

## UPLB Eliens ICPC Notebook (C++)

## Contents

## 1 Data Structures

1.1	Disjoint Set Union	1
1.2	Minimum Queue	1
1.3	Range Add Point Query	2
1.4	Range Add Range Query	2
1.5	Segment Tree	3
1.6	Sparse Table	4

## 2 Dynamic Programming

2.1	Divide And Conquer	4
2.2	Edit Distance	4
2.3	Knapsack	5
2.4	Knuth Optimization	5
2.5	Longest Common Subsequence	5
2.6	Longest Increasing Subsequence	5
2.7	Subset Sum	6

## 3 Geometry

3.1	Circle Line Intersection	6
3.2	Convex Hull	6
3.3	Line Sweep	7
3.4	Nearest Points	8

## 4 Graph Theory

4.1	Articulation Point	8
4.2	Bellman Ford	8
4.3	Bridge	9
4.4	Dijkstra	9
4.5	Find Cycle	10
4.6	Floyd Warshall	10
4.7	Hierholzer	10
4.8	Is Bipartite	11
4.9	Is Cyclic	11
4.10	Kahn	11
4.11	Kruskal Mst	12
4.12	Lowest Common Ancestor	12
4.13	Maximum Bipartite Matching	13
4.14	Max Flow	13
4.15	Prim Mst	14
4.16	Strongly Connected Component	14
4.17	Topological Sort	14

## 5 Miscellaneous

5.1	Gauss	15
5.2	Ternary Search	15

## 6 Number Theory

6.1	Extended Euclidean	15
6.2	Find All Solutions	16
6.3	Linear Sieve	16
6.4	Miller Rabin	16
6.5	Modulo Inverse	17
6.6	Pollard Rho Brent	17
6.7	Range Sieve	17
6.8	Segmented Sieve	18
6.9	Tonelli Shanks	18

## 7 Strings

7.1	Hashing	19
7.2	Knuth Morris Pratt	19
7.3	Rabin Karp	19
7.4	Suffix Array	19
7.5	Z Function	20

## 1 Data Structures

## 1.1 Disjoint Set Union

```

1 struct DSU {
2     vector<int> parent, size;
3     DSU(int n) {
4         parent.resize(n);
5         size.resize(n);
6         for (int i = 0; i < n; i++) make_set(i);
7     }
8     void make_set(int v) {
9         parent[v] = v;
10        size[v] = 1;
11    }
12    bool is_same(int a, int b) { return find_set(a) ==
13        find_set(b); }
14    int find_set(int v) { return v == parent[v] ? v :
15        parent[v] = find_set(parent[v]); }
16    void union_sets(int a, int b) {
17        a = find_set(a);
18        b = find_set(b);
19        if (a != b) {
20            if (size[a] < size[b]) swap(a, b);
21            parent[b] = a;
22            size[a] += size[b];
23        }
24    }
25 };

```

## 1.2 Minimum Queue

```

11 get_minimum(stack<pair<ll, ll>> &s1, stack<pair<ll,
12 ll>> &s2) {
13     if (s1.empty() || s2.empty()) {
14         return s1.empty() ? s2.top().second : s1.top().
15             second;
16     } else {
17         return min(s1.top().second, s2.top().second);
18     }
19 }
20 void add_element(ll new_element, stack<pair<ll, ll>> &s1
21 ) {
22     ll minimum = s1.empty() ? new_element : min(
23         new_element, s1.top().second);
24     s1.push({new_element, minimum});
25 }
26 ll remove_element(stack<pair<ll, ll>> &s1, stack<pair<ll
27 , ll>> &s2) {

```

```

if (s2.empty()) {
    while (!s1.empty()) {
        ll element = s1.top().first;
        s1.pop();
        ll minimum = s2.empty() ? element : min(element,
            s2.top().second);
        s2.push({element, minimum});
    }
}
ll removed_element = s2.top().first;
s2.pop();
return removed_element;
}

```

### 1.3 Range Add Point Query

```

template<typename T, typename InType = T>
class SegTreeNode {
public:
    const T IDN = 0, DEF = 0;
    int i, j;
    T val;
    SegTreeNode<T, InType>* lc, * rc;
    SegTreeNode(int i, int j) : i(i), j(j) {
        if (j - i == 1) {
            val = DEF;
            lc = rc = nullptr;
            return;
        }
        val = 0;
        int k = (i + j) / 2;
        lc = new SegTreeNode<T, InType>(i, k);
        rc = new SegTreeNode<T, InType>(k, j);
    }
    SegTreeNode(const vector<InType>& a, int i, int j) : i
        (i), j(j) {
        if (j - i == 1) {
            val = (T) a[i];
            lc = rc = nullptr;
            return;
        }
        val = 0;
        int k = (i + j) / 2;
        lc = new SegTreeNode<T, InType>(a, i, k);
        rc = new SegTreeNode<T, InType>(a, k, j);
    }
    void range_add(int l, int r, T x) {
        if (r <= i || j <= l) return;
        if (l <= i && j <= r) {
            val += x;

```

```

        return;
    }
    lc->range_add(l, r, x);
    rc->range_add(l, r, x);
}
T point_query(int k) {
    if (k < i || j <= k) return IDN;
    if (j - i == 1) return val;
    return val + lc->point_query(k) + rc->point_query(k)
        ;
}
};
template<typename T, typename InType = T>
class SegTree {
public:
    SegTreeNode<T, InType> root;
    SegTree(int n) : root(0, n) {}
    SegTree(const vector<InType>& a) : root(a, 0, a.size()
        ) {}
    void range_add(int l, int r, T x) { root.range_add(l,
        r, x); }
    T point_query(int k) { return root.point_query(k); }
};

```

### 1.4 Range Add Range Query

```

template<typename T, typename InType = T>
class SegTreeNode {
public:
    const T IDN = 0, DEF = 0;
    int i, j;
    T val, to_add = 0;
    SegTreeNode<T, InType>* lc, * rc;
    SegTreeNode(int i, int j) : i(i), j(j) {
        if (j - i == 1) {
            lc = rc = nullptr;
            val = DEF;
            return;
        }
        int k = (i + j) / 2;
        lc = new SegTreeNode<T, InType>(i, k);
        rc = new SegTreeNode<T, InType>(k, j);
        val = operation(lc->val, rc->val);
    }
    SegTreeNode(const vector<InType>& a, int i, int j) : i
        (i), j(j) {
        if (j - i == 1) {
            lc = rc = nullptr;
            val = a[i];
            return;

```

```

    }
    int k = (i + j) / 2;
    lc = new SegTreeNode<T, InType>(a, i, k);
    rc = new SegTreeNode<T, InType>(a, k, j);
    val = operation(lc->val, rc->val);
}
void propagate() {
    if (to_add == 0) return;
    val += to_add;
    if (j - i > 1) {
        lc->to_add += to_add;
        rc->to_add += to_add;
    }
    to_add = 0;
}
void range_add(int l, int r, T delta) {
    propagate();
    if (r <= i || j <= l) return;
    if (l <= i && j <= r) {
        to_add += delta;
        propagate();
    } else {
        lc->range_add(l, r, delta);
        rc->range_add(l, r, delta);
        val = operation(lc->val, rc->val);
    }
}
T range_query(int l, int r) { // [l, r)
    propagate();
    if (l <= i && j <= r) return val;
    if (j <= l || r <= i) return IDN;
    return operation(lc->range_query(l, r), rc->
        range_query(l, r));
}
T operation(T x, T y) {}
};

template<typename T, typename InType = T>
class SegTree {
public:
    SegTreeNode<T, InType> root;
    SegTree(int n) : root(0, n) {}
    SegTree(const vector<InType>& a) : root(a, 0, a.size()
        ()) {}
    void range_add(int l, int r, T delta) { root.
        range_add(l, r, delta); }
    T range_query(int l, int r) { return root.
        range_query(l, r); }
};

