”Alexandru Ioan Cuza” University of Iasi

Clown Fiesta

Valeriu Motroi, Cristian Vintur, Alexandru Ionita

ACM-ICPC World Finals 2019

April 2019

Contest (1)

templateAlex.cpp
28 lines

```
#include <bits/stdc++.h>
// #include <ext/pb_ds/assoc_container.hpp> //required
// #include <ext/pb_ds/tree_policy.hpp> //required

#define dbg(x) cerr<<#x": "<<x<<"\n"
#define dbg_v(x, n) do{cerr<<#x"[]: ";for(int _=0;_<n;++_)cerr
    <<x[_]<<" ";cerr<<'\n';}while(0)
#define dbg_ok cerr<<"OK!\n"

#define all(v) v.begin(), v.end()
#define st first
#define nd second

// using namespace __gnu_pbds;
using namespace std;
using ll = long long;
using pii = pair<int, int>;

// template <typename T> using ordered_set = tree<T, null_type
    , less<T>, rb_tree_tag, tree_order_statistics_node_update
    >; // ordered_set <int> s;
template<class T> ostream& prnt(ostream& out, T v) { out << v.
    size() << '\n'; for(auto e : v) out << e << ' '; return
    out;}
template<class T> ostream& operator<<(ostream& out, vector <T>
    v) { return prnt(out, v); }
template<class T> ostream& operator<<(ostream& out, set <T> v)
    { return prnt(out, v); }
template<class T1, class T2> ostream& operator<<(ostream& out,
    map <T1, T2> v) { return prnt(out, v); }
template<class T1, class T2> ostream& operator<<(ostream& out,
    pair<T1, T2> p) { return out << '(' << p.st << ' ' << p.nd
     << ')'; }

int main() {
  ios_base::sync_with_stdio(false);
  return 0;
}
```

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 datastructures 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 team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.

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

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

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

stresstest.sh
31 lines

```
#/bin/bash

i=0
while true
do
  #python3 gen.py >in
  #./gen >in
  ./generators/graph >in
  ./c <in >out
  ./d <in >ok
  #python3 verif.py

  #if [ $? -eq 1 ]; then
  #   echo $?
  #   exit 1
  #fi

  if ! diff out ok; then
    echo $?
    exit 1
  fi

  #if ((i == 1000)); then
  # exit 0
  #fi

  let i=i+1
  if ((i % 1 == 0)); then
    echo $i
  fi
done
```

Mathematics (2)

2.1 Equations

Theoretical Computer Science Cheat Sheet

Definitions

$f(n)=O(g(n))$	iff ∃ positive c, n_0 such that $0 \le f(n) \le cg(n)\ \forall n \ge n_0$.
$f(n)=\Omega(g(n))$	iff ∃ positive c, n_0 such that $f(n) \ge cg(n) \ge 0\ \forall n \ge n_0$.
$f(n)=\Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n)=\Omega(g(n))$.
$f(n)=o(g(n))$	iff $\lim_{n\to\infty} f(n)/g(n) = 0$.
$\lim_{n\to\infty} a_n = a$	iff $\forall \epsilon > 0$, $\exists n_0$ such that $\lvert a_n - a\rvert < \epsilon$, $\forall n \ge n_0$.
$\sup S$	least $b \in \mathbb{R}$ such that $b \ge s$, $\forall s \in S$.
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \le s$, $\forall s \in S$.
$\liminf_{n\to\infty} a_n$	$\lim_{n\to\infty} \inf\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\limsup_{n\to\infty} a_n$	$\lim_{n\to\infty} \sup\{a_i \mid i \ge n, i \in \mathbb{N}\}$.
$\binom{n}{k}$	Combinations: Size k subsets of a size n set.
${n \brack k}$	Stirling numbers (1st kind): Arrangements of an n element set into k cycles.
${n \brace k}$	Stirling numbers (2nd kind): Partitions of an n element set into k non-empty sets.
$\left\langle {n \atop k} \right\rangle$	1st order Eulerian numbers: Permutations $\pi_1\pi_2\dots\pi_n$ on $\{1,2,\dots,n\}$ with k ascents.
$\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle$	2nd order Eulerian numbers.
C_n	Catalan Numbers: Binary trees with $n+1$ vertices.

Series

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \qquad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^n i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^n \left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

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

Geometric series:

$$\sum_{i=0}^n c^i = \frac{c^{n+1}-1}{c-1}, \quad c \ne 1, \qquad \sum_{i=0}^\infty c^i = \frac{1}{1-c}, \qquad \sum_{i=1}^\infty c^i = \frac{c}{1-c}, \quad |c| < 1,$$

$$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \ne 1, \qquad \sum_{i=0}^\infty ic^i = \frac{c}{(1-c)^2}, \quad |c| < 1.$$

Harmonic series:

$$H_n = \sum_{i=1}^n \frac{1}{i}, \qquad \sum_{i=1}^n iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^n H_i = (n+1)H_n - n, \qquad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

Identities

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^n \binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k\le n} \binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^n \binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}$, **11.** ${n \brace 1} = {n \brace n} = 1$,

12. ${n \brace 2} = 2^{n-1} - 1$, **13.** ${n \brace k} = k{n-1 \brace k} + {n-1 \brace k-1}$,

14. ${n \brack 1} = (n-1)!$, **15.** ${n \brack 2} = (n-1)!H_{n-1}$, **16.** ${n \brack n} = 1$, **17.** ${n \brack k} \ge {n \brace k}$,

18. ${n \brack k} = (n-1){n-1 \brack k} + {n-1 \brack k-1}$, **19.** ${n \brace n-1} = {n \brack n-1} = \binom{n}{2}$, **20.** $\sum_{k=0}^n {n \brack k} = n!$, **21.** $C_n = \frac{1}{n+1}\binom{2n}{n}$,

22. $\left\langle {n \atop 0} \right\rangle = \left\langle {n \atop n-1} \right\rangle = 1$, **23.** $\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-1-k} \right\rangle$, **24.** $\left\langle {n \atop k} \right\rangle = (k+1)\left\langle {n-1 \atop k} \right\rangle + (n-k)\left\langle {n-1 \atop k-1} \right\rangle$,

25. $\left\langle {0 \atop k} \right\rangle = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases}$ **26.** $\left\langle {n \atop 1} \right\rangle = 2^n - n - 1$, **27.** $\left\langle {n \atop 2} \right\rangle = 3^n - (n+1)2^n + \binom{n+1}{2}$,

28. $x^n = \sum_{k=0}^n \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$, **29.** $\left\langle {n \atop m} \right\rangle = \sum_{k=0}^m \binom{n+1}{k}(m+1-k)^n(-1)^k$, **30.** $m!{n \brace m} = \sum_{k=0}^n \left\langle {n \atop k} \right\rangle \binom{k}{n-m}$,

31. $\left\langle {n \atop m} \right\rangle = \sum_{k=0}^n {n \brace k}\binom{n-k}{m}(-1)^{n-k-m}k!$, **32.** $\left\langle\!\!\left\langle {n \atop 0} \right\rangle\!\!\right\rangle = 1$, **33.** $\left\langle\!\!\left\langle {n \atop n} \right\rangle\!\!\right\rangle = 0$ for $n \ne 0$,

34. $\left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = (k+1)\left\langle\!\!\left\langle {n-1 \atop k} \right\rangle\!\!\right\rangle + (2n-1-k)\left\langle\!\!\left\langle {n-1 \atop k-1} \right\rangle\!\!\right\rangle$, **35.** $\sum_{k=0}^n \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle = \frac{(2n)^{\underline{n}}}{2^n}$,

36. ${x \brace x-n} = \sum_{k=0}^n \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+n-1-k}{2n}$, **37.** ${n+1 \brace m+1} = \sum_k \binom{n}{k}{k \brace m} = \sum_{k=0}^n {k \brace m}(m+1)^{n-k}$,

Theoretical Computer Science Cheat Sheet

Identities Cont.

38. ${n+1 \brack m+1} = \sum_k {n \brack k}\binom{k}{m} = \sum_{k=0}^n {k \brack m} n^{\underline{n-k}} = n!\sum_{k=0}^n \frac{1}{k!}{k \brack m}$, **39.** ${x \brack x-n} = \sum_{k=0}^n \left\langle\!\!\left\langle {n \atop k} \right\rangle\!\!\right\rangle \binom{x+k}{2n}$,

40. ${n \brace m} = \sum_k \binom{n}{k}{k+1 \brace m+1}(-1)^{n-k}$, **41.** ${n \brack m} = \sum_k {n+1 \brack k+1}\binom{k}{m}(-1)^{m-k}$,

42. ${m+n+1 \brace m} = \sum_{k=0}^m k{n+k \brace k}$, **43.** ${m+n+1 \brack m} = \sum_{k=0}^m k(n+k){n+k \brack k}$,

44. $\binom{n}{m} = \sum_k {n+1 \brace k+1}{k \brack m}(-1)^{m-k}$, **45.** $(n-m)!\binom{n}{m} = \sum_k {n+1 \brack k+1}{k \brace m}(-1)^{m-k}$, for $n \ge m$,

46. ${n \brace n-m} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}{m+k \brack k}$, **47.** ${n \brack n-m} = \sum_k \binom{m-n}{m+k}\binom{m+n}{n+k}{m+k \brace k}$,

48. ${n \brace \ell+m}\binom{\ell+m}{\ell} = \sum_k {k \brace \ell}{n-k \brace m}\binom{n}{k}$, **49.** ${n \brack \ell+m}\binom{\ell+m}{\ell} = \sum_k {k \brack \ell}{n-k \brack m}\binom{n}{k}$.

Trees

Every tree with n vertices has $n-1$ edges.

Kraft inequality: If the depths of the leaves of a binary tree are $d_1, \dots, d_n$:

$$\sum_{i=1}^n 2^{-d_i} \le 1,$$

and equality holds only if every internal node has 2 sons.

Recurrences

Master method:
$T(n) = aT(n/b) + f(n), \quad a \ge 1, b > 1$

If $\exists \epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$ then

$$T(n) = \Theta(n^{\log_b a}).$$

If $f(n) = \Theta(n^{\log_b a})$ then

$$T(n) = \Theta(n^{\log_b a}\log_2 n).$$

If $\exists \epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and $\exists c < 1$ such that $af(n/b) \le cf(n)$ for large n, then

$$T(n) = \Theta(f(n)).$$

Substitution (example): Consider the following recurrence

$$T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.$$

Note that T_i is always a power of two. Let $t_i = \log_2 T_i$. Then we have

$$t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.$$

Let $u_i = t_i/2^i$. Dividing both sides of the previous equation by 2^{i+1} we get

$$\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.$$

Substituting we find

$$u_{i+1} = \tfrac{1}{2} + u_i, \qquad u_1 = \tfrac{1}{2},$$

which is simply $u_i = i/2$. So we find that T_i has the closed form $T_i = 2^{i2^{i-1}}$. Summing factors (example): Consider the following recurrence

$$T(n) = 3T(n/2) + n, \quad T(1) = 1.$$

Rewrite so that all terms involving T are on the left side

$$T(n) - 3T(n/2) = n.$$

Now expand the recurrence, and choose a factor which makes the left side "telescope"

$$1\big(T(n) - 3T(n/2) = n\big)$$
$$3\big(T(n/2) - 3T(n/4) = n/2\big)$$
$$\vdots \quad \vdots \quad \vdots$$
$$3^{\log_2 n - 1}\big(T(2) - 3T(1) = 2\big)$$

Let $m = \log_2 n$. Summing the left side we get $T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k$ where $k = \log_2 3 \approx 1.58496$. Summing the right side we get

$$\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n\sum_{i=0}^{m-1}\left(\tfrac{3}{2}\right)^i.$$

Let $c = \frac{3}{2}$. Then we have

$$n\sum_{i=0}^{m-1} c^i = n\left(\frac{c^m - 1}{c-1}\right)$$
$$= 2n(c^{\log_2 n} - 1)$$
$$= 2n(c^{(k-1)\log_c n} - 1)$$
$$= 2n^k - 2n,$$

and so $T(n) = 3n^k - 2n$. Full history recurrences can often be changed to limited history ones (example): Consider

$$T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.$$

Note that

$$T_{i+1} = 1 + \sum_{j=0}^{i} T_j.$$

Subtracting we find

$$T_{i+1} - T_i = 1 + \sum_{j=0}^{i} T_j - 1 - \sum_{j=0}^{i-1} T_j$$
$$= T_i.$$

And so $T_{i+1} = 2T_i = 2^{i+1}$.

Generating functions:

1. Multiply both sides of the equation by x^i.
2. Sum both sides over all i for which the equation is valid.
3. Choose a generating function $G(x)$. Usually $G(x) = \sum_{i=0}^\infty x^i g_i$.
3. Rewrite the equation in terms of the generating function $G(x)$.
4. Solve for $G(x)$.
5. The coefficient of x^i in $G(x)$ is g_i.

Example:

$$g_{i+1} = 2g_i + 1, \quad g_0 = 0.$$

Multiply and sum:

$$\sum_{i\ge0} g_{i+1}x^i = \sum_{i\ge0} 2g_i x^i + \sum_{i\ge0} x^i.$$

We choose $G(x) = \sum_{i\ge0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i\ge0} x^i.$$

Simplify:

$$\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.$$

Solve for $G(x)$:

$$G(x) = \frac{x}{(1-x)(1-2x)}.$$

Expand this using partial fractions:

$$G(x) = x\left(\frac{2}{1-2x} - \frac{1}{1-x}\right)$$
$$= x\left(2\sum_{i\ge0} 2^i x^i - \sum_{i\ge0} x^i\right)$$
$$= \sum_{i\ge0}(2^{i+1} - 1)x^{i+1}.$$

So $g_i = 2^i - 1$.

