

# Zoids ACM-ICPC Notebook 2018 (C++)

## Contents

### 1 Data Structures

1.1	Binary Indexed Tree (BIT)	1
1.2	SegmentTree	1
1.3	SegmentTree + LazyPropagation	2
1.4	Implicit Segment Tree	2
1.5	ImplicitSegTree with Lazy Propagation	2
1.6	Persistent Segment Tree	2
1.7	SparseTable1	3
1.8	SparseTable2	3
1.9	CentroidDecomposition	3
1.10	HeavyLightDecomp	3
1.11	HeavyLightDecomp Lazy	4
1.12	ImplicitTreap	5
1.13	LCA	5
1.14	LinkCutTree connectivity	6
1.15	LinkCutTree dynalca	6
1.16	MO Algorithm	7
1.17	PersistentSegTree inTree	7
1.18	Treap	8
1.19	Joshua's Segment Tree with Lazy (APIsh)	8

### 2 Math

2.1	Extended Euclid's Algorithm	9
2.2	PollardRho + MillerRabin	10
2.3	Sieve	10
2.4	Fermat's Little Theorem	10
2.5	Euler's Theorem	10
2.6	Chinese Remainder Theorem	10
2.7	Phi Sieve	11
2.8	Linear Sieve and logarithmic factorization	11
2.9	Fast Fourier Transform	11
2.10	Modular inverse	12
2.11	Mobius Function	12
2.12	Phillai Sieve	12
2.13	Lucas Theorem (small prime moduli and big n and k)	13
2.14	Catalan, dearrangements and other formulas	13

### 3 Flows

3.1	Dinic (Also maximum bipartite matching)	14
3.2	Maximum Flow with upper bound cost	14
3.3	Minimum Cost Maximum Flow	15
3.4	Hungarian Algorithm	15

### 4 Graphs

4.1	Biconnected Components, bridges and articulation points $O(E + V)$	16
4.2	Dijkstra	16
4.3	Bellman Ford (and applications)	17
4.4	Floyd Warshall	17
4.5	SCC (Strongly Connected Components)	18
4.6	2-SAT (with value assignation)	18
4.7	Union - Find	19
4.8	Euler Path	19
4.9	Topological Sort	20
4.10	Adjacency Matrix	20
4.11	Kruskal (Minimum Spanning Tree)	20

### 5 Games

5.1	Nim de la miseria	21
-----	-------------------	----

### 6 DP

6.1	Subsets of the subsets iteration	21
-----	----------------------------------	----

### 7 Strings

7.1	AhoCorasick	21
7.2	SuffixAutomata	21

### 8 Geometry

8.1	geo-enteros	22
8.2	geo-reales	22
8.3	closest-pairs	23
8.4	ConvexHullTrick	23

### 9 Techniques

9.1	Various algorithm techniques	23
-----	------------------------------	----

## 1 Data Structures

### 1.1 Binary Indexed Tree (BIT)

```
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

```
#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

```

#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;
    }
}

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;

```

## 1.6 Persistent 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 <= len * 2) {
        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 <= len * 2) {
        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 CentroidDecomposition

```

vector<int> adj[N];
bool isCentroid[N];
int sizeST[N];

void build_subtrees(int from, int parent) {
    sizeST[from] = 1;
    for (int to: adj[from]) {
        if (to == parent) continue;
        if (isCentroid[to]) continue;
        build_subtrees(to, from);
        sizeST[from] += sizeST[to];
    }
}

int find_centroid(int from, int parent, int size_total) {
    for (int to: adj[from]) {
        if (to == parent) continue;
        if (isCentroid[to]) continue;
        if (sizeST[to] > size_total/2) {
            return find_centroid(to, from, size_total);
        }
    }
    return from;
}

int parentC[N];

int buildCT(int from) {
    build_subtrees(from, 0);
    int centroid = find_centroid(from, 0, sizeST[from]);
    isCentroid[centroid] = true;
    for (int to: adj[centroid]) {
        if (!isCentroid[to]) {
            int centroid_ = buildCT(to);
            parentC[centroid_] = centroid;
        }
    }
    return centroid;
}

```

## 1.10 HeavyLightDecomp

```

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

const int N = 1e5 + 2;
const int INF = 1e9 + 7;

vector<int> adj[N];

/*****/
typedef vector<int> vInt;
typedef vector<vInt> vvInt;

int depth[N];
int parent[N];
int heavy[N];

int dfs(int from, int parent_, int level) {
    depth[from] = level;
    parent[from] = parent_;
    int size = 1;
    int max_size_ST = 0;
    for (int to: adj[from]) {
        if (to == parent_) continue;
        int size_ST = dfs(to, from, level + 1);
        size += size_ST;
        if (max_size_ST < size_ST) {
            max_size_ST = size_ST;
            heavy[from] = to;
        }
    }
    return size;
}

vvInt chain;
int root[N];
int posInChain[N];
int idChain[N];

```

```

int nroChain;

void HeavyLight(int n) {
    dfs(1, 0, 0);
    for (int u = 1; u <= n; u++) {
        if (heavy[parent[u]] == u) continue;
        int pos = 0;
        chain.push_back(vInt());
        for (int v = u; v; v = heavy[v]) {
            root[v] = u;
            posInChain[v] = pos++;
            idChain[v] = nroChain;
            chain[nroChain].push_back(v);
        }
        nroChain++;
    }
}

/*****
vInt tree;

void buildST(int node, int b, int e, int id) {}

void update(int node, int b, int e, int pos, int new_val, int id) {}

int query(int node, int b, int e, int l, int r, int id) {}

/*****/
void updateGraph(int u, int newVal) {
    int id = idChain[u];
    int pos = posInChain[u];
    update(1, 0, chain[id].size() - 1, pos, newVal, id);
}

int queryTree(int id, int u, int v) {
    int l = posInChain[u], r = posInChain[v];
    return query(1, 0, chain[id].size() - 1, l, r, id);
}

int queryGraph(int u, int v) {
    int ans = 0;
    for (; root[u] != root[v]; v = parent[root[v]]) {
        if (depth[root[u]] > depth[root[v]]) swap(u, v);
        ans += queryTree(idChain[v], root[v], v);
    }
    if (depth[u] > depth[v]) swap(u, v);
    ans += queryTree(idChain[v], u, v);
    return ans;
}

/*****/
void buildTrees() {
    tree = vInt(nroChain);
    for (int id = 0; id < nroChain; id++) {
        int size = chain[id].size();
        tree[id] = vInt(size < 2);
        buildST(1, 0, size - 1, id);
    }
}

/*****/
void Clear(int n) {
    tree.clear();
    chain.clear();
    nroChain = 0;
    for (int u = 1; u <= n; u++) {
        adj[u].clear();
        heavy[u] = 0;
    }
}

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

int main() {
    int t, n;
    int u, v;
    cin >> t;
    while (t--) {
        cin >> n;
        Clear(n);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &num[i]);
        }
        for (int i = 1; i < n; i++) {
            scanf("%d %d", &u, &v);
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        HeavyLight(n);
        buildTrees();
        int q, type;

```

```

        cin >> q;
        int newVal;
        while (q--) {
            scanf("%d", &type);
            if (type == 1) {
                scanf("%d %d", &u, &newVal);
                updateGraph(u, newVal);
            }
            if (type == 2) {
                scanf("%d %d", &u, &v);
                printf("%d\n", queryGraph(u, v));
            }
        }
    }
    return 0;
}

```

## 1.11 HeavyLightDecomp Lazy

```

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

const int N = 1e5 + 2;
const int INF = 1e9 + 7;

vector<int> adj[N];

/*****/
typedef vector<int> vInt;
typedef vector<vInt> vvInt;

int depth[N];
int parent[N];
int heavy[N];

int dfs(int from, int parent_, int level) {
    depth[from] = level;
    parent[from] = parent_;
    int size = 1;
    int max_size_ST = 0;
    for (int to: adj[from]) {
        if (to == parent_) continue;
        int size_ST = dfs(to, from, level + 1);
        size += size_ST;
        if (max_size_ST < size_ST) {
            max_size_ST = size_ST;
            heavy[from] = to;
        }
    }
    return size;
}

vInt chain;
int root[N];
int posInChain[N];
int idChain[N];
int nroChain;

void HeavyLight(int n) {
    dfs(1, 0, 0);
    for (int u = 1; u <= n; u++) {
        if (heavy[parent[u]] == u) continue;
        int pos = 0;
        chain.push_back(vInt());
        for (int v = u; v; v = heavy[v]) {
            root[v] = u;
            posInChain[v] = pos++;
            idChain[v] = nroChain;
            chain[nroChain].push_back(v);
        }
        nroChain++;
    }
}

/*****/
vInt tree;
vInt lazy;
#define LEFT(x) (2*x)
#define RIGHT(x) (2*x + 1)

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

```

```

        lazy[id][rightChild] += lazy[id][node];
    }
    lazy[id][node] = 0;
}

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

int query(int node, int b, int e, int l, int r, int id) {
    if (r < b || e < l) return 0;
    aplicate(node, b, e, id);
    if (l <= b && e <= r) return tree[id][node];
    int me = (b + e) >> 1;
    int leftChild = LEFT(node), rightChild = RIGHT(node);
    int q1 = query(leftChild, b, me, l, r, id);
    int q2 = query(rightChild, me + 1, e, l, r, id);
    return q1 + q2;
}

/*****
void updateTree(int id, int u, int v, int add) {
    int l = posInChain[u], r = posInChain[v];
    update(l, 0, chain[id].size() - 1, l, r, add, id);
}

void updateGraph(int u, int v, int add) {
    int ans = 0;
    for (; root[u] != root[v]; v = parent[root[v]]) {
        if (depth[root[u]] > depth[root[v]]) swap(u, v);
        updateTree(idChain[v], root[v], v, add);
    }
    if (depth[u] > depth[v]) swap(u, v);
    updateTree(idChain[v], u, v, add);
}

int queryTree(int id, int u, int v) {
    int l = posInChain[u], r = posInChain[v];
    return query(l, 0, chain[id].size() - 1, l, r, id);
}

int queryGraph(int u, int v) {
    int ans = 0;
    for (; root[u] != root[v]; v = parent[root[v]]) {
        if (depth[root[u]] > depth[root[v]]) swap(u, v);
        ans += queryTree(idChain[v], root[v], v);
    }
    if (depth[u] > depth[v]) swap(u, v);
    ans += queryTree(idChain[v], u, v);
    return ans;
}

/*****
void buildTrees() {
    tree = vvInt(nroChain);
    lazy = vvInt(nroChain);
    for (int id = 0; id < nroChain; id++) {
        int size = chain[id].size();
        tree[id] = vInt(size << 2);
        lazy[id] = vInt(size << 2);
    }
}

/*****
int main() {
    int n, q;
    cin >> n >> q;
    int u, v, k;
    for (int i = 1; i < n; i++) {
        scanf("%d %d", &u, &v);
        u++, v++;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    HeavyLight(n);
    buildTrees();
    while (q--) {
        scanf("%d %d %d", &u, &v, &k);
        u++, v++;
        updateGraph(u, v, k);
    }
}

```

```

    }
    int ans = 0;
    for (int u = 1; u <= n; u++) {
        ans = max(ans, queryGraph(u, u));
    }
    cout << ans;

    return 0;
}

