Khulna University

KU SecondTeam

Contents

Contest (1)

.bashrc

12 lines

```
{
    "shell_cmd": "g++ -std=c++14 \"$file\" -o \"$file_base_name\" &&
        timeout 4s ./$file_base_name < inputf.in > outputf.in",
    "selector": "source.c, source.c++",
    "working_dir": "$file_path",
    "file_regex": "^(.+\\.cpp):(\\d+):(\\d+): (.*)$",
    "variants": [
        {
            "name": "Run",
            "shell_cmd": "timeout 4s ./$file_base_name < inputf.in >
                outputf.in"
        }
    ]
}
```

Data structures (2)

oSet.h

Description: Ordered Set 3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

<bits/stdc++.h>, <ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> d0c877, 18 lines

```
using namespace __gnu_pbds;
using namespace std;
template <typename T> using o_set = tree<T, null_type, less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  o_set<int> se;
  se.insert(4);
  se.insert(2);
  se.insert(5);
  // sorted set se = [2, 4, 5]
  cout << se.order_of_key(5) << '\n'; // number of elements < 5
  cout << se.order_of_key(6) << '\n'; // number of elements < 6
  cout << (*se.find_by_order(1)) << '\n'; // if you imagine this as a 0-
      indexed vector, what is se[1]?
  return 0;
}
```

DSU.h

Description: DSU to find parent. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

e22ea4, 16 lines

```
void make_set(int v) {
    parent[v] = v;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b)
        parent[b] = a;
}
```

HashMap.h

Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

d77092, 7 lines

```
#include <bits/extc++.h>
// To use most bits rather than just the lowest ones:
struct chash { // large odd number for C
  const uint64_t C = ll(4e18 * acos(0)) | 71;
  ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
__gnu_pbds::gp_hash_table<ll,int,chash> h({},{},{},{},{1<<16});
```

LazySegmentTree.h

Description: Segment tree. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

<bits/stdc++.h> c465a1, 62 lines

```
using namespace std;

const int N = 5e5 + 9;
int a[N];
struct ST {
  #define lc (n << 1)
  #define rc ((n << 1) | 1)
  long long t[4 * N], lazy[4 * N];
  ST() {
    memset(t, 0, sizeof t);
    memset(lazy, 0, sizeof lazy);
  }
  inline void push(int n, int b, int e) {
    if (lazy[n] == 0) return;
    t[n] = t[n] + lazy[n] * (e - b + 1);
    if (b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline long long combine(long long a,long long b) {
    return a + b;
  }
  inline void pull(int n) {
    t[n] = t[lc] + t[rc];
  }
  void build(int n, int b, int e) {
    lazy[n] = 0;
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, long long v) {
    push(n, b, e);
    if (j < b || e < i) return;
    if (i <= b && e <= j) {
      lazy[n] = v; //set lazy
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  long long query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if (i > e || b > j) return 0; //return null
    if (i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc, mid + 1, e, i, j));
  }
};
int32_t main() {

}
```

LCA.h

Description: LEAST COMMON ANCESTOR Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

c5424a, 65 lines

```
#define mx 100002
int L[mx]; // Level of each node
int P[mx][22]; // Sparse table
int T[mx]; // Parent array
vector<int> g[mx];

void dfs(int from, int u, int dep) {
    T[u] = from;
    L[u] = dep;
    for (int i = 0; i < (int)g[u].size(); i++) {
        int v = g[u][i];
        if (v == from) continue;
        dfs(u, v, dep + 1);
    }
}

int lca_query(int N, int p, int q) { // N = number of nodes
    int tmp, log, i;

    if (L[p] < L[q])
        tmp = p, p = q, q = tmp;

    log = 1;
    while (1) {
        int next = log + 1;
        if ((1 << next) > L[p]) break;
        log++;
    }

    for (i = log; i >= 0; i--)
        if (L[p] - (1 << i) >= L[q])
```

```
            p = P[p][i];

    if (p == q)
        return p;

    for (i = log; i >= 0; i--)
        if (P[p][i] != -1 && P[p][i] != P[q][i])
            p = P[p][i], q = P[q][i];

    return T[p];
}

void lca_init(int N) {
    memset(P, -1, sizeof(P)); // Initialize all values to -1
    int i, j;
    for (i = 0; i < N; i++)
        P[i][0] = T[i];

    for (j = 1; (1 << j) < N; j++)
        for (i = 0; i < N; i++)
            if (P[i][j - 1] != -1)
                P[i][j] = P[P[i][j - 1]][j - 1];
}

int main(void) {
    g[0].push_back(1);
    g[0].push_back(2);
    g[2].push_back(3);
    g[2].push_back(4);
    dfs(0, 0, 0);
    lca_init(5);
    printf("%d\n", lca_query(5, 3, 4));
    return 0;
}
```

