University of Wollongong

!DirectCurrent beta version

Phu Hong Can Khang Pham, Thanh Trung Nguyen, Nguyen Vu Nguyen

2024-10-18

Contest (1)

.bashrc

3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

10 lines

```
set background=dark
set number
syntax on
set mouse=a
set backspace=indent,eol,start
set autoindent
set hlsearch

ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

3 lines

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

troubleshoot.txt

52 lines

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

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases 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.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

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

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

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

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \Rightarrow \begin{aligned} x &= \frac{ed - bf}{ad - bc} \\ y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k - c_1 x^{k-1} - \cdots - c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v+w) = \sin v\cos w + \cos v\sin w$$
$$\cos(v+w) = \cos v\cos w - \sin v\sin w$$

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

$$(V+W)\tan(v-w)/2 = (V-W)\tan(v+w)/2$$

where V, W are lengths of sides opposite angles v, w.

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

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

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a+b+c}{2}$

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

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

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

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

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

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

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

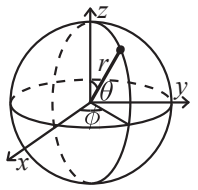
2.4.2 Quadrilaterals

With side lengths a, b, c, d, diagonals e, f, diagonals angle θ, area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef\cdot\sin\theta = F\tan\theta = \sqrt{4e^2f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.3 Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \text{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \text{atan2}(y, x)$$

2.5 Derivatives/Integrals

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

Integration by parts:

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

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$
$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

2.8 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) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ 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. 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 $f_X(x)$.

Expectation is linear:

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

For independent X and Y,

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

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n, p)$, $n = 1, 2, \ldots$, $0 \le p \le 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\mathrm{Bin}(n, p)$ is approximately $\mathrm{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\mathrm{Fs}(p)$, $0 \le p \le 1$.

$$p(k) = p(1-p)^{k-1},\ k = 1, 2, \ldots$$

$$\mu = \frac{1}{p},\ \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0, 1, 2, \ldots$$

$$\mu = \lambda,\ \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\mathrm{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \ge 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda},\ \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 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.

π is a stationary distribution if $\pi = \pi\mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i. π_j/π_i is the expected number of visits in state j between two visits in state i.

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets $\mathbf{A}$ and $\mathbf{G}$, such that all states in $\mathbf{A}$ are absorbing ($p_{ii} = 1$), and all states in $\mathbf{G}$ leads to an absorbing state in $\mathbf{A}$. The probability for absorption in state $i \in \mathbf{A}$, when the initial state is j, is $a_{ij} = p_{ij} + \sum_{k\in\mathbf{G}} a_{ik}p_{kj}$. The expected time until absorption, when the initial state is i, is $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Combinatorial (3)

3.1 Permutations

3.1.1 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;
}
```

3.1.2 Cycles

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

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

3.1.3 Derangements

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

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

3.1.4 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

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

3.2 Partitions and subsets

3.2.1 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

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

3.2.3 Binomials

multinomial.h

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

a0a312, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

3.3 General purpose numbers

3.3.1 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:

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

3.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

3.3.3 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. $\pi(j) > j$.

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

3.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

3.3.5 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,

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

3.3.6 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: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

3.3.7 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.

Data structures (4)

OrderStatisticTree.cpp

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

d14429, 13 lines

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

// find_by_order(x): return iterator to k-th largest element(count from 0)
// order_of_key(x): number of element strictly less than x

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

DSU.cpp

Description: Disjoint-set data structure.
Time: $\mathcal{O}(\alpha(N))$

2c5238, 53 lines

```
// Remember to check limit
struct DSU {
  // CAREFUL
  static constexpr int N_MAX = 2e5 + 1;

  int n, t;
  int par[N_MAX];
  int temp[N_MAX];

  DSU() {}
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i += 1) par[i] = -1;
  }

  int findRoot(int vt) {
    t = 0;
    while (par[vt] >= 0) {
      temp[t++] = vt;
      vt = par[vt];
    }
    for (int i = 0; i < t; i += 1) par[temp[i]] = vt;
    return vt;
  }

  void join(int u, int v) {
    u = findRoot(u);
    v = findRoot(v);
    if (u == v) return;
    if (par[u] > par[v]) {u^=v;v^=u;u^=v;}
    par[u] += par[v];
    par[v] = u;
  }

  bool sameSet(int u, int v) {
    return findRoot(u) == findRoot(v);
  }
};
/*
DSU dsu;

int main() {
  int n, q; cin >> n >> q;
  dsu.init(n);

  int t, u, v;
  for (int i = 0; i < q; i += 1) {
    cin >> t >> u >> v;
    if (!t) dsu.join(u, v);
    else cout << dsu.sameSet(u, v) << "\n";
  }
}
*/
```