```

## 1.12 ImplicitTreap

```

typedef struct Nd * pnd;
struct Nd {
    int cnt, pri, val;
    pnd l, r;
    bool rev;
    Nd(int val, int pri):cnt(1),pri(pri),val(val),l(NULL),r(NULL),rev(false){}
};

inline int cnt(pnd t) { return t?t->cnt:0; }
inline void upd_cnt(pnd t) { if (t)t->cnt=1+cnt(t->l)+cnt(t->r); }

inline void rev(pnd t) {
    if (t) {
        t->rev = !t->rev;
    }
}

inline void push(pnd t) {
    if (t && t->rev) {
        t->rev=false;
        rev(t->l), rev(t->r);
        swap(t->l, t->r);
    }
}

void print(pnd t, bool end=true) {
    if (t) {
        push(t);
        print(t->l, false);
        if (t->l)cout<<" ";
        cout<<t->val;
        if (t->r)cout<<" ";
        print(t->r, false);
    }
    if (end) cout << endl;
}

void merge(pnd &t, pnd l, pnd r) {
    if (!l || !r) return void(t=l?l:r);
    push(l), push(r);
    if (l->pri > r->pri) merge(l->r, l->r, r), t=l;
    else merge(r->l, l, r->l), t=r;
    upd_cnt(t);
}

void split(pnd t, int i, pnd &l, pnd &r) {
    if (!t) return void(l=r=NULL);
    push(t);
    if (cnt(t->l) < i) split(t->r, i - cnt(t->l) - 1, t->r, r), l=t;
    else split(t->l, i, l, t->l), r=t;
    upd_cnt(l), upd_cnt(r);
}

void rev(pnd &t, int i, int j) {
    pnd a, b, c;
    split(t, j+1, b, c);
    split(b, i, a, b);
    rev(b);
    merge(t, a, b);
    merge(t, t, c);
}

```

## 1.13 LCA

```

vector<int> adj[N];
int depth[N];
int st[N][logN];

void dfs(int from, int parent, int level) {

```

```

depth[from] = level;
st[from][0] = parent;
for (int to: adj[from]) {
    if (to == parent) continue;
    dfs(to, from, level + 1);
}

}

void buildST(int n) {
    dfs(1, 0, 0);
    for (int loglen = 1, len = 2; len <= n; loglen ++, len <= 1) {
        for (int u = 1; u <= n; u++) {
            if (!st[u][loglen]) continue;
            int v = st[u][loglen - 1];
            st[u][loglen] = st[v][loglen - 1];
        }
    }
}

int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    int limit = log2(depth[u]) + 1;
    int len = 1 << limit;
    for (int loglen = limit; loglen >= 0; loglen --, len >= 1) {
        if (depth[u] - len >= depth[v]) {
            u = st[u][loglen];
        }
    }
    if (u == v) return u;
    for (int loglen = limit; loglen >= 0; loglen --) {
        if (!st[u][loglen]) continue;
        if (st[u][loglen] != st[v][loglen]) {
            u = st[u][loglen];
            v = st[v][loglen];
        }
    }
    return st[u][0];
}

```

## 1.14 LinkCutTree connectivity

```

struct Node {
    int left = 0;
    int righth = 0;
    int parent = 0;
    bool revert = 0;
};

Node node[ V ];
char type[ 10 ];
int u, v;

bool isRoot( int u ) {
    int p = node[ u ].parent;
    return !p || ( node[ p ].left != u && node[ p ].righth != u );
}

void push( int x ) {
    if( !node[ x ].revert ) return;
    node[ x ].revert = false;
    swap( node[ x ].left, node[ x ].righth );
    if( node[ x ].left ) node[ node[ x ].left ].revert ^= 1;
    if( node[ x ].righth ) node[ node[ x ].righth ].revert ^= 1;
}

void connect( int ch, int p, int isLeftChild ) {
    if( ch ) node[ ch ].parent = p;
    if( isLeftChild == -1 ) return;
    if( isLeftChild ) node[ p ].left = ch;
    else node[ p ].righth = ch;
}

void rotate( int x ) {
    int p = node[ x ].parent;
    int g = node[ p ].parent;
    bool isRootP = isRoot( p );
    bool leftChildX = x == node[ p ].left;
    connect( leftChildX ? node[ x ].righth : node[ x ].left, p, leftChildX );
    connect( p, x, !leftChildX );
    connect( x, g, !isRootP ? p == node[ g ].left : -1 );
}

void splay( int x ) {
    while( !isRoot( x ) ) {
        int p = node[ x ].parent;
        int g = node[ p ].parent;
        if( !isRoot( p ) )

```

```

        push( g );
        push( p );
        push( x );
        if( !isRoot( p ) ) {
            bool onLeft1 = x == node[ p ].left;
            bool onLeft2 = p == node[ g ].left;
            rotate( onLeft1 == onLeft2 ? p : x );
        }
        rotate( x );
    }
    push( x );
}

int expose( int x ) {
    int last = 0;
    for( int y = x; y; y = node[ y ].parent ) {
        splay( y );
        node[ y ].left = last;
        last = y;
    }
    splay( x );
    return last;
}

void makeRoot( int x ) {
    expose( x );
    node[ x ].revert ^= 1;
}

bool connected( int x, int y ) {
    if( x == y ) return true;
    expose( x );
    expose( y );
    return node[ x ].parent;
}

void link( int x, int y ) {
    if( connected( x, y ) ) return;
    makeRoot( x );
    node[ x ].parent = y;
}

void cut( int x, int y ) {
    makeRoot( x );
    expose( y );
    if( node[ y ].righth != x || node[ x ].left || node[ x ].righth ) return;
    node[ node[ y ].righth ].parent = 0;
    node[ y ].righth = 0;
}

int main() {
    int n, m;
    cin >> n >> m;
    REP( i, m ) {
        scanf( "%s", &type );
        if( type[ 0 ] == 'a' ) {
            sc2( u, v );
            link( u, v );
        }
        if( type[ 0 ] == 'r' ) {
            sc2( u, v );
            cut( u, v );
        }
        if( type[ 0 ] == 'c' ) {
            sc2( u, v );
            if( connected( u, v ) ) printf( "YES\n" );
            else printf( "NO\n" );
        }
    }
    return 0;
}

```

## 1.15 LinkCutTree dynalca

```

struct Node {
    int left = 0;
    int righth = 0;
    int parent = 0;
};

Node node[ V ];
char type[ 10 ];
int u, v;

bool isRoot( int u ) {
    int p = node[ u ].parent;

```

```

    return !p || ( node[ p ].left != u && node[ p ].righ != u );
}

void connect( int ch , int p , int isLeftChild ) {
    if( ch ) node[ ch ].parent = p ;
    if( isLeftChild == -1 ) return ;
    if( isLeftChild ) node[ p ].left = ch ;
    else node[ p ].righ = ch ;
}

void rotate( int x ) {
    int p = node[ x ].parent ;
    int g = node[ p ].parent ;
    bool isRootP = isRoot( p ) ;
    bool leftChildX = ( x == node[ p ].left ) ;
    connect( leftChildX ? node[ x ].righ : node[ x ].left , p , leftChildX ) ;
    connect( p , x , !leftChildX ) ;
    connect( x , g , !isRootP ? p == node[ g ].left : -1 ) ;
}

void splay( int x ) {
    while( !isRoot( x ) ) {
        int p = node[ x ].parent ;
        int g = node[ p ].parent ;
        if( !isRoot( p ) ) {
            bool onLeft1 = x == node[ p ].left ;
            bool onLeft2 = p == node[ g ].left ;
            rotate( onLeft1 == onLeft2 ? p : x ) ;
        }
        rotate( x ) ;
    }
}

int expose( int x ) {
    int last = 0 ;
    for( int y = x ; y ; y = node[ y ].parent ) {
        splay( y ) ;
        node[ y ].left = last ;
        last = y ;
    }
    splay( x ) ;
    return last ;
}

int findRoot( int x ) {
    expose( x ) ;
    while( node[ x ].righ )
        x = node[ x ].righ ;
    splay( x ) ;
    return x ;
}

void link( int x , int y ) {
    if( findRoot( x ) == findRoot( y ) ) return ;
    expose( x ) ;
    // if( node[ x ].righ ) return ;
    node[ x ].parent = y ;
    // node[ y ].left = x ;
}

void cut( int x ) {
    expose( x ) ;
    if( !node[ x ].righ ) return ;
    node[ node[ x ].righ ].parent = 0 ;
    node[ x ].righ = 0 ;
}

int lca( int x , int y ) {
    if( findRoot( x ) != findRoot( y ) ) return 0 ;
    expose( x ) ;
    return expose( y ) ;
}

int main() {
    int n , m ;
    cin >> n >> m ;
    REP( i , m ) {
        scanf( "%s" , &type ) ;
        if( type[ 1 ] == 'i' ) {
            sc2( u , v ) ;
            link( u , v ) ;
        }
        if( type[ 1 ] == 'u' ) {
            sc( u ) ;
            cut( u ) ;
        }
        if( type[ 1 ] == 'c' ) {
            sc2( u , v ) ;
            pf( lca( u , v ) ) ;
        }
    }
}

```

```

    return 0 ;
}

```

## 1.16 MO Algorithm

```

struct Query {
    int l , r , pos ;
};

Query q[N];
int ans[N];
int curr_ans;

void remove( int num ) {}

void add( int num ) {}

int main() {
    int n , m ;
    cin >> n >> m ;
    for( int i = 0 ; i < n ; i ++ ) {
        scanf( "%d" , &num[i] );
    }
    int l , r ;
    for( int i = 0 ; i < m ; i ++ ) {
        scanf( "%d %d" , &l , &r );
        l -- , r -- ; /*indexed of zero+*/
        q[i] = {l , r , i};
    }
    int lenBlock = sqrt( 1.*n ) + 1 ;
    sort( q , q + m , [lenBlock]( Query q1 , Query q2 ){
        if( q1.l/lenBlock != q2.l/lenBlock )
            return q1.l/lenBlock < q2.l/lenBlock ;
        return q1.r < q2.r ;
    });
    int curr_l = 0 ;
    int curr_r = -1 ;
    for( int i = 0 ; i < n ; i ++ ) {
        while( curr_l < q[i].l )
            remove( num[curr_l ++] );
        while( q[i].l < curr_l )
            add( num[-- curr_l] );
        while( q[i].r < curr_r )
            remove( num[curr_r --] );
        while( curr_r < q[i].r )
            add( num[++ curr_r] );
        ans[q[i].pos] = curr_ans ;
    }
    for( int i = 0 ; i < m ; i ++ ) {
        printf( "%d\n" , ans[i] );
    }
    return 0 ;
}

```

## 1.17 PersistantSegTree inTree

```

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 );
}

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]);
    }
}

