IIUM cat-us-trophy

Contents

.vimrc

```
set ai ts=4 sw=4 st=4 noet nu nohls
syntax enable
filetype plugin indent on
map <F6> :w<CR>:!g++ % -g && (ulimit -c unlimited; ./a.out < ~/input.txt) <CR>
map <F5> <F6>
colo pablo
map <F12> :!gdb ./a.out -c core <CR>
```

template.cpp

```
#include<cstdio>
#include<sstream>
#include<cstdlib>
#include<cctype>
#include<cmath>
#include<algorithm>
#include<set>
#include<queue>
#include<stack>
#include<list>
#include<iostream>
#include<string>
#include<vector>
#include<cstring>
#include<map>
#include<cassert>
#include<climits>
using namespace std;

#define REP(i,n) for(int i=0, _e(n); i<_e; i++)
#define FOR(i,a,b) for(int i(a), _e(b); i<=_e; i++)
#define FORD(i,a,b) for(int i(a), _e(b); i>=_e; i--)
#define FORIT(i, m) for (__typeof((m).begin()) i=(m).begin(); i!=(m).end(); ++i)
#define SET(t,v) memset((t), (v), sizeof(t))
#define ALL(x) x.begin(), x.end()
#define UNIQUE(c) (c).resize( unique( ALL(c) ) - (c).begin() )

#define sz size()
#define pb push_back
#define VI vector<int>
#define VS vector<string>
```

```
typedef long long LL;
typedef long double LD;
typedef pair<int,int> pii;

#define D(x) if(1) cout << __LINE__ <<" "<< #x " = " << (x) << endl;
#define D2(x,y) if(1) cout << __LINE__ <<" "<< #x " = " << (x) \
     <<", " << #y " = " << (y) << endl;
```

Combinatorics

Mathematical Sums

$\sum_{k=0}^{n} k = n(n+1)/2$ $\quad$ $\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$ $\quad$ $\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$

$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$ $\quad$ $\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$

$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$ $\quad$ $\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$

Binomial coefficients

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1												
1	1	1											
2	1	2	1										
3	1	3	3	1									
4	1	4	6	4	1								
5	1	5	10	10	5	1							
6	1	6	15	20	15	6	1						
7	1	7	21	35	35	21	7	1					
8	1	8	28	56	70	56	28	8	1				
9	1	9	36	84	126	126	84	36	9	1			
10	1	10	45	120	210	252	210	120	45	10	1		
11	1	11	55	165	330	462	462	330	165	55	11	1	
12	1	12	66	220	495	792	924	792	495	220	66	12	1
	0	1	2	3	4	5	6	7	8	9	10	11	12

$\binom{n}{k} = \frac{n!}{(n-k)!k!}$

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$

$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$

$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$

$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$

$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$

$\sum_{k=1}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$

$\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$

$\binom{n}{k} = \prod_{i=1}^{k}\frac{n-k+i}{i}$

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$

Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$

Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$

Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$. $\quad C_0 = 1$, $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$. $\quad C_{n+1} = C_n\frac{4n+2}{n+2}$.

$C_0, C_1, \ldots = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \ldots$

C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements . Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1}+D_{n-2}) = nD_{n-1}+(-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k}D_{n-k}$.

Stirling numbers of 1^{st} **kind** . $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $\left[\begin{smallmatrix} n \\ k \end{smallmatrix}\right] = |s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$ $\quad s(0,0) = 1$ and $s(n,0) = s(0,n) = 0$.

n/k	0	1	2	3	4	5	6	7	8	9
0	1									
1	0	1								
2	0	−1	1							
3	0	2	−3	1						
4	0	−6	11	−6	1					
5	0	24	−50	35	−10	1				
6	0	−120	274	−225	85	−15	1			
7	0	720	−1764	1624	−735	175	−21	1		
8	0	−5040	13068	−13132	6769	−1960	322	−28	1	
9	0	40320	−109584	118124	−67284	22449	−4536	546	−36	1
	0	1	2	3	4	5	6	7	8	9

Stirling numbers of 2^{nd} kind . $S_{n,k} = \left\{ {n \atop k} \right\}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $\left\{ {n \atop k} \right\} = S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$; $S_{n,1} = S_{n,n} = 1$.
$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n$.

n/k	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	0	1									
2	0	1	1								
3	0	1	3	1							
4	0	1	7	6	1						
5	0	1	15	25	10	1					
6	0	1	31	90	65	15	1				
7	0	1	63	301	350	140	21	1			
8	0	1	127	966	1701	1050	266	28	1		
9	0	1	255	3025	7770	6951	2646	462	36	1	
10	0	1	511	9330	34105	42525	22827	5880	750	45	1
	0	1	2	3	4	5	6	7	8	9	10

Bell numbers . B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, 877, \ldots$
$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n+1} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Eulerian numbers . $E(n,k) = \left\langle {n \atop k} \right\rangle$ is the number of permutations with exactly k descents ($i : \pi_i < \pi_{i+1}$) / ascents ($\pi_i > \pi_{i+1}$) / excedances ($\pi_i > i$) / $k+1$ weak excedances ($\pi_i \geq i$).
Formula: $E(n,m) = (m+1)E(n-1,m) + (n-m)E(n-1,m-1)$. $E(n,0) = E(n,n-1) = 1$.
$E(n,m) = \sum_{k=0}^{m} (-1)^k \binom{n+1}{k} (m+1-k)^n$.

Double factorial . Permutations of the multiset $\{1, 1, 2, 3, \ldots n, n\}$ such that for each k, all the numbers between two occurrences of k in the permutation are greater than k. $(2n-1)!! = \prod_{k=1}^{n} (2k-1)$.

Eulerian numbers of 2^{nd} kind . Related to Double factorial, number of all such permutations that have exactly m ascents. $\left\langle\left\langle {n \atop m} \right\rangle\right\rangle = (2n-m-1) \left\langle\left\langle {n-1 \atop m-1} \right\rangle\right\rangle + (m+1) \left\langle\left\langle {n-1 \atop m} \right\rangle\right\rangle$. $\left\langle\left\langle {n \atop 0} \right\rangle\right\rangle = 1$

Multinomial theorem . $(a_1 + \cdots + a_k)^n = \sum \binom{n}{n_1, \ldots, n_k} a_1^{n_1} \ldots a_k^{n_k}$, where $n_i \geq 0$ and $\sum n_i = n$.
$\binom{n}{n_1, \ldots, n_k} = M(n_1, \ldots, n_k) = \frac{n!}{n_1! \ldots n_k!}$. $M(a, \ldots, b, c, \ldots) = M(a + \cdots + b, c, \ldots) M(a, \ldots, b)$

Necklaces and bracelets Necklace of length n is an equivalence class of n-character strings over an alphabet of size k, taking all rotations as equivalent. Number of necklases: $N_k(n) = \frac{1}{n} \sum_{d|n} \varphi(d) k^{n/d}$
Bracelet is a necklace such that strings may also be equivalent under reflection. Number of bracelets:

$$B_k(n) = \begin{cases} \frac{1}{2} N_k(n) + \frac{1}{4}(k+1) k^{n/2} & \text{if } n \text{ is even} \\ \frac{1}{2} N_k(n) + \frac{1}{2} k^{(n+1)/2} & \text{if } n \text{ is odd} \end{cases}$$

An aperiodic necklace of length n is an equivalence class of size n, i.e., no two distinct rotations of a necklace from such class are equal. Number of aperiodic necklaces: $M_k(n) = \frac{1}{n} \sum_{d|n} \mu(d) k^{n/d}$.
Each aperiodic necklace contains a single Lyndon word. Lyndon word is n-character string over an alphabet of size k, and which is the minimum element in the lexicographical ordering of all its rotations. $w = uv$ is a Lyndon word if and only if u and v are Lyndon words and $u < v$.

