Universidad Complutense de Madrid

#define int long long

Jorge Hernández Palop, Noah Dris Sánchez, Daniel López Piris

SWERC 2025

23 de Noviembre de 2025

Contest (1)

template.cpp

14 lines

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define int long long
typedef pair<int, int> pii;
typedef vector<int> vi;

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

troubleshoot.txt

52 lines

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

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

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

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

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

Strings (2)

KMP.h

Description: kmp[i] = The length of the longest non trivial suffix that ends at position i and coincides with a prefix of s.
Time: $\mathcal{O}(n)$

8 lines

```
vi kmp(const string& s) {
  vi res(sz(s));
  rep(i, 1, sz(s)) {
    int k = res[i - 1];
    while(k > 0 && s[k] != s[i]) k = res[k - 1];
    res[i] = k + (s[k] == s[i]);
  }
  return res; }
```

Zfunction.h

Description: zfun[i] = The length of the longest non trivial prefix that starts at position i and coincides with a prefix of s. zfun[0] = 0.
Time: $\mathcal{O}(n)$

8 lines

```
vi zfun(const string& s) {
  vi z(sz(s)); int l = 0;
  rep(i, 1, sz(s)) {
    z[i] = max(min(z[i - l], z[l] + l - i), 0LL);
    while(i+z[i] < sz(s) && s[z[i]]==s[i+z[i]]) z[i]++;
    if(z[i] + i > z[l] + l) l = i;
  }
  return z; }
```

Manacher.h

Description: For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).
Time: $\mathcal{O}(N)$

13 lines

```
array<vi, 2> manacher(const string& s) {
  int n = sz(s);
  array<vi,2> p = {vi(n+1), vi(n)};
  rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
    int t = r-i+!z;
    if (i<r) p[z][i] = min(t, p[z][l+t]);
    int L = i-p[z][i], R = i+p[z][i]-!z;
    while (L>=1 && R+1<n && s[L-1] == s[R+1])
      p[z][i]++, L--, R++;
    if (R>r) l=L, r=R;
  }
  return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.
Usage: `rotate(v.begin(), v.begin()+minRotation(v), v.end());`
Time: $\mathcal{O}(N)$

8 lines

```
int minRotation(string s) {
  int a=0, N=sz(s); s += s;
  rep(b,0,N) rep(k,0,N) {
    if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1);
        break;}
    if (s[a+k] > s[b+k]) { a = b; break; }
  }
  return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. `sa[i]` is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size $n+1$, and `sa[0] = n`. The `lcp` array contains longest common prefixes for neighbouring strings in the suffix array: `lcp[i] = lcp(sa[i], sa[i-1])`, `lcp[0] = 0`. The input string must not contain any nul chars.
Time: $\mathcal{O}(n \log n)$

22 lines

```
struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string s, int lim=256) { // or vector<int>
    s.push_back(0); int n = sz(s), k = 0, a, b;
    vi x(all(s)), y(n), ws(max(n, lim));
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    for (int i = 0, j; i < n - 1; lcp[x[i++]] = k)
      for (k && k--, j = sa[x[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching.
Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns.

32 lines

```
#define FIRST 'a'
#define ALPHA 26
struct Node { int s = 0, cnt = 0, ch[ALPHA] = {}; };
vector<int> AhoCora(vector<string>& str, string& s) {
    int n = str.size();
    vector<Node> t(1);
    vector<int> idx(n);
    rep(i, 0, n) { // Build Trie
        int u = 0;
        for (char c : str[i]) {
            if (!t[u].ch[c -= FIRST]) t[u].ch[c] = t.size(), t.emplace_back();
            u = t[u].ch[c];
        }
        idx[i] = u;
    }
    vector<int> q; // Build Automaton
    rep(i, 0, ALPHA) if (t[0].ch[i]) q.push_back(t[0].ch[i]);
    rep(head, 0, q.size()) {
        int u = q[head];
        rep(i, 0, ALPHA) {
            int &v = t[u].ch[i], f = t[t[u].s].ch[i];
            if (v) t[v].s = f, q.push_back(v);
            else v = f;
        }
    }
    int u = 0; // Process text
    for (char c : s) t[u = t[u].ch[c - FIRST]].cnt++;
    for (int i = q.size(); i--; ) t[t[q[i]].s].cnt += t[q[i]].cnt;
    vector<int> res(n);
    rep(i, 0, n) res[i] = t[idx[i]].cnt;
    return res;
}
```

Number theory (3)

3.1 Modular arithmetic

euclid.h

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

5 lines

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

3.1.1 Bézout's identity

$$ax + by = d$$

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

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

ModInverse.h

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

3 lines

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

ModMulLL.h

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

11 lines

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

CRT.h

Description: Chinese Remainder Theorem.
`crt(a, m, b, n)` computes x such that $x \equiv a \pmod m$, $x \equiv b \pmod n$. If $|a| < m$ and $|b| < n$, x will obey $0 \leq x < \operatorname{lcm}(m,n)$. Assumes $mn < 2^{62}$.
Time: $\log(n)$

"euclid.h" 7 lines

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

ModLog.h

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

9 lines

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

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
`modsum(to, c, k, m)` $= \sum_{i=0}^{\mathrm{to}-1} (ki+c)\%m$. `divsum` is similar but for floored division.
Time: $\log(m)$, with a large constant.

11 lines

```
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }
ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  ull to2 = (to * k + c) / m;
  return res+(k?(to-1)*to2-divsum(to2, m-1-c, m, k):0);
}
ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m, k = ((k % m) + m) % m;
  return to*c + k*sumsq(to) -m*divsum(to, c, k, m);
}
```

ModSqrt.h

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

"ModPow.h" 17 lines

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

PrimRoot.h

Description: Given an integer n it return an integer g that $\{g^k : k \in \mathbb{N}_0\} = \mathbb{Z}_n*$. The primitive Root is also called generator of the group $\mathbb{Z}_n*$. This code is only valid when n is a prime number.
if n is p^k and g is a primitive root of p:
$g^p \equiv g \pmod{p^2} \iff g+p$ is a primitive root of n
$g^p \not\equiv g \pmod{p^2} \iff g$ is a primitive root of n
if n is $2 \cdot p^k$ and g is a primitive root of p^k:
g is odd $\iff g$ is a primitive root of n
g is even $\iff g + p^k$ is a primitive root of n
There are $\phi(\phi(p^a))$ many. For $a > 2$, the group $\mathbb{Z}_{2^a}^\times \cong \mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.
Time: Assuming the generalized Riemann hypothesis, $\mathcal{O}\left(\log^8 n\right)$

"ModPow.h" 13 lines

```
int generator (int p) {
  vector<int> fact; int phi=p-1,n=phi,i,res,ok=0;
  for (i=2; i*i<=n; ++i)
    if (n % i == 0) {
      fact.push_back(i);
      while (n % i == 0) n /= i;
    }
  if(n > 1) fact.push_back (n);
  for (res = 2; (res<=p)&&(!ok); ++res)
    for (ok = true, i=0; i<fact.size() && ok; ++i)
      ok &= powmod(res, phi/fact[i], p) != 1;
  return (ok? res-1:-1);
}
```

3.1.2 Digital Root

Given an integer n and a base b. We call the digital root,$db_b(n)$, of n the sum of its digits in the base b:
$db_b(n) = 1 + ((n-1) \mod (b-1))$

1. $db_b(x+y) = db_b(db_b(x) + db_b(y))$
2. $db_b(xy) = db_b(db_b(x)db_b(y))$
3. $db_b(x-y) \equiv db_b(x) - db_b(y) \pmod{b-1}$

3.2 Important Functions

PrimeCounting.h

Description: Given an integer it gives you $\pi(n)$.
Time: $\mathcal{O}\left(n^{3/4}F(n) + \sqrt{n}PREF(n)\right)$

19 lines

```
int count_primes(int n) {
    auto f = [&](int n) {return 1;}; // (f(ab) = f(a)f(
        b))
    auto pref = [&](int n) {return n;}; // should
        return sum_{i=1..n} f(i)
    vector<int> v; v.reserve((int)sqrt(n) * 2 + 20);
    int sq, k = 1;
    for (; k * k <= n; k++) v.push_back(k);
    sq = --k;
    if (k * k == n) k--;
    for (; k >= 1; k--) v.push_back(n / k);
    vector<int> s(v.size());
    for (int i = 0; i < s.size(); i++) s[i] = pref(v[i
        ]) - 1;
    auto geti = [&](int x) {return (x<=sq? (int)x-1:(
        int)(v.size() - (n / x)));};
    for (int p = 2; p * p <= n; p++)
        if (s[p - 1] != s[p - 2])
            for (int i = (int)v.size() - 1; p*p <= v[i]
                && i >= 0; --i)
                s[i] -= (s[geti(v[i] / p)] - s[p-2]) *
                    f(p);

    return s.back();
}
```

MillerRabin.h

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

"ModMulLL.h" 10 lines

```
bool MillerRabin(unsigned int n) { // returns true if n
    is prime, else returns false.
  unsigned int r = __builtin_ctzll(n-1); int d = n >> r
    ;
  for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
      37, 41}) {
    if (n == a) return true;
    unsigned int x = modpow(a, d, n), res = !(x == 1 ||
        x == n - 1);
    for(int i = 1; i < r; i++) x = (__int128)x*x%n, res
        &=(x!=n-1);
    if(res) return false;
  }
  return (n>=2);
}
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).
Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

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

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

3.2.1 Möbius function

$$\mu(n) = \begin{cases} 0 & n \text{ is not square free} \\ (-1)^{\Omega(n)} & \text{otherwise} \end{cases}$$

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

$$(f * g)(n) = \sum_{d|n} f(d) g\left(\frac{n}{d}\right)$$

1. **Distributive**: $f * (g + h) = f * g + f * h$
2. **Möbius inversion**: $f * 1 = g \iff g * \mu = f$

The most important relations are:

$$\begin{aligned} \epsilon &= 1 * \mu & \sigma_1 &= \varphi * \sigma_0 \\ \Omega &= 1_{\mathcal{P}} * 1 & \varphi * 1 &= Id \\ \sigma_k &= Id_k * 1 & \sigma_0^3 * 1 &= (\sigma_0 * 1)^2 \\ \omega &= 1_{\mathbb{P}} * 1 & |\mu| * 1 &= 2^{\omega} \end{aligned}$$

$\mathcal{P}$ are the prime powers and $\mathbb{P}$ the primes.

3.2.2 Multiplicative functions

LinearSieve.h

Description: Linear Sieve for prime numbers. Can be used for precomputing multiplicative functions

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

13 lines

```
void sieve () {
  //asignar 1
  for (int i=2; i <= N; i++) {
    if (lp[i] == 0){ // i es primo
      lp[i] = i; pr.push_back(i); // asignar primo
    }
    for (int j = 0; i * pr[j] <= N; j++) {
      lp[i*pr[j]] = pr[j];
      if (i%pr[j] == 0) {/*asignar multiplo*/;break;}
      else{/*asignar coprimo*/}
    }
  }
}
```

PrefixSumOpt.h

Description: Let f the multiplactive function to compute its prefix sum. Let g and c multiplicative functions so that $f * g = c$ and both 3 can be computed in constant time.

Time: $\mathcal{O}(n^{2/3})$

8 lines

```
unordered_map<int, int> mem;
int calc (int x, int ans = 0) {
  if(x<= th) return (x<=0? 0: p_f(x));
  if(mem.count(x) != 0) return mem[x];
  for (int i = 2, la; i <= x; i = la + 1)
    ans += (p_g(la = x/(x/i))-p_g(i-1))*calc(x / i);
  return mem[x] = (p_c(x) - ans)/p_g(1);
}
```

3.3 Misc

3.3.1 Primes

$p = 962592769$ is such that $2^{21} \mid p-1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). $pi(10^6) = 78498$.