Theoretical Computer Science Cheat Sheet

$\pi \approx 3.14159$, $e \approx 2.71828$, $\gamma \approx 0.57721$, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$, $\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -.61803$

i	2^i	p_i
1	2	2
2	4	3
3	8	5
4	16	7
5	32	11
6	64	13
7	128	17
8	256	19
9	512	23
10	1,024	29
11	2,048	31
12	4,096	37
13	8,192	41
14	16,384	43
15	32,768	47
16	65,536	53
17	131,072	59
18	262,144	61
19	524,288	67
20	1,048,576	71
21	2,097,152	73
22	4,194,304	79
23	8,388,608	83
24	16,777,216	89
25	33,554,432	97
26	67,108,864	101
27	134,217,728	103
28	268,435,456	107
29	536,870,912	109
30	1,073,741,824	113
31	2,147,483,648	127
32	4,294,967,296	131

Pascal's Triangle

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

General

Bernoulli Numbers ($B_i = 0$, odd $i \neq 1$):

$$B_0 = 1,\ B_1 = -\tfrac{1}{2},\ B_2 = \tfrac{1}{6},\ B_4 = -\tfrac{1}{30},$$
$$B_6 = \tfrac{1}{42},\ B_8 = -\tfrac{1}{30},\ B_{10} = \tfrac{5}{66}.$$

Change of base, quadratic formula:

$$\log_b x = \frac{\log_a x}{\log_a b}, \qquad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$
$$\lim_{n\to\infty}\left(1 + \frac{x}{n}\right)^n = e^x.$$
$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$
$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

$$1,\ \tfrac{3}{2},\ \tfrac{11}{6},\ \tfrac{25}{12},\ \tfrac{137}{60},\ \tfrac{49}{20},\ \tfrac{363}{140},\ \tfrac{761}{280},\ \tfrac{7129}{2520},\ \ldots$$

$$\ln n < H_n < \ln n + 1,$$
$$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$$

Factorial, Stirling's approximation:

1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$$

Ackermann's function and inverse:

$$a(i,j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$$
$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{n} k \binom{n}{k} p^k q^{n-k} = np.$$

Poisson distribution:

$$\Pr[X = k] = \frac{e^{-\lambda}\lambda^k}{k!}, \quad \mathrm{E}[X] = \lambda.$$

Normal (Gaussian) distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad \mathrm{E}[X] = \mu.$$

The "coupon collector": We are given a random coupon each day, and there are n different types of coupons. The distribution of coupons is uniform. The expected number of days to pass before we to collect all n types is

$$nH_n.$$

Probability

Continuous distributions: If

$$\Pr[a < X < b] = \int_a^b p(x)\,dx,$$

then p is the probability density function of X. If

$$\Pr[X < a] = P(a),$$

then P is the distribution function of X. If P and p both exist then

$$P(a) = \int_{-\infty}^{a} p(x)\,dx.$$

Expectation: If X is discrete

$$\mathrm{E}[g(X)] = \sum_x g(x)\Pr[X = x].$$

If X continuous then

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x)p(x)\,dx = \int_{-\infty}^{\infty} g(x)\,dP(x).$$

Variance, standard deviation:

$$\mathrm{VAR}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2,$$
$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

$$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$$
$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

$$\Pr[A|B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$$

For random variables X and Y:

$$\mathrm{E}[X \cdot Y] = \mathrm{E}[X] \cdot \mathrm{E}[Y],$$

if X and Y are independent.

$$\mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y],$$
$$\mathrm{E}[cX] = c\,\mathrm{E}[X].$$

Bayes' theorem:

$$\Pr[A_i|B] = \frac{\Pr[B|A_i]\Pr[A_i]}{\sum_{j=1}^{n}\Pr[A_j]\Pr[B|A_j]}.$$

Inclusion-exclusion:

$$\Pr\left[\bigvee_{i=1}^{n} X_i\right] = \sum_{i=1}^{n}\Pr[X_i] +$$
$$\sum_{k=2}^{n}(-1)^{k+1}\sum_{i_i < \cdots < i_k}\Pr\left[\bigwedge_{j=1}^{k} X_{i_j}\right].$$

Moment inequalities:

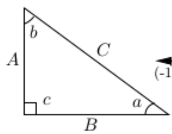
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

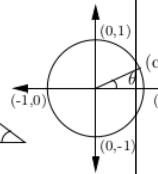
Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$
$$\mathrm{E}[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$$

Theoretical Computer Science Cheat Sheet

Trigonometry





Pythagorean theorem:

$$C^2 = A^2 + B^2.$$

Definitions:

$$\sin a = A/C, \quad \cos a = B/C,$$
$$\csc a = C/A, \quad \sec a = C/B,$$
$$\tan a = \frac{\sin a}{\cos a} = \frac{A}{B}, \quad \cot a = \frac{\cos a}{\sin a} = \frac{B}{A}.$$

Area, radius of inscribed circle:

$$\tfrac{1}{2}AB, \quad \frac{AB}{A + B + C}.$$

Identities:

$$\sin x = \frac{1}{\csc x}, \qquad \cos x = \frac{1}{\sec x},$$
$$\tan x = \frac{1}{\cot x}, \qquad \sin^2 x + \cos^2 x = 1,$$
$$1 + \tan^2 x = \sec^2 x, \qquad 1 + \cot^2 x = \csc^2 x,$$
$$\sin x = \cos\left(\tfrac{\pi}{2} - x\right), \qquad \sin x = \sin(\pi - x),$$
$$\cos x = -\cos(\pi - x), \qquad \tan x = \cot\left(\tfrac{\pi}{2} - x\right),$$
$$\cot x = -\cot(\pi - x), \qquad \csc x = \cot\tfrac{x}{2} - \cot x,$$
$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$
$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$
$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$
$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$
$$\sin 2x = 2\sin x \cos x, \qquad \sin 2x = \frac{2\tan x}{1 + \tan^2 x},$$
$$\cos 2x = \cos^2 x - \sin^2 x, \qquad \cos 2x = 2\cos^2 x - 1,$$
$$\cos 2x = 1 - 2\sin^2 x, \qquad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$
$$\tan 2x = \frac{2\tan x}{1 - \tan^2 x}, \qquad \cot 2x = \frac{\cot^2 x - 1}{2\cot x},$$
$$\sin(x + y)\sin(x - y) = \sin^2 x - \sin^2 y,$$
$$\cos(x + y)\cos(x - y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

$$e^{ix} = \cos x + i\sin x, \qquad e^{i\pi} = -1.$$

v2.02 ©1994 by Steve Seiden
sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Matrices

Multiplication:

$$C = A \cdot B, \quad c_{i,j} = \sum_{k=1}^{n} a_{i,k}b_{k,j}.$$

Determinants: $\det A \neq 0$ iff A is non-singular.

$$\det A \cdot B = \det A \cdot \det B,$$
$$\det A = \sum_{\pi}\prod_{i=1}^{n}\mathrm{sign}(\pi)a_{i,\pi(i)}.$$

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$
$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g\begin{vmatrix} b & c \\ e & f \end{vmatrix} - h\begin{vmatrix} a & c \\ d & f \end{vmatrix} + i\begin{vmatrix} a & b \\ d & e \end{vmatrix}$$
$$= \begin{matrix} aei + bfg + cdh \\ -ceg - fha - ibd. \end{matrix}$$

Permanents:

$$\mathrm{perm}\,A = \sum_{\pi}\prod_{i=1}^{n} a_{i,\pi(i)}.$$

Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \mathrm{csch}\,x = \frac{1}{\sinh x},$$
$$\mathrm{sech}\,x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

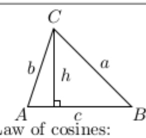
Identities:

$$\cosh^2 x - \sinh^2 x = 1, \qquad \tanh^2 x + \mathrm{sech}^2 x = 1,$$
$$\coth^2 x - \mathrm{csch}^2 x = 1, \qquad \sinh(-x) = -\sinh x,$$
$$\cosh(-x) = \cosh x, \qquad \tanh(-x) = -\tanh x,$$
$$\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$$
$$\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$$
$$\sinh 2x = 2\sinh x \cosh x,$$
$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$
$$\cosh x + \sinh x = e^x, \qquad \cosh x - \sinh x = e^{-x},$$
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$
$$2\sinh^2\tfrac{x}{2} = \cosh x - 1, \qquad 2\cosh^2\tfrac{x}{2} = \cosh x + 1.$$

θ	$\sin\theta$	$\cos\theta$	$\tan\theta$
0	0	1	0
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$	1
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$	$\sqrt{3}$
$\frac{\pi}{2}$	1	0	∞

... in mathematics you don't understand things, you just get used to them.
– J. von Neumann

More Trig.



Law of cosines:

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

Area:

$$A = \tfrac{1}{2}hc,$$
$$= \tfrac{1}{2}ab\sin C,$$
$$= \frac{c^2 \sin A \sin B}{2\sin C}.$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$
$$s = \tfrac{1}{2}(a + b + c),$$
$$s_a = s - a,$$
$$s_b = s - b,$$
$$s_c = s - c.$$

More identities:

$$\sin\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$
$$\cos\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$
$$\tan\tfrac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$
$$= \frac{1 - \cos x}{\sin x},$$
$$= \frac{\sin x}{1 + \cos x},$$
$$\cot\tfrac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$
$$= \frac{1 + \cos x}{\sin x},$$
$$= \frac{\sin x}{1 - \cos x},$$
$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$
$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$
$$\tan x = -i\frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$
$$= -i\frac{e^{2ix} - 1}{e^{2ix} + 1},$$
$$\sin x = \frac{\sinh ix}{i},$$
$$\cos x = \cosh ix,$$
$$\tan x = \frac{\tanh ix}{i}.$$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

15. $\int \arccos \frac{x}{a} dx = \arccos \frac{x}{a} - \sqrt{a^2 - x^2}, \quad a > 0,$

16. $\int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2), \quad a > 0,$

17. $\int \sin^2(ax) dx = \frac{1}{2a}\big(ax - \sin(ax)\cos(ax)\big),$

18. $\int \cos^2(ax) dx = \frac{1}{2a}\big(ax + \sin(ax)\cos(ax)\big),$

19. $\int \sec^2 x \, dx = \tan x,$

20. $\int \csc^2 x \, dx = -\cot x,$

21. $\int \sin^n x \, dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x \, dx,$

22. $\int \cos^n x \, dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x \, dx,$

23. $\int \tan^n x \, dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x \, dx, \quad n \neq 1,$

24. $\int \cot^n x \, dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x \, dx, \quad n \neq 1,$

25. $\int \sec^n x \, dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x \, dx, \quad n \neq 1,$

26. $\int \csc^n x \, dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x \, dx, \quad n \neq 1,$

27. $\int \sinh x \, dx = \cosh x,$

28. $\int \cosh x \, dx = \sinh x,$

29. $\int \tanh x \, dx = \ln |\cosh x|,$

30. $\int \coth x \, dx = \ln |\sinh x|,$

31. $\int \operatorname{sech} x \, dx = \arctan \sinh x,$

32. $\int \operatorname{csch} x \, dx = \ln \left|\tanh \frac{x}{2}\right|,$

33. $\int \sinh^2 x \, dx = \frac{1}{4} \sinh(2x) - \frac{1}{2}x,$

34. $\int \cosh^2 x \, dx = \frac{1}{4} \sinh(2x) + \frac{1}{2}x,$

35. $\int \operatorname{sech}^2 x \, dx = \tanh x,$

36. $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$

37. $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$

38. $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$

39. $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln\left(x + \sqrt{a^2 + x^2}\right), \quad a > 0,$

40. $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$

41. $\int \sqrt{a^2 - x^2} \, dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

42. $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8}(5a^2 - 2x^2)\sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

43. $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$

44. $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left|\frac{a + x}{a - x}\right|,$

45. $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2\sqrt{a^2 - x^2}},$

46. $\int \sqrt{a^2 \pm x^2} \, dx = \frac{x}{2}\sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left|x + \sqrt{a^2 \pm x^2}\right|,$

47. $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left|x + \sqrt{x^2 - a^2}\right|, \quad a > 0,$

48. $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left|\frac{x}{a + bx}\right|,$

49. $\int x\sqrt{a + bx} \, dx = \frac{2(3bx - 2a)(a + bx)^{3/2}}{15b^2},$

50. $\int \frac{\sqrt{a + bx}}{x} dx = 2\sqrt{a + bx} + a \int \frac{1}{x\sqrt{a + bx}} dx,$

51. $\int \frac{x}{\sqrt{a + bx}} dx = \frac{1}{\sqrt{2}} \ln \left|\frac{\sqrt{a + bx} - \sqrt{a}}{\sqrt{a + bx} + \sqrt{a}}\right|, \quad a > 0,$

52. $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

53. $\int x\sqrt{a^2 - x^2} \, dx = -\frac{1}{3}(a^2 - x^2)^{3/2},$

54. $\int x^2\sqrt{a^2 - x^2} \, dx = \frac{x}{8}(2x^2 - a^2)\sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$

55. $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left|\frac{a + \sqrt{a^2 - x^2}}{x}\right|,$

56. $\int \frac{x \, dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$

57. $\int \frac{x^2 \, dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$

58. $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left|\frac{a + \sqrt{a^2 + x^2}}{x}\right|,$

59. $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$

60. $\int x\sqrt{x^2 \pm a^2} \, dx = \frac{1}{3}(x^2 \pm a^2)^{3/2},$

61. $\int \frac{dx}{x\sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left|\frac{x}{a + \sqrt{a^2 + x^2}}\right|,$

Theoretical Computer Science Cheat Sheet

Calculus Cont.

62. $\int \frac{dx}{x\sqrt{x^2 - a^2}} = \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0,$

63. $\int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} = \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x},$

64. $\int \frac{x \, dx}{\sqrt{x^2 \pm a^2}} = \sqrt{x^2 \pm a^2},$

65. $\int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx = \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3},$

66. $\int \frac{dx}{ax^2 + bx + c} = \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left|\frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}}\right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases}$

67. $\int \frac{dx}{\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{1}{\sqrt{a}} \ln \left|2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c}\right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases}$

68. $\int \sqrt{ax^2 + bx + c} \, dx = \frac{2ax + b}{4a}\sqrt{ax^2 + bx + c} + \frac{4ax - b^2}{8a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$

69. $\int \frac{x \, dx}{\sqrt{ax^2 + bx + c}} = \frac{\sqrt{ax^2 + bx + c}}{a} - \frac{b}{2a} \int \frac{dx}{\sqrt{ax^2 + bx + c}},$

70. $\int \frac{dx}{x\sqrt{ax^2 + bx + c}} = \begin{cases} \frac{-1}{\sqrt{c}} \ln \left|\frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x}\right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases}$

71. $\int x^3\sqrt{x^2 + a^2} \, dx = (\frac{1}{3}x^2 - \frac{2}{15}a^2)(x^2 + a^2)^{3/2},$

72. $\int x^n \sin(ax) \, dx = -\frac{1}{a}x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) \, dx,$

73. $\int x^n \cos(ax) \, dx = \frac{1}{a}x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) \, dx,$

74. $\int x^n e^{ax} \, dx = \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} \, dx,$

75. $\int x^n \ln(ax) \, dx = x^{n+1}\left(\frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2}\right),$

76. $\int x^n (\ln ax)^m \, dx = \frac{x^{n+1}}{n+1}(\ln ax)^m - \frac{m}{n+1} \int x^n (\ln ax)^{m-1} \, dx.$

$$
\begin{aligned}
x^1 &= x^{\underline{1}} &&= x^{\overline{1}} \\
x^2 &= x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{2}} - x^{\overline{1}} \\
x^3 &= x^{\underline{3}} + 3x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{3}} - 3x^{\overline{2}} + x^{\overline{1}} \\
x^4 &= x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}} &&= x^{\overline{4}} - 6x^{\overline{3}} + 7x^{\overline{2}} - x^{\overline{1}} \\
x^5 &= x^{\underline{5}} + 15x^{\underline{4}} + 25x^{\underline{3}} + 10x^{\underline{2}} + x^{\underline{1}} &&= x^{\underline{5}} - 15x^{\overline{4}} + 25x^{\overline{3}} - 10x^{\overline{2}} + x^{\overline{1}}
\end{aligned}
$$

$$
\begin{aligned}
x^{\overline{1}} &= x^1 & x^{\underline{1}} &= x^1 \\
x^{\overline{2}} &= x^2 + x^1 & x^{\underline{2}} &= x^2 - x^1 \\
x^{\overline{3}} &= x^3 + 3x^2 + 2x^1 & x^{\underline{3}} &= x^3 - 3x^2 + 2x^1 \\
x^{\overline{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^{\underline{4}} &= x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\overline{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\underline{5}} &= x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}
$$