RMQ.h

Description: Find minimum range query Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

a5c3c2, 11 lines

```
int st[K + 1][MAXN];

std::copy(array.begin(), array.end(), st[0]);

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);

int i = log2[R - L + 1];
int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
```

SquareRoot.h

Description: SQRT decomposition Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

88b98e, 35 lines

```
int query(int input[], int segment_size, int l, int r) {
    int sum = 0;

    //loop the first segment
    //until we reach r or a starting index

    while (l < r && l % segment_size != 0) {
  sum += input[l];
  l++;
    }

    //Loop until we reach
    //segment that contains r
    while (l + segment_size <= r) {
        sum += segment[l / segment_size];
        l += segment_size;
    }

    //loop until r
    while (l<=r) {
        sum += input[l];
        l++;
    }

    return sum;
}

void update(int input[], int segment_size, int i, int val) {
    int segment_no = i / segment_size;

    segment[segment_no] -= input[i];
    segment[segment_no] += val;
    input[i] = val;
}
```

Trie.h

Description: TRIE Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).

7b7a52, 66 lines

```
struct node {
    bool endmark;
    node* next[26 + 1];
    node()
    {
        endmark = false;
        for (int i = 0; i < 26; i++)
            next[i] = NULL;
    }
```

```
} * root;
void insert(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            curr->next[id] = new node();
        curr = curr->next[id];
    }
    curr->endmark = true;
}
bool search(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            return false;
        curr = curr->next[id];
    }
    return curr->endmark;
}
void del(node* cur)
{
    for (int i = 0; i < 26; i++)
        if (cur->next[i])
            del(cur->next[i]);

    delete (cur);
}
int main()
{

    puts("ENTER NUMBER OF WORDS");
    root = new node();
    int num_word;
    cin >> num_word;
    for (int i = 1; i <= num_word; i++) {
        char str[50];
        scanf("%s", str);
        insert(str, strlen(str));
    }
    puts("ENTER NUMBER OF QUERY";);
    int query;
    cin >> query;
    for (int i = 1; i <= query; i++) {
        char str[50];
        scanf("%s", str);
        if (search(str, strlen(str)))
            puts("FOUND");
        else
            puts("NOT FOUND");
    }
    del(root);
    return 0;
}
```

Numerical (3)

3.1 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFTMod.

Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ ($\sim 1s$ for $N = 2^{22}$)

96096f, 54 lines

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
```

```
    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}
vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

Number theory (4)

4.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set `mod` to some number first and then you can use the structure.

"euclid.h" 35bfea, 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime.

6f684f, 3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
rep(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModPow.h

b83e45, 8 lines

```
const ll mod = 1000000007; // faster if const

ll modpow(ll b, ll e) {
  ll ans = 1;
  for (; e; b = b * b % mod, e /= 2)
    if (e & 1) ans = ans * b % mod;
  return ans;
}
```

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.

Time: $\mathcal{O}(\sqrt{m})$

c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.

`modsum(to, c, k, m)` $= \sum_{i=0}^{\text{to}-1} (ki + c)\%m$. `divsum` is similar but for floored division.

Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
```

```
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for $0 \le a, b \le c \le 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ for `modmul`, $\mathcal{O}(\log b)$ for `modpow`

bbbd8f, 11 lines

```
typedef unsigned long long ull;
ull modmul(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(1.L / M * a * b);
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
  ull ans = 1;
  for (; e; b = modmul(b, b, mod), e /= 2)
    if (e & 1) ans = modmul(ans, b, mod);
  return ans;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

4.2 Primality

FastEratosthenes.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

6b2912, 20 lines

```
const int LIM = 1e6;
bitset<LIM> isPrime;
vi eratosthenes() {
  const int S = (int)round(sqrt(LIM)), R = LIM / 2;
  vi pr = {2}, sieve(S+1); pr.reserve(int(LIM/log(LIM)*1.1));
  vector<pii> cp;
  for (int i = 3; i <= S; i += 2) if (!sieve[i]) {
    cp.push_back({i, i * i / 2});
    for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
  }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto &[p, idx] : cp)
      for (int i=idx; i < S+L; idx = (i+=p)) block[i-L] = 1;
    rep(i,0,min(S, R - L))
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

MillerRabin.h

Description: Deterministic Miller-Rabin primality test. Guaranteed to work for numbers up to $7 \cdot 10^{18}$; for larger numbers, use Python and extend A randomly.
Time: 7 times the complexity of $a^b \mod c$.