3.3.2 Fractions

ContinuedFractions.h

Description: Given N and a real number $x \geq 0$, finds the closest rational approximation p/q with $p, q \leq N$. It will obey $|p/q - x| \leq 1/qN$. For consecutive convergents, $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$. ($p_k/q_k$ alternates between $> x$ and $< x$.) If x is rational, y eventually becomes ∞; if x is the root of a degree 2 polynomial the a's eventually become cyclic.

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

21 lines

```
typedef double d; // for N ~ 1e7; long double for N ~ 1
    e9
pair<ll, ll> approximate(d x, ll N) {
  ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y
      = x;
  for (;;) {
    ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q :
         inf),
       a = (ll)floor(y), b = min(a, lim),
       NP = b*P + LP, NQ = b*Q + LQ;
    if (a > b) {
      // If b > a/2, we have a semi-convergent that
        gives us a
      // better approximation; if b = a/2, we *may*
        have one.
      // Return {P, Q} here for a more canonical
        approximation.
      return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (
          d)Q)) ?
        make_pair(NP, NQ) : make_pair(P, Q);
    }
    if (abs(y = 1/(y - (d)a)) > 3*N) {
      return {NP, NQ};
    }
    LP = P; P = NP;
    LQ = Q; Q = NQ;
  }
}
```

FracBinarySearch.h

Description: Given f and N, finds the smallest fraction $p/q \in [0,1]$ such that $f(p/q)$ is true, and $p, q \leq N$. You may want to throw an exception from f if it finds an exact solution, in which case N can be removed.

Usage: `fracBS([](Frac f) { return f.p>=3*f.q; }, 10);` `// {1,3}`

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

25 lines

```
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
  bool dir = 1, A = 1, B = 1;
  Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search
      (0, N]
  if (f(lo)) return lo;
  assert(f(hi));
  while (A || B) {
    ll adv = 0, step = 1; // move hi if dir, else lo
    for (int si = 0; step; (step *= 2) >>= si) {
      adv += step;
      Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
      if (abs(mid.p) > N || mid.q > N || dir == !f(mid)
          ) {
        adv -= step; si = 2;
      }
    }
    hi.p += lo.p * adv;
    hi.q += lo.q * adv;
    dir = !dir;
    swap(lo, hi);
    A = B; B = !!adv;
  }
  return dir ? hi : lo;
}
```

3.3.3 Pythagorean Triples

The Pythagorean triples are uniquely generated by

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

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

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0
        +b, b=c;
    a.pop_back();
  }
};
```

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 $x[k] = c * \cos(k/(n-1)*\pi), k = 0 \dots n-1$.

Time: $\mathcal{O}(n^2)$

13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

BerlekampMassey.h

Description: Recovers any n-order linear recurrence relation from the first $2n$ terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size $\leq n$.
Usage: `berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}`
Time: $\mathcal{O}\left(N^2\right)$

"../number-theory/ModPow.h" 20 lines

```
vector<ll> berlekampMassey(vector<ll> s) {
  int n = sz(s), L = 0, m = 0;
  vector<ll> C(n), B(n), T;
  C[0] = B[0] = 1;

  ll b = 1;
  rep(i,0,n) { ++m;
    ll d = s[i] % mod;
    rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
    if (!d) continue;
    T = C; ll coef = d * modpow(b, mod-2) % mod;
    rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
    if (2 * L > i) continue;
    L = i + 1 - L; B = T; b = d; m = 0;
  }

  C.resize(L + 1); C.erase(C.begin());
  for (ll& x : C) x = (mod - x) % mod;
  return C;
}
```

LinearRecurrence.h

Description: Generates the k'th term of an n-order linear recurrence $S[i] = \sum_j S[i-j-1]tr[j]$, given $S[0 \ldots \geq n-1]$ and $tr[0 \ldots n-1]$. Faster than matrix multiplication. Useful together with Berlekamp–Massey.
Usage: `linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number`
Time: $\mathcal{O}\left(n^2 \log k\right)$

26 lines

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
  int n = sz(tr);

  auto combine = [&](Poly a, Poly b) {
    Poly res(n * 2 + 1);
    rep(i,0,n+1) rep(j,0,n+1)
      res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
    for (int i = 2 * n; i > n; --i) rep(j,0,n)
      res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j
          ]) % mod;
    res.resize(n + 1);
    return res;
  };

  Poly pol(n + 1), e(pol);
  pol[0] = e[1] = 1;

  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
  }

  ll res = 0;
  rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
  return res;
}
```

4.2 Optimization

GoldenSectionSearch.h

Description: Finds the argument minimizing the function f in the interval $[a, b]$ assuming f is unimodal on the interval, i.e. has only one local minimum and no local maximum. The maximum error in the result is *eps*. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.
Usage: `double func(double x) { return 4+x+.3*x*x; }`
`double xmin = gss(-1000,1000,func);`
Time: $\mathcal{O}\left(\log((b-a)/\epsilon)\right)$

14 lines

```
double gss(double a, double b, double (*f)(double)) {
  double r = (sqrt(5)-1)/2, eps = 1e-7;
  double x1 = b - r*(b-a), x2 = a + r*(b-a);
  double f1 = f(x1), f2 = f(x2);
  while (b-a > eps)
    if (f1 < f2) { //change to > to find maximum
      b = x2; x2 = x1; f2 = f1;
      x1 = b - r*(b-a); f1 = f(x1);
    } else {
      a = x1; x1 = x2; f1 = f2;
      x2 = a + r*(b-a); f2 = f(x2);
    }
  return a;
}
```

4.3 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}\left(N^3\right)$

15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b =
        j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$

38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 7 lines

```
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
  rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for 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: $\mathcal{O}(n^3)$

35 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c
        ]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

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

This is useful for solving problems on the type

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

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

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

Fails if the solution is not unique.

If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$

26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>&
    super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  rep(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[
        i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i
        ];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
}
```

4.4 Fourier transforms

FastFourierTransform.h

Description: fft(a) computes $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$ for all k. N must be a power of 2. Useful for convolution: `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by n, reverse(start+1, end), FFT back. Rounding is safe if $(\sum a_i^2 + \sum b_i^2)\log_2 N < 9 \cdot 10^{14}$ (in practice 10^{16}; higher for random inputs). Otherwise, use NTT/FFT-Mod.
Time: $\mathcal{O}(N \log N)$ with $N = |A| + |B|$ (~1s for $N = 2^{22}$)

35 lines

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vector<complex<long double>> R(2, 1);
  static vector<C> rt(2, 1);  // (^ 10% faster if
      double)
  for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i
      /2];
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-
          rolled)
      a[i + j + k] = a[i + j] - z;
      a[i + j] += z;
    }
}
vd conv(const vd& a, const vd& b) {
  if (a.empty() || b.empty()) return {};
  vd res(sz(a) + sz(b) - 1);
  int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
  vector<C> in(n), out(n);
  copy(all(a), begin(in));
  rep(i,0,sz(b)) in[i].imag(b[i]);
  fft(in);
  for (C& x : in) x *= x;
  rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
  fft(out);
  rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
  return res;
}
```

FastFourierTransformMod.h

Description: Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$ (in practice 10^{16} or higher). Inputs must be in $[0, \text{mod})$.
Time: $\mathcal{O}(N \log N)$, where $N = |A| + |B|$ (twice as slow as NTT or FFT)

"FastFourierTransform.h" 22 lines

```
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  vl res(sz(a) + sz(b) - 1);
  int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt
      (M));
  vector<C> L(n), R(n), outs(n), outl(n);
  rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] %
      cut);
  rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] %
      cut);
  fft(L), fft(R);
  rep(i,0,n) {
    int j = -i & (n - 1);
    outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
    outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) /
        1i;
  }
  fft(outl), fft(outs);
  rep(i,0,sz(res)) {
    ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])
        +.5);
    ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5)
        ;
    res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
  }
  return res;
}
```

NumberTheoreticTransform.h

Description: ntt(a) computes $\hat{f}(k) = \sum_x a[x]g^{xk}$ for all k, where $g = \text{root}^{(mod-1)/N}$. N must be a power of 2. Useful for convolution modulo specific nice primes of the form $2^a b + 1$, where the convolution result has size at most 2^a. For arbitrary modulo, see FFTMod. `conv(a, b) = c`, where $c[x] = \sum a[i]b[x-i]$. For manual convolution: NTT the inputs, multiply pointwise, divide by n, reverse(start+1, end), NTT back. Inputs must be in [0, mod).
Time: $\mathcal{O}(N \log N)$

"../number-theory/ModPow.h" 35 lines

```
const ll mod = (119 << 23) + 1, root = 62; // =
    998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26,
    479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
  int n = sz(a), L = 31 - __builtin_clz(n);
  static vl rt(2, 1);
  for (static int k = 2, s = 2; k < n; k *= 2, s++) {
    rt.resize(n);
    ll z[] = {1, modpow(root, mod >> s)};
    rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
  }
  vi rev(n);
  rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
  rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
  for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
      ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i
          + j];
      a[i + j + k] = ai - z + (z > ai ? mod : 0);
      ai += (ai + z >= mod ? z - mod : z);
    }
}
vl conv(const vl &a, const vl &b) {
  if (a.empty() || b.empty()) return {};
  int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s),
      n = 1 << B;
  int inv = modpow(n, mod - 2);
  vl L(a), R(b), out(n);
  L.resize(n), R.resize(n);
  ntt(L), ntt(R);
  rep(i,0,n)
    out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv %
        mod;
  ntt(out);
  return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

Description: Transform to a basis with fast convolutions of the form $c[z] = \sum_{z=x\oplus y} a[x] \cdot b[y]$, where $\oplus$ is one of AND, OR, XOR. The size of a must be a power of two.
Time: $\mathcal{O}(N \log N)$

16 lines

```
void FST(vi& a, bool inv) {
  for (int n = sz(a), step = 1; step < n; step *= 2) {
    for (int i = 0; i < n; i += 2 * step) rep(j,i,i+
        step) {
      int &u = a[j], &v = a[j + step]; tie(u, v) =
        inv ? pii(v - u, u) : pii(v, u + v); // AND
        inv ? pii(v, u - v) : pii(u + v, u); // OR
        pii(u + v, u - v);                   // XOR
    }
  }
  if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
  FST(a, 0); FST(b, 0);
  rep(i,0,sz(a)) a[i] *= b[i];
  FST(a, 1); return a;
}
```

Combinatorial (5)

Factorial

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.
Time: $\mathcal{O}(n)$

6 lines

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

Cycles

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

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

Derangements

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

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

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

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

Partition function

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

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

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

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

Lucas' Theorem

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

Binomials

multinomial.h

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

5 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;
}
```

Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able).

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

Sums of powers:

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

Stirling numbers of the first kind

Number of permutations on n items with k cycles.

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

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

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. $\pi(j) > j$.

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

Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

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

Bell numbers

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

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

Labeled unrooted trees

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

Catalan numbers

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

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

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

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

Catalan Convolution

$$C_n^{(k)} = \sum_{a_1+a_2+\dots+a_k=n} C_{a_1} C_{a_2} \dots C_{a_k}$$

$$C_n^{(k)} = \frac{k+1}{n+k+1}\binom{2n+k}{n}$$

Mathematics (6)

6.1 Trigonometry

$$\sin(v+w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v+w) = \cos v \cos w - \sin v \sin w$$
$$\sin v + \sin w = 2 \sin \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v+w}{2} \cos \frac{v-w}{2}$$
$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

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

6.2 Sums and Series

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

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$
$$1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$$
$$1^6 + 2^6 + 3^6 + \dots + n^6 = \frac{(n^3+2n^2+x)(3n^4+6n^3-3n+1)}{42}$$

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

6.3 Probability theory

Binomial distribution

$\mathrm{Bin}(n,p)$, $n = 1, 2, \dots,$ $0 \leq p \leq 1$.

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

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

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

First success distribution

Amount of trials needed to get the first success in independent yes/no experiments. $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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

Poisson distribution

Amount 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 Po(λ), $\lambda = t\kappa$.

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

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

Uniform distribution

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

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

Exponential distribution

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

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

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

Normal distribution

$\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

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

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

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

6.3.1 Markov chains

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

Stationary 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.

Ergodicity

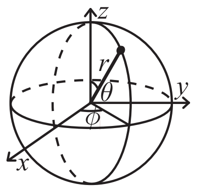
A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k\to\infty}\mathbf{P}^k = \mathbf{1}\pi$.

Absorption

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 $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Geometry (7)

7.0.1 Spherical coordinates



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

7.1 Geometric primitives

Point.h

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

35 lines