Finite Calculus

Difference, shift operators:

$$\Delta f(x) = f(x+1) - f(x),$$
$$\mathrm{E}\, f(x) = f(x+1).$$

Fundamental Theorem:

$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$

$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

$\Delta(cu) = c\Delta u, \qquad \Delta(u + v) = \Delta u + \Delta v,$

$\Delta(uv) = u\Delta v + \mathrm{E}\, v\Delta u,$

$\Delta(x^{\underline{n}}) = nx^{\underline{n-1}},$

$\Delta(H_x) = x^{\underline{-1}}, \qquad \Delta(2^x) = 2^x,$

$\Delta(c^x) = (c-1)c^x, \qquad \Delta\binom{x}{m} = \binom{x}{m-1}.$

Sums:

$\sum cu \, \delta x = c \sum u \, \delta x,$

$\sum (u + v) \, \delta x = \sum u \, \delta x + \sum v \, \delta x,$

$\sum u\Delta v \, \delta x = uv - \sum \mathrm{E}\, v\Delta u \, \delta x,$

$\sum x^{\underline{n}} \, \delta x = \frac{x^{\underline{n+1}}}{m+1}, \qquad \sum x^{\underline{-1}} \, \delta x = H_x,$

$\sum c^x \, \delta x = \frac{c^x}{c-1}, \qquad \sum \binom{x}{m} \, \delta x = \binom{x}{m+1}.$

Falling Factorial Powers:

$x^{\underline{n}} = x(x-1)\cdots(x-n+1), \quad n > 0,$

$x^{\underline{0}} = 1,$

$x^{\underline{n}} = \frac{1}{(x+1)\cdots(x+|n|)}, \quad n < 0,$

$x^{\underline{n+m}} = x^{\underline{m}}(x-m)^{\underline{n}}.$

Rising Factorial Powers:

$x^{\overline{n}} = x(x+1)\cdots(x+n-1), \quad n > 0,$

$x^{\overline{0}} = 1,$

$x^{\overline{n}} = \frac{1}{(x-1)\cdots(x-|n|)}, \quad n < 0,$

$x^{\overline{n+m}} = x^{\overline{m}}(x+m)^{\overline{n}}.$

Conversion:

$x^{\underline{n}} = (-1)^n(-x)^{\overline{n}} = (x-n+1)^{\overline{n}}$

$\quad = 1/(x+1)^{\overline{-n}},$

$x^{\overline{n}} = (-1)^n(-x)^{\underline{n}} = (x+n-1)^{\underline{n}}$

$\quad = 1/(x-1)^{\underline{-n}},$

$$x^n = \sum_{k=1}^n \left\{ n \atop k \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ n \atop k \right\} (-1)^{n-k} x^{\overline{k}},$$

$$x^{\underline{n}} = \sum_{k=1}^n \left[n \atop k \right] (-1)^{n-k} x^k,$$

$$x^{\overline{n}} = \sum_{k=1}^n \left[n \atop k \right] x^k.$$

Theoretical Computer Science Cheat Sheet

Series

Taylor's series:

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2}f''(a) + \cdots = \sum_{i=0}^{\infty} \frac{(x-a)^i}{i!} f^{(i)}(a).$$

Expansions:

$$\begin{aligned}
\frac{1}{1-x} &= 1 + x + x^2 + x^3 + x^4 + \cdots &&= \sum_{i=0}^{\infty} x^i, \\
\frac{1}{1-cx} &= 1 + cx + c^2x^2 + c^3x^3 + \cdots &&= \sum_{i=0}^{\infty} c^i x^i, \\
\frac{1}{1-x^n} &= 1 + x^n + x^{2n} + x^{3n} + \cdots &&= \sum_{i=0}^{\infty} x^{ni}, \\
\frac{x}{(1-x)^2} &= x + 2x^2 + 3x^3 + 4x^4 + \cdots &&= \sum_{i=0}^{\infty} i x^i, \\
\sum_{k=0}^{n} \left\{ n \atop k \right\} \frac{k! z^k}{(1-z)^{k+1}} &= x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots &&= \sum_{i=0}^{\infty} i^n x^i, \\
e^x &= 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{x^i}{i!}, \\
\ln(1+x) &= x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots &&= \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}, \\
\ln \frac{1}{1-x} &= x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots &&= \sum_{i=1}^{\infty} \frac{x^i}{i}, \\
\sin x &= x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}, \\
\cos x &= 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}, \\
\tan^{-1} x &= x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots &&= \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)}, \\
(1+x)^n &= 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{n}{i} x^i, \\
\frac{1}{(1-x)^{n+1}} &= 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{i+n}{i} x^i, \\
\frac{x}{e^x - 1} &= 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots &&= \sum_{i=0}^{\infty} \frac{B_i x^i}{i!}, \\
\frac{1}{2x}(1 - \sqrt{1-4x}) &= 1 + x + 2x^2 + 5x^3 + \cdots &&= \sum_{i=0}^{\infty} \frac{1}{i+1} \binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}} &= 1 + 2x + 6x^2 + 20x^3 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i}{i} x^i, \\
\frac{1}{\sqrt{1-4x}} \left(\frac{1-\sqrt{1-4x}}{2x} \right)^n &= 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots &&= \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i, \\
\frac{1}{1-x} \ln \frac{1}{1-x} &= x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots &&= \sum_{i=1}^{\infty} H_i x^i, \\
\frac{1}{2} \left(\ln \frac{1}{1-x} \right)^2 &= \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots &&= \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i}, \\
\frac{x}{1-x-x^2} &= x + x^2 + 2x^3 + 3x^4 + \cdots &&= \sum_{i=0}^{\infty} F_i x^i, \\
\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} &= F_n x + F_{2n} x^2 + F_{3n} x^3 + \cdots &&= \sum_{i=0}^{\infty} F_{ni} x^i.
\end{aligned}$$

Ordinary power series:

$$A(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Exponential power series:

$$A(x) = \sum_{i=0}^{\infty} a_i \frac{x^i}{i!}.$$

Dirichlet power series:

$$A(x) = \sum_{i=1}^{\infty} \frac{a_i}{i^x}.$$

Binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

Difference of like powers:

$$x^n - y^n = (x-y) \sum_{k=0}^{n-1} x^{n-1-k} y^k.$$

For ordinary power series:

$$\begin{aligned}
\alpha A(x) + \beta B(x) &= \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i, \\
x^k A(x) &= \sum_{i=k}^{\infty} a_{i-k} x^i, \\
\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} &= \sum_{i=0}^{\infty} a_{i+k} x^i, \\
A(cx) &= \sum_{i=0}^{\infty} c^i a_i x^i, \\
A'(x) &= \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i, \\
x A'(x) &= \sum_{i=1}^{\infty} i a_i x^i, \\
\int A(x)\, dx &= \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i, \\
\frac{A(x) + A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i} x^{2i}, \\
\frac{A(x) - A(-x)}{2} &= \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.
\end{aligned}$$

Summation: If $b_i = \sum_{j=0}^{i} a_i$ then

$$B(x) = \frac{1}{1-x} A(x).$$

Convolution:

$$A(x)B(x) = \sum_{i=0}^{\infty} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) x^i.$$

God made the natural numbers; all the rest is the work of man.
– Leopold Kronecker

Theoretical Computer Science Cheat Sheet

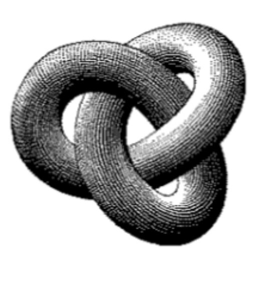
Series

Expansions:

$$\begin{aligned}
\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x} &= \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i, &\quad \left(\frac{1}{x} \right)^{\overline{-n}} &= \sum_{i=0}^{\infty} \left\{ i \atop n \right\} x^i, \\
x^{\overline{n}} &= \sum_{i=0}^{\infty} \left[n \atop i \right] x^i, & (e^x - 1)^n &= \sum_{i=0}^{\infty} \left\{ i \atop n \right\} \frac{n! x^i}{i!}, \\
\left(\ln \frac{1}{1-x} \right)^n &= \sum_{i=0}^{\infty} \left[i \atop n \right] \frac{n! x^i}{i!}, & x \cot x &= \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!}, \\
\tan x &= \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i}(2^{2i}-1) B_{2i} x^{2i-1}}{(2i)!}, & \zeta(x) &= \sum_{i=1}^{\infty} \frac{1}{i^x}, \\
\frac{1}{\zeta(x)} &= \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x}, & \frac{\zeta(x-1)}{\zeta(x)} &= \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},
\end{aligned}$$

$$\begin{aligned}
\zeta(x) &= \prod_p \frac{1}{1 - p^{-x}}, \\
\zeta^2(x) &= \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \textstyle\sum_{d|n} 1, \\
\zeta(x)\zeta(x-1) &= \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \textstyle\sum_{d|n} d, \\
\zeta(2n) &= \frac{2^{2n-1}|B_{2n}|}{(2n)!} \pi^{2n}, \quad n \in \mathbb{N}, \\
\frac{x}{\sin x} &= \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!}, \\
\left(\frac{1 - \sqrt{1-4x}}{2x} \right)^n &= \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i, \\
e^x \sin x &= \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i, \\
\sqrt{\frac{1 - \sqrt{1-x}}{x}} &= \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2} (2i)! (2i+1)!} x^i, \\
\left(\frac{\arcsin x}{x} \right)^2 &= \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.
\end{aligned}$$

Escher's Knot



Stieltjes Integration

If G is continuous in the interval $[a,b]$ and F is nondecreasing then

$$\int_a^b G(x)\, dF(x)$$

exists. If $a \le b \le c$ then

$$\int_a^c G(x)\, dF(x) = \int_a^b G(x)\, dF(x) + \int_b^c G(x)\, dF(x).$$

If the integrals involved exist

$$\begin{aligned}
\int_a^b \big(G(x) + H(x)\big)\, dF(x) &= \int_a^b G(x)\, dF(x) + \int_a^b H(x)\, dF(x), \\
\int_a^b G(x)\, d\big(F(x) + H(x)\big) &= \int_a^b G(x)\, dF(x) + \int_a^b G(x)\, dH(x), \\
\int_a^b c \cdot G(x)\, dF(x) &= \int_a^b G(x)\, d\big(c \cdot F(x)\big) = c \int_a^b G(x)\, dF(x), \\
\int_a^b G(x)\, dF(x) &= G(b)F(b) - G(a)F(a) - \int_a^b F(x)\, dG(x).
\end{aligned}$$

If the integrals involved exist, and F possesses a derivative F' at every point in $[a,b]$ then

$$\int_a^b G(x)\, dF(x) = \int_a^b G(x) F'(x)\, dx.$$

Cramer's Rule

If we have equations:

$$\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\
a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\
\vdots \qquad\quad \vdots \qquad\qquad\quad \vdots & \\
a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n
\end{aligned}$$

Let $A = (a_{i,j})$ and B be the column matrix (b_i). Then there is a unique solution iff $\det A \neq 0$. Let A_i be A with column i replaced by B. Then

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

Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius.
– William Blake (The Marriage of Heaven and Hell)

00	47	18	76	29	93	85	34	61	52
86	11	57	28	70	39	94	45	02	63
95	80	22	67	38	71	49	56	13	04
59	96	81	33	07	48	72	60	24	15
73	69	90	82	44	17	58	01	35	26
68	74	09	91	83	55	27	12	46	30
37	08	75	19	92	84	66	23	50	41
14	25	36	40	51	62	03	77	88	99
21	32	43	54	65	06	10	89	97	78
42	53	64	05	16	20	31	98	79	87

The Fibonacci number system: Every integer n has a unique representation

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$$

where $k_i \ge k_{i+1} + 2$ for all i, $1 \le i < m$ and $k_m \ge 2$.

Fibonacci Numbers

$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$

Definitions:

$$\begin{aligned}
F_i &= F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1, \\
F_{-i} &= (-1)^{i-1} F_i, \\
F_i &= \frac{1}{\sqrt{5}} \left(\phi^i - \hat{\phi}^i \right),
\end{aligned}$$

Cassini's identity: for $i > 0$:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$$

Additive rule:

$$\begin{aligned}
F_{n+k} &= F_k F_{n+1} + F_{k-1} F_n, \\
F_{2n} &= F_n F_{n+1} + F_{n-1} F_n.
\end{aligned}$$

Calculation by matrices:

$$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$$

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

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

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):

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

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

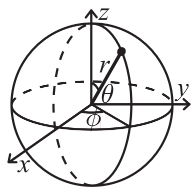
2.4.2 Quadrilaterals

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

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

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



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \operatorname{acos}(z/\sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \operatorname{atan2}(y, x) \end{aligned}$$

2.5 Linear algebra

2.5.1 Matrix inverse

The inverse of a 2x2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

In general:

$$A^{-1} = \frac{1}{det(A)} A^*$$

where $A^*_{i,j} = (-1)^{i+j} * \Delta_{i,j}$ and $\Delta_{i,j}$ is the determinant of matrix A crossing out line i and column j.

2.6 Derivatives/Integrals

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

Integration by parts:

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

2.7 Sums

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

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

2.8 Series

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

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

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

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

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

2.9 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,

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

2.9.1 Discrete distributions

Binomial distribution

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

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

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

$\mathrm{Bin}(n, p)$ 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 wich yields success with probability p is $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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

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 $\mathrm{Po}(\lambda)$, $\lambda = t\kappa$.

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

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

2.9.2 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 $\mathrm{Exp}(\lambda)$, $\lambda > 0$.

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

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

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

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

2.10 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let $X_1, X_2, \dots$ 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 $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki} t_k$.

Data structures (3)

OrderStatisticTree.h

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

16 lines

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

template <class T>
using Tree = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;

void example() {
  Tree<int> t, t2; t.insert(8);
  auto it = t.insert(10).first;
  assert(it == t.lower_bound(9));
  assert(t.order_of_key(10) == 1);
  assert(t.order_of_key(11) == 2);
  assert(*t.find_by_order(0) == 8);
  t.join(t2); // assuming T < T2 or T > T2, merge t2 into t
}
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.
Usage: `Node* tr = new Node(v, 0, sz(v));`
Time: $\mathcal{O}(\log N)$.

