# Team Notebook

Metropolitan University

December 6, 2024

## Contents

# 1 Array

## 1.1 Kadane in circular array

```
int maxSubarraySumCircular(vector<int> &nums)
{
    int mxsf = -30000, mnsf = 30000, sum = 0, csum = 0;
    for (int x : nums)
    {
        csum += x; sum += x;
        mxsf = max(mxsf, csum);
        csum = max(csum, 0);
    }
    csum = 0;
    for (int x : nums)
    {
        csum += x;
        mnsf = min(mnsf, csum);
        if (csum > 0) csum = 0;
    }
    if (mnsf == sum) mnsf = 0;
    return max(mxsf, sum - mnsf);
}
```

## 1.2 Kadane

```
int maxSubArraySum(int a[], int size)
{
    int max_so_far = INT_MIN, max_ending_here = 0;

    for (int i = 0; i < size; i++) {
        max_ending_here = max_ending_here + a[i];
        if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;

        if (max_ending_here < 0)
            max_ending_here = 0;
    }
    return max_so_far;
}
```

# 2 Bitmasks

## 2.1 Change Bit for a given position

```
int changeBit(int n, int x){
    n ^= 1<<(x-1); // flip xth bit
    n |= 1<<(x-1); // on xth bit
    if(n&(1<<(x-1))) n ^= 1<<(x-1); // off xth bit

    return n;
}
```

## 2.2 Know every Bit of an Integer

```
void keepBit(vector<int> &b, int n, int h = 1)
{
    int i = 25;
    for (i = 25; i >= 0; i--)
    {
        if (n >= (2 << i))
        {
            n -= (2 << i);
            b[i + 1] = b[i + 1] + h;
        }
    }
    if (n)
        b[i + 1] = b[i + 1] + h;
}

// for array b
//for 12 it keeps 0 0 1 1 0 0 0 0 0 0 0 0 0 0
//for 13 it keeps 1 0 1 1 0 0 0 0 0 0 0 0 0 0
```

# 3 Boilerplate

```
#define pi acos(-1)
#define MOD 1000000007
#define dpos(n) fixed << setprecision(n)
#define clear1(a) memset(a, -1, sizeof(a))
#define clear0(a) memset(a, 0, sizeof(a))
#define unik(a) unique(a.begin(), a.end()) - a.begin()
#define iotav(s, x) iota(s.begin(), s.end(), x)
#define lowerbound(v,x) lower_bound(v.begin(), v.end(), x)-v
     .begin()
#define upperbound(v,x) upper_bound(v.begin(), v.end(), x)-v
     .begin()
#define print(x) cout << #x << " = " << x << endl
long long dx[] = {1, -1, 0, 0, 1, 1, -1, -1};
long long dy[] = {0, 0, 1, -1, 1, -1, 1, -1};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(0),cout.tie(0);
    #ifndef ONLINE_JUDGE
     freopen("input.txt", "r", stdin);
     freopen("output.txt", "w", stdout);
    #endif
}
```

# 4 Combitatorics

## 4.1 Formula

```
Summation of squares of n natural numbers: (n*(n+1)*(2n+1))
     /6

C(n,r): n! / (r! * (n-r)!)
C(n,r): (n*(n-1)*..*(n-r+1)) / r!

P(n,k): n! / (n-k)!
-> nCk = nCn-k
-> Ways to go from (0,0) to (r,c):(r+c)Cr or (r+c)Cc
-> Ways to go from (0,0,0) to (x,y,z): (x+y+z)Cx * (y+z)Cy

-> a1+a2+...+an = k , ai>=0: C(k+n-1,n-1)

-> Catalan Numbers: C(n) = (2n)! / ((n+1)! * r!)
```

# 5 Data Structure

## 5.1 BIT with Lazy

```
template <class T>
struct Fenwick {
    int n;
    vector<T> t;
    Fenwick() {}
    Fenwick(int _n) {
        n = _n;
        t.assign(n + 1, 0); // 1-indexed, so size is n + 1
    }
    void upd(int i, T val) {
        if (i <= 0) return;
        for (; i <= n; i += (i & -i))
            t[i] += val;
    }
    void upd(int l, int r, T val) {
        upd(l, val);
        upd(r + 1, -val);
    }
    T query(int i) {
        T sum = 0;
        for (; i >= 1; i -= (i & -i))
            sum += t[i];
        return sum;
    }
    T query(int l, int r) {
        return query(r) - query(l - 1);
    }
};

int main() {
    long long t = 1;
    while (t--) {
        long long n, q;
        cin >> n >> q;
        vector<long long> a(n + 1); // 1-indexed
        Fenwick<long long> tree(n);
        for (long long i = 1; i <= n; i++) {
            cin >> a[i];
        }
        for (long long i = 1; i <= n; i++) {
            tree.upd(i, i, a[i]);
        }
        while (q--) {
            int type;
            cin >> type;
            if (type == 1) {
                long long l, r, val;
                cin >> l >> r >> val;
                tree.upd(l, r, val);
            } else {
                long long l;
                cin >> l;
                cout << tree.query(l) << endl;
            }
        }
    }
}
```

