Northern University of Bussiness and Technology Khulna

NUBTK $(^w^)$

Sourav Mondal, Rahul Kumar Ghosh, Ayon Adikary Arko

NWU CSE FEST 2025

November 11, 2025

1 Contest 1
2 Mathematics 1
3 Data Structures 2
4 Number Theory 3
5 Combinatorial 4
6 Graphs 5
7 Various 6

Contest (1)

template.cpp — 13 lines

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

#ifdef LOCAL
#include "debug.h"
#else
#define dbg(...)
#endif

int32_t main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc — 5 lines

```
# write every function in single line
alias clr="printf '\33c'"
co() { g++ -std=c++23 -O2 -Wall -Wextra
 -Wshadow -Wconversion -o $1 $1.cpp; }
run() { co $1 && ./$1; }
```

hash.sh — 3 lines

```
# Hash file ignoring whitespace and comments. Verifies that
# code was correctly typed. Usage: 'sh hash.sh < A.cpp'
cpp -dD -P -fpreprocessed|tr -d '[:space:]'|md5sum|cut -c-6
```

stress.sh — 20 lines

```
#!/usr/bin/env bash

for ((testNum=0;testNum<$4;testNum++))
do
    "./$3".out > input
    "./$2".out < input > outSlow
    "./$1".out < input > outWrong
    if !(cmp -s "outWrong" "outSlow")
    then
        echo "Error found!"
        echo "Input:"
        cat input
        echo "Wrong Output:"
        cat outWrong
        echo "Slow Output:"
        cat outSlow
        exit
    fi
done
echo Passed $4 tests
```

troubleshoot.txt — 75 lines

```
General:
* Write down most of your thoughts, even if you're not sure
whether they're useful.
* Give your variables (and files) meaningful names.
* Stay organized and don't leave papers all over the place!
* You should know what your code is doing ...

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

Wrong answer:
* Print your solution! Print debug output as well.
* Read the full problem statement again.
* Have you understood the problem correctly?
* Are you sure your algorithm works?
* Try writing a slow (but correct) solution.
* Can your algorithm handle the whole range of input?
* Did you consider corner cases (ex. n=1)?
* Is your output format correct? (including whitespace)
* Are you clearing all data structures between test cases?
* Any uninitialized variables?
* Any undefined behavior (array out of bounds)?
* Any overflows or NaNs (or shifting ll by >=64 bits)?
* Confusing N and M, i and j, etc.?
* Confusing ++i and i++?
* Return vs continue vs break?
* Are you sure the STL functions you use work as you think?
* Add some assertions, maybe resubmit.
* Create some test cases to run your algorithm on.
* Go through the algorithm for a simple case.
* Go through this list again.
* Explain your algorithm to a teammate.
* Ask the teammate to look at your code.
* Go for a small walk, e.g. to the toilet.
* Rewrite your solution from the start or let a teammate do it.

Geometry:
* Work with ints if possible.
* Correctly account for numbers close to (but not) zero.
   ↪Related:
for functions like acos make sure absolute val of input is not
(slightly) greater than one.
* Correctly deal with vertices that are collinear, concyclic,
coplanar (in 3D), etc.
* Subtracting a point from every other (but not itself)?

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

Time limit exceeded:
* Do you have any possible infinite loops?
* What's your complexity? Large TL does not mean that something
simple (like NlogN) isn't intended.
* Are you copying a lot of unnecessary data? (References)
* Avoid vector, map. (use arrays/unordered_map)
* How big is the input and output? (consider FastIO)
* What do your teammates think about your algorithm?
* Calling count() on multiset?

Memory limit exceeded:
* What is the max amount of memory your algorithm should need?
* Are you clearing all data structures between test cases?
* If using pointers try BumpAllocator.
```

Mathematics (2)

2.1 Trigonometry

$$\sin(v+w) = \sin v\cos w + \cos v\sin w$$
$$\cos(v+w) = \cos v\cos w - \sin v\sin w$$
$$\tan(v+w) = \frac{\tan v + \tan w}{1 - \tan v\tan w}$$
$$\sin v + \sin w = 2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$
$$\cos v + \cos w = 2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$a\cos x + b\sin x = r\cos(x-\phi)$$
$$a\sin x + b\cos x = r\sin(x+\phi)$$

where $r = \sqrt{a^2+b^2}, \phi = \operatorname{atan2}(b,a)$.