```
typedef double uds;
const uds EPS = 1e-12;
int sign(uds x) { return x < -EPS ? -1 : x > EPS; }
bool eq(uds a, uds b) { return abs(a-b) <= EPS; }
bool lt(uds a, uds b) { return a < b && abs(a-b) > EPS;
     }
bool gt(uds a, uds b) { return a > b && abs(a-b) > EPS;
     }
bool le(uds a, uds b) { return !gt(a, b); }
bool ge(uds a, uds b) { return !lt(a, b); }
#define div0(v) assert(!eq(v, 0))

struct vec2d {
  uds x, y;
  vec2d(): x(0), y(0) {}
  vec2d(uds x, uds y): x(x), y(y) {}

  vec2d operator+ (vec2d o) const { return {x+o.x, y+o.
      y}; }
  vec2d operator- (vec2d o) const { return {x-o.x, y-o.
      y}; }
  vec2d operator* (  uds k) const { return {x*k  , y*k
       }; }
  vec2d operator/ (  uds k) const { div0(k); return {x/
      k  , y/k  }; }
  bool operator==(const vec2d o) const { return eq(x, o
      .x) && eq(y, o.y); }
  bool operator!=(const vec2d o) const { return !(*this
       == o); }
  bool operator<(const vec2d o) const { return lt(x, o.
      x) || eq(x, o.x) && lt(y, o.y); }

  uds len2() const { return x*x + y*y; }
  double len() const { return sqrt(len2()); }

  vec2d perp() { return {y, -x}; }
};
uds dot(vec2d a, vec2d b) { return a.x*b.x + a.y*b.y; }
uds cross(vec2d a, vec2d b) { return a.x*b.y - a.y*b.x;
     }
vec2d proj(vec2d u, vec2d v) { return u*dot(u, v)/dot(u
    , u); }
uds orient(vec2d a, vec2d b, vec2d c) { return cross(b-
    a, c-a); }
uds dist2(vec2d a, vec2d b) {return dot(a-b,a-b);}
double dist(vec2d a, vec2d b) {return sqrt(dist2(a,b));
     }
int orientation(vec2d a, vec2d b, vec2d c){return sign(
    orient(a,b,c));}
```

Line.h

16 lines

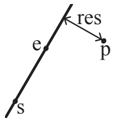
```
struct line {
  vec2d v; uds c; // Vector and offset (ax+by=c)
  line(vec2d v, uds c) : v(v), c(c) { div0(v.len2()); }
  line(uds a, uds b, uds c) : line({b, -a}, c) {}
  line(vec2d p, vec2d q) : line(q-p, cross(q-p, p)) {}

  line translate(vec2d t) { return {v, c+cross(v, t)};
      }
  line shiftPerp(uds d) { return {v, c + d*v.len()}; }
  bool cmpProj(vec2d p, vec2d q) { return dot(v,p) <
      dot(v,q); }
};

uds side(vec2d p, line l) { return cross(l.v, p)-l.c; }
uds dist2(vec2d p, line l) { uds s = side(p, l); return
     s*s/l.v.len2(); }
double dist(vec2d p, line l) { return abs(side(p, l))/l
    .v.len(); }
vec2d proj(vec2d p, line l) { return p + l.v.perp()  *
    side(p, l)/l.v.len2(); }
vec2d refl(vec2d p, line l) { return p + l.v.perp()*2*
    side(p, l)/l.v.len2(); }
```

lineDistance.h

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

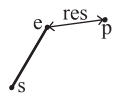


"Point.h" 4 lines

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

SegmentDistance.h

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



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

10 lines

```
double dist(segment s, vec2d p) {
  if (s.a == s.b) return (p-s.a).len();
  line l(s.a, s.b);
  return l.cmpProj(s.a, p) && l.cmpProj(p, s.b) ?
    dist(p, l) : min((p-s.a).len(), (p-s.b).len());
}
double dist(segment s, segment t) {
  vec2d i; if (properInter(s, t, i)) return 0;
  return min({dist(s, t.a), dist(s, t.b), dist(t, s.a),
       dist(t, s.b)});
}
```

Bisector.h

7 lines

```
bool bisector(line l1, line l2, bool interior, line &
    out) {
  if (cross(l1.v, l2.v) == 0) return false;
  int s = interior ? 1 : -1;
  out = {l2.v/l2.v.len() + l1.v/l1.v.len()*s,
       l2.c/l2.v.len() + l1.c/l1.v.len()*s};
  return true;
}
```

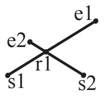
Segment.h

6 lines

```
struct segment {
  vec2d a, b;
  segment(vec2d a, vec2d b) : a(a), b(b) {}
};
bool inDisk(segment s, vec2d p) { return dot(s.a-p, s.b
    -p) <= 0; }
bool onSegment(segment s, vec2d p) { return eq(orient(s
    .a, s.b, p), 0) && inDisk(s, p); }
```

SegmentIntersection.h

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



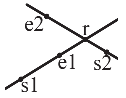
Usage: vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;

15 lines

```
bool properInter(segment s, segment t, vec2d &out) {
  uds oa = orient(t.a, t.b, s.a), ob = orient(t.a, t.b,
       s.b),
    oc = orient(s.a, s.b, t.a),od = orient(s.a, s.b, t.
       b);
  return lt(oa*ob, 0) && lt(oc*od, 0) ?
    out = (s.a*ob - s.b*oa) / (ob-oa), true : false;
}
set<vec2d> inter(segment s, segment t) {
  vec2d out; if (properInter(s, t, out)) return {out};
  set<vec2d> r;
  if (onSegment(t, s.a)) r.insert(s.a);
  if (onSegment(t, s.b)) r.insert(s.b);
  if (onSegment(s, t.a)) r.insert(t.a);
  if (onSegment(s, t.b)) r.insert(t.b);
  return r;
}
```

lineIntersection.h

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



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

"Point.h" 5 lines

```
bool inter(line l1, line l2, vec2d &out) {
  uds d = cross(l1.v, l2.v);
  if (eq(d, 0)) return false; // Parallel or equivalent
  return out = (l2.v*l1.c - l1.v*l2.c)/d, true;
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: bool left = sideOf(p1,p2,q)==1;

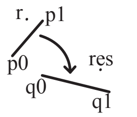
"Point.h" 9 lines

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

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

linearTransformation.h

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



"Point.h" 6 lines

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

PolarSort.h

Description: Given N points in the plane, it sorts the points looking angle with center at (0,0)
Time: $\mathcal{O}(N \log N)$

<Point.h> 16 lines

```
template <typename T>
void polarSort(vector<Point<T>> &v, Point<T> o, Point<T
    > u = {1, 0}) {
    sort(v.begin(), v.end(), [&](const Point<T> &a,
        const Point<T> &b) {
        Point<T> oa = a-o, ob = b-o;
        int xa = u.cross(oa), xb = u.cross(ob);
        if (xa == 0 && u.dot(oa) >= 0) return true; //
            angle(oa, u) = 0
        if (xb == 0 && u.dot(ob) >= 0) return false; //
             angle(ob, u) = 0
        if (xa*xb >= 0) { // Same side (up/down)
            int x = ob.cross(oa);
            if (x < 0) return true;
            else if (x > 0) return false;
            return oa.dist2() < ob.dist2();
        }
        return xa > 0;
    });
}
```

7.2 Circles

CircleIntersection.h

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

22 lines

```
//number of intersection points
int circleLine(vec2d o, double r, line l, pair<vec2d,
    vec2d> &out) {
  double h2 = r*r - dist2(o,l);
  if (ge(h2, 0)) { // the line touches the circle
    vec2d p = proj(o,l); // point P
    vec2d h = l.v*sqrt(h2)/l.v.len(); // vector
        parallel to l, of length h
    out = {p-h, p+h};
  }
  return 1 + sign(h2);
}
//number of intersection points
int circleCircle(vec2d o1, double r1, vec2d o2, double
    r2, pair<vec2d,vec2d> &out) {
  vec2d d=o2-o1; double d2=d.len2();
  if (d2 == 0) {assert(r1 != r2); return 0;} //
      concentric circles
  double pd = (d2 + r1*r1 - r2*r2)/2; // = |O_1P| * d
  double h2 = r1*r1 - pd*pd/d2; // = h^2
  if (h2 >= 0) {
    vec2d p = o1 + d*pd/d2, h = (d.perp())*sqrt(h2/d2);
    out = {p+h, p-h};
  }
  return 1 + sign(h2);
}
```

CircleTangents.h

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

11 lines

```
int tangents(vec2d o1, double r1, vec2d o2, double r2,
    bool inner, vector<pair<vec2d,vec2d>> &out) {
  if (inner) r2 = -r2;
  vec2d d = o2-o1;
  double dr = r1-r2, d2 = (d.len2()), h2 = d2-dr*dr;
  if (eq(d2, 0) || lt(h2, 0)) {div0(h2); return 0;}
  for (double sign : {-1,1}) {
    vec2d v = (d*dr + (d.perp())*sqrt(h2)*sign)/d2;
    out.push_back({o1 + v*r1, o2 + v*r2});
  }
  return 1 + (gt(h2, 0));
}
```

CirclePolygonIntersection.h

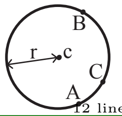
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

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

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

circumcircle.h

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



12 lines

```
vec2d circumCenter(vec2d a, vec2d b, vec2d c) {
  double d = 2 * cross(b - a, c - a);
  div0(d); // no circumcircle if A,B,C aligned
  vec2d ab = b - a;
  vec2d ac = c - a;

  double ab2 = ab.len2();
  double ac2 = ac.len2();

  vec2d num = (ab * ac2 - ac * ab2).perp(); // ojo
      cuidado con la implementacion del perp
  return a - num * (1.0 / d);
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

28 lines

```
// given n points, find the minimum enclosing circle of
    the points
// expected O(n)
pair<vec2d, double> minimum_enclosing_circle(vector<
    vec2d> &p) {
  random_shuffle(p.begin(), p.end());
  int n = p.size();
  // circle c(p[0], 0);
  pair<vec2d, uds> c = {p[0],0};
  for (int i = 1; i < n; i++) {
    if (sign(dist(c.first, p[i]) - c.second) > 0) {
      // c = circle(p[i], 0);
      c = {p[i], 0};
      for (int j = 0; j < i; j++) {
        if (sign(dist(c.first, p[j]) - c.second) > 0) {
          // c = circle((p[i] + p[j]) / 2, dist(p[i], p
              [j]) / 2);
          c = {(p[i] + p[j]) / 2, dist(p[i], p[j]) / 2}
          ;
          for (int k = 0; k < j; k++) {
            if (sign(dist(c.first, p[k]) - c.second) >
                0) {
              // c = circle(p[i], p[j], p[k]);
              vec2d circum = circumCenter(p[i], p[j], p
                  [k]);
              c = {circum, dist(p[i], circum)};
            }
          }
        }
      }
    }
  }
  return c;
}
```

Apolonium.h

17 lines

```
// returns the circle such that for all points w on the
     circumference of the circle
// dist(w, a) : dist(w, b) = rp : rq
// rp != rq
// https://en.wikipedia.org/wiki/Circles_of_Apollonius
pair<vec2d, double> apollonius(vec2d p, vec2d q, double
     rp, double rq){
  rq *= rq ;
  rp *= rp ;
  double a = rq - rp ;
  assert(sign(a));
  double g = rq * p.x - rp * q.x ; g /= a ;
  double h = rq * p.y - rp * q.y ; h /= a ;
  double c = rq * p.x * p.x - rp * q.x * q.x + rq * p.y
       * p.y - rp * q.y * q.y ;
  c /= a ;
  vec2d o(g, h);
  double r = g * g + h * h - c ;
  return {o,sqrt(r)};
}
```

7.3 Polygons

PointInPolygon.h

16 lines