"../various/BumpAllocator.h" 50 lines

```
const int inf = 1e9;
struct Node {
  Node *l = 0, *r = 0;
  int lo, hi, mset = inf, madd = 0, val = -inf;
  Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
  Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
    if (lo + 1 < hi) {
      int mid = lo + (hi - lo)/2;
      l = new Node(v, lo, mid); r = new Node(v, mid, hi);
      val = max(l->val, r->val);
    }
    else val = v[lo];
  }
  int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
  }
  void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
      push(), l->set(L, R, x), r->set(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
      if (mset != inf) mset += x;
      else madd += x;
      val += x;
    }
    else {
      push(), l->add(L, R, x), r->add(L, R, x);
      val = max(l->val, r->val);
    }
  }
  void push() {
    if (!l) {
      int mid = lo + (hi - lo)/2;
      l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
      l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
      l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
  }
};
```

implicitTreapsMaxValeriu.cpp

Description: None
Usage: `ask Djok`

<bits/stdc++.h> 140 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 200005;

int i, n, q, a[N], x, y, z;

struct node;
typedef node* ln;

struct node
{
    int pr;

    int v;
    int dp;

    int id,sz;

    ln l, r;

    node (int v=0) : pr(rand() * rand() * rand()),v(v),l(0),r
         (0) { upd(); }

    void upd()
    {
        dp = v;
        if (l) dp=max(dp,l->dp);
        if (r) dp=max(dp,r->dp);

        sz = 1;
        if (l) sz+=l->sz;
        id = sz;
        if (r) sz+=r->sz;
    }
};

ln root;

void split ( ln t, int x, ln &l, ln &r)
{
    l=r=0;
    if (!t) return;
    if (t->id <= x)
    {
        split(t->r, x - t->id, t->r, r);
        l = t;
    } else
    {
        split(t->l, x, l, t->l);
        r = t;
    }
    t->upd();
}

ln merge(ln l, ln r)
{
    if (!l || !r) return (l?l:r);

    if (l->pr > r->pr)
    {
        l->r = merge(l->r, r);
        l->upd();
        return l;
    } else
    {
        r->l = merge(l,r->l);
        r->upd();
        return r;
    }
}

void insert(int x, int p)
{
    ln l,r;
    split(root,p,l,r);
    root = merge(merge(l,new node(x)),r);
}

void erase(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    root = merge(l,t);
}

int query(int x, int y)
{
    ln l,t,r;
    split(root, x, l, t);
    split(t, y-x+1, t, r);

    int m = t->dp;

    root = merge(merge(l,t),r);
    return m;
}

void show(ln t)
{
    if (!t) return;
    show(t->l);
    cout<<' '<<t->v;
    show(t->r);
}

int getPoz(int p)
{
    ln l,r,t;
    split(root,p,l,r);
    split(r,1,r,t);
    int ans = r->v;
    r = merge(r,t);
    root = merge(l, r);
    return ans;
}

int main() {
  srand(time(0));
```

```
  root = 0;
  scanf("%d %d", &n, &q);
  for(i = 0; i < n; ++i) scanf("%d", a + i), insert(a[i], i);
  while(q--) {
    scanf("%d %d %d", &x, &y, &z);
    if(x == 1) {
      printf("%d\n", query(y - 1, z - 1));
      continue;
    }

    --z; x = getPoz(z);
    erase(z);
    if(y == 1) {
      insert(x, n - 1);
    } else {
      insert(x, 0);
    }
  }
  return 0;
}
```

convexhulltrick.cpp

Description: Add lines of the form ax+b and query maximum. Lines should be sorted in increasing order of slope

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

39 lines

```
template<class T = pll, class U = ll>
struct hull {
  struct frac {
    ll x, y;
    frac(ll _x, ll _y) : x(_x), y(_y) {
      if(y < 0) x = -x, y = -y;
    }
    bool operator <(const frac &other) const {
      return 1.0 * x * other.y < 1.0 * other.x * y;
    }
  };
  frac inter(T l1, T l2) { return { l2.se - l1.se, l1.fi - l2.
      fi }; }

  int nr = 0;
  vector<T> v;
  void add(T line) {
    // change signs for min
    if(!v.empty() && v.back().fi == line.fi) {
      if(v.back().se < line.se) v.back() = line;
      return;
    }
    while(nr >= 2 && inter(line, v[nr - 2]) < inter(v[nr - 1],
        v[nr - 2])) --nr, v.pop_back();
    v.push_back(line);
    ++nr;
  }
  U query(ll x) {
    int l, r, mid;
    for(l = 0, r = nr - 1; l < r; ) {
      mid = (l + r) / 2;
      if(inter(v[mid], v[mid + 1]) < frac(x, 1)) l = mid + 1;
      else r = mid;
    }
    // while(p + 1 < nr && eval(v[p + 1], x) < eval(v[p], x))
        ++p;
    return v[l];
  }
  ll eval(T line, ll x) {
    return line.fi * x + line.se;
  }
};
```

FenwickTree2d.h

Description: Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j]. Requires that the elements to be updated are known in advance (call FakeUpdate() before Init()).

Time: $\mathcal{O}(\log^2 N)$. (Use persistent segment trees for $\mathcal{O}(\log N)$.)

"FenwickTree.h" 32 lines

```
struct Fenwick2D {
  vector<vector<int>> ys;
  vector<vector<int>> T;
  Fenwick2D(int n) : ys(n + 1) {}

  void FakeUpdate(int x, int y) {
    for (++x; x < (int)ys.size(); x += (x & -x))
      ys[x].push_back(y);
  }
  void Init() {
    for (auto& v : ys) {
      sort(v.begin(), v.end());
      T.emplace_back(v.size());
    }
  }
  int ind(int x, int y) {
    auto it = lower_bound(ys[x].begin(), ys[x].end(), y);
    return distance(ys[x].begin(), it);
  }
  void Update(int x, int y, int val) {
    for (++x; x < (int)ys.size(); x += (x & -x))
    for (int i = ind(x,y); i < (int)T[x].size(); i += (i & -i))
      trees[x][i] = trees[x][i] + val;
  }
  int Query(int x, int y) {
    int sum = 0;
    for (; x > 0; x -= (x & -x))
    for (int i = ind(x,y); i > 0; i -= (i & -i))
      sum = sum + T[x][i];
    return sum;
  }
};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time. Set inf to something reasonable before use.

Usage: `RMQ rmq(values);`
`rmq.Query(inclusive, exclusive);`

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

21 lines

```
template <class T>
struct RMQ {
  const int kInf = numeric_limits<T>::max();
  vector<vector<T>> rmq;

  RMQ(const vector<T>& V) {
    int n = V.size(), on = 1, depth = 1;
    while (on < n) on *= 2, ++depth;
    rmq.assign(depth, V);
    for (int i = 0; i < depth - 1; ++i)
      for (int j = 0; j < n; ++j) {
        jmp[i + 1][j] = min(jmp[i][j],
          jmp[i][min(n - 1, j + (1 << i))]);
    }

  T Query(int a, int b) {
    if (b <= a) return kInf;
    int dep = 31 - __builtin_clz(b - a); // log(b - a)
    return min(rmq[dep][a], rmq[dep][b - (1 << dep)]);
  }
};
```

Numerical (4)

Polynomial.h

Description: Different operations on polynomials. Should work on any field.

<bits/stdc++.h> 114 lines

```
using TElem = double;
using Poly = vector<TElem>;

TElem Eval(const Poly& P, TElem x) {
  TElem val = 0;
  for (int i = (int)P.size() - 1; i >= 0; --i)
    val = val * x + P[i];
  return val;
}

// Differentiation
Poly Diff(Poly P) {
  for (int i = 1; i < (int)P.size(); ++i)
    P[i - 1] = i * P[i];
  P.pop_back();
  return P;
}

// Integration
Poly Integrate(Poly p) {
  P.push_back(0);
  for (int i = (int)P.size() - 2; i >= 0; --i)
    P[i + 1] = P[i] / (i + 1);
  P[0] = 0;
  return P;
}

// Division by (X - x0)
Poly DivRoot(Poly P, TElem x0) {
  int n = P.size();
  TElem a = P.back(), b; P.back() = 0;
  for (int i = n--; i--; )
    b = P[i], P[i] = P[i + 1] * x0 + a, a = b;
  P.pop_back();
  return P;
}

// Multiplication modulo X^sz
Poly Multiply(Poly A, Poly B, int sz) {
  static FFTSolver fft;
  A.resize(sz, 0); B.resize(sz, 0);
  auto R = fft.Multiply(A, B);
  R.resize(sz, 0);
  return r;
}

// Scalar multiplication
Poly Scale(Poly P, TElem s) {
  for (auto& x : P)
    x = x * s;
  return P;
}

// Addition modulo X^sz
Poly Add(Poly A, Poly B, int sz) {
  A.resize(sz, 0); B.resize(sz, 0);
  for (int i = 0; i < sz; ++i)
    A[i] = A[i] + B[i];
  return A;
}

// ***********************************************
```

```
// For Invert, Sqrt, size of argument should be 2^k
// ************************************************

Poly inv_step(Poly res, Poly P, int n) {
  auto res_sq = Multiply(res, res, n);
  auto sub = Multiply(res_sq, P, n);
  res = Add(Scale(res, 2), Scale(sub, -1), n);
  return res;
}

// Inverse modulo X^sz
// EXISTS ONLY WHEN P[0] IS INVERTIBLE
Poly Invert(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(-1)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    res = inv_step(res, P, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, P, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == (i == 0));
  }
  return res;
};

// Square root modulo X^sz
// EXISTS ONLY WHEN P[0] HAS SQUARE ROOT
Poly Sqrt(Poly P) {
  assert(P[0].Get() == 1);
  Poly res(1, 1);          // i.e., P[0]^(-1)
  Poly inv(1, 1);          // i.e., P[0]^(1/2)

  int n = P.size();
  for (int step = 2; step <= n; step *= 2) {
    auto now = inv_step(inv, res, step);
    now = Multiply(P, move(now), step);
    res = Add(res, now, step);
    res = Scale(res, (kMod + 1) / 2);
    inv = inv_step(inv, res, step);
  }

  // Optional, but highly encouraged
  auto check = Multiply(res, res, n);
  for (int i = 0; i < n; ++i) {
    assert(check[i].Get() == P[i].Get());
  }
  return res;
}
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `Poly p = {2, -3, 1} // x^2 - 3x + 2 = 0`
`auto roots = GetRoots(p, -1e18, 1e18); // {1, 2}`

`<bits/stdc++.h>, "Polynomial.h"` 26 lines

```
vector<double> GetRoots(Poly p, double xmin, double xmax) {
  if (p.size() == 2) { return {-p.front() / p.back()}; }
  else {
    Poly d = Diff(p);
    vector<double> dr = GetRoots(d, xmin, xmax);
    dr.push_back(xmin - 1);
    dr.push_back(xmax + 1);
    sort(dr.begin(), dr.end());

    vector<double> roots;
    for (auto i = dr.begin(), j = i++; i != dr.end(); j = i++){
      double lo = *j, hi = *i, mid, f;
      bool sign = Eval(p, lo) > 0;
      if (sign ^ (Eval(p, hi) > 0)) {
        // for (int it = 0; it < 60; ++it) {
        while (hi - lo > 1e-8) {
          mid = (lo + hi) / 2, f = Eval(p, mid);
          if ((f <= 0) ^ sign) lo = mid;
          else hi = mid;
        }
        roots.push_back((lo + hi) / 2);
      }
    }
    return roots;
  }
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

`<bits/stdc++.h>, "Polynomial.h"` 15 lines

```
Poly Interpolate(vector<TElem> x, vector<TElem> y) {
  int n = x.size();
  Poly res(n), temp(n);
  for (int k = 0; k < n; ++k)
    for (int i = k + 1; i < n; ++i)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  TElem last = 0; temp[0] = 1;
  for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i) {
    res[i] = res[i] + y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] = temp[i] - last * x[k];
  }
  return res;
}
```

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first 2*n terms of the recurrence. Very useful for guessing linear recurrences after brute-force / backtracking the first terms. Should work on any field. Numerical stability for floating-point calculations is not guaranteed.
Usage: `BerlekampMassey({0, 1, 1, 3, 5, 11}) => {1, 2}`

`<bits/stdc++.h>, "ModOps.h"` 29 lines

```
vector<ModInt> BerlekampMassey(vector<ModInt> s) {
  int n = s.size();
  vector<ModInt> C(n, 0), B(n, 0);
  C[0] = B[0] = 1;

  ModInt b = 1; int L = 0;
  for (int i = 0, m = 1; i < n; ++i) {

    ModInt d = s[i];
    for (int j = 1; j <= L; ++j)
      d = d + C[j] * s[i - j];

    if (d.get() == 0) { ++m; continue; }

    auto T = C; ModInt coef = d * inv(b);
    for (int j = m; j < n; ++j)
      C[j] = C[j] - coef * B[j - m];

    if (2 * L > i) { ++m; continue; }

    L = i + 1 - L; B = T; b = d; m = 1;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (auto& x : C) x = ModInt(0) - x;

  return C;
}
```

LinearRecurrence.h

Description: Generates the k-th term of a n-th order linear recurrence given the first n terms and the recurrence relation. Faster than matrix multiplication. Useful to use along with Berlekamp Massey.
Usage: `LinearRec<double>({0, 1}, {1, 1}).Get(k)` gives k-th Fibonacci number (0-indexed)
Time: $\mathcal{O}\left(n^2 log(k)\right)$ per query

`<bits/stdc++.h>` 43 lines

```
template<typename T>
struct LinearRec {
  using Poly = vector<T>;
  int n; Poly first, trans;

  // Recurrence is S[i] = sum(S[i-j-1] * trans[j])
  // with S[0..(n-1)] = first
  LinearRec(const Poly &first, const Poly &trans) :
    n(first.size()), first(first), trans(trans) {}

  Poly combine(Poly a, Poly b) {
    Poly res(n * 2 + 1, 0);
    // You can apply constant optimization here to get a
    // ~10x speedup
    for (int i = 0; i <= n; ++i)
      for (int j = 0; j <= n; ++j)
        res[i + j] = res[i + j] + a[i] * b[j];

    for (int i = 2 * n; i > n; --i)
      for (int j = 0; j < n; ++j)
        res[i - 1 - j] = res[i - 1 - j] + res[i] * trans[j];

    res.resize(n + 1);
    return res;
  }

  // Consider caching the powers for multiple queries
  T Get(int k) {
    Poly r(n + 1, 0), b(r);
    r[0] = 1; b[1] = 1;

    for (++k; k; k /= 2) {
      if (k % 2)
        r = combine(r, b);
      b = combine(b, b);
    }

    T res = 0;
    for (int i = 0; i < n; ++i)
      res = res + r[i + 1] * first[i];
    return res;
  }
};
```

Integrate.h

Description: Simple integration of a function over an interval using Simpson's rule. The error should be proportional to h^4, although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

9 lines

```
template<typename Func>
double Quad(Func f, double a, double b) {
```

```
  const int n = 1000;
  double h = (b - a) / 2 / n;
  double v = f(a) + f(b);
  for (int i = 1; i < 2 * n; ++i)
    v += f(a + i * h) * (i & 1 ? 4 : 2);
  return v * h / 3;
}
```

