Zoids ACM-ICPC Notebook 2018 (C++)

Contents

1 Data Structures

1.1 Binary Indexed Tree (BIT)

```cpp
int BIT[N];

void update(int x, int n, int add) {
	for (; x <= n; x += x&-x) {
		BIT[x] += add;
	}
}

int query(int x) {
	int sum = 0;
	for (; x; x -= x&-x) {
		sum += BIT[x];
	}
	return sum;
}

int Query(int l, int r) {
	return query(r) - query(l - 1);
}
```

1.2 SegmentTree

```cpp
#define LEFT(x) (2*x)
#define RIGHT(x) (2*x + 1)
int tree[N << 2];
int num[N];

void buildST(int node, int b, int e) {
	if (b == e) {
		tree[node] = num[b];
		return;
	}
	int me = (b + e) >> 1;
	int leftChild = LEFT(node), rightChild = RIGHT(node);
	buildST(leftChild, b, me);
	buildST(rightChild, me + 1, e);
	tree[node] = tree[leftChild] + tree[rightChild];
}

void update(int node, int b, int e, int pos, int newVal) {
	if (b == e) {
		tree[node] = newVal;
		return;
	}
	int me = (b + e) >> 1;
	int leftChild = LEFT(node), rightChild = RIGHT(node);
	if (pos <= me) update(leftChild, b, me , pos, newVal);
	else update(rightChild, me + 1, e, pos, newVal);
	tree[node] = tree[leftChild] + tree[rightChild];
}

int query(int node, int b, int e, int l, int r) {
	if (r < b || e < l) return 0;
	if (l <= b && e <= r) return tree[node];
	int me = (b + e) >> 1;
	int leftChild = LEFT(node), rightChild = RIGHT(node);
	return query(leftChild, b, me, l, r) + query(rightChild, me + 1, e, l , r);
}
```

1.3 SegmentTree + LazyPropagation

```cpp
#define LEFT(x) (2*x)
#define RIGHT(x) (2*x + 1)
int tree[N << 2];
int lazy[N << 2];
int num[N];

void buildST(int node, int b, int e) {
	if (b == e) {
		tree[node] = num[b];
		return;
	}
	int me = (b + e) >> 1;
	int leftChild = LEFT(node), rightChild = RIGHT(node);
	buildST(leftChild, b, me);
```

```
        buildST(rightChild, me + 1, e);
        tree[node] = tree[leftChild] + tree[rightChild];
}

void aplicate(int node, int b, int e) {
        if (!lazy[node]) return;
        tree[node] += (e - b + 1)*lazy[node];
        if (b != e) {
                int leftChild = LEFT(node), rightChild = RIGHT(node);
                lazy[leftChild] += lazy[node];
                lazy[rightChild] += lazy[node];
        }
        lazy[node] = 0;
}

void update(int node, int b, int e, int l, int r, int add) {
        aplicate(node, b, e);
        if (r < b || e < l) return;
        if (l <= b && e <= r) {
                lazy[node] = add;
                aplicate(node, b, e);
                return;
        }
        int me = (b + e) >> 1;
        int leftChild = LEFT(node), rightChild = RIGHT(node);
        update(leftChild, b, me , l, r, add);
        update(rightChild, me + 1, e, l, r, add);
        tree[node] = tree[leftChild] + tree[rightChild];
}

int query(int node, int b, int e, int l, int r) {
        if (r < b || e < l) return 0;
        aplicate(node, b, e);
        if (l <= b && e <= r) return tree[node];
        int me = (b + e) >> 1;
        int leftChild = LEFT(node), rightChild = RIGHT(node);
        return query(leftChild, b, me, l, r) + query(rightChild, me + 1, e, l , r);
}
```

1.4 Implicit Segment Tree

```
typedef long long Long;

struct Node {
        Node* left = NULL;
        Node* right = NULL;
        Long ans = 0;
        node() {}
};

void update(Node* node, Long b, Long e, Long pos, Long add) {
        if (b == e) {
                node->ans += add;
                return;
        }
        Long me = (b + e) >> 1;
        if (!node->left) node->left = new Node;
        if (!node->right) node->right = new Node;
        if (pos <= me) update(node->left, b, me, pos, add);
        else update(node->right, me + 1, e, pos, add);
        node->ans = node->left->ans + node->right->ans;
}

Long query(Node *node, Long b, Long e, Long l, Long r) {
        if (r < b || e < l) return 0;
        if (l <= b && e <= r) return node->ans;
        Long me = (b + e) >> 1;
        Long q1 = node->left? query(node->left, b, me, l, r): 0;
        Long q2 = node->right? query(node->right, me + 1, e, l, r): 0;
        return q1 + q2;
}

Node *tree = new Node;
```

1.5 ImplicitSegTree with Lazy Propagation

```
typedef long long Long;

struct Node {
        Node* left = NULL;
        Node* right = NULL;
        int ans = 0;
        bool lazy = false;
        node() {}
};

void aplicate(Node *node, Long b, Long e) {
        if (!node->lazy) return;
        node->ans = (e - b + 1 - node->ans)%mod;
        if (b != e) {
                if (!node->left) node->left = new Node;
                if (!node->right) node->right = new Node;
                node->left->lazy ^= 1;
                node->right->lazy ^= 1;
        }
        node->lazy = false;
}

void update(Node* node, Long b, Long e, Long l, Long r) {
        aplicate(node, b, e);
        if (r < b || e < l) return;
        if (l <= b && e <= r) {
                node->lazy = true;
                aplicate(node, b, e);
                return;
        }
        Long me = (b + e) >> 1;
        if (!node->left) node->left = new Node;
        if (!node->right) node->right = new Node;
        update(node->left, b, me, l, r);
        update(node->right, me + 1, e, l, r);
        node->ans = (node->left->ans + node->right->ans)%mod;
}

Long query(Node *node, Long b, Long e, Long l, Long r) {
        if (r < b || e < l) return 0;
        aplicate(node, b, e);
        if (l <= b && e <= r) return node->ans;
        Long me = (b + e) >> 1;
        if (!node->left) node->left = new Node;
        if (!node->right) node->right = new Node;
        int q1 = query(node->left, b, me, l, r);
        int q2 = query(node->right, me + 1, e, l, r);
        return (q1 + q2)%mod;
}

Node *tree[2];

void initTrees(int n) {
        for (int i = 0; i < n; i ++) {
                tree[i] = new Node;
        }
}
```

1.6 Persistant Segment Tree

```
struct Node {
        Node *left = NULL;
        Node *right = NULL;
        int cant = 0;
        Node() {}
        Node(int cant, Node *l, Node *r):
                cant(cant), left(l), right(r) {}
};

Node *tree[N];
Node *null;

Node* insert(Node *node, int b, int e, int pos) {
        if (pos < b || e < pos) return node;
        if (b == e) return new Node(node->cant + 1, null, null);
        int me = (b + e) >> 1;
        Node *l = insert(node->left, b, me, pos);
        Node *r = insert(node->right, me + 1, e, pos);
        return new Node(node->cant + 1, l, r);
}

Pair query(Node *node1, Node *node2, int b, int e, int k) {
        if (b == e) return {node1->cant - node2->cant, b};
        int me = (b + e) >> 1;
        int cantLeft = node1->left->cant - node2->left->cant;
        if (k <= cantLeft)
                return query(node1->left, node2->left, b, me, k);
        return query(node1->right, node2->right, me + 1, e, k - cantLeft);
}

int query2(Node *node1, Node *node2, int b, int e, int l, int r) {
        if (r < b || e < l) return 0;
        if (l <= b && e <= r) return node1->cant - node2->cant;
        int me = (b + e) >> 1;
```

```
        int q1 = query2(node1->left, node2->left, b, me, l, r);
        int q2 = query2(node1->right, node2->right, me + 1, e, l, r);
        return q1 + q2;
}

/************************************/
int num[N];

void buildTrees(int n) {
        null = new Node;
        null->left = null;
        null->right = null;
        tree[0] = null;
        for (int i = 1; i <= n; i ++) {
                tree[i] = insert(tree[i - 1], 0, INF, num[i]);
        }
}
```

1.7 SparseTable1

```
int num[N];
int st[N][logN];

void buildST(int n) {
        for (int i = 1; i <= n; i ++) {
                st[i][0] = num[i];
        }
        for (int loglen = 1, len = 2; len <= n; loglen ++, len <<= 1) {
                for (int i = 1; i + len -1 <= n; i ++) {
                        st[i][loglen] = min(st[i][loglen - 1], st[i + len/2][loglen - 1]);
                }
        }
}

int query(int l, int r) {
        int loglen = 31 - __builtin_clz(r - l + 1);
        int len = 1 << loglen;
        return min(st[l][loglen], st[r - len + 1][loglen]);
}
```

1.8 SparseTable2

```
int num[N];
int st[N][logN];

void buildST(int n) {
        for (int i = 1; i <= n; i ++) {
                st[i][0] = num[i];
        }
        for (int loglen = 1, len = 2; len <= n; loglen ++, len <<= 1) {
                for (int i = 1; i + len -1 <= n; i ++) {
                        st[i][loglen] = min(st[i][loglen - 1], st[i + len/2][loglen - 1]);
                }
        }
}

int query(int l, int r) {
        int len = r - l + 1;
        int ans = INF;
        for (int loglen = 31 - __builtin_clz(r - l + 1); loglen >= 0; loglen --) {
                if ((len >> loglen)&1) {
                        ans = min(ans, st[l][loglen]);
                        l += 1 << loglen;
                }
        }
        return ans;
}
```

1.9 Joshua's Segment Tree with Lazy (APIsh)

```
// tested on http://codeforces.com/contest/718/submission/34911387
typedef pair<int, int> Pair;

const int MAXN = (int)1e5 + 5;
const int MAXSIZE = 2;
const Long MOD = (Long)1e9 + 7;

const int size = 2;

vector<Long> ar;

struct Matrix
{
        Long X[MAXSIZE][MAXSIZE];

        Matrix ()
        {
        memset(X, 0, sizeof(X));
        }
        Matrix (int k)
        {
                memset(X, 0, sizeof(X));

                for(int i=0; i<size; i++)
                        X[i][i] = k;
        }
        void show ()
        {
                for (int i = 0; i < size; i++) {
                        for (int j = 0; j < size; j++) {
                                cout << X[i][j] << " ";
                        }
                        puts("");
                }
        }
} MA;

Matrix operator *(Matrix &A, Matrix &B)
{
        Matrix M;

        for(int i=0; i<size; i++)
        {
                for(int j=0; j<size; j++)
                {
                        long long tmp = 0;
                        for(int k=0; k<size; k++)
                                tmp = (tmp + ((A.X[i][k] * B.X[k][j])%MOD))%MOD;
                        M.X[i][j] = tmp;
                }
        }

        return M;
}

void mulInplace(Matrix &A, Matrix &B)
{
        Matrix M;

        for(int i=0; i<size; i++)
        {
                for(int j=0; j<size; j++)
                {
                        long long tmp = 0;
                        for(int k=0; k<size; k++)
                                tmp = (tmp + ((A.X[i][k] * B.X[k][j])%MOD))%MOD;
                        M.X[i][j] = tmp;
                }
        }

    A = M;
}

Matrix pows[64];
bool haspow[64];

Matrix pow(Matrix x, long long n)
{
        Matrix P(1);
    int cnt = 0;
        while(n)
        {
        if (haspow[cnt]) {
            if (n & 1) mulInplace(P, pows[cnt]);
        } else {
            haspow[cnt] = 1;
            if (cnt == 0) pows[cnt] = x;
            else pows[cnt] = pows[cnt - 1] * pows[cnt - 1];

            if (n & 1) mulInplace(P, pows[cnt]);
        }
                n >>= 1;
        cnt++;
        }

        return P;
}

void initA() {
    Matrix m;
```

```
    m.X[0][0] = 1;
    m.X[0][1] = 1;
    m.X[1][0] = 1;
    m.X[1][1] = 0;
    MA = m;
}

struct LazyNode{
    //contiene la informacion para actualizar Node
    Matrix m;
        LazyNode()
        {
                //elemento neutro:
        m = Matrix(1);
        }
        void operator +=(LazyNode &ln)
        {
        mulInplace(m, ln.m);
        }
};
void m42(Matrix &m, pair<Long, Long> &f) {
    Long f0 = f.first;
    Long f1 = f.second;
    Long nf0 = (((m.X[0][0]*f0)%MOD) + ((m.X[0][1]*f1)%MOD))%MOD;
    Long nf1 = (((m.X[1][0]*f0)%MOD) + ((m.X[1][1]*f1)%MOD))%MOD;
    f.first = nf0;
    f.second = nf1;
}

struct Node{
    pair<Long, Long> f;
    Node(){
        //elemento neutro:
        f.first = 0;
        f.second = 0;
    }
        void operator +=(LazyNode &ln)
        {
        m42(ln.m, f);
        }
        Node operator+( const Node &a) const {
            Node c;
                c.f.first = (f.first + a.f.first)%MOD;
                c.f.second = (f.second + a.f.second)%MOD;
            return c;
        }
};
struct ST{
    Node T[ MAXN * 4 ];
    LazyNode LazyT[ MAXN * 4 ];
    int n;
    ST(){}
    ST( int tam ){
        n = tam;
        build_tree( 0 , 0 , n - 1 );
    }
    // for reusing this structure
    void setSizeAndBuild( int tam ){
        n = tam;
        build_tree( 0 , 0 , n - 1 );
    }
    void build_tree( int node , int a , int b ){
        if( a == b ){
            LazyT[ node ] = LazyNode();

            //inicializando el elemento 'a'
            Long po = ar[a];
            pair<Long, Long> ini = make_pair(1, 0);

            Matrix m = pow(MA, po - 1);
            m42(m, ini);
            T[ node ].f = ini;
            return;
        }
        build_tree( ((node<<1) + 1) , a , ((a+b)>>1) ) , build_tree( ((node<<1) + 2) , ((a+b)>>1) + 1
             , b );
        T[ node ] = T[ ((node<<1) + 1) ] + T[ ((node<<1) + 2) ];
        LazyT[ node ] = LazyNode();
    }
    void push( int node , int a , int b ){
        T[ node ] += LazyT[ node ];
        if( a != b ){
            LazyT[ node*2 + 1 ] += LazyT[ node ];
                        LazyT[ node*2 + 2 ] += LazyT[ node ];
        }

        LazyT[ node ] = LazyNode();
    }
    Node query( int node , int a , int b , int lo , int hi ){
        push( node , a , b );
        if( lo > b || a > hi ) return Node();
        if( a >= lo && hi >= b ) return T[ node ];
        return query( ((node<<1) + 1) , a , ((a+b)>>1) , lo , hi ) + query( ((node<<1) + 2) , ((a+b)
              >>1) + 1  , b , lo , hi );
    }
    void update( int node , int a , int b , int lo , int hi, const LazyNode& val){
        push( node , a , b );
        if( lo > b || a > hi ) return ;
        if( a >= lo && hi >= b ) {
            LazyT[ node ] = val;
            push( node , a , b );
            return;
        }
        update( ((node<<1) + 1) , a , ((a+b)>>1) , lo , hi , val);
        update( ((node<<1) + 2) , ((a+b)>>1) + 1  , b , lo , hi , val);
        T[ node ] = T[ ((node<<1) + 1) ] + T[ ((node<<1) + 2) ] ;
    }
    Node query( int lo , int hi ){
        return query( 0 , 0 , n - 1 , lo , hi );

    }
    void update( int lo , int hi ,const LazyNode& val){
        update( 0 , 0 , n - 1 , lo , hi , val );
    }
}st;

int main() {
    initA();

    int n, m;
    scanf("%d%d", &n, &m);
    REP (i, n) {
        int x;
        scanf("%d", &x);
        ar.push_back(x);
    }
        st.setSizeAndBuild(n);

    REP (i, m) {
        int tp, l, r, x;
        scanf("%d%d%d", &tp, &l, &r);
        l--;r--;
        if (tp == 1) {
            scanf("%d", &x);
            LazyNode ln;
            ln.m = pow(MA, x);
            st.update(l, r , ln);
        } else {
            Node node = st.query(l, r);
            printf("%d\n", (int)node.f.first);
        }
    }
}
```