```
// true if P at least as high as A
bool above(vec2d a, vec2d p) {return ge(p.y, a.y);}
// check if [PQ] crosses ray from A
bool crossesRay(vec2d a, vec2d p, vec2d q) {
  return gt((above(a,q) - above(a,p)) * orient(a,p,q),
      0);
}
// if strict, returns false when A is on the boundary
bool inPolygon(vector<vec2d> p, vec2d a, bool strict =
    true) {
  int numCrossings = 0;
  for (int i = 0, n = p.size(); i < n; i++) {
    segment s(p[i], p[(i+1)%n]);
    if (onSegment(s, a)) return !strict;
    numCrossings += crossesRay(a, p[i], p[(i+1)%n]);
  }
  return numCrossings & 1; // inside if odd number of
      crossings
}
```

IsConvex.h

10 lines

```
bool is_convex(vector<vec2d> &p) {
  bool s[3]; s[0] = s[1] = s[2] = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n, k = (j + 1) % n;
    s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
    if (s[0] && s[2]) return 0;
  }
  return 1;
}
```

MaxDistPolToPol.h

21 lines

```
// ROTATING CALIPERS (no es el calipers pero bueno,
    algo es algo)
// maximum distance from a convex polygon to another
    convex polygon
double maximum_dist_from_polygon_to_polygon(vector<
    vec2d> &u, vector<vec2d> &v){ //O(n)
  int n = (int)u.size(), m = (int)v.size();
  uds ans = 0;
  if (n < 3 || m < 3) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) ans = max(ans, dist2(
          u[i], v[j]));
    }
    return sqrt(ans);
  }
  if (gt(u[0].x, v[0].x)) swap(n, m), swap(u, v);
  int i = 0, j = 0, step = n + m + 10;
  while (j + 1 < m && v[j].x < v[j + 1].x) j++ ;
  while (step--) {
    if (ge(cross(u[(i + 1)%n] - u[i], v[(j + 1)%m] - v[
        j]), 0)) j = (j + 1) % m; // CALIPER
    else i = (i + 1) % n;
    ans = max(ans, dist2(u[i], v[j]));
  }
  return sqrt(ans);
}
```

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

InsidePolygon.h

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

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

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

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

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

PolygonArea.h

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

5 lines

```
uds areaPolygon(vector<vec2d> p) {
  uds area = 0.0;
  for (int i = 0, n = p.size(); i < n; i++) area +=
      cross(p[i], p[(i+1)%n]);
  return abs(area) / 2.0;
}
```

PolygonCenter.h

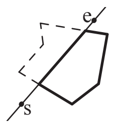
Description: Returns the center of mass for a polygon.

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

"Point.h" 9 lines

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

PolygonCut.h



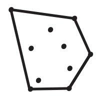
Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.

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

"Point.h" 13 lines

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

ConvexHull.h



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

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

20 lines

```
vector<vec2d> convex_hull(vector<vec2d> &p) {
  if (p.size() <= 1) return p;
  vector<vec2d> v = p, up, dn;
  sort(v.begin(), v.end());
  for (auto& p : v) {
    while (dn.size() > 1 && orientation(dn[dn.size() -
        2], dn.back(), p) < 0)
      dn.pop_back();

    while (up.size() > 1 && orientation(up[up.size() -
        2], up.back(), p) > 0)
      up.pop_back();
    up.push_back(p); dn.push_back(p);
  }
  v = dn;
  if (v.size() > 1) v.pop_back();
  reverse(up.begin(), up.end());
  up.pop_back();
  for (auto& p : up) v.push_back(p);
  if (v.size() == 2 && v[0] == v[1]) v.pop_back();
  return v;
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).
Time: $\mathcal{O}(n)$

"Point.h" 12 lines

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

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

18 lines

```
// -1 if strictly inside, 0 if on the polygon, 1 if
    strictly outside
// it must be strictly convex, otherwise make it
    strictly convex first
int in_convex(vector<vec2d> &p, const vec2d& x) { // O(
    log n)
  int n = p.size(); assert(n >= 3);
  int a = orientation(p[0], p[1], x), b = orientation(p
      [0], p[n - 1], x);
  if (a < 0 || b > 0) return 1;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
    int mid = l + r >> 1;
    if (orientation(p[0], p[mid], x) >= 0) l = mid;
    else r = mid;
  }
  int k = orientation(p[l], p[r], x);
  if (k <= 0) return -k;
  if (l == 1 && a == 0) return 0;
  if (r == n - 1 && b == 0) return 0;
  return -1;
}
```

LineHullIntersection.h

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

"Point.h" 39 lines

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

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)
        ) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

7.4 Misc

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(
        p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

SemiplaneInter.h

57 lines

```
struct HP {
    vec2d a, b;
    HP() {}
    HP(vec2d a, vec2d b) : a(a), b(b) {}
    HP(const HP& rhs) : a(rhs.a), b(rhs.b) {}
    int operator < (const HP& rhs) const {
        vec2d p = b - a;
        vec2d q = rhs.b - rhs.a;
        int fp = (lt(p.y,0) || (eq(p.y,0) && lt(p.x,0))
            );
        int fq = (lt(q.y,0) || (eq(q.y,0) && lt(q.x,0))
            );
        if (fp != fq) return fp == 0;
        if (cross(p, q)) return gt(cross(p, q),0);
        return lt(cross(p, rhs.b - a),0);
    }
    vec2d line_line_intersection(vec2d a, vec2d b,
        vec2d c, vec2d d) {
        b = b - a; d = c - d; c = c - a;
        return a + b * cross(c, d) / cross(b, d);
    }
    vec2d intersection(const HP &v) {
        return line_line_intersection(a, b, v.a, v.b);
    }
};
int check(HP a, HP b, HP c) {
    return cross(a.b - a.a, b.intersection(c) - a.a) >
        -EPS; //-eps to include polygons of zero area (
        straight lines, points)
}
// consider half-plane of counter-clockwise side of
    each line
// if lines are not bounded add infinity rectangle
// returns a convex polygon, a point can occur multiple
     times though
// complexity: O(n log(n))
vector<vec2d> half_plane_intersection(vector<HP> h) {
    sort(h.begin(), h.end());
    vector<HP> tmp;
    for (int i = 0; i < h.size(); i++) {
        if (!i || cross(h[i].b - h[i].a, h[i - 1].b - h
            [i - 1].a)) {
            tmp.push_back(h[i]);
        }
    }
    h = tmp;
    vector<HP> q(h.size() + 10);
    int qh = 0, qe = 0;
    for (int i = 0; i < h.size(); i++) {
        while (qe - qh > 1 && !check(h[i], q[qe - 2], q
            [qe - 1])) qe--;
        while (qe - qh > 1 && !check(h[i], q[qh], q[qh
            + 1])) qh++;
        q[qe++] = h[i];
    }
    while (qe - qh > 2 && !check(q[qh], q[qe - 2], q[qe
        - 1])) qe--;
    while (qe - qh > 2 && !check(q[qe - 1], q[qh], q[qh
        + 1])) qh++;
    vector<HP> res;
    for (int i = qh; i < qe; i++) res.push_back(q[i]);
    vector<vec2d> hull;
    if (res.size() > 2) {
        for (int i = 0; i < res.size(); i++) {
            hull.push_back(res[i].intersection(res[(i +
                1) % ((int)res.size())]));
        }
    }
    return hull;
}
```

Data structures (8)

OrderStatisticTree.h

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

<ext/pb_ds/assoc_container.hpp> 4 lines

```
using namespace __gnu_pbds;

template<typename T> using ordered_set = tree<T,
    null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
i = s.order_of_key(k); it = s.find_by_order(i);
```

BST.h

Description: Configurable BST To get a map, change null_type.
Time: $\mathcal{O}(\log N)$

27 lines

```
using namespace detail;

#define left get_l_child() #define right get_r_child()
#define value m_p_nd->m_value // Set: T Map: pair<K, V>
#define meta m_p_nd->get_metadata()#define valid m_p_nd
struct Meta { int sum; };
template<typename TIT, typename CTIT, typename C,
    typename A>
struct CustomUpdate { typedef Meta metadata_type;
  template<typename T> void operator()(T n, T null) {
    Meta &data = n.meta; data.sum = n.value;
    if (n.left.m_p_nd) data.sum += n.left.meta.sum;
    if (n.right.m_p_nd) data.sum += n.right.meta.sum;
  } template<typename T> int sum_lt(T n, int k) {
    if (!n.valid) return 0; int r = 0, v = n.value;
    if (v < k) r += v;
    if (n.left.valid && v < k) r += n.left.meta.sum;
    else       r += sum_lt(n.left, k);
    if (v < k) r += sum_lt(n.right, k);
    return r;
  }
  int sum_lt(int k) { return sum_lt(node_begin(), k); }
  virtual TIT node_begin() = 0;
  virtual CTIT node_begin() const = 0;
};
#undef left #undef right #undef value
#undef meta #undef valid
typedef tree<int, null_type, less<int>, rb_tree_tag,
    CustomUpdate> myset;
```

HashMap.h

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

7 lines

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

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.
Time: $\mathcal{O}(\log N)$

19 lines

```
using T = int;
const T id = 0;
T f(T a, T b) { return a+b; }
struct segment_tree {
  int n; vector<T> st;
  segment_tree(int n) : n(n), st(2*n, id) {}
  T query(int l, int r) { // query [l, r)
    T rl = id, rr = id;
    for (l += n, r += n; l <= r; l /= 2, r /= 2) {
      if (l%2 == 1) rl = f(rl, st[l++]);
      if (r%2 == 1) rr = f(st[--r], rr);
    }
    return f(rl, rr);
  }
  void set(int i, T x) {
    for (st[i += n] = x; i /= 2;)
      st[i] = f(st[i*2], st[i*2+1]);
  }
};
```

LazySegmentTreeVec.h

Description: SegmentTree with lazy propagation
Time: $\mathcal{O}(1)$

57 lines

```
using T = int;
const T fid = 0;
T f(T a, T b) { return a+b; }
T frep(T a, int cnt) { return a*cnt; }
const T gid = 0;
T g(T a, T b) { return b == 0 ? a : 1-a; }
// d(g(a, b, ...), x, c): Distribute g over f
// Ex: max(a+x, b+x, ...) = max(a, b, ...)+x
// Ex: sum(a+x, b+x, ...) = sum(a, b, ...)+x*c
T d(T v, T x, int cnt) { return x == 0 ? v : cnt-v; }
struct segment_tree {
    int n; vector<T> st, lz, lzs;

    void build(int i, int l, int r, const vector<T> &v)
        {
        if (l == r) { st[i] = v[l]; return; }
        int m = (l+r)/2;
        build(2*i, l, m, v), build(2*i+1, m+1, r, v);
        st[i] = f(st[2*i], st[2*i+1]);
    }
    segment_tree(const vector<T> &v) : n(v.size()), st
        (4*n), lz(4*n, gid), lzs(4*n, INF) { build(1,
        0, n-1, v); }

    void _set(int i, int l, int r, T x) { st[i] = frep(
        x, r-l+1), lzs[i] = x, lz[i] = gid; }
    void _apply(int i, int l, int r, T x) { st[i] = d(
        st[i], x, r-l+1), lz[i] = g(lz[i], x); }
    inline void push(int i, int l, int r) {
        int m = (l+r)/2;
        if (lzs[i] != INF) _set(2*i, l, m, lzs[i]),
            _set(2*i+1, m+1, r, lzs[i]);
        _apply(2*i, l, m, lz[i]), _apply(2*i+1, m+1, r,
            lz[i]);
        lzs[i] = INF, lz[i] = gid;
    }
    T query(int i, int l, int r, int ql, int qr) {
        if (ql <= l && r <= qr) return st[i];
        if (r < ql || qr < l) return fid;
        int m = (l+r)/2; push(i, l, r);
        return f(query(2*i, l, m, ql, qr), query(2*i+1,
            m+1, r, ql, qr));
    }
    void set(int i, int l, int r, int ql, int qr, T x)
        {
        if (ql <= l && r <= qr) { _set(i, l, r, x);
            return; }
        if (r < ql || qr < l) return;
        int m = (l+r)/2; push(i, l, r);
        set(2*i, l, m, ql, qr, x), set(2*i+1, m+1, r,
            ql, qr, x);
        st[i] = f(st[2*i], st[2*i+1]);
    }
    void apply(int i, int l, int r, int ql, int qr, T x
        ) {
        if (ql <= l && r <= qr) { _apply(i, l, r, x);
            return; }
        if (r < ql || qr < l) return;
        int m = (l+r)/2; push(i, l, r);
        apply(2*i, l, m, ql, qr, x), apply(2*i+1, m+1,
            r, ql, qr, x);
        st[i] = f(st[2*i], st[2*i+1]);
    }

    T query(int l, int r) { return query(1, 0, n-1, l,
        r); }
    void set(int l, int r, T x) { set(1, 0, n-1, l, r,
        x); }
    void apply(int l, int r, T x) { apply(1, 0, n-1, l,
        r, x); }
    T get(int i) { return query(i, i); }
    void apply(int i, T x) { apply(i, i, x); }
    void set(int i, T x) { set(i, i, x); }
};
```