"ModMulLL.h" 60dcd1, 12 lines

```
bool isPrime(ull n) {
  if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
  ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
      s = __builtin_ctzll(n-1), d = n >> s;
  for (ull a : A) {   // ^ count trailing zeroes
    ull p = modpow(a%n, d, n), i = s;
    while (p != 1 && p != n - 1 && a % n && i--)
      p = modmul(p, p, n);
    if (p != n-1 && i != s) return 0;
  }
  return 1;
}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" d8d98d, 18 lines

```
ull pollard(ull n) {
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  auto f = [&](ull x) { return modmul(x, x, n) + i; };
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

4.3 Divisibility

euclid.h

Description: Finds two integers x and y, such that $ax + by = \gcd(a, b)$. If you just need gcd, use the built in __gcd instead. If a and b are coprime, then x is the inverse of $a \pmod b$.

33ba8f, 5 lines

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

CRT.h

Description: Chinese Remainder Theorem.

`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \le x < \text{lcm}(m, n)$. Assumes $mn < 2^{62}$.

Time: $\log(n)$

"euclid.h" 04d93a, 7 lines

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  ll x, y, g = euclid(m, n, x, y);
  assert((a - b) % g == 0); // else no solution
  x = (b - a) % n * x % n / g * m + a;
  return x < 0 ? x + m*n/g : x;
}
```

4.3.1 Bézout's identity

For $a \neq, b \neq 0$, then $d = gcd(a, b)$ is the smallest positive integer for which there are integer solutions to

$$ax + by = d$$

If (x, y) is one solution, then all solutions are given by

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.h

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\le n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} ... (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \le k \le n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod n$.

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod p$ $\forall a$.

cf7d6d, 8 lines

```
const int LIM = 5000000;
int phi[LIM];

void calculatePhi() {
  rep(i,0,LIM) phi[i] = i&1 ? i : i/2;
  for (int i = 3; i < LIM; i += 2) if(phi[i] == i)
    for (int j = i; j < LIM; j += i) phi[j] -= phi[j] / i;
}
```

4.4 Pythagorean Triples

The Pythagorean triples are uniquely generated by

$$a = k \cdot (m^2 - n^2), \ \ b = k \cdot (2mn), \ \ c = k \cdot (m^2 + n^2),$$

with $m > n > 0$, $k > 0$, $m \perp n$, and either m or n even.

4.5 Primes

$p = 962592769$ is such that $2^{21} \mid p - 1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.6 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

4.7 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ 1 & n \text{ has even number of prime factors} \\ -1 & n \text{ has odd number of prime factors} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

Combinatorial (5)

5.1 Permutations

5.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11		12		13		14	15	16	17
$n!$	4.0e7		4.8e8		6.2e9		8.7e10	1.3e12	2.1e13	3.6e14
n	20		25		30	40	50	100	150	171
$n!$	2e18		2e25		3e32	8e47	3e64	9e157	6e262	>DBL_MAX

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.

Time: $\mathcal{O}(n)$

044568, 6 lines

```
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
    use |= 1 << x;                    // (note: minus, not ~!)
  return r;
}
```

5.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

5.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1) + D(n-2)) = nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

5.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|} \sum_{g \in G} |X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n} \sum_{k|n} f(k)\phi(n/k).$$

5.2 Partitions and subsets

5.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

5.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

5.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$.

<bits/stdc++.h> 15df88, 37 lines

```
using namespace std;