2 Math

2.1 Extended Euclid's Algorithm

```
// tested on https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=
      show_problem&problem=1045

struct EuclidReturn{
    Long u , v , d;
    EuclidReturn( Long u , Long v, Long d ) : u( u ) , v( v ) , d( d ) {}
};

EuclidReturn Extended_Euclid( Long a , Long b){
    if( b == 0 ) return EuclidReturn( 1 , 0 , a );
    EuclidReturn aux = Extended_Euclid( b , a%b );
    Long v = aux.u - (a/b)*aux.v;
    return EuclidReturn( aux.v , v , aux.d );
}
```

2.2 PollardRho + MillerRabin

```
// tested on https://uva.onlinejudge.org/index.php?option=onlinejudge&Itemid=99999999&category=791&
      page=show_problem&problem=2471
typedef unsigned long long ull;
typedef vector<ull> vull;

struct Pollard_Rho
{
```

```
    ull q;
    vull v;
    Pollard_Rho(){}
    Pollard_Rho( ull x ) {
            q = x;
    }
    ull gcd(ull a, ull b){
        if(b == 0) return a;
        return gcd( b , a % b );
    }
    ull mul(ull a,ull b,ull c){
        ull x = 0, y = a % c;
        while(b > 0){
            if(b%2 == 1){
                x = (x+y)%c;
            }
            y = (y*2)%c;
            b /= 2;
        }
        return x%c;
    }
    ull modd(ull a,ull b,ull c){
        ull x=1,y=a;
        while(b > 0){
            if(b%2 == 1){
                x=mul(x,y,c);
            }
            y = mul(y,y,c);
            b /= 2;
        }
        return x%c;
    }
    bool Miller(ull p,int iteration){ // isPrime? O(iteration * (log(n)) ^ 3 )
        if(p<2){
            return false;
        }
        if(p!=2 && p%2==0){
            return false;
        }
        ull s=p-1;
        while(s%2==0){
            s/=2;
        }
        for(int i=0;i<iteration;i++){
            ull a=rand()%(p-1)+1,temp=s;
            ull mod=modd(a,temp,p);
            while(temp!=p-1 && mod!=1 && mod!=p-1){
                mod=mul(mod,mod,p);
                temp *= 2;
            }
            if(mod!=p-1 && temp%2==0){
                return false;
            }
        }
        return true;
    }
    ull rho(ull n){
        if( n % 2 == 0 ) return 2;
        ull x = 2 , y = 2 , d = 1;
        int c = rand() % n + 1;
        while( d == 1 ){
            x = (mul( x , x , n ) + c)%n;
            y = (mul( y , y , n ) + c)%n;
            y = (mul( y , y , n ) + c)%n;
            if( x - y >= 0 ) d = gcd( x - y , n );
            else d = gcd( y - x , n );
        }
        return d;
    }
    void factor(ull n){
        if (n == 1) return;
        if( Miller(n , 10) ){ // 10 is good enough for most cases
            if(q != n) v.push_back(n);
            return;
        }
        ull divisor = rho(n);
        factor(divisor);
        factor(n/divisor);
    }
    vull primefact( ull num ) // O(num ^ (1/4))
    {
            v.clear();
            q = num;
            factor( num );
            sort( ALL(v) );
            if( v.empty() ) // primos o 1
                    v.push_back( num );
            return v;
    }
map<ull, int> primeFactorsDescomposition(ull num) { // returns pairs of {prime, exponent}
    vull pf = primefact(num);
    map<ull, int> pd; // prime descomposition
        for (int i = 0; i < (int)pf.size(); i++) {
            pd[pf[i]]++;
        }
        return pd;
    }
};
```

2.3 Sieve

```
const int MAXN = (int)1e5;

bool prime[MAXN+1];

void sieve() {// O(nlglgn)
        memset(prime, true, sizeof(prime));

        prime[0] = false;
        prime[1] = false;

        for(int i=2; i*i<=MAXN; i++)
                if(prime[i])
                        for(int j=i*i; j<=MAXN; j+=i)
                                prime[j]=false;
}

const int MAXN = (int)3e8;

bitset <MAXN+1> notprime;

void sieve() { // careful as pair numbers are not marked as notprime
    for(int i=3; i*i<=MAXN; i+=2)
        if(!notprime[i])
                for(int j=i*i; j<=MAXN; j+=(i<<1))
                        notprime[j] = true;
}
```

2.4 Fermat's Little Theorem

```
if P is prime then:
a ^ p = a mod p

And if a is not divisible by p then:
a ^ (p - 1) = 1 mod p
```

2.5 Euler's Theorem

```
a ^ phi(n) = 1 mod n  iff (if and only if)  n and a are coprimes

Bonus:

let n = p1 ^ a1 * p2 ^ a2 ...

phi(n) = (p1 - 1) * p1 ^ (a1 - 1) * (p2 - 1) * p2 ^ (a2 - 1) ...

phi(n) = n * (for each distinct prime 'p' that divides n: the product of (1 - 1 / p))
```

2.6 Chinese Remainder Theorem

```
Dados k enteros positivos {ni}, tales que ni y nj son coprimos (i!=j).
Para cualquier {ai}, existe x tal que:

x % ni = ai

Todas las soluciones son congruentes modulo N = n1*n2*...*nk

r*ni + s*N/ni = 1 -> ei = s*N/ni   -> ei % nj = 0
                     r*ni + ei = 1 -> ei % ni = 1

x = a1*e1 + a2*e2 + ... + ak*ek

// ax = 1(mod n)
Long modular_inverse(Long a, Long n){
    EuclidReturn aux = Extended_Euclid(a,n);
    return ((aux.u/aux.d)%n+n)%n;
}
```

```
// rem y mod tienen el mismo numero de elementos
long long chinese_remainder(vector<Long> rem, vector<Long> mod){
    long long ans = rem[0],m = mod[0];
    int n = rem.size();

    for(int i=1;i<n;++i){
        int a = modular_inverse(m,mod[i]);
        int b = modular_inverse(mod[i],m);
        ans = (ans*b*mod[i]+rem[i]*a*m)%(m*mod[i]);
        m *= mod[i];
    }

    return ans;
}


Chinese Remainder Theorem: Strong Form
======================================
(thanks to https://forthright48.com/2017/11/chinese-remainder-theorem-part-2-non-coprime-moduli.html)

Given two sequences of numbers A=[a1,a2,   ,an] and M=[m1,m2,   ,mn], a solution to x exists for the
      following n congruence equations:

 x  a 1 (mod m1)
 x  a 2 (mod m2)

 x  a n (mod mn)


if,  a i  a j  (mod GCD(mi,mj)) and the solution will be unique modulo L=LCM(m1,m2,   ,mn)

Implementation O(n * log(L)):

// tested on https://open.kattis.com/problems/generalchineseremainder

/**
    A CRT solver which works even when moduli are not pairwise coprime
    1. Add equations using addEquation() method
    2. Call solve() to get {x, N} pair, where x is the unique solution modulo N. (returns -1, -1 if no
         solution)
    Assumptions:
        1. LCM of all mods will fit into long long.
*/
class ChineseRemainderTheorem {
    typedef long long vlong;
    typedef pair<vlong,vlong> pll;
    typedef __int128 overflowtype;
    //typedef long long overflowtype;

    /** CRT Equations stored as pairs of vector. See addEquation()*/
    vector<pll> equations;

public:
    void clear() {
        equations.clear();
    }

    /** Add equation of the form x = r (mod m)*/
    void addEquation( vlong r, vlong m ) {
        equations.push_back({r, m});
    }
    pll solve() {
        if (equations.size() == 0) return {-1,-1}; /// No equations to solve

        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        /** Initially x = a_0 (mod m_0)*/

        /** Merge the solution with remaining equations */
        for ( int i = 1; i < equations.size(); i++ ) {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;

            EuclidReturn euclidReturn1 = Extended_Euclid(m1, m2);
            vlong g = euclidReturn1.d;
            if ( a1 % g != a2 % g ) return {-1,-1}; /// Conflict in equations

            /** Merge the two equations*/
            vlong p, q;
            EuclidReturn euclidReturn = Extended_Euclid(m1/g, m2/g);
            p = euclidReturn.u;
            q = euclidReturn.v;

            vlong mod = m1 / g * m2;
            vlong x = ( (overflowtype)a1 * (m2/g) % mod *q % mod + (overflowtype)a2 * (m1/g) % mod * p
                  % mod ) % mod;

            /** Merged equation*/
            a1 = x;
            if ( a1 < 0 ) a1 += mod;
            m1 = mod;
        }
        return {a1, m1};
    }
};
```

2.7 Phi Sieve

```
// not tested, I just use the prime decomposition to obtain phi

#define MAXN 10000
int phi[MAXN + 1]
for(i = 1; i <= MAXN; ++i) phi[i] = i;
for(i = 1; i <= MAXN; ++i) for (j = i * 2; j <= MAXN; j += i) phi[j] -= phi[i];

#define MAXN 3000000
int phi[MAXN + 1], prime[MAXN/10], sz;
bitset <MAXN + 1> mark;

for (int i = 2; i <= MAXN; i++ ){
        if(!mark[i]){
                phi[i] = i-1;
                prime[sz++]= i;
        }
        for (int j=0; j<sz && prime[j]*i <= MAXN; j++ ){
                mark[prime[j]*i]=1;
                if(i%prime[j]==0){
                        phi[i*prime[j]] = phi[i]*prime[j];
                        break;
                }
                else phi[i*prime[j]] = phi[i]*(prime[j]-1 );
        }
}
```

2.8 Linear Sieve and logarithmic factorization

```
// tested on https://www.spoj.com/problems/FACTCG2/

// O(N)
// Comentarios generales :
// p[i] para 0 < i indica el valor del primo i-esimo
//      Ejm : p[1] = 2 , p[2] = 3 ....
// A[i] indica que el menor factor primo de i es el primo A[i] - esimo
//      Ejm: si 15 = 3*5 , entonces A[12] = 2 porque el menor factor primo de 12 es 3 y 3 es el 2do
      primo
const int MAXN = (int) 1e7 + 5;
int A[MAXN + 1], p[MAXN + 1], pc = 0;
void sieve()
{
    for(int i=2; i<=MAXN; i++){
        if(!A[i]) p[A[i] = ++pc] = i;
        for(int j=1; j<=A[i] && (long long)i*p[j]<=MAXN; j++)
            A[i*p[j]] = j;
    }
}

vector<int> primeFact(int n) { // O(log(n))
    vector<int> v;
    while (n != 1) {
        v.push_back(p[A[n]]);
        n /= p[A[n]];
    }
    return v;
}
```

