Cheatsheet
BentengMaria

Contents

1. Combinatorics

1.1. Combination

```
struct Binomial {
  vector<vector<mint>> C;
  vector<mint> fact, inv;
  Binomial(int N, int M)
    : C(N + 1,
        vector<mint>(M + 1, 0)) {
    for (int i = 0; i <= N; i++)
      for (int j = 0;
           j <= min(i, M); j++)
        C[i][j] =
          (!j || j == i
             ? 1
             : C[i - 1][j - 1] +
                 C[i - 1][j]);
  }
  Binomial(int N)
    : fact(N + 1, 1), inv(N + 1) {
    for (int i = 1; i <= N; i++)
      fact[i] = fact[i - 1] * i;
    inv[N] = inverse(fact[N]);
    for (int i = N - 1; i >= 0;
         i--)
      inv[i] =
        inv[i + 1] * (i + 1);
  }
  mint comb(int A, int B) {
    if (!B)
      return 1;
    if (A < B)
      return 0;
    return fact[A] * inv[A - B] *
           inv[B];
  }
  mint perm(int A, int B) {
    if (!B)
      return 1;
    if (A < B)
      return 0;
    return fact[A] * inv[A - B];
  }
  mint
  perm_rep(int A,
           const vector<int> &B) {
    mint res = fact[A];
    for (const int &b : B)
      res *= inv[b];
    return res;
  }
  mint comb_rep(int A, int B) {
    return comb(A + B - 1, A);
  }
  // x1 +x2 +···+xn = s, where l ≤
  // xi ≤ r in O(N)
  mint comb_rep_range(int n, int s,
                      int l,
                      int r) {
    if (s < l * n)
      return 0;
    s -= l * n, r -= l;
    mint ans = 0;
    for (int k = 0; k <= n; k++) {
      mint c =
        comb(s - k - k * r + n,
             n) *
        comb(n, k);
    }
    vector<mint> &operator[](
      int i) {
      assert(0 <= i && i < sz(C));
      return C[i];
    }
    mint operator()(int A, int B) {
      return comb(A, B);
    }
  }
  comb(1e5);
```

1.2. Permutation

```
struct Permutation {
  Fenwick<ll> fenwick;
  ll order(vector<int> A) {
    ll N = sz(A), K = 1;
    vector<ll> fact(N + 1, 1);
    for (int i = 1; i <= N; i++)
      fact[i] = fact[i - 1] * i;
    fenwick = Fenwick<ll>(N + 1);
    for (int &a : A)
      a--;
    reverse(all(A));
    for (int i = 0; i < N; i++) {
      K += fact[i] *
           fenwick.calc(A[i]);
      fenwick.update(A[i], 1);
    }
    return K;
  }
  vector<ll> kth_perm(ll N, ll K) {
    fenwick = Fenwick<ll>(N, 1);
    vector<ll> A(N);
    A[N - 1] = K - 1;
    for (int i = N - 1; i > 0;
         i--) {
      A[i - 1] += A[i] / (N - i);
      A[i] %= (N - i);
    }
    A[0] %= N;
    for (int i = 0; i < N; i++) {
      A[i] =
        bin_search(1, N, A[i] + 1);
      fenwick.update(A[i] - 1, -1);
    }
    return A;
```

```
  }
  int bin_search(int l, int r,
                 ll x) {
    if (l >= r)
      return r;
    int m = (l + r) >> 1;
    if (fenwick.calc(m - 1) >= x)
      return bin_search(l, m, x);
    return bin_search(m + 1, r, x);
  }
} perm;
```

2. Number

2.1. Factorization

```
struct Factor {
  int N;
  vector<int> mind;
  Factor(int n) : N(n) {
    mind.resize(N + 1, 1);
    for (inti = 2; i <= N; i++)
      if (mind[i] == 1)
        for (intj = i; j <= N;
             j += i)
          if (mind[j] == 1)
            mind[j] = i;
  }
  vector<pair<int, int>>
  operator[](intn) {
    vector<pair<int, int>> div;
    while (n > 1) {
      if (div.empty() ||
          div.back().fi != mind[n])
        div.pb(
          {mind[n],
           max_pow(n, mind[n])});
      n /= mind[n];
    }
    returndiv;
  }
  intmax_pow(intn, intp) {
    if (n % p)
      return0;
    return1 + max_pow(n / p, p);
  }
} factor(1e6);

struct Factorization {
  int N, M;
  vector<int> primes;
  Factorization(int n)
    : N(n), M(sqrt(N) + 1) {
    vector<bool> prime(M + 1, 1);
    for (inti = 2; i * i <= M; i++)
      if (prime[i])
        for (intj = i * i; j <= M;
             j += i)
          prime[j] = 0;
    for (inti = 2; i <= M; i++)
      if (prime[i])
        primes.pb(i);
  }
  vector<pair<int, int>>
  operator[](intn) {
    vector<pair<int, int>> div;
    for (int &p : primes) {
      if (n < p)
        break;
      if (n % p)
        continue;
      inta = 0;
      while (!(n % p))
        n /= p, a++;
      div.pb({p, a});
    }
    if (n > 1)
      div.pb({n, 1});
    returndiv;
  }
} factorize(1e9);
```

2.2. Fraction

```
struct frac {
  ll a, b;
  frac(pair<ll, ll> p) : frac(p.fi,
p.se) {}
  frac(llx, lly) : a(x), b(y) {
    assert(b != 0);
    if (a > 0 && b < 0)
      a = -a, b = -b;
    llg = gcd(abs(a), abs(b));
    if (g)
      a /= g, b /= g;
  }
  frac &operator+=(constfrac &y) {
    return *this = frac(a * y.b +
                        y.a * b,
                        b * y.b);
  }
  frac &operator*=(constfrac &y) {
    return *this = frac(a * y.a,
                        b * y.b);
  }
  frac &operator++() {
    a += b;
    return *this = frac(a, b);
  }
  friendfracoperator -
    (constfrac & a) {
    returnfrac(-a.a, a.b);
  }
  friendfracoperator +
    (constfrac & a,
     constfrac & b) {
    returnfrac(a) += b;
```

```
  }
};
```

2.3. Diophantine

```
int gcd(int a, int b, int &x,
        int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int x1, y1;
  int d = gcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}

bool find_any_solution(
  int a, int b, int c, int &x0,
  int &y0, int &g) {
  g = gcd(abs(a), abs(b), x0, y0);
  if (c % g) {
    return false;
  }

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0)
    x0 = -x0;
  if (b < 0)
    y0 = -y0;
  return true;
}

void shift_solution(int &x, int &y,
                    int a, int b,
                    int cnt) {
  x += cnt * b;
  y -= cnt * a;
}

int find_all_solutions(
  int a, int b, int c, int minx,
  int maxx, int miny, int maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c,
                         x, y, g))
    return 0;
  a /= g;
  b /= g;

  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;

  shift_solution(x, y, a, b,
                 (minx - x) / b);
  if (x < minx)
    shift_solution(x, y, a, b,
                   sign_b);
  if (x > maxx)
    return 0;
  int lx1 = x;

  shift_solution(x, y, a, b,
                 (maxx - x) / b);
  if (x > maxx)
    shift_solution(x, y, a, b,
                   -sign_b);
  int rx1 = x;

  shift_solution(x, y, a, b,
                 -(miny - y) / a);
  if (y < miny)
    shift_solution(x, y, a, b,
                   -sign_a);
  if (y > maxy)
    return 0;
  int lx2 = x;

  shift_solution(x, y, a, b,
                 -(maxy - y) / a);
  if (y > maxy)
    shift_solution(x, y, a, b,
                   sign_a);
  int rx2 = x;

  if (lx2 > rx2)
    swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);

  if (lx > rx)
    return 0;
  return (rx - lx) / abs(b) + 1;
}
```