2.2 Geometry

2.2.1 Triangles

Side lengths: a, b, c

Semiperimeter: $s = \frac{a+b+c}{2}$

Area: $A = \sqrt{s(s-a)(s-b)(s-c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a = \frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

Law of sines: $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Law of tangents: $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

2.3 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.4 Sums/Series

$$1^4+2^4+3^4+\cdots+n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots,\ (-\infty<x<\infty)$$

$$\ln(1+x) = x-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\ldots,\ (-1<x\le 1)$$

$$\sqrt{1+x} = 1+\frac{x}{2}-\frac{x^2}{8}+\frac{2x^3}{32}-\frac{5x^4}{128}+\ldots,\ (-1\le x\le 1)$$

$$\sin x = x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots,\ (-\infty<x<\infty)$$

$$\cos x = 1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots,\ (-\infty<x<\infty)$$

$$e^{ix} = \cos x + i\sin x$$

2.5 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation.

$$\mathbb{E}(aX+bY) = a\mathbb{E}(X)+b\mathbb{E}(Y)$$

If X, Y are independent,

$$V(aX+bY) = a^2V(X)+b^2V(Y).$$

Expectation is linear If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

2.6 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \ldots$ be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n\mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

Data Structures (3)

3.1 STL

MapComparator.h

Description: example of function object (functor) for map or set
Usage: `set<int,cmp> s; map<int,int,cmp> m;`

5bfa6c, 1 lines

```
struct cmp{bool operator()(int l,int r)const{return l>r;}};
```

HashMap.h

Description: Hash map with similar API as unordered_map. Initial capacity must be a power of 2 if provided.
Usage: `ht<int,int> h({},{},{},{},{1<<16});`
Memory: ~1.5x unordered map
Time: ~3x faster than unordered map

<ext/pb_ds/assoc_container.hpp> 5872b2, 9 lines

```
using namespace __gnu_pbds;
struct chash {
  const uint64_t C = ll(4e18*acos(0))+71; // large odd number
  const int RANDOM = rng();
  ll operator()(ll x) const { return __builtin_bswap64((x^
    ↪RANDOM)*C); }
};
template<class K,class V> using ht = gp_hash_table<K,V,chash>;
template<class K,class V> V get(ht<K,V>& u, K x) {
  auto it = u.find(x); return it == end(u) ? 0 : it->s; }
```

OrderStatisticTree.h

Description: A set (not multiset!) with support for finding the n'th element, and finding the index of an element. Change `null_type` to get a map.
Time: $\mathcal{O}(\log N)$

<ext/pb_ds/assoc_container.hpp> fe2351, 12 lines

```
using namespace __gnu_pbds;
tcT> using Tree = tree<T, null_type, less<T>,
  rb_tree_tag, tree_order_statistics_node_update>;
#define ook order_of_key
#define fbo find_by_order

void treeExample() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).f; assert(it == t.lb(9));
  assert(t.ook(10) == 1 && t.ook(11) == 2 && *t.fbo(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

3.2 1D Range Queries

SparseTable.h

Description: Static 1D range (min/max/gcd/lcm/or/and) query. If TL is an issue, use arrays instead of vectors and store values instead of indices.
Memory: $\mathcal{O}(N \log N)$
Time: $\mathcal{O}(1)$

471d4b, 19 lines

```
template<class T> struct SparseTable {
  int N, LOG;
  vector<vector<T>> jmp;
  T cmb(T a, T b) { return min(a, b); }
  SparseTable(const vector<T>& v) {
    N = v.size(); LOG = __lg(N);
    jmp.resize(N, vector<T>(LOG + 1));
    for (int i = 0; i < N; ++i) jmp[i][0] = v[i];
    for (int j = 1; j <= LOG; ++j) {
      for (int i = 0; i + (1<<j) <= N; ++i) {
        jmp[i][j] = cmb(jmp[i][j-1], jmp[i+(1<<(j-1))][j-1]);
      }
    }
  }
  T query(int l, int r) {
    int d = __lg(r-l+1);
    return cmb(jmp[l][d], jmp[r-(1<<d)+1][d]);
  }
};
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value. 0-Indexed.
Time: Both operations are $\mathcal{O}(\log N)$.