2.9 Fast Fourier Transform

```
// tested on https://www.spoj.com/problems/POLYMUL/
// multiply two polynomials (use the multiply function) O(n * log(n))
//CDC_MOREFB
#define MOD 99991LL

typedef long double ld;
typedef vector< ld > vld;
typedef vector< vld > vvld;
typedef long long ll;
typedef pair< int , int > pii;
typedef vector< int > vi;
typedef vector< vi > vvi;
```

```
ld PI = acos( (ld)(-1.0) );
ll pow( ll a , ll b , ll c ){
        ll ans = 1;
        while( b ){
                if( b & 1 ) ans = (ans * a)%c;
                a = (a * a)%c;
                b >>= 1;
        }
        return ans;
}
ll mod_inv( ll a , ll p ){ return pow(a , p - 2 , p);}

typedef complex<ld> base;

void fix( base &x ){
        if(abs(x.imag()) < 1e-16 ){
                x = base( (((ll)round(x.real()))%MOD + MOD)%MOD , 0);
        }
}
void fft (vector<base> & a, bool invert) {
        int n = (int) a.size();

        for (int i=1, j=0; i<n; ++i) {
                int bit = n >> 1;
                for (; j>=bit; bit>>=1)
                        j -= bit;
                j += bit;
                if (i < j)
                        swap (a[i], a[j]);
        }

        for (int len=2; len<=n; len<<=1) {
                ld ang = 2.0 * PI /len * (invert ? -1 : 1);
                base wlen (cos(ang), sin(ang));
                for (int i=0; i<n; i+=len) {
                        base w (1);
                        for (int j=0; j<len/2; ++j) {
                                base u = a[i+j],  v = a[i+j+len/2] * w;
                                a[i+j] = u + v;
                                a[i+j+len/2] = u - v;
                                w *= wlen;
                        }
                }
        }
        if (invert)
                for (int i=0; i<n; ++i)
                        a[i] /= n;
}
void multiply (const vector<ld> & a, const vector<ld> & b, vector<ld> & res) {
        vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
        size_t n = 1;
        while (n < max (a.size(), b.size()))  n <<= 1;
        n <<= 1;
        fa.resize (n),  fb.resize (n);

        fft (fa, false),  fft (fb, false);
        for (size_t i=0; i<n; ++i)
                fa[i] *= fb[i];

        fft (fa, true);

        res.resize (n);
        for (size_t i=0; i<n; ++i){
                // res[i] = (((ll)round( fa[i].real() ))%MOD + MOD)%MOD;
        res[i] = ((ll)round( fa[i].real() ));
        }
}

void impr( vi &x ){
        REP( i , SZ(x) ) printf( "%d%c" , x[ i ] , (i + 1 == SZ(x)) ? 10 : 32 );
}

vld rec( vvld &T , int lo , int hi ){
        if( lo == hi ) return T[ lo ];
        int mid = (lo + hi) >> 1;
        vld L = rec( T , lo , mid );
        vld R = rec( T , mid + 1 , hi );
        vld X;
        multiply( L , R , X );
        return X;
}
ll solve( ll base , vi &x , int n , int k ){
        // p( x ) = (x + base^v[0]) * ( x + base ^ v[1] ) ....
        vvld T( n );
        REP( i , n )
                T[ i ] = { (ld)pow(base , x[ i ] , MOD) , (ld)1.0 };

        vld v = rec( T , 0 , n - 1 );
        ld target = v[ n - k ];

        ll num = (((ll)round( target ))%MOD + MOD)%MOD;
        return num;
}

int main(){
        ll A = 55048LL , B = 44944LL , C = 22019LL;
        //f( n ) = C(A^n - B^n)
        int n , K;
        while( sc( n ) == 1 ){
                sc( K );
                vi x( n );
                REP( i , n ) sc( x[ i ] );

                ll SA = solve( A , x , n , K );
                ll SB = solve( B , x , n , K );
                printf( "%lld\n" , (C * (SA - SB + MOD)%MOD)%MOD );
        }
}
```

2.10 Modular inverse

```
// ax = 1(mod n)
Long modular_inverse(Long a, Long n){
    EuclidReturn aux = Extended_Euclid(a,n);
    if (aux.d != 1) return -1; // not coprimes, so impossible to get a modular inverse
    return ((aux.u % n) + n)%n;
}
```

2.11 Mobius Function

```
// credits to Bryan

mobius(n) = 1 si n es libre de cuadrados y tiene un n m e r o par de factores primos distintos.
mobius(n) = -1 si n es libre de cuadrados y tiene un n m e r o impar de factores primos distintos.
mobius(n) = 0 si n es divisible por a l g n cuadrado.

/************************************/
int mobius( int num ) {
        int cantPrimes = fact( num ) ;
        if( cantPrimes == INF ) return 0 ; // INF is a flag for divisible by some square
        return (cantPrimes&1) ? -1 : 1 ;
}

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

2.12 Phillai Sieve

```
// phill[n] = sum of (for i from 1 to n: gcd(i, n) )
// also : phill[n] = sum of (for d a divisor of n: d * phi(n / d) )
long long phill[ N ] ;

void sievePhillai( int n ) {
        for( int num = 1 ; num <= n ; num ++ ) {
                for( int mult = num ; mult <= n ; mult += num ) {
                        phill[ mult ] += 1LL*num*phi[ mult/num ] ;
                }
        }
}
```

2.13 Lucas Theorem (small prime moduli and big n and k)

```
// Generalized lucas theorem
// tested on http://codeforces.com/gym/100637/problem/D
//http://codeforces.com/blog/entry/10271

struct EuclidReturn{
    Long u , v , d;
    EuclidReturn( Long u , Long v, Long d ) : u( u ) , v( v ) , d( d ) {}
};

EuclidReturn Extended_Euclid( Long a , Long b){
    if( b == 0 ) return EuclidReturn( 1 , 0 , a );
    EuclidReturn aux = Extended_Euclid( b , a%b );
    Long v = aux.u - (a/b)*aux.v;
    return EuclidReturn( aux.v , v , aux.d );
}
```

```
// ax = 1(mod n)
Long modular_inverse( Long a , Long n ){
    EuclidReturn aux = Extended_Euclid( a , n );
    return ((aux.u/aux.d)%n+n)%n;
}

Long chinese_remainder( vector<Long> &rem, vector<Long> &mod ){
        Long ans = rem[ 0 ] , m = mod[ 0 ];
    for( int i = 1 ; i < SZ(rem) ; ++i ){
        int a = modular_inverse( m , mod[ i ] );
        int b = modular_inverse( mod[ i ] , m );
        ans = ( ans * b * mod[ i ] + rem[ i ] * a * m)%( m*mod[ i ] );
        m *= mod[i];
    }
    return ans;
}


void primefact( int n , vector<Long> &p , vector<Long> &e , vector<Long> &pe ){
        for( int i = 2 ; i * i <= n ; ++i ){
                if( n % i == 0 ){
                        int exp = 0 , pot = 1;
                        while( n % i == 0 ){
                                n /= i;
                                exp ++;
                                pot *= i;
                        }
                        p.push_back( i ) , e.push_back( exp ) , pe.push_back( pot );
                }
        }
        if( n > 1 ) p.push_back( n ) , e.push_back( 1 ) , pe.push_back( n );
}

Long pow( Long a , Long b , Long c ){
        Long ans = 1;
        while( b ){
                if( b & 1 ) ans = (ans * a)%c;
                a = (a * a)%c;
                b >>= 1;
        }
        return ans;
}
Long factmod( Long n , Long p , Long pe ){
        if( n == 0 ) return 1;
        Long cpa = 1;
    Long ost = 1;
    for( Long i = 1; i <= pe; i++ ){
        if( i % p != 0 ) cpa = (cpa * i) % pe;
        if( i == (n % pe) ) ost = cpa;
    }
    cpa = pow(cpa, n / pe, pe);
    cpa = (cpa * ost) % pe;
    ost = factmod(n / p, p, pe);
    cpa = (cpa * ost) % pe;
    return cpa;
}
Long factst( Long a , Long b ){
        Long ans = 0;
        while( a ){
                ans += a / b;
                a /= b;
        }
        return ans;
}

Long solve( Long n , Long k , Long p , Long e , Long pe ){
        Long np = factmod( n , p , pe );
        Long kp = factmod( k , p , pe );
        Long nkp = factmod( n - k , p , pe );
        Long cnt = factst( n , p ) - factst( k , p ) - factst( n - k , p );
        if( cnt >= e ) return 0;
        Long r = ((np * modular_inverse( kp , pe ))% pe);
        r = (r * modular_inverse( nkp , pe ))%pe;
        REP( i , cnt ) r = (r * p) % pe;
        return r;
}

int main(){
        Long n , k , mod;
        while( cin >> n >> k >> mod ){
                vector<Long> p , e , pe;// pe = p ^ e
                primefact( mod , p , e , pe );
                vector<Long> rem;
                REP( i , SZ( p ) ) rem.push_back( solve( n , k , p[ i ] , e[ i ] , pe[ i ] ) );
                cout << chinese_remainder( rem ,  pe ) << '\n';
        }
}
```

2.14 Catalan, dearrangements and other formulas

```
// Series conocidas
// A000217       Triangular numbers: a(n) = C(n+1,2) = n(n+1)/2 = 0+1+2+...+n. 0, 1, 3, 6, 10, 15, 21,
       28 ...( 0 , 0 + 1 , 0 + 1 + 2 ,...)
// f* = (-1+sqrt( 8*x + 1 ))/2
// A000292 Tetrahedral (or triangular pyramidal) numbers: a(n) = C(n+2,3) = n*(n+1)*(n+2)/6. 0, 1, 4,
      10, 20, 35, 56, 84, 120.... ( 0 , 0 + 1 , 0 + 1 + 3 , 0 + 1 + 3 + 6 , ..)
// A000010 Euler totient function phi(n): count numbers <= n and prime to n.  1, 1, 2, 2, 4, 2, 6, 4,
      6, 4, 10, 4, 12, 6, 8, 8, 16, 6, 18, 8, 12, 10, 22, 8, 20, 12, 18, 12, 28, 8, 30, 16, 20, 16,
      24, 12, 36, 18, 24, 16, 40, 12, 42, 20, 24, 22, 46, 16, 42, 20, 32, 24, 52, 18, 40, 24, 36, 28,
      58, 16, 60, 30, 36, 32, 48, 20, 66, 32, 44
// binomial = combination
// A000108      Catalan numbers: C(n) = binomial(2n,n)/(n+1) = (2n)!/(n!(n+1)!). Also called Segner
      numbers.1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440,
      9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640,
      343059613650, 1289904147324, 4861946401452, 18367353072152, 69533550916004, 263747951750360,
      1002242216651368, 3814986502092304
/**

Let Cn be Catalan number of n:
Cn = binomial(2n, n)  - binomial(2n, n + 1)
* Cn is the number of Dyck words of length 2n. A Dyck word is a string consisting of n X's and n Y's
      such that no initial
segment of the string has more Y's than X's. For example, the following are the Dyck words of length
      6:
XXXYYY     XYXXYY     XYXYXY     XXYYXY     XXYXYY.
* Re-interpreting the symbol X as an open parenthesis and Y as a close parenthesis, Cn counts the
      number of expressions containing n pairs of parentheses which are correctly matched:
((()))     ()(())     ()()()     (())()     (()())
* Cn is the number of different ways n + 1 factors can be completely parenthesized (or the number of
      ways of associating n applications of a binary operator). For n = 3, for example, we have the
      following five different parenthesizations of four factors:
((ab)c)d     (a(bc))d     (ab)(cd)     a((bc)d)     a(b(cd))

* Successive applications of a binary operator can be represented in terms of a full binary tree. (A
      rooted binary tree is full if every vertex has either two children or no children.) It follows
      that Cn is the number of full binary trees with n + 1 leaves
* Cn is the number of monotonic lattice paths along the edges of a grid with n    n square cells,
      which do not pass above the diagonal. A monotonic path is one which starts in the lower left
      corner, finishes in the upper right corner, and consists entirely of edges pointing rightwards
      or upwards. Counting such paths is equivalent to counting Dyck words: X stands for "move right"
      and Y stands for "move up".
* A convex polygon with n + 2 sides can be cut into triangles by connecting vertices with non-crossing
       line segments (a form of polygon triangulation). The number of triangles formed is n and the
      number of different ways that this can be achieved is Cn. The following hexagons illustrate the
      case n = 4:
* Cn is the number of stack-sortable permutations of {1, ..., n}. A permutation w is called stack-
      sortable if S(w) = (1, ..., n), where S(w) is defined recursively as follows: write w = unv
      where n is the largest element in w and u and v are shorter sequences, and set S(w) = S(u)S(v)n,
       with S being the identity for one-element sequences.
* Cn is the number of permutations of {1, ..., n} that avoid the permutation pattern 123 (or,
      alternatively, any of the other patterns of length 3); that is, the number of permutations with
      no three-term increasing subsequence. For n = 3, these permutations are 132, 213, 231, 312 and
      321. For n = 4, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231,
       4312 and 4321.
* Cn is the number of noncrossing partitions of the set {1, ..., n}. A fortiori, Cn never exceeds the
      nth Bell number. Cn is also the number of noncrossing partitions of the set {1, ..., 2n} in
      which every block is of size 2. The conjunction of these two facts may be used in a proof by
      mathematical induction that all of the free cumulants of degree more than 2 of the Wigner
      semicircle law are zero. This law is important in free probability theory and the theory of
      random matrices.
* Cn is the number of ways to tile a stairstep shape of height n with n rectangles.
* Cn is the number of ways that the vertices of a convex 2n-gon can be paired so that the line
      segments joining paired vertices do not intersect. This is precisely the condition that
      guarantees that the paired edges can be identified (sewn together) to form a closed surface of
      genus zero (a topological 2-sphere).