```

## 1.5 Segment Tree

```

template<typename T, typename InType = T>
class SegTreeNode {
public:
    const T IDN = 0, DEF = 0;
    int i, j;
    T val;
    SegTreeNode<T, InType>* lc, * rc;
    SegTreeNode(int i, int j) : i(i), j(j) {
        if (j - i == 1) {
            lc = rc = nullptr;
            val = DEF;
            return;
        }
        int k = (i + j) / 2;
        lc = new SegTreeNode<T, InType>(i, k);
        rc = new SegTreeNode<T, InType>(k, j);
        val = op(lc->val, rc->val);
    }
    SegTreeNode(const vector<InType>& a, int i, int j) : i
        (i), j(j) {
        if (j - i == 1) {
            lc = rc = nullptr;
            val = a[i];
            return;
        }
        int k = (i + j) / 2;
        lc = new SegTreeNode<T, InType>(a, i, k);
        rc = new SegTreeNode<T, InType>(a, k, j);
        val = op(lc->val, rc->val);
    }
    void set(int k, T x) { // update a[k] := x
        if (k < i || j <= k) return;
        if (j - i == 1) {
            val = x;
            return;
        }
        lc->set(k, x);
        rc->set(k, x);
        val = op(lc->val, rc->val);
    }
    T range_query(int l, int r) { // [l, r)
        if (l <= i && j <= r) return val;
        if (j <= l || r <= i) return IDN;
        return op(lc->range_query(l, r), rc->range_query(l,
            r));
    }
    T op(T x, T y) {}
};

```

```

};
template<typename T, typename InType = T>
class SegTree {
public:
    SegTreeNode<T, InType> root;
    SegTree(int n) : root(0, n) {}
    SegTree(const vector<InType>& a) : root(a, 0, a.size()
        ) {}
    void set(int k, T x) { root.set(k, x); }
    T range_query(int l, int r) { return root.range_query(
        l, r); }
};

```

## 1.6 Sparse Table

```

ll log2_floor(ll i) {
    return i ? __builtin_clzll(1) - __builtin_clzll(i) :
        -1;
}
vector<vector<ll>> build_sum(ll N, ll K, vector<ll> &
    array) {
    vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
    for (ll i = 0; i < N; i++) st[0][i] = array[i];
    for (ll i = 1; i <= K; i++)
        for (ll j = 0; j + (1 << i) <= N; j++)
            st[i][j] = st[i - 1][j] + st[i - 1][j + (1 << (i -
                1))];
    return st;
}
ll sum_query(ll L, ll R, ll K, vector<vector<ll>> &st) {
    ll sum = 0;
    for (ll i = K; i >= 0; i--) {
        if ((1 << i) <= R - L + 1) {
            sum += st[i][L];
            L += 1 << i;
        }
    }
    return sum;
}
vector<vector<ll>> build_min(ll N, ll K, vector<ll> &
    array) {
    vector<vector<ll>> st(K + 1, vector<ll>(N + 1));
    for (ll i = 0; i < N; i++) st[0][i] = array[i];
    for (ll i = 1; i <= K; i++)
        for (ll j = 0; j + (1 << i) <= N; j++)
            st[i][j] = min(st[i - 1][j], st[i - 1][j + (1 << (
                i - 1))]);
    return st;
}
ll min_query(ll L, ll R, vector<vector<ll>> &st) {

```

```

    ll i = log2_floor(R - L + 1);
    return min(st[i][L], st[i][R - (1 << i) + 1]);
}

```

## 2 Dynamic Programming

### 2.1 Divide And Conquer

```

ll m, n;
vector<ll> dp_before(n), dp_cur(n);
ll C(ll i, ll j);
void compute(ll l, ll r, ll optl, ll optr) {
    if (l > r) {
        return;
    }
    ll mid = (l + r) >> 1;
    pair<ll, ll> best = {LLONG_MAX, -1};
    for (ll k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k,
            mid), k});
    }
    dp_cur[mid] = best.first;
    ll opt = best.second;
    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}
ll solve() {
    for (ll i = 0; i < n; i++) {
        dp_before[i] = C(0, i);
    }
    for (ll i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }
    return dp_before[n - 1];
}

```

### 2.2 Edit Distance

```

ll edit_distance(string x, string y, ll n, ll m) {
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, INF))
        ;
    dp[0][0] = 0;
    for (int i = 1; i <= n; i++) {
        dp[i][0] = i;
    }
    for (int j = 1; j <= m; j++) {
        dp[0][j] = j;
    }
}

```

```

}
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        dp[i][j] = min({dp[i - 1][j] + 1, dp[i][j - 1] +
            1, dp[i - 1][j - 1] + (x[i - 1] != y[j - 1])});
    }
}
return dp[n][m];
}

```

## 2.3 Knapsack

```

ll knapsack(ll W, vector<ll> &wt, vector<ll> &val, ll n)
{
    vector<ll> dp(W + 1, 0);
    for (ll i = 1; i <= n; i++) {
        for (ll w = W; w >= 0; w--) {
            if (wt[i - 1] <= w) {
                dp[w] = max(dp[w], dp[w - wt[i - 1]] + val[i - 1]);
            }
        }
    }
    return dp[W];
}

```

## 2.4 Knuth Optimization

```

ll solve() {
    ll N;
    // read N and input
    vector<vector<ll>> dp(N, vector<ll>(N)), opt(N, vector<ll>(N));
    auto C = [&](ll i, ll j) {
        // Implement cost function C.
    };
    for (ll i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the problem
    }
    for (ll i = N - 2; i >= 0; i--) {
        for (ll j = i + 1; j < N; j++) {
            ll mn = ll_MAX, cost = C(i, j);
            for (ll k = opt[i][j - 1]; k <= min(j - 1, opt[i + 1][j]); k++) {
                if (mn >= dp[i][k] + dp[k + 1][j] + cost) {
                    opt[i][j] = k;

```

```

                mn = dp[i][k] + dp[k + 1][j] + cost;
            }
        }
        dp[i][j] = mn;
    }
    cout << dp[0][N - 1] << '\n';
}

```

## 2.5 Longest Common Subsequence

```

ll LCS(string x, string y, ll n, ll m) {
    vector<vector<ll>> dp(n + 1, vector<ll>(m + 1));
    for (ll i = 0; i <= n; i++) {
        for (ll j = 0; j <= m; j++) {
            if (i == 0 || j == 0) {
                dp[i][j] = 0;
            } else if (x[i - 1] == y[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
    ll index = dp[n][m];
    vector<char> lcs(index + 1);
    lcs[index] = '\0';
    ll i = n, j = m;
    while (i > 0 && j > 0) {
        if (x[i - 1] == y[j - 1]) {
            lcs[index - 1] = x[i - 1];
            i--;
            j--;
            index--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }
    return dp[n][m];
}

```

## 2.6 Longest Increasing Subsequence

```

ll get_ceil_idx(vector<ll> &a, vector<ll> &T, ll l, ll r
, ll x) {
    while (r - l > 1) {

```

```

    ll m = 1 + (r - 1) / 2;
    if (a[T[m]] >= x) {
        r = m;
    } else {
        l = m;
    }
}
return r;
}
ll LIS(ll n, vector<ll> &a) {
    ll len = 1;
    vector<ll> T(n, 0), R(n, -1);
    T[0] = 0;
    for (ll i = 1; i < n; i++) {
        if (a[i] < a[T[0]]) {
            T[0] = i;
        } else if (a[i] > a[T[len - 1]]) {
            R[i] = T[len - 1];
            T[len++] = i;
        } else {
            ll pos = get_ceil_idx(a, T, -1, len - 1, a[i]);
            R[i] = T[pos - 1];
            T[pos] = i;
        }
    }
    return len;
}