```

## 1.18 Treap

```

typedef int tnd;

struct Nd {
    tnd key;
    int pri;
    Nd * l, * r;
    Nd() {}
    Nd(tnd key, int pri):key(key), pri(pri), l(NULL), r(NULL) {}
};

typedef Nd * pnd;

void split(pnd t, tnd key, pnd &l, pnd &r) {
    if (!t) l=r=NULL;
    else if (key<t->key) split(t->l, key, l, t->l), r=t;
    else split(t->r, key, t->r, r), l=t;
}

void insert(pnd &t, pnd nnd) {
    if (!t) t=nnd;
    else if (nnd->pri > t->pri) split(t, nnd->key, nnd->l, nnd->r), t=nnd;
    else insert(nnd->key < t->key?t->l:t->r, nnd);
}

void merge(pnd &t, pnd l, pnd r) {
    if (!l) t=r;
    else if (!r) t = l;
    else if (l->pri > r->pri) merge(l->r, l->r, r), t=l;
    else merge(r->l, l, r->l), t=r;
}

void erase(pnd &t, tnd key) {
    if (!t) return;
    else if (t->key==key) merge(t, t->l, t->r);
    else erase(key<t->key?t->l:t->r, key);
}

pnd unite(pnd a, pnd b) {
    if (!a) return b;
    if (!b) return a;
    if (a->pri > b->pri) swap(a, b);

    pnd al, ar;
    split(a, b->key, al, ar);
    b->l = unite(b->l, al);
    b->r = unite(b->r, ar);
    return b;
}

void to_vector(pnd t, vector<tnd> &a) {
    if (!t) return;
    to_vector(t->l, a);
    a.push_back(t->key);
    to_vector(t->r, a);
}

pnd load_treap() {
    int n, x;
    cin >> n;
    pnd ans=NULL;
    while (n--) {
        cin>>x;
        insert(ans, new Nd(x, random()));
    }
    return ans;
}

```

## 1.19 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](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=9999999&category=791&page=show\\_problem&problem=2471](https://uva.onlinejudge.org/index.php?option=onlinejudge&Itemid=9999999&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(n log n)
    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:  
 $a^p \equiv a \pmod p$

And if a is not divisible by p then:  
 $a^{p-1} \equiv 1 \pmod p$

## 2.5 Euler's Theorem

$a^{\phi(n)} \equiv 1 \pmod n$  iff (if and only if) n and a are coprimes

Bonus:

let  $n = p_1^{a_1} * p_2^{a_2} \dots$

$\phi(n) = (p_1 - 1) * p_1^{a_1 - 1} * (p_2 - 1) * p_2^{a_2 - 1} \dots$

$\phi(n) = n * (\text{for each distinct prime 'p' that divides n: the product of } (1 - 1/p))$

## 2.6 Chinese Remainder Theorem

Dados k enteros positivos  $\{n_i\}$ , tales que  $n_i$  y  $n_j$  son coprimos ( $i \neq j$ ).  
 Para cualquier  $\{a_i\}$ , existe x tal que:

$x \equiv a_i \pmod{n_i}$

```

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*ei + 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 a1 (mod m1)
x a2 (mod m2)
...
x an (mod mn)

if, ai aj (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 999911LL

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( vld &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] ) ....
        vld 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 nmero par de factores primos distintos.
mobius(n) = -1 si n es libre de cuadrados y tiene un nmero impar de factores primos distintos.
mobius(n) = 0 si n es divisible por alg 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 ;
}

*****/

//By mariocv
int N = 15;
int mu[N+1];

void mobius() {
    CLR(mu, 0);
    mu[1] = 1;

    for(int i = 1; i <= N; ++i)
        for(int j = 2*i; j <= N; j += i)
            mu[j] -= mu[i];
}

```

## 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  $C_n$  be Catalan number of  $n$ :

$$C_n = \text{binomial}(2n, n) - \text{binomial}(2n, n+1)$$

- \*  $C_n$  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:

```

XXXXYY  XYXXYY  XYXYXY  XXYYXY  XXYXYY.

```

- \* Re-interpreting the symbol X as an open parenthesis and Y as a close parenthesis,  $C_n$  counts the number of expressions containing  $n$  pairs of parentheses which are correctly matched:

```

(( ))      ( ( ) )      ( ) ( )      ( ) ( )      ( ) ( )

```

- \*  $C_n$  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:

```

((abc)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  $C_n$  is the number of full binary trees with  $n+1$  leaves
- \*  $C_n$  is the number of monotonic lattice paths along the edges of a grid with  $n$  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  $C_n$ . The following hexagons illustrate the case  $n=4$ :
- \*  $C_n$  is the number of stack-sortable permutations of  $\{1, \dots, n\}$ . A permutation  $w$  is called stack-sortable if  $S(w) = \{1, \dots, n\}$ , where  $S(w)$  is defined recursively as follows: write  $w = unw$  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.
- \*  $C_n$  is the number of permutations of  $\{1, \dots, n\}$  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.
- \*  $C_n$  is the number of noncrossing partitions of the set  $\{1, \dots, n\}$ . A fortiori,  $C_n$  never exceeds the  $n$ th Bell number.  $C_n$  is also the number of noncrossing partitions of the set  $\{1, \dots, 2n\}$  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.
- \*  $C_n$  is the number of ways to tile a staircase shape of height  $n$  with  $n$  rectangles.



```

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;
}
}

```

### 3.4 Hungarian Algorithm

```

// tested on https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&
// category=problem=1687&mosmsg=Submission+received+with+ID+22270778
// 10746 - Crime Wave - The Sequel
// para problemas rectangulares : O ( n ^ 2 * m )
struct HungarianAlgorithm {
    typedef Double costtype;
    const costtype INF = 1e10;

    // internals
    int n, m;
    vector<costtype> u, v;
    vector<int> p, way;

    HungarianAlgorithm() {}

    // returns the matches of the left set (of size n)
    // given the matrix of costs 'a' (n * m)
    vector<int> solve(vector<vector<costtype>> &a) {
        n = (int)a.size();
        if ( n == 0 ) {
            return vector<int>(0); // empty
        }
        m = (int)a[0].size();

        u = vector<costtype>(n + 1);
        v = vector<costtype>(m + 1);
        p = way = vector<int>(m + 1);
        for (int i=1; i<=n; ++i) {
            p[0] = i;
            int j0 = 0;
            vector<costtype> minv (m+1, INF);
            vector<char> used (m+1, false);
            do {
                used[j0] = true;
                int i0 = p[j0], j1;
                costtype delta = INF;
                for (int j=1; j<=m; ++j)
                    if (!used[j]) {
                        costtype cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                        if (cur < minv[j])
                            minv[j] = cur, way[j] = j0;
                        if (minv[j] < delta)
                            delta = minv[j], j1 = j;
                    }
                for (int j=0; j<=m; ++j)
                    if (used[j])
                        u[p[j]] += delta, v[j] -= delta;
                    else

```

```

        minv[j] -= delta;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}

vector<int> ans (n);
for (int j=1; j<=m; ++j) {
    if (p[j] == 0) { // j - th element (1-indexed) is not matched with anyone in left set
    } else {
        ans[p[j] - 1] = j - 1;
    }
}
return ans;
}
}hung;

vector<vector<Double> > a;
int main() {
    int n, m;
    while (sc(n) == 1) {
        sc(m);
        if (!n) break;

        a = vector<vector<Double> >(n);
        REP (i, n) {
            a[i] = vector<Double>(m);
            REP (j, m) {
                cin >> a[i][j];
            }
        }
        auto ans = hung.solve(a);
        Double avg = 0;
        REP (i, n) {
            avg += a[i][ans[i]];
        }
        avg /= n;

        avg = floor(avg*100.0+0.5+1e-9)/100.0; // never trust printf rounding (do it yourself)
        printf("%.2lf\n", (double)avg);
    }
}

```

## 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 bcomp[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];
                    bcomp[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](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 ^
            // 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)
        */
    }
}

```

// tested on <https://open.kattis.com/problems/allpairspath>

## 4.4 Floyd Warshall

```

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 condicions 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.buildScdDag();
    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)]];
}
}

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( 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);
            }
        }

        // recursively
        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 ni = 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 adyacency 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" ) );
}
}
```

### 7.2 SuffixAutomata

```

struct St {
    int len, link;
    map<char, int> next;
} st[2*N];

int sz = 0, last;

void sa_init() {
    REP( i , sz ) st[i].next.clear();
    sz = last = 0;
    st[last].len = 0;
    st[last].link = -1;
    sz++;
}

void sa_extend(char c) {
    int cur = sz++, p;
    st[last].next[c] = cur;
    st[last].len = st[last].len + 1;
    for (p = st[last].link; p != -1 && !st[p].next.count(c); p = st[p].link)
        st[p].next[c] = cur;

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

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

            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
}
```

## 8 Geometry

### 8.1 geo-enteros

```

struct Pto {
    Long x=0, y=0;
    Pto() {}
    Pto(Long x, Long y): x(x), y(y) {}
    Pto ort() { return Pto(-y, x); }
    Long mod2() { return x*x + y*y; }
    Pto operator +(const Pto &p) const { return Pto(x + p.x, y + p.y); }
    Pto operator -(const Pto &p) const { return Pto(x - p.x, y - p.y); }
    Pto operator *(Long k) const { return Pto(x*k, y*k); }
    Long operator *(const Pto &p) const { return x*p.x + y*p.y; }
    Long operator ^(const Pto &p) const { return x*p.y - y*p.x; }
    bool operator <(const Pto &p) const { return x==p.x?y<p.y:x<p.x; }
    bool operator ==(const Pto &p) const { return x==p.x && y==p.y; }
    bool operator !=(const Pto &p) const { return x!=p.x || y!=p.y; }
};

vector<Pto> chull(vector<Pto> &pts) {
    if (pts.size() < 3) {
        return pts;
    }

    Pto _ref = *min_element(all(pts), [](const Pto &a, const Pto &b) {
        return a.y==b.y?a.x<b.x:a.y<b.y;
    });

    sort(all(pts), [_ref](const Pto &a, const Pto &b) {
        Pto ra = a - _ref, rb = b - _ref;
        Long ar = ra^rb;
        return ar==0?ra.mod2()<rb.mod2():ar>0;
    });

    vector<Pto> ans;
    int i = 0, s;
    while (i < SZ(pts)) {
        s = SZ(ans);
        if (s > 1 && ((pts[i] - ans[s-1])^(ans[s-2] - ans[s-1])) <= 0) ans.pop_back(); // < 0
        else ans.push_back(pts[i++]);
    }

    return ans;
}

bool isConvex(vector<Pto> &pts) {
    int cnt1 = 0, cnt2 = 0, n = SZ(pts);
    REP(i, n) {
        Pto &a = pts[i], &b = pts[(i+1)%n], &c = pts[(i+2)%n];
        Long ar = (c-b)^(a-b);
        if (ar < 0) cnt1++;
        else if (ar > 0) cnt2++;
    }
    return !cnt1 || !cnt2;
}

Long area(vector<Pto> &pts) {
    Long ans = 0;
    int n = SZ(pts);
    REPR(i, 1, n) {
        ans += pts[i]^pts[i-1];
    }
    return abs(ans);
}

```

### 8.2 geo-reales

```

#include<bits/stdc++.h>
#define REP(i,n) for (int i = 0; i < (n); i++)
#define FOR(i,ini,n) for (int i = (ini); i < (n); i++)
#define SZ(a) ((int)a.size())
#define endl "\n"
#define CLR(a, v) memset(a, v, sizeof(a))
#define ALL(v) v.begin(), v.end()
#define sc(x) scanf("%d", &(x))
using namespace std;

typedef long long Long;
typedef long double Double;