sparseTable.cpp

Description: Sparse table for RMQ.
Time: $\mathcal{O}(N \log N + Q)$

d36974, 43 lines

```
struct SparseTable {
  // CAREFUL
  static constexpr int N_MAX = 5e5 + 1;
  static constexpr int L_MAX = 20; // 2^19 > N_MAX

  // Length of the array
  int len;

  // st[i][j] = minimum value in the segment length 2^i starting at j
  int st[L_MAX][N_MAX];

  SparseTable() {}
  void init(int _len, int (&a)[]) {
    len = _len;
    for (int i = 0; i < len; i += 1) st[0][i] = a[i];
    for (int i = 1; i < L_MAX; i += 1)
    // CAREFUL(MIN-MAX)
    for (int j = 0; j < len; j += 1) st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i-1))]);
  }

  // min value in [l, r]
  int retrieve(int l, int r) {
    int i = __lg(r-l+1);
    // CAREFUL(MIN-MAX)
    return min(st[i][l], st[i][r - (1 << i) + 1]);
  }
};
```

```
/*
SparseTable rmq;

int main() {
  int n, q; cin >> n >> q;
  int arr[n]; for (int i = 0; i < n; i += 1) cin >> arr[i];

  rmq.init(n, arr);

  int l, r;
  for (int i = 0; i < q; i += 1) {
    cin >> l >> r;
    cout << rmq.retrieve(l, r-1) << "\n";
  }
}
*/
```

Graph (5)

5.1 Traversal

TarjanBAP.cpp
Description: Bridges and Articulation point with graph traversal.
Time: $\mathcal{O}(V+E)$

c9f525, 67 lines

```
// Remember to check limits
struct Graph {
  // CAREFUL
  static constexpr int N_MAX = 1e5;

  int earliest[N_MAX];
  int visitTime[N_MAX];
  vector<int> nv[N_MAX];
  bitset<N_MAX> isArticulationPoint;
  int n, bridgeCnt, dfsRoot, rootChildren, curTime;

  Graph() {}
  void init(int _n) {
    n = _n;
  }

  void addEdge(int u, int v) {
    nv[u].push_back(v);
    nv[v].push_back(u);
  }

  void traverse(int u, int p) {
    earliest[u] = curTime;
    visitTime[u] = curTime++;

    for (int v: nv[u]) if (v != p) {
      if (visitTime[v]) earliest[u] = min(earliest[u], visitTime[v]);
      else {
        if (u == dfsRoot) rootChildren += 1;
        traverse(v, u);
        if (earliest[v] >= visitTime[u]) {
          isArticulationPoint[u] = 1;
          // (u, v) is bridge
          if (earliest[v] > visitTime[u]) bridgeCnt += 1;
        }
        earliest[u] = min(earliest[u], earliest[v]);
      }
    }
  }

  void startFrom(int root) {
    curTime = 1;
    dfsRoot = root;
    rootChildren = 0;
    traverse(root, root);
    isArticulationPoint[root] = rootChildren > 1;
  }
};
/*
Graph g;

int main() {
  int n, m; cin >> n >> m;
  g.init(n);
  for (int i = 0; i < m; i += 1) {
    int u, v; cin >> u >> v;
    g.addEdge(u, v);
  }

  // When graph is not connected
  for (int i = 1; i <= n; i += 1) if (!g.visitTime[i]) g.startFrom(i);

  int ap = 0;
  for (int i = 1; i <= n; i += 1) ap += g.isArticulationPoint[i];
  cout << ap << " " << g.bridgeCnt;
}
*/
```