2.4. GCD Pairs Queries

```
struct GCD_CNT {
  int MX;
  vector<ll> cnt, ans;
  GCD_CNT(intmx = 1e6)
    : MX(mx), cnt(MX + 1, 0),
      ans(MX + 1, 0) {}
  void add(int x) { cnt[x]++; }
  void solve() {
    for (inti = 1; i <= MX; i++) {
      for (int j = i; j <= MX;
           j += i)
        ans[i] += cnt[j];
      ans[i] = ans[i] * (ans[i] -
1) / 2;
    }
    for (int i = MX; i > 0; i--)
      for (int j = 2 * i; j <= MX;j
```

```
+= i)
        ans[i] -= ans[j];
  }
  ll &operator[](int i) {
    assert(1 <= i && i <= MX);
    return ans[i];
  }
};
```

2.5. Mod Int

```
using i64 = int64_t;
i64 mod = 1e9 + 7;

struct SafeInt {
  i64 value;

  SafeInt(i64 v)
    : value(v % mod) {}

  SafeInt
  operator*(const SafeInt &other) {
    return (value * other.value) %
           mod;
  }

  SafeInt
  operator+(const SafeInt &other) {
    return (value + other.value) %
           mod;
  }

  SafeInt
  operator-(const SafeInt &other) {
    SafeInt result = *this;
    result.value -= other.value;
    result.value += mod;
    result.value %= mod;
    if (result.value < 0)
      throw domain_error("aneh");
    return result;
  }

  SafeInt operator^(i64 n) {
    SafeInt result(1);
    SafeInt a = *this;

    while (n) {
      if (n & 1)
        result = result * a;

      a = a * a;

      n >>= 1;
    }
    return result;
  }

  SafeInt inv() {
    return *this ^ (mod - 2);
  }
};
```

2.6. Mobius

```
struct Mobius {
  int N;
  vector<int> mind, mob, div;
  Mobius(int n)
    : N(n), mob(N + 1, 0),
      mind(N + 1, 1),
      div(N + 1, 0) {
    for (int i = 2; i <= N; i++)
      if (mind[i] == 1)
        for (int j = i; j <= N;
             j += i)
          if (mind[j] == 1)
            mind[j] = i;
    mob[1] = 1;
    for (int i = 2; i <= N; i++)
      if (mind[i] !=
          mind[i / mind[i]])
        mob[i] =
          -1 * mob[i / mind[i]];
    for (int i = 1; i <= N; i++)
      for (int j = i; j <= N;
           j += i)
        div[j] += mob[i];
  }
  int &operator[](int i) {
    assert(0 <= i && i < N);
    return mob[i];
  }
  ll coprime_pairs(int n) {
    ll res = 0;
    for (int d = 1; d <= N; d++)
      res +=
        mob[d] * (N / d) * (N / d);
    return res / 2;
  }
  ll gcd_sum(int n) {
    ll res = 0;
    for (int d = 1; d <= N; d++)
      res +=
        1LL * mob[d] * div[N / d];
    return res;
  }
  ll lcm_sum(int n) {
    ll res = 0;
    for (int d = 1; d <= N; d++)
      res += mob[d] * div[N / d] *
             (N / d);
    return res;
  }
} mob(1e6);
```

2.7. Number of Solution

```
// Number of solution of ax+by ≥ c
// where x,y ≥ 0
ll lattice_cnt(ll a, ll b, ll c) {
  assert(a >= 0 && b >= 0);
  if (c < 0)
    return 0;
  if (!a || !b)
    return -1;
  assert(a > 0 && b > 0);
  if (a > b)
    swap(a, b);
  ll ans = 0;
  while (c >= 0) {
    ll k = b / a, l = b % a,
       f = c / b;
    ll e = c % b / a,
       g = c % b % a;
    ans += (f + 1) * (e + 1) +
           (f + 1) * f / 2 * k;
    c = f * l - a + g, b = a,
    a = l;
  }
  return ans;
}

mint compute(int n, int m) {
  mint ans = mint(n) * mint(m);
  for (int l = 1, r;
       l <= n && l <= m;
       l = r + 1) {
    r = min(n / (n / l), n);
    mint x = mint(n / l) *
             mint(r - l + 1);
    ans -= mint(n / l) *
           (r - l + 1) * (l + r) *
           mint(1) / 2;
  }
}
template <
  typename T> // Compute min{ax + b
              // (mod m) | 0 ≤ x <
              // n}
T go(T n, const T &m, T a, T b,
     bool is_min = 1, T p = 1,
     T q = 1) {
  if (a == 0)
    return b;
  if (is_min) {
    if (b >= a) {
      T t = (m - b + a - 1) / a,
        c = (t - 1) * p + q;
      if (n <= c)
        return b;
      n -= c;
      b += a * t - m;
    }
    b = a - 1 - b;
  } else {
    if (b < m - a) {
      T t = (m - b - 1) / a,
        c = t * p;
      if (n <= c)
        return a * ((n - 1) / p) +
               b;
      n -= c;
      b += a * t;
    }
    b = m - 1 - b;
  }
  T d = m / a;
  T c =
    go(n, a, m % a, b, !is_min,
       (d - 1) * p + q, d * p + q);
  return is_min ? a - 1 - c
                : m - 1 - c;
}
ll first_k(int a, int b, int k) {
  double y =
    double(b) * log10(a * 1.0);
  y -= (ll)y;
  double temp = pow(10.0, y);
  ll fk =
    temp * 1LL * pow(10, k - 1);
  return fk;
}
```

2.8. Power Tower

```
const int MOD = 1e8 + 37;
vector<int> A;
map<int, int> mp;
template <typename T>
T totient(T &n) {
  if (mp.count(n))
    return mp[n];
  T ans = n;
  for (int i = 2; i * i <= n;
       i++) {
    if (n % i)
      continue;
    ans /= i, ans *= (i - 1);
    while (!(n % i))
      n /= i;
  }
  if (n > 1)
    ans /= n, ans *= (n - 1);
  return ans;
}
int M(ll n, int m) {
  return (n < m ? n : n % m);
}
int power(int a, int b, int m) {
  int c = 1;
  for (; b > 0;
       a = M(1LL * a * a, m),
```

```
      b >>= 1)
    if (b & 1)
      c = M(1LL * c * a, m);
  return c;
}
int pow_tow(int l, int r, int m) {
  if (l == r)
    return M(A[l], m);
  if (m == 1)
    return 1;
  return power(
    A[l],
    pow_tow(l + 1, r, totient(m)),
    m);
}
```

2.9. Sieve Linear

```
const int N = 1e8 + 1;
vector<int> primes, mind(N, 0);
vector<bool> is_prime(N, false);
void linear_sieve() {
  for (int i = 2; i < N; i++) {
    if (!mind[i]) {
      mind[i] = i;
      primes.pb(i);
      is_prime[i] = 1;
    }
    for (int j = 0;
         j < sz(primes) &&
         i * primes[j] < N &&
         primes[j] <= mind[i];
         j++)
      mind[i * primes[j]] =
        primes[j];
  }
}
```