```

```

const Double EPS = 1e-8;

struct Pto {
    Double x=0, y=0;
    Pto() {}
    Pto(Double x, Double y): x(x), y(y) {}

    Double mod2() { return x*x + y*y; }
    Double mod() { return sqrt(mod2()); }
    Pto ort() { return Pto(-y, x); }
    Pto unit() { Double k = mod(); return Pto(x/k, y/k); }
    Pto operator +(const Pto &p) const { return Pto(x + p.x, y + p.y); }
    Pto operator -(const Pto &p) const { return Pto(x - p.x, y - p.y); }
    Pto operator *(Double k) const { return Pto(x*k, y*k); }
    Pto operator /(Double k) const { return Pto(x/k, y/k); }
    Double operator *(const Pto &p) const { return x*p.x + y*p.y; }
    Double operator ^(const Pto &p) const { return x*p.y - y*p.x; }
    bool operator <(const Pto &p) const { return fabs(x-p.x)<EPS?y<p.y:x<p.x; }
    bool operator ==(const Pto &p) const { return fabs(x-p.x)<EPS && fabs(y-p.y)<EPS; }
    bool operator !=(const Pto &p) const { return fabs(x-p.x)>EPS || fabs(y-p.y)>EPS; }
};

// Positivo si anti-horario
Double area(vector<Pto> &pts) {
    Double ans = 0;
    int n = SZ(pts);
    FOR(i, 1, n) {
        ans += pts[i]^pts[i-1];
    }
    return abs(ans);
}

// a,b,c
Double area(const Pto &a, const Pto &b, const Pto &c) {
    return (c - b)^(a - b);
}

bool isConvex(vector<Pto> &pts) {
    int cnt1 = 0, cnt2 = 0, n = SZ(pts);
    REP(i, n) {
        Pto &a = pts[i], &b = pts[(i+1)%n], &c = pts[(i+2)%n];
        Double ar = area(a,b,c);
        if (ar < -EPS) cnt1++;
        else if (ar > EPS) cnt2++;
    }
    return !cnt1 || !cnt2;
}

struct Segm {
    Pto s, e;
    Segm() {}
    Segm(Pto s, Pto e): s(s), e(e) {}
    Double len() { return (e-s).mod(); }
    Pto dir() const { return e-s; }
};

bool areParallel(const Segm &a, const Segm &b) {
    return fabs(a.dir()^b.dir()) < EPS;
}

Pto lineIntersect(const Segm &a, const Segm &b) {
    assert(!areParallel(a, b));
    Pto bort = b.dir().ort();
    double r = ((b.s - a.s)*bort)/(b.dir()*bort);
    return a.s + a.dir()*r;
}

bool onLine(const Segm &a, const Pto &b) {
    return fabs((a.s - b)^(a.e - b)) < EPS;
}

bool onSegment(const Segm &a, const Pto &b) {
    if (!onLine(a, b)) return false;
    return (a.s - b)*(a.e - b) < EPS;
}

bool segmIntersect(const Segm &a, const Segm &b) {
    if (areParallel(a, b)) return false;
    Pto I = lineIntersect(a, b);
    return onSegment(a, I) && onSegment(b, I);
}

// Proyeccion de un punto sobre una recta
Pto proy(const Segm &a, const Pto &b) {
    if (onLine(a, b)) return b;
    Pto se = a.dir();
    Double r = ((b - a.s)*se)/(se*se);
    return a.s + se*r;
}

// Reflexion de un punto respecto a una recta
Pto reflect(const Segm &a, const Pto &b) {

```

```

    if (online(a, b)) return b;
    Pto py = proy(a, b);
    return py*2 - b;
}

// Poligono convexo, en sentido antihorario
bool isinPoligon(vector<Pto> &pol, Pto p) {
    int hi = SZ(pol)-1, lo=1, mid;
    Pto pp0 = p - pol[0];
    if ((pp0*(pol[hi] - pol[0])) > EPS
        || ((pol[hi] - pol[0])*pp0) > EPS) return false;

    while (hi - lo > 1) {
        mid = (hi + lo)>>1;
        if ((pp0*(pol[mid] - pol[0])) > EPS) hi = mid;
        else lo = hi;
    }

    return ((pol[hi] - pol[lo])*(p - pol[0])) > EPS;
}

vector<Pto> chull(vector<Pto> &pts) {
    if (pts.size() < 3) {
        return pts;
    }

    Pto _ref = *min_element(ALL(pts), [](const Pto &a, const Pto &b) {
        return abs(a.y-b.y)<EPS?a.x<b.x:a.y<b.y;
    });

    sort(ALL(pts), [_ref](const Pto &a, const Pto &b) {
        Pto ra = a - _ref, rb = b - _ref;
        Long ar = ra*rb;
        return abs(ar)<EPS?ra.mod2()<rb.mod2():ar>0;
    });

    vector<Pto> ans;
    int i = 0, s;
    while (i < SZ(pts)) {
        s = SZ(ans);
        if (s > 1 && ((pts[i] - ans[s-1])*(ans[s-2] - ans[s-1])) <= EPS) ans.pop_back(); // < -EPS
        else ans.push_back(pts[i++]);
    }
    return ans;
}

```

## 8.3 closest-pairs

```

Double closestPair(vector<Pto> &pts) {
    if (SZ(pts) < 2) {
        return 0;
    }

    Double ans = INF;

    sort(ALL(pts));
    set<pair<Double, Double>> stY;

    deque<Pto> deq;
    for (Pto &p: pts) {
        while (SZ(deq) > 0 && p.x - deq.front().x >= ans) {
            stY.erase({deq.front().y, deq.front().x});
            deq.pop_front();
        }

        auto it = stY.lower_bound({p.y - ans, -INF});
        while (it != stY.end() && it->first < p.y + ans) {
            ans = min(ans, (Pto(it->second, it->first) - p).mod());
            it++;
        }

        deq.push_back(p);
        stY.insert({p.y, p.x});
    }
    return ans;
}

```

## 8.4 ConvexHullTrick

```

/*****
const long long isQuery = -(1LL<<62);

```

```

struct Line {
    long long m, b;

    mutable multiset<Line>::iterator it, end;
    const Line* Next(multiset<Line>::iterator it) const {
        return (++it == end ? NULL: &*it);
    }

    bool operator < (const Line &curr) const {
        if (curr.b != isQuery) {
            return m < curr.m;
        }
        long long X = curr.m;
        const Line *nxt = Next(it);
        return (!nxt) ? false : b - nxt->b < (nxt->m - m)*X;
    }
};

struct HullDynamic: public multiset<Line> {
    bool bad(iterator curr) {
        iterator nxt = next(curr);
        if (curr == begin()) {
            if (nxt == end()) {
                return false;
            }
            return (curr->m == nxt->m) && (curr->b <= nxt->b);
        }
        iterator prv = prev(curr);
        if (nxt == end()) {
            return (prv->m == curr->m) && (curr->b <= prv->b);
        }
        return 1.0L*(prv->b - curr->b)*(nxt->m - curr->m) >= 1.0L*(curr->m - prv->m)*(curr->b -
            -nxt->b); // (b1 - b2)/(m2 - m1) = x (coordenada x de la interseccion de L1 y
            L2)
    }

    void add(long long m, long long b) {
        iterator curr = insert((Line){m, b});
        curr->it = curr;
        curr->end = end();
        if (bad(curr)) {
            erase(curr);
            return;
        }
        for (; next(curr) != end() && bad(next(curr)); erase(next(curr)));
        for (; prev(curr) != begin() && bad(prev(curr)); erase(prev(curr)));
    }

    long long f(long long x) {
        Line L = *lower_bound((Line){x, isQuery});
        return L.m*x + L.b;
    }
};

/*****

```

## 9 Techniques

### 9.1 Various algorithm techniques

```

Recursion
Divide and conquer
    Finding interesting points in N log N
Greedy algorithm
    Scheduling
    Max contiguous subvector sum
    Invariants
    Huffman encoding
Graph theory
    Dynamic graphs (extra book-keeping)
    Breadth first search
    Depth first search
    * Normal trees / DFS trees
    Dijkstra's algorithm
    MST: Prim's algorithm
    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

```

- \* Augmenting 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 ( $\Rightarrow$  trees)
- \*  $3^n$  (special case of set cover)
- Diameter and centroid
- K'th shortest path
- Shortest cycle
- Dynamic programming
- 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
- Quadrees
- 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](http://wcipeg.com/wiki/Convex_hull_trick))
- Monotone queues / monotone stacks / sliding queues
- Sliding queue using 2 stacks
- Persistent segment tree

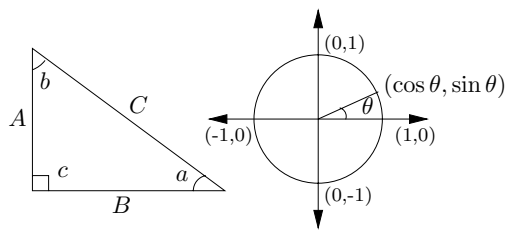
---



$f(n) = O(g(n))$	iff $\exists$ positive $c, n_0$ such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0$ .	$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}.$
$f(n) = \Omega(g(n))$	iff $\exists$ positive $c, n_0$ such that $f(n) \geq cg(n) \geq 0 \forall n \geq n_0$ .	In general:
$f(n) = \Theta(g(n))$	iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ .	$\sum_{i=1}^n i^m = \frac{1}{m+1} \left[ (n+1)^{m+1} - 1 - \sum_{i=1}^n ((i+1)^{m+1} - i^{m+1} - (m+1)i^m) \right]$
$f(n) = o(g(n))$	iff $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$ .	$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}.$
$\lim_{n \rightarrow \infty} a_n = a$	iff $\forall \epsilon > 0, \exists n_0$ such that $ a_n - a  < \epsilon, \forall n \geq n_0$ .	Geometric series:
$\sup S$	least $b \in \mathbb{R}$ such that $b \geq s, \forall s \in S$ .	$\sum_{i=0}^n c^i = \frac{c^{n+1} - 1}{c - 1}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} c^i = \frac{1}{1 - c}, \quad \sum_{i=1}^{\infty} c^i = \frac{c}{1 - c}, \quad  c  < 1,$
$\inf S$	greatest $b \in \mathbb{R}$ such that $b \leq s, \forall s \in S$ .	$\sum_{i=0}^n ic^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, \quad  c  < 1.$
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	Harmonic series:
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$
$\binom{n}{k}$	Combinations: Size $k$ sub-sets of a size $n$ set.	$\sum_{i=1}^n H_i = (n+1)H_n - n, \quad \sum_{i=1}^n \binom{i}{m} H_i = \binom{n+1}{m+1} \left( H_{n+1} - \frac{1}{m+1} \right).$
$[n]$	Stirling numbers (1st kind): Arrangements of an $n$ element set into $k$ cycles.	1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$	Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle \begin{matrix} n \\ k \end{matrix} \rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with $k$ ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle$	2nd order Eulerian numbers.	8. $\sum_{k=0}^n \binom{k}{m} = \binom{n+1}{m+1}, \quad 9. \sum_{k=0}^n \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n},$
$C_n$	Catalan Numbers: Binary trees with $n+1$ vertices.	10. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{matrix} n \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} n \\ n \end{matrix} \right\} = 1,$
14. $\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!,$	15. $\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)!H_{n-1},$	16. $\begin{bmatrix} n \\ n \end{bmatrix} = 1, \quad 17. \begin{bmatrix} n \\ k \end{bmatrix} \geq \left\{ \begin{matrix} n \\ k \end{matrix} \right\},$
18. $\begin{bmatrix} n \\ k \end{bmatrix} = (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix},$	19. $\left\{ \begin{matrix} n \\ n-1 \end{matrix} \right\} = \begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2},$	20. $\sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
22. $\langle \begin{matrix} n \\ 0 \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1 \end{matrix} \rangle = 1,$	23. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = \langle \begin{matrix} n \\ n-1-k \end{matrix} \rangle,$	24. $\langle \begin{matrix} n \\ k \end{matrix} \rangle = (k+1) \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle + (n-k) \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle,$
25. $\langle \begin{matrix} 0 \\ k \end{matrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\langle \begin{matrix} n \\ 1 \end{matrix} \rangle = 2^n - n - 1,$	27. $\langle \begin{matrix} n \\ 2 \end{matrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
28. $x^n = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{x+k}{n},$	29. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	30. $m! \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^n \langle \begin{matrix} n \\ k \end{matrix} \rangle \binom{k}{n-m},$
31. $\langle \begin{matrix} n \\ m \end{matrix} \rangle = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\langle \langle \begin{matrix} n \\ 0 \end{matrix} \rangle \rangle = 1,$	33. $\langle \langle \begin{matrix} n \\ n \end{matrix} \rangle \rangle = 0 \quad \text{for } n \neq 0,$
34. $\langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = (k+1) \langle \langle \begin{matrix} n-1 \\ k \end{matrix} \rangle \rangle + (2n-1-k) \langle \langle \begin{matrix} n-1 \\ k-1 \end{matrix} \rangle \rangle,$	35. $\sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle = \frac{(2n)n}{2^n},$	36. $\left\{ \begin{matrix} x \\ x-n \end{matrix} \right\} = \sum_{k=0}^n \langle \langle \begin{matrix} n \\ k \end{matrix} \rangle \rangle \binom{x+n-1-k}{2n},$
37. $\left\{ \begin{matrix} n+1 \\ m+1 \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} = \sum_{k=0}^n \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (m+1)^{n-k},$		