sccTarjan.cpp
Description: SCC with Tarjan algo.
Time: $\mathcal{O}(V+E)$

72da39, 65 lines

```
struct Graph {
  // CAREFUL
  static constexpr int N_MAX = 1e4 + 1;

  int vt[N_MAX];
  int et[N_MAX];
  stack<int> st;
  int scc[N_MAX];
  int dfsTime = 1;
  int scc_count, n;
  bitset<N_MAX> finished;
  vector<int> graph[N_MAX];

  void addEdge(int u, int v) {
    graph[u].push_back(v);
  }

  void traverse(int u) {
    if (vt[u]) return;

    st.push(u);
    vt[u] = dfsTime;
    et[u] = dfsTime++;

    for (int v: graph[u]) if (!finished[v]) {
      if (vt[v] == 0) {
        traverse(v);
        et[u] = min(et[v], et[u]);
      }
      else et[u] = min(et[u], vt[v]);
    }

    if (vt[u] == et[u]) {
      int cur;
      do {
        cur = st.top();
        finished[cur] = 1;
        scc[cur] = scc_count;
        st.pop();
      } while (cur != u);
```

```
      scc_count += 1;
    }
  }

  void findAllScc() {
    // 1-indexed vertices
    for (int i = 1; i <= n; i += 1) traverse(i);
  }
};
/*
Graph g;

int main() {
  int m;
  cin >> g.n >> m;
  for (int i = 0; i < m; i += 1) {
    int u, v; cin >> u >> v;
    if (u != v) g.addEdge(u, v);
  }

  g.findAllScc();
  cout << g.scc_count;
}
*/
```

sccKosaraju.cpp

Description: SCC finding with kosaraju algo.
Time: $\mathcal{O}(V+E)$

3342ec, 56 lines

```
struct Graph {
  // CAREFUL
  static constexpr int N_MAX = 1e4 + 1;

  int scc[N_MAX];
  int n, scc_count;
  int finishTime = 1;
  int finishAt[N_MAX];
  bitset<N_MAX> visited;
  vector<int> graph[N_MAX];
  vector<int> revGraph[N_MAX];

  void addEdge(int u, int v) {
    graph[u].push_back(v);
    revGraph[v].push_back(u);
  }

  void markFinishTime(int u) {
    if (visited[u]) return;
    visited[u] = 1;
    for (int v: graph[u]) markFinishTime(v);
    finishAt[finishTime++] = u;
  }

  void markScc(int u) {
    if (visited[u]) return;
    visited[u] = 1;
    scc[u] = scc_count;
    for (int v: revGraph[u]) markScc(v);
  }

  void findAllScc() {
    // 1-indexed vertices
    for (int i = 1; i <= n; i += 1) markFinishTime(i);
    visited = 0;
    for (int i = n; i; i -= 1) if (!visited[finishAt[i]]) {
      markScc(finishAt[i]);
      scc_count += 1;
    }
  }
};
/*
Graph g;

int main() {
  int m;
  cin >> g.n >> m;
  for (int i = 0; i < m; i += 1) {
    int u, v; cin >> u >> v;
    if (u != v) g.addEdge(u, v);
  }

  g.findAllScc();
  cout << g.scc_count;
}
*/
```