Burnside's lemma . The number of orbits under group G's action on set X:
$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.")
Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o\in X/G} w(o) = \frac{1}{|G|}\sum_{g\in G}\sum_{x\in X_g} w(x)$.

Data structures (BIT, BIT2D, RMQ-DP, RMQ-segment tree, union-find, misof's tree, treap)

BIT - Binary indexed trees

```
int bit[M],n;
void update(int x, int v) { while( x <= n ) { bit[x] += v; x += x & -x; } }
int sum(int x) { int ret=0; while(x>0){ ret += bit[x]; x -= x & -x; } return ret; }
```

BIT 2D

```
int bit[M][M], n;
int sum( int x, int y ){
    int ret = 0;
    while( x > 0 ){
        int yy = y; while( yy > 0 ) ret += bit[x][yy], yy -= yy & -yy;
        x -= (x & -x);
    }
    return ret ;
}
void update(int x , int y , int val){
    int y1;
    while (x <= n){
        y1 = y;
        while (y1 <= n){ bit[x][y1] += val; y1 += (y1 & -y1); }
        x += (x & -x);
    }
}
```

RMQ DP

```
#define better(a,b) A[a]<A[b]?(a):(b)
int make_dp(int n) { // N log N
    REP(i,n) H[i][0]=i;
    for(int l=0,k; (k=1<<l) < n; l++) for(int i=0;i+k<n;i++)
        H[i][l+1] = better(H[i][l], H[i+k][l]);
}
int query_dp(int a, int b) {
int l = __lg(b-a);
return better(H[a][l], H[b-(1<<l)+1][l]);
}
```

RMQ segment tree

```
const int M = 100005;
int n, in[M], f[M], st[M], en[M];
struct data { int l, r, ans, next_l, next_r; };
data d[4*M]; // which is the range? :S
int nd;
int build( int l, int r, int id ) {
    d[ id ].l = l, d[ id ].r = r;
    if( l == r ) d[ id ].ans = f[l];
    else {
```

```
        int bar = ( r-l ) / 2 + l;
        d[id].next_l = ++nd, d[id].next_r = ++nd;
        int left = build( l, bar, d[id].next_l );
        int right = build( bar+1, r, d[id].next_r );
        d[id].ans = max( left, right );
    }
    return d[ id ].ans;
}
int query( int l, int r, int id = 0 ) {
    if( l > r ) return 0;
    if( d[id].l == l && d[id].r == r ) return d[id].ans;
    else {
        int bar = (d[id].r-d[id].l) / 2 + d[id].l;
        int left = 0, right = 0;
        if( l <= bar ) {
            if( r <= bar ) left = query( l, r, d[id].next_l );
            else {
                left = query( l, bar, d[id].next_l );
                right = query( bar+1, r, d[id].next_r );
            }
        }else right = query( l, r, d[id].next_r );
        return max( left, right );
    }
}
```

Union find - rank by number of elements

```
struct unionfind {
    int p[MAX], r[MAX]; // r contains the population
    unionfind() { REP(i,MAX) p[i] = i, r[i] = 1; }
    int find( int x ) { if( p[x] == x ) return x; else return p[x] = find( p[x] ); }
    void Union(int x, int y) {
        int px = find( x ), py = find( y );
        if( px == py ) return; //already joined
        if( r[ px ] < r[ py ] ) p[px] = py, r[py] += r[px];
        else p[ py ] = px, r[px] += r[py];
    }
};
```

misof's tree

```
int tree[17][65536];
void insert(int x) { for (int i=0; i<17; i++) { tree[i][x]++; x/=2; } }
void erase(int x) { for (int i=0; i<17; i++) { tree[i][x]--; x/=2; } }
int kThElement(int k) { int a=0, b=16;
    while (b--) { a*=2; if (tree[b][a]<k) k-=tree[b][a++]; }
    return a; }
```

Treaps

```
const int N = 100 * 1000;
struct node { int value, weight, ch[2], size; } T[ N+10 ] ; int nodes;
#define Child(x,c) T[x].ch[c]
#define Value(x) T[x].value
#define Weight(x) T[x].weight
```

```
#define Size(x) T[x].size
#define Left Child(x,0)
#define Right Child(x,1)
int update(int x) { if(!x)return 0; Size(x) = 1+Size(Left)+Size(Right); return x; }
int newnode(int value, int prio) {
    T[++nodes]=(node){value,prio,0,0};
    return update(nodes);
}
void split(int x, int by, int &L, int &R)
{
    if(!x) { L=R=0; }
    else if(Value(x) < Value(by)) { split(Right,by,Right,R); update(L=x); }
    else { split(Left,by,L,Left); update(R=x); }
}
int merge(int L, int R)
{
    if(!L) return R; if(!R) return L;
    if(Weight(L)<Weight(R)) { Child(L,1) = merge(Child(L,1), R); return update(L);}
    else { Child(R,0) = merge( L, Child(R, 0)); return update(R); }
}
int insert(int x, int n)
{
    if(!x) { return update(n); }
    if(Weight(n)<=Weight(x)) {split(x,n,Child(n,0),Child(n,1)); return update(n);}
    else if(Value(n) < Value(x)) Left=insert(Left,n); else Right=insert(Right,n);
    return update(x);
}
int del(int x, int value)
{
    if(!x) return 0;
    if(value == Value(x)) { int q = merge(Left,Right); return update(q); }
    if(value < Value(x)) Left = del(Left,value); else Right = del(Right, value);
    return update(x);
}
int find_GE(int x, int value) {
    int ret=0;
    while(x) { if(Value(x)==value)return x;
        if(Value(x)>value) ret=x, x=Left; else x=Right; }
    return ret;
}
int find(int x, int value) {
    for(; x; x=Child(x,Value(x)<value)) if(Value(x)==value)return x;
    return 0;
}
int findmin(int x) { for(;Left;x=Left); return x; }
int findmax(int x) { for(;Right;x=Right); return x; }
int findkth(int x, int k) {
    while(x) {
        if(k<=Size(Left)) x=Left;
        else if(k==Size(Left)+1)return x;
        else { k-=Size(Left)+1; x=Right; }
    }
```

```
}
int queryrangekth(int &x, int a1, int a2, int k) {
    a1 = find(x, a1); a2 = find(x, a2);
    assert(a1 && a2);
    int a,b,c; split(x,a1,a,b); split(b,a2,b,c);
    int ret = findkth(b,k);
    x = merge(a, merge(b,c));
    return Value(ret);
}
```

Number theory (*gcd, totient, sieve, modexp, rabinmiller)

Linear diophantine equation . $ax + by = c$. Let $d = \gcd(a, b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x, y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a, b)$, and multiply its solutions by $\frac{c}{d}$.
Linear diophantine equation in n variables: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1, \ldots, a_n)|c$. To find some solution, let $b = \gcd(a_2, \ldots, a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$.
Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\varphi(m)-1} \pmod m$.