```

## 2.7 Subset Sum

```

bool subset_sum(ll n, vector<ll> &arr, ll sum) {
    vector<vector<ll>> dp(n + 1, vector<ll>(sum + 1, false));
    dp[0][0] = true;
    for (ll i = 1; i <= n; i++) {
        for (ll j = 0; j <= sum; j++) {
            dp[i][j] = dp[i - 1][j];
            if (j >= arr[i]) {
                dp[i][j] |= dp[i - 1][j - arr[i]];
            }
        }
    }
    return dp[n][sum];
}

```

## 3 Geometry

### 3.1 Circle Line Intersection

```

double r, a, b, c; // given as input
double x0 = -a * c / (a * a + b * b);
double y0 = -b * c / (a * a + b * b);
if (c * c > r * r * (a * a + b * b) + EPS) {
    puts ("no points");
} else if (abs (c * c - r * r * (a * a + b * b)) < EPS) {
    {
        puts ("1 point");
        cout << x0 << ' ' << y0 << '\n';
    }
} else {
    double d = r * r - c * c / (a * a + b * b);
    double mult = sqrt (d / (a * a + b * b));
    double ax, ay, bx, by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
}
}

```

### 3.2 Convex Hull

```

struct pt {
    double x, y;
};
ll orientation(pt a, pt b, pt c) {
    double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x
        * (a.y - b.y);
    if (v < 0) {
        return -1;
    } else if (v > 0) {
        return +1;
    }
    return 0;
}
bool cw(pt a, pt b, pt c, bool include_collinear) {
    ll o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) {
    return orientation(a, b, c) == 0;
}
void convex_hull(vector<pt>& a, bool include_collinear = false) {
    pt p0 = *min_element(a.begin(), a.end(), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
}

```

```

sort(a.begin(), a.end(), [&p0](const pt& a, const pt&
    b) {
    ll o = orientation(p0, a, b);
    if (o == 0) {
        return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y)
            * (p0.y - a.y)
            < (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y)
            * (p0.y - b.y);
    }
    return o < 0;
});
if (include_collinear) {
    ll i = (ll) a.size() - 1;
    while (i >= 0 && collinear(p0, a[i], a.back())) i--;
    reverse(a.begin() + i + 1, a.end());
}
vector<pt> st;
for (ll i = 0; i < (ll) a.size(); i++) {
    while (st.size() > 1 && !cw(st[st.size() - 2], st.
        back(), a[i], include_collinear)) {
        st.pop_back();
    }
    st.push_back(a[i]);
}
a = st;
}

```

### 3.3 Line Sweep

```

const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg {
    pt p, q;
    ll id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS) {
            return p.y;
        }
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};
bool intersectld(double l1, double r1, double l2, double
    r2) {
    if (l1 > r1) {
        swap(l1, r1);
    }
    if (l2 > r2) {
        swap(l2, r2);
    }

```

```

    }
    return max(l1, l2) <= min(r1, r2) + EPS;
}
ll vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (
        c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b) {
    return intersectld(a.p.x, a.q.x, b.p.x, b.q.x) &&
        intersectld(a.p.y, a.q.y, b.p.y, b.q.y) &&
        vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
        vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg& a, const seg& b) {
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x))
        ;
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x;
    ll tp, id;
    event() {}
    event(double x, ll tp, ll id) : x(x), tp(tp), id(id)
        {}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS) {
            return x < e.x;
        }
        return tp > e.tp;
    }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<ll, ll> solve(const vector<seg>& a) {
    ll n = (ll) a.size();
    vector<event> e;
    for (ll i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());
    s.clear();
    where.resize(a.size());

```

```

for (size_t i = 0; i < e.size(); ++i) {
    ll id = e[i].id;
    if (e[i].tp == +1) {
        set<seg>::iterator nxt = s.lower_bound(a[id]), prv
            = prev(nxt);
        if (nxt != s.end() && intersect(*nxt, a[id])) {
            return make_pair(nxt->id, id);
        }
        if (prv != s.end() && intersect(*prv, a[id])) {
            return make_pair(prv->id, id);
        }
        where[id] = s.insert(nxt, a[id]);
    } else {
        set<seg>::iterator nxt = next(where[id]), prv =
            prev(where[id]);
        if (nxt != s.end() && prv != s.end() && intersect
            (*nxt, *prv)) {
            return make_pair(prv->id, nxt->id);
        }
        s.erase(where[id]);
    }
}
return make_pair(-1, -1);
}

```

### 3.4 Nearest Points

```

struct pt {
    ll x, y, id;
};
struct cmp_x {
    bool operator()(const pt & a, const pt & b) const {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
};
struct cmp_y {
    bool operator()(const pt & a, const pt & b) const {
        return a.y < b.y;
    }
};
ll n;
vector<pt> a;
double mindist;
pair<ll, ll> best_pair;
void upd_ans(const pt & a, const pt & b) {
    double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a.y -
        b.y) * (a.y - b.y));
    if (dist < mindist) {
        mindist = dist;
        best_pair = {a.id, b.id};
    }
}

```

```

}
vector<pt> t;
void rec(ll l, ll r) {
    if (r - l <= 3) {
        for (ll i = l; i < r; ++i) {
            for (ll j = i + 1; j < r; ++j) {
                upd_ans(a[i], a[j]);
            }
        }
        sort(a.begin() + l, a.begin() + r, cmp_y());
        return;
    }
    ll m = (l + r) >> 1, midx = a[m].x;
    rec(l, m);
    rec(m, r);
    merge(a.begin() + l, a.begin() + m, a.begin() + m, a.
        begin() + r, t.begin(), cmp_y());
    copy(t.begin(), t.begin() + r - l, a.begin() + l);
    ll tsz = 0;
    for (ll i = l; i < r; ++i) {
        if (abs(a[i].x - midx) < mindist) {
            for (ll j = tsz - 1; j >= 0 && a[i].y - t[j].y <
                mindist; --j) {
                upd_ans(a[i], t[j]);
            }
            t[tsz++] = a[i];
        }
    }
    t.resize(n);
    sort(a.begin(), a.end(), cmp_x());
    mindist = 1E20;
    rec(0, n);
}

```

## 4 Graph Theory

### 4.1 Articulation Point

```

void APUtil(vector<vector<ll>> &adj, ll u, vector<bool>
    &visited,
    vector<ll> &disc, vector<ll> &low, ll &time, ll parent,
    vector<bool> &isAP) {
    ll children = 0;
    visited[u] = true;
    disc[u] = low[u] = ++time;
    for (auto v : adj[u]) {
        if (!visited[v]) {
            children++;

```



```

    APUtil(adj, v, visited, disc, low, time, u, isAP);
    low[u] = min(low[u], low[v]);
    if (parent != -1 && low[v] >= disc[u]) {
        isAP[u] = true;
    }
} else if (v != parent) {
    low[u] = min(low[u], disc[v]);
}
}
if (parent == -1 && children > 1) {
    isAP[u] = true;
}
}
void AP(vector<vector<ll>> &adj, ll n) {
    vector<ll> disc(n), low(n);
    vector<bool> visited(n), isAP(n);
    ll time = 0, par = -1;
    for (ll u = 0; u < n; u++) {
        if (!visited[u]) {
            APUtil(adj, u, visited, disc, low, time, par, isAP);
        }
    }
    for (ll u = 0; u < n; u++) {
        if (isAP[u]) {
            cout << u << " ";
        }
    }
}