122df3, 23 lines

```
template<class T> struct FenwickTree {
  int N; vector<T> bit;
  FenwickTree(int _N) { N = _N; bit.resize(N); }
  void add(int p, T x) {
    for (++p; p <= N; p += p&-p) bit[p-1] += x;
  }
  T sum(int l, int r) { return sum(r+1)-sum(l); }
  T sum(int r) {
    T res = 0;
    for(; r; r -= r&-r) res += bit[r-1];
    return res;
  }
  int lower_bound(T sum) {
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1<<25; pw; pw >>= 1) {
      int npos = pos+pw;
      if (npos <= N && bit[npos-1] < sum)
        pos = npos, sum -= bit[pos-1];
    }
    return pos;
  }
};
```

SegmentTree.h

Description: 1D point update and range query where `cmb` is any associative operation. `seg[1]==query(0,N-1)`.
Time: $\mathcal{O}(\log N)$

749f3e, 29 lines

```
template<class T> struct SegTree {
  const T ID{};
  T cmb(T a, T b) { return a + b; }
  int n; vector<T> seg;
  SegTree(int _n) {
    for (n = 1; n < _n; ) n *= 2;
    seg.assign(2*n, ID);
  }
  void pull(int p) { seg[p] = cmb(seg[2*p], seg[2*p+1]); }
  void update(int p, T val) { // set val at position p
    seg[p += n] = val;
    for (p /= 2; p; p /= 2) pull(p);
  }
  T query(int l, int r) { // zero-indexed, inclusive
    T ra = ID, rb = ID;
    for (l += n, r += n + 1; l < r; l /= 2, r /= 2) {
```

```
      if (l & 1) ra = cmb(ra, seg[l++]);
      if (r & 1) rb = cmb(seg[--r], rb);
    }
    return cmb(ra, rb);
  }
  // int first_at_least(int lo, int val, int ind, int l, int r)
     ↪ { // if seg stores max across range
  //   if (r < lo || val > seg[ind]) return -1;
  //   if (l == r) return l;
  //   int m = (l+r)/2;
  //   int res = first_at_least(lo,val,2*ind,l,m); if (res !=
     ↪-1) return res;
  //   return first_at_least(lo,val,2*ind+1,m+1,r);
  // }
};
```

LazySegmentTree.h
Description: 1D range increment and sum query.
Usage: `LazySeg<int64_t, 1<<20> T; T.update(l, r, val);`
Time: $\mathcal{O}(\log N)$
e0742f, 42 lines

```
template<class T, int SZ>struct LazySeg {
  // SZ must be power of 2
  static_assert(__builtin_popcount(SZ) == 1);
  const T ID{};
  T cmb(T a, T b) { return a + b; }
  T seg[2*SZ], lazy[2*SZ];
  LazySeg() {
    for (int i = 0; i < 2*SZ; ++i) seg[i] = lazy[i] = ID;
  }
  // modify values for current node
  void push(int ind, int L, int R) {
    seg[ind] += (R-L+1) * lazy[ind]; // dependent on operation
    if (L != R) for (int i = 0; i < 2; ++i) lazy[2*ind+i] +=
       ↪lazy[ind];
    lazy[ind] = 0;
  }
  void pull(int ind) {
    seg[ind] = cmb(seg[2*ind], seg[2*ind+1]);
  }
  void build() {
    for (int i = SZ-1; i >= 1; --i) pull(i);
  }
  void update(int lo, int hi, T inc, int ind = 1, int L = 0,
     ↪int R = SZ - 1) {
    push(ind, L, R);
    if (hi < L || R < lo) return;
    if (lo <= L && R <= hi) {
      lazy[ind] = inc;
      push(ind, L, R);
      return;
    }
    int M = (L + R) / 2;
    update(lo, hi, inc, 2*ind, L, M);
    update(lo, hi, inc, 2*ind+1, M+1, R);
    pull(ind);
  }
  T query(int lo, int hi, int ind = 1, int L = 0, int R = SZ -
     ↪1) {
    push(ind, L, R);
    if (lo > R || L > hi) return ID;
    if (lo <= L && R <= hi) return seg[ind];
    int M = (L + R) / 2;
    return cmb(query(lo, hi, 2*ind, L, M), query(lo, hi, 2*ind
       ↪+1, M+1, R));
  }
};
```