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

int power(long long n, long long k) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
int f[N], invf[N];
int nCr(int n, int r) {
  if (n < r or n < 0) return 0;
  return 1LL * f[n] * invf[r] % mod * invf[n - r] % mod;
}
int nPr(int n, int r) {
  if (n < r or n < 0) return 0;
  return 1LL * f[n] * invf[n - r] % mod;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  f[0] = 1;
  for (int i = 1; i < N; i++) {
    f[i] = 1LL * i * f[i - 1] % mod;
  }
  invf[N - 1] = power(f[N - 1], mod - 2);
  for (int i = N - 2; i >= 0; i--) {
    invf[i] = 1LL * invf[i + 1] * (i + 1) % mod;
  }
  cout << nCr(6, 2) << '\n';
  cout << nPr(6, 2) << '\n';
  return 0;
}
```

5.3 General purpose numbers

5.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able).
$B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$
$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

5.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

5.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$
$$E(n,0) = E(n,n-1) = 1$$
$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

5.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$
$$S(n,1) = S(n,n) = 1$$
$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

5.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

5.3.6 Labeled unrooted trees

on n vertices: n^{n-2}
on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

5.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (6)

6.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max|w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] =$ `inf` if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, `inf` if no path, or `-inf` if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$

<bits/stdc++.h> ef1438, 50 lines

```
using namespace std;

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
vector<int> ord;
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
  ord.push_back(u);
}
```

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

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

  // print the order
  for (auto u: ord) cout << u << ' ';
  cout << '\n';
  return 0;
}
// https://cses.fi/problemset/task/1679
```

6.2 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: `scc(graph, [&](vi& v) { ... })` visits all components in reverse topological order. `comp[i]` holds the component index of a node (a component only has edges to components with lower index). `ncomps` will contain the number of components.

Time: $\mathcal{O}(E+V)$

<bits/stdc++.h> ea07dd, 62 lines

```
using namespace std;
const int N = 3e5 + 9;
// given a directed graph return the minimum number of edges to be added
    so that the whole graph become an SCC
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //G is the condensed graph
void dfs1(int u) {
  vis[u] = 1;
  for(auto v: g[u]) if(!vis[v]) dfs1(v);
  vec.push_back(u);
}

vector<int> comp;
void dfs2(int u) {
  comp.push_back(u);
  vis[u] = 1;
  for(auto v: r[u]) if(!vis[v]) dfs2(v);
}

int idx[N], in[N], out[N];
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    r[v].push_back(u);
  }
  for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
  reverse(vec.begin(), vec.end());
  memset(vis, 0, sizeof vis);
  int scc = 0;
  for(auto u: vec) {
    if(!vis[u]) {
      comp.clear();
      dfs2(u);
      scc++;
      for(auto x: comp) idx[x]=scc;
    }
  }
  for(int u = 1; u <= n; u++) {
    for(auto v: g[u]) {
      if(idx[u] != idx[v]) {
        in[idx[v]]++, out[idx[u]]++;
        G[idx[u]].push_back(idx[v]);
      }
    }
  }
```

```
  int needed_in=0, needed_out=0;
  for(int i = 1; i <= scc; i++) {
    if(!in[i]) needed_in++;
    if(!out[i]) needed_out++;
  }
  int ans = max(needed_in, needed_out);
  if(scc == 1) ans = 0;
  cout << ans << '\n';
  return 0;
}
```

bridge.h

Description: Articulation Point + bridge representing the maximal clique.

Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

fe94e5, 55 lines

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u

int dfsAP(int u, int p) {
  int children = 0;
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back through the same
        path.
                          // if we go back is because we found another way
                              back
    if (!disc[v]) { // if V has not been discovered before
      children++;
      dfsAP(v, u); // recursive DFS call
      if (disc[u] <= low[v]) // condition #1
        ap[u] = 1;
      low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
    } else // if v was already discovered means that we found an ancestor
      low[u] = min(low[u], disc[v]); // finds the ancestor with the least
          discovery time
  }
  return children;
}

void AP() {
  ap = low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      ap[u] = dfsAP(u, u) > 1; // condition #2
}
vector<pair<int, int>> br;

int dfsBR(int u, int p) {
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back through the same
        path.
                          // if we go back is because we found another way
                              back
    if (!disc[v]) { // if V has not been discovered before
      dfsBR(v, u);  // recursive DFS call
      if (disc[u] < low[v]) // condition to find a bridge
        br.push_back({u, v});
      low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
    } else // if v was already discovered means that we found an ancestor
      low[u] = min(low[u], disc[v]); // finds the ancestor with the least
          discovery time
  }
}

void BR() {
  low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      dfsBR(u, u)
}
```

6.3 Trees

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.

Time: $\mathcal{O}(N \log N + Q)$

<bits/stdc++.h> ce5352, 56 lines