<p>38. <math>\begin{bmatrix} n+1 \\ m+1 \end{bmatrix} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} = \sum_{k=0}^n \begin{bmatrix} k \\ m \end{bmatrix} n^{n-k} = n! \sum_{k=0}^n \frac{1}{k!} \begin{bmatrix} k \\ m \end{bmatrix},</math></p> <p>40. <math>\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{matrix} k+1 \\ m+1 \end{matrix} \right\} (-1)^{n-k},</math></p> <p>42. <math>\left\{ \begin{matrix} m+n+1 \\ m \end{matrix} \right\} = \sum_{k=0}^m k \left\{ \begin{matrix} n+k \\ k \end{matrix} \right\},</math></p> <p>44. <math>\binom{n}{m} = \sum_k \left\{ \begin{matrix} n+1 \\ k+1 \end{matrix} \right\} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^{m-k},</math> 45. <math>(n-m)! \binom{n}{m} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \left\{ \begin{matrix} k \\ m \end{matrix} \right\} (-1)^{m-k},</math> for <math>n \geq m,</math></p> <p>46. <math>\left\{ \begin{matrix} n \\ n-m \end{matrix} \right\} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix},</math> 47. <math>\begin{bmatrix} n \\ n-m \end{bmatrix} = \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \left\{ \begin{matrix} m+k \\ k \end{matrix} \right\},</math></p> <p>48. <math>\left\{ \begin{matrix} n \\ \ell+m \end{matrix} \right\} \binom{\ell+m}{\ell} = \sum_k \left\{ \begin{matrix} k \\ \ell \end{matrix} \right\} \left\{ \begin{matrix} n-k \\ m \end{matrix} \right\} \binom{n}{k},</math> 49. <math>\begin{bmatrix} n \\ \ell+m \end{bmatrix} \binom{\ell+m}{\ell} = \sum_k \begin{bmatrix} k \\ \ell \end{bmatrix} \begin{bmatrix} n-k \\ m \end{bmatrix} \binom{n}{k}.</math></p>	<p>39. <math>\begin{bmatrix} x \\ x-n \end{bmatrix} = \sum_{k=0}^n \left\langle \begin{matrix} n \\ k \end{matrix} \right\rangle \begin{bmatrix} x+k \\ 2n \end{bmatrix},</math></p> <p>41. <math>\begin{bmatrix} n \\ m \end{bmatrix} = \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^{m-k},</math></p> <p>43. <math>\begin{bmatrix} m+n+1 \\ m \end{bmatrix} = \sum_{k=0}^m k(n+k) \begin{bmatrix} n+k \\ k \end{bmatrix},</math></p>	<p>Every tree with <math>n</math> vertices has <math>n-1</math> edges.</p> <p>Kraft inequality: If the depths of the leaves of a binary tree are <math>d_1, \dots, d_n</math>:</p> $\sum_{i=1}^n 2^{-d_i} \leq 1,$ <p>and equality holds only if every internal node has 2 sons.</p>
Recurrences		
<p>Master method:  <math>T(n) = aT(n/b) + f(n), \quad a \geq 1, b &gt; 1</math>          If <math>\exists \epsilon &gt; 0</math> such that <math>f(n) = O(n^{\log_b a - \epsilon})</math> then  <math>T(n) = \Theta(n^{\log_b a}).</math>          If <math>f(n) = \Theta(n^{\log_b a})</math> then  <math>T(n) = \Theta(n^{\log_b a} \log_2 n).</math>          If <math>\exists \epsilon &gt; 0</math> such that <math>f(n) = \Omega(n^{\log_b a + \epsilon})</math>, and <math>\exists c &lt; 1</math> such that <math>af(n/b) \leq cf(n)</math> for large <math>n</math>, then  <math>T(n) = \Theta(f(n)).</math></p> <p>Substitution (example): Consider the following recurrence  <math>T_{i+1} = 2^{2^i} \cdot T_i^2, \quad T_1 = 2.</math>          Note that <math>T_i</math> is always a power of two. Let <math>t_i = \log_2 T_i</math>. Then we have  <math>t_{i+1} = 2^i + 2t_i, \quad t_1 = 1.</math>          Let <math>u_i = t_i/2^i</math>. Dividing both sides of the previous equation by <math>2^{i+1}</math> we get  <math>\frac{t_{i+1}}{2^{i+1}} = \frac{2^i}{2^{i+1}} + \frac{t_i}{2^i}.</math>          Substituting we find  <math>u_{i+1} = \frac{1}{2} + u_i, \quad u_1 = \frac{1}{2},</math>          which is simply <math>u_i = i/2</math>. So we find that <math>T_i</math> has the closed form <math>T_i = 2^{i2^{i-1}}</math>.          Summing factors (example): Consider the following recurrence  <math>T(n) = 3T(n/2) + n, \quad T(1) = 1.</math>          Rewrite so that all terms involving <math>T</math> are on the left side  <math>T(n) - 3T(n/2) = n.</math>          Now expand the recurrence, and choose a factor which makes the left side “telescope”</p>	<p style="text-align: center;"> <math>1(T(n) - 3T(n/2) = n)</math>  <math>3(T(n/2) - 3T(n/4) = n/2)</math>  <math>\vdots</math>  <math>3^{\log_2 n - 1}(T(2) - 3T(1) = 2)</math> </p> <p>Let <math>m = \log_2 n</math>. Summing the left side we get <math>T(n) - 3^m T(1) = T(n) - 3^m = T(n) - n^k</math> where <math>k = \log_2 3 \approx 1.58496</math>. Summing the right side we get  <math>\sum_{i=0}^{m-1} \frac{n}{2^i} 3^i = n \sum_{i=0}^{m-1} \left(\frac{3}{2}\right)^i.</math></p> <p>Let <math>c = \frac{3}{2}</math>. Then we have  <math>n \sum_{i=0}^{m-1} c^i = n \left( \frac{c^m - 1}{c - 1} \right)</math>  <math>= 2n(c^{\log_2 n} - 1)</math>  <math>= 2n(c^{(\log_2 n) \log_2 c} - 1)</math>  <math>= 2n^k - 2n,</math></p> <p>and so <math>T(n) = 3n^k - 2n</math>. Full history recurrences can often be changed to limited history ones (example): Consider  <math>T_i = 1 + \sum_{j=0}^{i-1} T_j, \quad T_0 = 1.</math></p> <p>Note that  <math>T_{i+1} = 1 + \sum_{j=0}^i T_j.</math></p> <p>Subtracting we find  <math>T_{i+1} - T_i = 1 + \sum_{j=0}^i T_j - 1 - \sum_{j=0}^{i-1} T_j</math>  <math>= T_i.</math></p> <p>And so <math>T_{i+1} = 2T_i = 2^{i+1}.</math></p>	<p>Generating functions:          1. Multiply both sides of the equation by <math>x^i</math>.          2. Sum both sides over all <math>i</math> for which the equation is valid.          3. Choose a generating function <math>G(x)</math>. Usually <math>G(x) = \sum_{i=0}^{\infty} x^i g_i</math>.          3. Rewrite the equation in terms of the generating function <math>G(x)</math>.          4. Solve for <math>G(x)</math>.          5. The coefficient of <math>x^i</math> in <math>G(x)</math> is <math>g_i</math>.</p> <p>Example:  <math>g_{i+1} = 2g_i + 1, \quad g_0 = 0.</math></p> <p>Multiply and sum:  <math>\sum_{i \geq 0} g_{i+1} x^i = \sum_{i \geq 0} 2g_i x^i + \sum_{i \geq 0} x^i.</math></p> <p>We choose <math>G(x) = \sum_{i \geq 0} x^i g_i</math>. Rewrite in terms of <math>G(x)</math>:  <math>\frac{G(x) - g_0}{x} = 2G(x) + \sum_{i \geq 0} x^i.</math></p> <p>Simplify:  <math>\frac{G(x)}{x} = 2G(x) + \frac{1}{1-x}.</math></p> <p>Solve for <math>G(x)</math>:  <math>G(x) = \frac{x}{(1-x)(1-2x)}.</math></p> <p>Expand this using partial fractions:  <math>G(x) = x \left( \frac{2}{1-2x} - \frac{1}{1-x} \right)</math>  <math>= x \left( 2 \sum_{i \geq 0} 2^i x^i - \sum_{i \geq 0} x^i \right)</math>  <math>= \sum_{i \geq 0} (2^{i+1} - 1) x^{i+1}.</math></p> <p>So <math>g_i = 2^i - 1.</math></p>