3.3 2D Range Queries

PrefixSum2D.h
Description: calculates rectangle sums in constant time
65b070, 17 lines

```
template<typename T> struct PrefixSum2D {
  vector<vector<T>> sum;
  PrefixSum2D(const vector<vector<T>> &v) {
    int n = v.size(), m = v[0].size();
    sum.assign(n+1, vector<T>(m+1, T{}));
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < m; ++j) {
        sum[i+1][j+1] = v[i][j]
        - sum[i][j] + sum[i+1][j] + sum[i][j+1];
      }
    }
  }
  T query(int x1, int y1, int x2, int y2) {
    return sum[x2][y2] + sum[x1-1][y1-1]
    - sum[x1-1][y2] - sum[x2][y1-1];
  }
};
```

Number Theory (4)

4.1 Modular Arithmetic

ModIntShort.h
Description: Modular arithmetic. Assumes MOD is prime.
Usage: `Z a = MOD+5; inv(a); // 400000003`
28ffda, 23 lines

```
struct Z {
  int v;
  static const int MOD = 1E9 + 7;
  explicit operator int() const { return v; }
  Z() : v(0) {}
  Z(int64_t _v) : v(int(_v % MOD)) { v += (v < 0) * MOD; }
  Z &operator+=(Z o) {
    if ((v += o.v) >= MOD) v -= MOD;
    return *this; }
  Z &operator-=(Z o) {
    if ((v -= o.v) < 0) v += MOD;
    return *this; }
  Z &operator*=(Z o) {
    v = int((int64_t)v * o.v % MOD);
    return *this; }
  friend Z pow(Z a, int64_t p) {
    return p == 0 ? 1 : pow(a*a, p/2)*(p&1 ? a:1); }
  friend Z inv(Z a) {
    return pow(a, MOD - 2); }
  friend Z operator+(Z a, Z b) { return a += b; }
  friend Z operator-(Z a, Z b) { return a -= b; }
  friend Z operator*(Z a, Z b) { return a *= b; }
};
```

4.2 Primality

4.2.1 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 $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

4.2.2 Divisors

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

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 $n < 1e19$.

Dirichlet Convolution: Given a function $f(x)$, let

$(f * g)(x) = \sum_{d|x} g(d) f(x/d)$.

If the partial sums $s_{f*g}(n), s_g(n)$ can be computed in $O(1)$ and $s_f(1 \dots n^{2/3})$ can be computed in $O\left(n^{2/3}\right)$ then all $s_f\left(\frac{n}{d}\right)$ can as well. Use

$$s_{f*g}(n) = \sum_{d=1}^{n} g(d) s_f(n/d).$$

If $f(x) = \mu(x)$ then $g(x) = 1, (f * g)(x) = (x == 1)$, and $s_f(n) = 1 - \sum_{i=2}^{n} s_f(n/i)$.

If $f(x) = \phi(x)$ then $g(x) = 1, (f * g)(x) = x$, and $s_f(n) = \frac{n(n+1)}{2} - \sum_{i=2}^{n} s_f(n/i)$.

Sieve.h
Description: Tests primality up to SZ. Runs faster if only odd indices are stored.
Time: $\mathcal{O}(SZ \log \log SZ)$ or $\mathcal{O}(SZ)$
41c6ed, 20 lines

```
template<int SZ> struct Sieve {
  bitset<SZ> is_prime; vi primes;
  Sieve() {
    is_prime.set(); is_prime[0] = is_prime[1] = 0;
    for (int i = 4; i < SZ; i += 2) is_prime[i] = 0;
    for (int i = 3; i*i < SZ; i += 2) if (is_prime[i])
      for (int j = i*i; j < SZ; j += i*2) is_prime[j] = 0;
    F0R(i,SZ) if (is_prime[i]) primes.pb(i);
  }
  // int sp[SZ]{}; // smallest prime that divides
  // Sieve() { // above is faster
  //  FOR(i,2,SZ) {
  //    if (sp[i] == 0) sp[i] = i, primes.pb(i);
  //    for (int p: primes) {
  //      if (p > sp[i] || i*p >= SZ) break;
  //      sp[i*p] = p;
  //    }
  //  }
  // }
};
```

4.3 Euclidean Algorithm

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 Lifting the Exponent

For $n > 0$, p prime, and ints x, y s.t. $p \nmid x, y$ and $p|x-y$:

- $p \neq 2$ or $p = 2, 4|x-y \implies v_p(x^n - y^n) = v_p(x-y) + v_p(n)$.
- $p = 2, 2|n \implies v_2(x^n - y^n) = v_2((x^2)^{n/2} - (y^2)^{n/2})$.

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		

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 $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

5.3 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:

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

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.