```
using namespace std;

const int N = 3e5 + 9, LG = 18;

vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] = par[par[u][i - 1]][i - 1];
  for (auto v: g[u]) if (v != p) {
    dfs(v, u);
    sz[u] += sz[v];
  }
}
```

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

6.4 Math

6.4.1 Number of Spanning Trees

Create an $N \times N$ matrix `mat`, and for each edge $a \to b \in G$, do `mat[a][b]--`, `mat[b][b]++` (and `mat[b][a]--`, `mat[a][a]++` if G is undirected). Remove the ith row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

6.4.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even and for every $k = 1 \ldots n$,

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

Geometry (7)

7.1 Geometric primitives

Point.h

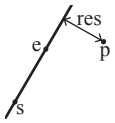
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.) 47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
```

```
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

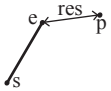


"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



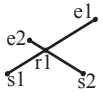
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



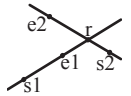
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage:
```
auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" 3af81c, 9 lines

```
template<class P>
```

```
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

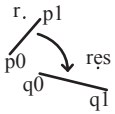
OnSegment.h

Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h



Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

LineProjectionReflection.h

Description: Projects point p onto line ab. Set refl=true to get reflection of point p across line ab instead. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.

"Point.h" b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
  P v = b - a;
  return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage: `vector<Angle> v = {w[0], w[0].t360() ...}; // sorted`
`int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }`
`// sweeps j such that (j-i) represents the number of positively oriented triangles with vertices at 0 and i`

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

7.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
```

```
  return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

CircleLine.h

Description: Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h" e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
  P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
  double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
  if (h2 < 0) return {};
  if (h2 == 0) return {p};
  P h = ab.unit() * sqrt(h2);
  return {p - h, p + h};
}
```

CirclePolygonIntersection.h

Description: Returns the area of the intersection of a circle with a ccw polygon.

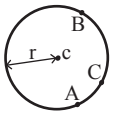
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

Description:
The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.

Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

7.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`

Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

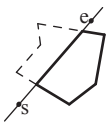
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h

Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.



Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonUnion.h

Description: Calculates the area of the union of n polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

Time: $\mathcal{O}\left(N^2\right)$, where N is the total number of points

"Point.h", "sideOf.h" 3931c6, 33 lines

```
typedef Point<double> P;
double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) {
  double ret = 0;
  rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
    P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
    vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
    rep(j,0,sz(poly)) if (i != j) {
      rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
          double sa = C.cross(D, A), sb = C.cross(D, B);
          if (min(sc, sd) < 0)
            segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        } else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
          segs.emplace_back(rat(C - A, B - A), 1);
          segs.emplace_back(rat(D - A, B - A), -1);
        }
      }
    }
    sort(all(segs));
    for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0;
    int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
      if (!cnt) sum += segs[j].first - segs[j - 1].first;
```

```
      cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
  }
  return ret / 2;
}
```

ConvexHull.h

Description:
Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.

Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/-collinear points).

Time: $\mathcal{O}(n)$

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.

Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. `extrVertex` returns the point of a hull with the max projection onto a line.

Time: $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
```

```
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

Strings (8)

Zfunc.h

Description: z[i] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)
Time: $\mathcal{O}(n)$

ee09e2, 12 lines

```
vi Z(const string& S) {
  vi z(sz(S));
  int l = -1, r = -1;
  rep(i,1,sz(S)) {
    z[i] = i >= r ? 0 : min(r - i, z[i - l]);
    while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
      z[i]++;
    if (i + z[i] > r)
      l = i, r = i + z[i];
  }
  return z;
}
```

Hashing.h

Description: Self-explanatory methods for string hashing.