UnionFindRollback.h

Description: Disjoint-set data structure with undo. If undo is not needed, skip st, time() and rollback().
Usage: `int t = uf.time(); ...; uf.rollback(t);`
Time: $\mathcal{O}(\log(N))$

14 lines

```
vi rt; vpii rb;
int root(int u) { return rt[u] < 0 ? u : r[u] = root(rt
    [u]); }
void join(int u, int v) {
    u = root(u), v = root(v);
    if (-rt[u] > -rt[v]) swap(u, v);
    if (u == v) return;
    rb.push_back({u, rt[u]});
    rt[v] += rt[u], rt[u] = v;
}
void rollback() { // Remove Path Compression
    auto [u, sz] = rb.back(); rb.pop_back();
    if (rt[u] < 0) return;
    rt[rt[u]] -= sz; rt[u] = sz;
}
```

SubMatrix.h

Description: Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).
Usage: `SubMatrix<int> m(matrix);`
`m.sum(0, 0, 2, 2); // top left 4 elements`
Time: $\mathcal{O}(N^2 + Q)$

13 lines

```
template<class T>
struct SubMatrix {
  vector<vector<T>> p;
  SubMatrix(vector<vector<T>>& v) {
    int R = sz(v), C = sz(v[0]);
    p.assign(R+1, vector<T>(C+1));
    rep(r,0,R) rep(c,0,C)
      p[r+1][c+1] = v[r][c] + p[r][c+1] + p[r+1][c] - p
          [r][c];
  }
  T sum(int u, int l, int d, int r) {
    return p[d][r] - p[d][l] - p[u][r] + p[u][l];
  }
};
```

Matrix.h

Description: Basic operations on square matrices.
Usage: `Matrix<int, 3> A;`
`A.d = {{{{1,2,3}}, {{4,5,6}}, {{7,8,9}}}};`
`array<int, 3> vec = {1,2,3};`
`vec = (A^N) * vec;`

26 lines

```
template<class T, int N> struct Matrix {
  typedef Matrix M;
  array<array<T, N>, N> d{};
  M operator*(const M& m) const {
    M a;
    rep(i,0,N) rep(j,0,N)
      rep(k,0,N) a.d[i][j] += d[i][k]*m.d[k][j];
    return a;
  }
  array<T, N> operator*(const array<T, N>& vec) const {
    array<T, N> ret{};
    rep(i,0,N) rep(j,0,N) ret[i] += d[i][j] * vec[j];
    return ret;
  }
  M operator^(ll p) const {
    assert(p >= 0);
    M a, b(*this);
    rep(i,0,N) a.d[i][i] = 1;
    while (p) {
      if (p&1) a = a*b;
      b = b*b;
      p >>= 1;
    }
    return a;
  }
};
```

LineContainer.h

Description: Container where you can add lines of the form kx+m, and query maximum values at points x. Useful for dynamic programming ("convex hull trick").
Time: $\mathcal{O}(\log N)$

30 lines

```
struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const { return k < o.k;
      }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf, 0;
    if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y =
        erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p)
      isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

Treap.h

Description: A short self-balancing tree. It acts as a sequential container with log-time splits/joins, and is easy to augment with additional data.
Time: $\mathcal{O}(\log N)$

53 lines

```
struct Node {
  Node *l = 0, *r = 0;
  int val, y, c = 1;
  Node(int val) : val(val), y(rand()) {}
  void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
  if (n) { each(n->l, f); f(n->val); each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
  if (!n) return {};
  if (cnt(n->l) >= k) { // "n->val >= k" for lower_bound(k)
    auto [L,R] = split(n->l, k);
    n->l = R;
    n->recalc();
    return {L, n};
  } else {
    auto [L,R] = split(n->r,k - cnt(n->l) - 1); // and just "k"
    n->r = L;
    n->recalc();
    return {n, R};
  }
}

Node* merge(Node* l, Node* r) {
  if (!l) return r;
  if (!r) return l;
  if (l->y > r->y) {
    l->r = merge(l->r, r);
    return l->recalc(), l;
  } else {
    r->l = merge(l, r->l);
    return r->recalc(), r;
  }
}

Node* ins(Node* t, Node* n, int pos) {
  auto [l,r] = split(t, pos);
  return merge(merge(l, n), r);
}

// Example application: move the range [l, r) to index k
void move(Node*& t, int l, int r, int k) {
  Node *a, *b, *c;
  tie(a,b) = split(t, l); tie(b,c) = split(b, r - l);
  if (k <= l) t = merge(ins(a, b, k), c);
  else t = merge(a, ins(c, b, k - r));
}
```

FenwickTree.h

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

22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

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" 22 lines

```
struct FT2 {
  vector<vi> ys; vector<FT> ft;
  FT2(int limx) : ys(limx) {}
  void fakeUpdate(int x, int y) {
    for (; x < sz(ys); x |= x + 1) ys[x].push_back(y);
  }
  void init() {
    for (vi& v : ys) sort(all(v)), ft.emplace_back(sz(v));
  }
  int ind(int x, int y) {
    return (int)(lower_bound(all(ys[x]), y) - ys[x].begin()); }
  void update(int x, int y, ll dif) {
    for (; x < sz(ys); x |= x + 1)
      ft[x].update(ind(x, y), dif);
  }
  ll query(int x, int y) {
    ll sum = 0;
    for (; x; x &= x - 1)
      sum += ft[x-1].query(ind(x-1, y));
    return sum;
  }
};
```

RMQ.h

Description: Range Minimum Queries on an array. Returns min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage: `RMQ rmq(values);`
`rmq.query(inclusive, exclusive);`
Time: $\mathcal{O}(|V| \log |V| + Q)$

16 lines

```
template<class T>
struct RMQ {
  vector<vector<T>> jmp;
  RMQ(const vector<T>& V) : jmp(1, V) {
    for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2, ++k) {
      jmp.emplace_back(sz(V) - pw * 2 + 1);
      rep(j,0,sz(jmp[k]))
        jmp[k][j] = min(jmp[k - 1][j], jmp[k - 1][j + pw]);
    }
  }
  T query(int a, int b) {
    assert(a < b); // or return inf if a == b
    int dep = 31 - __builtin_clz(b - a);
    return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
  }
};
```

MoQueries.h

Description: Answer interval or tree path queries by finding an approximate TSP through the queries, and moving from one query to the next by adding/removing points at the ends. If values are on tree edges, change `step` to add/remove the edge (a, c) and remove the initial `add` call (but keep `in`).
Time: $\mathcal{O}(N\sqrt{Q})$

49 lines

```
void add(int ind, int end) { ... } // add a[ind] (end = 0 or 1)
void del(int ind, int end) { ... } // remove a[ind]
int calc() { ... } // compute current answer

vi mo(vector<pii> Q) {
  int L = 0, R = 0, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s;
#define K(x) pii(x.first/blk, x.second ^ -(x.first/blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });
  for (int qi : s) {
    pii q = Q[qi];
    while (L > q.first) add(--L, 0);
    while (R < q.second) add(R++, 1);
    while (L < q.first) del(L++, 0);
    while (R > q.second) del(--R, 1);
    res[qi] = calc();
  }
  return res;
}
```

```
vi moTree(vector<array<int, 2>> Q, vector<vi>& ed, int
    root=0){
  int N = sz(ed), pos[2] = {}, blk = 350; // ~N/sqrt(Q)
  vi s(sz(Q)), res = s, I(N), L(N), R(N), in(N), par(N)
    ;
  add(0, 0), in[0] = 1;
  auto dfs = [&](int x, int p, int dep, auto& f) ->
      void {
    par[x] = p;
    L[x] = N;
    if (dep) I[x] = N++;
    for (int y : ed[x]) if (y != p) f(y, x, !dep, f);
    if (!dep) I[x] = N++;
    R[x] = N;
  };
  dfs(root, -1, 0, dfs);
#define K(x) pii(I[x[0]] / blk, I[x[1]] ^ -(I[x[0]] /
    blk & 1))
  iota(all(s), 0);
  sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[
      t]); });
  for (int qi : s) rep(end,0,2) {
    int &a = pos[end], b = Q[qi][end], i = 0;
#define step(c) { if (in[c]) { del(a, end); in[a] = 0;
    } \
                  else { add(c, end); in[c] = 1; } a =
                      c; }
    while (!(L[b] <= L[a] && R[a] <= R[b]))
      I[i++] = b, b = par[b];
    while (a != b) step(par[a]);
    while (i--) step(I[i]);
    if (end) res[qi] = calc();
  }
  return res;
}
```

FunctionQueue.h

Description: A queue where you can query any Commutative and Associative function

Time: $\mathcal{O}(1)$

23 lines

```
using T = int;
T f(T a, T b) { return min(a, b); }
struct QueueFn {
    vector<pair<T, T>> l, r;
    void dump() {
        if (!l.empty()) return;
        while (!r.empty()) {
            T v = r.back().first; r.pop_back();
            l.push_back({v, !l.empty() ? f(l.back().
                second, v) : v});
        }
    }
    T query() {
        if (l.empty()) return r.back().second;
        if (r.empty()) return l.back().second;
        return f(l.back().second, r.back().second);
    }
    void push(T v) { r.push_back({v, !r.empty() ? f(r.
        back().second, v) : v}); }
    void pop() { dump(); l.pop_back(); }
    T front() { dump(); return l.back().first; }
    T back() { return !r.empty() ? r.back().first : l.
        front().first; }
    int size() { return l.size()+r.size(); }
    bool empty() { return size() == 0; }
};
```

Graph (9)

9.1 Fundamentals

BellmanFord.h

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

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

5 lines

```
vi dist(n, INF); dist[i] = 0;
for (int k = 0; k < n; k++)
  for (int u = 0; u < n; u++)
    for (auto [v, vc] : e[u])
      dist[v] = min(dist[v], dist[u]+vc);
```

FloydWarshall.h

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

Time: $\mathcal{O}(N^3)$

4 lines

```
for (int k = 0; k < n; k++)
  for (int u = 0; u < n; u++)
    for (int v = 0; v < n; v++)
      am[u][v] = min(am[u][v], am[u][k]+am[k][v]);
```

BridgesAndArticulationPoints.h

Description: Bridges and Articulation Points

Usage: `Bridges and Articulation Points`

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

22 lines

```
vi parent, tin, low; int dfst;
void dfs(int u) {
  low[u] = tin[u] = dfst++;
  int cut = false, children = 0;
  for (auto v : e[u]) {
    if (parent[v] == -1) {
      parent[v] = u, children++;
      dfs(v);
      low[u] = min(low[u], low[v]);
      if (tin[u] <= low[v]) cut = true;
      if (tin[u] <  low[v])
        cout << u << " " << v << " Cut Edge\n";
    }else if (v != parent[u])
      low[u] = min(low[u], tin[v]);
  }
  if (parent[u] == u) cut = children > 1;
  if (cut) cout << u << " Cut Vertex\n";
}
tin.assign(n, -1), low.assign(n, -1);
parent.assign(n, -1), dfst = 0;
for (int u = 0; u < n; u++)
  if (parent[u] == -1) parent[u] = u, dfs(u);
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.