5.4 Young Tableaux

Let a **Young diagram** have shape $\lambda = (\lambda_1 \geq \cdots \geq \lambda_k)$, where λ_i equals the number of cells in the i-th (left-justified) row from the top. A **Young tableau** of shape λ is a filling of the $n = \sum \lambda_i$ cells with a permutation of $1 \ldots n$ such that each row and column is increasing.

Hook-Length Formula: For the cell in position (i,j), let $h_\lambda(i,j) = |\{(I,J)|i \leq I, j \leq J, (I = i \text{ or } J = j)\}|$. The number of Young tableaux of shape λ is equal to $f^\lambda = \frac{n!}{\prod h_\lambda(i,j)}$.

Schensted's Algorithm: converts a permutation σ of length n into a pair of Young Tableaux $(S(\sigma), T(\sigma))$ of the same shape. When inserting $x = \sigma_i$,

1. Add x to the first row of S by inserting x in place of the largest y with $x < y$. If y doesn't exist, push x to the end of the row, set the value of T at that position to be i, and stop.

2. Add y to the second row using the same rule, keep repeating as necessary.

All pairs $(S(\sigma), T(\sigma))$ of the same shape correspond to a unique σ, so $n! = \sum (f^\lambda)^2$. Also, $S(\sigma^R) = S(\sigma)^T$.

Let $d_k(\sigma), a_k(\sigma)$ be the lengths of the longest subseqs which are a union of k decreasing/ascending subseqs, respectively. Then $a_k(\sigma) = \sum_{i=1}^{k} \lambda_i, d_k(\sigma) = \sum_{i=1}^{k} \lambda_i^*$, where λ_i^* is size of the i-th column.

Graphs (6)

Erdos-Gallai: $d_1 \geq \cdots \geq d_n$ can be degree sequence of simple graph on n vertices iff their sum is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k), \forall 1 \leq k \leq n$.

6.1 Basics

DSU.h
Description: Disjoint Set Union with path compression and union by size. Add edges and test connectivity. Use for Kruskal's or Boruvka's minimum spanning tree.
Time: $\mathcal{O}(\alpha(N))$
a876ce, 12 lines

```
struct DSU {
  vector<int> e;
  DSU(int N) { e = vector<int>(N, -1); }
  int get(int x) { return e[x] < 0 ? x : e[x] = get(e[x]); }
  bool sameSet(int a, int b) { return get(a) == get(b); }
  int size(int x) { return -e[get(x)]; }
  bool unite(int x, int y) { // union by size
    x = get(x), y = get(y); if (x == y) return false;
    if (e[x] > e[y]) swap(x, y);
    e[x] += e[y]; e[y] = x; return true;
  }
};
```

Dijkstra.h
Description: shortest path
Time: $\mathcal{O}(N \log N + M)$
5afc43, 27 lines

```
template<class T, bool directed> struct Dijkstra {
  int SZ;
  vector<T> dist;
  vector<vector<pair<int, T>>> adj;
  Dijkstra(int _SZ) {
    SZ = _SZ;
    adj.resize(SZ);
  }
  void ae(int u, int v, T w) {
    adj[u].push_back({v, w});
    if (!directed) adj[v].push_back({u, w});
  }
  void gen(int st) {
    dist.assign(SZ, numeric_limits<T>::max());
    priority_queue<pair<T, int>> PQ;
    auto relax = [&](int v, T d) {
      if (dist[v] <= d) return;
      dist[v] = d;
      PQ.push({-dist[v], v});
    }; relax(st, 0);
    while (!PQ.empty()) {
      auto [d, u] = PQ.top(); PQ.pop(); d = -d;
      if (dist[u] < d) continue;
      for (auto& [v, w]: adj[u]) relax(v, w+d);
    }
  }
};
```

TopoSort.h
Description: sorts vertices such that if there exists an edge x->y, then x goes before y
55695d, 15 lines

```
struct TopoSort {
  int N; vi in, res;
  V<vi> adj;
  void init(int _N) { N = _N; in.rsz(N); adj.rsz(N); }
  void ae(int x, int y) { adj[x].pb(y), ++in[y]; }
  bool sort() {
    queue<int> todo;
    F0R(i,N) if (!in[i]) todo.push(i);
    while (sz(todo)) {
      int x = todo.ft; todo.pop(); res.pb(x);
      each(i,adj[x]) if (!(--in[i])) todo.push(i);
    }
    return sz(res) == N;
  }
};
```