Chinese Remainder Theorem . System $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1m_2 \ldots m_n$: $x = a_1b_1\frac{M}{m_1} + \cdots + a_nb_n\frac{M}{m_n} \pmod M$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.
System $x \equiv a \pmod m$, $x \equiv b \pmod n$ has solutions iff $a \equiv b \pmod g$, where $g = \gcd(m, n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b - a)m/g \equiv b + S(a - b)n/g \pmod L$, where S and T are integer solutions of $mT + nS = \gcd(m, n)$.

Prime-counting function . $\pi(n) = |\{p \le n : p\text{isprime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n \ln n$.
List of primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71...

Miller-Rabin's primality test . Given $n = 2^rs + 1$ with odd s, and a random integer $1 < a < n$. If $a^s \equiv 1 \pmod n$ or $a^{2^js} \equiv -1 \pmod n$ for some $0 \le j \le r - 1$, then n is a probable prime. With bases 2, 7 and 61, the test indentifies all composites below 2^{32}. Probability of failure for a random a is at most $1/4$.

Pollard-ρ . Choose random x_1, and let $x_{i+1} = x_i^2 - 1 \pmod n$. Test $\gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, \ldots$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes . A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Perfect numbers . $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers . A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod n$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

Number/sum of divisors . $\tau(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k}(a_j + 1)$. $\sigma(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1}-1}{p_j-1}$. $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x}-1}{p_i^x-1}$

Euler's phi function . $\varphi(n) = |\{m \in \mathbb{N}, m \le n, \gcd(m, n) = 1\}|$. $\varphi(n) = n \cdot \prod_{p|n}\left(1 - \frac{1}{p}\right)$
$\varphi(mn) = \frac{\varphi(m)\varphi(n)\gcd(m,n)}{\varphi(\gcd(m,n))}$. $\varphi(p^a) = p^{a-1}(p - 1)$. $\sum_{d|n}\varphi(d) = \sum_{d|n}\varphi(\frac{n}{d}) = n$.

Euler's theorem . $a^{\varphi(n)} \equiv 1 \pmod n$, if $\gcd(a, n) = 1$.
Wilson's theorem . p is prime iff $(p - 1)! \equiv -1 \pmod p$.

Mobius function . $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$,

then $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$, and vice versa. $\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d) f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

Legendre symbol . If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod p$.

Jacobi symbol . If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{k_i}$.

Primitive roots . If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod m$) is $\varphi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\varphi(\varphi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \mathrm{ind}_g(a)$ modulo $\varphi(m)$, such that $g^i \equiv a \pmod m$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.
If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod p$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod p$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod p$, $g^u \equiv x \pmod p$. $x^n \equiv a \pmod p$ iff $g^{nu} \equiv g^i \pmod p$ iff $nu \equiv i \pmod p$.)

Discrete logarithm problem . Find x from $a^x \equiv b \pmod m$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil \sqrt{m} \rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod m$. Precompute all values that the RHS can take for $z = 0, 1, \ldots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples . Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod 2$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

Postage stamps/McNuggets problem . Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax+by$ $(x, y \geq 0)$, and the largest is $(a-1)(b-1)-1 = ab-a-b$.

Fermat's two-squares theorem . Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

Congruence $ax \equiv b \pmod n$

```
int congruence( int a, int b, int n ) { // finds ax = b(mod n)
    int d = gcd( a, n );
    if( b % d != 0 ) return 1<<30; // no solution
    pii ans = egcd( a, n );
    int ret = ans.x * ( b/d + 0LL ), mul = n/d;
    ret %= mul;
    if( ret < 0 ) ret += mul;
    return ret;
}
```

Extended GCD

```
pii egcd( LL a, LL b ) {  // returns x,y | ax + by = gcd(a,b)
    if( b == 0 ) return pii( 1, 0 );
    else {
        pii d = egcd( b, a % b );
        return pii( d.y, d.x - d.y * ( a / b ) );
    }
}
```

GCD

```
LL gcd( LL a, LL b ) { return !b ? a : gcd( b, a%b ); }
```

General EGCD

```
template<class T> inline T euclid(T a,T b,T &X,T &Y)
{
    if(a<0) { T d=euclid(-a,b,X,Y); X=-X; return d; }
    if(b<0) { T d=euclid(a,-b,X,Y); Y=-Y; return d; }
    if(b==0) { X=1; Y=0; return a; }
    else { T d=euclid(b,a%b,X,Y); T t=X; X=Y; Y=t-(a/b)*Y; return d; }
}
int X[110],Y[110]; LL v[110];
void gen_euclid(int n)
{
    int g = a[0];
    FOR(i,1,n) g = euclid(g, a[i], X[i], Y[i]);
    LL mult = 1;
    FORD(i,n,1) v[i] = (mult * Y[i]) % m, mult = (mult * X[i]) % m;
    v[0] = mult;
}
```

Phi

```
int phi (int n) {
    int ret = n;
    for (int i=2; i*i<=n; ++i) if( n%i == 0) {
        while(n % i == 0) n /= i;
        ret -= ret / i;
    }
    if (n > 1) ret -= ret / n;
    return ret;
}
```

Power iterative - modexp

```
LL power( LL a, LL b, LL mod ) {
    LL x = 1, y = a;
    while( b ) {
        if( b&1 ) x *= y, x %= mod;
        y *= y, y %= mod, b/=2;
    }
    return x%mod;
}
```

Rabin-Miller

```
bool Miller(LL p, LL s, int a){
    if(p==a) return 1;
    LL mod=expmod(a,s,p);
    for(;s-p+1 && mod-1 && mod-p+1;s*=2) mod=mulmod(mod,mod,p);
    return mod==p-1 || s%2;
}
bool isprime(LL n) {
    if(n<2)return 0; if(n%2==0) return n==2;
    LL s=n-1;
    while(s%2==0) s/=2;
    return Miller(n,s,2) && Miller(n,s,7) && Miller(n,s,61);
} // for 341*10^12  primes <= 17
```