$n \sim 0.11109,$			$\psi - 2 \sim 1.01009,$	$\psi - 2 \sim .01009$
$i$	$2^i$	$p_i$	General	Probability
1	2	2	Bernoulli Numbers ( $B_i = 0$ , odd $i \neq 1$ ):	Continuous distributions: If
2	4	3	$B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30},$	$\Pr[a < X < b] = \int_a^b p(x) dx,$
3	8	5	$B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}.$	then $p$ is the probability density function of $X$ . If
4	16	7	Change of base, quadratic formula:	$\Pr[X < a] = P(a),$
5	32	11	$\log_b x = \frac{\log_a x}{\log_a b}, \quad \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$	then $P$ is the distribution function of $X$ . If $P$ and $p$ both exist then
6	64	13	Euler's number $e$ :	$P(a) = \int_{-\infty}^a p(x) dx.$
7	128	17	$e = 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$	Expectation: If $X$ is discrete
8	256	19	$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x.$	$E[g(X)] = \sum_x g(x) \Pr[X = x].$
9	512	23	$\left(1 + \frac{1}{n}\right)^n < e < \left(1 + \frac{1}{n}\right)^{n+1}.$	If $X$ continuous then
10	1,024	29	$\left(1 + \frac{1}{n}\right)^n = e - \frac{e}{2n} + \frac{11e}{24n^2} - O\left(\frac{1}{n^3}\right).$	$E[g(X)] = \int_{-\infty}^{\infty} g(x)p(x) dx = \int_{-\infty}^{\infty} g(x) dP(x).$
11	2,048	31	Harmonic numbers:	Variance, standard deviation:
12	4,096	37	$1, \frac{3}{2}, \frac{11}{6}, \frac{25}{12}, \frac{137}{60}, \frac{49}{20}, \frac{363}{140}, \frac{761}{280}, \frac{7129}{2520}, \dots$	$\text{VAR}[X] = E[X^2] - E[X]^2,$
13	8,192	41	$\ln n < H_n < \ln n + 1,$	$\sigma = \sqrt{\text{VAR}[X]}.$
14	16,384	43	$H_n = \ln n + \gamma + O\left(\frac{1}{n}\right).$	For events $A$ and $B$ :
15	32,768	47	Factorial, Stirling's approximation:	$\Pr[A \vee B] = \Pr[A] + \Pr[B] - \Pr[A \wedge B]$
16	65,536	53	$1, 2, 6, 24, 120, 720, 5040, 40320, 362880, \dots$	$\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B],$
17	131,072	59	$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right).$	iff $A$ and $B$ are independent.
18	262,144	61	Ackermann's function and inverse:	$\Pr[A B] = \frac{\Pr[A \wedge B]}{\Pr[B]}$
19	524,288	67	$a(i, j) = \begin{cases} 2^j & i = 1 \\ a(i-1, 2) & j = 1 \\ a(i-1, a(i, j-1)) & i, j \geq 2 \end{cases}$	For random variables $X$ and $Y$ :
20	1,048,576	71	$\alpha(i) = \min\{j \mid a(j, j) \geq i\}.$	$E[X \cdot Y] = E[X] \cdot E[Y],$
21	2,097,152	73		if $X$ and $Y$ are independent.
22	4,194,304	79		$E[X + Y] = E[X] + E[Y],$
23	8,388,608	83		$E[cX] = cE[X].$
24	16,777,216	89	Binomial distribution:	Bayes' theorem:
25	33,554,432	97	$\Pr[X = k] = \binom{n}{k} p^k q^{n-k}, \quad q = 1 - p,$	$\Pr[A_i B] = \frac{\Pr[B A_i] \Pr[A_i]}{\sum_{j=1}^n \Pr[A_j] \Pr[B A_j]}.$
26	67,108,864	101	$E[X] = \sum_{k=1}^n k \binom{n}{k} p^k q^{n-k} = np.$	Inclusion-exclusion:
27	134,217,728	103	Poisson distribution:	$\Pr\left[\bigvee_{i=1}^n X_i\right] = \sum_{i=1}^n \Pr[X_i] +$
28	268,435,456	107	$\Pr[X = k] = \frac{e^{-\lambda} \lambda^k}{k!}, \quad E[X] = \lambda.$	$\sum_{k=2}^n (-1)^{k+1} \sum_{i_1 < \dots < i_k} \Pr\left[\bigwedge_{j=1}^k X_{i_j}\right].$
29	536,870,912	109	Normal (Gaussian) distribution:	Moment inequalities:
30	1,073,741,824	113	$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}, \quad E[X] = \mu.$	$\Pr[ X  \geq \lambda E[X]] \leq \frac{1}{\lambda},$
31	2,147,483,648	127	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	$\Pr[ X - E[X]  \geq \lambda \cdot \sigma] \leq \frac{1}{\lambda^2}.$
32	4,294,967,296	131	$nH_n.$	Geometric distribution:
Pascal's Triangle				$\Pr[X = k] = pq^{k-1}, \quad q = 1 - p,$
1				$E[X] = \sum_{k=1}^{\infty} kpq^{k-1} = \frac{1}{p}.$
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				



Pythagorean theorem:

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

Definitions:

$$\begin{aligned}\sin a &= A/C, & \cos a &= B/C, \\ \csc a &= C/A, & \sec a &= C/B, \\ \tan a &= \frac{\sin a}{\cos a} = \frac{A}{B}, & \cot a &= \frac{\cos a}{\sin a} = \frac{B}{A}.\end{aligned}$$

Area, radius of inscribed circle:

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

Identities:

$$\sin x = \frac{1}{\csc x}, \quad \cos x = \frac{1}{\sec x},$$

$$\tan x = \frac{1}{\cot x}, \quad \sin^2 x + \cos^2 x = 1,$$

$$1 + \tan^2 x = \sec^2 x, \quad 1 + \cot^2 x = \csc^2 x,$$

$$\sin x = \cos\left(\frac{\pi}{2} - x\right), \quad \sin x = \sin(\pi - x),$$

$$\cos x = -\cos(\pi - x), \quad \tan x = \cot\left(\frac{\pi}{2} - x\right),$$

$$\cot x = -\cot(\pi - x), \quad \csc x = \cot\frac{x}{2} - \cot x,$$

$$\sin(x \pm y) = \sin x \cos y \pm \cos x \sin y,$$

$$\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y,$$

$$\tan(x \pm y) = \frac{\tan x \pm \tan y}{1 \mp \tan x \tan y},$$

$$\cot(x \pm y) = \frac{\cot x \cot y \mp 1}{\cot x \pm \cot y},$$

$$\sin 2x = 2 \sin x \cos x, \quad \sin 2x = \frac{2 \tan x}{1 + \tan^2 x},$$

$$\cos 2x = \cos^2 x - \sin^2 x, \quad \cos 2x = 2 \cos^2 x - 1,$$

$$\cos 2x = 1 - 2 \sin^2 x, \quad \cos 2x = \frac{1 - \tan^2 x}{1 + \tan^2 x},$$

$$\tan 2x = \frac{2 \tan x}{1 - \tan^2 x}, \quad \cot 2x = \frac{\cot^2 x - 1}{2 \cot x},$$

$$\sin(x + y) \sin(x - y) = \sin^2 x - \sin^2 y,$$

$$\cos(x + y) \cos(x - y) = \cos^2 x - \sin^2 y.$$

Euler's equation:

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

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.

$$\det A \cdot B = \det A \cdot \det B,$$

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

$2 \times 2$  and  $3 \times 3$  determinant:

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

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = g \begin{vmatrix} b & c \\ e & f \end{vmatrix} - h \begin{vmatrix} a & c \\ d & f \end{vmatrix} + i \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$

$$= aei + bfg + cdh - ceg - fha - ibd.$$

Permanents:

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

### Hyperbolic Functions

Definitions:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \quad \cosh x = \frac{e^x + e^{-x}}{2},$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{csch } x = \frac{1}{\sinh x},$$

$$\text{sech } x = \frac{1}{\cosh x}, \quad \coth x = \frac{1}{\tanh x}.$$

Identities:

$$\cosh^2 x - \sinh^2 x = 1, \quad \tanh^2 x + \text{sech}^2 x = 1,$$

$$\coth^2 x - \text{csch}^2 x = 1, \quad \sinh(-x) = -\sinh x,$$

$$\cosh(-x) = \cosh x, \quad \tanh(-x) = -\tanh x,$$

$$\sinh(x + y) = \sinh x \cosh y + \cosh x \sinh y,$$

$$\cosh(x + y) = \cosh x \cosh y + \sinh x \sinh y,$$

$$\sinh 2x = 2 \sinh x \cosh x,$$

$$\cosh 2x = \cosh^2 x + \sinh^2 x,$$

$$\cosh x + \sinh x = e^x, \quad \cosh x - \sinh x = e^{-x},$$

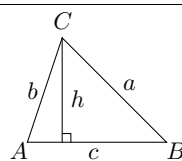
$$(\cosh x + \sinh x)^n = \cosh nx + \sinh nx, \quad n \in \mathbb{Z},$$

$$2 \sinh^2 \frac{x}{2} = \cosh x - 1, \quad 2 \cosh^2 \frac{x}{2} = \cosh x + 1.$$

$\theta$	$\sin \theta$	$\cos \theta$	$\tan \theta$
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	$\infty$

... in mathematics  
you don't under-  
stand things, you  
just get used to  
them.

– J. von Neumann



Law of cosines:

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

Area:

$$\begin{aligned}A &= \frac{1}{2}hc, \\ &= \frac{1}{2}ab \sin C, \\ &= \frac{c^2 \sin A \sin B}{2 \sin C}.\end{aligned}$$

Heron's formula:

$$A = \sqrt{s \cdot s_a \cdot s_b \cdot s_c},$$

$$s = \frac{1}{2}(a + b + c),$$

$$s_a = s - a,$$

$$s_b = s - b,$$

$$s_c = s - c.$$

More identities:

$$\sin \frac{x}{2} = \sqrt{\frac{1 - \cos x}{2}},$$

$$\cos \frac{x}{2} = \sqrt{\frac{1 + \cos x}{2}},$$

$$\tan \frac{x}{2} = \sqrt{\frac{1 - \cos x}{1 + \cos x}},$$

$$= \frac{1 - \cos x}{\sin x},$$

$$= \frac{\sin x}{1 + \cos x},$$

$$\cot \frac{x}{2} = \sqrt{\frac{1 + \cos x}{1 - \cos x}},$$

$$= \frac{1 + \cos x}{\sin x},$$

$$= \frac{\sin x}{1 - \cos x},$$

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i},$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2},$$

$$\tan x = -i \frac{e^{ix} - e^{-ix}}{e^{ix} + e^{-ix}},$$

$$= -i \frac{e^{2ix} - 1}{e^{2ix} + 1},$$

$$\sin x = \frac{\sinh ix}{i},$$

$$\cos x = \cosh ix,$$

$$\tan x = \frac{\tanh ix}{i}.$$

v2.02 ©1994 by Steve Seiden

sseiden@acm.org

<http://www.csc.lsu.edu/~seiden>

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

$$C \equiv r_1 \pmod{m_1}$$

$$\vdots \quad \vdots \quad \vdots$$

$$C \equiv r_n \pmod{m_n}$$

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

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

Fermat's theorem:

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

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

$$(n-1)! \equiv -1 \pmod{n}.$$

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  $k \cdot c(G-S) \leq |S|$ .

*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 embedded 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  $\leq 5$ .