6.2 Trees

TreeDiameter.h
Description: Calculates longest path in tree. The vertex furthest from 0 must be an endpoint of the diameter.
3c22a4, 38 lines

```
struct TreeDiameter {
  int N, diaLen;
  vector<int> dist, dia;
  vector<vector<int>> adj;
  TreeDiameter(int _N) {
    N = _N;
    dia = {0, 0};
    adj.resize(N);
    dist.resize(N);
  }
  void ae(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void genDist(int R) { dist[R] = 0; dfs(R); }
  void gen() {
    genDist(0);
    for (int i = 0; i < N; ++i) {
      if (dist[i] > dist[dia[0]]) dia[0] = i;
    }
    genDist(dia[0]);
    for (int i = 0; i < N; ++i) {
      if (dist[i] > dist[dia[1]]) dia[1] = i;
    }
    diaLen = dist[dia[1]];
    // center vertex of dia[0] -> dia[1]
    // int cen = dia[1];
    // for (int i = 0; i < diaLen/2; ++i) cen = par[cen];
    // center = {cen};
    // if (diaLen&1) center.push_back(par[cen]);
  }
  void dfs(int u, int p = -1) {
    for (auto& v: adj[u]) {
      if (v == p) continue;
      dist[v] = dist[u] + 1; dfs(v, u);
    }
  }
};
```

LCAjump.h
Description: Calculates least common ancestor in tree with verts $0 \ldots N-1$ and root R using binary jumping.
Memory: $\mathcal{O}(N \log N)$
Time: $\mathcal{O}(N \log N)$ build, $\mathcal{O}(\log N)$ query
5c0112, 47 lines

```
struct LCA {
  int N, LOG;
  vector<int> depth;
  vector<vector<int>> par, adj;
  LCA(int _N) {
    N = _N; LOG = 1;
    while ((1<<LOG) < N) ++LOG;
    adj.resize(N);
    depth.resize(N);
    par.resize(N, vector<int>(LOG));
  }
  void ae(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void gen(int R = 0) { par[R][0] = R; dfs(R); }
  void dfs(int u = 0) {
    for (int j = 1; j < LOG; ++j) {
      par[u][j] = par[par[u][j-1]][j-1];
    }
    for (auto& v: adj[u]) {
      if (v == par[u][0]) continue;
      depth[v] = depth[u]+1;
      par[v][0] = u;
      dfs(v);
    }
  }
  int jump(int x, int d) {
    for (int j = 0; j < LOG; ++j) {
      if ((d>>j) & 1) x = par[x][j];
    }
    return x;
  }
  int lca(int u, int v) {
    if (depth[u] > depth[v]) swap(u, v);
    v = jump(v, depth[v]-depth[u]);
    if (u == v) {return u; }
    for (int j = LOG-1; j >= 0; --j) {
      int U = par[u][j], V = par[v][j];
      if (U != V) u = U, v = V;
    }
    return par[u][0];
  }
  int dist(int u, int v) {
    return depth[u] + depth[v] - 2*depth[lca(u, v)];
  }
};
```

6.2.1 SqrtDecompton

HLD generally suffices. If not, here are some common strategies:

- Rebuild the tree after every $\sqrt{N}$ queries.
- Consider vertices with $>$ or $< \sqrt{N}$ degree separately.
- For subtree updates, note that there are $O(\sqrt{N})$ distinct sizes among child subtrees of any node.

Block Tree: Use a DFS to split edges into contiguous groups of size $\sqrt{N}$ to $2\sqrt{N}$.

Mo's Algorithm for Tree Paths: Maintain an array of vertices where each one appears twice, once when a DFS enters the vertex (`st`) and one when the DFS exists (`en`). For a tree path $u \leftrightarrow v$ such that `st[u]<st[v]`,

- If u is an ancestor of v, query `[st[u],st[v]]`.
- Otherwise, query `[en[u],st[v]]` and consider $LCA(u, v)$ separately.

Solutions with worse complexities can be faster if you optimize the operations that are performed most frequently. Use arrays instead of vectors whenever possible. Iterating over an array in order is faster than iterating through the same array in some other order (ex. one given by a random permutation) or DFSing on a tree of the same size. Also, the difference between $\sqrt{N}$ and the optimal block (or buffer) size can be quite large. Try up to 5x smaller or larger (at least).