2.10. Sieve Many

```
// Generate prime numbers in O(N
// logN)
vector<bool> prime;
void gen_prime(int n) {
  prime.assign(n + 1, 1);
  prime[0] = prime[1] = 0;
  for (int i = 2; i * i <= n; i++)
    if (prime[i])
      for (int j = i * i; j <= n;
           j += i)
        prime[j] = 0;
}
// Generate minimum divisors in
// O(N)
vector<int> mind;
void gen_mind(int n) {
  mind.assign(n + 1, 1);
  for (int i = 2; i <= n; i++)
    if (mind[i] == 1)
      for (int j = i; j <= n;
           j += i)
        if (mind[j] == 1)
          mind[j] = i;
}
// Generate all totient in O(N
// logN)
vector<int> totient;
void gen_totient(int n) {
  totient.resize(n + 1);
  iota(all(totient), 0);
  for (int i = 2; i <= n; i++)
    if (totient[i] == i)
      for (int j = i; j <= n;
           j += i)
        totient[j] -=
          totient[j] / i;
}
```

```
// Generate different primes
// divisors in O(N)
vector<int> pdc;
vector<int> gen_pdc(int n) {
  gen_mind(n);
  pdc.resize(n + 1);
  for (int i = 2, j; i <= n; i++) {
    j = i / mind[i];
    pdc[i] = pdc[j] +
             (mind[i] != mind[j]);
  }
}
// Generate inverse modulo in O(N)
vector<ll> inv;
void gen_inv(int n) {
  inv.assign(n + 1, 1);
  for (int i = 2; i <= n; i++)
    inv[i] = 1LL *
             (MOD - MOD / i) *
             inv[MOD % i] % MOD;
}
// Generate coprime in O(NM
// log(min(N,M)))
vector<vector<bool>> coprime;
void gen_coprime(int n, int m) {
  coprime.assign(
    n + 1,
    vector<bool>(m + 1, true));
  for (int k = 2; k <= min(n, m);
       k++)
    for (int i = k; i <= n; i += k)
      for (int j = k; j <= m;
           j += k)
        coprime[i][j] = false;
}
template <typename T>
bool perfect_square(T &n) {
  T s = sqrt(n);
  return s * s == n;
```

```
}
// Generate divisors count in
// O(n1/3)
ll div_cnt(ll n) {
  if (n == 1)
    return 1;
  int ans = 1, cnt;
  for (int &p : primes) {
    if (1LL * p * p * p > n)
      break;
    cnt = 1;
    while (!(n % p))
      n /= p, cnt++;
    ans *= cnt;
  }
  if (prime[n])
    ans *= 2;
  else if (perfect_square(n))
    ans *= 3;
  else if (n != 1)
    ans *= 4;
  return ans;
}
```

2.11. Sieve Segmented

```
// Generated primes from L to R,
// which S = sqrt(R) in
// O((R-L+1)loglogR+SloglogS) wh
// R-L+1 ≤ 107 & R ≤ 1012 x is
// prime if isprime[x-L] = true
vector<ll> segmented_sieve(ll L,
                           ll R) {
  ll S = sqrt(R);
  vector<bool> mark(S + 1, 1);
  vector<ll> primes, ans;
  for (ll i = 2; i <= S; i++)
    if (!mark[i]) {
      for (ll j = i * i; j <= S;
           j += i)
        mark[j] = 0;
      primes.pb(i);
    }
  vector<bool> is_prime(R - L + 1,
                        1);
  for (ll p : primes)
    for (ll j =
           max(p * p, (L + p - 1) /
                          p * p);
         j <= R; j += p)
      is_prime[j - L] = 1;
  if (L == 1)
    is_prime[0] = 1;
  for (ll x = L; x <= R; x++)
    if (is_prime[x - L])
      ans.pb(x);
  return ans;
}
```

2.12. Sieve Single Number

```
while (!(n % i))
  n /= i;
vector<pair<ll, int>>
factor(ll n) {
  vector<pair<ll, int>> fac;
  for (ll i = 2; i * i <= n; i++) {
    if (n % i)
      continue;
    fac.pb({i, 0});
    while (!(n % i))
      n /= i, fac.back().se++;
  }
  if (n > 1)
    fac.pb({n, 1});
  return fac;
}
vector<ll> divisor(ll n) {
  vector<ll> D;
  for (int i = 1; i * i <= n;
       i++) {
    if (n % i)
      continue;
    if (i * i != n)
      D.pb(n / i);
    D.pb(i);
  }
  sort(all(D));
  return D;
}
vector<pair<ll, ll>>
divisor_pair(ll n) {
  vector<pair<ll, ll>> D;
  vector<ll> div = divisor(n);
  for (int i = 0, j = sz(div) - 1;
       i <= j; i++, j--)
    D.pb({div[i], div[j]});
  return D;
}
ll div_sum(ll n) {
  ll ans = n;
  for (ll p = 2, e; p * p <= n;
       p++) {
    if (n % p)
      continue;
    e = 0;
    while (!(n % p))
      n /= p, e++;
    ans *= pow(p, e + 1) - 1;
    ans /= p - 1;
  }
  return ans;
}
bool is_prime(ll n) {
  if (n < 2)
    return 0;
  if (n < 4)
```

```
    return 1;
  if (!(n % 2) || !(n % 3))
    return 0;
  for (int i = 5; 1LL * i * i <= n;
       i += 6)
    if (!(n % i) || !(n % (i + 2)))
      return 0;
  return 1;
}
template <typename T> T phi(T &n) {
  T ans = n;
  for (int i = 2; i * i <= n;
       i++) {
    if (n % i)
      continue;
    ans /= i, ans *= (i - 1);
  }
  if (n > 1)
    ans /= n, ans *= (n - 1);
  return ans;
}
```

3. Math

3.1. FFT

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

void fft(vector<cd> &a,
         bool invert) {
  int n = a.size();
  if (n == 1)
    return;

  vector<cd> a0(n / 2), a1(n / 2);
  for (int i = 0; 2 * i < n; i++) {
    a0[i] = a[2 * i];
    a1[i] = a[2 * i + 1];
  }
  fft(a0, invert);
  fft(a1, invert);

  double ang =
    2 * PI / n * (invert ? -1 : 1);
  cd w(1), wn(cos(ang), sin(ang));
  for (int i = 0; 2 * i < n; i++) {
    a[i] = a0[i] + w * a1[i];
    a[i + n / 2] =
      a0[i] - w * a1[i];
    if (invert) {
      a[i] /= 2;
      a[i + n / 2] /= 2;
    }
    w *= wn;
  }
}

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;
}
```

3.2. Matrix

```
struct Matrix {
  int data[11][11];

  Matrix() {
    for (int i = 0; i < 11; i++) {
      for (int j = 0; j < 11;
           j++) {
        data[i][j] = 0;
      }
    }
  }
};

Matrix mult(Matrix a, Matrix b) {
  Matrix c;

  for (int i = 0; i < 11; i++) {
    for (int j = 0; j < 11; j++) {
      for (int k = 0; k < 11;
           k++) {
        c.data[i][j] +=
          (a.data[i][k] *
           b.data[k][j]) %
          mod;
        c.data[i][j] %= mod;
      }
    }
  }

  return c;
}
```

```
Matrix binpow(Matrix a, int exp) {
  Matrix res;
  for (int i = 0; i < 11; i++) {
    res.data[i][i] = 1;
  }

  Matrix c = a;
  while (exp) {
    if (exp & 1) {
      res = mult(res, c);
    }

    c = mult(c, c);

    exp >>= 1;
  }

  return res;
}
```

3.3. Point

```
ftype dot(point2d a, point2d b) {
    return a.x * b.x + a.y * b.y;
}

ftype dot(point3d a, point3d b) {
    return a.x * b.x + a.y * b.y +
a.z * b.z;
}

ftype norm(point2d a) {
    return dot(a, a);
}
double abs(point2d a) {
    return sqrt(norm(a));
}
double proj(point2d a, point2d b) {
    return dot(a, b) / abs(b);
}
double angle(point2d a, point2d b) {
    return acos(dot(a, b) / abs(a) /
abs(b));
}

point3d cross(point3d a, point3d b) {
    return point3d(a.y * b.z - a.z
* b.y,
                   a.z * b.x - a.x
* b.z,
                   a.x * b.y - a.y
* b.x);
}
ftype triple(point3d a, point3d b,
point3d c) {
    return dot(a, cross(b, c));
}
ftype cross(point2d a, point2d b) {
    return a.x * b.y - a.y * b.x;
}
```

3.4. BigNum

```
bool is_smaller(string a,
                string b) {
  return sz(a) != sz(b)
         ? sz(a) < sz(b)
         : a < b;
}
string big_int_plus(string a,
                    string b) {
  string res = "";
  int i = sz(a) - 1, j = sz(b) - 1,
      cry = 0;
  while (i >= 0 || j >= 0 || cry) {
    int s = cry;
    if (i >= 0)
      s += a[i--] - '0';
    if (j >= 0)
      s += b[j--] - '0';
    res += to_string(s % 10);
    cry = s / 10;
  }
  reverse(all(res));
  return res;
}
string big_int_min(string a,
                   string b) {
  if (a == b)
    return "0";
  if (is_smaller(a, b))
    return "-" + big_int_min(b, a);
  string res = "";
  int i = sz(a) - 1, j = sz(b) - 1,
      br = 0;
  for (int sub; i >= 0; i--, j--) {
    sub = (a[i] - '0') - br;
    if (j >= 0)
      sub -= (b[j] - '0');
    if (sub < 0)
      sub += 10, br = 1;
    else
      br = 0;
    res += to_string(sub);
  }
  reverse(all(res));
  size_t fi_dig =
    res.find_first_not_of('0');
  return string::npos != fi_dig
         ? res.substr(fi_dig)
         : "0";
}
```

3.5. INT 128