Sieve prime

```
const int MAX = 100000000;
int p[ MAX/64 + 2 ], np = 0;
#define on(x) ( p[x/64] & (1<<( (x%64)/2 ) ) )
#define turn(x)  p[x/64] |= (1<<( (x%64)/2 ) )
void sieve() {
    for(int i=3;i*i<MAX; i+=2)  { if( !on(i) ) {
            int ii = i*i, i2 = i+i;
            for(int j=ii; j<MAX; j+=i2) turn(j); }}}
inline bool prime(int num){return num>1 && (num==2 || ((num&1) && !on(num) ));}
```

Sieve totient

```
FOR(i,1,M) f[i] = i;
FOR(n,2,M) if( f[n] == n ) for(int k=n; k<=M; k+=n) f[k] *= n-1, f[k] /= n;
```

String algorithms (SuffixA, Aho-Corasick, KMP)

Suffix arrays

```
const int N = 100 * 1000 + 10;
char str[N]; bool bh[N], b2h[N];
int rank[N], pos[N], cnt[N], next[N], lcp[N];
bool smaller(int a, int b) { return str[a]<str[b];}
void suffix_array(int n) {
    REP(i,n)pos[i]=i, b2h[i]=false;
    sort(pos,pos+n,smaller);
    REP(i,n) bh[i]=!i||str[pos[i]] != str[pos[i-1]];
    for(int h=1;h<n;h*=2) {
        int buckets=0;
        for(int i=0,j; i<n; i=j) {
            j=i+1;
            while(j<n && !bh[j])j++;
            next[i]=j;
            buckets++;
        }
        if(buckets==n)break;
        for(int i=0;i<n;i=next[i]) {
            cnt[i] = 0;
            FOR(j, i, next[i]-1) rank[pos[j]]=i;
        }
        cnt[rank[n-h]]++;
        b2h[rank[n-h]]=true;
        for(int i=0;i<n;i=next[i]) {
            FOR(j, i, next[i]-1) {
                int s = pos[j]-h;
                if(s>=0){
                    rank[s] = rank[s] + cnt[rank[s]]++;
                    b2h[rank[s]]=true;
            }   }
            FOR(j, i, next[i]-1) {
                int s = pos[j]-h;
                if(s>=0 && b2h[rank[s]])
                    for(int k=rank[s]+1;!bh[k] && b2h[k]; k++) b2h[k]=false;
        }   }
        REP(i,n) pos[rank[i]]=i, bh[i]|=b2h[i];
```

```
    }
    REP(i,n) pos[rank[i]]=i;
}
void get_lcp(int n) {
    lcp[0]=0;
    int h=0;
    REP(i,n) if(rank[i]) {
        int j=pos[rank[i]-1];
        while(i+h<n && j+h<n && str[i+h] == str[j+h]) h++;
        lcp[rank[i]]=h;
        if(h)h--;
}   }
//slower version of SA, also works with get_lcp
struct data {
int nr[2], p;
bool operator<(const data &v)const{ return nr[0] < v.nr[0] || ...;}
bool operator==(const data &v)const{return nr[1]==v.nr[1]&&nr[0]==v.nr[0];}
} L[MAXN];
int P[MAXLG+2][MAXN], pos[MAXN], rank[MAXN];
int suffix_array(char *A, int N)
{
    int step,cnt;
    REP(i,N) P[0][i] = A[i];
    for(step=1,cnt=1;cnt/2<N;cnt*=2,step++) {
        REP(i,N) L[i]=(data){P[step-1][i],(i+cnt<N)?P[step-1][i+cnt]:-1,i};
        sort(L,L+N);
        REP(i,N) P[step][L[i].p] = i && L[i]==L[i-1]? P[step][L[i-1].p]:i;
    }
    REP(i,N) rank[L[i].p]=i;
    REP(i,N) pos[rank[i]]=i;
    return step-1;
}
```

Aho-Corasick

```
#define NC 26
#define NP 10005
#define M 100005
#define MM 500005
char a[M];
char b[NP][105];
int nb, cnt[NP], lenb[NP], alen;
int g[MM][NC], ng, f[MM], marked[MM];
int output[MM], pre[MM];
#define init(x) {REP(_i,NC)g[x][_i] = -1; f[x]=marked[x]=0; output[x]=pre[x]=-1; }
void match() {
    ng = 0;
    init( 0 );
    // part 1 - building trie
    REP(i,nb) {
        cnt[i] = 0;
        int state = 0, j = 0;
        while(g[state][b[i][j]] != -1 && j < lenb[i]) state = g[state][b[i][j]], j++;
```

```
        while( j < lenb[i] ) {
            g[state][ b[i][j] ] = ++ng;
            state = ng;
            init( ng );
            ++j;
        }
        if( ng >= MM ) { cerr <<"i am dying"<<endl; while(1); // suicide }
        output[ state ] = i;
    }
    // part 2 - building failure function
    queue< int > q;
    REP(i,NC) if( g[0][i] != -1 ) q.push( g[0][i] );
    while( !q.empty() ) {
        int r = q.front(); q.pop();
        REP(i,NC) if( g[r][i] != -1 ) {
            int s = g[r][i];
            q.push( s );
            int state = f[r];
            while( g[state][i] == -1 && state ) state = f[state];
            f[s] = g[state][i] == -1 ? 0 : g[state][i];
        }
    }
    // final smash
    int state = 0;
    REP(i,alen) {
        while( g[state][a[i]] == -1 ) {
            state = f[state];
            if( !state ) break;
        }
        state = g[state][a[i]] == -1 ? 0 : g[state][a[i]];
        if( state && output[ state ] != -1 ) marked[ state ] ++;
    }
    // counting
    REP(i,ng+1) if( i && marked[i] ) {
        int s = i;
        while( s != 0 ) cnt[ output[s] ] += marked[i], s = f[s];
    }
}
```

KMP

```
int f[ len ];
f[0] = f[1] = 0;
FOR(i,2,len) {
    int j = f[i-1];
    while( true ) {
        if( s[j] == s[i-1] ) { f[i] = j + 1; break;
        }else if( !j ) { f[i] = 0; break;
        }else j = f[j];
    }
}
i = j = 0;
while( true ) {
```

```
    if( i == len ) break;
    if( text[i] == s[j] ) { i++, j++;
        if( j == slen ) // match found
    }else if( j > 0 ) j = f[j];
    else i++;
}
```

Longest Increasing Subsequence

```
int n,total, nprob = 0;
vector< int > table;
while(scanf("%d", &total)==1){
    if( total == 0 ) break;
    table.clear();
    REP(kkk, total) {
        scanf("%d",&n);
        vector< int >::iterator i = lower_bound( table.begin(), table.end(), n );
        if( i== table.end() ) table.push_back( n );
        else *i <?= n;
    }
    printf("Set %d: %d\n",++nprob,table.size());
}
```