6.3 DFS Algorithms

DFSBasic.h
Description: Basic DFS implementation for undirected graphs
Time: $\mathcal{O}(N + M)$
14b944, 21 lines

```
struct DFS {
  int N;
  vector<bool> visited;
  vector<vector<int>> adj;
  DFS(int _N) {
    N = _N;
    adj.resize(N);
    visited.resize(N);
  }
  void ae(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  void dfs(int u) {
    visited[u] = true;
    for (auto& v: adj[u]) {
      if (visited[v]) continue;
      dfs(v);
    }
  }
};
```

Various (7)

7.1 Binary search

fstTrue.h
Description: Finds the first value x in [lo, hi] such that predicate f(x) is true.
Usage: `int ans = fstTrue()`
Time: $\mathcal{O}(\log(hi - lo))$
2654fa, 9 lines

```
template<typename T, typename U> T fstTrue(T lo, T hi, U f) {
  ++hi;
  assert(lo <= hi);
  while (lo < hi) {
    T mid = lo + (hi - lo) / 2;
    f(mid) ? hi = mid : lo = mid + 1;
  }
  return lo;
};
```

lstTrue.h
Description: Finds the last value x in [lo, hi] such that predicate f(x) is true.
Usage: `int ans = fstTrue()`
Time: $\mathcal{O}(\log(hi - lo))$
8aff85, 9 lines

```
template<typename T, typename U> T lstTrue(T lo, T hi, U f) {
  --lo;
  assert(lo <= hi);
  while (lo < hi) {
    T mid = lo + (hi - lo + 1) / 2;
    f(mid) ? lo = mid : hi = mid - 1;
  }
  return lo;
};
```

realTrue.h
Description: Binary search on continuous (floating-point) domain.
Usage: `int ans = realTrue()`
Time: $\mathcal{O}(\log(precision))$ — write 100 iterations and have a relax
691e1d, 7 lines

```
template<typename T, typename U>T realTrue(T lo, T hi, U f) {
  for (int i = 0; i < 100; ++i) {
    T mid = 0.5 * (hi + lo);
    f(mid) ? lo = mid : hi = mid;
  }
  return lo;
};
```

7.2 Dynamic programming

When doing DP on intervals:
$a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i, j)$, where the (minimal) optimal k increases with both i and j,

- one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$.
- This is known as Knuth DP. Sufficient criteria for this are if $f(b, c) \leq f(a, d)$ and $f(a, c) + f(b, d) \leq f(a, d) + f(b, c)$ for all $a \leq b \leq c \leq d$.
- Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.

7.3 Debugging tricks

- `signal(SIGSEGV, [](int) { _Exit(0); });` converts segfaults into Wrong Answers. Similarly one can catch SIGABRT (assertion failures) and SIGFPE (zero divisions). `_GLIBCXX_DEBUG` violations generate SIGABRT (or SIGSEGV on gcc 5.4.0 apparently).
- `feenableexcept(29);` kills the program on NaNs (1), 0-divs (4), infinities (8) and denormals (16).

7.4 Optimization tricks

7.4.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `F0R(b,k) F0R(i,1<<K) if (i&1<<b) D[i] += D[i^(1<<b)];` computes all sums of subsets.

7.4.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize for loops and optimizes floating points better (assumes associativity and turns off denormals).
- `#pragma GCC target ("avx,avx2")` can double performance of vectorized code, but causes crashes on old machines. Also consider older `#pragma GCC target ("sse4")`.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

7.5 Other languages

Python3.py
Description: Python review.
40 lines

```
from math import *
import sys, random
def nextInt():
  return int(input())
def nextStrs():
  return input().split()
def nextInts():
  return list(map(int,nextStrs()))

n = nextInt()
v = [n]
def process(x):
  global v
  x = abs(x)
  V = []
  for t in v:
    g = gcd(t,x)
    if g != 1:
      V.append(g)
    if g != t:
      V.append(t//g)
  v = V
for i in range(50):
  x = random.randint(0,n-1)
  if gcd(x,n) != 1:
    process(x)
  else:
    sx = x*x%n # assert(gcd(sx,n) == 1)
    print(f"sqrt {sx}")
    sys.stdout.flush()
    X = nextInt()
    process(x+X)
    process(x-X)
print(f'! {len(v)}',end='')
for i in v:
  print(f' {i}',end='')
print()
sys.stdout.flush() # sys.exit(0) -> exit
# sys.setrecursionlimit(int(1e9)) -> stack size
# print(f'{ans:=.6f}') -> print ans to 6 decimal places
```