```
using lint = __int128_t;
// Up to 2127 or 1038
lint read() {
  lint x = 0;
  string s;
  cin >> s;
  for (char &c : s)
    if (isdigit(c))
      x = 10 * x + (c - '0');
  return s[0] == '-' ? -x : x;
}
void print(lint x) {
  if (x < 0) {
    cout << '-';
    x = -x;
  }
  if (x > 9)
    print(x / 10);
  cout << char(x % 10 + '0');
}
```

3.6. Convex Hull

```
coord2_t cross(const Point &O, const
Point &A, const Point &B)
{
  return (A.x - O.x) * (B.y - O.y) -
(A.y - O.y) * (B.x - O.x);
}

vector<Point>
convex_hull(vector<Point> P)
{
  size_t n = P.size(), k = 0;
  if (n <= 3) return P;
  vector<Point> H(2*n);

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

 for (size_t i = 0; i < n; ++i) {
    while (k >= 2 && cross(H[k-2],
H[k-1], P[i]) <= 0) k--;
    H[k++] = P[i];
  }

  for (size_t i = n-1, t = k+1; i >
0; --i) {
    while (k >= t && cross(H[k-2],
H[k-1], P[i-1]) <= 0) k--;
    H[k++] = P[i-1];
  }

  H.resize(k-1);
  return H;
}
```

4. Searching

4.1. Meet in Middle

```
struct MeetInMid {
  vector<int>
  subset(const vector<int> &A,
         intx) {
    int n = sz(A), sum = 0;
    vector<int> sub;
    for (inti = 0; i < (1 << n);
         i++, sum = 0) {
      for (intj = 0; j < n; j++)
        if ((i >> j) & 1)
          sum = (sum + A[j]) % x;
      sub.pb(sum);
    }
    sort(all(sub));
    return sub;
  }
  int solve(constvector<int> &A,
            intx) {
    intn = sz(A), ans = 0;
    autosub_a = subset(
      vector<int>(all_range(A) +
                  n / 2),
      x);
    autosub_b = subset(
      vector<int>(n / 2 + all(A)),
      x);
    for (int &a : sub_a) {
      autopos = upper_bound(
        all(sub_b),
        (2 * x - a - 1) % x);
      ans = max(
        ans, (*(--pos) + a) % x);
    }
    return ans;
  }
};
```

4.2. Ordered Multi Set

```
#include <ext/
pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
usingnamespace__gnu_pbds;
template <typename T>
using ordmulset = tree<
  T, null_type, less_equal<T>,
  rb_tree_tag,
  tree_order_statistics_node_update>;
template <typename T>
struct multi_set {
  ordmulset<T> S;
  multi_set() {}
  multi_set(const ordmulset<T> &s)
    : S(s) {}
  boolexist(Tv) {
    return (returnS.find(v) !=
            S.end());
```

```
  }
  bool empty() { returnS.empty(); }
  void insert(Tv) { S.insert(v); }
  void erase(Tv) {
    if (exist(v))
      S.erase(S.find_by_order(
        S.order_of_key(v)));
  }
  void clear() { S.clear(); }
  int val(inti) {
    return *S.find_by_order(i);
  }
  intfirst_idx(Tv) {
    return (exist(v)
              ? S.order_of_key(v)
              : -1);
  }
  intlast_idx(Tv) {
    if (!exist(v))
      return -1;
    returnS.order_of_key(
      *S.upper_bound(v)) -
      1;
  }
  intcount(Tv) {
    return S.order_of_key(
             *S.upper_bound(v)) -
           S.order_of_key(v);
  }
  intfind_by_order(inti) {
    return *S.find_by_order(i);
  }
  intorder_of_key(Tv) {
    returnS.order_of_key(v);
  }
  intsize() { returnsz(S); }
};
```

4.3. Closest Left Right

```
// forall i, find largest j < i
// s.t. comp(A[j],A[i]) = true
vector<int> cl_left(
  vector<int> &A,
  function<bool(int, int)> &&cmp) {
  int n = sz(A);
  vector<int> closest(n);
  stack<int> st;
  for (int i = 0; i < n; i++) {
    while (!st.empty() &&
           !cmp(A[st.top()], A[i]))
      st.pop();
    closest[i] =
      st.empty() ? -1 : st.top();
    st.push(i);
  }
  return closest;
}
// forall i, find smallest j > i
// s.t. comp(A[j],A[i]) = true
vector<int> cl_right(
  vector<T> &A,
  function<bool(int, int)> &&cmp) {
  int n = sz(A);
  vector<int> closest(n);
  stack<int> st;
  for (int i = n - 1; i >= 0;
       i--) {
    while (!st.empty() &&
           !cmp(A[st.top()], A[i]))
      st.pop();
    closest[i] =
      st.empty() ? -1 : st.top();
    st.push(i);
  }
  return closest;
}
```

5. Data Structure

5.1. DSU

```
int parent[N], sz[N];

void make_set(int u) {
  parent[u] = u;
  sz[u] = 1;
}

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

void union_set(int u, int v) {
  int a = find_set(u);
  int b = find_set(v);
  if (a == b)
    return;
  if (sz[a] < sz[b])
    swap(a, b);
  parent[b] = a;
  sz[a] += sz[b];
}
```

5.2. DSU Weighted

```
struct DSU {
  int N;
  vector<ll> par, weight;
  DSU(int n)
    : N(n), par(N), weight(N, 0) {
    iota(all(par), 0);
  }
  int find(int x) {
    if (x == par[x])
```

```
    return x;
  int p = find(par[x]);
  weight[x] += weight[par[x]];
  return par[x] = p;
 }
 bool unite(int x, int y, int w) {
  int rx = find(x), ry = find(y);
  if (rx == ry)
   return weight[y] -
           weight[x] ==
          w;
  par[rx] = ry;
  weight[rx] =
   weight[y] - weight[x] - w;
  return 1;
 }
 int query(int x, int y) {
  if (find(x) != find(y))
   return -1;
  return weight[y] - weight[x];
 }
 int operator[](int x) {
  find(x);
  return weight[x];
 }
};
```

5.3. DSU Rollback

```
struct DSU {
 int N, M;
 vector<int> par, size;
 stack<int> hist, ver;
 DSU(int n)
  : N(n), M(N), par(N),
    size(N, 1) {
  iota(all(par), 0);
 }
 int find(int u) {
  return u == par[u]
           ? u
           : find(par[u]);
 }
 int unite(int u, int v) {
  u = find(u), v = find(v);
  if (u == v)
   return M;
  if (size[u] < size[v])
   swap(u, v);
  hist.push(v);
  size[u] += size[v];
  par[v] = u;
  return (--M);
 }
 void persist() {
  ver.push(sz(hist));
 }
 int rollback() {
  int target = ver.top();
  for (int u; sz(hist) > target;
       M++) {
   u = hist.top();
   size[par[u]] -= size[u];
   par[u] = u;
   hist.pop();
  }
  ver.pop();
  return M;
 }
};
```

5.4. DSU Dynamic Connectivity