IntegrateAdaptive.h

Description: Fast integration using an adaptive Simpson's rule.
Usage:
```
double z, y;
double h(double x) { return x*x + y*y + z*z <= 1; }
double g(double y) { ::y = y; return Quad(h, -1, 1); }
double f(double z) { ::z = z; return Quad(g, -1, 1); }
double sphereVol = Quad(f, -1, 1), pi = sphereVol*3/4;
```
23 lines

```
template<typename Func>
double simpson(Func f, double a, double b) {
  double c = (a + b) / 2;
  return (f(a) + 4 * f(c) + f(b)) * (b - a) / 6;
}

template<typename Func>
double recurse(Func f, double a, double b,
               double eps, double S) {
  double c = (a + b) / 2;
  double S1 = simpson(f, a, c);
  double S2 = simpson(f, c, b);
  double T = S1 + S2;
  if (abs(T - S) < 15 * eps || b - a < 1e-10)
    return T + (T - S) / 15;
  return recurse(f, a, c, eps / 2, S1) +
         recurse(f, c, b, eps / 2, S2);
}

template<typename Func>
double Quad(Func f, double a, double b, double eps = 1e-8) {
  return recurse(f, a, b, eps, simpson(f, a, b));
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage:
```
A = {{1,-1}, {-1,1}, {-1,-2}};
b = {1,1,-4}, c = {-1,-1}, x;
T val = LPSolver(A, b, c).solve(x);
```
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.
68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define sz(x) (int)(x).size()

struct LPSolver {
  int m, n; vi N, B; vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
      rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      rep(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      rep(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T Solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      rep(i,0,m) if (B[i] == -1) {
        int s = 0;
        rep(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

SolveLinear.h

Description: Solves $M * x = b$. If there are multiple solutions, returns a solution which has all free variables set to 0. To compute rank, count the number of values in pivot. vector which are not -1. For inverse modulo prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}(vars^2 cons)$
79 lines

```
// Transforms a matrix into its row echelon form
// Returns a vector of pivots for each variable
// vars is the number of variables to do echelon for
vector<int> ToRowEchelon(vector<vector<double>> &M, int vars) {
  int n = M.size(), m = M[0].size();
  vector<int> pivots(vars, -1);

  int cur = 0;
  for (int var = 0; var < vars; ++var) {
    if (cur >= n) break;

    for (int con = cur + 1; con < n; ++con)
      if (sgn(M[con][var]) != 0)
        swap(M[con], M[cur]);

    if (sgn(M[cur][var]) != 0) {
      pivots[var] = cur;
      auto aux = M[cur][var];

      for (int i = 0; i < m; ++i)
        M[cur][i] = M[cur][i] / aux;

      for (int con = 0; con < n; ++con) {
        if (con != cur) {
          auto mul = M[con][var];
          for (int i = 0; i < m; ++i) {
            M[con][i] = M[con][i] - mul * M[cur][i];
          }
        }
      }
      ++cur;
    }
  }

  return pivots;
}

// Computes the inverse of a nxn square matrix.
// Returns true if successful
bool Invert(vector<vector<double>> &M) {
  int n = M.size();
  for (int i = 0; i < n; ++i) {
    M[i].resize(2 * n, 0); M[i][n + i] = 1;
  }

  auto pivs = ToRowEchelon(M, n);
  for (auto x : pivs) if (x == -1) return false;

  for (int i = 0; i < n; ++i)
    M[i].erase(M[i].begin(), M[i].begin() + n);
  return true;
}

// Returns the solution of a system
// Will change matrix
// Throws 5 if inconsistent
vector<double> SolveSystem(vector<vector<double>> &M,
                           vector<double>& b) {
  int vars = M[0].size();
  for (int i = 0; i < (int)M.size(); ++i)
    M[i].push_back(b[i]);

  auto pivs = ToRowEchelon(M, vars);
  vector<double> solution(vars);
  for (int i = 0; i < vars; ++i) {
    solution[i] = (pivs[i] == -1) ? 0 : M[pivs[i]][vars];
  }

  // Check feasible (optional)
  for (int i = 0; i < (int)M.size(); ++i) {
    double check = 0;
    for (int j = 0; j < vars; ++j)
      check = check + M[i][j] * solution[j];
    if (sgn(check - M[i][vars]) != 0)
      throw 5;
  }
```

```
  return solution;
}
```

Tridiagonal.h

Description: Solves a linear equation system with a tridiagonal matrix with diagonal diag, subdiagonal sub and superdiagonal super, i.e., $x = \text{Tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

The size of diag and b should be the same and super and sub should be one element shorter. T is intended to be double.
This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \leq i \leq n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{Tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Usage: `int n = 1000000;`
`vector<double> diag(n,-1), sup(n-1,.5), sub(n-1,.5), b(n,1);`
`vector<double> x = tridiagonal(diag, super, sub, b);`
Time: $\mathcal{O}(N)$

14 lines

```
template <typename T>
vector<T> Tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  for (int i = 0; i < b.size() - 1; ++i) {
    diag[i + 1] -= super[i] * sub[i] / diag[i];
    b[i + 1] -= b[i] * sub[i] / diag[i];
  }
  for (int i = b.size(); --i > 0;) {
    b[i] /= diag[i];
    b[i - 1] -= b[i] * super[i - 1];
  }
  b[0] /= diag[0];
  return b;
}
```

Number theory (5)

5.1 General

mathValeriu.h

Description: None
Usage: ask Djok

118 lines

```
bool isPrime(int x) {
  if(x < 2) return 0;
  if(x == 2) return 1;
  if(x % 2 == 0) return 0;
  for(int i = 3; i * i <= x; i += 2)
    if(x % i == 0) return 0;
  return 1;
}

int mul(int a, int b) {
  return (long long)a * b % MOD;
}

int add(int a, int b) {
  a += b;
  if(a >= MOD) return a - MOD;
  return a;
}

int getPw(int a, int b) {
  int ans = 1;
  for(; b > 0; b /= 2) {
    if(b & 1) ans = mul(ans, a);
    a = mul(a, a);
  }
  return ans;
}

long long modInv(long long a, long long m) {
  if(a == 1) return 1;
  return (1 - modInv(m % a, a) * m) / a + m;
}

long long CRT(vector<long long> &r, vector<long long> &p) {
  long long ans = r[0] % p[0], prod = p[0];
  for(int i = 1; i < r.size(); ++i) {
    long long coef = ((r[i] - (ans % p[i]) + p[i]) % p[i]) *
         modInv(prod % p[i], p[i]) % p[i];
    ans += coef * prod;
    prod *= p[i];
  }
  return ans;
}

long long getPhi(long long n) {
  long long ans = n - 1;
  for(int i = 2; i * i <= n; ++i) {
    if(n % i) continue;
    while(n % i == 0) n /= i;
    ans -= ans / i;
  }
  if(n > 1) ans -= ans / n;
  return ans;
}

// fact is a vector with prime divisors of N - 1 (N here is
     modulo) and N is prime
// the idea is that if N is prime, then N-1 is phi(N), which
     means the cycle has length N - 1
// now, lets try to see if X is a generator
// we know that if x ^ phi(N) == 1 then x ^ 2*phi(N) is also ==
     1, and here we get the idea
// if for some divisor of phi(N), x ^ div == 1, then obviously
     X is not a generator
// because the cycle is not of length N
// good luck to understand this after one year :)
bool isGenerator(int x, int n) {
  if(cmmdc(x, n) != 1) return 0;

  for(auto it : fact)
    if(Pow(x, (n - 1) / it, n) == 1)
      return 0;
  return 1;
}

// Lucas Theorem
// calc COMB(N, R) if N and R is VERY VERY BIG and MOD is PRIME
r -= 2; n += m - 2;
while(r > 0 || n > 0) {
  ans = (1LL * ans * comb(n % MOD, r % MOD)) % MOD;
  n /= MOD; r /= MOD;
}

// GAUSS FOR F2 space
// SZ is the size of basis
void gauss(int mask) {
  for(int i = 0; i < n; ++i) {
    if(!(mask & (1 << i))) continue;
    if(!basis[i]) {
      basis[i] = mask;
      ++sz;
      break;
    }
    mask ^= basis[i];
  }
}

// if A is a permutation of B, then A == B mod 9

bool isSquare(int x) {
  int a = sqrt(x) + 0.5;
  return a * a == x;
}

int getDiscreteLog(int a, int b, int m) {
  if(b == 1) return 0;
  int n = sqrt(m) + 1;
  int an = 1;
  for(int i = 0; i < n; ++i) an = (an * a) % m;
  unordered_map<int, int> vals;
  for(int i = 1, cur = an; i <= n; ++i) {
    if(!vals.count(cur)) vals[cur] = i;
    cur = (cur * an) % m;
  }
  for(int i = 0, cur = b; i <= n; ++i) {
    if(vals.count(cur)) {
      int ans = vals[cur] * n - i;
      return ans;
    }
    cur = (cur * a) % m;
  }
  return -1;
}
```

nrPentagonaleValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 20 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int MOD = 999123;
const int N = 500005;

int i, j, p[N];

int main() {
  p[0] = 1;
  for(i = 1; i < N; ++i) {
    for(j = 1; j * (3 * j - 1) / 2 <= i; ++j)
      if(j & 1) p[i] = (p[i] + p[i - j * (3 * j - 1) / 2]) %
           MOD;
      else p[i] = (p[i] - p[i - j * (3 * j - 1) / 2] + MOD) %
           MOD;
    for(j = 1; j * (3 * j + 1) / 2 <= i; ++j)
      if(j & 1) p[i] = (p[i] + p[i - j * (3 * j + 1) / 2]) %
           MOD;
```

```
      else p[i] = (p[i] - p[i - j * (3 * j + 1) / 2] + MOD) %
          MOD;
  }
  return 0;
}
```

5.2 Modular arithmetic

ModInverse.h

Description: Pre-computation of modular inverses. Assumes lim < kMod and that kMod is a prime.

"ModOps.h" 7 lines

```
vector<ModInt> ComputeInverses(int lim) {
  vector<ModInt> inv(lim + 1); inv[1] = 1;
  for (int i = 2; i <= lim; ++i) {
    inv[i] = ModInt(0) - ModInt(kMod / i) * inv[kMod % i];
  }
  return inv;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.

Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, often $\mathcal{O}\left(\log p\right)$

"ModPow.h" 30 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1;
  int r = 0;
  while (s % 2 == 0)
    ++r, s /= 2;
  ll n = 2; // find a non-square mod p
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p);
  ll g = modpow(n, s, p);
  for (;;) {
    ll t = b;
    int m = 0;
    for (; m < r; ++m) {
      if (t == 1) break;
      t = t * t % p;
    }
    if (m == 0) return x;
    ll gs = modpow(g, 1 << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
    r = m;
  }
}
```

5.3 Number theoretic transform

NTT.h

Description: Number theoretic transform. Can be used for convolutions modulo specific nice primes of the form $2^a b+1$, where the convolution result has size at most 2^a. For other primes/integers, use two different primes and combine with CRT. If NTT is not fast enough and you are multiplying a lot, consider doing naive solution for the small ones.

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

"ModPow.h" 65 lines

```
const int kMod = (119 << 23) + 1, kRoot = 3; // = 998244353
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

struct FFTSolver {
  vector<int> rev;

  int __lg(int n) { return n == 1 ? 0 : 1 + __lg(n / 2);  }

  void compute_rev(int n, int lg) {
    rev.resize(n); rev[0] = 0;
    for (int i = 1; i < n; ++i) {
      rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (lg - 1));
    }
  }

  vector<ModInt> fft(vector<ModInt> V, bool invert) {
    int n = V.size(), lg = __lg(n);
    if ((int)rev.size() != n) compute_rev(n, lg);

    for (int i = 0; i < n; ++i) {
      if (i < rev[i])
        swap(V[i], V[rev[i]]);
    }

    for (int step = 2; step <= n; step *= 2) {
      ModInt eps = lgpow(kRoot, (kMod - 1) / step);
      if (invert) eps = inv(eps);

      for (int i = 0; i < n; i += step) {
        ModInt w = 1;
        for (int a = i, b = i+step/2; b < i+step; ++a, ++b) {
          ModInt aux = w * V[b];
          V[b] = V[a] - aux;
          V[a] = V[a] + aux;
          w = w * eps;
        }
      }
    }

    return V;
  }

  vector<ModInt> Multiply(vector<ModInt> A, vector<ModInt> B) {
    int n = A.size() + B.size() - 1, sz = n;
    while (n != (n & -n)) ++n;

    A.resize(n, 0); B.resize(n, 0);

    A = fft(move(A), false);
    B = fft(move(B), false);

    vector<ModInt> ret(n);
    ModInt inv_n = inv(n);

    for (int i = 0; i < n; ++i) {
      ret[i] = A[i] * B[i] * inv_n;
    }

    ret = fft(move(ret), true);
    ret.resize(sz);

    return ret;
  }
};
```