Permutation index

```
int perm_index (char str[], int size) {
    int index = 0;
    REP(i,size-1) {
        FOR(j,i+1,size-1) if (str[i] > str[j]) index ++;
        index *= size-i-1; }
    return index;
}
```

Graph algorithms (LCA, SCC, BPM-konig, Bellman-Ford, NetFlow, Min-Cost MaxFlow)

Tarjan's offline LCA

```
function TarjanOLCA(u)
    MakeSet(u); u.ancestor := u;
    for each v in u.children do
        TarjanOLCA(v); Union(u,v); Find(u).ancestor := u;
    u.colour := black;
    for each v such that {u,v} in P and v.color==black do
        print "LCA", u, v, Find(v).ancestor
```

Tarjan's Strong Connected Components

```
procedure tarjan(v)
  index = count; v.lowlink = count++; S.push(v);color[v] = 1;
  for all (v, v2) in E do
     if (!color[v2])
        tarjan(v2); v.lowlink = min(v.lowlink, v2.lowlink);
     else if (color[v2]==1)
        v.lowlink = min(v.lowlink, v2.lowlink);
  if (v.lowlink == index)
    do { v2 = S.top(); S.pop(); print v2; color[v2]=2; } while (v2 != v);
for all v in V do if(!color[v]) tarjan(v);
```

Bipartite matching with Konig

```
int grid[M][M], l[M], r[M], seen[M], rows, cols; bool lT[M], rT[M];
bool dfs(int x) {
    if( seen[x] ) return false;
    seen[x] = true;
    Rep(i,cols) if( grid[x][i] ) if( r[i] == -1 || dfs( r[i] ) )
    { r[i] = x, l[x] = i; return true; }
    return false;
}
int bpm() {
    SET( l, -1 ); SET( r, -1 ); int ret = 0;
    Rep(i,rows) { SET( seen, 0 ); if( dfs( i ) ) ret ++; }
    return ret;
}
void konigdfs(int x) {
    if( !lT[x] ) return; lT[x] = 0;
    Rep(i,cols) if(grid[x][i] && i != l[x])
    { rT[i] = true; if( r[i] != -1) konigdfs(r[i]); }
}
int konig() { SET(lT,1); SET(rT,0); Rep(i,rows) if(l[i]==-1) konigdfs(i);}
```

Bellman-Ford

```
VI e[M], c[M];
int n, d[M], p[M];
int inf = 1<<29;
int bford( int s, int f ) {
    REP(i,n) d[i] = i == s ? 0 : inf, p[i] = -1;
    REP(_,n-1) {
        bool done = 1;
        REP(i,n) REP(j,e[i].sz) {
            int u = i, v = e[i][j], uv = c[i][j];
            if( d[u] + uv < d[v] ) d[v] = d[u] + uv, p[v] = u, done = 0;
        }
        if( done ) break;
    }
    REP(i,n) REP(j,e[i].sz) {
        int u = i, v = e[i][j], uv = c[i][j];
        if( d[u] + uv < d[v] ) return -33;
    }
    if( d[f] == inf ) return -33;
    return d[f];
}
```

Network flow - Slow

```
#define M 750
int nr, nc, o = 355, source = 740, sink = 741;
vector<int> edge[M];
int cap[M][M];
bool vis[M];
void init() {
    REP(i,M) edge[i].clear();
    SET( cap, 0 );
```

```
}
void add( int a, int b, int c, int d ) {
    edge[a].pb(b), edge[b].pb(a);
    cap[a][b] += c, cap[b][a] += d;
}
int dfs( int src, int snk, int fl ) {
    if( vis[src] ) return 0;
    if( snk == src ) return fl;
    vis[src] = 1;

    REP(i,edge[src].sz) {
        int v = edge[src][i];
        int x = min( fl, cap[src][v] );
        if( x > 0 ) {
            x = dfs( v, snk, x );
            if( !x ) continue;
            cap[src][v] -= x;
            cap[v][src] += x;
            return x;
        }
    }
    return 0;
}
int flow( int src, int snk ) {
    int ret = 0;
    while( 1 ) {
        SET( vis, 0 );
        int delta = dfs( src, snk, 1<<30 );
        if( !delta ) break;
        ret += delta;
    }
    return ret;
}
```

Network flow - Dinic fast

```
const int maxN = 5005;
const int maxE = 70000;
const int inf = 1000000005;
int nnode, nedge, src, snk;
int Q[ maxN ], pro[ maxN ], fin[ maxN ], dist[ maxN ];
int flow[ maxE ], cap[ maxE ], to[ maxE ], next[ maxE ];
void init( int _nnode, int _src, int _snk ) {
    nnode = _nnode, nedge = 0, src = _src, snk = _snk;
    FOR(i,1,nnode) fin[i] = -1;
}
void add( int a, int b, int c1, int c2 ) {
    to[nedge]=b, cap[nedge]=c1, flow[nedge]=0, next[nedge]=fin[a], fin[a]=nedge++;
    to[nedge]=a, cap[nedge]=c2, flow[nedge]=0, next[nedge]=fin[b], fin[b]=nedge++;
}
bool bfs() {
    SET( dist, -1 );
    dist[src] = 0;
```

```
    int st = 0, en = 0;
    Q[en++] = src;
    while( st < en ) {
        int u = Q[ st++ ];
        for(int e = fin[u]; e >= 0; e = next[e] ) {
            int v = to[e];
            if( flow[e] < cap[e] && dist[v] == -1 ) {
                dist[v] = dist[u] + 1;
                Q[en++] = v;
            }
        }
    }
    return dist[snk] != -1;
}
int dfs(int u, int fl) {
    if( u == snk ) return fl;
    for( int& e = pro[u]; e >= 0; e = next[e] ) {
        int v = to[e];
        if( flow[e] < cap[e] && dist[v] == dist[u]+1 ) {
            int x = dfs( v, min( cap[e] - flow[e] , fl ) );
            if( x > 0 ) {
                flow[ e ] += x, flow[ e^1 ] -= x;
                return x;
            }
        }
    }
    return 0;
}
LL dinic() {
    LL ret = 0;
    while( bfs() ) {
        FOR(i,1,nnode) pro[i] = fin[i];
        while( 1 ) {
            int delta = dfs( src, inf );
            if( !delta ) break;
            ret += delta;
        }
    }
    return ret;
}
```