```
struct DQ {
 int N, Q;
 vector<vector<pair<int, int>>>
  tree;
 pair<int, int> cur_node;
 vector<int> ans;
 DSU dsu; // DSU ROLLBACK
 DQ(int n, int q)
  : N(n), Q(q), tree(4 * Q),
    dsu(N), ans(Q, 0) {}
 void update(int u, int v, int l,
             int r) {
  cur_node = {u, v};
  update(1, 0, Q - 1, l, r);
 }
 void update(int x, int l, int r,
             int ql, int qr) {
  if (qr < l || r < ql)
   return;
  if (ql <= l && r <= qr) {
   tree[x].pb(cur_node);
   return;
  }
  int m = (l + r) >> 1;
  update(left(x), l, m, ql, qr);
  update(right(x), m + 1, r, ql,
         qr);
 }
 void solve() {
  process(0, 0, Q - 1);
 }
 void process(int x, int l,
              int r) {
  dsu.persist();
  for (auto &[u, v] : tree[x])
   dsu.unite(u, v);
  if (l == r) {
   ans[l] = dsu.M;
   dsu.rollback();
   return;
  }
  int m = (l + r) >> 1;
  process(left(x), l, m);
  process(right(x), m + 1, r);
```

```
    dsu.rollback();
  }
};
int main() {
  int n, m, q;
  cin >> n >> m >> q;
  map<pair<int, int>, int> pos;
  DQ dc(n, q + 1);
  for (int i = 0, u, v; i < m;
       i++) {
    cin >> u >> v, u--, v--;
    pos[{u, v}] = 0;
  }
  for (int i = 1, op, u, v; i <= q;
       i++) {
    cin >> op >> u >> v, u--, v--;
    if (op == 2) {
      dc.update(u, v, pos[{u, v}],
                i - 1);
      pos.erase({u, v});
    } else
      pos[{u, v}] = i;
  }
  for (auto &[p, i] : pos)
    dc.update(p.fi, p.se, i, q);
  dc.solve();
  for (int &a : dc.ans)
    cout << a << ' ';
}
```

5.5. Segment Tree

```
vector<int> data;
int size;

int build(vector<int> &v, int t,
          int l, int r) {
  if (l == r) {
    data[t] = v[l];
  } else {
    int mid = (l + r) / 2;
    data[t] =
      build(v, 2 * t, l, mid) +
      build(v, 2 * t + 1, mid + 1,
            r);
  }

  return data[t];
}

int query(int t, int l, int r,
          int tl, int tr) {
  if (tl > tr) {
    return 0;
  }
  if (l >= tl && r <= tr) {
    return data[t];
  }
  int mid = (l + r) / 2;
  return query(2 * t, l, mid, tl,
               min(mid, tr)) +
         query(2 * t + 1, mid + 1,
               r, max(mid + 1, tl),
               tr);
}

int query(int tl, int tr) {
  return query(1, 0, size - 1, tl,
               tr);
}

int update(int t, int l, int r,
           int i, int v) {
  if (i < l || i > r)
    return data[t];

  if (l == r && l == i) {
    data[t] += v;
    return data[t];
  }

  int mid = (l + r) / 2;
  data[t] =
    update(2 * t, l, mid, i, v) +
    update(2 * t + 1, mid + 1, r,
           i, v);

  return data[t];
}

void update(int i, int v) {
  update(1, 0, size - 1, i, v);
}
```

5.6. Lazy Segment Tree

```
#define N 20
#define MAX (1 + (1 << 6))
#define inf 0x7fffffff

int arr[N];
int tree[MAX];
int lazy[MAX];

void build_tree(int node, int a,
                int b) {
  if (a > b)
    return;

  if (a == b) {
    tree[node] = arr[a];
    return;
  }

  build_tree(node * 2, a,
             (a + b) / 2);
  build_tree(node * 2 + 1,
```

```
            1 + (a + b) / 2, b);

  tree[node] =
    max(tree[node * 2],
        tree[node * 2 + 1]);
}

// Increment elements within range
// [i, j] with value value
void update_tree(int node, int a,
                 int b, int i,
                 int j,
                 int value) {

  if (lazy[node] != 0) {
    tree[node] += lazy[node];

    if (a != b) {
      lazy[node * 2] += lazy[node];
      lazy[node * 2 + 1] +=
        lazy[node];
    }

    lazy[node] = 0;
  }

  if (a > b || a > j || b < i)
    return;

  if (a >= i && b <= j) {
    tree[node] += value;

    if (a != b) {
      lazy[node * 2] += value;
      lazy[node * 2 + 1] += value;
    }

    return;
  }

  update_tree(node * 2, a,
              (a + b) / 2, i, j,
              value);
  update_tree(1 + node * 2,
              1 + (a + b) / 2, b,
              i, j, value);

  tree[node] =
    max(tree[node * 2],
        tree[node * 2 + 1]);
}

// Query tree to get max element
// value within range [i, j]
int query_tree(int node, int a,
               int b, int i,
               int j) {

  if (a > b || a > j || b < i)
    return -inf;

  if (lazy[node] != 0) {
    tree[node] += lazy[node];

    if (a != b) {
      lazy[node * 2] += lazy[node];
      lazy[node * 2 + 1] +=
        lazy[node];
    }

    lazy[node] = 0;
  }

  if (a >= i && b <= j)
    return tree[node];

  int q1 =
    query_tree(node * 2, a,
               (a + b) / 2, i, j);
  int q2 = query_tree(
    1 + node * 2, 1 + (a + b) / 2,
    b, i, j);

  int res = max(q1, q2);

  return res;
}
```

5.7. Segment Tree MeX

```
struct SegTree {
  int N;
  vector<int> tree;
  void modify(int x, int l, int r,
              int j, int v) {
    if (l == r) {
      tree[x] = v;
      return;
    }
    int m = (l + r) >> 1;
    j <= m
      ? modify(left(x), l, m, j, v)
      : modify(right(x), m + 1, r,
               j, v);
    tree[x] = min(tree[left(x)],
                  tree[right(x)]);
  }
  int process(int x, int l, int r,
              int v) {
    if (l == r)
      return l;
    int m = (l + r) >> 1;
    // if the last occurence is
    // less from left
    if (tree[left(x)] < v)
      return process(left(x), l, m,
                     v);
```

```
    return process(right(x), m + 1,
                   r, v);
  }
};
```

5.8. SQRT Decomposition

```
struct SQRT {
  int N, B;
  vector<int> A;
  vector<ll> block;
  SQRT(vector<int> &arr)
    : A(arr), N(sz(A)),
      B(sqrt(N) + 1), block(B, 0) {
    for (int i = 0; i < N; i++)
      block[i / B] += A[i];
  }
  void update(int i, int x) {
    block[i / B] += x - A[i];
    A[i] = x;
  }
  ll calc(int l, int r) {
    ll s = 0;
    while (l <= r && l % B)
      s += A[l++];
    while (l + B - 1 <= r)
      s += block[l / B], l += B;
    while (l <= r)
      s += A[l++];
    return s;
  }
};
```

5.9. SQRT

```
const int MX = 2e5 + 5;
const int B = 700;
struct MO {
  struct query {
    int l, r, i;
    bool operator<(
      const query &oth) const {
      return make_pair(l / B, r) <
             make_pair(oth.l / B,
                       oth.r);
    }
  };
  int N, Q, ans;
  vector<int> arr, res, occ;
  vector<query> queries;
  MO(int n)
    : N(n), Q(0), ans(0), arr(N),
      occ(MX, 0) {}
  void add_query(int l, int r) {
    queries.pb({l, r, Q++});
  }
  void add(int i) {
    if (!(occ[arr[i]]++))
      ans++;
  }
  void remove(int i) {
    if (!(--occ[arr[i]]))
      ans--;
  }
  void process() {
    res.resize(Q);
    sort(all(queries));
    int L = queries[0].l,
        R = queries[0].l - 1;
    for (const auto &[l, r, i] :
         queries) {
      while (L > l)
        add(--L);
      while (L < l)
        remove(L++);
      while (R < r)
        add(++R);
      while (R > r)
        remove(R--);
      res[i] = ans;
    }
  }
};
```

5.10. Treap

```
struct node {
  int val, prior, size;
  bool rev;
  ll sum;
  node *l, *r;
  node(int v)
    : val(v), sum(v),
      prior(
        rand()), // srand(time(0));
      size(1), l(NULL), r(NULL),
      rev(0) {}
};
struct Treap {
  node *root;
  int size(node *x) {
    return x ? x->size : 0;
  }
  ll suma(node *x) {
    return x ? x->sum : 0;
  }
  void insert(int v) {
    merge(root, root, new node(v));
  }
  void pull(node *x) {
    if (!x)
      return;
    push(x->l), push(x->r);
    x->size =
      size(x->l) + size(x->r) + 1;
    x->sum = suma(x->l) +
             suma(x->r) + x->val;
  }
```