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

8 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), q;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push_back(i);
  rep(j,0,sz(q)) for (int x : gr[q[j]])
    if (--indeg[x] == 0) q.push_back(x);
  return q;
}
```

9.2 Network flow

PushRelabel.h

Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.

Time: $\mathcal{O}\left(V^2\sqrt{E}\right)$

48 lines

```
struct PushRelabel {
  struct Edge {
    int dest, back;
    ll f, c;
  };
  vector<vector<Edge>> g;
  vector<ll> ec;
  vector<Edge*> cur;
  vector<vi> hs; vi H;
  PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(
      n) {}

  void addEdge(int s, int t, ll cap, ll rcap=0) {
    if (s == t) return;
    g[s].push_back({t, sz(g[t]), 0, cap});
    g[t].push_back({s, sz(g[s])-1, 0, rcap});
  }

  void addFlow(Edge& e, ll f) {
    Edge &back = g[e.dest][e.back];
    if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.
        dest);
    e.f += f; e.c -= f; ec[e.dest] += f;
    back.f -= f; back.c += f; ec[back.dest] -= f;
  }
  ll calc(int s, int t) {
    int v = sz(g); H[s] = v; ec[t] = 1;
    vi co(2*v); co[0] = v-1;
    rep(i,0,v) cur[i] = g[i].data();
    for (Edge& e : g[s]) addFlow(e, e.c);

    for (int hi = 0;;) {
      while (hs[hi].empty()) if (!hi--) return -ec[s];
      int u = hs[hi].back(); hs[hi].pop_back();
      while (ec[u] > 0)  // discharge u
        if (cur[u] == g[u].data() + sz(g[u])) {
          H[u] = 1e9;
          for (Edge& e : g[u]) if (e.c && H[u] > H[e.
              dest]+1)
            H[u] = H[e.dest]+1, cur[u] = &e;
          if (++co[H[u]], !--co[hi] && hi < v)
            rep(i,0,v) if (hi < H[i] && H[i] < v)
              --co[H[i]], H[i] = v + 1;
          hi = H[u];
        } else if (cur[u]->c && H[u] == H[cur[u]->dest
            ]+1)
          addFlow(*cur[u], min(ec[u], cur[u]->c));
        else ++cur[u];
    }
  }
  bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h

Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}(FE\log(V))$ where F is max flow. $\mathcal{O}(VE)$ for setpi.

79 lines

```
#include <bits/extc++.h>

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
  struct edge {
    int from, to, rev;
    ll cap, cost, flow;
  };
  int N;
  vector<vector<edge>> ed;
  vi seen;
  vector<ll> dist, pi;
  vector<edge*> par;

  MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N),
      par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,
        cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-
        cost,0 });
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (edge& e : ed[s]) if (!seen[e.to]) {
        ll val = di - pi[e.to] + e.cost;
        if (e.cap - e.flow > 0 && val < dist[e.to]) {
          dist[e.to] = val;
          par[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({ -dist[e.to], e.to });
          else
            q.modify(its[e.to], { -dist[e.to], e.to });
        }
      }
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (edge* x = par[t]; x; x = par[x->from])
        fl = min(fl, x->cap - x->flow);

      totflow += fl;
      for (edge* x = par[t]; x; x = par[x->from]) {
        x->flow += fl;
        ed[x->to][x->rev].flow -= fl;
      }
    }
    rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost *
        e.flow;
    return {totflow, totcost/2};
  }

  // If some costs can be negative, call this before
      maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (edge& e : ed[i]) if (e.cap)
          if ((v = pi[i] + e.cost) < pi[e.to])
            pi[e.to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

MinCostMaxFlowDalo.h

Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}(FE\log(V))$ where F is max flow. $\mathcal{O}(VE)$ for setpi.

25 lines

```
#include <bits/extc++.h>

int mf = 0, mc = 0;
while (true) {
  queue<int> q; vi dist(n, INF), parent(n, -1), f(n,
      INF), inq(n, false), cnt(n, 0);
  dist[s] = 0, parent[s] = s, f[s] = INF, inq[s] = true
      , q.push(s);
  while (!q.empty()) {
    int u = q.front(); q.pop(); inq[u] = false;
    for (auto v : e[u]) {
      if (am[u][v] == 0) continue;
      int vw = dist[u]+amw[u][v];
      if (vw < dist[v]) {
        dist[v] = vw, parent[v] = u, f[v] = min(f[u],
            am[u][v]);
        if (cnt[v]++ == n) throw runtime_error("
            Negative cycle");
        if (!inq[v]) inq[v] = true, q.push(v);
      }
    }
  }
  if (dist[t] == INF) break;

  int u = parent[t], v = t;
  while (u != v)
    am[u][v] -= f[t], am[v][u] += f[t], v = u, u =
        parent[u];
  mf += f[t], mc += dist[t]*f[t];
}
```

EdmondsKarp.h

Description: Flow algorithm with guaranteed complexity $O(VE^2)$. To get edge flow values, compare capacities before and after, and take the positive values only.

20 lines

```
int r = 0;
while (true) {
  queue<int> q; vi par(n, -1), cost(n, INF);
  q.push(s), par[s] = s;
  while (!q.empty()) {
    int u = q.front(); q.pop();
    for (auto v : e[u]) {
      if (am[u][v] > 0 && par[v] == -1) {
        par[v] = u, cost[v] = min(cost[u], am[u][v]);
        if (v == t) break;
        q.push(v);
      }
    }
  }
  if (par[t] == -1) break;
  int u = par[t], v = t;
  while (u != v)
    am[u][v] -= cost[t], am[v][u] += cost[t], v = u, u
        = par[u];
  r += cost[t];
}
```

Dinic.h

Description: Flow algorithm with complexity $O(VE\log U)$ where $U = \max|\text{cap}|$. $O(\min(E^{1/2}, V^{2/3})E)$ if $U = 1$; $O(\sqrt{V}E)$ for bipartite matching.

32 lines

```
vi lev, ei;
int dfs(int u, int c) {
  if (u == t) return c;
  for (int &i = ei[u]; i < e[u].size(); i++) {
    int v = e[u][i];
    if (lev[v] <= lev[u] || am[u][v] == 0) continue;
    if (int mc = dfs(v, min(c, am[u][v]))) {
      am[u][v] -= mc, am[v][u] += mc;
      if (am[u][v] == 0) i++;
      return mc;
    }
  }
  return 0;
}
int r = 0;
while (true) {
  queue<int> q; lev.assign(n, -1);
  lev[s] = 0; q.push(s);
  while (!q.empty()) {
    int u = q.front(); q.pop();
    for (auto v : e[u]) {
      if (lev[v] == -1 && am[u][v] > 0) {
        lev[v] = lev[u]+1;
        if (v == t) goto endbfs;
        q.push(v);
      }
    }
  } endbfs:
  if (lev[t] == -1) break;
  ei.assign(n, 0);
  while (int f = dfs(s, INF)) r += f;
}
```

MinCut.h

Description: After running max-flow, the left side of a min-cut from s to t is given by all vertices reachable from s, only traversing edges with positive residual capacity.

GlobalMinCut.h

Description: Find a global minimum cut in an undirected graph, as represented by an adjacency matrix.
Time: $\mathcal{O}(V^3)$

21 lines

```
pair<int, vi> globalMinCut(vector<vi> mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i,0,n) co[i] = {i};
  rep(ph,1,n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it,0,n-ph) { // O(V^2) -> O(E log V) with prio.
        queue
      w[t] = INT_MIN;
      s = t, t = max_element(all(w)) - w.begin();
      rep(i,0,n) w[i] += mat[t][i];
    }
    best = min(best, {w[t] - mat[t][t], co[t]});
    co[s].insert(co[s].end(), all(co[t]));
    rep(i,0,n) mat[s][i] += mat[t][i];
    rep(i,0,n) mat[i][s] = mat[s][i];
    mat[0][t] = INT_MIN;
  }
  return best;
}
```

GomoryHu.h

Description: Given a list of edges representing an undirected flow graph, returns edges of the Gomory-Hu tree. The max flow between any pair of vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time: $\mathcal{O}(V)$ Flow Computations

"PushRelabel.h" 13 lines

```
typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
  vector<Edge> tree;
  vi par(N);
  rep(i,1,N) {
    PushRelabel D(N); // Dinic also works
    for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2])
        ;
    tree.push_back({i, par[i], D.calc(i, par[i])});
    rep(j,i+1,N)
      if (par[j] == par[i] && D.leftOfMinCut(j)) par[j]
          = i;
  }
  return tree;
}
```

9.3 Matching

hopcroftKarp.h

Description: Fast bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); hopcroftKarp(g, btoa);`
Time: $\mathcal{O}\left(\sqrt{V}E\right)$

42 lines

```
bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A,
    vi& B) {
  if (A[a] != L) return 0;
  A[a] = -1;
  for (int b : g[a]) if (B[b] == L + 1) {
    B[b] = 0;
    if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A
        , B))
      return btoa[b] = a, 1;
  }
  return 0;
}

int hopcroftKarp(vector<vi>& g, vi& btoa) {
  int res = 0;
  vi A(g.size()), B(btoa.size()), cur, next;
  for (;;) {
    fill(all(A), 0);
    fill(all(B), 0);
    cur.clear();
    for (int a : btoa) if(a != -1) A[a] = -1;
    rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
    for (int lay = 1;; lay++) {
      bool islast = 0;
      next.clear();
      for (int a : cur) for (int b : g[a]) {
        if (btoa[b] == -1) {
          B[b] = lay;
          islast = 1;
        }
        else if (btoa[b] != a && !B[b]) {
          B[b] = lay;
          next.push_back(btoa[b]);
        }
      }
      if (islast) break;
      if (next.empty()) return res;
      for (int a : next) A[a] = lay;
      cur.swap(next);
    }
    rep(a,0,sz(g))
      res += dfs(a, 0, g, btoa, A, B);
  }
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph *g* should be a list of neighbors of the left partition, and *btoa* should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. *btoa*[*i*] will be the match for vertex *i* on the right side, or -1 if it's not matched.
Usage: `vi btoa(m, -1); dfsMatching(g, btoa);`
Time: $\mathcal{O}(VE)$

22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1; int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis)) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
  vi vis;
  rep(i,0,sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m);
  for (int it : match) if (it != -1) lfound[it] = false
      ;
  vi q, cover;
  rep(i,0,n) if (lfound[i]) q.push_back(i);
  while (!q.empty()) {
    int i = q.back(); q.pop_back();
    lfound[i] = 1;
    for (int e : g[i]) if (!seen[e] && match[e] != -1)
        {
      seen[e] = true;
      q.push_back(match[e]);
    }
  }
  rep(i,0,n) if (!lfound[i]) cover.push_back(i);
  rep(i,0,m) if (seen[i]) cover.push_back(n+i);
  assert(sz(cover) == res);
  return cover;
}
```

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes cost[N][M], where cost[i][j] = cost for L[i] to be matched with R[j] and returns (min cost, match), where L[i] is matched with R[match[i]]. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}(N^2M)$

31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
  if (a.empty()) return {0, {}};
  int n = sz(a) + 1, m = sz(a[0]) + 1;
  vi u(n), v(m), p(m), ans(n - 1);
  rep(i,1,n) {
    p[0] = i;
    int j0 = 0; // add "dummy" worker 0
    vi dist(m, INT_MAX), pre(m, -1);
    vector<bool> done(m + 1);
    do { // dijkstra
      done[j0] = true;
      int i0 = p[j0], j1, delta = INT_MAX;
      rep(j,1,m) if (!done[j]) {
        auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
        if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
        if (dist[j] < delta) delta = dist[j], j1 = j;
      }
      rep(j,0,m) {
        if (done[j]) u[p[j]] += delta, v[j] -= delta;
        else dist[j] -= delta;
      }
      j0 = j1;
    } while (p[j0]);
    while (j0) { // update alternating path
      int j1 = pre[j0];
      p[j0] = p[j1], j0 = j1;
    }
  }
  rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
  return {-v[0], ans}; // min cost
}
```