5.4 Fast Fourier Transform

fftValeriu.h

Description: None

Usage: ask Djok

221 lines

```
const double PI = acos(-1);
typedef complex<double> ftype;

int rev(int x, int lg) {
  int ans = 0;
  for(int i = 0; i < lg; ++i)
    if(x & (1 << i)) ans += (1 << (lg - i - 1));
  return ans;
}

void fft(vector<ftype> &a, bool inv) {
  int lg = 1, sz = a.size();
  while((1 << lg) < sz) ++lg;

  for(int i = 0; i < sz; ++i) if(i < rev(i, lg)) swap(a[i], a[
      rev(i, lg)]);

  for(int len = 2; len <= sz; len <<= 1) {
    double ang = (inv ? -2 : 2) * PI / len;
    ftype wlen(cos(ang), sin(ang));

    for(int i = 0; i < sz; i += len) {
      ftype w(1, 0);
      for(int j = 0; j < len / 2; ++j) {
        ftype u = a[i + j], v = w * a[i + len / 2 + j];
        a[i + j] = u + v;
        a[i + len / 2 + j] = u - v;
        w *= wlen;
      }
    }
  }

  if(inv)
    for(int i = 0; i < sz; ++i) a[i] /= sz;
}

void conv(vector<int> &a, vector<int> &b, vector<int> &c) {
  vector<ftype> na(a.begin(), a.end());
  vector<ftype> nb(b.begin(), b.end());

  int sz = 2 * max(a.size(), b.size()), lg = 1;
  while((1 << lg) < sz) ++lg;
  sz = (1 << lg);

  na.resize(sz); nb.resize(sz);
  fft(na, 0); fft(nb, 0);

  for(int i = 0; i < sz; ++i) na[i] *= nb[i];

  fft(na, 1);

  c.resize(a.size() + b.size() - 1);
  for(int i = 0; i < c.size(); ++i) c[i] = na[i].real() + 0.5;
}

// Tourist FFT
namespace fft {
  typedef double dbl;

  struct num {
    dbl x, y;
    num() { x = y = 0; }
    num(dbl x, dbl y) : x(x), y(y) { }
```

```
};

inline num operator+(num a, num b) { return num(a.x + b.x, a.y + b.y); }
inline num operator-(num a, num b) { return num(a.x - b.x, a.y - b.y); }
inline num operator*(num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
inline num conj(num a) { return num(a.x, -a.y); }

int base = 1;
vector<num> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};

const dbl PI = acosl(-1.0);

void ensure_base(int nbase) {
  if (nbase <= base) {
    return;
  }
  rev.resize(1 << nbase);
  for (int i = 0; i < (1 << nbase); i++) {
    rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (nbase - 1));
  }
  roots.resize(1 << nbase);
  while (base < nbase) {
    dbl angle = 2 * PI / (1 << (base + 1));
//      num z(cos(angle), sin(angle));
    for (int i = 1 << (base - 1); i < (1 << base); i++) {
      roots[i << 1] = roots[i];
//        roots[(i << 1) + 1] = roots[i] * z;
      dbl angle_i = angle * (2 * i + 1 - (1 << base));
      roots[(i << 1) + 1] = num(cos(angle_i), sin(angle_i));
    }
    base++;
  }
}

void fft(vector<num> &a, int n = -1) {
  if (n == -1) {
    n = a.size();
  }
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = base - zeros;
  for (int i = 0; i < n; i++) {
    if (i < (rev[i] >> shift)) {
      swap(a[i], a[rev[i] >> shift]);
    }
  }
  for (int k = 1; k < n; k <<= 1) {
    for (int i = 0; i < n; i += 2 * k) {
      for (int j = 0; j < k; j++) {
        num z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}

vector<num> fa, fb;

vector<int> multiply(vector<int> &a, vector<int> &b) {
  int need = a.size() + b.size() - 1;
  int nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  if (sz > (int) fa.size()) {
    fa.resize(sz);
  }
  for (int i = 0; i < sz; i++) {
    int x = (i < (int) a.size() ? a[i] : 0);
    int y = (i < (int) b.size() ? b[i] : 0);
    fa[i] = num(x, y);
  }
  fft(fa, sz);
  num r(0, -0.25 / sz);
  for (int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    num z = (fa[j] * fa[j] - conj(fa[i] * fa[i])) * r;
    if (i != j) {
      fa[j] = (fa[i] * fa[i] - conj(fa[j] * fa[j])) * r;
    }
    fa[i] = z;
  }
  fft(fa, sz);
  vector<int> res(need);
  for (int i = 0; i < need; i++) {
    res[i] = fa[i].x + 0.5;
  }
  return res;
}

vector<int> multiply_mod(vector<int> &a, vector<int> &b, int m, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int nbase = 0;
  while ((1 << nbase) < need) nbase++;
  ensure_base(nbase);
  int sz = 1 << nbase;
  if (sz > (int) fa.size()) {
    fa.resize(sz);
  }
  for (int i = 0; i < (int) a.size(); i++) {
    int x = (a[i] % m + m) % m;
    fa[i] = num(x & ((1 << 15) - 1), x >> 15);
  }
  fill(fa.begin() + a.size(), fa.begin() + sz, num {0, 0});
  fft(fa, sz);
  if (sz > (int) fb.size()) {
    fb.resize(sz);
  }
  if (eq) {
    copy(fa.begin(), fa.begin() + sz, fb.begin());
  } else {
    for (int i = 0; i < (int) b.size(); i++) {
      int x = (b[i] % m + m) % m;
      fb[i] = num(x & ((1 << 15) - 1), x >> 15);
    }
    fill(fb.begin() + b.size(), fb.begin() + sz, num {0, 0});
    fft(fb, sz);
  }
  dbl ratio = 0.25 / sz;
  num r2(0, -1);
  num r3(ratio, 0);
  num r4(0, -ratio);
  num r5(0, 1);
  for (int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    num a1 = (fa[i] + conj(fa[j]));
    num a2 = (fa[i] - conj(fa[j])) * r2;
    num b1 = (fb[i] + conj(fb[j])) * r3;
    num b2 = (fb[i] - conj(fb[j])) * r4;
    if (i != j) {
      num c1 = (fa[j] + conj(fa[i]));
      num c2 = (fa[j] - conj(fa[i])) * r2;
      num d1 = (fb[j] + conj(fb[i])) * r3;
      num d2 = (fb[j] - conj(fb[i])) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz);
  fft(fb, sz);
  vector<int> res(need);
  for (int i = 0; i < need; i++) {
    long long aa = fa[i].x + 0.5;
    long long bb = fb[i].x + 0.5;
    long long cc = fa[i].y + 0.5;
    res[i] = (aa + ((bb % m) << 15) + ((cc % m) << 30)) % m;
  }
  return res;
}

vector<int> square_mod(vector<int> &a, int m) {
  return multiply_mod(a, a, m, 1);
}
};
```

5.5 Primality

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.

Time: 15 times the complexity of $a^b \mod c$.

"ModMulLL.h" 18 lines

```
using ull = unsigned long long;

bool IsPrime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  for (int i = 0; i < 15; ++i) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = ModPow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = ModMul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

5.6 Divisibility

5.6.1 Bézout's identity

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

$$ax + by = d$$

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

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

phiFunction.h
Description: *Euler's totient* or *Euler's phi* function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. The *cototient* is $n - \phi(n)$. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1-1} ... (p_r - 1)p_r^{k_r-1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.
$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$
Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.
Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}\ \forall a$.
11 lines

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

void ComputePhi() {
  for (int i = 0; i < kLim; ++i)
    phi[i] = (i % 2) ? i : i / 2;
  for (int i = 3; i < kLim; i += 2)
    if (phi[i] == i)
      for (int j = i; j < kLim; j += i)
        (phi[j] /= i) *= i - 1;
}
```

5.7 Chinese remainder theorem

CRT.h
Description: Chinese Remainder Theorem.
Find z such that $z\%m_1 = r_1, z\%m_2 = r_2$. Here, z is unique modulo M = lcm(m1, m2). The vector version solves a system of equations of type $z\%m_i = p_i$. On output, return { 0, -1 } . Note that all numbers must be less than 2^{31} if you have type unsigned long long.
Time: $\log(m+n)$
"Euclid.h" 17 lines

```
pair<int, int> CRT(int m1, int r1, int m2, int r2) {
  int s, t;
  int g = Euclid(m1, m2, s, t);
  if (r1 % g != r2 % g) return make_pair(0, -1);
  int z = (s * r2 * m1 + t * r1 * m2) % (m1 * m2);
  if (z < 0) z += m1 * m2;
  return make_pair(m1 * m2 / g, z / g);
}

pair<int, int> CRT(vector<int> m, vector<int> r) {
  pair<int, int> ret = make_pair(m[0], r[0]);
  for (int i = 1; i < m.size(); i++) {
    ret = CRT(ret.first, ret.second, m[i], r[i]);
    if (ret.second == -1) break;
  }
  return ret;
}
```

5.8 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.

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

5.10 Estimates

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

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

Combinatorial (6)

6.1 The Twelvefold Way

Counts the # of functions $f : N \to K$, $|N| = n$, $|K| = k$. The elements in N and K can be distinguishable or indistinguishable, while f can be injective (one-to-one) of surjective (onto).

N	K	none	injective	surjective
dist	dist	k^n	$\frac{k!}{(k-n)!}$	$k!S(n,k)$
indist	dist	$\binom{n+k-1}{n}$	$\binom{k}{n}$	$\binom{n-1}{n-k}$
dist	indist	$\sum_{t=0}^{k} S(n,t)$	$[n \leq k]$	$S(n,k)$
indist	indist	$\sum_{t=1}^{k} p(n,t)$	$[n \leq k]$	$p(n,k)$

Here, $S(n,k)$ is the Stirling number of the second kind, and $p(n,k)$ is the partition number.

6.2 Permutations

6.2.1 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

6.2.2 Cycles

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

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

6.2.3 Derangements

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

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

derangements.h
Description: Generates the i:th derangement of S_n (in lexicographical order).
38 lines

```
template <class T, int N>
struct derangements {
  T dgen[N][N], choose[N][N], fac[N];
  derangements() {
    fac[0] = choose[0][0] = 1;
    memset(dgen, 0, sizeof(dgen));
    rep(m,1,N) {
      fac[m] = fac[m-1] * m;
      choose[m][0] = choose[m][m] = 1;
      rep(k,1,m)
        choose[m][k] = choose[m-1][k-1] + choose[m-1][k];
    }
  }
  T DGen(int n, int k) {
    T ans = 0;
    if (dgen[n][k]) return dgen[n][k];
    rep(i,0,k+1)
      ans += (i&1?-1:1) * choose[k][i] * fac[n-i];
    return dgen[n][k] = ans;
  }
  void generate(int n, T idx, int *res) {
    int vals[N];
    rep(i,0,n) vals[i] = i;
    rep(i,0,n) {
      int j, k = 0, m = n - i;
      rep(j,0,m) if (vals[j] > i) ++k;
      rep(j,0,m) {
        T p = 0;
        if (vals[j] > i) p = DGen(m-1, k-1);
        else if (vals[j] < i) p = DGen(m-1, k);
        if (idx <= p) break;
        idx -= p;
      }
      res[i] = vals[j];
      memmove(vals + j, vals + j + 1, sizeof(int)*(m-j-1));
    }
  }
};
```

6.2.4 Involutions

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152

6.2.5 Stirling numbers of the first kind

$$s(n,k) = (-1)^{n-k}c(n,k)$$

$c(n,k)$ is the unsigned Stirling numbers of the first kind, and they count the number of permutations on n items with k cycles.

$$s(n,k) = s(n-1,k-1) - (n-1)s(n-1,k)$$

$$s(0,0) = 1, s(n,0) = s(0,n) = 0$$

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k)$$

$$c(0,0) = 1, c(n,0) = c(0,n) = 0$$

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

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

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

6.2.7 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

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

6.3 Partitions and subsets

6.3.1 Partition function

Partitions of n with exactly k parts, $p(n,k)$, i.e., writing n as a sum of k positive integers, disregarding the order of the summands.

$$p(n,k) = p(n-1,k-1) + p(n-k,k)$$

$$p(0,0) = p(1,n) = p(n,n) = p(n,n-1) = 1$$

For partitions with any number of parts, $p(n)$ obeys

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

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

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

6.3.2 Binomials

binomial.h

Description: The number of k-element subsets of an n-element set, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Time: $\mathcal{O}(\min(k, n-k))$ 6 lines

```
ll choose(int n, int k) {
  ll c = 1, to = min(k, n-k);
  if (to < 0) return 0;
  rep(i,0,to) c = c * (n - i) / (i + 1);
  return c;
}
```

binomialModPrime.h

Description: Lucas' thm: 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 $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}(\log_p n)$ 10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

RollingBinomial.h

Description: $\binom{n}{k}$ (mod m) in time proportional to the difference between (n, k) and the previous (n, k). 14 lines

```
const ll mod = 1000000007;
vector<ll> invs; // precomputed up to max n, inclusively
struct Bin {
  int N = 0, K = 0;  ll r = 1;
  void m(ll a, ll b) { r = r * a % mod * invs[b] % mod; }
  ll choose(int n, int k) {
    if (k > n || k < 0) return 0;
    while (N < n) ++N, m(N, N-K);
    while (K < k) ++K, m(N-K+1, K);
    while (K > k) m(K, N-K+1), --K;
    while (N > n) m(N-K, N), --N;
    return r;
  }
};
```

multinomial.h

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

Time: $\mathcal{O}((\sum k_i) - k_1)$ 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;
}
```

6.3.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

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

6.3.4 Bell numbers

Total number of partitions of n distinct elements.

$$B(n) = \sum_{k=1}^{n}\binom{n-1}{k-1}B(n-k) = \sum_{k=1}^{n} S(n,k)$$

$$B(0) = B(1) = 1$$

The first are 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597. For a prime p

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

6.3.5 Triangles

Given rods of length $1, \ldots, n$,

$$T(n) = \frac{1}{24}\begin{cases} n(n-2)(2n-5) & n \text{ even} \\ (n-1)(n-3)(2n-1) & n \text{ odd} \end{cases}$$

is the number of distinct triangles (positive are) that can be constructed, i.e., the # of 3-subsets of $[n]$ s.t. $x \le y \le z$ and $z \ne x + y$.

6.4 General purpose numbers

6.4.1 Catalan numbers

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

$$C_{n+1} = \frac{2(2n+1)}{n+2}C_n$$

$$C_0 = 1, C_{n+1} = \sum C_i C_{n-i}$$

First few are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900.

- # of monotonic lattice paths of a $n \times n$-grid which do not pass above the diagonal.
- # of expressions containing n pairs of parenthesis which are correctly matched.
- # of full binary trees with with $n+1$ leaves (0 or 2 children).
- # of non-isomorphic ordered trees with $n+1$ vertices.
- # of ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- # of permutations of $[n]$ with no three-term increasing subsequence.

6.4.2 Super Catalan numbers

The number of monotonic lattice paths of a $n \times n$-grid that do not touch the diagonal.

$$S(n) = \frac{3(2n-3)S(n-1) - (n-3)S(n-2)}{n}$$

$$S(1) = S(2) = 1$$

1, 1, 3, 11, 45, 197, 903, 4279, 20793, 103049, 518859

6.4.3 Motzkin numbers

Number of ways of drawing any number of nonintersecting chords among n points on a circle. Number of lattice paths from $(0,0)$ to $(n,0)$ never going below the x-axis, using only steps NE, E, SE.

$$M(n) = \frac{3(n-1)M(n-2) + (2n+1)M(n-1)}{n+2}$$

$$M(0) = M(1) = 1$$

1, 1, 2, 4, 9, 21, 51, 127, 323, 835, 2188, 5798, 15511, 41835, 113634

6.4.4 Narayana numbers

Number of lattice paths from $(0,0)$ to $(2n,0)$ never going below the x-axis, using only steps NE and SE, and with k peaks.

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$$

$$N(n,1) = N(n,n) = 1$$

$$\sum_{k=1}^{n} N(n,k) = C_n$$

1, 1, 1, 1, 3, 1, 1, 6, 6, 1, 1, 10, 20, 10, 1, 1, 15, 50

6.4.5 Schröder numbers

Number of lattice paths from $(0,0)$ to (n,n) using only steps N,NE,E, never going above the diagonal. Number of lattice paths from $(0,0)$ to $(2n,0)$ using only steps NE, SE and double east EE, never going below the x-axis. Twice the Super Catalan number, except for the first term. 1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098

Graph (7)

7.1 Network flow

DinicValeriu.h

Description: None

Usage: ask Djok

<bits/stdc++.h> 68 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 205;
const int INF = 0x3f3f3f3f;
const int Source = 0;
const int Sink = N - 2;

struct edge {
  int to, flow;
};

int i, ptr[N], dist[N], q[N];
vector<int> lda[N];
vector<edge> E;

void clearNetwork() {
  E.clear();
  for(int i = 0; i < N; ++i) lda[i].clear();
}

void addEdge(int x, int y, int flow) {
  E.push_back({y, flow});
  E.push_back({x, 0});
  lda[x].push_back(E.size() - 2);
  lda[y].push_back(E.size() - 1);
}

bool bfs() {
  int st = 0, dr = 0;
  memset(dist, INF, sizeof(dist));
  for(q[st] = Source, dist[Source] = 0; st <= dr; ++st)
    for(int it : lda[q[st]])
      if(E[it].flow && dist[E[it].to] > dist[q[st]] + 1) {
        dist[E[it].to] = dist[q[st]] + 1;
        q[++dr] = E[it].to;
      }
  return dist[Sink] != INF;
}

int dfs(int x, int flow) {
  if(!flow || x == Sink) return flow;
  for(; ptr[x] < lda[x].size(); ++ptr[x]) {
    int it = lda[x][ptr[x]];
    if(E[it].flow <= 0 || dist[E[it].to] != dist[x] + 1)
        continue;
    int pushed = dfs(E[it].to, min(flow, E[it].flow));
    if(pushed) {
      E[it].flow -= pushed;
      E[it ^ 1].flow += pushed;
      return pushed;
    }
  }
  return 0;
}

int dinic() {
  int flow = 0;
  while(bfs()) {
    memset(ptr, 0, sizeof(ptr));
    while(int pushed = dfs(Source, INF)) flow += pushed;
  }
  return flow;
}

int main() {

  return 0;
}
```

EZFlow.h

Description: A slow, albeit very easy-to-implement flow algorithm.