*/


// A000169  Number of labeled rooted trees with n nodes: n^(n-1).       1, 2, 9, 64, 625, 7776,
      117649, 2097152, 43046721, 1000000000, 25937424601, 743008370688, 23298085122481,
      793714773254144, 29192926025390625, 1152921504606846976, 48661191875666868481,
      2185911559738696531968, 104127350297911241532841, 5242880000000000000000000
// A006717  Number of toroidal semi-queens on a (2n+1) X (2n+1) board.  1, 3, 15, 133, 2025, 37851,
      1030367, 36362925, 1606008513, 87656896891, 5778121715415, 452794797220965, 41609568918940625

//Derangement In combinatorial mathematics, a derangement is a permutation of the elements of a set
      such that none of the elements appear in their original position.
// http://en.wikipedia.org/wiki/Derangement
// DP[ n ] = ( n - 1 ) * ( DP[ n - 1 ] + DP[ n - 2 ] ) , DP[ 0 ] = 1 , DP[ 1 ] = 0;   11282_UVA
```

3 Flows

3.1 Dinic (Also maximum bipartite matching)

```
// tested in at least 4 problems
struct flowGraph{
    // O (E * V ^ 2) => but you can expect a lot less in practice (up to 100 times better)
        // O (E * sqrt(V)) => on bipartite graphs or unit flow through nodes
    // O (min(V ^ (2/3), sqrt(E)) * E) => in network with unit capacities
        // memory = O(E + V)

        /**

        On bipartite graphs:
        //maximum independent set + maxflow = nodes
        //maximum independent set = minimun edge cover
        //maxflow = minimum vertex cover
        Grafos bipartitos:
        Any tree is 2-colorable.
        The following are equivalent:
        1. G is bipartite.
        2. G is 2-colorable.
        3. G has no cycles of odd length.

        Reconstruccion de Vertex Cover en grafo bipartito:
        DFS the residual graph and mark those nodes you visit,
        Answer is the nodes on the left that you don't visit and
        the nodes on the right that you visit.

        Reconstrucci n del Min-Cut:
        Hacer un BFS o DFS desde s (source) sobre el grafo residual y todos los nodos
        visitados s e r n parte del corte de s, las aristas que entren a alguno de estos nodos
        pero no hayan sido visitados por el DFS s e r n las que forman parte del corte.

        Dilworth Theorem(Max antichain = Min path cover)
        How to find a maxim chain
        OJO : El grafo tiene que ser un dag.


        */
        typedef Long flowtype;
    const flowtype INF = (flowtype)2e10;
        const int bfsINF = (1<<28);
        int n , m , s , t , E;
        vector<int> to , NEXT;//maxe * 2
        vector<flowtype> cap; //maxe * 2
        vector<int> last, now , dist;// maxv
        flowGraph(){}
        flowGraph( int n , int m , int s , int t ) {
                init(n, m, s, t);
        }
        void init( int n , int m , int s , int t ) {
                this->n = n;
                this->m = m;
                this->s = s;
                this->t = t;
                cap = vector<flowtype>( 2 * m + 5 );
                to = NEXT = vector<int>( 2 * m + 5 );
                now = dist = vector<int>( n + 5 );
                E = 0;
                last = vector<int>( n + 5 , -1 );
        }
        void add( int u , int v , flowtype uv , flowtype vu = 0 ){
                to[ E ] = v ; cap[ E ] = uv ; NEXT[ E ] = last[ u ] ; last[ u ] = E ++;
                to[ E ] = u ; cap[ E ] = vu ; NEXT[ E ] = last[ v ] ; last[ v ] = E ++;
        }
        bool bfs(){
                REP( i , n ) dist[ i ] = bfsINF;
                queue< int > Q;
                dist[ t ] = 0;
                Q.push( t );
                while( !Q.empty() ){
                        int u = Q.front() ; Q.pop();
                        for( int e = last[ u ] ; e != -1 ; e = NEXT[ e ] ){
                                int v = to[ e ];
                                if( cap[ e ^ 1 ] && dist[ v ] >= bfsINF ){
                                        dist[ v ] = dist[ u ] + 1;
                                        Q.push( v );
                                }
                        }
                }
                return dist[ s ] < bfsINF;
        }
        flowtype dfs( int u , flowtype f ){
                if( u == t ) return f;
                for( int &e = now[ u ] ; e != -1 ; e = NEXT[ e ] ){
                        int v = to[ e ];
                        if( cap[ e ] && dist[ u ] == dist[ v ] + 1 ){
                                flowtype ret = dfs( v  , min( f , cap[ e ] ) );
                                if( ret ){
                                        cap[ e ] -= ret;
                                        cap[ e ^ 1 ] += ret;
                                        return ret;
                                }
                        }
                }
                return 0;
        }

        flowtype maxFlow(){
                flowtype flow = 0;
                while( bfs() ){
                        REP( i , n ) now[ i ] = last[ i ];
                        while( 1 ){
                                flowtype f = dfs( s , INF );
                                if( !f ) break;
                                flow += f;
                        }
                }
                return flow;
        }

        /**
         * Gets residual capacity per edge
         * **/
        vector<pair<pair<int, int>, flowtype> > getResPerEdge() {
                vector<pair<pair<int, int>, flowtype> > res;
                REP (u, n) {
                        for( int e = last[ u ] ; e != -1 ; e = NEXT[ e ] ) {
                                int v = to[ e ];
                                res.push_back(make_pair(make_pair(u, v), cap[e]));
                        }
                }
                return res;
        }
}fg;
```

3.2 Maximum Flow with upper bound cost

```
// Plan-ChotaV2.cpp
//Codeforces Round #212 (Div. 2) E. Petya and Pipes
// accepted with V = 50, E = V ^ 2, K = 1000, cap[i][j] <= 1e6
typedef int Flow;
typedef int Cost;
const Flow INF = 0x3f3f3f3f;
struct Edge {
    int src, dst;
    Cost cst; // cost per unit of flow in this edge
    Flow cap;
    int rev;
    Edge(){}
    Edge( int src , int dst , Cost cst , Flow cap , int rev ) : src( src ) , dst( dst ) , cst( cst ) ,
         cap( cap ) , rev( rev ){}
};
bool operator<(const Edge a, const Edge b) {
    return a.cst>b.cst;
}

typedef vector<Edge> Edges;
typedef vector<Edges> Graph;

void add_edge( Graph&G , int u , int v , Flow c , Cost l ) {
    G[u].pb( Edge( u , v , l , c , G[v].size() ) );
    G[v].push_back( Edge( v , u , -l, 0 , (int)G[u].size() - 1 ) );
}
// returns the max_flow_mincost with cost <= K

pair< Flow, Cost > flow( Graph &G , int s , int t , int K = INF ) {
    int n = G.size();
    Flow flow = 0;
    Cost cost = 0;
    while( 1 ) {
        priority_queue< Edge > Q;
        vector< int > prev( n , -1 ), prev_num( n , -1 );
        vector< Cost > length( n , INF );
        Q.push( Edge( -1 , s , 0 , 0 , 0 ) );
        prev[ s ] = s;
        while( !Q.empty() ) {
            Edge e = Q.top(); Q.pop();
            int v = e.dst;
            for ( int i = 0 ; i < (int) G[v].size() ; i++ ) {
                if ( G[v][i].cap > 0 && length[ G[v][i].dst ] > e.cst + G[v][i].cst ) {
```

```
                prev[ G[v][i].dst ] = v;
                Q.push( Edge( v, G[v][i].dst , e.cst + G[v][i].cst , 0 , 0 ) );
                prev_num[ G[v][i].dst ] = i;
                length[ G[v][i].dst  ] = e.cst + G[v][i].cst;
            }
        }
    }
    if( prev[t] < 0 ) return make_pair( flow , cost );
    Flow mi = INF;
    Cost cst = 0;
    for( int v = t ; v != s ; v = prev[v] ) {
        mi = min( mi , G[prev[v]][prev_num[v]].cap );
        cst += G[prev[v]][prev_num[v]].cst;
    }
        if( cst > K ) return make_pair(flow, cost);
        if( cst != 0 ) mi = min(mi, K/cst);
        K -= cst*mi;
    cost+=cst*mi;

    for ( int v = t ; v != s ; v = prev[v] ) {
        Edge &e = G[prev[v]][prev_num[v]];
        e.cap -= mi;
        G[ e.dst ][ e.rev ].cap += mi;
    }
    flow+=mi;
  }
}
```

3.3 Minimun Cost Maximum Flow

```
// Plan-ChotaV2.cpp
// For no Integer Cost ( long double  ld )

//10746 UVA - Crime Wave - The Sequel
// assignment problem on a bipartite graph:
// n <= m <= 20 (n = nodes on the left, m = nodes on the right)
// unit flow on each edge
// cost is a real number
typedef int Flow;
typedef ld Cost;
const Flow INF = 0x3f3f3f3f;
struct Edge {
    int src, dst;
    Cost cst;
    Flow cap;
    int rev;
    Edge(){}
    Edge( int src , int dst , Cost cst , Flow cap , int rev ) : src( src ) , dst( dst ) , cst( cst ) ,
        cap( cap ) , rev( rev ){}
};
bool operator<(const Edge a, const Edge b) {
    return a.cst>b.cst;
}

typedef vector<Edge> Edges;
typedef vector<Edges> Graph;

void add_edge( Graph&G , int u , int v , Flow c , Cost l ) {
    G[u].pb( Edge( u , v , l , c , G[v].size() ) );
    G[v].push_back( Edge( v , u , -l, 0 , (int)G[u].size() - 1 ) );
}
pair< Flow, Cost > flow( Graph &G , int s , int t ) {
    int n = G.size();
    Flow flow = 0;
    Cost cost = 0;
    while( 1 ) {
        priority_queue< Edge > Q;
        vector< int > prev( n , -1 ), prev_num( n , -1 );
        vector< Cost > length( n , INF );
        Q.push( Edge( -1 , s , 0 , 0 , 0 ) );
        prev[ s ] = s;
        while( !Q.empty() ) {
            Edge e = Q.top(); Q.pop();
            int v = e.dst;
            for ( int i = 0 ; i < (int) G[v].size() ; i++ ) {
                if ( G[v][i].cap > 0 && length[ G[v][i].dst ] > e.cst + G[v][i].cst ) {
                    prev[ G[v][i].dst ] = v;
                    Q.push( Edge( v, G[v][i].dst , e.cst + G[v][i].cst , 0 , 0 ) );
                    prev_num[ G[v][i].dst ] = i;
                    length[ G[v][i].dst  ] = e.cst + G[v][i].cst;
                }
            }
        }
        if( prev[t] < 0 ) return make_pair( flow , cost );
        Flow mi = INF;
        Cost cst = 0;
        for( int v = t ; v != s ; v = prev[v] ) {
            mi = min( mi , G[prev[v]][prev_num[v]].cap );
            cst += G[prev[v]][prev_num[v]].cst;
        }

        cost+=cst*mi;

        for ( int v = t ; v != s ; v = prev[v] ) {
            Edge &e = G[prev[v]][prev_num[v]];
            e.cap -= mi;
            G[ e.dst ][ e.rev ].cap += mi;
        }
        flow+=mi;
    }
}
```

4 Graphs

4.1 Biconnected Components, bridges and articulation points O(E + V)

```
// tested on http://codeforces.com/gym/101462/problem/D

const int N = (int)1e5 + 5;
const int M = (int)1e5 + 5;

// finding the 2-vertex-connected components (BCC, biconnected components)
// k-vertex-connected: has more than k vertices and
//     if you remove less than k vertices the component remains connected
// for practical purposes, we will consider a bridge as a BCC in this algorithm
struct Graph {
    // INPUTS
    int n = 0; // nodes

    // internals for the graph
    int m = 0;
    vector<int> E[N + 1]; // edges
    int orig[M + 1], dest[M + 1];

    // internals for BCC algorithm
    int pila[M + 1], top, fin;
    int low[N + 1], timer;
    int dfsn[N + 1]; // dfs arrival time

    // OUTPUTS
    // artp: articulation point (its removal from the graph increases the
    //        number of connected components)
    // bridge: edge that when removed increases the number of connected components
    int bicomp[M + 1], nbicomp;
    bool bridge[M + 1], artp[N + 1];

    Graph() {
    }

    void clear(int n) {
        REP (i, n) E[i].clear();
        m = 0;
        this -> n = n;
    }

    int otherVertex(int e, int u) {
        return orig[e] == u? dest[e] : orig[e];
    }

    // it supports multiple edges
    void addEdge(int a, int b) {
        orig[m] = a;
        dest[m] = b;
        E[a].push_back(m);
        E[b].push_back(m);
        m++;
    }

    int dfsbcc (int u, int p = -1){
        low[u] = dfsn[u] = ++timer;
        int ch = 0;
        for( auto e : E[ u ] ){
            int v = otherVertex(e, u);
            if (dfsn[v] == 0){
                pila[top++] = e;
                dfsbcc (v, e);
                low[u] = min (low[u], low[v]);
                ch++;
                if (low[v] >= dfsn[u]){
```