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

$r(k, \ell)$  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

$$(x, y) \quad (x, y, 1)$$

$$y = mx + b \quad (m, -1, b)$$

$$x = c \quad (1, 0, -c)$$

Distance formula,  $L_p$  and  $L_\infty$  metric:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

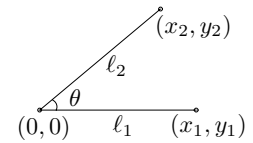
$$[|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p},$$

$$\lim_{p \rightarrow \infty} [|x_1 - x_0|^p + |y_1 - y_0|^p]^{1/p}.$$

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

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

Angle formed by three points:



$$\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{l_1 l_2}.$$

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:

$$A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$$

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:

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

Gregory's series:

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

Newton's series:

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

Sharp's series:

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

Euler's series:

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

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2} + \cdots$$

$$\frac{\pi^2}{12} = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \cdots$$

#### 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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Integrals:

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

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

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

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

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

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

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

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.  $\int \sin^2(ax) dx = \frac{1}{2a}(ax - \sin(ax) \cos(ax)),$
18.  $\int \cos^2(ax) dx = \frac{1}{2a}(ax + \sin(ax) \cos(ax)),$
19.  $\int \sec^2 x dx = \tan x,$
20.  $\int \csc^2 x dx = -\cot x,$
21.  $\int \sin^n x dx = -\frac{\sin^{n-1} x \cos x}{n} + \frac{n-1}{n} \int \sin^{n-2} x dx,$
22.  $\int \cos^n x dx = \frac{\cos^{n-1} x \sin x}{n} + \frac{n-1}{n} \int \cos^{n-2} x dx,$
23.  $\int \tan^n x dx = \frac{\tan^{n-1} x}{n-1} - \int \tan^{n-2} x dx, \quad n \neq 1,$
24.  $\int \cot^n x dx = -\frac{\cot^{n-1} x}{n-1} - \int \cot^{n-2} x dx, \quad n \neq 1,$
25.  $\int \sec^n x dx = \frac{\tan x \sec^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2} x dx, \quad n \neq 1,$
26.  $\int \csc^n x dx = -\frac{\cot x \csc^{n-1} x}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2} x dx, \quad n \neq 1,$
27.  $\int \sinh x dx = \cosh x,$
28.  $\int \cosh x dx = \sinh x,$
29.  $\int \tanh x dx = \ln |\cosh x|,$
30.  $\int \coth x dx = \ln |\sinh x|,$
31.  $\int \operatorname{sech} x dx = \arctan \sinh x,$
32.  $\int \operatorname{csch} x dx = \ln \left| \tanh \frac{x}{2} \right|,$
33.  $\int \sinh^2 x dx = \frac{1}{4} \sinh(2x) - \frac{1}{2} x,$
34.  $\int \cosh^2 x dx = \frac{1}{4} \sinh(2x) + \frac{1}{2} x,$
35.  $\int \operatorname{sech}^2 x dx = \tanh x,$
36.  $\int \operatorname{arcsinh} \frac{x}{a} dx = x \operatorname{arcsinh} \frac{x}{a} - \sqrt{x^2 + a^2}, \quad a > 0,$
37.  $\int \operatorname{arctanh} \frac{x}{a} dx = x \operatorname{arctanh} \frac{x}{a} + \frac{a}{2} \ln |a^2 - x^2|,$
38.  $\int \operatorname{arccosh} \frac{x}{a} dx = \begin{cases} x \operatorname{arccosh} \frac{x}{a} - \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} > 0 \text{ and } a > 0, \\ x \operatorname{arccosh} \frac{x}{a} + \sqrt{x^2 + a^2}, & \text{if } \operatorname{arccosh} \frac{x}{a} < 0 \text{ and } a > 0, \end{cases}$
39.  $\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln \left( x + \sqrt{a^2 + x^2} \right), \quad a > 0,$
40.  $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \arctan \frac{x}{a}, \quad a > 0,$
41.  $\int \sqrt{a^2 - x^2} dx = \frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
42.  $\int (a^2 - x^2)^{3/2} dx = \frac{x}{8} (5a^2 - 2x^2) \sqrt{a^2 - x^2} + \frac{3a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
43.  $\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a}, \quad a > 0,$
44.  $\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left| \frac{a+x}{a-x} \right|,$
45.  $\int \frac{dx}{(a^2 - x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 - x^2}},$
46.  $\int \sqrt{a^2 \pm x^2} dx = \frac{x}{2} \sqrt{a^2 \pm x^2} \pm \frac{a^2}{2} \ln \left| x + \sqrt{a^2 \pm x^2} \right|,$
47.  $\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln \left| x + \sqrt{x^2 - a^2} \right|, \quad a > 0,$
48.  $\int \frac{dx}{ax^2 + bx} = \frac{1}{a} \ln \left| \frac{x}{a+bx} \right|,$
49.  $\int x \sqrt{a+bx} dx = \frac{2(3bx-2a)(a+bx)^{3/2}}{15b^2},$
50.  $\int \frac{\sqrt{a+bx}}{x} dx = 2\sqrt{a+bx} + a \int \frac{1}{x\sqrt{a+bx}} dx,$
51.  $\int \frac{x}{\sqrt{a+bx}} dx = \frac{1}{\sqrt{2}} \ln \left| \frac{\sqrt{a+bx} - \sqrt{a}}{\sqrt{a+bx} + \sqrt{a}} \right|, \quad a > 0,$
52.  $\int \frac{\sqrt{a^2 - x^2}}{x} dx = \sqrt{a^2 - x^2} - a \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
53.  $\int x \sqrt{a^2 - x^2} dx = -\frac{1}{3} (a^2 - x^2)^{3/2},$
54.  $\int x^2 \sqrt{a^2 - x^2} dx = \frac{x}{8} (2x^2 - a^2) \sqrt{a^2 - x^2} + \frac{a^4}{8} \arcsin \frac{x}{a}, \quad a > 0,$
55.  $\int \frac{dx}{\sqrt{a^2 - x^2}} = -\frac{1}{a} \ln \left| \frac{a + \sqrt{a^2 - x^2}}{x} \right|,$
56.  $\int \frac{x dx}{\sqrt{a^2 - x^2}} = -\sqrt{a^2 - x^2},$
57.  $\int \frac{x^2 dx}{\sqrt{a^2 - x^2}} = -\frac{x}{2} \sqrt{a^2 - x^2} + \frac{a^2}{2} \arcsin \frac{x}{a}, \quad a > 0,$
58.  $\int \frac{\sqrt{a^2 + x^2}}{x} dx = \sqrt{a^2 + x^2} - a \ln \left| \frac{a + \sqrt{a^2 + x^2}}{x} \right|,$
59.  $\int \frac{\sqrt{x^2 - a^2}}{x} dx = \sqrt{x^2 - a^2} - a \arccos \frac{a}{|x|}, \quad a > 0,$
60.  $\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2},$
61.  $\int \frac{dx}{x \sqrt{x^2 + a^2}} = \frac{1}{a} \ln \left| \frac{x}{a + \sqrt{a^2 + x^2}} \right|,$

$$\begin{aligned}
62. \int \frac{dx}{x\sqrt{x^2-a^2}} &= \frac{1}{a} \arccos \frac{a}{|x|}, \quad a > 0, & 63. \int \frac{dx}{x^2\sqrt{x^2 \pm a^2}} &= \mp \frac{\sqrt{x^2 \pm a^2}}{a^2 x}, \\
64. \int \frac{x dx}{\sqrt{x^2 \pm a^2}} &= \sqrt{x^2 \pm a^2}, & 65. \int \frac{\sqrt{x^2 \pm a^2}}{x^4} dx &= \mp \frac{(x^2 + a^2)^{3/2}}{3a^2 x^3}, \\
66. \int \frac{dx}{ax^2 + bx + c} &= \begin{cases} \frac{1}{\sqrt{b^2 - 4ac}} \ln \left| \frac{2ax + b - \sqrt{b^2 - 4ac}}{2ax + b + \sqrt{b^2 - 4ac}} \right|, & \text{if } b^2 > 4ac, \\ \frac{2}{\sqrt{4ac - b^2}} \arctan \frac{2ax + b}{\sqrt{4ac - b^2}}, & \text{if } b^2 < 4ac, \end{cases} \\
67. \int \frac{dx}{\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{1}{\sqrt{a}} \ln \left| 2ax + b + 2\sqrt{a}\sqrt{ax^2 + bx + c} \right|, & \text{if } a > 0, \\ \frac{1}{\sqrt{-a}} \arcsin \frac{-2ax - b}{\sqrt{b^2 - 4ac}}, & \text{if } a < 0, \end{cases} \\
68. \int \sqrt{ax^2 + bx + c} dx &= \frac{2ax + b}{4a} \sqrt{ax^2 + bx + c} + \frac{4ac - b^2}{8a} \int \frac{dx}{\sqrt{ax^2 + bx + c}}, \\
69. \int \frac{x dx}{\sqrt{ax^2 + bx + c}} &= \frac{\sqrt{ax^2 + bx + c}}{a} - \frac{b}{2a} \int \frac{dx}{\sqrt{ax^2 + bx + c}}, \\
70. \int \frac{dx}{x\sqrt{ax^2 + bx + c}} &= \begin{cases} \frac{-1}{\sqrt{c}} \ln \left| \frac{2\sqrt{c}\sqrt{ax^2 + bx + c} + bx + 2c}{x} \right|, & \text{if } c > 0, \\ \frac{1}{\sqrt{-c}} \arcsin \frac{bx + 2c}{|x|\sqrt{b^2 - 4ac}}, & \text{if } c < 0, \end{cases} \\
71. \int x^3 \sqrt{x^2 + a^2} dx &= \left(\frac{1}{3}x^2 - \frac{2}{15}a^2\right)(x^2 + a^2)^{3/2}, \\
72. \int x^n \sin(ax) dx &= -\frac{1}{a}x^n \cos(ax) + \frac{n}{a} \int x^{n-1} \cos(ax) dx, \\
73. \int x^n \cos(ax) dx &= \frac{1}{a}x^n \sin(ax) - \frac{n}{a} \int x^{n-1} \sin(ax) dx, \\
74. \int x^n e^{ax} dx &= \frac{x^n e^{ax}}{a} - \frac{n}{a} \int x^{n-1} e^{ax} dx, \\
75. \int x^n \ln(ax) dx &= x^{n+1} \left( \frac{\ln(ax)}{n+1} - \frac{1}{(n+1)^2} \right), \\
76. \int x^n (\ln ax)^m dx &= \frac{x^{n+1}}{n+1} (\ln ax)^m - \frac{m}{n+1} \int x^n (\ln ax)^{m-1} dx.
\end{aligned}$$

$$\begin{aligned}
x^1 &= x^1 & x^{\bar{1}} &= x^{\bar{1}} \\
x^2 &= x^2 + x^1 & x^{\bar{2}} &= x^{\bar{2}} - x^{\bar{1}} \\
x^3 &= x^3 + 3x^2 + x^1 & x^{\bar{3}} &= x^{\bar{3}} - 3x^{\bar{2}} + x^{\bar{1}} \\
x^4 &= x^4 + 6x^3 + 7x^2 + x^1 & x^{\bar{4}} &= x^{\bar{4}} - 6x^{\bar{3}} + 7x^{\bar{2}} - x^{\bar{1}} \\
x^5 &= x^5 + 15x^4 + 25x^3 + 10x^2 + x^1 & x^{\bar{5}} &= x^{\bar{5}} - 15x^{\bar{4}} + 25x^{\bar{3}} - 10x^{\bar{2}} + x^{\bar{1}} \\
x^{\bar{1}} &= x^1 & x^1 &= x^1 \\
x^{\bar{2}} &= x^2 + x^1 & x^2 &= x^2 - x^1 \\
x^{\bar{3}} &= x^3 + 3x^2 + 2x^1 & x^3 &= x^3 - 3x^2 + 2x^1 \\
x^{\bar{4}} &= x^4 + 6x^3 + 11x^2 + 6x^1 & x^4 &= x^4 - 6x^3 + 11x^2 - 6x^1 \\
x^{\bar{5}} &= x^5 + 10x^4 + 35x^3 + 50x^2 + 24x^1 & x^{\bar{5}} &= x^5 - 10x^4 + 35x^3 - 50x^2 + 24x^1
\end{aligned}$$