```
void push(node *x) {
  if (!x || !x->rev)
    return;
  x->rev = 0;
  swap(x->l, x->r);
  if (x->l)
    x->l->rev ^= true;
  if (x->r)
    x->r->rev ^= true;
}
void merge(node *&x, node *l,
           node *r) {
  if (!l || !r) {
    x = l ? l : r;
    return;
  }
  push(l), push(r);
  if (l->prior < r->prior)
    merge(l->r, l->r, r), x = l;
  else
    merge(r->l, l, r->l), x = r;
  pull(x);
}
void split(node *x, node *&l,
           node *&r, int v) {
  if (!x) {
    l = r = NULL;
    return;
  }
  push(x);
  if (size(x->l) < v) {
    split(x->r, x->r, r,
          v - size(x->l) - 1);
    l = x;
  } else {
    split(x->l, l, x->l, v);
    r = x;
  }
  pull(x);
}
```

```
void insert(int i, int x) {
  split(root, a, b,
        i); // a = [0, i-1], b =
            // [i, n-1]
  auto v =
    new node(x); // v = [x]
  merge(a, a,
        v); // a = [0, i-1] + [x]
  merge(root, a,
        b); // root = [0, i-1] +
            // [x] + [i, n-1]
}
void del(int i) {
  split(root, a, b,
        i); // a = [0, i-1], b =
            // [i, n-1]
  split(b, root, b,
        1); // root = [i], b =
            // [i+1, n-1]
  merge(root, a,
        b); // root = [0, i-1] +
            // [i+1, n-1]
}
void cut(int l, int r) {
  split(root, a, b,
        l); // a = [0, l-1], b =
            // [l, n-1]
  split(b, c, d,
        r - l +
          1); // c = [l, r], d =
              // [r+1, n-1]
  merge(root, a,
        d); /// root = [0, l-1] +
            /// [r+1, n-1]
  merge(
    root, root,
    c); // root = [0, l-1] +
        // [r+1, n-1] + [l, r]
}
```

```
void reverse(int l, int r) {
  split(root, a, b,
        l); // a = [0, l-1], b =
            // [l, n-1]
  split(b, b, c,
        r - l +
          1); // b = [l, r], c =
              // [r+1, n-1]
  if (b)
    b->rev ^=
      true; // reverse [l, r]
  merge(root, a,
        b); // root = [0, l-1] +
            // [l, r] = [0, r]
  merge(
    root, root,
    c); // root = [0, r] + [r+1,
        // n-1] = [0, n-1]
}
ll sum(int l, int r) {
  return prefix(root, r + 1) -
         prefix(root, l);
}
ll prefix(node *x, int pref) {
  push(x);
  if (!x || !pref)
    return 0;
  if (size(x) == pref)
    return suma(x);
  if (pref <= size(x->l))
    return prefix(x->l, pref);
  return suma(x->l) + (x->val) +
         prefix(x->r,
                pref -
                  size(x->l) -
                  1);
}
int get(int i) {
```

```
    split(root, a, b,
          i); // a = [0, i-1], b =
              // [i, n-1]
    split(b, b, c,
          1); // b = [i, i], c =
              // [i+1, n-1]
    int x =
      b ? b->val : -1; // x = b[0]
    merge(b, b,
          c); // b = [i, i] + [i+1,
              // n-1] = [i, n-1]
    merge(
      root, a,
      b); // root = [0, i-1] + [i,
          // n-1] = [0, n-1]
    return x;
  }
} treap;
ostream &operator<<(ostream &out,
                    node *x) {
  if (!x)
    return out;
  out << x->l << ' ' << x->val << ' '
      << x->r;
  return out;
}
```

5.11. Trie String

```
struct Trie {
  static const int K = 26;
  struct Node {
    int next[26], pref_cnt = 0;
    bool is_word = false;
    Node() {
      fill(next, next + K, -1);
    }
    bool empty() {
      for (int i = 0; i < K; i++)
        if (next[i])
          return 0;
      return 1;
    }
    int &operator[](int i) {
      assert(0 <= i && i < K);
      return next[i];
    }
    int &operator[](char c) {
      assert('a' <= c && c <= 'z');
      return next[c - 'a'];
    }
  };
  int N;
  vector<Node> trie;
  Trie() : trie(1), N(0) {}
  void insert(const string &s) {
    int x = 0;
    for (char c : s) {
      if (trie[x][c] == -1) {
        trie[x][c] = sz(trie);
        trie.pb(Node());
      }
      x = trie[x][c];
      trie[x].pref_cnt++;
    }
    trie[x].is_word = true;
  }
  bool search(const string &s) {
    int x = 0;
    for (char c : s) {
      if (trie[x][c] == -1)
        return 0;
      x = trie[x][c];
    }
    return trie[x].is_word;
  }
  void remove(string &s) {
    remove(0, s, 0);
  }
  bool remove(int x, string &s,
              int d) {
    if (x == -1)
      return 0;
    trie[x].pref_cnt--;
    if (d == sz(s)) {
      if (trie[x].is_word) {
        trie[x].is_word = 0;
        return trie[x].empty();
      }
      return 0;
    }
    if (remove(trie[x][s[d]], s,
               d + 1))
      trie[x][s[d]] = -1;
    return trie[x].is_word &&
           trie[x].empty();
  }
};
```

5.12. Trie XOR

```
template <typename T> struct Trie {
  int N, cur = 1;
  const int LOG = 31;
  vector<array<int, 2>> tree;
  vector<int> cnt;
  Trie(int n)
    : N(n), tree(LOG * N),
      cnt(LOG * N) {
    insert(0);
  }
  void insert(T x) {
    for (int i = LOG, k = 1, bit;
         i >= 0; i--) {
      bit = (x >> i) & 1;
      if (!tree[k][bit])
        tree[k][bit] = ++cur;
```

```
      k = tree[k][bit];
      cnt[k]++;
    }
  }
  void remove(T x) {
    for (int i = LOG, k = 1, bit;
         i >= 0; i--) {
      bit = (x >> i) & 1;
      k = tree[k][bit];
      cnt[k]--;
    }
  }
  T max_xor(T x) {
    T ans = 0;
    for (int i = LOG, k = 1, j,
             bit;
         i >= 0; i--) {
      bit = (x >> i) & 1;
      j = tree[k][bit ^ 1];
      ans <<= 1;
      if (cnt[j])
        ans++, k = j;
      else
        k = tree[k][bit];
    }
    return ans;
  }
};
```

6. Graph

6.1. LCA

```
struct LCA {
  vector<int> height, euler, first,
    segtree;
  vector<bool> visited;
  int n;

  LCA(vector<vector<int>> &adj,
      int root = 0) {
    n = adj.size();
    height.resize(n);
    first.resize(n);
    euler.reserve(n * 2);
    visited.assign(n, false);
    dfs(adj, root);
    int m = euler.size();
    segtree.resize(m * 4);
    build(1, 0, m - 1);
  }

  void
  dfs(vector<vector<int>> &adj,
      int node, int h = 0) {
    visited[node] = true;
    height[node] = h;
    first[node] = euler.size();
    euler.push_back(node);
    for (auto to : adj[node]) {
      if (!visited[to]) {
        dfs(adj, to, h + 1);
        euler.push_back(node);
      }
    }
  }

  void build(int node, int b,
             int e) {
    if (b == e) {
      segtree[node] = euler[b];
    } else {
      int mid = (b + e) / 2;
      build(node << 1, b, mid);
      build(node << 1 | 1, mid + 1,
            e);
      int l = segtree[node << 1],
          r =
            segtree[node << 1 | 1];
      segtree[node] =
        (height[l] < height[r])
          ? l
          : r;
    }
  }

  int query(int node, int b, int e,
            int L, int R) {
    if (b > R || e < L)
      return -1;
    if (b >= L && e <= R)
      return segtree[node];
    int mid = (b + e) >> 1;

    int left = query(node << 1, b,
                     mid, L, R);
    int right =
      query(node << 1 | 1, mid + 1,
            e, L, R);
    if (left == -1)
      return right;
    if (right == -1)
      return left;
    return height[left] <
              height[right]
             ? left
             : right;
  }

  int lca(int u, int v) {
    int left = first[u],
        right = first[v];
    if (left > right)
      swap(left, right);
    return query(1, 0,
                 euler.size() - 1,
                 left, right);
```