5.2 Network Flow

Dinitz.cpp

Description: Max flow.
Time: $\mathcal{O}(V^2E)$

fe9238, 121 lines

```
struct Edge {
  int u, v;
  long long capacity;
  long long flow = 0;

  Edge() {}
  Edge(int a, int b, long long c) {
    u = a;
    v = b;
    capacity = c;
  }

  bool passable() {return capacity > flow;}
  long long remaining() {return capacity - flow;}
};

struct Network {
  // CAREFUL
  static constexpr int N_MAX = 101;
  static constexpr long long INF = 1e14;

  int n;
  int s, t;

  // bfs distance from source
  // used to determine if an edge is in layer graph
  int dist[N_MAX];

  // Next edge to send flow while DFS-ing
  // If we can't send flow through this edge, it won't be used until next layer
  // In that case, increase next[u] by 1(progress to next edge)
  int next[N_MAX];

  vector<Edge> edges;
  vector<int> edgesFrom[N_MAX];

  // Initialisation functions
  Network() {}
  void init(int _n, int _s, int _t) {
    n = _n;
    s = _s;
    t = _t;
  }

  void addEdge(int u, int v, long long w) {
    edges.push_back(Edge(u, v, w));
```

```
    edgesFrom[u].push_back(edges.size()-1);
    edges.push_back(Edge(v, u, 0));
    edgesFrom[v].push_back(edges.size()-1);
  }
  // End initialisation functions

  // BFS to create layer graph
  queue<int> bfs;
  bool pathExist() {
    while (bfs.size()) bfs.pop();
    for (int i = 0; i <= n; i += 1) if (i != s) dist[i] = -1;

    bfs.push(s);
    while (bfs.size()) {
      int u = bfs.front(); bfs.pop();
      if (u == t) break;
      for (int id: edgesFrom[u]) if (edges[id].passable()) {
        int v = edges[id].v;
        if (dist[v] != -1) continue;
        dist[v] = dist[u] + 1;
        bfs.push(v);
      }
    }

    return dist[t] != -1;
  }

  // DFS try to send a flow through each edges of a vertice
  long long flowSent(int u, long long f=INF) {
    if (u == t) return f;
    if (f == 0) return 0;

    for (int& i = next[u]; i < edgesFrom[u].size(); i += 1) {
      int id = edgesFrom[u][i];
      if (dist[edges[id].v] != dist[u] + 1) continue;
      long long attempt = flowSent(edges[id].v, min(f, edges[id].remaining()));
      if (attempt) {
        edges[id].flow += attempt;
        edges[id^1].flow -= attempt;
        return attempt;
      }
    }

    return 0;
  }

  // Dinitz algorithm
  long long maxFlow() {
    long long res = 0;
    while (pathExist()) {
      for (int i = 0; i <= n; i += 1) next[i] = 0;
      while (true) {
        long long f = flowSent(s);
        if (f == 0) break;
        res += f;
      }
    }
    return res;
  }
};
/*
Network graph;

int main() {
  graph.init(6, 0, 5);
  int m = 9;
  for (int i = 0; i < m; i += 1) {
    int u, v, c;
    cin >> u >> v >> c;
    graph.addEdge(u, v, c);
  }

  cout << graph.maxFlow();
}
*/
```

5.3 Math

5.3.1 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).

5.3.2 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 $k = 1 \ldots n$,

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

Number Theory (6)

modInverse.cpp
Description: Only works if m is prime
Time: $\mathcal{O}(50)$ 1ab4af, 3 lines

```
long long m_inv(long long a, long long m) {
  return a <= 1 ? a : m - (m/a)*inv(m%a) % m;
}
```

Geometry (7)

angle.cpp
Description: rad - deg converter.
Time: $\mathcal{O}(1)$ 1865c3, 2 lines

```
long double deg_to_rad(long double d) {return d*M_PI/180.0;}
long double rad_to_deg(long double r) {return r*180.0/M_PI;}
```

point.cpp
Description: Point and Vector.
Time: N/A 88d9f1, 48 lines

```
#define Vector Point

template<class T>
struct Point {
  // CAREFUL
  static constexpr T eps = 1e-9;

  T x, y;
  Point() {}
  Point(T _x, T _y) {x = _x; y = _y;}

  // Comparison
  bool operator<(Point p) const {
    if (abs(x - p.x) > eps) return x < p.x;
    return y < p.y;
  }

  bool operator==(Point p) const {
```

```
    return (abs(x-p.x) < eps) && (abs(y-p.y) < eps);
  }

  // Calculation
  T operator&(Vector v) const {return x*v.y - y*v.x;}
  T operator*(Vector v) const {return x*v.x + y*v.y;}
  Vector operator*(T d) const {return Vector(x*d, y*d);}
  Vector operator/(T d) const {return Vector(x/d, y/d);}
  Vector operator+(Vector v) const {return Vector(x+v.x,y+v.y);}
  Vector operator-(Vector v) const {return Vector(x-v.x,y-v.y);}

  void operator*=(T d) {x *= d; y *= d;}
  void operator/=(T d) {x /= d; y /= d;}
  void operator+=(Vector v) {x += v.x; y += v.y;}
  void operator-=(Vector v) {x -= v.x; y -= v.y;}

  T len2() {return x*x + y*y;}
  long double len() {return sqrt((long double)(len2()));}
  T dist2(Point p) {return (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y);}
  long double dist(Point p) {return sqrt(dist2(p));}

  // Rotate to the left
  Point rotated(long double a)
  {return Point(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a));}
  void rotate(long double a) {
    T cx = x*cos(a) - y*sin(a);
    T cy = x*sin(a) + y*cos(a);
    x = cx; y = cy;
  }
};
```