GeneralMatching.h

Description: Matching for general graphs.
Time: $\mathcal{O}(N^3)$

52 lines

```
vector<int> Blossom(vector<vector<int>>& graph) {
  int n = graph.size(), timer = -1;
  vector<int> mate(n, -1), label(n), parent(n),
              orig(n), aux(n, -1), q;
  auto lca = [&](int x, int y) {
    for (timer++; ; swap(x, y)) {
      if (x == -1) continue;
      if (aux[x] == timer) return x;
      aux[x] = timer;
      x = (mate[x] == -1 ? -1 : orig[parent[mate[x]]]);
    }
  };
  auto blossom = [&](int v, int w, int a) {
    while (orig[v] != a) {
      parent[v] = w; w = mate[v];
      if (label[w] == 1) label[w] = 0, q.push_back(w);
      orig[v] = orig[w] = a; v = parent[w];
    }
  };
  auto augment = [&](int v) {
    while (v != -1) {
      int pv = parent[v], nv = mate[pv];
      mate[v] = pv; mate[pv] = v; v = nv;
    }
  };
  auto bfs = [&](int root) {
    fill(label.begin(), label.end(), -1);
    iota(orig.begin(), orig.end(), 0);
    q.clear();
    label[root] = 0; q.push_back(root);
    for (int i = 0; i < (int)q.size(); ++i) {
      int v = q[i];
      for (auto x : graph[v]) {
        if (label[x] == -1) {
          label[x] = 1; parent[x] = v;
          if (mate[x] == -1)
            return augment(x), 1;
          label[mate[x]] = 0; q.push_back(mate[x]);
        } else if (label[x] == 0 && orig[v] != orig[x])
            {
          int a = lca(orig[v], orig[x]);
          blossom(x, v, a); blossom(v, x, a);
        }
      }
    }
    return 0;
  };
  // Time halves if you start with (any) maximal
      matching.
  for (int i = 0; i < n; i++)
    if (mate[i] == -1)
      bfs(i);
  return mate;
}
```

9.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.
Usage: `scc(graph, [&](vi& v) { ... }) visits all components`
`in reverse topological order. comp[i] holds the component`
`index of a node (a component only has edges to components with`
`lower index). ncomps will contain the number of components.`
Time: $\mathcal{O}(E+V)$

16 lines

```
vi low, tin, scc; int dfst = 0, scci = 0;
```

```
void dfs(int u) {
  low[u] = tin[u] = dfst++;
  scc.push_back(u);
  for (auto v : e[u]) {
    if (tin[v] == -1) dfs(v);
    if (scc[v] == -1) low[u] = min(low[u], low[v]);
  }
  if (low[u] == tin[u]) {
    while (scc.back() != u)
      scc[scc.back()] = scci, scc.pop_back();
    scc[scc.back()] = scci++, scc.pop_back();
  }
}
tin.assign(n, -1), low.resize(n), dfst = 0, scci = 0;
dfs(s);
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, 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.
Usage: `int eid = 0; ed.resize(N);`
`for each edge (a,b) {`
`ed[a].emplace_back(b, eid);`
`ed[b].emplace_back(a, eid++); }`
`bicomps([&](const vi& edgelist) {...});`
Time: $\mathcal{O}(E+V)$

32 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, top = me;
  for (auto [y, e] : ed[at]) if (e != par) {
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
      else { /* e is a bridge */ }
    }
  }
  return top;
}

template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).
Usage: `TwoSat ts(number of boolean variables);`
`ts.either(0, ~3); // Var 0 is true or var 3 is false`
`ts.setValue(2); // Var 2 is true`
`ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true`
`ts.solve(); // Returns true iff it is solvable`
`ts.values[0..N-1] holds the assigned values to the vars`
Time: $\mathcal{O}(N+E)$, where N is the number of boolean variables, and E is the number of clauses.

56 lines

```
struct TwoSat {
  int N;
  vector<vi> gr;
  vi values; // 0 = false, 1 = true

  TwoSat(int n = 0) : N(n), gr(2*n) {}

  int addVar() { // (optional)
    gr.emplace_back();
    gr.emplace_back();
    return N++;
  }

  void either(int f, int j) {
    f = max(2*f, -1-2*f);
    j = max(2*j, -1-2*j);
    gr[f].push_back(j^1);
    gr[j].push_back(f^1);
  }
  void setValue(int x) { either(x, x); }

  void atMostOne(const vi& li) { // (optional)
    if (sz(li) <= 1) return;
    int cur = ~li[0];
    rep(i,2,sz(li)) {
      int next = addVar();
      either(cur, ~li[i]);
      either(cur, next);
      either(~li[i], next);
      cur = ~next;
    }
    either(cur, ~li[1]);
  }

  vi val, comp, z; int time = 0;
  int dfs(int i) {
    int low = val[i] = ++time, x; z.push_back(i);
    for(int e : gr[i]) if (!comp[e])
      low = min(low, val[e] ?: dfs(e));
    if (low == val[i]) do {
      x = z.back(); z.pop_back();
      comp[x] = low;
      if (values[x>>1] == -1)
        values[x>>1] = x&1;
    } while (x != i);
    return val[i] = low;
  }

  bool solve() {
    values.assign(N, -1);
    val.assign(2*N, 0); comp = val;
    rep(i,0,2*N) if (!comp[i]) dfs(i);
    rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
    return 1;
  }
};
```

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.
Time: $\mathcal{O}(V+E)$

15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int
    src=0) {
  int n = sz(gr);
  vi D(n), its(n), eu(nedges), ret, s = {src};
  D[src]++; // to allow Euler paths, not just cycles
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = sz(gr[x
        ]);
    if (it == end){ ret.push_back(x); s.pop_back();
        continue; }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      eu[e] = 1; s.push_back(y);
    }}
  for (int x : D) if (x < 0 || sz(ret) != nedges+1)
      return {};
  return {ret.rbegin(), ret.rend()};
}
```

EulerianCycle.h

Description: Hierholzer
Time: $\mathcal{O}(m)$

9 lines

```
vi used, r;
void dfs(int u) {
    while (!e[u].empty()) {
        auto [v, i] = e[u].back(); e[u].pop_back();
        if (used[i]) continue; used[i] = true;
        dfs(v);
    }
    r.push_back(u);
}
```

9.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D, computes a $(D+1)$-coloring of the edges such that no neighboring edges share a color. (D-coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)
Time: $\mathcal{O}(NM)$

31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
  vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
  for (pii e : eds) ++cc[e.first], ++cc[e.second];
  int u, v, ncols = *max_element(all(cc)) + 1;
  vector<vi> adj(N, vi(ncols, -1));
  for (pii e : eds) {
    tie(u, v) = e;
    fan[0] = v;
    loc.assign(ncols, 0);
    int at = u, end = u, d, c = free[u], ind = 0, i =
       0;
    while (d = free[v], !loc[d] && (v = adj[u][d]) !=
       -1)
      loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
    cc[loc[d]] = c;
    for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at
       ][cd])
      swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
    while (adj[fan[i]][d] != -1) {
      int left = fan[i], right = fan[++i], e = cc[i];
      adj[u][e] = left;
      adj[left][e] = u;
      adj[right][e] = -1;
      free[right] = e;
    }
    adj[u][d] = fan[i];
    adj[fan[i]][d] = u;
    for (int y : {fan[0], u, end})
      for (int& z = free[y] = 0; adj[y][z] != -1; z++);
  }
  rep(i,0,sz(eds))
    for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++
       ret[i];
  return ret;
}
```

9.6 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.
Time: $\mathcal{O}(3^{n/3})$, much faster for sparse graphs

12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B
     R={}) {
  if (!P.any()) { if (!X.any()) f(R); return; }
  auto q = (P | X)._Find_first();
  auto cands = P & ~eds[q];
  rep(i,0,sz(eds)) if (cands[i]) {
    R[i] = 1;
    cliques(eds, f, P & eds[i], X & eds[i], R);
    R[i] = P[i] = 0; X[i] = 1;
  }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.
       i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d;
        });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i]) T.push_back
          ({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q)
            + 1, 1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)),
      old(S) {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertexCover.

9.7 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

15 lines

```
int ln = log2(n)+2;
for (int u = 0; u < n; u++) up[u][0] = parent[u];
for (int i = 1; i < ln; i++)
    for (int u = 0; u < n; u++)
        up[u][i] = up[up[u][i-1]][i-1];

if (depth[u] < depth[v]) swap(u, v);
for (int i = ln-1; depth[u] > depth[v]; i--)
    if (depth[up[u][i]] >= depth[v])
        u = up[u][i];
if (u == v) return;
for (int i = ln-1; i >= 0; i--)
    if (up[u][i] != up[v][i])
        u = up[u][i], v = up[v][i];
u = up[u][0], v = up[v][0];
```

LCA.h

Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time: $\mathcal{O}(N \log N + Q)$

"../data-structures/RMQ.h" 21 lines

```
struct LCA {
  int T = 0;
  vi time, path, ret;
  RMQ<int> rmq;

  LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1),
      ret)) {}
  void dfs(vector<vi>& C, int v, int par) {
    time[v] = T++;
    for (int y : C[v]) if (y != par) {
      path.push_back(v), ret.push_back(time[v]);
      dfs(C, y, v);
    }
  }

  int lca(int a, int b) {
    if (a == b) return a;
    tie(a, b) = minmax(time[a], time[b]);
    return path[rmq.query(a, b)];
  }
  //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(
      a,b)];}
};
```

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 a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.
Time: $\mathcal{O}(|S|\log|S|)$

"LCA.h" 21 lines

```
typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
  static vi rev; rev.resize(sz(lca.time));
  vi li = subset, &T = lca.time;
  auto cmp = [&](int a, int b) { return T[a] < T[b]; };
  sort(all(li), cmp);
  int m = sz(li)-1;
  rep(i,0,m) {
    int a = li[i], b = li[i+1];
    li.push_back(lca.lca(a, b));
  }
  sort(all(li), cmp);
  li.erase(unique(all(li)), li.end());
  rep(i,0,sz(li)) rev[li[i]] = i;
  vpi ret = {pii(0, li[0])};
  rep(i,0,sz(li)-1) {
    int a = li[i], b = li[i+1];
    ret.emplace_back(rev[lca.lca(a, b)], b);
  }
  return ret;
}
```

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. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.
Time: $\mathcal{O}((\log N)^2)$

"../data-structures/LazySegmentTree.h" 46 lines

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, rt, pos;
  Node *tree;
  HLD(vector<vi> adj_)
    : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
      rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0);
          dfsHld(0); }
  void dfsSz(int v) {
    for (int& u : adj[v]) {
      adj[u].erase(find(all(adj[u]), v));
      par[u] = v;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfsHld(u);
    }
  }
  template <class B> void process(int u, int v, B op) {
    for (;; v = par[rt[v]]) {
      if (pos[u] > pos[v]) swap(u, v);
      if (rt[u] == rt[v]) break;
      op(pos[rt[v]], pos[v] + 1);
    }
    op(pos[u] + VALS_EDGES, pos[v] + 1);
  }
  void modifyPath(int u, int v, int val) {
    process(u, v, [&](int l, int r) { tree->add(l, r,
        val); });
  }
  int queryPath(int u, int v) { // Modify depending on
      problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
        res = max(res, tree->query(l, r));
    });
    return res;
  }
  int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES, pos[v] +
        siz[v]);
  }
};
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.
Time: All operations take amortized $\mathcal{O}(\log N)$.

90 lines