<bits/stdc++.h> bf11cb, 80 lines

```
using namespace std;

const int N = 1e6 + 9;
const int p1 = 137, mod1 = 127657753, p2 = 277, mod2 = 987654319;

int power(long long n, long long k, int mod) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}

int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % mod1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % mod2;
  }
  ip1 = power(p1, mod1 - 2, mod1);
  ip2 = power(p2, mod2 - 2, mod2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % mod1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 % mod2;
  }
}

pair<int, int> string_hash(string s) {
  int n = s.size();
  pair<int, int> hs({0, 0});
  for (int i = 0; i < n; i++) {
    hs.first += 1LL * s[i] * pw[i].first % mod1;
    hs.first %= mod1;
    hs.second += 1LL * s[i] * pw[i].second % mod2;
    hs.second %= mod2;
  }
  return hs;
}
pair<int, int> pref[N];
void build(string s) {
  int n = s.size();
  for (int i = 0; i < n; i++) {
    pref[i].first = 1LL * s[i] * pw[i].first % mod1;
    if (i) pref[i].first = (pref[i].first + pref[i - 1].first) % mod1;
    pref[i].second = 1LL * s[i] * pw[i].second % mod2;
    if (i) pref[i].second = (pref[i].second + pref[i - 1].second) % mod2;
  }
}
pair<int, int> get_hash(int i, int j) {
  assert(i <= j);
  pair<int, int> hs({0, 0});
```

```
  hs.first = pref[j].first;
  if (i) hs.first = (hs.first - pref[i - 1].first + mod1) % mod1;
  hs.first = 1LL * hs.first * ipw[i].first % mod1;
  hs.second = pref[j].second;
  if (i) hs.second = (hs.second - pref[i - 1].second + mod2) % mod2;
  hs.second = 1LL * hs.second * ipw[i].second % mod2;
  return hs;
}

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  prec();
  string a, b; cin >> a >> b;
  build(a);
  int ans = 0, n = a.size(), m = b.size();
  auto hash_b = string_hash(b);
  for (int i = 0; i + m - 1 < n; i++) {
    ans += get_hash(i, i + m - 1) == hash_b;
  }
  cout << ans << '\n';
  return 0;
}
```

Various (9)

9.1 Misc. algorithms

TernarySearch.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

78abec, 34 lines

```
// If only take Integers
  template <typename F> int find_min(int l, int r, const F &f) {
  while (r - l > 3) {
    int m1 = l + (r - l) / 3;
    int m2 = r - (r - l) / 3;
    f(m1) > f(m2) ? l = m1 : r = m2;
  }

  int res = l;
  for (int i = l + 1; i <= r; i++) {
    if (f(i) < f(res)) { res = i; }
  }

  return res;
}
// floating point
template <typename F> double find_min(double l, double r, double eps,
    const F &f) {
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    f(m1) > f(m2) ? l = m1 : r = m2;
  }

  return l;
}
// Binary search
  template <typename F> int find_min(int l, int r, const F &f) {
  while (l < r) {
    int m = (l + r) / 2;
    f(m) < f(m + 1) ? r = m : l = m + 1;
  }

  return l;
}
```

LIS.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

2932a0, 17 lines

```
template<class I> vi lis(const vector<I>& S) {
  if (S.empty()) return {};
  vi prev(sz(S));
  typedef pair<I, int> p;
  vector<p> res;
  rep(i,0,sz(S)) {
    // change 0 -> i for longest non-decreasing subsequence
    auto it = lower_bound(all(res), p{S[i], 0});
    if (it == res.end()) res.emplace_back(), it = res.end()-1;
    *it = {S[i], i};
    prev[i] = it == res.begin() ? 0 : (it-1)->second;
  }
  int L = sz(res), cur = res.back().second;
  vi ans(L);
  while (L--) ans[L] = cur, cur = prev[cur];
  return ans;
}
```

boundedknapsack.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

ddd2b6, 44 lines

```
const int mod = 100000007;
int coin[50], amount[50], dp[51][1001];
int coinChange(int n, int sum)
{
    if (sum == 0)
```

```
        return 1;
    if (n == 0)
        return 0;
    if (dp[n][sum] != -1)
        return dp[n][sum];
    dp[n][sum] = 0;
    for (int i = 0; i <= amount[n - 1] && i * coin[n - 1] <= sum; i++)
        dp[n][sum] += (coinChange(n - 1, sum - i * coin[n - 1]) % mod);
    return (dp[n][sum] % mod);
}
void solve()
{
    int n, sum;
    cin >> n >> sum;
    for (int i = 0; i < n; ++i)
        cin >> coin[i];
    for (int i = 0; i < n; ++i)
        cin >> amount[i];
    int bottom_up_dp[n + 1][sum + 1];
    for (int i = 0; i <= sum; ++i)
        bottom_up_dp[0][i] = 0;
    for (int i = 0; i <= n; ++i)
        bottom_up_dp[i][0] = 1;
    for (int i = 1; i <= n; ++i)
    {
        for (int j = 1; j <= sum; ++j)
        {
            bottom_up_dp[i][j] = 0;
            for (int k = 0; (k <= amount[i - 1]) && ((k * coin[i - 1]) <=
                j); ++k)
            {
                bottom_up_dp[i][j] += (bottom_up_dp[i - 1][j - k * coin[i
                    - 1]]);
                bottom_up_dp[i][j] %= mod;
            }
        }
    }
    cout << bottom_up_dp[n][sum] << '\n';
    memset(dp, -1, sizeof(dp));
    cout << coinChange(n, sum) << '\n';
}
```