```

## 4.2 Bellman Ford

```

void bellman_ford(vector<vector<ll>> &edges, ll n, ll m,
    ll src, vector<ll> &dis) {
    for (ll i = 0; i < n; i++) {
        dis[i] = INF;
    }
    for (ll i = 0; i < n - 1; i++) {
        for (ll j = 0; j < m; j++) {
            ll u = edges[j][0], v = edges[j][1], w = edges[j][2];
            if (dis[u] < INF) {
                dis[v] = min(dis[v], dis[u] + w);
            }
        }
    }
    for (ll i = 0; i < m; i++) {
        ll u = edges[i][0], v = edges[i][1], w = edges[i][2];
        if (dis[u] < INF && dis[u] + w < dis[v]) {

```

```

        cout << "The graph contains a negative cycle." <<
            '\n';
    }
}

```

## 4.3 Bridge

```

void bridge_util(vector<vector<ll>> &adj, ll u, vector<
    bool> &visited, vector<ll> &disc, vector<ll> &low,
    vector<ll> &parent) {
    static ll time = 0;
    visited[u] = true;
    disc[u] = low[u] = ++time;
    list<ll>::iterator i;
    for (auto v : adj[u]) {
        if (!visited[v]) {
            parent[v] = u;
            bridge_util(adj, v, visited, disc, low, parent);
            low[u] = min(low[u], low[v]);
            if (low[v] > disc[u]) {
                cout << u << ' ' << v << '\n';
            }
        } else if (v != parent[u]) {
            low[u] = min(low[u], disc[v]);
        }
    }
}
void bridge(vector<vector<ll>> &adj, ll n) {
    vector<bool> visited(n, false);
    vector<ll> disc(n), low(n), parent(n, -1);
    for (ll i = 0; i < n; i++) {
        if (!visited[i]) {
            bridge_util(adj, i, visited, disc, low, parent);
        }
    }
}

```

## 4.4 Dijkstra

```

void dijkstra(ll n, vector<vector<pair<ll, ll>>> &adj,
    vector<ll> &dis) {
    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
        greater<pair<ll, ll>>> pq;
    for (int i = 0; i < n; i++) {
        dis[i] = INF;
    }
    dis[0] = 0;

```

```

pq.push({0, 0});
while (!pq.empty()) {
    auto p = pq.top();
    pq.pop();
    ll u = p.second;
    if (dis[u] != p.first) {
        continue;
    }
    for (auto x : adj[u]) {
        ll v = x.first, w = x.second;
        if (dis[v] > dis[u] + w) {
            dis[v] = dis[u] + w;
            pq.push({dis[v], v});
        }
    }
}
}

```

## 4.5 Find Cycle

```

bool dfs(ll v) {
    color[v] = 1;
    for (ll u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u)) {
                return true;
            }
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;
    for (ll v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v)) {
            break;
        }
    }
    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<ll> cycle;

```

```

cycle.push_back(cycle_start);
for (ll v = cycle_end; v != cycle_start; v = parent[v]) {
    cycle.push_back(v);
}
cycle.push_back(cycle_start);
reverse(cycle.begin(), cycle.end());
cout << "Cycle found: ";
for (ll v : cycle) {
    cout << v << ' ';
}
cout << '\n';
}
}

```

## 4.6 Floyd Warshall

```

void floyd_warshall(vector<vector<ll>> &dis, ll n) {
    for (ll i = 0; i < n; i++) {
        for (ll j = 0; j < n; j++) {
            dis[i][j] = (i == j ? 0 : INF);
        }
    }
    for (ll k = 0; k < n; k++) {
        for (ll i = 0; i < n; i++) {
            for (ll j = 0; j < n; j++) {
                if (dis[i][k] < INF && dis[k][j] < INF) {
                    dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
                }
            }
        }
    }
    for (ll i = 0; i < n; i++) {
        for (ll j = 0; j < n; j++) {
            for (ll k = 0; k < n; k++) {
                if (dis[k][k] < 0 && dis[i][k] < INF && dis[k][j] < INF) {
                    dis[i][j] = -INF;
                }
            }
        }
    }
}

```

## 4.7 Hierholzer

```

void print_circuit(vector<vector<ll>> &adj) {

```

```

map<ll, ll> edge_count;
for (ll i = 0; i < adj.size(); i++) {
    edge_count[i] = adj[i].size();
}
if (!adj.size()) {
    return;
}
stack<ll> curr_path;
vector<ll> circuit;
curr_path.push(0);
ll curr_v = 0;
while (!curr_path.empty()) {
    if (edge_count[curr_v]) {
        curr_path.push(curr_v);
        ll next_v = adj[curr_v].back();
        edge_count[curr_v]--;
        adj[curr_v].pop_back();
        curr_v = next_v;
    } else {
        circuit.push_back(curr_v);
        curr_v = curr_path.top();
        curr_path.pop();
    }
}
for (ll i = circuit.size() - 1; i >= 0; i--) {
    cout << circuit[i] << ' ';
}
}

```

## 4.8 Is Bipartite

```

bool is_bipartite(vector<ll> &col, vector<vector<ll>> &
adj, ll n) {
    queue<pair<ll, ll>> q;
    for (ll i = 0; i < n; i++) {
        if (col[i] == -1) {
            q.push({i, 0});
            col[i] = 0;
            while (!q.empty()) {
                pair<ll, ll> p = q.front();
                q.pop();
                ll v = p.first, c = p.second;
                for (ll j : adj[v]) {
                    if (col[j] == c) {
                        return false;
                    }
                    if (col[j] == -1) {
                        col[j] = (c ? 0 : 1);
                        q.push({j, col[j]});
                    }
                }
            }
        }
    }
}

```

```

    }
}
return true;
}

```

## 4.9 Is Cyclic

```

bool is_cyclic_util(int u, vector<vector<int>> &adj,
vector<bool> &vis, vector<bool> &rec) {
    vis[u] = true;
    rec[u] = true;
    for(auto v : adj[u]) {
        if (!vis[v] && is_cyclic_util(v, adj, vis, rec)) {
            return true;
        } else if (rec[v]) {
            return true;
        }
    }
    rec[u] = false;
    return false;
}

bool is_cyclic(int n, vector<vector<int>> &adj) {
    vector<bool> vis(n, false), rec(n, false);
    for (int i = 0; i < n; i++) {
        if (!vis[i] && is_cyclic_util(i, adj, vis, rec)) {
            return true;
        }
    }
    return false;
}

```

## 4.10 Kahn

```

void kahn(vector<vector<ll>> &adj) {
    ll n = adj.size();
    vector<ll> in_degree(n, 0);
    for (ll u = 0; u < n; u++) {
        for (ll v : adj[u]) {
            in_degree[v]++;
        }
    }
    queue<ll> q;
    for (ll i = 0; i < n; i++) {
        if (in_degree[i] == 0) {
            q.push(i);
        }
    }
}

```

```

}
ll cnt = 0;
vector<ll> top_order;
while (!q.empty()) {
    ll u = q.front();
    q.pop();
    top_order.push_back(u);
    for (ll v : adj[u]) {
        if (--in_degree[v] == 0) {
            q.push(v);
        }
    }
    cnt++;
}
if (cnt != n) {
    cout << -1 << '\n';
    return;
}
for (ll i = 0; i < (ll) top_order.size(); i++) {
    cout << top_order[i] << ' ';
}
cout << '\n';
}

```

## 4.11 Kruskal Mst

```

struct Edge {
    ll u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};
ll n;
vector<Edge> edges;
ll cost = 0;
vector<ll> tree_id(n);
vector<Edge> result;
for (ll i = 0; i < n; i++) {
    tree_id[i] = i;
}
sort(edges.begin(), edges.end());
for (Edge e : edges) {
    if (tree_id[e.u] != tree_id[e.v]) {
        cost += e.weight;
        result.push_back(e);
        ll old_id = tree_id[e.u], new_id = tree_id[e.v];
        for (ll i = 0; i < n; i++) {
            if (tree_id[i] == old_id) {
                tree_id[i] = new_id;
            }
        }
    }
}