```
struct Node { // Splay tree. Root's pp contains tree's
    parent.
  Node *p = 0, *pp = 0, *c[2];
  bool flip = 0;
  Node() { c[0] = c[1] = 0; fix(); }
  void fix() {
    if (c[0]) c[0]->p = this;
    if (c[1]) c[1]->p = this;
    // (+ update sum of subtree elements etc. if wanted
        )
  }
  void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
  }
  int up() { return p ? p->c[1] == this : -1; }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ?
        y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
      x->c[h] = y->c[h ^ 1];
      y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
  }
  void splay() {
    for (pushFlip(); p; ) {
      if (p->p) p->p->pushFlip();
      p->pushFlip(); pushFlip();
      int c1 = up(), c2 = p->up();
      if (c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
  }
};

struct LinkCut {
  vector<Node> node;
  LinkCut(int N) : node(N) {}

  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v));
    makeRoot(&node[u]);
    node[u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top); x->splay();
    assert(top == (x->pp ?: x->c[0]));
    if (x->pp) x->pp = 0;
    else {
      x->c[0] = top->p = 0;
      x->fix();
    }
  }
  bool connected(int u, int v) { // are u, v in the
      same tree?
    Node* nu = access(&node[u])->first();
    return nu == access(&node[v])->first();
  }
  void makeRoot(Node* u) {
    access(u);
    u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0;
      u->fix();
    }
  }
  Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(); u->pp = 0;
      if (pp->c[1]) {
        pp->c[1]->p = 0; pp->c[1]->pp = pp; }
      pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
  }
};
```

DirectedMST.h

Description: Finds a minimum spanning tree/arborescence of a directed graph, given a root node. If no MST exists, returns -1.
Time: $\mathcal{O}(E \log V)$

"../data-structures/UnionFindRollback.h" 60 lines

```
struct Edge { int a, b; ll w; };
struct Node {
  Edge key;
  Node *l, *r;
  ll delta;
  void prop() {
    key.w += delta;
    if (l) l->delta += delta;
    if (r) r->delta += delta;
    delta = 0;
  }
  Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
  if (!a || !b) return a ?: b;
  a->prop(), b->prop();
  if (a->key.w > b->key.w) swap(a, b);
  swap(a->l, (a->r = merge(b, a->r)));
  return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r);
    }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
  RollbackUF uf(n);
  vector<Node*> heap(n);
  for (Edge e : g) heap[e.b] = merge(heap[e.b], new
      Node{e});
  ll res = 0;
  vi seen(n, -1), path(n), par(n);
  seen[r] = r;
  vector<Edge> Q(n), in(n, {-1,-1}), comp;
  deque<tuple<int, int, vector<Edge>>> cycs;
  rep(s,0,n) {
    int u = s, qi = 0, w;
    while (seen[u] < 0) {
      if (!heap[u]) return {-1,{}};
      Edge e = heap[u]->top();
      heap[u]->delta -= e.w, pop(heap[u]);
      Q[qi] = e, path[qi++] = u, seen[u] = s;
      res += e.w, u = uf.find(e.a);
      if (seen[u] == s) {
        Node* cyc = 0;
        int end = qi, time = uf.time();
        do cyc = merge(cyc, heap[w = path[--qi]]);
        while (uf.join(u, w));
        u = uf.find(u), heap[u] = cyc, seen[u] = -1;
        cycs.push_front({u, time, {&Q[qi], &Q[end]}});
      }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
  }

  for (auto& [u,t,comp] : cycs) { // restore sol (
      optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
  }
  rep(i,0,n) par[i] = in[i].a;
  return {res, par};
}
```

9.8 Other

DynamicConnectivity.h

Description: Dynamic connectivity
Time: $\mathcal{O}(tlog^2 n)$

20 lines

```
using E = pair<int, int>; // DSU
int T, t = 0; VV<E> st;
void _add(E e) { join(e.first, e.second); }
void _remove(E e) { rollback(); }
void add_edge(E v, int l, int r) {
  for (l += T, r += T; l <= r; l /= 2, r /= 2) {
    if (l%2 == 1) st[l].push_back(v), l++;
    if (r%2 == 0) st[r].push_back(v), r--;
  }
}
void _add_all(int t) { for (auto e : st[t]) _add(e); }
void _remove_all(int t) { for (auto e : st[t]) _remove(
    e); }
void set(int i) {
  int b = 1<<log2(i += T);
  while ((t&b) == (i&b)) b /= 2;
  for (int m = b; m; m /= 2) _remove_all(t), t /= 2;
  for (; b; b /= 2) _add_all(t = i&b ? 2*t+1 : 2*t);
}
void next() { set(t-T+1); }
T = 1<<(log2(t-1)+1), st.resize(2*T), rt.assign(n, -1);
```

DeBrujin.h

Description: Confio en Tete Number of sequences: $\frac{k!^{k^{n-1}}}{k^n}$
Time: $\mathcal{O}(k^n)$

14 lines

```
vector<int> de_bruijn(int n, int k) {
  int N = pow(k, n-1); vi ui(N, 0);
  vector<int> res; res.reserve(N*k+n-1);
  stack<pii> s; s.push({0, -1});
  while (!s.empty()) {
    auto& [u, i] = s.top();
    if (i != -1) res.push_back(i);
    i = ui[u]++;
    if (i < k) s.push({(u*k+i)%N, -1});
    else s.pop();
  };
  for (int i = 0; i < n-1; i++) res.push_back(0);
  reverse(res.begin(), res.end()); return res;
}
```

9.9 Math

9.9.1 Number of Spanning Trees

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

9.9.2 Erdős–Gallai theorem

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

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

Various (10)

10.1 Intervals

IntervalContainer.h

Description: Add and remove intervals from a set of disjoint intervals. Will merge the added interval with any overlapping intervals in the set when adding. Intervals are [inclusive, exclusive).
Time: $\mathcal{O}(\log N)$

23 lines

```
set<pii>::iterator addInterval(set<pii>& is, int L, int
     R) {
  if (L == R) return is.end();
  auto it = is.lower_bound({L, R}), before = it;
  while (it != is.end() && it->first <= R) {
    R = max(R, it->second);
    before = it = is.erase(it);
  }
  if (it != is.begin() && (--it)->second >= L) {
    L = min(L, it->first);
    R = max(R, it->second);
    is.erase(it);
  }
  return is.insert(before, {L,R});
}

void removeInterval(set<pii>& is, int L, int R) {
  if (L == R) return;
  auto it = addInterval(is, L, R);
  auto r2 = it->second;
  if (it->first == L) is.erase(it);
  else (int&)it->second = L;
  if (R != r2) is.emplace(R, r2);
}
```

IntervalCover.h

Description: Compute indices of smallest set of intervals covering another interval. Intervals should be [inclusive, exclusive). To support [inclusive, inclusive], change (A) to add `|| R.empty()`. Returns empty set on failure (or if G is empty).
Time: $\mathcal{O}(N \log N)$

19 lines

```
template<class T>
vi cover(pair<T, T> G, vector<pair<T, T>> I) {
  vi S(sz(I)), R;
  iota(all(S), 0);
  sort(all(S), [&](int a, int b) { return I[a] < I[b];
      });
  T cur = G.first;
  int at = 0;
  while (cur < G.second) { // (A)
    pair<T, int> mx = make_pair(cur, -1);
    while (at < sz(I) && I[S[at]].first <= cur) {
      mx = max(mx, make_pair(I[S[at]].second, S[at]));
      at++;
    }
    if (mx.second == -1) return {};
    cur = mx.first;
    R.push_back(mx.second);
  }
  return R;
}
```

ConstantIntervals.h

Description: Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value. Runs a callback g for each such interval.
Usage: `constantIntervals(0, sz(v), [&](int x){return v[x];}, [&](int lo, int hi, T val){...});`
Time: $\mathcal{O}\left(k \log \frac{n}{k}\right)$

19 lines

```
template<class F, class G, class T>
void rec(int from, int to, F& f, G& g, int& i, T& p, T
    q) {
  if (p == q) return;
  if (from == to) {
    g(i, to, p);
    i = to; p = q;
  } else {
    int mid = (from + to) >> 1;
    rec(from, mid, f, g, i, p, f(mid));
    rec(mid+1, to, f, g, i, p, q);
  }
}
template<class F, class G>
void constantIntervals(int from, int to, F f, G g) {
  if (to <= from) return;
  int i = from; auto p = f(i), q = f(to-1);
  rec(from, to-1, f, g, i, p, q);
  g(i, to, q);
}
```

10.2 Misc. algorithms

TernarySearch.h

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \dots < f(i) \geq \dots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

11 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) a = mid; // (A)
    else b = mid+1;
  }
  rep(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

LIS.h

Description: Compute indices for the longest increasing subsequence.
Time: $\mathcal{O}(N \log N)$

17 lines

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

FastKnapsack.h

Description: Given N non-negative integer weights w and a non-negative target t, computes the maximum S <= t such that S is the sum of some subset of the weights.
Time: $\mathcal{O}(N \max(w_i))$

16 lines

```
int knapsack(vi w, int t) {
  int a = 0, b = 0, x;
  while (b < sz(w) && a + w[b] <= t) a += w[b++];
  if (b == sz(w)) return a;
  int m = *max_element(all(w));
  vi u, v(2*m, -1);
  v[a+m-t] = b;
  rep(i,b,sz(w)) {
    u = v;
    rep(x,0,m) v[x+w[i]] = max(v[x+w[i]], u[x]);
    for (x = 2*m; --x > m;) rep(j, max(0,u[x]), v[x])
      v[x-w[j]] = max(v[x-w[j]], j);
  }
  for (a = t; v[a+m-t] < 0; a--) ;
  return a;
}
```

10.3 Dynamic programming

KnuthDP.h

Description: When doing DP on intervals: $a[i][j] = \min_{i<k<j}(a[i][k] + a[k][j]) + f(i,j)$, where the (minimal) optimal k increases with both i and j, one can solve intervals in increasing order of length, and search $k = p[i][j]$ for $a[i][j]$ only between $p[i][j-1]$ and $p[i+1][j]$. This is known as Knuth DP. Sufficient criteria for this are if $f(b,c) \le f(a,d)$ and $f(a,c) + f(b,d) \le f(a,d) + f(b,c)$ for all $a \le b \le c \le d$. Consider also: LineContainer (ch. Data structures), monotone queues, ternary search.
Time: $\mathcal{O}(N^2)$

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
      ); }
};
```

10.4 Other

Dates.h

Description: Dates
Usage: `Dates`

21 lines

```
int m2d  [12] =
  {31,28,31,30,1,30,31,31,30,31,30,31};
int m2da [13] =
  {0,31,59,90,120,151,181,212,243,273,304,334,365};
bool isLeap(int y) { // Change for leap years
  return y % 4 == 0 && (y%100 != 0 || y%400 == 0);
} int dateToInt(int d, int m, int y) {
  d--, m--, y--;
  return y*365 + m2dAc[m] + d +
  y/4 + (y % 4 == 3 && m > 1) + // Leap years every 4
  -(y/100 + (y % 100 == 99 && m > 1))+
  (y/400 + (y % 400 == 399 && m > 1));
  // Leap years excluding 100x including 400x
} void intToDate(int n, int &d, int &m, int &y) {
  y = n/365; n %= 365;
  n -= y/4 - y/100 + y/400; // Change for leap years
  y++; while (n < 0) n += 365+isLeap(--y);
  for (m = 0; m < 12 &&
    m2da[m+1]+(isLeap(y) && m+1 > 1) <= n; m++);
  d = n-m2da[m]-(isLeap(y) && m > 1)+1, m++;
}
```

10.5 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` failures 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`).

10.6 Optimization tricks

`__builtin_ia32_ldmxcsr(40896);` disables denormals (which make floats 20x slower near their minimum value).

10.6.1 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.

10.6.2 Pragmas

- `#pragma GCC optimize ("Ofast")` will make GCC auto-vectorize loops and optimizes floating points better.
- `#pragma GCC target ("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).