dfslargevalueofnodes.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> 631c01, 24 lines

```
using namespace std;

map<int, vector<int>> adj;
set<int> visited;
void dfs(int n)
{
    if (visited.find(n) != visited.end())
        return;
    visited.insert(n);
    for (auto i : adj[n])
        dfs(i);
}
void solve(int testCaseNo)
{
    int n, u, v;
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        cin >> u >> v;
        adj[u].push_back(v);
    }
    dfs(n);
    cout << *visited.rbegin() << nl;
}
```

dsupathcompressionandorderbysize.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> 28c2f8, 30 lines

```
using namespace std;

int parent[1000001], size[1000001];
void initialize(int n)
{
    for (int i = 1; i <= n; ++i)
    {
        parent[i] = i;
        size[i] = 1;
    }
}
int find_set(int n)
{
    if (parent[n] == n)
        return n;
    return parent[n] = find_set(parent[n]);
}

void union_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);
    if (a != b)
    {
        if (size[a] < size[b])
```

```
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

inversemodusingextendedeuclid.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> 1c8e4a, 20 lines

```
using namespace std;

const int mod = 1000000007;

pair<int, int> Extended_Euclid(int a, int b) {
    if (b == 0)
        return {1, 0};
    pair<int, int> r = Extended_Euclid(b, a % b);
    return {r.second, r.first - r.second * (a / b)};
}

int inverseMod(int a) {
    pair<int, int> ans = Extended_Euclid(a, mod);
    return ans.first;
}

int main() {
    int a; cin >> a;
    cout << inverseMod(a);
}
```

segmenttree.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> 4c0f62, 53 lines

```
using namespace std;

const int N = 3e5 + 9;

int a[N];
struct ST {
  int t[4 * N];
  static const int inf = 1e9;
  ST() {
    memset(t, 0, sizeof t);
  }
  void build(int n, int b, int e) {
    if (b == e) {
      t[n] = a[b];
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    build(l, b, mid);
    build(r, mid + 1, e);
    t[n] = max(t[l], t[r]); // change this
  }
  void upd(int n, int b, int e, int i, int x) {
    if (b > i || e < i) return;
    if (b == e && b == i) {
      t[n] = x; // update
      return;
    }
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    upd(l, b, mid, i, x);
    upd(r, mid + 1, e, i, x);
    t[n] = max(t[l], t[r]); // change this
  }
  int query(int n, int b, int e, int i, int j) {
    if (b > j || e < i) return -inf; // return appropriate value
    if (b >= i && e <= j) return t[n];
    int mid = (b + e) >> 1, l = n << 1, r = l | 1;
    return max(query(l, b, mid, i, j), query(r, mid + 1, e, i, j)); //
        change this
  }
}t;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n = 5;
  for (int i = 1; i <= n; i++) {
    a[i] = i;
  }
  t.build(1, 1, n); // building the segment tree
  t.upd(1, 1, n, 2, 10); // assiging 10 to the index 2 (a[2] := 10)
  cout << t.query(1, 1, n, 1, 5) << '\n'; // range max query on the
      segment [1, 5]

  return 0;
}
```

MatrixExponentiation.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> df9dec, 85 lines