Min-Cost Max Flow

```
#define N 705
int n, nE;
int d[N], pre[N];

struct edge {
    int to, cost, cap;
    int back;
};

edge E[N*N];
```

```
vector< int > e[N];

int mincost( int s, int t, int lim ) {

    int flow = 0, ret = 0;
    while( flow < lim ) {

        SET( d, -1 ); SET( pre, -1 );
        d[s] = 0;
//      cout <<"source "<< s <<" sink " << t << endl;
        // bellman ford
        int jump = n-1;
        bool done = 0;
        while( !done && --jump >= 0) {
            done = 1;
            REP(i,n) if( d[i] != -1 ) REP(j,e[i].sz) {
                edge& x = E[ e[i][j] ];
                int v = x.to;
                if( x.cap > 0 && ( d[v] == -1 || d[v] > d[i] + x.cost )) {
                    d[v] = d[i] + x.cost;
                    pre[v] = x.back;
                    done = 0;
//                  cout<<v<<" "<<d[v]<<endl;
                }
            }
            if( done ) break;
        }
//      cout << d[t] << endl;
        if( d[t] == -1 ) break;
//      cout <<"found one path "<<endl;
        // traverse back
        int x = t, cflow = 1<<30;
        while( x != s ) {
            edge& ed = E[ pre[x] ];
            cflow = min( cflow, E[ ed.back ].cap );
//          cout << ed.to <<" to "<< x << endl;
            x = ed.to;
        }
        if( !cflow ) break;
        int take = min( lim - flow, cflow );
        ret += d[t] * take;
        flow += take;
//      cout <<"taken flow "<< take <<" with cost "<< d[t] * take << endl << endl;
        x = t;
        while( x != s ) {
            edge& back = E[ pre[x] ];
            edge& forw = E[ back.back ];
            back.cap += take;
            forw.cap -= take;
            x = back.to;
        }
    }
```

```
//  cout << "total flow " << flow << endl;
    if( flow < lim ) return -1;
    return ret;
}
// remember to add -cost in the opposite direction
void add( int u, int v, int uv, int vu, int fuv, int fvu ) {
    int a = nE, b = nE+1;
    nE += 2;
    E[ a ].to = v, E[ a ].cost = uv, E[ a ].cap = fuv, E[ a ].back = b;
    E[ b ].to = u, E[ b ].cost = vu, E[ b ].cap = fvu, E[ b ].back = a;
    e[ u ].pb( a ), e[ v ].pb( b );
}
```

Euler's theorem . For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets . Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leq |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

Matrix-tree theorem . Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -\deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Euler tours . Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected.

Stable marriages problem . While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner's min-cut algorithm . Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

2-SAT . Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

Randomized algorithm for non-bipartite matching . Let G be a simple undirected graph with even $|V(G)|$. Build a matrix A, which for each edge $(u, v) \in E(G)$ has $A_{i,j} = x_{i,j}$, $A_{j,i} = -x_{i,j}$, and is zero elsewhere. Tutte's theorem: G has a perfect matching iff $\det G$ (a multivariate polynomial) is identically zero. Testing the latter can be done by computing the determinant for a few random values of $x_{i,j}$'s over some field. (e.g. Z_p for a sufficiently large prime p)

Prufer code of a tree . Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n - 2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Erdos-Gallai theorem . A sequence of integers $\{d_1, d_2, \ldots, d_n\}$, with $n-1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \cdots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \ldots, n-1$.

Games theory (Grundy, Misère)

Grundy numbers . For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x, y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Misère Nim . A position with pile sizes $a_1, a_2, \ldots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Geometry

Circle using three points Let $A = (0, 0)$ centers are $(C_y(B_x^2 + B_y^2) - B_y(C_x^2 + C_y^2))/D$ and $(B_x(C_x^2 + C_y^2) - C_x(B_x^2 + B_y^2))/D$ where $D = 2(B_x C_y - B_y C_x)$.

Geodistance

```
double gdist ( double a_lat, double b_lat, double a_long, double b_long ) {
return acos( cos(a_lat) * cos(b_lat) * cos(a_long - b_long) + sin(a_lat) * sin(b_lat));
} // don't forget to multiply radius with it! ;)
```

Point in a triangle

```
bool isInside(const Vector &P) {
    Vector a = C - A, b = B - A, c = P - A;
    T under = a.x*b.y-b.x*a.y;
    T u = (c.x*b.y-c.y*b.x), v = (a.x*c.y-a.y*c.x);
    return u >= 0 && v >= 0 && u+v <= under || u<=0 && v<=0 && u+v >= under;
} //remove equalities if you dont want the boundary
```

Pick's theorem . $I = A - B/2 + 1$, where A is the area of a lattice polygon, I is number of lattice points inside it, and B is number of lattice points on the boundary. Number of lattice points minus one on a line segment from $(0, 0)$ and (x, y) is $\gcd(x, y)$.

$a \cdot b = a_x b_x + a_y b_y = |a| \cdot |b| \cdot \cos(\theta)$

$a \times b = a_x b_y - a_y b_x = |a| \cdot |b| \cdot \sin(\theta)$

3D: $a \times b = (a_y b_z - a_z b_y,\ a_z b_x - a_x b_z,\ a_x b_y - a_y b_x)$

Line $ax + by = c$ through $A(x_1, y_1)$ and $B(x_2, y_2)$: $a = y_1 - y_2$, $b = x_2 - x_1$, $c = ax_1 + by_1$.

Half-plane to the left of the directed segment AB: $ax + by \geq c$.

Normal vector: (a, b). Direction vector: $(b, -a)$. Perpendicular line: $-bx + ay = d$.

Point of intersection of $a_1 x + b_1 y = c_1$ and $a_2 x + b_2 y = c_2$ is $\frac{1}{a_1 b_2 - a_2 b_1}(c_1 b_2 - c_2 b_1, a_1 c_2 - a_2 c_1)$.

Distance from line $ax + by + c = 0$ to point (x_0, y_0) is $|ax_0 + by_0 + c|/\sqrt{a^2 + b^2}$.

Distance from line AB to P (for any dimension): $\frac{|(A-P)\times(B-P)|}{|A-B|}$.

Point-line segment distance:

```
if (dot(B-A, P-A) < 0) return dist(A,P);
if (dot(A-B, P-B) < 0) return dist(B,P);
return fabs(cross(P,A,B) / dist(A,B));
```

Projection of point C onto line AB is $\frac{AB \cdot AC}{AB \cdot AB} AB$.

Projection of (x_0, y_0) onto line $ax + by = c$ is $(x_0, y_0) + \frac{1}{a^2+b^2}(ad, bd)$, where $d = c - ax_0 - by_0$.

Projection of the origin is $\frac{1}{a^2+b^2}(ac, bc)$.

Segment-segment intersection . Two line segments intersect if one of them contains an endpoint of the other segment, or each segment straddles the line, containing the other segment (AB straddles line l if A and B are on the opposite sides of l.)

Circle-circle and circle-line intersection .

```
a = x2 - x1;
b = y2 - y1;
```

```
c = [(r1^2 - x1^2 - y1^2) - (r2^2 - x2^2 - y2^2)] / 2;
d = sqrt(a^2 + b^2);
if not |r1 - r2| <= d <= |r1 + r2|, return "no solution"
if d == 0, circles are concentric, a special case
// Now intersecting circle (x1,y1,r1) with line ax+by=c
Normalize line: a /= d; b /= d; c /= d;  // d=sqrt(a^2+b^2)
e = c - a*x1 - b*y1;
h = sqrt(r1^2 - e^2);   // check if r1<e for circle-line test
return (x1, y1) + (a*e, b*e) +/- h*(-b, a);
```

Angular bisector of angle ABC is line BD, where $D = \frac{BA}{|BA|} + \frac{BC}{|BC|}$.
Center of incircle of triangle ABC is at the intersection of angular bisectors, and is $\frac{a}{a+b+c}A + \frac{b}{a+b+c}B + \frac{c}{a+b+c}C$, where a, b, c are lengths of sides, opposite to vertices A, B, C. Radius $= \frac{2S}{a+b+c}$.