```
                    artp[u] = 1;
                    do {
                        fin = pila[--top];
                        bicomp[fin] = nbicomp;
                    } while (fin != e);
                    nbicomp++;
                }
                if (low[v] == dfsn[v]) bridge[e] = 1;
            } else if (e != p && dfsn[v] < dfsn[u]){
                pila[top++] = e;
                low[u] = min (low[u], dfsn[v]);
            }
        }
        return ch;
    }
    void bcc (){
        REP( i , n ) artp[ i ] = dfsn[ i ] = 0;
        REP( i , m ) bridge[ i ] = 0;
        fin = top = nbicomp = timer = 0;
        REP( i , n ) if (dfsn[ i ] == 0) artp[ i ] = dfsbcc( i ) >= 2;
    }
}graph;
```

4.2 Dijkstra

```
// tested on http://codeforces.com/contest/20/problem/C

const int MAXE = (int)1e5 + 5;
const int MAXV = (int)1e5 + 5;

vector<int> adj[MAXV]; // adjacent edges
int to[2 * MAXE]; // to
Long weight[2 * MAXE]; // weight
Long dis[MAXV];
int parent[MAXV];
int edges = 0;

void addDirectedEdge(int u, int v, Long w) {
    adj[u].push_back(edges++);
    to[edges - 1] = v;
    weight[edges - 1] = w;
}
void addUndirectedEdge(int u, int v, Long w) {
    addDirectedEdge(u, v, w);
    addDirectedEdge(v, u, w);
}

// O ( (E + V) * log(V) )
Long dijkstra(int source, int target) {
    priority_queue<pair<Long, int> > pq; // weight, vertex
    CLR(dis, -1);
    CLR(parent, -1);

    dis[source] = 0;
    pq.push({0, source});
    parent[source] = source;

    while (!pq.empty()) {
        auto nnp = pq.top();
        pq.pop();
        Long nndist = -nnp.first;
        int nn = nnp.second;

        if (nndist > dis[nn]) continue; // to save time ignoring improved nodes (which are already in
            the heap)

        if (nn == target) break;

        for (int i = 0; i < (int)adj[nn].size(); i++) {
            int e = adj[nn][i]; // edge
            int son = to[e];
            Long w = weight[e];
            Long dson = nndist + w;
            if (dis[son] == -1 || dis[son] > dson) {
                dis[son] = dson;
                parent[son] = nn; // only saving the first shortest path found
                pq.push({-dson, son});
            }
        }
    }

    return dis[target];
}


int main() {
    int n, m;
    sc(n);
    sc(m);
    REP (i, m) {
        int a, b, w;
        sc(a);
        sc(b);
        sc(w);
        a--;
        b--;
        addUndirectedEdge(a, b, w);
    }
    Long ans = dijkstra(0, n - 1);
    if (ans != -1) {
        int p = n - 1;
        vector<int> path;
        path.push_back(p);
        while (p != parent[p]) {
            p = parent[p];
            path.push_back(p);
        }
        reverse(ALL(path));
        REP (i, SZ(path)) {
            if (i) putchar(' ');
            printf("%d", path[i] + 1);
        }
        puts("");
    } else {
        puts("-1");
    }
}
```

4.3 Bellman Ford (and applications)

```
// tested on https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=165&page=
    show_problem&problem=499

const int MAXE = (int)2e3 + 3;
const int MAXV = (int)1e3 + 3;

Long dis[MAXV];
pair<int, int> edge[MAXE];
Long weight[MAXE];
int edges, nodes, q;
const Long INF = (int)1e7;

// returns -1 if no vertex was relaxed
int relax(Long dis[MAXV]) {
    int lastRelaxed = -1;
    for (int i = 0; i < edges; i++) {
        int from = edge[i].first;
        int to = edge[i].second;
        Long w = weight[i];
        // INF check is not only for overflow when dis[from] = INF,
        // it is also for avoiding distances like INF - 1, INF -2, ...
        if (dis[from] != INF && dis[to] > dis[from] + w) {
            dis[to] = max(dis[from] + w, -INF); // because distances may go far in the negative (- 2 ^
                V)
            // save parent here p[to] = from;
            lastRelaxed = to;
        }
    }
    return lastRelaxed;
}

int main() {
    int tc;
    sc(tc);
    REP (itc, tc) {
        sc(nodes);
        sc(edges);

        REP (i, edges) {
            int a, b;
            int w;
            sc(a);
            sc(b);
            sc(w);
            edge[i] = {a, b};
            weight[i] = w;
        }

        // bellman ford O(E * V)
        REP (i, nodes) {
            dis[i] = INF;
        }
        dis[0] = 0;
        REP (i, nodes - 1) {
```

```
            relax(dis);
        }
        // one more to check for negative cycles
        int lastRelaxed = relax(dis);

        if (lastRelaxed == -1) {
            puts("not possible");
        } else {
            puts("possible");
            // to rebuild the negative cycle closer to the source:
            // int y = lastRelaxed;
            // for (int i=0; i<n; ++i)
            //     y = p[y];

            // vector<int> path;
            // for (int cur=y; ; cur=p[cur])
            // {
            //     path.push_back (cur);
            //     if (cur == y && path.size() > 1)
            //         break;
            // }
            // reverse (path.begin(), path.end());

            // cout << "Negative cycle: ";
            // for (size_t i=0; i<path.size(); ++i)
            //     cout << path[i] << ' ';
        }
        // The above implementation looks for a negative cycle reachable from some starting vertex
              source; however, the algorithm can be modified to just looking for any negative cycle in
               the graph. For this we need to put all the distance d[i] to zero and not infinity
              as if we are looking for the shortest path from all vertices simultaneously; the
              validity of the detection of a negative cycle is not affected.

        /*
            Solving a set of inequalities:
            Building the constraint graph:
                Each variable Xi corresponds to a node Vi
                Each constraint Xj - Xi <= bij corresponds to an
                    edge from Xi to Xj with weight bij
                We add a special node V0 and we add edges from
                    this special node to all other nodes. The weights of
                    these edges are 0
                We run bellman ford with source V0.

            There are no negative cycles if and only if the set on inequalities has solution (the
                  solution is the final distances)
        */
    }
}
```

4.4 Floyd Warshall

```
// tested on https://open.kattis.com/problems/allpairspath

const int MAXV = (int)155;

Long dis[MAXV][MAXV];
int edges, nodes, q;
const Long INF = 150 * 1000 * 2;

void init() {
    REP (i, nodes) {
        REP (j, nodes) {
            dis[i][j] = INF;
        }
        dis[i][i] = 0;
    }
}

void floydWarshall() {
    REP (k, nodes) {
        REP (i, nodes) {
            REP (j, nodes) {
                if (dis[i][k] != INF && dis[k][j] != INF &&
                    dis[i][j] > dis[i][k] + dis[k][j]) {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }
}

int main() {
    int q;
    while (scanf("%d%d%d", &nodes, &edges, &q) == 3) {
        if (nodes == 0) break;
        init();

        REP (i, edges) {
            int a, b;
            int w;
            sc(a);
            sc(b);
            sc(w);
            dis[a][b] = min(dis[a][b], (Long)w);
        }

        floydWarshall();

        // detecting negative cycles
        REP (i, nodes) {
            REP (j, nodes) {
                REP (k, nodes) {
                    // there is a negative cycle passing by k and there is connectivity from i to k
                          and from k to j
                    if (dis[k][k] < 0 && dis[i][k] != INF && dis[k][j] != INF) {
                        dis[i][j] = -INF;
                    }
                }
            }
        }

        REP (i, q) {
            int from, to;
            sc(from);
            sc(to);
            if (dis[from][to] == -INF) {
                puts("-Infinity");
            } else if (dis[from][to] == INF) {
                puts("Impossible");
            } else {
                printf("%d\n", (int)dis[from][to]);
            }
        }
        puts("");
    }
}
```

4.5 SCC (Strongly Connected Components)

```
const int N = 2 * (int)5e4 + 4; // for 2-sat must be twice as the max number of variables

// tested on https://codeforces.com/gym/100430/problem/A
struct DirectedGraph {
    // inputs
    int n = 0;
    vector<int> G[ N + 5 ];
    vector<int> dag[ N + 5 ];

    // internals
    int timer , top;
    int dfsn[ N + 5 ] , pila[ N + 5 ] , inpila[ N + 5 ];

    // output
    int comp[ N + 5 ];

    DirectedGraph() {}

    void init(int _n) {
        REP (i, n) G[i].clear();
        n = _n;
    }

    void addEdge(int from , int to) {
        G[from].push_back(to);
    }

    int dfs( int u ){
        int low = dfsn[ u ] = ++timer;
        inpila[ pila[ top ++ ] = u ] = 1;
        for( int v : G[ u ] ){
            if( dfsn[ v ] == 0 ) low = min( low , dfs( v ) );
            else if( inpila[ v ] ) low = min( low , dfsn[ v ] );
        }
        if( low == dfsn[ u ] ){
            int fin;
            do{
                fin = pila[ --top ];
                inpila[ fin ] = 0;
                comp[ fin ] = u;
            }while( fin != u );
        }
        return low;
```

```
    }

    void SCC(){
        CLR( dfsn , 0 );
        top = timer = 0;
        REP( i , n ) if( !dfsn[ i ] ) dfs( i );
    }

    void buildSccDag() {
        REP (i, n) dag[i].clear();
        REP (u, n)for( auto v : G[ u ] ){
            int i = comp[ u ] , j = comp[ v ];
            if( i != j ) dag[ i ].push_back( j );
        }
    }
}dg;
```

4.6 2-SAT (with value assignation)

```
// tested on https://codeforces.com/gym/100430/problem/A

// you need the SCC struct with a dg instance

//Consider f=(x1 or y1) and (x2 or y2) and ...  and (xn or yn).
// All you need is to add condictions with addClause

// remember:
// x == true  is  x or x
// x == false  is  !x or !x
// x != y is (x or y) and (!x or !y)
// x == y is (!x or y) and (!y or x)


struct TwoSat { // 2-sat
    int n = 0; // number of variables

    // internals
    int vis[ N + 5 ], cola[ N + 5 ], sz;

    // outputs
    int decision[ N + 5 ];

    TwoSat() {}
    void init(int _n) {
        n = _n;
        dg.init(2 * n);
    }
    int getVar(bool s, int x) {
        if (s) return 2 * x; // even
        return 2 * x + 1;
    }
    int neg(int var) { // not
        return var ^ 1;
    }

    // adds a clause
    void addClause(bool xsign, int x, bool ysign, int y) { // or-clause
        ///Now consider a graph with 2n vertices; For each of ( x i      y i ) s we add two directed
              edges
        //From !xi to yi
        //From !yi to xi

        int a = getVar(xsign, x);
        int b = getVar(ysign, y);
        dg.addEdge(neg(a), b);
        dg.addEdge(neg(b), a);
    }

    // checks wether a solution exists
    bool solve() {
        dg.SCC();
        REP( i , n ) {
            if( dg.comp[ getVar(1, i) ] == dg.comp[ getVar(0, i) ] ){
                return 0;
            }
        }
        return 1;
    }

    void topsort( int u ){
        vis[ u ] = 1;
        for( auto v : dg.dag[ u ] )
            if( !vis[ v ] ) topsort( v );
        cola[ sz ++ ] = u;
    }
    void paint( int u ){
        decision[ u ] = 1;
        for( auto v : dg.dag[ u ] )
            if( decision[ v ] == -1 ) paint( v );
    }

    /**
     * This assigns a boolean value (decision) to all dag components (not values)
     * You may call it only if a solution exists.
     * **/
    void rebuild() {
        dg.buildSccDag();
        REP( i , 2 * n ) vis[ i ] = 0;
        sz = 0;
        REP( i , 2 * n ) if( dg.comp[ i ] == i && !vis[ i ] ) topsort( i );
        REP( i , 2 * n ) decision[ i ] = -1;
        reverse( cola , cola + sz );
        REP( i , sz )
            if( decision[ cola[ i ] ] == -1 ){
                decision[ cola[ i ] ] = 0;
                paint( dg.comp[ cola[ i ] ^ 1 ] );
            }
    }

    // use only after calling rebuild
    bool getValueForVariable(int x) {
        return decision[dg.comp[getVar(1, x)]];
    }
}ts;

int color[N]; // color per wire
pair<int, int> sockets[N]; // sockets per wire

int main() {
        freopen( "chip.in" , "r" , stdin );
        freopen( "chip.out" , "w" , stdout );
    int n;
    while (sc(n) == 1) {
        REP (i, n) {
            sc(color[i]);
            sockets[i] = {-1, -1};
        }
        ts.init(n);

        int firstWire;
        bool firstSign;
        int lastWire;
        bool lastSign;
        REP (i, 2 * n) {
            int w;
            sc(w);
            w--;

            bool mySign;
            if (sockets[w].first == -1) {
                sockets[w].first = i;
                mySign = 0;
            } else {
                sockets[w].second = i;
                mySign = 1;
            }

            if (i == 0) {
                firstWire = w;
                firstSign = mySign;
            } else {
                if (color[lastWire] == color[w]) {
                    ts.addClause(!lastSign, lastWire, !mySign, w);
                }
            }

            lastSign = mySign;
            lastWire = w;
        }
        if (color[lastWire] == color[firstWire]) {
            ts.addClause(!lastSign, lastWire, !firstSign, firstWire);
        }

        bool hasSolution = ts.solve();
        if (!hasSolution) {
            puts("NO");
        } else {
            puts("YES");
            ts.rebuild();