Time: $\mathcal{O}(EF)$ where E is the number of edges and F is the maximum flow. 29 lines

```
struct EZFlow {
  vector<vector<int>> G;
  vector<bool> vis;
  int t;

  EZFlow(int n) : G(n), vis(n) {}

  bool dfs(int node) {
    if (node == t) return true;
    vis[node] = true;

    for (auto& vec : G[node]) {
      if (!vis[vec] && dfs(vec)) {
        G[vec].push_back(node);
        swap(vec, G[node].back());
        G[node].pop_back();
        return true;
      }
    }
    return false;
  }

  void AddEdge(int a, int b) { G[a].push_back(b); }
  int ComputeFlow(int s, int t) {
    this->t = t; int ans = 0;
    while (dfs(s)) {++ans; fill(vis.begin(),vis.end(), false);}
    return ans;
  }
};
```

edmonsblossomValeriu.h

Description: None

Usage: ask Djok

<bits/stdc++.h> 94 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 105;

int i, match[N], p[N], base[N], q[N];
bool used[N], viz[N], blossom[N];
vector<int> lda[N];

int lca(int a, int b) {
  memset(viz, 0, sizeof(viz));
  while(1) {
    a = base[a];
    viz[a] = 1;
    if(match[a] == -1) break;
    a = p[match[a]];
  }
  while(1) {
    b = base[b];
    if(viz[b]) break;
    b = p[match[b]];
  }
  return b;
}

void markPath(int x, int y, int children) {
  while(base[x] != y) {
    blossom[base[x]] = blossom[base[match[x]]] = 1;
    p[x] = children;
    children = match[x];
    x = p[match[x]];
  }
}

int findPath(int x) {
  memset(used, 0, sizeof(used));
  memset(p, -1, sizeof(p));
  for(int i = 0; i < N; ++i) base[i] = i;

  int qh = 0, qt = 0;
  q[qt++] = x; used[x] = 1;
  while(qh < qt) {
    int v = q[qh++];
    for(int to : lda[v]) {
      if(base[v] == base[to] || match[v] == to) continue;
      if(to == x || match[to] != -1 && p[match[to]] != -1) {
        int curbase = lca(v, to);
        memset(blossom, 0, sizeof(blossom));
        markPath(v, curbase, to);
        markPath(to, curbase, v);
        for(int i = 0; i < N; ++i)
          if(blossom[base[i]]) {
            base[i] = curbase;
            if(!used[i]) {
              used[i] = 1;
              q[qt++] = i;
            }
          }
      }
      else if(p[to] == -1) {
        p[to] = v;
        if(match[to] == -1) return to;
        to = match[to];
        used[to] = 1;
        q[qt++] = to;
      }
```

```
    }
  }
  return -1;
}

int main() {
  // add edge x, y and y, x to lda
  memset(match, -1, sizeof(match));
  for(i = 0; i < N; ++i)
    if(match[i] == -1)
      for(int to : lda[i])
        if(match[to] == -1) {
          match[to] = i;
          match[i] = to;
          break;
        }

  for(i = 0; i < N; ++i)
    if(match[i] == -1) {
      int v = findPath(i);
      while(v != -1) {
        int pv = p[v], ppv = match[pv];
        match[v] = pv; match[pv] = v; v = ppv;
      }
    }

  return 0;
}
```

minCostMaxFlowValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 81 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
    ,tune=native")

const int N = 205;
const int INF = 0x3f3f3f3f;
const int Source = 0;
const int Sink = N - 3;

struct edge {
  int to, flow, cost;
};

int i, dist[N], q[N], inQ[N], poz[N], from[N];
vector<int> lda[N];
vector<edge> E;

void clearNetwork() {
  E.clear();
  for(int i = 0; i < N; ++i) lda[i].clear();
}

void addEdge(int x, int y, int flow, int cost) {
  E.push_back({y, flow, cost});
  E.push_back({x, 0, -cost});
  lda[x].push_back(E.size() - 2);
  lda[y].push_back(E.size() - 1);
}

void updateFlow(int &currFlow, int &currCost, int addFlow) {
  for(int x = Sink; x != Source; x = from[x]) addFlow = min(
      addFlow, E[poz[x]].flow);
  for(int x = Sink; x != Source; x = from[x]) {
    currCost += addFlow * E[poz[x]].cost;
    E[poz[x]].flow -= addFlow;
    E[poz[x] ^ 1].flow += addFlow;
  }
  currFlow += addFlow;
}

pair<int, int> getMinCostMaxFlow(int limitFlow = INF) {
  int currFlow = 0, currCost = 0;

  while(currFlow < limitFlow) {
    memset(dist, INF, sizeof(dist));
    memset(inQ, 0, sizeof(inQ));

    int st = 0, dr = 1;
    q[st] = Source; inQ[Source] = 1; dist[Source] = 0;
    while(st != dr) {
      int x = q[st];
      st = (st + 1) % N;
      inQ[x] = 2;
      for(int it : lda[x]) {
        if(E[it].flow <= 0 || dist[E[it].to] <= dist[x] + E[it
            ].cost) continue;

        dist[E[it].to] = dist[x] + E[it].cost;
        from[E[it].to] = x;

        if(inQ[E[it].to] == 0) {
          q[dr] = E[it].to;
          if(dr == N - 1) dr = 0; else ++dr;
        } else if(inQ[E[it].to] == 2) {
          if(st == 0) st = N - 1; else --st;
          q[st] = E[it].to;
        }

        inQ[E[it].to] = 1; poz[E[it].to] = it;
      }
    }

    if(dist[Sink] == INF) break;

    updateFlow(currFlow, currCost, limitFlow - currFlow);
  }

  return make_pair(currFlow, currCost);
}

int main() {

  return 0;
}
```

StoerWagnerValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 84 lines

```
#define sz(x) ((int) (x).size())
#define forn(i,n) for (int i = 0; i < int(n); ++i)
#define forab(i,a,b) for (int i = int(a); i < int(b); ++i)

typedef long long ll;
typedef long double ld;

const int INF = 1000001000;
const ll INFL = 2000000000000001000;

const int maxn = 500;

ll g[maxn][maxn];
ll dist[maxn];
bool used[maxn];

void addEdge(int u, int v, ll c)
{
    g[u][v] += c;
    g[v][u] += c;
}

int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    ll total = 0;
    forn (i, m)
    {
        int k, f;
        scanf("%d%d", &k, &f);
        total += 2 * f;
        vector<int> group;
        forn (j, k)
        {
            int u;
            scanf("%d", &u);
            --u;
            group.push_back(u);
        }
        if (k == 2)
            addEdge(group[0], group[1], 2 * f);
        else
        {
            addEdge(group[0], group[1], f);
            addEdge(group[1], group[2], f);
            addEdge(group[2], group[0], f);
        }
    }
    vector<int> vertices;
    forn (i, n)
        vertices.push_back(i);
    ll mincut = total + 1;
    while (sz(vertices) > 1)
    {
        int u = vertices[0];
        for (auto v: vertices)
            used[v] = false,
            dist[v] = g[u][v];
        used[u] = true;
        forn (ii, sz(vertices) - 2)
        {
            for (auto v: vertices)
                if (!used[v])
                    if (used[u] || dist[v] > dist[u])
                        u = v;
            used[u] = true;
            for (auto v: vertices)
                if (!used[v])
                    dist[v] += g[u][v];
        }
        int t = -1;
        for (auto v: vertices)
            if (!used[v])
                t = v;
        mincut = min(mincut, dist[t]);
        vertices.erase(find(vertices.begin(), vertices.end(), t
            ));
        for (auto v: vertices)
            addEdge(u, v, g[v][t]);
    }

    cout << (total - mincut) / 2 << '\n';
```

```
    return 0;
}
```

7.2 Matching

matching.cpp

Description: Returns the maximum matching of a bipartite graph
Time: $\mathcal{O}(\log N)$.

58 lines

```
int l[NMAX], r[NMAX];
bool vis[NMAX], ok[NMAX], coverL[NMAX], coverR[NMAX];
vector<int> adj[NMAX], adjt[NMAX];
bool pairUp(int v) {
  if(vis[v]) return false;
  vis[v] = true;
  for(auto u : adj[v])
    if(!r[u]) {
      l[v] = u;
      r[u] = v;
      return true;
    }
  for(auto u : adj[v])
    if(pairUp(r[u])) {
      l[v] = u;
      r[u] = v;
      return true;
    }
  return false;
}
int matching(int n) {
  int sz;
  bool changed;
  for(sz = 0, changed = true; changed; ) {
    memset(vis, 0, sizeof vis);
    changed = false;
    for(int i = 1; i <= n; ++i)
      if(!l[i] && pairUp(i)) ++sz, changed = true;
  }
  return sz;
}
void bfs(vector<int> adj[], int l[], int r[], int n) {
  queue<int> q;
  memset(vis, 0, sizeof vis);
  for(int i = 1; i <= n; ++i) if(!l[i]) q.push(i), vis[i] =
       true;
  for(; !q.empty(); q.pop()) {
    int v = q.front();
    ok[v] = true;
    for(auto u : adj[v])
      if(!vis[r[u]]) q.push(r[u]), vis[r[u]] = true;
  }
}
void cover(int v) {
  for(auto u : adj[v])
    if(!coverR[u]) {
      coverR[u] = true;
      coverL[r[u]] = false;
      cover(r[u]);
    }
}
sz = matching(n);
// getting all vertices which do NOT belong to ALL maximum
     matchings
// if ok[i] == false => i belongs to all maximum matchings
bfs(adj, l, r, n);
bfs(adjt, r, l, n);
//getting minimum vertex cover
for(int i = 1; i <= n; ++i) if(l[i]) coverL[i] = true;
for(int i = 1; i <= n; ++i) if(!l[i]) cover(i);
```

WeightedMatching.h

Description: Min cost perfect bipartite matching. Negate costs for max cost.
Time: $\mathcal{O}(N^3)$

57 lines

```
template<typename T>
int MinAssignment(const vector<vector<T>> &c) {
  int n = c.size(), m = c[0].size();       // assert(n <= m);
  vector<T> v(m), dist(m);                 // v: potential
  vector<int> L(n, -1), R(m, -1);          // matching pairs
  vector<int> index(m), prev(m);
  iota(index.begin(), index.end(), 0);

  auto residue = [&](int i, int j) { return c[i][j] - v[j]; };
  for (int f = 0; f < n; ++f) {
    for (int j = 0; j < m; ++j) {
      dist[j] = residue(f, j); prev[j] = f;
    }
    T w; int j, l;
    for (int s = 0, t = 0;;) {
      if (s == t) {
        l = s; w = dist[index[t++]];
        for (int k = t; k < m; ++k) {
          j = index[k]; T h = dist[j];
          if (h <= w) {
            if (h < w) { t = s; w = h; }
            index[k] = index[t]; index[t++] = j;
          }
        }
        for (int k = s; k < t; ++k) {
          j = index[k];
          if (R[j] < 0) goto aug;
        }
      }
      int q = index[s++], i = R[q];
      for (int k = t; k < m; ++k) {
        j = index[k];
        T h = residue(i,j) - residue(i,q) + w;
        if (h < dist[j]) {
          dist[j] = h; prev[j] = i;
          if (h == w) {
            if (R[j] < 0) goto aug;
            index[k] = index[t]; index[t++] = j;
          }
        }
      }
    }
  aug:
    for(int k = 0; k < l; ++k)
      v[index[k]] += dist[index[k]] - w;
    int i;
    do {
      R[j] = i = prev[j];
      swap(j, L[i]);
    } while (i != f);
  }
  T ret = 0;
  for (int i = 0; i < n; ++i) {
    ret += c[i][L[i]]; // (i, L[i]) is a solution
  }
  return ret;
}
```

7.3 DFS algorithms

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected multigraph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle. HOWEVER, note that we are outputting bridges as BCC's here, because we might be interested in vertex bcc's, not edge bcc's.
To get the articulation points, look for vertices that are in more than 1 BCC.
To get the bridges, look for biconnected components with one edge
Time: $\mathcal{O}(E+V)$

54 lines

```
struct BCC {
  vector<pair<int, int>> edges;
  vector<vector<int>> G;
  vector<int> enter, low, stk;

  BCC(int n) : G(n), enter(n, -1) {}

  int AddEdge(int a, int b) {
    int ret = edges.size();
    edges.emplace_back(a, b);
    G[a].push_back(ret);
    G[b].push_back(ret);
    return ret;
  }

  template<typename Iter>
  void Callback(Iter bg, Iter en) {
    for (Iter it = bg; it != en; ++it) {
      auto edge = edges[*it];
      // Do something useful
    }
  }

  void Solve() {
    for (int i = 0; i < (int)G.size(); ++i)
      if (enter[i] == -1) {
        dfs(i, -1);
      }
  }

  int timer = 0;
  int dfs(int node, int pei) {
    enter[node] = timer++;
    int ret = enter[node];

    for (auto ei : G[node]) if (ei != pei) {
      int vec = (edges[ei].first ^ edges[ei].second ^ node);
      if (enter[vec] != -1) {
        ret = min(ret, enter[vec]);
        if (enter[vec] < enter[node])
          stk.push_back(ei);
      } else {
        int sz = stk.size(), low = dfs(vec, ei);
        ret = min(ret, low);
        stk.push_back(ei);
        if (low >= enter[node]) {
          Callback(stk.begin() + sz, stk.end());
          stk.resize(sz);
        }
      }
    }
    return ret;
  }
};
```

2satValeriu.h

Description: None
Usage: `ask Djok`

<bits/stdc++.h> 39 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 100005;

int rs[N];
bool viz[N];
vector<int> lda[N], ldat[N], order;

void addEdge(int x, int y) {
  x = x < 0 ? -2 * x - 1 : 2 * x - 2;
  y = y < 0 ? -2 * y - 1 : 2 * y - 2;
  lda[x ^ 1].push_back(y);
  lda[y ^ 1].push_back(x);
  ldat[y].push_back(x ^ 1);
  ldat[x].push_back(y ^ 1);
}

void dfs1(int x) {
  viz[x] = 1;
  for(auto to : lda[x]) if(!viz[to]) dfs1(to);
  order.push_back(x);
}

void dfs2(int x) {
  if(rs[x]) puts("NO"), exit(0);
  viz[x] = 0; rs[x ^ 1] = 1;
  for(auto to : ldat[x]) if(viz[to]) dfs2(to);
}

void solve2SAT() {
  for(int i = 0; i < N; ++i) if(!viz[i]) dfs1(i);
  reverse(order.begin(), order.end());
  for(auto it : order) if(viz[it] && viz[it ^ 1]) dfs2(it);
}

int main() {
  return 0;
}
```

7.4 Trees

CompressTree.h

Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most $|S|-1$) pairwise LCA's and compressing edges. Returns the nodes of the reduced tree, while at the same time populating a link array that stores the new parents. The root points to -1.
Time: $\mathcal{O}(|S| * (\log |S| + LCA\_Q))$

"LCA.h" 18 lines

```
vector<int> CompressTree(vector<int> v, LCA& lca,
                         vector<int>& link) {
  auto cmp = [&](int a, int b) {
    return lca.enter[a] < lca.enter[b];
  };
  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = (int)v.size() - 1; i > 0; --i)
    v.push_back(lca.Query(v[i - 1], v[i]));

  sort(v.begin(), v.end(), cmp);
  v.erase(unique(v.begin(), v.end()), v.end());

  for (int i = 0; i < (int)v.size(); ++i)
    link[v[i]] = (i == 0 ? -1 : lca.Query(v[i - 1], v[i]));
  return v;
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most log(n) light edges.