Counter-clockwise rotation around the origin . $(x, y) \mapsto (x\cos\phi - y\sin\phi, x\sin\phi + y\cos\phi)$.
90-degrees counter-clockwise rotation: $(x, y) \mapsto (-y, x)$. Clockwise: $(x, y) \mapsto (y, -x)$.

3D rotation by ccw angle ϕ around axis $\mathbf{n}$: $\mathbf{r}' = \mathbf{r}\cos\phi + \mathbf{n}(\mathbf{n}\cdot\mathbf{r})(1-\cos\phi) + (\mathbf{n}\times\mathbf{r})\sin\phi$

Plane equation from 3 points . $N\cdot(x, y, z) = N\cdot A$, where N is normal: $N = (B-A)\times(C-A)$.

3D figures

Sphere	Volume $V = \frac{4}{3}\pi r^3$, surface area $S = 4\pi r^2$
	$x = \rho\sin\theta\cos\phi$, $y = \rho\sin\theta\sin\phi$, $z = \rho\cos\theta$, $\phi \in [-\pi, \pi]$, $\theta \in [0, \pi]$
Spherical section	Volume $V = \pi h^2(r - h/3)$, surface area $S = 2\pi rh$
Pyramid	Volume $V = \frac{1}{3}hS_{base}$
Cone	Volume $V = \frac{1}{3}\pi r^2 h$, lateral surface area $S = \pi r\sqrt{r^2 + h^2}$

Area of a simple polygon . $\frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i)$, where $x_n = x_0, y_n = y_0$.
Area is negative if the boundary is oriented clockwise.

Winding number . Shoot a ray from given point in an arbitrary direction. For each intersection of ray with polygon's side, add $+1$ if the side crosses it counterclockwise, and -1 if clockwise.

Convex Hull

```
struct Point { LL x,y; bool operator<(const Point &p)const{
return(x==p.x) ? y<p.y : x<p.x; } };
LL cross(const Point &O, const Point &A, const Point &B)
{ return (A.x-O.x) * (B.y-O.y) - (A.y-O.y) * (B.x-O.x); }
// Returns a list of convex hull in counter-clockwise order.
// In case of collinear points, only end-points are taken (not middle).
// The last point is the same as the first one.
vector<Point> ConvexHull(vector<Point> P) {
    int n = P.size(), k = 0; vector<Point> H(2*n);
    sort(P.begin(), P.end());
    for (int i = 0; i < n; i++)
    { while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; H[k++] = P[i]; }
    for (int i = n-2, t = k+1; i >= 0; i--)
    { while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--; H[k++] = P[i]; }
    H.resize(k); return H; }
```

Trigonometric identities

$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$ $\quad$ $\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$

$\sin(\alpha-\beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$ $\quad$ $\cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$

$\tan(\alpha+\beta) = \frac{\tan\alpha+\tan\beta}{1-\tan\alpha\tan\beta}$ $\quad$ $\sin 2\alpha = 2\sin\alpha\cos\alpha$, $\cos 2\alpha = \cos^2\alpha - \sin^2\alpha$

$\cos^2\alpha = \frac{1}{2}(1+\cos 2\alpha)$ $\quad$ $\sin^2\alpha = \frac{1}{2}(1-\cos 2\alpha)$

$\sin\alpha + \sin\beta = 2\sin\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$ $\quad$ $\cos\alpha + \cos\beta = 2\cos\frac{\alpha+\beta}{2}\cos\frac{\alpha-\beta}{2}$

$\sin\alpha - \sin\beta = 2\sin\frac{\alpha-\beta}{2}\cos\frac{\alpha+\beta}{2}$ $\quad$ $\cos\alpha - \cos\beta = -2\sin\frac{\alpha+\beta}{2}\sin\frac{\alpha-\beta}{2}$

$\tan\alpha + \tan\beta = \frac{\sin(\alpha+\beta)}{\cos\alpha\cos\beta}$ $\quad$ $\cot\alpha + \cot\beta = \frac{\sin(\alpha+\beta)}{\sin\alpha\sin\beta}$

$\sin\alpha\sin\beta = \frac{1}{2}[\cos(\alpha-\beta) - \cos(\alpha+\beta)]$ $\quad$ $\cos\alpha\cos\beta = \frac{1}{2}[\cos(\alpha-\beta) + \cos(\alpha+\beta)]$

$\sin\alpha\cos\beta = \frac{1}{2}[\sin(\alpha+\beta) + \sin(\alpha-beta)]$ $\quad$ $\sin' x = \cos x$, $\cos' x = -\sin x$

Law of sines: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R_{out}$. $\quad$ Inscribed/outscribed circles: $R_{out} = \frac{abc}{4S}$, $R_{in} = \frac{2S}{a+b+c}$

Law of cosines: $c^2 = a^2 + b^2 - 2ab\cos C$. $\quad$ Heron: $\sqrt{s(s-a)(s-b)(s-c)}$, $s = \frac{a+b+c}{2}$.

Law of tangents: $\frac{a+b}{a-b} = \frac{\tan[\frac{1}{2}(A+B)]}{\tan[\frac{1}{2}(A-B)]}$ $\quad$ Δ's area, given side and adjacent angles: $\frac{c^2}{2(\cot\alpha+\cot\beta)}$

Template for geometry

```
double eps = 1e-8;
struct point2d {
    double x,y;
    point2d operator-(point2d v) { return (point2d){x-v.x, y-v.y}; }
    point2d operator+(point2d v) { return (point2d){x+v.x, y+v.y}; }
};
struct point3d {
    double x,y,z;
    point3d operator-(point3d v) { return (point3d){x-v.x, y-v.y, z-v.z}; }
};
double trip(point3d a, point3d b, point3d c) {
    return a.x * ( b.y * c.z - b.z * c.y )
        - a.y *  ( b.x * c.z - b.z * c.x )
        + a.z *  ( b.x * c.y - b.y * c.x );
}
double len(point2d a) { return sqrt(a.x*a.x+a.y*a.y); }
double len(point3d a) { return sqrt(a.x*a.x+a.y*a.y+a.z*a.z); }
point3d cross(point3d a, point3d b) {
    return (point3d){
            a.y*b.z - a.z*b.y,
            a.z*b.x - a.x*b.z,
            a.x*b.y - a.y*b.x};
}
double dot(point3d a, point3d b) { return a.x*b.x + a.y*b.y + a.z*b.z; }
double dot(point2d a, point2d b) { return a.x*b.x + a.y*b.y; }
double cross(point2d a, point2d b) {
    return a.x*b.y - a.y*b.x;
}
point2d rotate(point2d c, double angle) {
    return (point2d) {
        c.x*cos(angle)-c.y*sin(angle),
        c.x*sin(angle)+c.y*cos(angle) };
}
//distance AB to C
double linePointDist(point2d A, point2d B, point2d C, bool isSegment){
    double dist = cross(B-A,C-A) / len(B-A);
    if(isSegment){
        if( dot(C-B,B-A) > eps) return len(B-C);
```

```
        if( dot(C-A,A-B) > eps) return len(A-C);
    }
    return fabs(dist);
}
struct Line { double A,B,C; };//Ax + By = C
Line makeline(point2d a1, point2d a2)
{
    Line ret = (Line){ a2.y-a1.y, a1.x-a2.x };
    ret.C = ret.A * a1.x + ret.B * a1.y;
    return ret;
}
double dist(Line l, point2d p)
{
    return fabs(p.x * l.A + p.y * l.B - l.C)/sqrt(l.A*l.A+l.B*l.B);;
}
Line rot90(Line l, point2d p)
{
    Line ret = (Line){ -l.B, l.A };
    ret.C = ret.A * p.x + ret.B * p.y;
    return ret;
}
point2d intersect(Line l1, Line l2)
{
    double det = l1.A*l2.B - l2.A*l1.B;
    if(fabs(det) < eps) det=0;//zero means parallel
    return (point2d) { (l2.B*l1.C - l1.B*l2.C)/det, (l1.A*l2.C - l2.A*l1.C)/det};
}
//for segment segment intersection, check additionally
//min(x1,x2) <= x <= max(x1,x2)
//min(y1,y2) <= x <= max(y1,y2)
bool segmentsIntersect( point2d A, point2d B, point2d C, point2d E )
{
    point2d in = intersect( makeline(A,B), makeline(C,E) );
    return linePointDist(A,B,in,true) < eps && linePointDist(C,E,in,true) < eps;
}
// get a line passing between two points
Line getmidline(point2d a, point2d b)
{
    point2d mid(a+b); mid.x/=2; mid.y/=2;
    return rot90( makeline(a,b), mid );
}
//reflect a point into it's "mirror" with repect to a line
point2d reflectPoint(Line l, point2d p)
{
    Line r = rot90(l, p);
    point2d Y=intersect(l,r);
    return Y-(p-Y);
}
```

Appendices (LU, Blossom, binary search, ASCII table)

LU decomposition

```
double A[N][N], B[N]; //input
```

```
double X[N], L[N][N], U[N][N], D[N];

for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        if(i == j)
            L[i][j] = 1.0;
        else
            L[i][j] = 0.0;

for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        U[i][j] = A[i][j];

for(k=0;k<N-1;k++)
{
    for(i=k+1;i<N;i++)
    {
        double m=U[i][k]/U[k][k];
        L[i][k] = m;
        for(j=k+1;j<N;j++)
        {
            U[i][j] -= m * U[k][j];
        }
    }
}

D[0] = B[0] / L[0][0];
for(i=1;i<N;i++)
{
    double s = 0;
    for(j=0;j<i;j++)
    {
        s += L[i][j] * D[j];
    }
    D[i] = (B[i] - s)/L[i][i];
}

X[N-1] = D[N-1] / U[N-1][N-1];
for(i=N-2;i>=0;i--)
{
    double s = 0;
    for(j=i+1;j<N;j++)
    {
        s += U[i][j] * X[j];
    }
    X[i] = (D[i] - s)/U[i][i];
}
```

Blossoms

```
int lf[205],mat[205][205],n;
VVI adj;
VI vis,inactive,match;
```

```
int N;
bool dfs(int x,VI &blossom){
    if(inactive[x]) return false;
    int i,y;
    vis[x] = 0;
    for(i = adj[x].size()-1;i>=0;i--){
        y = adj[x][i];
        if(inactive[y]) continue;
        if(vis[y]==-1){
            vis[y] = 1;
            if(match[y]==-1 || dfs(match[y],blossom)){
                match[y] = x;
                match[x] = y;
                return true;
            }
        }
        if(vis[y]==0 || blossom.size()){
            blossom.push_back(y);
            blossom.push_back(x);
            if(blossom[0]==x){
                match[x] = -1;
                return true;
            }
            return false;
        }
    }
    return false;
}
bool augment(){
    VI blossom,mark;

    int i,j,k,s,x;
    for(i = 0;i<N;i++){
        if(match[i]!=-1) continue;
        blossom.clear();
        vis = VI(N+1,-1);
        if(!dfs(i,blossom)) continue;
        s = blossom.size();
        if(s==0) return true;


        mark = VI(N+1,-1);
        for(j = 0;j<s-1;j++){
            for(k = adj[blossom[j]].size()-1;k>=0;k--)
                mark[adj[blossom[j]][k]] = j;
        }
        for(j = 0;j<s-1;j++){
            mark[blossom[j]] = -1;
            inactive[blossom[j]] = 1;
        }
        adj[N].clear();
        for(j = 0;j<N;j++){
```

```
            if(mark[j]!=-1)
                adj[N].pb(j),adj[j].pb(N);
        }
        match[N] = -1;
        N++;
        if(!augment()) return false;
        N--;

        for(j = 0;j<N;j++){
            if(mark[j]!=-1) adj[j].pop_back();
        }
        for(j = 0;j<s-1;j++){
            inactive[blossom[j]] = 0;
        }

        x = match[N];
        if(x!=-1){
            if(mark[x]!=-1){
                j = mark[x];
                match[blossom[j]] = x;
                match[x] = blossom[j];
                if(j & 1)
                    for(k = j+1;k<s;k+=2) {
                        match[blossom[k]] = blossom[k+1];
                        match[blossom[k+1]] = blossom[k];
                    }
                else
                    for(k = 0;k<j;k+=2) {
                        match[blossom[k]] = blossom[k+1];
                        match[blossom[k+1]] = blossom[k];
                    }
            }
        }

        return true;
    }
    return false;
}

int edmond(int n){
    int i,j,ret = 0;
    N = n;
    adj = VVI(2*N+1);
    for(i = 0;i<n;i++){
        for(j = i+1;j<n;j++){
            if(mat[i][j]){
                adj[i].pb(j);
                adj[j].pb(i);
            }
        }
    }
    match = VI(2*N+1,-1);
```

```
    inactive = VI(2*N+1);
    while(augment()) ret++;
    return ret;
}
```

Binary search

```
/*LE*/
while(l <= h){ m = (l+h) / 2; if(works(m))  l=m+1; else h=m-1; }
return l-1;
/*GE*/
while(l <= h){ m = (l+h) / 2; if(works(m))  h=m-1; else l=m+1; }
return h+1;
```