Difference, shift operators:

$$\begin{aligned}
\Delta f(x) &= f(x+1) - f(x), \\
\mathbb{E} f(x) &= f(x+1).
\end{aligned}$$

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:

$$\begin{aligned}
\Delta(cu) &= c\Delta u, & \Delta(u+v) &= \Delta u + \Delta v, \\
\Delta(uv) &= u\Delta v + \mathbb{E} v \Delta u, \\
\Delta(x^n) &= nx^{n-1}, \\
\Delta(H_x) &= x^{-1}, & \Delta(2^x) &= 2^x, \\
\Delta(c^x) &= (c-1)c^x, & \Delta\binom{x}{m} &= \binom{x}{m-1}.
\end{aligned}$$

Sums:

$$\begin{aligned}
\sum cu \delta x &= c \sum u \delta x, \\
\sum (u+v) \delta x &= \sum u \delta x + \sum v \delta x, \\
\sum u \Delta v \delta x &= uv - \sum \mathbb{E} v \Delta u \delta x, \\
\sum x^n \delta x &= \frac{x^{n+1}}{n+1}, & \sum x^{-1} \delta x &= H_x, \\
\sum c^x \delta x &= \frac{c^x}{c-1}, & \sum \binom{x}{m} \delta x &= \binom{x}{m+1}.
\end{aligned}$$

Falling Factorial Powers:

$$\begin{aligned}
x^{\underline{n}} &= x(x-1) \cdots (x-n+1), \quad n > 0, \\
x^{\underline{0}} &= 1, \\
x^{\underline{n}} &= \frac{1}{(x+1) \cdots (x+|n|)}, \quad n < 0, \\
x^{\underline{n+m}} &= x^{\underline{m}}(x-m)^{\underline{n}}.
\end{aligned}$$

Rising Factorial Powers:

$$\begin{aligned}
x^{\overline{n}} &= x(x+1) \cdots (x+n-1), \quad n > 0, \\
x^{\overline{0}} &= 1, \\
x^{\overline{n}} &= \frac{1}{(x-1) \cdots (x-|n|)}, \quad n < 0, \\
x^{\overline{n+m}} &= x^{\overline{m}}(x+m)^{\overline{n}}.
\end{aligned}$$

Conversion:

$$\begin{aligned}
x^{\underline{n}} &= (-1)^n (-x)^{\overline{n}} = (x-n+1)^{\overline{n}} \\
&= 1/(x+1)^{-\overline{n}}, \\
x^{\overline{n}} &= (-1)^n (-x)^{\underline{n}} = (x+n-1)^{\underline{n}} \\
&= 1/(x-1)^{-\underline{n}}, \\
x^n &= \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{\underline{k}} = \sum_{k=1}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}}, \\
x^{\underline{n}} &= \sum_{k=1}^n \left[ \begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k, \\
x^{\overline{n}} &= \sum_{k=1}^n \left[ \begin{matrix} n \\ k \end{matrix} \right] x^k.
\end{aligned}$$



Taylor's series:

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

Expansions:

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

Ordinary power series:

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

Exponential power series:

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

Dirichlet power series:

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

Binomial theorem:

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

Difference of like powers:

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

For ordinary power series:

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

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

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

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

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

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

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

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

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

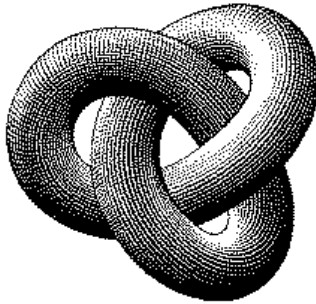
Summation: If  $b_i = \sum_{j=0}^i a_j$  then

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

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:																																																																																																								
$\frac{1}{(1-x)^{n+1}} \ln \frac{1}{1-x}$	$= \sum_{i=0}^{\infty} (H_{n+i} - H_n) \binom{n+i}{i} x^i,$		$\left(\frac{1}{x}\right)^{-n}$	$= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} x^i,$																																																																																																				
$x^{\overline{n}}$	$= \sum_{i=0}^{\infty} \left[ \begin{matrix} n \\ i \end{matrix} \right] x^i,$		$(e^x - 1)^n$	$= \sum_{i=0}^{\infty} \left\{ \begin{matrix} i \\ n \end{matrix} \right\} \frac{n! x^i}{i!},$																																																																																																				
$\left(\ln \frac{1}{1-x}\right)^n$	$= \sum_{i=0}^{\infty} \left[ \begin{matrix} i \\ n \end{matrix} \right] \frac{n! x^i}{i!},$		$x \cot x$	$= \sum_{i=0}^{\infty} \frac{(-4)^i B_{2i} x^{2i}}{(2i)!},$																																																																																																				
$\tan x$	$= \sum_{i=1}^{\infty} (-1)^{i-1} \frac{2^{2i} (2^{2i} - 1) B_{2i} x^{2i-1}}{(2i)!},$	$\zeta(x)$	$= \sum_{i=1}^{\infty} \frac{1}{i^x},$																																																																																																					
$\frac{1}{\zeta(x)}$	$= \sum_{i=1}^{\infty} \frac{\mu(i)}{i^x},$	$\frac{\zeta(x-1)}{\zeta(x)}$	$= \sum_{i=1}^{\infty} \frac{\phi(i)}{i^x},$																																																																																																					
$\zeta(x)$	$= \prod_p \frac{1}{1 - p^{-x}},$																																																																																																							
$\zeta^2(x)$	$= \sum_{i=1}^{\infty} \frac{d(i)}{x^i} \quad \text{where } d(n) = \sum_{d n} 1,$	Stieltjes Integration																																																																																																						
$\zeta(x)\zeta(x-1)$	$= \sum_{i=1}^{\infty} \frac{S(i)}{x^i} \quad \text{where } S(n) = \sum_{d n} d,$	If $G$ is continuous in the interval $[a, b]$ and $F$ is nondecreasing then																																																																																																						
$\zeta(2n)$	$= \frac{2^{2n-1}  B_{2n} }{(2n)!} \pi^{2n}, \quad n \in \mathbb{N},$	$\int_a^b G(x) dF(x)$																																																																																																						
$\frac{x}{\sin x}$	$= \sum_{i=0}^{\infty} (-1)^{i-1} \frac{(4^i - 2) B_{2i} x^{2i}}{(2i)!},$	exists. If $a \leq b \leq c$ then																																																																																																						
$\left(\frac{1 - \sqrt{1-4x}}{2x}\right)^n$	$= \sum_{i=0}^{\infty} \frac{n(2i+n-1)!}{i!(n+i)!} x^i,$	$\int_a^c G(x) dF(x) = \int_a^b G(x) dF(x) + \int_b^c G(x) dF(x).$																																																																																																						
$e^x \sin x$	$= \sum_{i=1}^{\infty} \frac{2^{i/2} \sin \frac{i\pi}{4}}{i!} x^i,$	If the integrals involved exist																																																																																																						
$\sqrt{\frac{1 - \sqrt{1-x}}{x}}$	$= \sum_{i=0}^{\infty} \frac{(4i)!}{16^i \sqrt{2} (2i)!(2i+1)!} x^i,$	$\int_a^b (G(x) + H(x)) dF(x) = \int_a^b G(x) dF(x) + \int_a^b H(x) dF(x),$																																																																																																						
$\left(\frac{\arcsin x}{x}\right)^2$	$= \sum_{i=0}^{\infty} \frac{4^i i!^2}{(i+1)(2i+1)!} x^{2i}.$	$\int_a^b G(x) d(F(x) + H(x)) = \int_a^b G(x) dF(x) + \int_a^b G(x) dH(x),$																																																																																																						
		$\int_a^b c \cdot G(x) dF(x) = \int_a^b G(x) d(c \cdot F(x)) = c \int_a^b G(x) dF(x),$																																																																																																						
		$\int_a^b G(x) dF(x) = G(b)F(b) - G(a)F(a) - \int_a^b F(x) dG(x).$																																																																																																						
		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		<table><tr><td>00</td><td>47</td><td>18</td><td>76</td><td>29</td><td>93</td><td>85</td><td>34</td><td>61</td><td>52</td></tr><tr><td>86</td><td>11</td><td>57</td><td>28</td><td>70</td><td>39</td><td>94</td><td>45</td><td>02</td><td>63</td></tr><tr><td>95</td><td>80</td><td>22</td><td>67</td><td>38</td><td>71</td><td>49</td><td>56</td><td>13</td><td>04</td></tr><tr><td>59</td><td>96</td><td>81</td><td>33</td><td>07</td><td>48</td><td>72</td><td>60</td><td>24</td><td>15</td></tr><tr><td>73</td><td>69</td><td>90</td><td>82</td><td>44</td><td>17</td><td>58</td><td>01</td><td>35</td><td>26</td></tr><tr><td>68</td><td>74</td><td>09</td><td>91</td><td>83</td><td>55</td><td>27</td><td>12</td><td>46</td><td>30</td></tr><tr><td>37</td><td>08</td><td>75</td><td>19</td><td>92</td><td>84</td><td>66</td><td>23</td><td>50</td><td>41</td></tr><tr><td>14</td><td>25</td><td>36</td><td>40</td><td>51</td><td>62</td><td>03</td><td>77</td><td>88</td><td>99</td></tr><tr><td>21</td><td>32</td><td>43</td><td>54</td><td>65</td><td>06</td><td>10</td><td>89</td><td>97</td><td>78</td></tr><tr><td>42</td><td>53</td><td>64</td><td>05</td><td>16</td><td>20</td><td>31</td><td>98</td><td>79</td><td>87</td></tr></table>		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	Fibonacci Numbers
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																																																																																															
If we have equations:		1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...																																																																																																						
$a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1$		Definitions:																																																																																																						
$a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2$		$F_i = F_{i-1} + F_{i-2}, \quad F_0 = F_1 = 1,$																																																																																																						
$\vdots$		$F_{-i} = (-1)^{i-1} F_i,$																																																																																																						
$\vdots$		$F_i = \frac{1}{\sqrt{5}} \left( \phi^i - \hat{\phi}^i \right),$																																																																																																						
$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$		Cassini's identity: for $i > 0$ :																																																																																																						
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		$F_{i+1}F_{i-1} - F_i^2 = (-1)^i.$																																																																																																						
$x_i = \frac{\det A_i}{\det A}.$		Additive rule:																																																																																																						
		$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n,$																																																																																																						
		$F_{2n} = F_n F_{n+1} + F_{n-1} F_n.$																																																																																																						
		Calculation by matrices:																																																																																																						
		$\begin{pmatrix} F_{n-2} & F_{n-1} \\ F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n.$																																																																																																						
Improvement makes strait roads, but the crooked roads without Improvement, are roads of Genius. – William Blake (The Marriage of Heaven and Hell)		The Fibonacci number system: Every integer $n$ has a unique representation $n = F_{k_1} + F_{k_2} + \cdots + F_{k_m},$ where $k_i \geq k_{i+1} + 2$ for all $i,$ $1 \leq i < m$ and $k_m \geq 2.$																																																																																																						