```
  }
};
```

6.2. Topological Sort

```
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
  visited[v] = true;
  for (int u : adj[v]) {
    if (!visited[u]) {
      dfs(u);
    }
  }
  ans.push_back(v);
}

void topological_sort() {
  visited.assign(n, false);
  ans.clear();
  for (int i = 0; i < n; ++i) {
    if (!visited[i]) {
      dfs(i);
    }
  }
  reverse(ans.begin(), ans.end());
}
```

6.3. Dijkstra

```
using Item =
  pair<int, int>; // cost, index
priority_queue<Item, vector<Item>,
               greater<Item>>
  pq;

vector<int> cost(N, LLONG_MAX);
vector<bool> visited(N, false);

cost[0] = 0;
pq.push({0, 0});

while (!pq.empty()) {
  int u = pq.top().second;
  pq.pop();

  if (visited[u])
    continue;
  visited[u] = true;

  int cost_u = cost[u];
  for (auto [v, w, _] : adj[u]) {
    int cost_v = cost_u + w;
    if (cost_v < cost[v]) {
      cost[v] = cost_v;
      pq.push({cost_v, v});
    }
  }
}
```

6.4. Strongly Connected Component

```
vector<bool> visited;

void dfs(
  int v,
  vector<vector<int>> const &adj,
  vector<int> &output) {
  visited[v] = true;
  for (auto u : adj[v])
    if (!visited[u])
      dfs(u, adj, output);
  output.push_back(v);
}

void strongly_connected_components(
  vector<vector<int>> const &adj,
  vector<vector<int>> &components,
  vector<vector<int>> &adj_cond) {
  int n = adj.size();
  components.clear(),
    adj_cond.clear();

  vector<int> order;

  visited.assign(n, false);

  for (int i = 0; i < n; i++)
    if (!visited[i])
      dfs(i, adj, order);

  vector<vector<int>> adj_rev(n);
  for (int v = 0; v < n; v++)
    for (int u : adj[v])
      adj_rev[u].push_back(v);

  visited.assign(n, false);
  reverse(order.begin(),
          order.end());

  vector<int> roots(n, 0);

  for (auto v : order)
    if (!visited[v]) {
      std::vector<int> component;
      dfs(v, adj_rev, component);
      components.push_back(
        component);
      int root = *min_element(
        begin(component),
        end(component));
      for (auto u : component)
        roots[u] = root;
    }
```

```
  adj_cond.assign(n, {});
  for (int v = 0; v < n; v++)
    for (auto u : adj[v])
      if (roots[v] != roots[u])
        adj_cond[roots[v]]
          .push_back(roots[u]);
}
```

6.5. Tarjan

```
int n, m, foundat = 1;
vector<vector<int>> graph, scc;
vector<int> disc,
  low; // init disc to -1
bool onstack[MAX]; // init to 0

void tarjan(int u) {
  static stack<int> st;

  disc[u] = low[u] = foundat++;
  st.push(u);
  onstack[u] = true;
  for (auto i : graph[u]) {
    if (disc[i] == -1) {
      tarjan(i);
      low[u] = min(low[u], low[i]);
    } else if (onstack[i])
      low[u] =
        min(low[u], disc[i]);
  }
  if (disc[u] == low[u]) {
    vector<int> scctem;
    while (1) {
      int v = st.top();
      st.pop();
      onstack[v] = false;
      scctem.push_back(v);
      if (u == v)
        break;
    }
    scc.push_back(scctem);
  }
}

int main() {
  // n= vertices of graph
  graph.clear();
  graph.resize(n + 1);
  disc.clear();
  disc.resize(n + 1, -1);
  low.clear();
  low.resize(n + 1);
  for (int i = 0; i < n; i++) {
    if (disc[i + 1] == -1)
      tarjan(i + 1);
  }
}
```

6.6. Bridge

```
int n, m;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> tin, low;
vector<int> sizes;
int timer;
vector<pair<int, int>> bridges;

void IS_BRIDGE(int v, int to) {
  bridges.push_back({v, to});
}

void dfs(int v, int p) {
  visited[v] = true;
  tin[v] = low[v] = timer++;
  sizes[v] = 1;
  for (int to : adj[v]) {
    if (to == p) {
      continue;
    }
    if (visited[to]) {
      low[v] =
        min(low[v], tin[to]);
    } else {
      dfs(to, v);
      sizes[v] += sizes[to];
      low[v] =
        min(low[v], low[to]);
      if (low[to] > tin[v])
        IS_BRIDGE(v, to);
    }
  }
};

void find_bridges() {
  timer = 0;
  visited.assign(n, false);
  tin.assign(n, -1);
  low.assign(n, -1);
  for (int i = 0; i < n; ++i) {
    if (!visited[i])
      dfs(i, -1);
  }
}
```

6.7. Floyd Warshall

```
// Solves the all-pairs shortest
// path problem using Floyd
// Warshall algorithm
void floydWarshall(
  vector<vector<int>> &dist) {
  int V = dist.size();
  for (int k = 0; k < V; k++) {
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
```

```
        // shortest path from
        // i to j
        if (dist[i][k] != 1e8 &&
            dist[k][j] != 1e8)
          dist[i][j] =
            min(dist[i][j],
                dist[i][k] +
                  dist[k][j]);
      }
    }
  }
}
```

6.8. Bellman Ford

```
struct Bellman {
  int N;
  vector<vector<pair<int, int>>>
    adj;
  vector<int> par, cyc;
  vector<ll> dist;

  Bellman(int n)
    : N(n), adj(n), par(N, -1),
      dist(n, INT_MIN) {}
  bool solve(int s) {
    dist[s] = 0;
    int x;
    for (int i = 1; i < N; i++) {
      x = -1;
      for (int u = 0; u < N; u++) {
        if (dist[u] != INT_MIN) {
          for (auto &[v, w] :
               adj[u]) {
            if (dist[u] + w <
                dist[v]) {
              dist[v] =
                dist[u] + w,
              par[v] = u, x = v;
            }
          }
        }
      }
    }

    if (x == -1)
      return 1;
    for (int i = 0; i < N; i++)
      x = par[x];
    for (int v = x;; v = par[v]) {
      cyc.pb(v);
      if (v == x && cyc.size() > 1)
        break;
    }

    reverse(cyc.begin(),
            cyc.end());
    return 0;
  }
};
```

6.9. Minimum Spanning Tree

```
struct Edge {
  int u, v, weight;
  bool
  operator<(Edge const &other) {
    return weight < other.weight;
  }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<int> tree_id(n);
vector<Edge> result;
for (int i = 0; i < n; i++)
  tree_id[i] = i;

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

for (Edge e : edges) {
  if (tree_id[e.u] !=
      tree_id[e.v]) {
    cost += e.weight;
    result.push_back(e);

    int old_id = tree_id[e.u],
        new_id = tree_id[e.v];
    for (int i = 0; i < n; i++) {
      if (tree_id[i] == old_id)
        tree_id[i] = new_id;
    }
  }
}
```

7. String

7.1. Hashing

```
constexpr int prime = 31; // 9973
i64 power[maxN];

power[0] = 1;
for (int i = 1; i <= N; i++) {
  power[i] =
    (power[i - 1] * prime) % mod;
}

vector<i64> buildaHash(string s) {
  vector<i64> aHash(s.length() +
                    1);
  aHash[0] = 0;
  for (int i = 1; i <= s.size();
       i++) {
    aHash[i] =
```

```
      aHash[i - 1] +
      ((s[i - 1] - 'a' + 1) *
       power[i]) %
       mod;
    aHash[i] %= mod;
  }
  return aHash;
}