            REP (i, n) {
                bool isSecond = ts.getValueForVariable(i);
                int socket;
                if (!isSecond) {
                    socket = sockets[i].first;
                } else {
                    socket = sockets[i].second;
                }
```

```
                socket++;
                if (i) putchar(' ');
                printf("%d", socket);
            }
            puts("");
        }
    }
}
```

4.7 Union - Find

```
REP( i , n ) id[i] = i ; // init
int Find( int x ){ return id[ x ] = (id[ x ] == x ? x : Find( id[ x ] ) );}
void Union(int x, int y) {id[Find(x)] = Find(y);} // Find( x ) != Find( y )
//  with path compression
REP( i , SZ ) parent[i] = i , rank[i] = 1;
void Union(int a, int b)
{
        int pa = Find(a);
        int pb = Find(b);
        if(pa != pb)
        {
                if(rank[pa] < rank[pb]) parent[pa] = pb;
                else if(rank[pa] > rank[pb]) parent[pb] = pa;
                else
                {
                        parent[pb] = pa;
                        rank[pa]++;
                }
        }
}
```

4.8 Euler Path

```
// Plan-ChotaV2, tested on Codeforces Round #288 (Div. 2)D. Tanya and Password

//Eulerian path reconstruction in directed graph O( E + V )
// same idea is for undirected graph
int next[ MAXE + 5 ] , to[ MAXE + 5 ] , last[ N + 5 ] , E;

void add( int u , int v ){
    next[ E ] = last[ u ] , to[ E ] = v , last[ u ] = E++;
}
bool vis_edge[ MAXE + 5 ];
int res[ MAXE + 5 ] , len;

void solve( int u ){
    for( int e = last[ u ] ; e != -1 ; e = next[ e ] ){
        int v = to[ e ];
        last[ u ] = next[ e ];
        if( vis_edge[ e ] ) break;
        vis_edge[ e ] = true;

        solve( v );
        res[ len++ ] = v;
    }
}

bool vis[ N + 5 ];
int in[ N + 5 ] , out[ N + 5 ] , cant;

void dfs( int u ){
    if( vis[ u ] ) return;
    vis[ u ] = 1;

    cant++;
    for( int e = last[ u ] ; e != -1; e = next[ e ] ) dfs( to[ e ] );
}
int used[ N + 5 ];
int main(){
        ios_base :: sync_with_stdio( 0 );
    int n;
    while( cin >> n ){
        vi nodes ;
        clr( last , -1 );
        E = 0;
                clr( used , 0 );
        REP( i , n ){
                string s;
            cin >> s;
            int u = s[ 0 ] * 300 + s[ 1 ];
            int v = s[ 1 ] * 300 + s[ 2 ];
            add( u , v );
            if( !used[ u ] ) nodes.pb( u ) , used[ u ] = 1;
            if( !used[ v ] ) nodes.pb( v ) , used[ v ] = 1;

            in[ v ]++;
            out[ u ]++;
        }

        int ip = 0 , ini = -1;

        REP( i , SZ( nodes ) ){
            int u = nodes[ i ];
            if( abs( in[ u ] - out[ u ] ) == 1 ) ip++;
            else if( in[ u ] != out[ u ] ) ip = 100;

            if( in[ u ] - out[ u ] == -1 ) ini = u;
            else if( ini == -1 && in[ u ] == out[ u ] ) ini = u;
        }

        cant = 0;
        clr( vis , 0 );
                if( ini != -1 ) dfs( ini );
        if( cant == SZ( nodes ) && ip <= 2 ){
            cout << "YES\n";
            len = 0;
            clr( vis_edge , 0 );
            solve( ini );

            cout << char( ini / 300 );
            cout << char( ini % 300 );
            for(int i = n - 1; i >= 0; i--) cout << char( res[ i ] % 300 );
            cout << '\n';
        }
        else cout << "NO\n";
    }
}
```

4.9 Topological Sort

```
// Plan-chotaV2
//http://ahmed-aly.com/Standings.jsp?ID=2954
//11371_SPOJ
#define MAXN 100

// this was is useful for some backtracking problem
// , also useful for breaking ties by other criteria (i.e: node index)
void bfsTopsort() {
        for( int i = 0 ; i < m ; ++i )
        {
                G[u].push_back(v);
                in[v]++;
        }
        priority_queue <int> Q;
        for( int i = 0 ; i < n ; ++i )
                if( in[i] == 0 )
                        Q.push(-i);
        vector< int >orden;
        while( !Q.empty() )
        {
                int u = Q.top();
                u = -u;
                Q.pop();
                orden.push_back(u);
                int nG = G[u].size();
                for( int i = 0 ; i < nG ; ++i )
                {
                        int v = G[u][i];
                        in[v]--;
                        if( in[v] == 0 )
                        Q.push(-v);
                }
        }
}


// recrusivily
void topsort( int u ){
        vis[ u ] = 1;
        FOR( v , dag[ u ] )
                if( !vis[ *v ] ) topsort( *v );
        cola[ sz ++ ] = u;
}
```

4.10 Adjacency Matrix

```
Matrix powers:
If A is the adjacency matrix of the directed or undirected graph G,
then the matrix A^n (i.e., the matrix product of n copies of A) has an interesting interpretation:
the element (i, j) gives the number of (directed or undirected) walks of length n from vertex i to
      vertex j.
If n is the smallest nonnegative integer, such that for some i, j, the element (i, j) of A^n is
      positive,
then n is the distance between vertex i and vertex j.
This implies, for example, that the number of triangles in an undirected graph G is exactly the trace
      of A^3
divided by 6.
```

4.11 Kruskal (Minimum Spanning Tree)

```
// tested on https://icpcarchive.ecs.baylor.edu/index.php?option=onlinejudge&page=show_problem&problem
      =516
// O (E * log(E))

const int N = 1e6;

int id[ N + 5 ];
int Find( int x ){ return id[ x ] = (id[ x ] == x ? x : Find( id[ x ] ) );}
struct Edge{
        int u , v;
    Long w;
        Edge(){}
        Edge( int u , int v , Long w ) : u( u ) , v( v ) , w( w ) {}
};
bool operator < ( const Edge &a , const Edge &b ){ return a.w < b.w ;}
int main(){
        int n , m , u , v , w;
        while( sc( n ) == 1 ){
                if( !n ) break;
                sc( m );
                REP( i , N ) id[ i ] = i;
                vector< Edge > E;
                REP( i , m ){
                        sc( u ) , sc( v ) , sc( w );
                        u -- , v --;
                        E.push_back( Edge( u , v , w ) );
                }
                sort( ALL( E ) );
                int ans = 0;
                REP( i , SZ( E ) ){
                        int pu = Find( E[ i ].u ) , pv = Find( E[ i ].v );
                        if( pu != pv ){
                                ans += E[ i ].w;
                                id[ pu ] = pv;
                        }
                }
                printf( "%d\n" , ans );
        }
}
```

5 Games

5.1 Nim de la miseria

```
// Es el juego de nim solo que el ultimo en jugar pierde (el que remueve la ultima piedra)

// It is both well-known and easy to verify that a Nim position (n1,    ,nk) is a second player win in
      mis re Nim if and only if some ni>1 and (n1 xor ...  xor nk)=0, or all n i  1 and (n1 xor ...
      xor nk)=1.
```

6 DP

6.1 Subsets of the subsets iteration

```
// O(3 ^ n)

for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
```

7 Strings

7.1 AhoCorasick

```
// Plan-chotaV2.cpp
// with adyancency list
// tested on https://www.spoj.com/problems/SUB_PROB/

const int ND = (int)2e6 + 6; // number of nodes
vector<int> V[ ND ]; // V[i] is the list of id's of words in the node i
vector< pair< char , int > > trie[ND];
int T[ ND ] , Node ; // T is the fallback table

inline int getNode( int node , char c )
{
        for (auto o : trie[node] )
                if( o.first == c )return o.second;
        return 0;
}
void add( char *s , int id )
{
        int ns = strlen( s ) , p = 0 ;
        REP( i , ns )
        {
                int v = getNode( p , s[i] );
                if( !v )
                {
                        trie[p].push_back( make_pair( s[i] , Node ) );
                        p = Node++;
                }
                else p = v;
        }
        V[ p ].push_back( id );
}

void aho()
{
        queue< int >Q;
        for (auto o : trie[0] ) {
                Q.push( o.second ) , T[ o.second ] = 0;
    }
        while( !Q.empty() )
        {
                int u = Q.front();
                Q.pop();
        for (auto o : trie[u]) {
                        int v = o.second;
                        char c = o.first;
                        int p = T[u];
                        while( p && getNode( p , c ) == 0 )p = T[p];
                        p = getNode( p , c );
                        T[ v ] = p;
                        Q.push( v );
            for (auto q : V[ T[v] ]) {
                V[ v ].push_back( q );
            }
                }
        }
}

const int M = 1000 + 3; // number of words (patterns to search for)
const int N = 100000 + 5; // number of chars in the haystack
bool ans[ M ];

int main()
{
        char s[ N ] , t[ M ];
        int n;
        scanf( "%s%d" , s , &n );
        Node = 1;
        REP( i , n ) scanf( "%s" , t ) , add( t , i );
        int ns = strlen( s );
        aho();
        int p = 0;
        REP( i , ns )
        {
                char c = s[i];
                while( p && getNode( p , c ) == 0 ) p = T[p];
                p = getNode( p , c );
```