69 lines

```
struct HeavyLight {
  struct Node {
    int jump, subsize, depth, lin, parent;
    vector<int> leg;
  };
  vector<Node> T;
  bool processed = false;

  HeavyLight(int n) : T(n) {}

  void AddEdge(int a, int b) {
    T[a].leg.push_back(b);
    T[b].leg.push_back(a);
  }

  void Preprocess() {
    dfs_sub(0, -1); dfs_jump(0, 0);
    processed = true;
  }

  // Gets the position in the HL linearization
  int GetPosition(int node) {
    assert(processed);
    return T[node].lin;
  }

  // Gets an array of ranges of form [li...ri)
  // that correspond to the ranges you would need
  // to query in the underlying structure
  vector<pair<int, int>> GetPathRanges(int a, int b) {
    assert(processed);
    vector<pair<int, int>> ret;
    while (T[a].jump != T[b].jump) {
      if (T[T[a].jump].depth < T[T[b].jump].depth)
        swap(a, b);

      ret.emplace_back(T[T[a].jump].lin, T[a].lin + 1);
      a = T[T[a].jump].parent;
    }
    if (T[a].depth < T[b].depth) swap(a, b);
    ret.emplace_back(T[b].lin, T[a].lin + 1);
    return ret;
  }

  int dfs_sub(int x, int par) {
    auto &node = T[x];
    node.subsize = 1; node.parent = par;
    if (par != -1) {
      node.leg.erase(find(node.leg.begin(),
                          node.leg.end(), par));
      node.depth = 1 + T[par].depth;
    }
    for (auto vec : node.leg)
      node.subsize += dfs_sub(vec, x);
    return node.subsize;
  }

  int timer = 0;
  void dfs_jump(int x, int jump) {
    auto &node = T[x];
    node.jump = jump; node.lin = timer++;
    iter_swap(node.leg.begin(), max_element(node.leg.begin(),
      node.leg.end(), [&](int a, int b) {
        return T[a].subsize < T[b].subsize;
    }));
    for (auto vec : node.leg)
      dfs_jump(vec, vec == node.leg.front() ? jump : vec);
  }
};
```

Strings (8)

ZFunction.h

Description: Given a string s, computes the length of the longest common prefix of s[i..] and s[0..] for each i > 0 !!
Usage: `Zfunction("abacaba") => {0, 0, 1, 0, 3, 0, 1}`
Time: $\mathcal{O}(N)$

<bits/stdc++.h> 21 lines

```
vector<int> ZFunction(string s) {
  int n = s.size();
  vector<int> z(n, 0);
  int L = 0, R = 0;
  for (int i = 1; i < n; i++) {
    if (i > R) {
      L = R = i;
      while (R < n && s[R - L] == s[R]) R++;
      z[i] = R - L; R--;
    } else {
      int k = i-L;
      if (z[k] < R - i + 1) z[i] = z[k];
      else {
        L = i;
        while (R < n && s[R - L] == s[R]) R++;
        z[i] = R - L; R--;
      }
    }
  }
  return z;
}
```

Manacher.h

Description: Given a string s, computes the length of the longest palindromes centered in each position (for parity == 1) or between each pair of adjacent positions (for parity == 0).
Usage: `Manacher("abacaba", 1) => {0, 1, 0, 3, 0, 1, 0}`
`Manacher("aabbaa", 0) => {1, 0, 3, 0, 1}`
Time: $\mathcal{O}(N)$

14 lines

```
vector<int> Manacher(string s, bool parity) {
  int n = s.size(), z = parity, l = 0, r = 0;
  vector<int> ret(n - !z, 0);

  for (int i = 0; i < n - !z; ++i) {
    if (i + !z < r) ret[i] = min(r - i, ret[l + r - i - !z]);
    int L = i - ret[i] + !z, R = i + ret[i];
    while (L - 1 >= 0 && R + 1 < n && s[L - 1] == s[R + 1])
      ++ret[i], --L, ++R;
    if (R > r) l = L, r = R;
  }

  return ret;
}
```

PalindromicTree.h

Description: A trie-like structure for keeping track of palindromes of a string s. It has two roots, 0 (for even palindromes) and 1 (for odd palindromes). Each node stores the length of the palindrome, the count and a link to the longest "aligned" subpalindrome. Can be made online from left to right
Time: $\mathcal{O}(N)$

53 lines

```
struct PalTree {
  struct Node {
    map<char, int> leg;
    int link, len, cnt;
  };
  vector<Node> T;
  int nodes = 2;

  PalTree(string str) : T(str.size() + 2) {
    T[1].link = T[1].len = 0;
    T[0].link = T[0].len = -1;

    int last = 0;
    for (int i = 0; i < (int)str.size(); ++i) {
      char now = str[i];

      int node = last;
      while (now != str[i - T[node].len - 1])
        node = T[node].link;

      if (T[node].leg.count(now)) {
        node = T[node].leg[now];
        T[node].cnt += 1;
        last = node;
        continue;
      }

      int cur = nodes++;
      T[cur].len = T[node].len + 2;
      T[node].leg[now] = cur;

      int link = T[node].link;
      while (link != -1) {
        if (now == str[i - T[link].len - 1] &&
            T[link].leg.count(now)) {
          link = T[link].leg[now];
          break;
        }
        link = T[link].link;
      }
      if (link <= 0) link = 1;

      T[cur].link = link;
      T[cur].cnt = 1;

      last = cur;
    }

    for (int node = nodes - 1; node > 0; --node) {
      T[T[node].link].cnt += T[node].cnt;
    }
  }
};
```

SuffAtumatValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 101 lines

```
// Kth lexi substring with repetitions, aaa = {a, a, a, aa, aa,
      aaa}

const int N = 200005;
const int ALFA = 26;

struct state {
  int link, len, next[ALFA];
  long long cnt;

  state(int _link = 0, int _len = 0, long long _cnt = 0) {
    memset(next, ~0, sizeof(next));
    link = _link; len = _len; cnt = _cnt;
  }
};

int i, n, last, sz, k;
long long dp[N];
bool viz[N];
string s;
state sfa[N];
vector<int> lens[N];

void addLetter(char c) {
  int now = sz++, p = last;
  sfa[now].len = sfa[last].len + 1;
  sfa[now].cnt = 1;

  for(; p != -1 && sfa[p].next[c] == -1; p = sfa[p].link) sfa[p
      ].next[c] = now;

  if(p == -1) sfa[now].link = 0;
  else {
    int q = sfa[p].next[c];
    if(sfa[q].len == sfa[p].len + 1) sfa[now].link = q;
    else {
      int clone = sz++;
      sfa[clone] = sfa[q];
      sfa[clone].len = sfa[p].len + 1;
      sfa[clone].cnt = 0;

      for(; p != -1 && sfa[p].next[c] == q; p = sfa[p].link)
          sfa[p].next[c] = clone;

      sfa[q].link = sfa[now].link = clone;
    }
  }

  last = now;
}

void dfs(int x) {
  viz[x] = 1;
  dp[x] = sfa[x].cnt;

  for(int i = 0; i < 26; ++i) {
    if(sfa[x].next[i] == -1) continue;

    if(!viz[sfa[x].next[i]]) dfs(sfa[x].next[i]);
    dp[x] += dp[sfa[x].next[i]];
  }
}

void Solve(int x, int need) {
  if(!need) return;

  if(x && sfa[x].cnt >= need) return;
  if(x) need -= sfa[x].cnt;

  for(int i = 0; i < 26 && need; ++i) {
    if(sfa[x].next[i] == -1) continue;

    if(dp[sfa[x].next[i]] < need) need -= dp[sfa[x].next[i]];
    else {
      cout << char(i + 'a');
      Solve(sfa[x].next[i], need);
      return;
    }
  }
}

int main() {
  ios_base::sync_with_stdio(0);

  sz = 1; sfa[0].link = -1;

  cin >> s >> k;

  if(k > 1LL * s.size() * (s.size() + 1) / 2) return cout << "
       No such line.\n", 0;

  for(auto it : s) addLetter(it - 'a');

  for(i = 1; i < sz; ++i) lens[sfa[i].len].push_back(i);

  for(i = s.size(); i; --i)
    for(auto it : lens[i])
      sfa[sfa[it].link].cnt += sfa[it].cnt;

  dfs(0);

  Solve(0, k);

  return 0;
}
```

stringHashValeriu.h

Description: None
Usage: ask Djok

20 lines

```
typedef unsigned long long ull;

const int N = 100005;
const int P = 239017;

string s;
ull h[N], deg[N];

void init() {
  int n = s.size();
  deg[0] = 1; h[0] = 0;
  for(int i = 0; i < n; ++i) {
    h[i + 1] = h[i] * P + s[i];
    deg[i + 1] = deg[i] * P;
  }
}

ull getHash(int i, int len) {
  return h[i + len] - h[i] * deg[len];
}
```

suffArrayValeriu.h

Description: None
Usage: ask Djok

<bits/stdc++.h> 41 lines

```
#pragma GCC optimize("Ofast")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx
     ,tune=native")

const int N = 200005;
```

```
struct lol {
  int x, y, poz;

  bool operator < (const lol &b) const {
    if(x == b.x) return y < b.y;
    return x < b.x;
  }
};

int i, j, sa[20][N], n;
lol b[N];
string s;

int main() {
  cin >> s; n = s.size();
  for(i = 1; i <= n; ++i) sa[0][i] = s[i - 1];

  for(i = 1; (1 << i) <= n; ++i) {
    for(j = 1; j <= n; ++j) {
      b[j].x = sa[i - 1][j];
      b[j].poz = j;
      if(j + (1 << (i - 1)) <= n) b[j].y = sa[i - 1][j + (1 <<
           i)];
      else b[j].y = -1;
    }

    sort(b + 1, b + n + 1);

    for(j = 1; j <= n; ++j)
      if(b[j].x != b[j - 1].x || b[j].y != b[j - 1].y) sa[i][b[
           j].poz] = j;
      else sa[i][b[j].poz] = sa[i][b[j - 1].poz];
  }

  for(int k = 1; k <= n; ++k) cerr << sa[i - 1][k] << '\n';

  return 0;
}
```

AhoCorasick.h

Description: Aho-Corasick algorithm builds an automaton for multiple pattern string matching

Time: $\mathcal{O}(N * log(sigma))$ where N is the total length

<bits/stdc++.h> 48 lines

```
struct AhoCorasick {
  struct Node {
    int link;
    map<char, int> leg;
  };
  vector<Node> T;
  int root = 0, nodes = 1;

  AhoCorasick(int sz) : T(sz) {}

  // Adds a word to trie and returns the end node
  int AddWord(const string &word) {
    int node = root;
    for (auto c : word) {
      auto &nxt = T[node].leg[c];
      if (nxt == 0) nxt = nodes++;
      node = nxt;
    }
    return node;
  }

  // Advances from a node with a character (like an automaton)
  int Advance(int node, char chr) {
    while (node != -1 && T[node].leg.count(chr) == 0)
      node = T[node].link;
    if (node == -1) return root;
    return T[node].leg[chr];
  }

  // Builds links
  void BuildLinks() {
    queue<int> Q;
    Q.push(root);
    T[root].link = -1;

    while (!Q.empty()) {
      int node = Q.front();
      Q.pop();

      for (auto &p : T[node].leg) {
        int vec = p.second;
        char chr = p.first;
        T[vec].link = Advance(T[node].link, chr);
        Q.push(vec);
      }
    }
  }
};
```

Various (9)

9.1 Misc. algorithms

AlphaBeta.h

Description: Uses the alpha-beta pruning method to find score values for states in games (minimax)

8 lines

```
int AlphaBeta(state s, int alpha, int beta) {
  if (s.finished()) return s.score();
  for (state t : s.next()) {
    alpha = max(alpha, -AlphaBeta(t, -beta, -alpha));
    if (alpha >= beta) break;
  }
  return alpha;
}
```

9.2 Dynamic programming

DivideAndConquerDP.h

Description: Given $a[i] = \min_{lo(i) \le k < hi(i)}(f(i,k))$ where the (minimal) optimal k increases with i, computes $a[i]$ for $i = L..R-1$.

Time: $\mathcal{O}((N + (hi - lo)) \log N)$

18 lines

```
struct DP { // Modify at will:
  int lo(int ind) { return 0; }
  int hi(int ind) { return ind; }
  ll f(int ind, int k) { return dp[ind][k]; }
  void store(int ind, int k, ll v) { res[ind] = pii(k, v); }

  void rec(int L, int R, int LO, int HI) {
    if (L >= R) return;
    int mid = (L + R) >> 1;
    pair<ll, int> best(LLONG_MAX, LO);
    rep(k, max(LO,lo(mid)), min(HI,hi(mid)))
      best = min(best, make_pair(f(mid, k), k));
    store(mid, best.second, best.first);
    rec(L, mid, LO, best.second+1);
    rec(mid+1, R, best.second, HI);
  }
  void solve(int L, int R) { rec(L, R, INT_MIN, INT_MAX); }
};
```

9.3 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`).

9.4 Optimization tricks

9.4.1 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.
- `rep(b,0,K) rep(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

9.4.2 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.
- `#pragma GCC optimize ("trapv")` kills the program on integer overflows (but is really slow).

Unrolling.h

5 lines

```
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment, if needed
while (i + 4 <= to) { F F F F }
while (i < to) F
```

Techniques (A)

techniques.txt

159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contigous subvector sum
  Invariants
  Huffman encoding
Graph teory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algoritm
  MST: Prim's algoritm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Eulercykler
  Flow networks
  * Augumenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cutvertices, cutedges och biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programmering
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euklidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's small theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynom hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```