// compare
i64 laHash =
  aHash[end] - aHash[start - 1];
laHash += mod;
laHash %= mod;
i64 baHash = baHash[rend] -
             baHash[rstart - 1];
baHash += mod;
baHash %= mod;

if (start > rstart) {
  baHash *= power[start - rstart];
  baHash %= mod;
} else if (rstart > start) {
  laHash *= power[rstart - start];
  laHash %= mod;
}
```

7.2. Aho Corasick

```
  const int MAX_N = 6e5;
const int SIGMA = 26;

int n;
string s;
// The number of nodes in trie
int nodes = 1;
int trie[MAX_N][SIGMA];
int fail[MAX_N];
int seen[MAX_N];
int ans[MAX_N];

// leaf[node] stores the indices of
// the words ending in node
vector<int> leaf[MAX_N];
vector<int> g[MAX_N];

/** Add a word to the trie */
void add_word(const string &word,
              const int &idx) {
  int node = 1;
  for (char ch : word) {
    if (trie[node][ch - 'a'] ==
        0) {
      trie[node][ch - 'a'] =
        ++nodes;
    }
    node = trie[node][ch - 'a'];
  }
  leaf[node].push_back(idx);
}

/** BFS to building the failure and
 * suffix links */
void build() {
  queue<int> q;
  int node = 1;
  fail[node] = 1;
  for (int i = 0; i < SIGMA; i++) {
    if (trie[node][i]) {
      fail[trie[node][i]] = node;
      q.push(trie[node][i]);
    } else {
      trie[node][i] = 1;
    }
  }

  while (!q.empty()) {
    int node = q.front();
    q.pop();
    for (int i = 0; i < SIGMA;
         i++) {
      if (trie[node][i]) {
        fail[trie[node][i]] =
          trie[fail[node]][i];
        q.push(trie[node][i]);
      } else {
        trie[node][i] =
          trie[fail[node]][i];
      }
    }
  }
  for (int i = 2; i <= nodes;
       i++) {
    g[fail[i]].push_back(i);
  }
}

void search() {
  int node = 1;
  for (char ch : s) {
    node = trie[node][ch - 'a'];
    seen[node]++;
  }
}

int dfs(int node) {
  int sol = seen[node];
  for (int son : g[node]) {
    sol += dfs(son);
  }
  for (int idx : leaf[node]) {
    ans[idx] = sol;
  }
  return sol;
}
```

7.3. Manacher

```
string menacher(string s) {
  // Preprocess the input so it can
  // handle even length palindromes
  string arr;
  for (int i = 0; i < s.size();
       i++) {
    arr.push_back('#');
    arr.push_back(s[i]);
  }
  arr.push_back('#');

  // dp[i] = palindrome's maximum
  // diameter centered at i
  vector<int> dp(arr.size());
  int left = 0;
  int right = 0;
  int lg_max = 0;
  int idx = 0;
  for (int i = 0;
       i < arr.size();) {
    // Expand the palindrome around
    // i
    while (left > 0 &&
           right <
             arr.size() - 1 &&
           arr[left - 1] ==
             arr[right + 1]) {
      left--;
      right++;
    }

    // Update the diameter
    dp[i] = right - left + 1;

    if (lg_max < dp[i]) {
      lg_max = dp[i];
      idx = i;
    }
    int new_center =
      right + (i % 2 == 0 ? 1 : 0);
    for (int j = i + 1; j <= right;
         j++) {
      dp[j] =
        min(dp[i - (j - i)],
            2 * (right - j) + 1);

      // Update the max diameter
      if (lg_max < dp[i]) {
        lg_max = dp[i];
        idx = i;
      }
      if (j +
            dp[i - (j - i)] / 2 ==
          right) {
        new_center = j;
        break;
      }
    }

    // Move to the new_center and
    // update the left and right
    // borders.
    i = new_center;
    right = i + dp[i] / 2;
    left = i - dp[i] / 2;
  }

  int lg = 0;
  string ans = "";
  for (int j = idx - dp[idx] / 2;
       j <= idx + dp[idx] / 2;
       j++) {
    if (arr[j] != '#') {
      ans.push_back(arr[j]);
    }
  }
  return ans;
}
```

7.4. KMP

```
vector<int>
prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j])
      j = pi[j - 1];
    if (s[i] == s[j])
      j++;
    pi[i] = j;
  }
  return pi;
}
```

8. DP

8.1. Convex Hull

```
// dp[i] = minj<i(dp[j] + b[j] ·
// a[i]) (b[j] ≥ b[j + 1] or a[i] ≤
// a[i +1]) Reduce from O(N2) → O(N
// logN)
struct CHT {
  struct line {
    ll m, c;
    ll operator()(ll x) {
      return m * x + c;
    }
    ld isect(line &l) {
      return (ld)(c - l.c) /
             (l.m - m);
    }
  };
```

```
  int N = 0;
  deque<line> hull;
  void add(ll m, ll c) {
    line L{m, c};
    while (N > 1 &&
           L.isect(hull.front()) <=
             hull.front().isect(
               hull[1]))
      hull.ppf(), N--;
    hull.pf(L), N++;
  }
  ll calc(ll x) {
    while (N > 1 &&
           hull.back()(x) >=
             hull[N - 2](x))
      hull.ppb(), N--;
    return hull.back()(x);
  }
};
```

8.2. Divide and Qonquer

```
// dp[i][j] = mink<j(dp[i - 1][k] +
// C[k][j]) opt[i][j] ≤ opt[i][j +
// 1] opt[i][j] the smallest k that
// gives the optimal answer Reduce
// from O(KN2) → O(KNlog N)
struct DNC_DP {
  int N, K;
  vector<ll> dp, new_dp;
  DNC_DP(int n, int k)
    : N(n), K(k), dp(N + 1, INFL),
      new_dp(N + 1) {}
  ll cost(int i, int j) {
    return pref[j] - pref[i - 1];
  }
  void dnc(int l, int r, int bestl,
           int bestr) {
    if (l > r)
      return;
    int m = (l + r) >> 1;
    pair<ll, int> best = {INFL, 1};
    for (int j = bestl;
         j <= min(m, bestr); j++)
      best =
        min(best, {dp[j - 1] +
                   cost(j, m),
                   j});
    new_dp[m] = best.fi;
    dnc(l, m - 1, bestl, best.se);
    dnc(m + 1, r, best.se, bestr);
  }
  ll solve() {
    dp[0] = 0;
    for (int k = 1; k <= K; k++) {
      fill(all(new_dp), INFL);
      dnc(1, N, 1, N);
      dp = new_dp;
    }
    return dp[N];
  }
};
```

8.3. Knuth

```
// dp[i][j] = mini<k<j(dp[i][k] +
// dp[k + 1][j] + C[i][j]) opt[i][j
// - 1] ≤ opt[i][j] ≤ opt[i + 1][j]
// opt[i][j] the smallest k that
// gives the optimal answer Reduce
// from O(N3) → O(N2)
struct DP_Knuth {
  int N;
  vector<vector<ll>> dp, opt;
  DP_Knuth(int n)
    : N(n),
      dp(N, vector<ll>(N, 0)),
      opt(N, vector<int>(N, 0)) {}
  ll cost(int i, int j) {
    return (pref[j + 1] -
            pref[i]) *
           *2;
  }
  ll solve() {
    for (int i = 0; i < N; i++)
      opt[i][i] = i, dp[i][i] = 0;
    for (int len = 2; len <= N;
         len++) {
      for (int i = 0;
           i + len - 1 < N; i++) {
        int j = i + len - 1;
        dp[i][j] = INF;
        int l = opt[i][j - 1],
            r = opt[i + 1][j];
        for (int k = l;
             k <= min(r, j - 1);
             k++) {
          ll val = dp[i][k] +
                   dp[k + 1][j] +
                   cost(i, j);
          if (val < dp[i][j]) {
            dp[i][j] = val;
            opt[i][j] = k;
          }
        }
      }
    }
    return dp[0][N - 1]
  }
};
```