```

```

}
}
}

```

## 4.12 Lowest Common Ancestor

```

struct LCA {
    vector<ll> height, euler, first, segtree;
    vector<bool> visited;
    ll n;
    LCA(vector<vector<ll>> &adj, ll root = 0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        ll m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }
    void dfs(vector<vector<ll>> &adj, ll node, ll h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }
    void build(ll node, ll b, ll e) {
        if (b == e) {
            segtree[node] = euler[b];
        } else {
            ll mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid + 1, e);
            ll l = segtree[node << 1], r = segtree[node << 1 | 1];
            segtree[node] = (height[l] < height[r]) ? l : r;
        }
    }
    ll query(ll node, ll b, ll e, ll L, ll R) {
        if (b > R || e < L) {
            return -1;
        }
        if (b >= L && e <= R) {

```

```

    return segtree[node];
}
ll mid = (b + e) >> 1;
ll left = query(node << 1, b, mid, L, R);
ll right = query(node << 1 | 1, mid + 1, e, L, R);
if (left == -1) return right;
if (right == -1) return left;
return height[left] < height[right] ? left : right;
}
ll lca(ll u, ll v) {
    ll left = first[u], right = first[v];
    if (left > right) {
        swap(left, right);
    }
    return query(1, 0, euler.size() - 1, left, right);
}
};

```

#### 4.13 Maximum Bipartite Matching

```

bool bpm(ll n, ll m, vector<vector<bool>> &bpGraph, ll u
, vector<bool> &seen, vector<ll> &matchR) {
    for (ll v = 0; v < m; v++) {
        if (bpGraph[u][v] && !seen[v]) {
            seen[v] = true;
            if (matchR[v] < 0 || bpm(n, m, bpGraph, matchR[v],
                seen, matchR)) {
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}
ll maxBPM(ll n, ll m, vector<vector<bool>> &bpGraph) {
    vector<ll> matchR(m, -1);
    ll result = 0;
    for (ll u = 0; u < n; u++) {
        vector<bool> seen(m, false);
        if (bpm(n, m, bpGraph, u, seen, matchR)) {
            result++;
        }
    }
    return result;
}

```

#### 4.14 Max Flow

```

bool bfs(ll n, vector<vector<ll>> &r_graph, ll s, ll t,
    vector<ll> &parent) {
    vector<bool> visited(n, false);
    queue<ll> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty()) {
        ll u = q.front();
        q.pop();
        for (ll v = 0; v < n; v++) {
            if (!visited[v] && r_graph[u][v] > 0) {
                if (v == t) {
                    parent[v] = u;
                    return true;
                }
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return false;
}
ll fordFulkerson(ll n, vector<vector<ll>> graph, ll s,
    ll t) {
    ll u, v;
    vector<vector<ll>> r_graph;
    for (u = 0; u < n; u++) {
        for (v = 0; v < n; v++) {
            r_graph[u][v] = graph[u][v];
        }
    }
    vector<ll> parent;
    ll max_flow = 0;
    while (bfs(n, r_graph, s, t, parent)) {
        ll path_flow = INF;
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            path_flow = min(path_flow, r_graph[u][v]);
        }
        for (v = t; v != s; v = parent[v]) {
            u = parent[v];
            r_graph[u][v] -= path_flow;
            r_graph[v][u] += path_flow;
        }
        max_flow += path_flow;
    }
    return max_flow;
}

```

## 4.15 Prim Mst

```
vector<ll> prim_mst(ll n, vector<vector<pair<ll, ll>>> &
    adj) {
    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
        greater<pair<ll, ll>>> pq;
    ll src = 0;
    vector<ll> key(n, INF), parent(n, -1);
    vector<bool> in_mst(n, false);
    pq.push(make_pair(0, src));
    key[src] = 0;
    while (!pq.empty()) {
        ll u = pq.top().second;
        pq.pop();
        if (in_mst[u]) {
            continue;
        }
        in_mst[u] = true;
        for (auto p : adj[u]) {
            ll v = p.first, w = p.second;
            if (in_mst[v] == false && w < key[v]) {
                key[v] = w;
                pq.push(make_pair(key[v], v));
                parent[v] = u;
            }
        }
    }
    return parent;
}
```

## 4.16 Strongly Connected Component

```
void dfs(ll u, vector<vector<ll>> &adj, vector<bool> &
    visited) {
    visited[u] = true;
    cout << u + 1 << ' ';
    for (ll v : adj[u]) {
        if (!visited[v]) {
            dfs(v, adj, visited);
        }
    }
}

vector<vector<ll>> get_transpose(ll n, vector<vector<ll>>
    >> &adj) {
    vector<vector<ll>> res(n);
    for (ll u = 0; u < n; u++) {
        for (ll v : adj[u]) {
            res[v].push_back(u);
        }
    }
}
```

```
    }
}
return res;
}

void fill_order(ll u, vector<vector<ll>> &adj, vector<
    bool> &visited, stack<ll> &stk) {
    visited[u] = true;
    for (auto v : adj[u]) {
        if (!visited[v]) {
            fill_order(v, adj, visited, stk);
        }
    }
    stk.push(u);
}

void get_scc(ll n, vector<vector<ll>> &adj) {
    stack<ll> stk;
    vector<bool> visited(n, false);
    for (ll i = 0; i < n; i++) {
        if (!visited[i]) {
            fill_order(i, adj, visited, stk);
        }
    }
    vector<vector<ll>> transpose = get_transpose(n, adj);
    for (ll i = 0; i < n; i++) {
        visited[i] = false;
    }
    while (!stk.empty()) {
        ll u = stk.top();
        stk.pop();
        if (!visited[u]) {
            dfs(u, transpose, visited);
            cout << '\n';
        }
    }
}
```

## 4.17 Topological Sort

```
void dfs(ll v) {
    visited[v] = true;
    for (ll u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
        }
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
}
```

```

for (ll i = 0; i < n; ++i) {
    if (!visited[i]) {
        dfs(i);
    }
}
reverse(ans.begin(), ans.end());
}

```

## 5 Miscellaneous

### 5.1 Gauss

```

const double EPS = 1e-9;
const ll INF = 2;
ll gauss(vector<vector<double>> a, vector<double> &ans)
{
    ll n = (ll) a.size(), m = (ll) a[0].size() - 1;
    vector<ll> where(m, -1);
    for (ll col = 0, row = 0; col < m && row < n; ++col) {
        ll sel = row;
        for (ll i = row; i < n; ++i) {
            if (abs(a[i][col]) > abs(a[sel][col])) {
                sel = i;
            }
        }
        if (abs(a[sel][col]) < EPS) {
            continue;
        }
        for (ll i = col; i <= m; ++i) {
            swap(a[sel][i], a[row][i]);
        }
        where[col] = row;
        for (ll i = 0; i < n; ++i) {
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (ll j = col; j <= m; ++j) {
                    a[i][j] -= a[row][j] * c;
                }
            }
        }
        ++row;
    }
    ans.assign(m, 0);
    for (ll i = 0; i < m; ++i) {
        if (where[i] != -1) {
            ans[i] = a[where[i]][m] / a[where[i]][i];
        }
    }
    for (ll i = 0; i < n; ++i) {

```

```

        double sum = 0;
        for (ll j = 0; j < m; ++j) {
            sum += ans[j] * a[i][j];
        }
        if (abs(sum - a[i][m]) > EPS) {
            return 0;
        }
    }
    for (ll i = 0; i < m; ++i) {
        if (where[i] == -1) {
            return INF;
        }
    }
    return 1;
}

```

### 5.2 Ternary Search

```

double ternary_search(double l, double r) {
    double eps = 1e-9;
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2) {
            l = m1;
        } else {
            r = m2;
        }
    }
    return f(l);
}

```

## 6 Number Theory

### 6.1 Extended Euclidean

```

ll gcd_extended(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1, g = gcd_extended(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}

```

}

## 6.2 Find All Solutions

```

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0,
    ll &g) {
    g = gcd_extended(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) {
        x0 = -x0;
    }
    if (b < 0) {
        y0 = -y0;
    }
    return true;
}

void shift_solution(ll & x, ll & y, ll a, ll b, ll cnt)
{
    x += cnt * b;
    y -= cnt * a;
}

ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx,
    ll miny, ll maxy) {
    ll x, y, g;
    if (!find_any_solution(a, b, c, x, y, g)) {
        return 0;
    }
    a /= g;
    b /= g;
    ll sign_a = a > 0 ? +1 : -1;
    ll sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx) {
        shift_solution(x, y, a, b, sign_b);
    }
    if (x > maxx) {
        return 0;
    }
    ll lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx) {
        shift_solution(x, y, a, b, -sign_b);
    }
    ll rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny) {

```

```

        shift_solution(x, y, a, b, -sign_a);
    }
    if (y > maxy) {
        return 0;
    }
    ll lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy) {
        shift_solution(x, y, a, b, sign_a);
    }
    ll rx2 = x;
    if (lx2 > rx2) {
        swap(lx2, rx2);
    }
    ll lx = max(lx1, lx2), rx = min(rx1, rx2);
    if (lx > rx) {
        return 0;
    }
    return (rx - lx) / abs(b) + 1;
}

```

## 6.3 Linear Sieve

```

void linear_sieve(ll N, vector<ll> &lowest_prime, vector
    <ll> &prime) {
    for (ll i = 2; i <= N; i++) {
        if (lowest_prime[i] == 0) {
            lowest_prime[i] = i;
            prime.push_back(i);
        }
        for (ll j = 0; i * prime[j] <= N; j++) {
            lowest_prime[i * prime[j]] = prime[j];
            if (prime[j] == lowest_prime[i]) {
                break;
            }
        }
    }
}

```

## 6.4 Miller Rabin

```

bool check_composite(u64 n, u64 a, u64 d, ll s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1) {
        return false;
    }
    for (ll r = 1; r < s; r++) {
        x = (u128) x * x % n;

```



```

    if (x == n - 1) {
        return false;
    }
}
return true;
}
bool miller_rabin(u64 n) {
    if (n < 2) {
        return false;
    }
    ll r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }
    for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a) {
            return true;
        }
        if (check_composite(n, a, d, r)) {
            return false;
        }
    }
    return true;
}

```

## 6.5 Modulo Inverse

```

ll mod_inv(ll a, ll m) {
    if (m == 1) {
        return 0;
    }
    ll m0 = m, x = 1, y = 0;
    while (a > 1) {
        ll q = a / m, t = m;
        m = a % m;
        a = t;
        t = y;
        y = x - q * y;
        x = t;
    }
    if (x < 0) {
        x += m0;
    }
    return x;
}

```

## 6.6 Pollard Rho Brent

```

ll mult(ll a, ll b, ll mod) {
    return (__int128_t) a * b % mod;
}
ll f(ll x, ll c, ll mod) {
    return (mult(x, x, mod) + c) % mod;
}
ll pollard_rho_brent(ll n, ll x0 = 2, ll c = 1) {
    ll x = x0, g = 1, q = 1, xs, y, m = 128, l = 1;
    while (g == 1) {
        y = x;
        for (ll i = 1; i < l; i++) {
            x = f(x, c, n);
        }
        ll k = 0;
        while (k < l && g == 1) {
            xs = x;
            for (ll i = 0; i < m && i < l - k; i++) {
                x = f(x, c, n);
                q = mult(q, abs(y - x), n);
            }
            g = __gcd(q, n);
            k += m;
        }
        l *= 2;
    }
    if (g == n) {
        do {
            xs = f(xs, c, n);
            g = __gcd(abs(xs - y), n);
        } while (g == 1);
    }
    return g;
}

```

## 6.7 Range Sieve

```

vector<bool> range_sieve(ll l, ll r) {
    ll n = sqrt(r);
    vector<bool> is_prime(n + 1, true);
    vector<ll> prime;
    is_prime[0] = is_prime[1] = false;
    prime.push_back(2);
    for (ll i = 4; i <= n; i += 2) {
        is_prime[i] = false;
    }
    for (ll i = 3; i <= n; i += 2) {

```

```

    if (is_prime[i]) {
        prime.push_back(i);
        for (ll j = i * i; j <= n; j += i) {
            is_prime[j] = false;
        }
    }
}
vector<bool> result(r - 1 + 1, true);
for (ll i : prime) {
    for (ll j = max(i * i, (1 + i - 1) / i * i); j <= r;
        j += i) {
        result[j - 1] = false;
    }
}
if (1 == 1) {
    result[0] = false;
}
return result;
}

```

## 6.8 Segmented Sieve

```

vector<ll> segmented_sieve(ll n) {
    const ll S = 10000;
    ll nsqrt = sqrt(n);
    vector<char> is_prime(nsqrt + 1, true);
    vector<ll> prime;
    is_prime[0] = is_prime[1] = false;
    prime.push_back(2);
    for (ll i = 4; i <= nsqrt; i += 2) {
        is_prime[i] = false;
    }
    for (ll i = 3; i <= nsqrt; i += 2) {
        if (is_prime[i]) {
            prime.push_back(i);
            for (ll j = i * i; j <= nsqrt; j += i) {
                is_prime[j] = false;
            }
        }
    }
    vector<ll> result;
    vector<char> block(S);
    for (ll k = 0; k * S <= n; k++) {
        fill(block.begin(), block.end(), true);
        for (ll p : prime) {
            for (ll j = max((k * S + p - 1) / p, p) * p - k *
                S; j < S; j += p) {
                block[j] = false;
            }
        }
    }
}

```

```

    if (k == 0) {
        block[0] = block[1] = false;
    }
    for (ll i = 0; i < S && k * S + i <= n; i++) {
        if (block[i]) {
            result.push_back(k * S + i);
        }
    }
}
return result;
}

```

## 6.9 Tonelli Shanks

```

ll legendre(ll a, ll p) {
    return bin_pow_mod(a, (p - 1) / 2, p);
}
ll tonelli_shanks(ll n, ll p) {
    if (legendre(n, p) == p - 1) {
        return -1;
    }
    if (p % 4 == 3) {
        return bin_pow_mod(n, (p + 1) / 4, p);
    }
    ll Q = p - 1, S = 0;
    while (Q % 2 == 0) {
        Q /= 2;
        S++;
    }
    ll z = 2;
    for (; z < p; z++) {
        if (legendre(z, p) == p - 1) {
            break;
        }
    }
    ll M = S, c = bin_pow_mod(z, Q, p), t = bin_pow_mod(n,
        Q, p), R = bin_pow_mod(n, (Q + 1) / 2, p);
    while (t % p != 1) {
        if (t % p == 0) {
            return 0;
        }
        ll i = 1, t2 = t * t % p;
        for (; i < M; i++) {
            if (t2 % p == 1) {
                break;
            }
            t2 = t2 * t2 % p;
        }
        ll b = bin_pow_mod(c, bin_pow_mod(2, M - i - 1, p),
            p);
    }
}

```

```

    M = i;
    c = b * b % p;
    t = t * c % p;
    R = R * b % p;
}
return R;
}

```

## 7 Strings

### 7.1 Hashing

```

ll compute_hash(string const& s) {
    const ll p = 31, m = 1e9 + 9;
    ll hash_value = 0, p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) %
            m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}

```

### 7.2 Knuth Morris Pratt

```

vector<ll> prefix_function(string s) {
    ll n = (ll) s.length();
    vector<ll> pi(n);
    for (ll i = 1; i < n; i++) {
        ll j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) {
            j = pi[j - 1];
        }
        if (s[i] == s[j]) {
            j++;
        }
        pi[i] = j;
    }
    return pi;
}

```

### 7.3 Rabin Karp

```

vector<ll> rabin_karp(string const& s, string const& t)
{
    const ll p = 31, m = 1e9 + 9;
    ll S = s.size(), T = t.size();

```

```

    vector<ll> p_pow(max(S, T));
    p_pow[0] = 1;
    for (ll i = 1; i < (ll) p_pow.size(); i++) {
        p_pow[i] = (p_pow[i-1] * p) % m;
    }
    vector<ll> h(T + 1, 0);
    for (ll i = 0; i < T; i++) {
        h[i + 1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    }
    ll h_s = 0;
    for (ll i = 0; i < S; i++) {
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
    }
    vector<ll> occurrences;
    for (ll i = 0; i + S - 1 < T; i++) {
        ll cur_h = (h[i + S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m) {
            occurrences.push_back(i);
        }
    }
    return occurrences;
}

```

### 7.4 Suffix Array

```

vector<ll> sort_cyclic_shifts(string const& s) {
    ll n = s.size();
    const ll alphabet = 256;
    vector<ll> p(n), c(n), cnt(max(alphabet, n), 0);
    for (ll i = 0; i < n; i++) {
        cnt[s[i]]++;
    }
    for (ll i = 1; i < alphabet; i++) {
        cnt[i] += cnt[i - 1];
    }
    for (ll i = 0; i < n; i++) {
        p[--cnt[s[i]]] = i;
    }
    c[p[0]] = 0;
    ll classes = 1;
    for (ll i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]]) {
            classes++;
        }
        c[p[i]] = classes - 1;
    }
    vector<ll> pn(n), cn(n);
    for (ll h = 0; (1 << h) < n; ++h) {
        for (ll i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);

```

```

    if (pn[i] < 0) {
        pn[i] += n;
    }
}
fill(cnt.begin(), cnt.begin() + classes, 0);
for (ll i = 0; i < n; i++) {
    cnt[c[pn[i]]]++;
}
for (ll i = 1; i < classes; i++) {
    cnt[i] += cnt[i - 1];
}
for (ll i = n-1; i >= 0; i--) {
    p[--cnt[c[pn[i]]]] = pn[i];
}
cn[p[0]] = 0;
classes = 1;
for (ll i = 1; i < n; i++) {
    pair<ll, ll> cur = {c[p[i]], c[(p[i] + (1 << h)) %
        n]};
    pair<ll, ll> prev = {c[p[i - 1]], c[(p[i - 1] + (1
        << h)) % n]};
    if (cur != prev) {
        ++classes;
    }
    cn[p[i]] = classes - 1;
}
c.swap(cn);
return p;

```

```

}
vector<ll> build_suff_arr(string s) {
    s += (char) 0;
    vector<ll> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}

```

---

## 7.5 Z Function

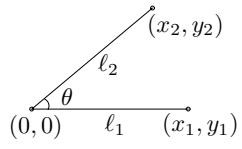
```

vector<ll> z_function(string s) {
    ll n = (ll) s.length();
    vector<ll> z(n);
    for (ll i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) {
            z[i] = min (r - i + 1, z[i - l]);
        }
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            ++z[i];
        }
        if (i + z[i] - 1 > r) {
            l = i, r = i + z[i] - 1;
        }
    }
    return z;
}

```

---

$f(n) = O(g(n))$	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 $b \leq s, \forall s \in S$ .	$\sum_{i=0}^n i c^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, \quad c \neq 1, \quad \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, \quad  c  < 1.$
$\liminf_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \inf \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	Harmonic series:
$\limsup_{n \rightarrow \infty} a_n$	$\lim_{n \rightarrow \infty} \sup \{a_i \mid i \geq n, i \in \mathbb{N}\}.$	$H_n = \sum_{i=1}^n \frac{1}{i}, \quad \sum_{i=1}^n i H_i = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}.$
$\binom{n}{k}$	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. $\binom{n}{k} = \frac{n!}{(n-k)!k!}, \quad 2. \sum_{k=0}^n \binom{n}{k} = 2^n, \quad 3. \binom{n}{k} = \binom{n}{n-k},$
$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$	Stirling numbers (2nd kind): Partitions of an $n$ element set into $k$ non-empty sets.	4. $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}, \quad 5. \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1},$
$\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle$	1st order Eulerian numbers: Permutations $\pi_1 \pi_2 \dots \pi_n$ on $\{1, 2, \dots, n\}$ with $k$ ascents.	6. $\binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k}, \quad 7. \sum_{k=0}^n \binom{r+k}{k} = \binom{r+n+1}{n},$
$\langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle$	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. $\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad 11. \left\{ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} n \\ n \end{smallmatrix} \right\} = 1,$
14. $\left[ \begin{smallmatrix} n \\ 1 \end{smallmatrix} \right] = (n-1)!,$	15. $\left[ \begin{smallmatrix} n \\ 2 \end{smallmatrix} \right] = (n-1)!H_{n-1},$	16. $\left[ \begin{smallmatrix} n \\ n \end{smallmatrix} \right] = 1, \quad 17. \left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] \geq \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\},$
18. $\left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] = (n-1) \left[ \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right] + \left[ \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right],$	19. $\left\{ \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right\} = \left[ \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \right] = \binom{n}{2},$	20. $\sum_{k=0}^n \left[ \begin{smallmatrix} n \\ k \end{smallmatrix} \right] = n!, \quad 21. C_n = \frac{1}{n+1} \binom{2n}{n},$
22. $\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \rangle = \langle \begin{smallmatrix} n \\ n-1 \end{smallmatrix} \rangle = 1,$	23. $\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle = \langle \begin{smallmatrix} n \\ n-1-k \end{smallmatrix} \rangle,$	24. $\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle = (k+1) \langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \rangle + (n-k) \langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \rangle,$
25. $\langle \begin{smallmatrix} 0 \\ k \end{smallmatrix} \rangle = \begin{cases} 1 & \text{if } k=0, \\ 0 & \text{otherwise} \end{cases}$	26. $\langle \begin{smallmatrix} n \\ 1 \end{smallmatrix} \rangle = 2^n - n - 1,$	27. $\langle \begin{smallmatrix} n \\ 2 \end{smallmatrix} \rangle = 3^n - (n+1)2^n + \binom{n+1}{2},$
28. $x^n = \sum_{k=0}^n \langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle \binom{x+k}{n},$	29. $\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \rangle = \sum_{k=0}^m \binom{n+1}{k} (m+1-k)^n (-1)^k,$	30. $m! \left\{ \begin{smallmatrix} n \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle \binom{k}{n-m},$
31. $\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \rangle = \sum_{k=0}^n \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} \binom{n-k}{m} (-1)^{n-k-m} k!,$	32. $\langle\langle \begin{smallmatrix} n \\ 0 \end{smallmatrix} \rangle\rangle = 1,$	33. $\langle\langle \begin{smallmatrix} n \\ n \end{smallmatrix} \rangle\rangle = 0 \quad \text{for } n \neq 0,$
34. $\langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle = (k+1) \langle\langle \begin{smallmatrix} n-1 \\ k \end{smallmatrix} \rangle\rangle + (2n-1-k) \langle\langle \begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \rangle\rangle,$	35. $\sum_{k=0}^n \langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle = \frac{(2n)n}{2^n},$	
36. $\left\{ \begin{smallmatrix} x \\ x-n \end{smallmatrix} \right\} = \sum_{k=0}^n \langle\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \rangle\rangle \binom{x+n-1-k}{2n},$	37. $\left\{ \begin{smallmatrix} n+1 \\ m+1 \end{smallmatrix} \right\} = \sum_k \binom{n}{k} \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} = \sum_{k=0}^n \left\{ \begin{smallmatrix} k \\ m \end{smallmatrix} \right\} (m+1)^{n-k},$	

<p>The Chinese remainder theorem: There exists a number <math>C</math> such that:</p> $C \equiv r_1 \pmod{m_1}$ $\vdots$ $C \equiv r_n \pmod{m_n}$ <p>if <math>m_i</math> and <math>m_j</math> are relatively prime for <math>i \neq j</math>. Euler's function: <math>\phi(x)</math> is the number of positive integers less than <math>x</math> relatively prime to <math>x</math>. If <math>\prod_{i=1}^n p_i^{e_i}</math> is the prime factorization of <math>x</math> then</p> $\phi(x) = \prod_{i=1}^n p_i^{e_i-1} (p_i - 1).$ <p>Euler's theorem: If <math>a</math> and <math>b</math> are relatively prime then</p> $1 \equiv a^{\phi(b)} \pmod{b}.$ <p>Fermat's theorem:</p> $1 \equiv a^{p-1} \pmod{p}.$ <p>The Euclidean algorithm: if <math>a &gt; b</math> are integers then</p> $\gcd(a, b) = \gcd(a \bmod b, b).$ <p>If <math>\prod_{i=1}^n p_i^{e_i}</math> is the prime factorization of <math>x</math> then</p> $S(x) = \sum_{d x} d = \prod_{i=1}^n \frac{p_i^{e_i+1} - 1}{p_i - 1}.$ <p>Perfect Numbers: <math>x</math> is an even perfect number iff <math>x = 2^{n-1}(2^n - 1)</math> and <math>2^n - 1</math> is prime. Wilson's theorem: <math>n</math> is a prime iff</p> $(n - 1)! \equiv -1 \pmod{n}.$ <p>Möbius inversion:</p> $\mu(i) = \begin{cases} 1 & \text{if } i = 1. \\ 0 & \text{if } i \text{ is not square-free.} \\ (-1)^r & \text{if } i \text{ is the product of } r \text{ distinct primes.} \end{cases}$ <p>If</p> $G(a) = \sum_{d a} F(d),$ <p>then</p> $F(a) = \sum_{d a} \mu(d) G\left(\frac{a}{d}\right).$ <p>Prime numbers:</p> $p_n = n \ln n + n \ln \ln n - n + n \frac{\ln \ln n}{\ln n} + O\left(\frac{n}{\ln n}\right),$ $\pi(n) = \frac{n}{\ln n} + \frac{n}{(\ln n)^2} + \frac{2!n}{(\ln n)^3} + O\left(\frac{n}{(\ln n)^4}\right).$	<p>Definitions:</p> <p><i>Loop</i> An edge connecting a vertex to itself.</p> <p><i>Directed</i> Each edge has a direction.</p> <p><i>Simple</i> Graph with no loops or multi-edges.</p> <p><i>Walk</i> A sequence <math>v_0 e_1 v_1 \dots e_\ell v_\ell</math>.</p> <p><i>Trail</i> A walk with distinct edges.</p> <p><i>Path</i> A trail with distinct vertices.</p> <p><i>Connected</i> A graph where there exists a path between any two vertices.</p> <p><i>Component</i> A maximal connected subgraph.</p> <p><i>Tree</i> A connected acyclic graph.</p> <p><i>Free tree</i> A tree with no root.</p> <p><i>DAG</i> Directed acyclic graph.</p> <p><i>Eulerian</i> Graph with a trail visiting each edge exactly once.</p> <p><i>Hamiltonian</i> Graph with a cycle visiting each vertex exactly once.</p> <p><i>Cut</i> A set of edges whose removal increases the number of components.</p> <p><i>Cut-set</i> A minimal cut.</p> <p><i>Cut edge</i> A size 1 cut.</p> <p><i>k-Connected</i> A graph connected with the removal of any <math>k - 1</math> vertices.</p> <p><i>k-Tough</i> <math>\forall S \subseteq V, S \neq \emptyset</math> we have <math>k \cdot c(G - S) \leq  S </math>.</p> <p><i>k-Regular</i> A graph where all vertices have degree <math>k</math>.</p> <p><i>k-Factor</i> A <math>k</math>-regular spanning subgraph.</p> <p><i>Matching</i> A set of edges, no two of which are adjacent.</p> <p><i>Clique</i> A set of vertices, all of which are adjacent.</p> <p><i>Ind. set</i> A set of vertices, none of which are adjacent.</p> <p><i>Vertex cover</i> A set of vertices which cover all edges.</p> <p><i>Planar graph</i> A graph which can be embedded in the plane.</p> <p><i>Plane graph</i> An embedding of a planar graph.</p> <hr/> $\sum_{v \in V} \deg(v) = 2m.$ <p>If <math>G</math> is planar then <math>n - m + f = 2</math>, so</p> $f \leq 2n - 4, \quad m \leq 3n - 6.$ <p>Any planar graph has a vertex with degree <math>\leq 5</math>.</p>	<p>Notation:</p> <p><math>E(G)</math> Edge set</p> <p><math>V(G)</math> Vertex set</p> <p><math>c(G)</math> Number of components</p> <p><math>G[S]</math> Induced subgraph</p> <p><math>\deg(v)</math> Degree of <math>v</math></p> <p><math>\Delta(G)</math> Maximum degree</p> <p><math>\delta(G)</math> Minimum degree</p> <p><math>\chi(G)</math> Chromatic number</p> <p><math>\chi_E(G)</math> Edge chromatic number</p> <p><math>G^c</math> Complement graph</p> <p><math>K_n</math> Complete graph</p> <p><math>K_{n_1, n_2}</math> Complete bipartite graph</p> <p><math>r(k, \ell)</math> Ramsey number</p> <hr/> <p style="text-align: center;">Geometry</p> <p>Projective coordinates: triples <math>(x, y, z)</math>, not all <math>x, y</math> and <math>z</math> zero.</p> $(x, y, z) = (cx, cy, cz) \quad \forall c \neq 0.$ <p>Cartesian      Projective</p> <hr/> <p><math>(x, y)</math>      <math>(x, y, 1)</math></p> <p><math>y = mx + b</math>      <math>(m, -1, b)</math></p> <p><math>x = c</math>      <math>(1, 0, -c)</math></p> <p>Distance formula, <math>L_p</math> and <math>L_\infty</math> metric:</p> $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$ $[ x_1 - x_0 ^p +  y_1 - y_0 ^p]^{1/p},$ $\lim_{p \rightarrow \infty} [ x_1 - x_0 ^p +  y_1 - y_0 ^p]^{1/p}.$ <p>Area of triangle <math>(x_0, y_0), (x_1, y_1)</math> and <math>(x_2, y_2)</math>:</p> $\frac{1}{2} \text{abs} \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix}.$ <p>Angle formed by three points:</p>  $\cos \theta = \frac{(x_1, y_1) \cdot (x_2, y_2)}{l_1 l_2}.$ <p>Line through two points <math>(x_0, y_0)</math> and <math>(x_1, y_1)</math>:</p> $\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0.$ <p>Area of circle, volume of sphere:</p> $A = \pi r^2, \quad V = \frac{4}{3} \pi r^3.$ <hr/> <p>If I have seen farther than others, it is because I have stood on the shoulders of giants. – Issac Newton</p>
---	---	--

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