```
        for (auto o : V[p]) {
            ans[o] = 1;
        }
        }
        REP( i , n )puts( (ans[i]?"Y":"N") );
}
```

8 Techniques

8.1 Various algorithm techniques

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

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

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

In general:

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

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

Geometric series:

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

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

Harmonic series:

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

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

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

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

6. $\displaystyle\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\displaystyle\sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n}$,

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

10. $\displaystyle\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$, **11.** $\displaystyle\left\{{n \atop 1}\right\} = \left\{{n \atop n}\right\} = 1$,

12. $\displaystyle\left\{{n \atop 2}\right\} = 2^{n-1} - 1$, **13.** $\displaystyle\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$,

14. $\displaystyle\left[{n \atop 1}\right] = (n-1)!$, **15.** $\displaystyle\left[{n \atop 2}\right] = (n-1)!H_{n-1}$, **16.** $\displaystyle\left[{n \atop n}\right] = 1$, **17.** $\displaystyle\left[{n \atop k}\right] \ge \left\{{n \atop k}\right\}$,

18. $\displaystyle\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$, **19.** $\displaystyle\left\{{n \atop n-1}\right\} = \left[{n \atop n-1}\right] = \binom{n}{2}$, **20.** $\displaystyle\sum_{k=0}^n \left[{n \atop k}\right] = n!$, **21.** $\displaystyle C_n = \frac{1}{n+1}\binom{2n}{n}$,

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

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

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

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

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

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

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

40. $\left\{ {n \atop m} \right\} = \sum_k \binom{n}{k} \left\{ {k+1 \atop m+1} \right\} (-1)^{n-k}$, **41.** $\left[{n \atop m} \right] = \sum_k \left[{n+1 \atop k+1} \right] \binom{k}{m} (-1)^{m-k}$,

42. $\left\{ {m+n+1 \atop m} \right\} = \sum_{k=0}^{m} k \left\{ {n+k \atop k} \right\}$, **43.** $\left[{m+n+1 \atop m} \right] = \sum_{k=0}^{m} k(n+k) \left[{n+k \atop k} \right]$,

44. $\binom{n}{m} = \sum_k \left\{ {n+1 \atop k+1} \right\} \left[{k \atop m} \right] (-1)^{m-k}$, **45.** $(n-m)! \binom{n}{m} = \sum_k \left[{n+1 \atop k+1} \right] \left\{ {k \atop m} \right\} (-1)^{m-k}$, for $n \ge m$,

46. $\left\{ {n \atop n-m} \right\} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left[{m+k \atop k} \right]$, **47.** $\left[{n \atop n-m} \right] = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ {m+k \atop k} \right\}$,

48. $\left\{ {n \atop \ell+m} \right\} \binom{\ell+m}{\ell} = \sum_k \left\{ {k \atop \ell} \right\} \left\{ {n-k \atop m} \right\} \binom{n}{k}$, **49.** $\left[{n \atop \ell+m} \right] \binom{\ell+m}{\ell} = \sum_k \left[{k \atop \ell} \right] \left[{n-k \atop m} \right] \binom{n}{k}$.

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

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

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

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

Recurrences

Master method:

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

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

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

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

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

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

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

Substitution (example): Consider the following recurrence

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

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

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

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

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

Substituting we find

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

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

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

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

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

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

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

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

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

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

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

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

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

Note that

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

Subtracting we find

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

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

Generating functions:

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

Example:

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

Multiply and sum:

$$\sum_{i \ge 0} g_{i+1} x^i = \sum_{i \ge 0} 2g_i x^i + \sum_{i \ge 0} x^i.$$

We choose $G(x) = \sum_{i \ge 0} x^i g_i$. Rewrite in terms of $G(x)$:

$$\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \ge 0} x^i.$$

Simplify:

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

Solve for $G(x)$:

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

Expand this using partial fractions:

$$\begin{aligned} G(x) &= x \left(\frac{2}{1-2x} - \frac{1}{1-x} \right) \\ &= x \left(2 \sum_{i \ge 0} 2^i x^i - \sum_{i \ge 0} x^i \right) \\ &= \sum_{i \ge 0} (2^{i+1} - 1) x^{i+1}. \end{aligned}$$

So $g_i = 2^i - 1$.

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

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

Pascal's Triangle

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

General

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

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

Change of base, quadratic formula:

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

Euler's number e:

$$e = 1 + \tfrac{1}{2} + \tfrac{1}{6} + \tfrac{1}{24} + \tfrac{1}{120} + \cdots$$

$$\lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n = e^x.$$

$$\left(1 + \tfrac{1}{n}\right)^n < e < \left(1 + \tfrac{1}{n}\right)^{n+1}.$$

$$\left(1 + \tfrac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$$

Harmonic numbers:

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

$$\ln n < H_n < \ln n + 1,$$

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

Factorial, Stirling's approximation:

$$1,\ 2,\ 6,\ 24,\ 120,\ 720,\ 5040,\ 40320,\ 362880,\ \ldots$$

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

Ackermann's function and inverse:

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

$$\alpha(i) = \min\{j \mid a(j,j) \geq i\}.$$

Binomial distribution:

$$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \qquad q = 1 - p,$$

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

Poisson distribution:

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

Normal (Gaussian) distribution:

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

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

$$nH_n.$$

Probability

Continuous distributions: If

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

then p is the probability density function of X. If

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

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

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

Expectation: If X is discrete

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

If X continuous then

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

Variance, standard deviation:

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

$$\sigma = \sqrt{\mathrm{VAR}[X]}.$$

For events A and B:

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

$$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$$

iff A and B are independent.

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

For random variables X and Y:

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

if X and Y are independent.

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

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

Bayes' theorem:

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

Inclusion-exclusion:

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

Moment inequalities:

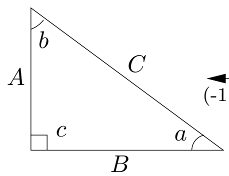
$$\Pr\left[|X| \geq \lambda\,\mathrm{E}[X]\right] \leq \frac{1}{\lambda},$$

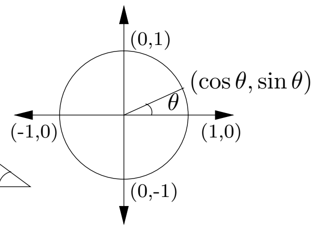
$$\Pr\left[\left|X - \mathrm{E}[X]\right| \geq \lambda \cdot \sigma\right] \leq \frac{1}{\lambda^2}.$$

Geometric distribution:

$$\Pr[X = k] = pq^{k-1}, \qquad q = 1 - p,$$

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





Pythagorean theorem:

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

Definitions:

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

Area, radius of inscribed circle:

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

Identities:

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

Euler's equation:

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

sseiden@acm.org
http://www.csc.lsu.edu/~seiden

Multiplication:

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

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

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

2×2 and 3×3 determinant:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

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

Permanents:

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

Hyperbolic Functions

Definitions:

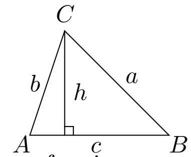
$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2},$$
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \operatorname{csch} x = \frac{1}{\sinh x},$$
$$\operatorname{sech} x = \frac{1}{\cosh x}, \qquad \coth x = \frac{1}{\tanh x}.$$

Identities:

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

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

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



Law of cosines:

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

Area:

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

Heron's formula:

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

More identities:

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

The Chinese remainder theorem: There exists a number C such that:

$$C \equiv r_1 \bmod m_1$$
$$\vdots \quad \vdots \quad \vdots$$
$$C \equiv r_n \bmod m_n$$

if m_i and m_j are relatively prime for $i \neq j$.

Euler's function: $\phi(x)$ is the number of positive integers less than x relatively prime to x. If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$\phi(x) = \prod_{i=1}^n p_i^{e_i - 1}(p_i - 1).$$

Euler's theorem: If a and b are relatively prime then

$$1 \equiv a^{\phi(b)} \bmod b.$$

Fermat's theorem:

$$1 \equiv a^{p-1} \bmod p.$$

The Euclidean algorithm: if $a > b$ are integers then

$$\gcd(a, b) = \gcd(a \bmod b, b).$$

If $\prod_{i=1}^n p_i^{e_i}$ is the prime factorization of x then

$$S(x) = \sum_{d|x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$$

Perfect Numbers: x is an even perfect number iff $x = 2^{n-1}(2^n - 1)$ and $2^n - 1$ is prime.

Wilson's theorem: n is a prime iff

$$(n-1)! \equiv -1 \bmod n.$$

Möbius inversion:

$$\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of} \\ & r \text{ distinct primes.} \end{cases}$$

If

$$G(a) = \sum_{d|a} F(d),$$

then

$$F(a) = \sum_{d|a} \mu(d) G\left(\frac{a}{d}\right).$$

Prime numbers:

$$p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$$

$$\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$$

Definitions:

Loop	An edge connecting a vertex to itself.
Directed	Each edge has a direction.
Simple	Graph with no loops or multi-edges.
Walk	A sequence $v_0 e_1 v_1 \dots e_\ell v_\ell$.
Trail	A walk with distinct edges.
Path	A trail with distinct vertices.
Connected	A graph where there exists a path between any two vertices.
Component	A maximal connected subgraph.
Tree	A connected acyclic graph.
Free tree	A tree with no root.
DAG	Directed acyclic graph.
Eulerian	Graph with a trail visiting each edge exactly once.
Hamiltonian	Graph with a cycle visiting each vertex exactly once.
Cut	A set of edges whose removal increases the number of components.
Cut-set	A minimal cut.
Cut edge	A size 1 cut.
k-Connected	A graph connected with the removal of any $k-1$ vertices.
k-Tough	$\forall S \subseteq V, S \neq \emptyset$ we have $k \cdot c(G - S) \leq \lvert S \rvert$.
k-Regular	A graph where all vertices have degree k.
k-Factor	A k-regular spanning subgraph.
Matching	A set of edges, no two of which are adjacent.
Clique	A set of vertices, all of which are adjacent.
Ind. set	A set of vertices, none of which are adjacent.
Vertex cover	A set of vertices which cover all edges.
Planar graph	A graph which can be embeded in the plane.
Plane graph	An embedding of a planar graph.

$$\sum_{v \in V} \deg(v) = 2m.$$

If G is planar then $n - m + f = 2$, so

$$f \leq 2n - 4, \quad m \leq 3n - 6.$$

Any planar graph has a vertex with degree ≤ 5.

Notation:

$E(G)$	Edge set
$V(G)$	Vertex set
$c(G)$	Number of components
$G[S]$	Induced subgraph
$\deg(v)$	Degree of v
$\Delta(G)$	Maximum degree
$\delta(G)$	Minimum degree
$\chi(G)$	Chromatic number
$\chi_E(G)$	Edge chromatic number
G^c	Complement graph
K_n	Complete graph
K_{n_1,n_2}	Complete bipartite graph
$\mathrm{r}(k, \ell)$	Ramsey number

Geometry

Projective coordinates: triples (x, y, z), not all x, y and z zero.

$$(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$$

Cartesian	Projective
(x, y)	$(x, y, 1)$
$y = mx + b$	$(m, -1, b)$
$x = c$	$(1, 0, -c)$

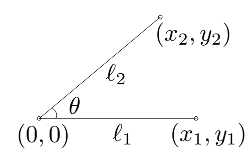
Distance formula, L_p and L_∞ metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$
$$\left[|x_1 - x_0|^p + |y_1 - y_0|^p \right]^{1/p},$$
$$\lim_{p \to \infty} \left[|x_1 - x_0|^p + |y_1 - y_0|^p \right]^{1/p}.$$

Area of triangle (x_0, y_0), (x_1, y_1) and (x_2, y_2):

$$\frac{1}{2} \operatorname{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$$

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{\ell_1 \ell_2}.$$

Line through two points (x_0, y_0) and (x_1, y_1):

