```

// Examples:
// str = "aabaacd"
// z[] = {x, 1, 0, 2, 1, 0, 0}
// str = "abababab"

```

```
// z[] = {x, 0, 6, 0, 4, 0, 2, 0}
```

```

void getZarr(string str)
{
    memset(z, 0, sizeof(z));
    int n = str.length();
    for(int i = 1, l = 0, r = 0; i < n; ++i){
        if(i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while(i + z[i] < n && str[z[i]] == str[i + z[i]])
            ++z[i];
        if(i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
}

```

DigitDP.cpp

Description: 1

<bits/stdc++.h> 47cc69, 27 lines

using namespace std;

const int B = 20;

```

int dp[10000][B];
int bit[B], b;
int pw2[B];
int A, B;

```

```

int get(int rem, int d, bool flag){
    if(rem < 0) return 0;
    if(d == -1) return rem >= 0;
    if(!flag && ~dp[rem][d]) return dp[rem][d];
    int lim = flag ? bit[d] : 9;
    int res = 0;
    for(int i = 0; i <= lim; i++){
        res += get(rem - i * pw2[d], d - 1, flag && lim == i);
    }
    return flag ? res : dp[rem][d] = res;
}

```

```

int solve(int x){
    b = 0;
    int t = x;
    while(t > 0){bit[b++] = t % 10; t /= 10;}
    return get(A, b-1, true);
}

```

Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	

Computation of binomial coefficients
Pigeon-hole principle
Inclusion/exclusion
Catalan number
Pick's theorem
Number theory
Integer parts
Divisibility
Euclidean algorithm
Modular arithmetic
* Modular multiplication
* Modular inverses
* Modular exponentiation by squaring
Chinese remainder theorem
Fermat's little theorem
Euler's theorem
Phi function
Frobenius number
Quadratic reciprocity
Pollard-Rho
Miller-Rabin
Hensel lifting
Vieta root jumping
Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences

Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array
Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree