1 Data structures

1.1 Treap (with segment updates)

```
mt19937 rnd(time(NULL));
struct Node {
    int x;
    int y;
    int sz;
    bool rev;
    Node* l;
    Node* r;
    Node() {};
};
typedef Node* treap;
typedef pair<treap, treap> ptt;

treap newNode(int x, int y, int sz, Node* l,
↪  Node* r) {
    treap t = new Node();
    t->x = x;
    t->y = y;
    t->sz = sz;
    t->rev = 0;
    t->l = l;
    t->r = r;
    return t;
}
treap createNode(int x) {
    return newNode(x, rnd(), 1, NULL, NULL);
}
int getSize(treap t) {
    if (! t) return 0;
    return t->sz;
}
treap fix (treap t) {
    if (t) t->sz = 1 + getSize(t->l) +
    ↪  getSize(t->r);
    return t;
}
treap rev(treap t) {
    if (t) t->rev ^= 1;
    return t;
}
treap push(treap t) {
    if (! t) return t;
    if (t->rev) {
        t->l = rev(t->l);
        t->r = rev(t->r);
        swap(t->l, t->r);
        t->rev = 0;
    }
    return fix(t);
}
treap merge(treap a, treap b) {
    if (! a) return b;
    if (! b) return a;
    a = push(a);
    b = push(b);
    if (a->y > b->y) {
        a->r = merge(a->r, b);
        return fix(a);
    }
    else {
        b->l = merge(a, b->l);
        return fix(b);
    }
}
ptt splitK(treap t, int k) {
    if (getSize(t) <= k) return {t, NULL};
    if (k == 0) return {NULL, t};
    t = push(t);
    int szl = getSize(t->l);
    if (szl < k) {
        ptt p = splitK(t->r, k - szl - 1);
        t->r = p.first;
        return {fix(t), p.second};
    }
    else {
        ptt p = splitK(t->l, k);
        t->l = p.second;
        return {p.first, fix(t)};
    }
}
treap revSeg(treap t, int l, int r) {
    ptt p = splitK(t, l);
    ptt q = splitK(p.second, r - l);
    q.first = rev(q.first);
    t = merge(p.first, merge(q.first,
    ↪  q.second));
    return fix(t);
}
treap shift(treap t, int l, int r) {
    ptt p = splitK(t, l);
    ptt q = splitK(p.second, r - l);
    treap seg = q.first;
    ptt s = splitK(seg, getSize(seg) - 1);
    seg = merge(s.second, s.first);
    t = merge(merge(p.first, seg), q.second);
    return fix(t);
}
treap createfromVector(const vector<int> &a) {
    treap t = NULL;
    for (auto x: a) t = merge(t,
    ↪  createNode(x));
    return t;
}
void getVector(treap t, vector<int> &a) {
    if (! t) return;
    t = push(t);
    getVector(t->l, a);
    a.push_back(t->x);
    getVector(t->r, a);
}
```

1.2 Treap (as a binary search tree)

```
ptt split(treap t, int x) {
    if (! t) return {t, t};
    if (t->x < x) {
        ptt p = split(t->r, x);
        t->r = p.first;
        return {fix(t), p.second};
    }
    else {
        ptt p = split(t->l, x);
        t->l = p.second;
        return {p.first, fix(t)};
    }
}
treap insert(treap t, int x) {
    ptt p = split(t, x);
    treap q = createNode(x);
    return merge(p.first, merge(q, p.second));
}
li sumltX(treap t, int x) {
    if (! t) return 0ll;
    if (t->x < x) return t->x + getSum(t->l) +
    ↪  sumltX(t->r, x);
    else return sumltX(t->l, x);
}
bool contains(treap t, int x) {
    if (! t) return false;
    if (t->x == x) return true;
    if (t->x > x) return contains(t->l, x);
    else return contains(t->r, x);
}
```

1.3 Segment tree

```
void build(int v, int l, int r) {
    if (l == r - 1) {
        T[v] = a[l];
        return;
    }
    int m = (l + r) / 2;
    build(2 * v + 1, l, m);
    build(2 * v + 2, m, r);
    T[v] = T[2 * v + 1] + T[2 * v + 2];
}
void push(int v, int l, int r) {
    if (l != r - 1) {
        ps[2 * v + 1] += ps[v];
        ps[2 * v + 2] += ps[v];
    }
    T[v] += ps[v] * (r - l);
    ps[v] = 0;
}
void upd(int v, int l, int r, int L, int R,
↪  int val) {
    push(v, l, r);
    if (L >= R) return;
    if (l == L && r == R) {
        ps[v] = val;
        push(v, l, r);
        return;
    }
    int m = (l + r) / 2;
    upd(2 * v + 1, l, m, L, min(m, R), val);
    upd(2 * v + 2, m, r, max(m, L), R, val);
    T[v] = T[2 * v + 1] + T[2 * v + 2];
}
int query(int v, int l, int r, int pos) {
    push(v, l, r);
    if (l == r - 1) return T[v];
    int m = (l + r) / 2;
    int res = 0;
    if (pos < m) {
        res = query(2 * v + 1, l, m, pos);
        push(2 * v + 2, m, r);
    }
    else {
        res = query(2 * v + 2, m, r, pos);
        push(2 * v + 1, l, m);
    }
    T[v] = T[2 * v + 1] + T[2 * v + 2];
    return res;
}
// Спуск по ДО
int trav(int v, int l, int r, int x) {
    if (l == r - 1) return T[v] >= x ? l : -1;
    int m = (l + r) / 2;
    if (T[2 * v + 1] >= x) return trav(2 * v +
    ↪  1, l, m, x);
    else return trav(2 * v + 2, m, r, x);
}
```

1.4 Merge sort tree

```
void recalc(int v) {
    vector<int> &left = tr[2 * v + 1];
    vector<int> &right = tr[2 * v + 2];
    int i = 0;
    int j = 0;
    while (i < left.size() && j <
    ↪  right.size()) {
        if (left[i] < right[j])
        ↪  tr[v].pb(left[i++]);
        else tr[v].pb(right[j++]);
    }
    while (i < left.size())
    ↪  tr[v].pb(left[i++]);
    while (j < right.size())
    ↪  tr[v].pb(right[j++]);
}
void build(int v, int l, int r, vector<ptt>
↪  &p) {
    if (l == r - 1) {
        tr[v].pb(p[l].second);
        return;
    }
```

```
    int m = (l + r) / 2;
    build(2 * v + 1, l, m, p);
    build(2 * v + 2, m, r, p);
    recalc(v);
}
int countltX(int v, int l, int r, int L, int
↪ R, int x) {
    if (L >= R) return 0;
    if (L == l && R == r)
        return lower_bound(tr[v].begin(),
        ↪ tr[v].end(), x) - tr[v].begin();
    int m = (l + r) / 2;
    int left = countltX(2 * v + 1, l, m, L,
    ↪ min(R, m), x);
    int right = countltX(2 * v + 2, m, r,
    ↪ max(L, m), R, x);
    return left + right;
}
```

1.5 Sparse table

```
int T[lg][N];
void build(vector<int> &a) {
    int n = a.size();
    for (int i = 0; i < n; i++)
        T[0][i] = a[i];
    for (int i = 1; i < lg; i++) {
        int len = 1 << (i - 1);
        for (int j = 0; j + (len << 1) <= n;
        ↪ j++) {
            T[i][j] = __gcd(T[i - 1][j], T[i -
            ↪ 1][j + len]);
        }
    }
}
int query(int l, int r) {
    int len = 32 - __builtin_clz(r - l);
    len--;
    return __gcd(T[len][l], T[len][r - (1 <<
    ↪ len)]);
}
```

1.6 DSU

```
int p[N], sz[N];
void init() {
    for (int i = 0; i < N; i++) {
        sz[i] = 1;
        p[i] = i;
    }
}
int get_leader(int v) {
    if (v == p[v]) return v;
    return (p[v] = get_leader(p[v]));
}
bool merge(int v, int u) {
    v = get_leader(v);
    u = get_leader(u);
    if (v == u) return false;
    if (sz[v] > sz[u]) swap(v, u);
    p[v] = u;
    sz[u] += sz[v];
    return true;
}
```

1.7 Priority queue

```
import heapq
heapq.heappush(arr, val)
ans = heapq.heappop(arr)
```

2 String algorithms

2.1 Trie

```
struct node {
    map<char, int> go;
    int term;
    node() {};
};
node trie[N];
int sz = 0;
int new_node() {
    trie[sz].go.clear();
    trie[sz].term = -1;
    return sz++;
}
void add_string(string s, int i) {
    int cur = 0;
    for (char c : s) {
        if (! trie[cur].go.count(c)) {
            trie[cur].go[c] = new_node();
        }
        cur = trie[cur].go[c];
    }
    trie[cur].term = i;
}
int find(string t) {
    int cur = 0;
    for (char c : t) {
        if (! trie[cur].go.count(c)) return
        ↪ -1;
        cur = trie[cur].go[c];
    }
    return trie[cur].term;
}
```

2.2 String hashing

```
const li P = 31;
const li M = 998244353;
vector<int> get_hash(string s) {
        int n = s.size();
        vector<int> f(n + 1, 0);
        for (int i = n - 1; i >= 0; i--) {
```

```
                f[i] = add(mul(f[i + 1], P),
                ↪  (s[i] - 'a' + 1));
        }
        return f;
}
int sub_hash(vector<int> &f, int j, int len) {
    int sub = add(f[j], -mul(f[j + len],
    ↪  mod_pow(P, len)));
    return sub;
}
```

2.3 KMP

```
def kmp(s):
    n = len(s)
    pr = prefix_function(s)
    aut = [[0] * 26 for _ in range(n + 1)]
    for i in range(n + 1):
        for j in range(26):
            if i < n and s[i] == chr(ord('A')
            ↪  + j):
                aut[i][j] = i + 1
            elif i == 0:
                aut[i][j] = 0
            else:
                aut[i][j] = aut[pr[i - 1]][j]
    return aut
```

2.4 Prefix function

```
def pr(s):
    n = len(s)
    pr = [0] * n
    for i in range(1, n):
        j = pr[i - 1]
        while j > 0 and s[j] != s[i]:
            j = pr[j - 1]
        if s[i] == s[j]:
            j += 1
        pr[i] = j
    return pr
```

2.5 Z-function

```
def z(s):
    n = len(s)
    z = [0] * n
    l, r = 0, 0
    for i in range(1, n):
        j = max(0, min(z[i - l], r - i))
        while i + j < n and s[j] == s[i + j]:
            j += 1
        z[i] = j
        if i + j > r:
            l = i
            r = i + j
    return z
```

3 Graph algorithms

3.1 BFS

```
vector<int> d(n, INF);
queue<int> q;
q.push(s);
d[s] = 0;
while (! q.empty()) {
    int v = q.front();
    q.pop();
    for (int u : g[v]) {
        if (d[u] == INF) {
            d[u] = d[v] + 1;
            q.push(u);
        }
    }
}
```

3.2 Djikstra

```
priority_queue<ptt> q;
q.push({0, v1});
vector<bool> used(n);
vector<int> d(n, INF);
d[v1] = 0;
while (! q.empty()) {
    int v = q.top().second;
    q.pop();
    if (used[v]) continue;
    used[v] = true;
    for (auto p: g[v]) {
        int u = p.first;
        int w = p.second;
        if (d[u] > d[v] + w) {
            d[u] = d[v] + w;
            q.push({-d[u], u});
        }
    }
}
```

3.3 Floyd

```
for (int i = 0; i < n; i++) {
    for (int v = 0; v < n; v++) {
        for (int u = 0; u < n; u++) {
            d[u][v] = min(d[u][v], d[u][i] +
            ↪  d[i][v]);
        }
    }
}
```

3.4 Ford-Bellman

```
d = [INF] * n
d[0] = 0
for i in range(n - 1):
    for v, u, w in g:
```

```
        d[u] = min(d[u], d[v] + w)
```

После n-ной итерации что-то изменилось ⇒ существует цикл отрицательного веса.

3.5 Cycle in directed graph

```
bool dfs(int v, vector<vector<int>> &g,
↪  vector<int> &used, vector<int> &p, int
↪  &c_st, int &c_end) {
    used[v] = 1;
    for (auto u : g[v]) {
        if (! used[u]) {
            p[u] = v;
            if (dfs(u, g, used, p, c_st,
            ↪  c_end)) return true;
        }
        if (used[u] == 1) {
            c_st = u;
            c_end = v;
            return true;
        }
    }
    used[v] = 2;
    return false;
}
void solve() {
    int c_st = -1, c_end = -1;
    for (int i = 0; i < n; i++) {
        if (used[i]) continue;
        dfs(i, g, used, p, c_st, c_end);
        if (c_st != -1) {
            vector<int> cycle;
            while (c_end != c_st) {
                cycle.push_back(c_end);
                c_end = p[c_end];
            }
            cycle.push_back(c_end);
            reverse(cycle.begin(),
            ↪  cycle.end());
            cout << cycle.size() << '\n';
            for (int x: cycle) cout << x + 1
            ↪  << ' ';
            cout << c_end + 1 << '\n';
            return;
        }
    }
}
```

3.6 Strongly connected components

```
void topsort(int v, vector<vector<int>> &g,
↪  vector<bool> &used, vector<int> &ord) {
    used[v] = true;
    for (auto u : g[v]) {
        if (! used[u]) topsort(u, g, used,
        ↪  ord);
    }
    ord.push_back(v);
}
void dfs(int v, int cnt, vector<vector<int>>
↪  &tg, vector<int> &comp) {
    comp[v] = cnt;
    for (int u : tg[v]) {
        if (! comp[u]) dfs(u, cnt, tg, comp);
    }
}
void solve() {
    vector<int> comp(n);
    vector<bool> used(n);
    vector<int> ord;
    for (int i = 0; i < n; i++) {
        if (! used[i]) topsort(i, g, used,
        ↪  ord);
    }
    reverse(ord.begin(), ord.end());
    int cnt = 1;
    for (auto v: ord) {
        if (! comp[v]) dfs(v, cnt++, tg,
        ↪  comp);
    }
}
```

3.7 Bridges

```
int dfs(int v, int p, vector<bool> &used,
↪  vector<vector<edge>> &g, vector<bool>
↪  &is_br, vector<int> &d) {
    used[v] = true;
    int ans = INF;
    for (auto e: g[v]) {
        int to = v ^ e.v ^ e.u;
        if (! used[to]) {
            d[to] = d[v] + 1;
            int mn = dfs(to, v, used, g,
            ↪  is_br, d);
            if (mn > d[v]) {
                is_br[e.idx] = true;
                cnt++;
            }
            ans = min(ans, mn);
        }
        else if (to != p) ans = min(ans,
        ↪  d[to]);
    }
    return ans;
}
```

3.8 Articulation points

```
int dfs(int v, int p, vector<bool> &used,
↪  vector<vector<int>> &g, vector<bool>
↪  &is_br, vector<int> &d) {
    used[v] = true;
    int ans = INF;
```

```
    int subtree = 0;
    for (auto u: g[v]) {
        if (! used[u]) {
            d[u] = d[v] + 1;
            int mn = dfs(u, v, used, g, is_br,
            ↪ d);
            if (mn >= d[v] && p != -1) {
                is_br[v] = true;
            }
            ans = min(ans, mn);
            subtree++;
        }
        else if (u != p) ans = min(ans, d[u]);
    }
    if (subtree > 1 && p == -1) {
        is_br[v] = true;
    }
    return ans;
}
```

3.9 Binary lifting, LCA, min on path

```
int up[N][lg];
int min_up[N][lg];
int tin[N], tout[N];
int t = 0;
void dfs(int v, int p, int w,
↪ vector<vector<ptt>> &g) {
    tin[v] = t++;
    for (ptt e : g[v]) {
        int u = e.first;
        int w = e.second;
        if (u == p) continue;
        up[u][0] = v;
        min_up[u][0] = w;
        for (int i = 1; i < lg; i++) {
            up[u][i] = up[up[u][i - 1]][i -
            ↪ 1];
            min_up[u][i] = min(min_up[u][i -
            ↪ 1], min_up[up[u][i - 1]][i -
            ↪ 1]);
        }
        d[u] = d[v] + 1;
        dfs(u, v, w, g);
    }
    tout[v] = t;
}
bool is_ancestor(int v, int u) {
    return (tin[v] <= tin[u] && tout[v] >=
    ↪ tout[u]);
}
int lca(int v, int u) {
    if (is_ancestor(v, u)) return v;
    if (is_ancestor(u, v)) return u;
    int cur = v;
    for (int i = lg - 1; i >= 0; i--) {
        if (! is_ancestor(up[cur][i], u)) {
            cur = up[cur][i];
        }
    }
    return up[cur][0];
}
int min_vert(int v, int u) {
    if (v == u) return INF;
    int ans = INF;
    int cur = v;
    for (int i = lg - 1; i >= 0; i--) {
        if (! is_ancestor(up[cur][i], u)) {
            ans = min(ans, min_up[cur][i]);
            cur = up[cur][i];
        }
    }
    return min(min_up[cur][0], ans);
}
int min_path(int v, int u) {
    if (is_ancestor(v, u)) return min_vert(u,
    ↪ v);
    if (is_ancestor(u, v)) return min_vert(v,
    ↪ u);
    int lc = lca(v, u);
    return min(min_vert(v, lc), min_vert(u,
    ↪ lc));
}
int dist(int v, int u) {
    return d[v] + d[u] - 2 * d[lca(v, u)];
}
```

3.10 LCA with RMQ

```
vector<int> g[N];
vector<ptt> ord;
int d[N];
int idx[N];
ptt T[lg][N];
void dfs(int v, int p) {
    ord.pb({d[v], v});
    for (int u : g[v]) {
        if (u != p) {
            d[u] = d[v] + 1;
            dfs(u, v);
            ord.pb({d[v], v});
        }
    }
}
void build(int n) {
    for (int i = 0; i < N; i++) idx[i] = -1;
    for (int i = 0; i < n; i++) {
        if (idx[ord[i].second] == -1)
            idx[ord[i].second] = i;
    }
    for (int i = 0; i < n; i++) T[0][i] =
    ↪ ord[i];
    for (int i = 1; i < lg; i++) {
        int len = 1 << (i - 1);
        for (int j = 0; j + len < n; j++) {
```

```
            T[i][j] = min(T[i - 1][j], T[i -
            ↪ 1][j + len]);
        }
    }
}
int lca(int v, int u) {
    v = idx[v];
    u = idx[u];
    if (v > u) swap(v, u);
    u++;
    int x = __lg(u - v);
    int len = 1 << x;
    return min(T[x][v], T[x][u - len]).second;
}
```

3.11 MST

```
sort(edges.begin(), edges.end());
li ans = 0;
for (edge e : edges) {
    if (merge(e.v, e.u)) {
        g_mst[e.v].pb({e.u, e.w});
        g_mst[e.u].pb({e.v, e.w});
        in_mst[e.i] = true;
        ans += e.w;
    }
}
```

3.12 Rerooting

```
int dp[N], sz[N], ans[N];
void link(int v, int to) {
    sz[v] += sz[to];
    dp[v] += dp[to] + sz[to];
}
void cut(int v, int to) {
    sz[v] -= sz[to];
    dp[v] -= dp[to] + sz[to];
}
void dfs(int v, int p, vector<vector<int>> &g)
↪ {
    sz[v] = 1;
    dp[v] = 0;
    for (int u : g[v]) {
        if (u == p) continue;
        dfs(u, v, g);
        link(v, u);
    }
}
void recalc(int v, int p, vector<vector<int>>
↪ &g) {
    ans[v] = dp[v];
    for (int u : g[v]) {
        if (u == p) continue;
        cut(v, u);
        link(u, v);
        recalc(u, v, g);
        cut(u, v);
        link(v, u);
    }
}
```

3.13 Maximum matching on trees

```
ptt dp[N][2];
ptt best(ptt x, ptt y) {
    if (x.first > y.first) return x;
    else if (y.first > x.first) return y;
    else return {x.first, add(x.second,
    ↪ y.second)};
}
ptt merge(ptt x, ptt y) {
    return {x.first + y.first, mul(x.second,
    ↪ y.second)};
}
void dfs(int v, int p, vector<vector<int>> &g)
↪ {
    vector<ptt> dp0;
    vector<ptt> dp1;
    for (auto u : g[v]) {
        if (u != p) {
            dfs(u, v, g);
            dp0.pb(dp[u][0]);
            dp1.pb(dp[u][1]);
        }
    }
    int n = dp0.size();
    vector<ptt> pr(n + 1, {0, 1});
    vector<ptt> suf(n + 1, {0, 1});
    for (int i = 0; i < n; i++) pr[i + 1] =
    ↪ merge(pr[i], best(dp0[i], dp1[i]));
    for (int i = n - 1; i >= 0; i--) suf[i] =
    ↪ merge(suf[i + 1], best(dp0[i],
    ↪ dp1[i]));
    dp[v][0] = pr[n];
    for (int i = 0; i < n; i++) {
        dp[v][1] = best(dp[v][1],
        ↪ merge(merge(pr[i], suf[i + 1]),
        ↪ {dp0[i].first + 1,
        ↪ dp0[i].second}));
    }
}
```

3.14 Maximum matching (Kuhn)

```
bool used[N];
int mt[N], mt_left[N];
vector<int> g[N];
bool kuhn(int x) {
    if (used[x]) return false;
    used[x] = true;
    for (auto y : g[x])
        if (mt[y] == -1) {
            mt[y] = x;
            mt_left[x] = y;
```

```
            return true;
        }
    for (auto y : g[x])
        if (kuhn(mt[y])) {
            mt[y] = x;
            mt_left[x] = y;
            return true;
        }
    return false;
}
void max_matching() {
    for (int i = 0; i < n1; i++)
        mt_left[i] = -1;
    for (int i = 0; i < n2; i++)
        mt[i] = -1;
    while (true) {
        bool changed = false;
        for (int i = 0; i < n1; i++)
            used[i] = false;
        for (int i = 0; i < n1; i++)
            if (mt_left[i] == -1)
                changed |= kuhn(i);
        if (! changed)
            break;
    }
}
```

3.15 Path cover, edge cover (Kuhn)

```
int mt[N];
bool dfs(int v, vector<vector<int>>& g) {
    if (used[v]) return false;
    used[v] = true;
    for (int u : g[v]) {
        if (mt[u] == -1 || dfs(mt[u], g)) {
            mt[u] = v;
            return true;
        }
    }
    return false;
}
void path_cover() {
    for (int i = 0; i < N; i++) mt[i] = -1;
    for (int i = 0; i < n; i++) {
        for (int i = 0; i < N; i++) used[i] =
        ↪ false;
        dfs(i, g);
    }
    int kl = 0;
    for (int i = 0; i < N; i++) {
        if (mt[i] != -1) {
            kl++;
        }
    }
    cout << n - kl << '\n';
    vector<vector<int>> paths(n, vector<int>
    ↪ (0));
    for (int i = 0; i < n; i++) {
        int cur = i;
        vector<int> ans;
        while (cur != -1) {
            ans.push_back(cur + 1);
            cur = mt[cur];
        }
        reverse(ans.begin(), ans.end());
        if (paths[ans[0] - 1].size() <
        ↪ ans.size()) {
            paths[ans[0] - 1] = ans;
        }
    }
}
void edge_cover() {
    for (int i = 0; i < N; i++) mt[i] = -1;
    for (int i = 0; i < n; i++) {
        for (int i = 0; i < N; i++) used[i] =
        ↪ false;
        dfs(i, g);
    }
    int kl = 0;
    vector<pair<int, int>> mec;
    vector<bool> cov_l(n, false);
    vector<bool> cov_r(m, false);
    for (int i = 0; i < N; i++) {
        if (mt[i] != -1) {
            kl++;
            mec.push_back(make_pair(mt[i] + 1,
            ↪ i + 1));
            cov_l[mt[i]] = true;
            cov_r[i] = true;
        }
    }
    for (int i = 0; i < n; i++) {
        if (!cov_l[i]) {
            if (g[i].size() > 0) {
                mec.push_back(make_pair(i + 1,
                ↪ g[i].front() + 1));
                kl++;
            }
        }
    }
    for (int i = 0; i < m; i++) {
        if (!cov_r[i]) {
            if (g1[i].size() > 0) {

                ↪ mec.push_back(make_pair(g1[i].front()
                ↪ + 1, i + 1));
                kl++;
            }
        }
    }
}
```

3.16 2-SAT

```
int getVertex(int x) {
    if (x > 0) return (x - 1) * 2;
```

```
    if (x < 0) return (abs(x) - 1) * 2 + 1;
}
void addEdge(int x, int y) {
    g[x].push_back(y);
    gt[y].push_back(x);
}
void addOR(int x, int y) {
    int vx = getVertex(x);
    int vy = getVertex(y);
    addEdge(vx ^ 1, vy);
    addEdge(vy ^ 1, vx);
}
int variable(int x) {
     return (x / 2 + 1) * (x % 2 == 0 ? 1 :
     ↪  -1);
}
pair<bool, vector<int>> twoSAT(int n,
↪  vector<pair<int, int>> ORs) v{
    int m = ORs.size();
    int V = 2 * n;
    for (int i = 0; i < V; i++) {
        g[i].clear();
        gt[i].clear();
    }
    for (int i = 0; i < m; i++) {
        int x = ORs[i].first;
        int y = ORs[i].second;
        addOR(x, y);
    }
    vector<int> comps = SCC(V);
    bool ans = true;
    vector<int> res;
    for (int i = 0; i < n; i++) {
        int v1 = i * 2;
        int v0 = i * 2 + 1;
        if (comp[v1] == comp[v0])
            ans = false;
        else if (comp[v1] > comp[v0])
            res.push_back(variable(v1));
        else
            res.push_back(variable(v0));
    }
    return make_pair(ans, res);
}
```

3.17 Flows

```
class Edge:
    def __init__(self, to, w, f):
        self.to = to
        self.w = w
        self.f = f

def add_edge(fr, to, w):
    g[fr].append(len(e))
    e.append(Edge(to, w, 0))
    g[to].append(len(e))
    e.append(Edge(fr, 0, 0))

def rem(edge):
    return edge.w - edge.f

def dfs(v, f, k):
    if used[v]:
        return 0
    used[v] = True
    if v == T:
        return f
    for idx in g[v]:
        a = e[idx]
        r = rem(a)
        if r < k:
            continue
        pushed = dfs(a.to, min(r, f), k)
        if pushed:
            a.f += pushed
            e[idx ^ 1].f -= pushed
            return pushed
    return 0

S = 0
T = n - 1
ans = 0
for k in range(30, -1, -1):
    min_flow = 2 ** k
    while True:
        used = [False] * n
        flow = dfs(S, INF, min_flow)
        if not flow:
            break
        ans += flow
print(ans)
```

4 SQRT heuristics

4.1 Sqrt decomposition

```
sq = int(n ** 0.5)
b = [0] * (sq + 2)
for i in range(n):
    b[i // sq] += a[i]
for _ in range(m):
    q = [int(x) for x in input().split()]
    if q[0] == 0:
        t, l, r = q
        l -= 1
        r -= 1
        sm = 0
        while l <= r:
            if l % sq == 0 and l + sq <= r:
                sm += b[l // sq]
                l += sq
            else:
                sm += a[l]
                l += 1
```

```
        print(sm)
    else:
        t, i, x = q
        i -= 1
        b[i // sq] += (x - a[i])
        a[i] = x
```

4.2 Query decomposition

```
int sq = sqrt(n);
vector<array<int, 3>> qs;
vector<li> diff(n);
for (int i = 0; i < n; i++) {
    diff[i] = a[i] - (i == 0 ? 0 : a[i - 1]);
}
for (int i = 0; i < q; i++) {
    int t;
    cin >> t;
    if (t == 1) {
        int l, r, x;
        cin >> l >> r >> x;
        l--;
        diff[l] += x;
        diff[r] -= x;
        qs.pb({l, r, x});
    }
    else {
        int i;
        cin >> i;
        i--;
        li ans = a[i];
        for (auto e : qs) {
            int l = e[0];
            int r = e[1];
            int x = e[2];
            if (l <= i && i < r) ans += x;
        }
        cout << ans << '\n';
    }
    if ((i + 1) % sq == 0) {
        qs.clear();
        for (int i = 1; i < n; i++) diff[i] +=
        ↪ diff[i - 1];
        a = diff;
        for (int i = 0; i < n; i++) {
            diff[i] = a[i] - (i == 0 ? 0 : a[i
            ↪ - 1]);
        }
    }
}
```

4.3 Mo's algorithm

```
vector<array<int, 3>> q;
for (int i = 0; i < m; i++) {
    int l, r;
    cin >> l >> r;
    l--;
    r--;
    q.pb({l, r, i});
}
int sq = sqrt(n);
auto comp = [&] (array<int, 3> a, array<int,
↪ 3> b) {
    int a0 = a[0] / sq;
    int b0 = b[0] / sq;
    return (a0 < b0 || (a0 == b0 && (a0 % 2 ==
    ↪ 0 ? a[1] < b[1] : a[1] > b[1])));
};
sort(q.begin(), q.end(), cmp);
int l = 0;
int r = -1;
int d = 0;
vector<int> ans(m);
vector<int> mp(n + 1);
auto add = [&](int x, int fl) {
    if (x > n) return;
    if (mp[x] == x) d--;
    mp[x] += fl;
    if (mp[x] == x) d++;
};
for (int i = 0; i < m; i++) {
    int L = q[i][0];
    int R = q[i][1];
    int idx = q[i][2];
    while (l > L) {
        add(a[--l], 1);
    }
    while (r < R) {
        add(a[++r], 1);
    }
    while (l < L) {
        add(a[l++], -1);
    }
    while (r > R) {
        add(a[r--], -1);
    }
    ans[idx] = d;
}
```

5 DP

5.1 LIS

```
int lis(vector<int> &v) {
    vector<int> ls;
    for (int i = 0; i < n; ++i) {
        auto it = lower_bound(ls.begin(),
        ↪ ls.end(), v[i]);
        if (it == ls.end())
            ls.push_back(v[i]);
        else
            *it = v[i];
    }
    return ls.size();
```

```
}
```

5.2 LCS

```
dp = [[0] * (m + 1) for _ in range(n + 1)]
for i in range(n + 1):
    for j in range(m + 1):
        if i < n:
            dp[i + 1][j] = max(dp[i + 1][j],
            ↪  dp[i][j])
        if j < m:
            dp[i][j + 1] = max(dp[i][j + 1],
            ↪  dp[i][j])
        if i < n and j < m:
            dp[i + 1][j + 1] = max(dp[i + 1][j
            ↪  + 1], dp[i][j] + (s[i] ==
            ↪  t[j]))
print(dp[-1][-1])
```

5.3 Sums with certificate

```
dp = [0] * (sm + 1)
last = [-1] * (sm + 1)
dp[0] = 1
for i in range(n):
    for j in range(sm, -1, -1):
        if j + a[i] <= sm:
            if dp[j] and not dp[j + a[i]]:
                dp[j + a[i]] = 1
                last[j + a[i]] = i
for x in t:
    if x > sm or not dp[x]:
        print(-1)
        continue
    ans = list()
    cur = x
    while cur > 0:
        ans.append(last[cur] + 1)
        cur -= a[last[cur]]
    print(*ans[::-1])
```

5.4 Log trick

```
vector<int> w; // веса
vector<int> c; // стоимости
for (auto& [x, y] : mp) {
    int cur = 0;
    for (int i = 0; i < 18; i++) {
        if (cur + (1 << i) <= y) {
            cur += (1 << i);
            w.pb((1 << i) * x);
            c.pb(1 << i);
        }
        else break;
    }
    for (int i = 0; i < 18; i++) {
        if ((y - cur) & (1 << i)) {
            w.pb((1 << i) * x);
            c.pb(1 << i);
        }
    }
}
```

5.5 Bitset optimization

```
int p[N];
fill(p, p + N, -1);
bitset<N> dp;
dp[0] = 1;
for (int i = 0; i < n; ++i) {
    auto dp2 = dp;
    dp2 |= dp << a[i];
    dp ^= dp2;
    for (int s = dp._Find_first(); s <
    ↪  dp.size(); s = dp._Find_next(s))
        p[s] = i;
    dp = dp2;
}
for (int i = 0; i < q; ++i) {
    int s; cin >> s;
    if (!dp[s]) continue;
    int pr = p[s];
    vector<int> path;
    while (~pr) {
        path.push_back(pr + 1);
        s -= a[p[s]];
        pr = p[s];
    }
    reverse(path.begin(), path.end());
}
```

5.6 Matrix optimization

```
const int K = 2;
typedef array<int, K> vec;
typedef array<vec, K> mtx;

mtx mul(const mtx &a, const mtx &b) {
    mtx res;
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < K; j++) {
            res[i][j] = 0;
        }
    }
    for (int k = 0; k < K; k++) {
        for (int i = 0; i < K ; i++) {
            for (int j = 0; j < K; j++) {
                res[i][j] = add(res[i][j],
                ↪  mul(a[i][k], b[k][j]));
            }
        }
    }
    return res;
}
vec mul(const mtx &a, const vec &b) {
    vec res;
```

```
    for (int i = 0; i < K; i++) {
        res[i] = 0;
    }
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < K; j++) {
            res[i] = add(res[i], mul(a[i][j],
            ↪ b[j]));
        }
    }
    return res;
}
mtx unit() {
    mtx res;
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < K; j++) {
            res[i][j] = (i == j ? 1 : 0);
        }
    }
    return res;
}
mtx binpow(const mtx &a, li n) {
    mtx ans = unit();
    mtx base = a;
    while (n) {
        if (n & 1) ans = mul(ans, base);
        base = mul(base, base);
        n >>= 1;
    }
    return ans;
}
void solve() { // n-ное число Фибоначчи
    li n;
    cin >> n;
    mtx T;
    T[0][0] = 0;
    T[0][1] = T[1][0] = T[1][1] = 1;
    vec f;
    f[0] = 0;
    f[1] = 1;
    vec ans = mul(binpow(T, n), f);
    cout << ans[0] << '\n';
}
```

5.7 Meet-in-the-middle

```
vector<int> sums(vector<int> &a) {
    int n = a.size();
    vector<int> s = {0};
    for (int i = 0; i < n; i++) {
        int k = s.size();
        for (int j = 0; j < k; j++) {
            s.pb(add(s[j], a[i]));
        }
    }
    sort(s.begin(), s.end());
    return s;
}
void solve() {
    int k = n / 2;
    vector<int> left(a.begin(), a.begin() +
    ↪ k);
    vector<int> right(a.begin() + k, a.end());
    auto lsm = sums(left);
    auto rsm = sums(right);
    int ans = 0;
    for (int i = 0; i < lsm.size(); i++) {
        int j = upper_bound(rsm.begin(),
        ↪ rsm.end(), M - lsm[i] - 1) -
        ↪ rsm.begin();
        if (j > 0) ans = max(ans, add(lsm[i],
        ↪ rsm[j - 1]));
        ans = max(ans, add(lsm[i],
        ↪ rsm.back()));
    }
}
```

5.8 SOS DP

```
vector<li> transform(vector<li> a, bool rev) {
    int n = int(a.size());
    int ln = __builtin_popcount(n - 1);
    for (int i = 0; i < ln; i++) {
        for (int j = 0; j < n; j++) {
            if ((j >> i) & 1) a[j] += (rev ?
            ↪ -1 : 1) * a[j ^ (1 << i)];
        }
    }
    return a;
}
```

5.9 Fence

```
vector<int> fence(vector<int> &a) {
    int n = a.size();
    vector<int> h(n);
    h[0] = -1;
    for (int i = 1; i < n; i++) {
        int cur = i - 1;
        while (cur != -1 && a[cur] >= a[i])
        ↪ cur = h[cur];
        h[i] = cur;
    }
    return h;
}
```

6 Permutations

6.1 Inversions (merge sort)

```
vector<int> merge_sort(int l, int r, li &inv)
↪ {
    if (l >= r) return {p[l]};
    int m = (l + r) / 2;
    auto left = merge_sort(l, m, inv);
    auto right = merge_sort(m + 1, r, inv);
```

```
    int i = 0;
    int j = 0;
    vector<int> res;
    while (i < left.size() && j <
    ↪  right.size()) {
        if (left[i] < right[j]) {
            inv += j;
            res.pb(left[i++]);
        }
        else res.pb(right[j++]);
    }
    while (i < left.size()) {
        res.pb(left[i++]);
        inv += j;
    }
    while (j < right.size()) {
        res.pb(right[j++]);
    }
    return res;
}
```

6.2 K-th permutation

```
vector<int> ans(n);
vector<bool> used(n + 1);
for (int i = 0; i < n; i++) {
    int num = -1;
    for (int j = 1; j <= n; j++) {
        if (used[j]) continue;
        if (cnt + fact[n - i - 1] < k) cnt +=
        ↪  fact[n - i - 1];
        else {
            num = j;
            break;
        }
    }
    ans[i] = num;
    used[num] = true;
}
```

6.3 Permutation index

```
cnt = 0
for i in range(n):
    diff = 0
    for j in range(i):
        if a[j] < a[i]:
            diff += 1
    cnt += (a[i] - diff - 1) * fact[n - i - 1]
print(cnt + 1)
```

6.4 Permutation cycles

```
set<int> used;
for (int i = 0; i < n; i++) used.insert(i);
int i = 0;
vector<int> cycles;
while (! used.empty()) {
    used.erase(i);
    if (! used.count(p[i])) {
        cycles.pb(p[i]);
        i = *used.begin();
    }
    else i = p[i];
}
for (auto st : cycles) {
    vector<int> cur;
    cur.pb(st + 1);
    int i = p[st];
    while (i != st) {
        cur.pb(i + 1);
        i = p[i];
    }
    for (int x : cur) cout << x << ' ';
}
```

7 Maths

7.1 Modular arithmetics

```
int add(int x, int y) {
        x += y;
        while (x >= M) x -= M;
        while (x < 0) x += M;
        return x;
}
int mul(int x, int y) {
    return (x * 1ll * y) % M;
}
int mod_pow(int base, int n) {
        int ans = 1;
        while (n != 0) {
                if (n & 1) ans = mul(ans,
                ↪  base);
                n >>= 1;
                base = mul(base, base);
        }
        return ans;
}
int gcd_(li x, li y) {
        while (x != 0 && y != 0) {
                if (x > y) swap(x, y);
                y = y % x;
        }
        return x + y;
}
int exgcd(int a, int b, li& x1, li& y1) {
        if (b == 0) {
                x1 = 1;
                y1 = 0;
                return a;
        }
        li x, y;
        int d = exgcd(b, a % b, x, y);
        x1 = y;
```

```
        y1 = x - (a / b) * y;
        return d;
}
void solve_diofantine() {
        int a, b, c;
        cin >> a >> b >> c;
        // ax + by = c
        if (a == 0 && b == 0) {
                if (c == 0) {
                        cout << "YES" << '\n';
                        cout << 1 << ' ' << 1
                        ↪  << '\n';
                }
                else cout << "NO" << '\n';
                return;
        }
        li x = 0, y = 0;
        int d = exgcd(a, b, x, y);
        if (c % d != 0) cout << "NO" << '\n';
        else {
                cout << "YES" << '\n';
                int m = c / d;
                cout << x * m << ' ' << y * m
                ↪  << '\n';
        }
}
```

7.2 CRT

```
def CRT(a, m):
    # returns x such that x % m[i] == a[i]
    n = len(a)
    M = 1
    for i in range(n):
        M *= m[i]
    z = [M // m[i] for i in range(n)]
    y = [z[i] * inv(z[i], m[i]) % M for i in
    ↪  range(n)]
    x = 0
    for i in range(n):
        x = (x + y[i] * a[i]) % M
    return x
```

7.3 The ultimate geometry template

```
from functools import total_ordering
from math import atan2, pi
EPS = 10 ** (-9)

@total_ordering
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y ==
        ↪  other.y

    def __lt__(self, other):
        return self.x < other.x or self.x ==
        ↪  other.x and self.y < other.y

    def __repr__(self):
        return f'{self.x} {self.y}'

class Line:
    def __init__(self, a, b):
        self.a = a.y - b.y
        self.b = b.x - a.x
        self.c = -self.a * a.x - self.b * a.y

    def belongs(self, point):
        return abs(self.side(point)) < EPS

    def side(self, point):
        return point.x * self.a + point.y *
        ↪  self.b + self.c

    def dist(self, point):
        return abs(self.a * point.x + self.b *
        ↪  point.y + self.c) / ((self.a ** 2
        ↪  + self.b ** 2) ** 0.5)

class LineCoeff(Line):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

@total_ordering
class Vector:
    def __init__(self, a, b):
        self.x = b.x - a.x
        self.y = b.y - a.y
        self.a = a
        self.b = b

    def __mul__(self, other):
        return self.x * other.y - self.y *
        ↪  other.x

    def __pow__(self, other):
        return self.x * other.x + self.y *
        ↪  other.y

    def __lt__(self, other):
        return self * other < 0

    def __eq__(self, other):
        return self * other == 0

    def __str__(self):
        return f'{self.x} {self.y}'
```

```
    def __repr__(self):
        return f'{self.x} {self.y}'

    def polar(self):
        ans = atan2(self.y, self.x)
        if ans < 0:
            ans += 2 * pi
        return ans

    def length(self):
        return (self.x ** 2 + self.y ** 2) **
        ↪ 0.5

class Segment:
    def __init__(self, a, b):
        self.line = Line(a, b)
        self.a = a
        self.b = b

    def belongs(self, point):
        if not self.line.belongs(point):
            return False
        return (min(self.a.x, self.b.x) - EPS
        ↪ <= point.x <= max(self.a.x,
        ↪ self.b.x) + EPS and
                min(self.a.y, self.b.y) - EPS
                ↪ <= point.y <=
                ↪ max(self.a.y, self.b.y) +
                ↪ EPS)

    def dist(self, point):
        if Vector(self.a, self.b) **
        ↪ Vector(self.a, point) < 0:
            return dist(point, self.a)
        if Vector(self.b, self.a) **
        ↪ Vector(self.b, point) < 0:
            return dist(point, self.b)
        return self.line.dist(point)

    def seg_dist(self, other):
        if (self.line.side(other.a) *
        ↪ self.line.side(other.b) < 0 and
                other.line.side(self.a) *
                ↪ other.line.side(self.b) <
                ↪ 0):
            return 0
        return min(self.dist(other.a),
        ↪ self.dist(other.b),
                   other.dist(self.a),
                   ↪ other.dist(self.b))

    def intersects(self, line):
        point = intersects(self.line, line)
        if point and Vector(point, self.a) **
        ↪ Vector(point, self.b) <= 0:
            return point
        return False

class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    def __repr__(self):
        return f'{self.x} {self.y} {self.r}'

    def belongs(self, p):
        return (self.x - p.x) ** 2 + (self.y -
        ↪ p.y) ** 2 <= self.r ** 2 + EPS

    def inside(self, circle):
        return (self.x - circle.x) ** 2 +
        ↪ (self.y - circle.y) ** 2 <=
        ↪ (self.r - circle.r) ** 2 + EPS

def det(a, b, c, d):
    return a * d - b * c

def intersects(line1, line2):
    D = det(line1.a, line1.b, line2.a,
    ↪ line2.b)
    if D == 0:
        return False
    D1 = det(line1.c, line1.b, line2.c,
    ↪ line2.b)
    D2 = det(line1.a, line1.c, line2.a,
    ↪ line2.c)
    return Point(-D1 / D, -D2 / D)

def dist(point1, point2):
    return ((point2.x - point1.x) ** 2 +
    ↪ (point2.y - point1.y) ** 2) ** 0.5

def cw(a, b, c):
    return Vector(b, a) * Vector(b, c) < 0

def ccw(a, b, c):
    return Vector(b, a) * Vector(b, c) > 0

def convex_hull(points):
    p = sorted(points)
    n = len(points)
    up = list()
    down = list()
    up.append(p[0])
    down.append(p[0])
    for i in range(1, n):
        if i == n - 1 or not cw(p[n - 1],
        ↪ p[0], p[i]):
            while len(up) >= 2 and not
            ↪ ccw(up[len(up) - 2],
            ↪ up[len(up) - 1], p[i]):
                up.pop()
```

```
            up.append(p[i])
        if i == n - 1 or cw(p[n - 1], p[0],
        ↪ p[i]):
            while len(down) >= 2 and not
            ↪ cw(down[len(down) - 2],
            ↪ down[len(down) - 1], p[i]):
                down.pop()
            down.append(p[i])
    return down + up[1:-1][::-1]

def intersects_c_l(circle, line, dx, dy):
    sq = line.a ** 2 + line.b ** 2
    if sq == 0:
        return []
    x0 = -line.a * line.c / sq
    y0 = -line.b * line.c / sq
    d0 = abs(line.c) / (sq ** 0.5)
    if d0 > circle.r + EPS:
        return []
    if d0 > circle.r - EPS:
        return [Point(x0 + dx, y0 + dy)]
    dsq = circle.r ** 2 - (line.c ** 2) / sq
    mult = (dsq / sq) ** 0.5
    P = Point(x0 + line.b * mult + dx, y0 -
    ↪ line.a * mult + dy)
    Q = Point(x0 - line.b * mult + dx, y0 +
    ↪ line.a * mult + dy)
    return [P, Q]

def intersects_c_c(circle1, circle2):
    circle2.x -= circle1.x
    circle2.y -= circle1.y
    line = LineCoeff(2 * circle2.x, 2 *
    ↪ circle2.y,
                     circle2.r ** 2 -
                     ↪ circle1.r ** 2 -
                     ↪ circle2.x ** 2 -
                     ↪ circle2.y ** 2)
    ans = intersects_c_l(Circle(0, 0,
    ↪ circle1.r), line, circle1.x,
    ↪ circle1.y)
    circle2.x += circle1.x
    circle2.y += circle1.y
    return ans
```

7.4 FFT

```
from math import pi, e, log

def FFT(P, inverse=0):
    n = len(P)
    if n == 1:
        return P
    p_e = FFT(P[::2], inverse)
    p_o = FFT(P[1::2], inverse)
    w = e ** ((-1) ** (inverse + 1) * 2j * pi
    ↪ / n)
    y = [0] * n
    mod = 1
    for j in range(n // 2):
        y[j] = (p_e[j] + p_o[j] * mod) /
        ↪ (inverse + 1)
        y[j + n // 2] = (p_e[j] - p_o[j] *
        ↪ mod) / (inverse + 1)
        mod *= w
    return y
```

7.5 Eratosphenes sieve

```
bool sieve[N];
for (int i = 0; i < N; i++) sieve[i] = 0;
for (int i = 2; i * i < N; i++) {
    if (!sieve[i]) {
        for (int j = i * i; j < N; j += i) {
            if (!sieve[j]) sieve[j] = 1;
        }
    }
}
```

8 Other algorithms

8.1 XOR hashing

```
mt19937 rnd(time(NULL));
set<int> used;
used.insert(0);
for (int i = 0; i < int(b.size()); i++) {
    int x = rnd();
    while (used.count(x)) x = rnd();
    b[i] = x;
    used.insert(x);
}
for (int i = 0; i < n; i++) a[i] = b[a[i]];
vector<int> pr(n + 1, 0);
for (int i = 0; i < n; i++) pr[i + 1] = pr[i]
↪ ^ a[i];
// pr[l - 1] ^ pr[r] == 0 -> все элементы на
↪ отрезке [l; r] встречаются чётное
↪ количество раз
```

8.2 Coordinate compression

```
vector<int> b = a;
sort(b.begin(), b.end());
b.erase(unique(b.begin(), b.end()), b.end());
for (int i = 0; i < n; i++) {
    a[i] = lower_bound(b.begin(), b.end(),
    ↪ a[i]) - b.begin();
}
```

8.3 Real-number binary search

```
ld l = 0;
ld r = INF;
```

```
for (int i = 0; i < 60; i++) {
    ld m = (l + r) / 2;
    if (f(m) < ans)
        l = m;
    else
        r = m;
}
cout << l << '\n';
```

8.4 Sprague-Grundy

```
int f[N];
int mex(vector<int> &a) {
    int n = a.size();
    vector<bool> used(n + 1);
    for (int i = 0; i < n; i++) {
        if (a[i] <= n) used[a[i]] = true;
    }
    for (int i = 0; i <= n; i++) {
        if (! used[i]) return i;
    }
    return -1;
}
int gr(int x) {
    if (f[x] != -1) return f[x];
    vector<int> trans;
    for (int i = 1; i <= x; i++) {
        trans.push_back(x - i);
    }
    return (f[x] = mex(trans));
}
```

Функция Гранди для комбинации игр = XOR-сумма функций Гранди для каждой из игр.
XOR-сумма = 0 $\Rightarrow$ выигрывает 2ой игрок
XOR-сумма $\neq 0$ $\Rightarrow$ выигрывает 1ый игрок.

9 Important formulas

Линейность матожидания: $M(x_1 + x_2 + \cdots + x_n) = M(x_1) + M(x_2) + \cdots + M(x_n)$

9.1 Sequence sums

Сумма геометрической прогрессии: $S_n = \frac{b_1(1-q^n)}{1-q}$
Сумма бесконечно убывающей геометрической прогрессии: $S = \frac{b_1}{1-q}$
Сумма арифметической прогрессии: $S_n = \frac{a_1+a_n}{2} \cdot n = \frac{2a_1+d(n-1)}{2} \cdot n$
$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n(n+1)(2n+1)}{6}$
$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$

9.2 Calculus

Площадь криволинейного сектора:
$\mu(F) = \frac{1}{2}\int_\alpha^\beta r^2(\varphi)d\varphi$

Интегралы:

1. $\int \sqrt{a^2 - x^2}dx = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2}arcsin\frac{x}{a} + C$
2. $\int \sqrt{x^2 \pm a^2}dx = \frac{x}{2}\sqrt{x^2 \pm a^2} \pm \frac{a^2}{2}ln|x + \sqrt{x^2 \pm a^2}| + C$
3. $\int \frac{dx}{a^2+x^2} = \frac{1}{a}arctg\frac{x}{a} + C(a \neq 0)$
4. $\int \frac{dx}{a^2-x^2} = \frac{1}{2a}ln|\frac{a+x}{a-x}| + C(a \neq 0)$
5. $\int \frac{xdx}{a^2 \pm x^2} = \pm\frac{1}{2}ln|a^2 \pm x^2| + C$
6. $\int \frac{dx}{\sqrt{a^2-x^2}} = arcsin\frac{x}{a} + C(a \neq 0)$
7. $\int \frac{dx}{\sqrt{x^2 \pm a^2}} = ln|x + \sqrt{x^2 \pm a^2}| + C$
8. $\int \frac{xdx}{\sqrt{a^2 \pm x^2}} = \pm\sqrt{a^2 \pm x^2} + C$

9.3 Combinatorics

Размещения:

- Без повторений: $A_n^k = \frac{n!}{(n-k)!}$
- С повторениями: $\overline{A}_n^k = n^k$

Сочетания:

- Без повторений: $C_n^k = \frac{n!}{k!(n-k)!}$
- С повторениями: $\overline{C}_n^k = \frac{(n+k-1)!}{k!(n-1)!}$

Перестановки:

- Без повторений: $P_n = n!$
- С повторениями: $P_n(n_1, n_2, \ldots, n_k) = \frac{n!}{n_1!n_2!\ldots n_k!}$

1. $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$
2. $C_n^k = C_n^{n-k}$