$$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$$

Area of circle, volume of sphere:

$$A = \pi r^2, \qquad V = \tfrac{4}{3} \pi r^3.$$

If I have seen farther than others, it is because I have stood on the shoulders of giants.
– Issac Newton

Wallis' identity:

$$\pi = 2 \cdot \frac{2 \cdot 2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdots}{1 \cdot 3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdots}$$

Brouncker's continued fraction expansion:

$$\tfrac{\pi}{4} = 1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \cdots}}}}$$

Gregrory's series:

$$\tfrac{\pi}{4} = 1 - \tfrac{1}{3} + \tfrac{1}{5} - \tfrac{1}{7} + \tfrac{1}{9} - \cdots$$

Newton's series:

$$\tfrac{\pi}{6} = \frac{1}{2} + \frac{1}{2 \cdot 3 \cdot 2^3} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5 \cdot 2^5} + \cdots$$

Sharp's series:

$$\tfrac{\pi}{6} = \frac{1}{\sqrt{3}}\left(1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \cdots\right)$$

Euler's series:

$$\tfrac{\pi^2}{6} = \tfrac{1}{1^2} + \tfrac{1}{2^2} + \tfrac{1}{3^2} + \tfrac{1}{4^2} + \tfrac{1}{5^2} + \cdots$$
$$\tfrac{\pi^2}{8} = \tfrac{1}{1^2} + \tfrac{1}{3^2} + \tfrac{1}{5^2} + \tfrac{1}{7^2} + \tfrac{1}{9^2} + \cdots$$
$$\tfrac{\pi^2}{12} = \tfrac{1}{1^2} - \tfrac{1}{2^2} + \tfrac{1}{3^2} - \tfrac{1}{4^2} + \tfrac{1}{5^2} - \cdots$$

Partial Fractions

Let $N(x)$ and $D(x)$ be polynomial functions of x. We can break down $N(x)/D(x)$ using partial fraction expansion. First, if the degree of N is greater than or equal to the degree of D, divide N by D, obtaining

$$\frac{N(x)}{D(x)} = Q(x) + \frac{N'(x)}{D(x)},$$

where the degree of N' is less than that of D. Second, factor $D(x)$. Use the following rules: For a non-repeated factor:

$$\frac{N(x)}{(x-a)D(x)} = \frac{A}{x-a} + \frac{N'(x)}{D(x)},$$

where

$$A = \left[\frac{N(x)}{D(x)}\right]_{x=a}.$$

For a repeated factor:

$$\frac{N(x)}{(x-a)^m D(x)} = \sum_{k=0}^{m-1} \frac{A_k}{(x-a)^{m-k}} + \frac{N'(x)}{D(x)},$$

where

$$A_k = \frac{1}{k!}\left[\frac{d^k}{dx^k}\left(\frac{N(x)}{D(x)}\right)\right]_{x=a}.$$

The reasonable man adapts himself to the world; the unreasonable persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable.
– George Bernard Shaw

Derivatives:

1. $\dfrac{d(cu)}{dx} = c\dfrac{du}{dx}$, **2.** $\dfrac{d(u+v)}{dx} = \dfrac{du}{dx} + \dfrac{dv}{dx}$, **3.** $\dfrac{d(uv)}{dx} = u\dfrac{dv}{dx} + v\dfrac{du}{dx}$,

4. $\dfrac{d(u^n)}{dx} = nu^{n-1}\dfrac{du}{dx}$, **5.** $\dfrac{d(u/v)}{dx} = \dfrac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$, **6.** $\dfrac{d(e^{cu})}{dx} = ce^{cu}\dfrac{du}{dx}$,

7. $\dfrac{d(c^u)}{dx} = (\ln c)c^u\dfrac{du}{dx}$, **8.** $\dfrac{d(\ln u)}{dx} = \dfrac{1}{u}\dfrac{du}{dx}$,

9. $\dfrac{d(\sin u)}{dx} = \cos u\dfrac{du}{dx}$, **10.** $\dfrac{d(\cos u)}{dx} = -\sin u\dfrac{du}{dx}$,

11. $\dfrac{d(\tan u)}{dx} = \sec^2 u\dfrac{du}{dx}$, **12.** $\dfrac{d(\cot u)}{dx} = \csc^2 u\dfrac{du}{dx}$,

13. $\dfrac{d(\sec u)}{dx} = \tan u \sec u\dfrac{du}{dx}$, **14.** $\dfrac{d(\csc u)}{dx} = -\cot u \csc u\dfrac{du}{dx}$,

15. $\dfrac{d(\arcsin u)}{dx} = \dfrac{1}{\sqrt{1-u^2}}\dfrac{du}{dx}$, **16.** $\dfrac{d(\arccos u)}{dx} = \dfrac{-1}{\sqrt{1-u^2}}\dfrac{du}{dx}$,

17. $\dfrac{d(\arctan u)}{dx} = \dfrac{1}{1+u^2}\dfrac{du}{dx}$, **18.** $\dfrac{d(\operatorname{arccot} u)}{dx} = \dfrac{-1}{1+u^2}\dfrac{du}{dx}$,

19. $\dfrac{d(\operatorname{arcsec} u)}{dx} = \dfrac{1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **20.** $\dfrac{d(\operatorname{arccsc} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$,

21. $\dfrac{d(\sinh u)}{dx} = \cosh u\dfrac{du}{dx}$, **22.** $\dfrac{d(\cosh u)}{dx} = \sinh u\dfrac{du}{dx}$,

23. $\dfrac{d(\tanh u)}{dx} = \operatorname{sech}^2 u\dfrac{du}{dx}$, **24.** $\dfrac{d(\coth u)}{dx} = -\operatorname{csch}^2 u\dfrac{du}{dx}$,

25. $\dfrac{d(\operatorname{sech} u)}{dx} = -\operatorname{sech} u \tanh u\dfrac{du}{dx}$, **26.** $\dfrac{d(\operatorname{csch} u)}{dx} = -\operatorname{csch} u \coth u\dfrac{du}{dx}$,

27. $\dfrac{d(\operatorname{arcsinh} u)}{dx} = \dfrac{1}{\sqrt{1+u^2}}\dfrac{du}{dx}$, **28.** $\dfrac{d(\operatorname{arccosh} u)}{dx} = \dfrac{1}{\sqrt{u^2-1}}\dfrac{du}{dx}$,

29. $\dfrac{d(\operatorname{arctanh} u)}{dx} = \dfrac{1}{1-u^2}\dfrac{du}{dx}$, **30.** $\dfrac{d(\operatorname{arccoth} u)}{dx} = \dfrac{1}{u^2-1}\dfrac{du}{dx}$,

31. $\dfrac{d(\operatorname{arcsech} u)}{dx} = \dfrac{-1}{u\sqrt{1-u^2}}\dfrac{du}{dx}$, **32.** $\dfrac{d(\operatorname{arccsch} u)}{dx} = \dfrac{-1}{|u|\sqrt{1+u^2}}\dfrac{du}{dx}$.

Integrals:

1. $\displaystyle\int cu\,dx = c\int u\,dx$, **2.** $\displaystyle\int (u+v)\,dx = \int u\,dx + \int v\,dx$,

3. $\displaystyle\int x^n\,dx = \frac{1}{n+1}x^{n+1}, \quad n \neq -1$, **4.** $\displaystyle\int \frac{1}{x}dx = \ln x$, **5.** $\displaystyle\int e^x\,dx = e^x$,

6. $\displaystyle\int \frac{dx}{1+x^2} = \arctan x$, **7.** $\displaystyle\int u\frac{dv}{dx}dx = uv - \int v\frac{du}{dx}dx$,

8. $\displaystyle\int \sin x\,dx = -\cos x$, **9.** $\displaystyle\int \cos x\,dx = \sin x$,

10. $\displaystyle\int \tan x\,dx = -\ln|\cos x|$, **11.** $\displaystyle\int \cot x\,dx = \ln|\cos x|$,

12. $\displaystyle\int \sec x\,dx = \ln|\sec x + \tan x|$, **13.** $\displaystyle\int \csc x\,dx = \ln|\csc x + \cot x|$,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

71. $\displaystyle\int x^3\sqrt{x^2+a^2}\,dx = (\tfrac13 x^2 - \tfrac{2}{15}a^2)(x^2+a^2)^{3/2},$

72. $\displaystyle\int x^n\sin(ax)\,dx = -\tfrac1a x^n\cos(ax) + \tfrac{n}{a}\int x^{n-1}\cos(ax)\,dx,$

73. $\displaystyle\int x^n\cos(ax)\,dx = \tfrac1a x^n\sin(ax) - \tfrac{n}{a}\int x^{n-1}\sin(ax)\,dx,$

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

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

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

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

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

Difference, shift operators:

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

Fundamental Theorem:

$$f(x) = \Delta F(x) \Leftrightarrow \sum f(x)\delta x = F(x) + C.$$
$$\sum_a^b f(x)\delta x = \sum_{i=a}^{b-1} f(i).$$

Differences:

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

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

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

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

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

Sums:

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

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

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

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

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

Falling Factorial Powers:

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

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

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

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

Rising Factorial Powers:

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

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

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

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

Conversion:

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

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

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

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

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

Taylor's series:

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

Expansions:

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \cdots = \sum_{i=0}^{\infty} x^i,$$

$$\frac{1}{1-cx} = 1 + cx + c^2x^2 + c^3x^3 + \cdots = \sum_{i=0}^{\infty} c^i x^i,$$

$$\frac{1}{1-x^n} = 1 + x^n + x^{2n} + x^{3n} + \cdots = \sum_{i=0}^{\infty} x^{ni},$$

$$\frac{x}{(1-x)^2} = x + 2x^2 + 3x^3 + 4x^4 + \cdots = \sum_{i=0}^{\infty} i x^i,$$

$$x^k \frac{d^n}{dx^n}\left(\frac{1}{1-x}\right) = x + 2^n x^2 + 3^n x^3 + 4^n x^4 + \cdots = \sum_{i=0}^{\infty} i^n x^i,$$

$$e^x = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!},$$

$$\ln(1+x) = x - \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 - \tfrac{1}{4}x^4 - \cdots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i},$$

$$\ln \frac{1}{1-x} = x + \tfrac{1}{2}x^2 + \tfrac{1}{3}x^3 + \tfrac{1}{4}x^4 + \cdots = \sum_{i=1}^{\infty} \frac{x^i}{i},$$

$$\sin x = x - \tfrac{1}{3!}x^3 + \tfrac{1}{5!}x^5 - \tfrac{1}{7!}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$

$$\cos x = 1 - \tfrac{1}{2!}x^2 + \tfrac{1}{4!}x^4 - \tfrac{1}{6!}x^6 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!},$$

$$\tan^{-1} x = x - \tfrac{1}{3}x^3 + \tfrac{1}{5}x^5 - \tfrac{1}{7}x^7 + \cdots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)},$$

$$(1+x)^n = 1 + nx + \tfrac{n(n-1)}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{n}{i} x^i,$$

$$\frac{1}{(1-x)^{n+1}} = 1 + (n+1)x + \tbinom{n+2}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{i+n}{i} x^i,$$

$$\frac{x}{e^x - 1} = 1 - \tfrac{1}{2}x + \tfrac{1}{12}x^2 - \tfrac{1}{720}x^4 + \cdots = \sum_{i=0}^{\infty} \frac{B_i x^i}{i!},$$

$$\frac{1}{2x}(1 - \sqrt{1-4x}) = 1 + x + 2x^2 + 5x^3 + \cdots = \sum_{i=0}^{\infty} \frac{1}{i+1}\binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}} = 1 + x + 2x^2 + 6x^3 + \cdots = \sum_{i=0}^{\infty} \binom{2i}{i} x^i,$$

$$\frac{1}{\sqrt{1-4x}}\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = 1 + (2+n)x + \tbinom{4+n}{2}x^2 + \cdots = \sum_{i=0}^{\infty} \binom{2i+n}{i} x^i,$$

$$\frac{1}{1-x} \ln \frac{1}{1-x} = x + \tfrac{3}{2}x^2 + \tfrac{11}{6}x^3 + \tfrac{25}{12}x^4 + \cdots = \sum_{i=1}^{\infty} H_i x^i,$$

$$\frac{1}{2}\left(\ln \frac{1}{1-x}\right)^2 = \tfrac{1}{2}x^2 + \tfrac{3}{4}x^3 + \tfrac{11}{24}x^4 + \cdots = \sum_{i=2}^{\infty} \frac{H_{i-1} x^i}{i},$$

$$\frac{x}{1-x-x^2} = x + x^2 + 2x^3 + 3x^4 + \cdots = \sum_{i=0}^{\infty} F_i x^i,$$

$$\frac{F_n x}{1 - (F_{n-1} + F_{n+1})x - (-1)^n x^2} = F_n x + F_{2n}x^2 + F_{3n}x^3 + \cdots = \sum_{i=0}^{\infty} F_{ni} x^i.$$

Ordinary power series:

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

Exponential power series:

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

Dirichlet power series:

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

Binomial theorem:

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

Difference of like powers:

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

For ordinary power series:

$$\alpha A(x) + \beta B(x) = \sum_{i=0}^{\infty} (\alpha a_i + \beta b_i) x^i,$$

$$x^k A(x) = \sum_{i=k}^{\infty} a_{i-k} x^i,$$

$$\frac{A(x) - \sum_{i=0}^{k-1} a_i x^i}{x^k} = \sum_{i=0}^{\infty} a_{i+k} x^i,$$

$$A(cx) = \sum_{i=0}^{\infty} c^i a_i x^i,$$

$$A'(x) = \sum_{i=0}^{\infty} (i+1) a_{i+1} x^i,$$

$$xA'(x) = \sum_{i=1}^{\infty} i a_i x^i,$$

$$\int A(x)\,dx = \sum_{i=1}^{\infty} \frac{a_{i-1}}{i} x^i,$$

$$\frac{A(x) + A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i} x^{2i},$$

$$\frac{A(x) - A(-x)}{2} = \sum_{i=0}^{\infty} a_{2i+1} x^{2i+1}.$$

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

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

Convolution:

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

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

Expansions:

$$\frac{1}{(1-x)^{n+1}}\ln\frac{1}{1-x} = \sum_{i=0}^{\infty}(H_{n+i}-H_n)\binom{n+i}{i}x^i,$$

$$x^{\overline{n}} = \sum_{i=0}^{\infty}\begin{bmatrix} n \\ i \end{bmatrix}x^i,$$

$$\left(\ln\frac{1}{1-x}\right)^n = \sum_{i=0}^{\infty}\begin{bmatrix} i \\ n \end{bmatrix}\frac{n!x^i}{i!},$$

$$\tan x = \sum_{i=1}^{\infty}(-1)^{i-1}\frac{2^{2i}(2^{2i}-1)B_{2i}x^{2i-1}}{(2i)!},$$

$$\frac{1}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\mu(i)}{i^x},$$

$$\zeta(x) = \prod_p \frac{1}{1-p^{-x}},$$

$$\zeta^2(x) = \sum_{i=1}^{\infty}\frac{d(i)}{x^i} \quad \text{where } d(n)=\textstyle\sum_{d|n}1,$$

$$\zeta(x)\zeta(x-1) = \sum_{i=1}^{\infty}\frac{S(i)}{x^i} \quad \text{where } S(n)=\textstyle\sum_{d|n}d,$$

$$\zeta(2n) = \frac{2^{2n-1}|B_{2n}|}{(2n)!}\pi^{2n}, \quad n\in\mathbb{N},$$

$$\frac{x}{\sin x} = \sum_{i=0}^{\infty}(-1)^{i-1}\frac{(4^i-2)B_{2i}x^{2i}}{(2i)!},$$

$$\left(\frac{1-\sqrt{1-4x}}{2x}\right)^n = \sum_{i=0}^{\infty}\frac{n(2i+n-1)!}{i!(n+i)!}x^i,$$

$$e^x\sin x = \sum_{i=1}^{\infty}\frac{2^{i/2}\sin\frac{i\pi}{4}}{i!}x^i,$$

$$\sqrt{\frac{1-\sqrt{1-x}}{x}} = \sum_{i=0}^{\infty}\frac{(4i)!}{16^i\sqrt{2}(2i)!(2i+1)!}x^i,$$

$$\left(\frac{\arcsin x}{x}\right)^2 = \sum_{i=0}^{\infty}\frac{4^i i!^2}{(i+1)(2i+1)!}x^{2i}.$$

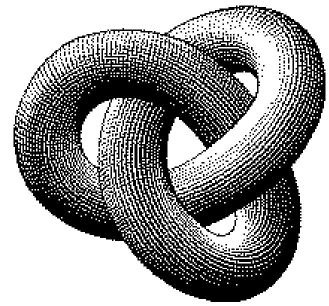
$$\left(\frac{1}{x}\right)^{\overline{-n}} = \sum_{i=0}^{\infty}\left\{\begin{matrix} i \\ n \end{matrix}\right\}x^i,$$

$$(e^x-1)^n = \sum_{i=0}^{\infty}\left\{\begin{matrix} i \\ n \end{matrix}\right\}\frac{n!x^i}{i!},$$

$$x\cot x = \sum_{i=0}^{\infty}\frac{(-4)^iB_{2i}x^{2i}}{(2i)!},$$

$$\zeta(x) = \sum_{i=1}^{\infty}\frac{1}{i^x},$$

$$\frac{\zeta(x-1)}{\zeta(x)} = \sum_{i=1}^{\infty}\frac{\phi(i)}{i^x},$$



Stieltjes Integration

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

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

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

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

If the integrals involved exist

$$\int_a^b \big(G(x)+H(x)\big)\,dF(x) = \int_a^b G(x)\,dF(x) + \int_a^b H(x)\,dF(x),$$

$$\int_a^b G(x)\,d\big(F(x)+H(x)\big) = \int_a^b G(x)\,dF(x) + \int_a^b G(x)\,dH(x),$$

$$\int_a^b c\cdot G(x)\,dF(x) = \int_a^b G(x)\,d\big(c\cdot F(x)\big) = c\int_a^b G(x)\,dF(x),$$

$$\int_a^b G(x)\,dF(x) = G(b)F(b)-G(a)F(a)-\int_a^b F(x)\,dG(x).$$

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

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

Cramer's Rule

If we have equations:

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

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

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

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

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

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

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

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

Fibonacci Numbers

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

Definitions:

$$F_i = F_{i-1}+F_{i-2}, \quad F_0=F_1=1,$$

$$F_{-i} = (-1)^{i-1}F_i,$$

$$F_i = \frac{1}{\sqrt{5}}\left(\phi^i - \hat{\phi}^i\right),$$

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

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

Additive rule:

$$F_{n+k} = F_kF_{n+1} + F_{k-1}F_n,$$

$$F_{2n} = F_nF_{n+1} + F_{n-1}F_n.$$

Calculation by matrices:

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