```
using namespace std;

const int mod = 998244353;

struct Mat {
```

```
  int n, m;
  vector<vector<int>> a;
  Mat() { }
  Mat(int _n, int _m) {n = _n; m = _m; a.assign(n, vector<int>(m, 0)); }
  Mat(vector< vector<int> > v) { n = v.size(); m = n ? v[0].size() : 0; a
      = v; }
  inline void make_unit() {
    assert(n == m);
    for (int i = 0; i < n; i++)  {
      for (int j = 0; j < n; j++) a[i][j] = i == j;
    }
  }
  inline Mat operator + (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] + b.a[i][j]) % mod;
      }
    }
    return ans;
  }
  inline Mat operator - (const Mat &b) {
    assert(n == b.n && m == b.m);
    Mat ans = Mat(n, m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < m; j++) {
        ans.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
      }
    }
    return ans;
  }
  inline Mat operator * (const Mat &b) {
    assert(m == b.n);
    Mat ans = Mat(n, b.m);
    for(int i = 0; i < n; i++) {
      for(int j = 0; j < b.m; j++) {
        for(int k = 0; k < m; k++) {
          ans.a[i][j] = (ans.a[i][j] + 1LL * a[i][k] * b.a[k][j] % mod) %
              mod;
        }
      }
    }
    return ans;
  }
  inline Mat pow(long long k) {
    assert(n == m);
    Mat ans(n, n), t = a; ans.make_unit();
    while (k) {
      if (k & 1) ans = ans * t;
      t = t * t;
      k >>= 1;
    }
    return ans;
  }
  inline Mat& operator += (const Mat& b) { return *this = (*this) + b; }
  inline Mat& operator -= (const Mat& b) { return *this = (*this) - b; }
  inline Mat& operator *= (const Mat& b) { return *this = (*this) * b; }
  inline bool operator == (const Mat& b) { return a == b.a; }
  inline bool operator != (const Mat& b) { return a != b.a; }
};

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; long long k; cin >> n >> k;
  Mat a(n, n);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cin >> a.a[i][j];
    }
  }
  Mat ans = a.pow(k);
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cout << ans.a[i][j] << ' ';
    }
    cout << '\n';
  }
  return 0;
}
// https://judge.yosupo.jp/problem/pow_of_matrix
```

profitbasedknapsack.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

93135f, 24 lines

```
const int infinity = 1e9 + 1;
void solve() {
    int n, w, sum = 0; cin >> n >> w;
    int weight[n], value[n];
    for (int i = 0; i < n; ++i) {
        cin >> weight[i] >> value[i];
        sum += value[i];
    }
    vector<int> dp(sum + 1, infinity);
    dp[0] = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = sum; j >= value[i]; --j) {
            int temp = dp[j - value[i]] + weight[i];
            if ((temp <= w) && (temp < dp[j]))
```

```
                dp[j] = temp;
            }
        }
        for (int i = sum; i >= 0; --i) {
            if (dp[i] != infinity) {
                cout << i;
                return;
            }
        }
}
```

segmentedsieve.h

Description: Example structures and functions that doesn't really do anything. Latex commands are supported here, though! Like this: $2^2 = 3$

<bits/stdc++.h> a994dc, 56 lines

```
using namespace std;
using namespace std::chrono;

auto start = high_resolution_clock::now();
#define int long long

bool marked[1000001];
vector<int> primes;

void sieve(int n) {
    memset(marked, false, n+1);
    marked[1] = true;
    for (int i = 4; i <= n; i += 2)
        marked[i] = true;
    for (int i = 3; i*i <= n; i += 2) {
        if (!marked[i]) {
            for (int j = i*i; j <= n; j += i*2)
                marked[j] = true;
        }
    }
    for (int i = 2; i <= n; ++i) {
        if (!marked[i])
            primes.push_back(i);
    }
}

int32_t main() {
    int a, b; cin >> a >> b;
    int n = sqrt(b);
    sieve(n);
    vector<int> rangePrimes;
    int segments = ceil(sqrt(b-a+1));
    int low = a, high = a + segments - 1;
    for (int i = 0; i < segments; ++i) {
        int range = high - low + 1;
        memset(marked, false, range);
        for (auto j : primes) {
            for (int k = max(j*j, (low+j-1)/j * j); k <= high; k += j)
                marked[k-low] = true;
        }

        if (low == 1)
            marked[0] = true;
        for (int j = 0; j < range; ++j) {
            if (!marked[j])
                rangePrimes.push_back(j+low);
        }
        low = high + 1;
        high = min(b, high + segments);
    }

    cout << "Total Primes: " << rangePrimes.size();
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds> (stop - start);
    cout << "\nTime Taken: " << duration.count() << " milliseconds.";
}
```