## 5.2 BIT

```
void update(int i, int val, int n, int *tree)
{
    while (i <= n)
    {
        tree[i] += val;
        i += (i & -i);
    }
}
//sum from 1 to i
int getSum(int i, int *tree)
{
    int sum = 0;
    while (i > 0)
    {
        sum += tree[i];
        i ^= (i & -i);
    }
    return sum;
}
```

## 5.3 LCA

```
const int N = 1e5 + 10;
int tin[N], tout[N];
int up[N][32];
vector<int> adj[N];
int n, lg, timer;
void dfs(int src, int par)
{
    tin[src] = ++timer;
    up[src][0] = par;
    for(int i = 1; i <= lg; i++)
    {
        up[src][i] = up[up[src][i - 1]][i - 1];
    }
    for(auto child : adj[src])
    {
        if(child != par)
            dfs(child, src);
    }
    tout[src] = ++timer;
}
bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v)
{
    if(is_ancestor(u, v)) return u;
    if(is_ancestor(v, u)) return v;
    for(int i = lg; i >= 0; i--)
    {
        if(!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
void pre_process(int root)
{
    timer = 0;
    lg = ceil(log2(n));
    dfs(root, root);
}
```

## 5.4 Maximum Subarray Sum

```
const ll M = 2e5 + 10;
const ll inf = 1e9 + 10;
ll arr[M];
struct Node
{
    ll sum, pref, suf, ans;
} seg[4 * M];
Node ck(Node l, Node r)
{
    Node res;
    if (l.ans == inf)
        return r;
    if (r.ans == inf)
        return l;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, r.pref + l.sum);
    res.suf = max(r.suf, l.suf + r.sum);
    res.ans = max({l.ans, r.ans, l.suf + r.pref});
    return res;
}
Node make(ll val)
{
    Node tmp;
    tmp.sum = val;
    tmp.pref = tmp.suf = tmp.ans = val;
    return tmp;
}
void build(ll node, ll l, ll r)
{
    if (l == r)
    {
        seg[node] = make(arr[l]);
        return;
    }
    ll mid = (l + r) >> 1;
    build(node * 2, l, mid);
    build(node * 2 + 1, mid + 1, r);
    seg[node] = ck(seg[node * 2], seg[node * 2 + 1]);
}
void update(ll node, ll l, ll r, ll idx, ll val)
{
    if (l == r)
    {
        seg[node] = make(val);
        return;
    }
    ll mid = (l + r) >> 1;

    if (idx <= mid)
        update(node * 2, l, mid, idx, val);
    else
        update(node * 2 + 1, mid + 1, r, idx, val);

    seg[node] = ck(seg[node * 2], seg[node * 2 + 1]);
}
Node query(ll node, ll l, ll r, ll L, ll R)
{
    if (l > R || r < L)
        return make(inf);
    if (L <= l && R >= r)
        return seg[node];

    ll mid = (l + r) >> 1;
    return ck(query(node * 2, l, mid, L, R), query(node * 2 + 1, mid + 1, r, L, R));
}
int main()
{
    ll n, q;
    cin >> n;
    for (ll i = 1; i <= n; i++)
        cin >> arr[i];
    build(1, 1, n);
    cin >> q;
    while (q--)
    {
        ll ty, id, val, i, j;
        cin >> ty;
        if (ty == 0)
        {
            cin >> id >> val;
            update(1, 1, n, id, val);
        }
        else
        {
            cin >> i >> j;
            cout << query(1, 1, n, i, j).ans << endl;
        }
    }
}
```

## 5.5 Monotonic Stack

```
void printNGE(int arr[], int n)
{
    stack<int> s;
    s.push(arr[0]);
    for (int i = 1; i < n; i++) {
        if (s.empty()) {
            s.push(arr[i]);
            continue;
        }
        while (s.empty() == false && s.top() < arr[i]) {
            cout << s.top() << " --> " << arr[i] << endl;
            s.pop();
        }
        s.push(arr[i]);
    }
    while (s.empty() == false) {
        cout << s.top() << " --> " << -1 << endl;
        s.pop();
    }
}
```

## 5.6 MOs

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

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

bool compare(const Query &q1, const Query &q2) {
    int block_size = sqrt(q1.l);
    if (q1.l / block_size != q2.l / block_size)
        return q1.l < q2.l;
    return q1.r < q2.r;
}

class MosAlgorithm {
public:
    vector<int> arr, answers, freq;
    vector<Query> queries;
    int current_answer = 0;

    MosAlgorithm(int n, int q) {
        arr.resize(n);
        answers.resize(q);
        freq.resize(n + 1, 0);
    }

    void add(int idx) {
        freq[arr[idx]]++;
        if (freq[arr[idx]] == 1)
            current_answer++;
    }

    void remove(int idx) {
        if (freq[arr[idx]] == 1)
            current_answer--;
        freq[arr[idx]]--;
    }

    vector<int> processQueries(int n) {
        int block_size = sqrt(n);
        sort(queries.begin(), queries.end(), compare);
        int curr_l = 0, curr_r = -1;
        for (const Query &q : queries) {
            int l = q.l, r = q.r;
            while (curr_r < r) add(++curr_r);
            while (curr_r > r) remove(curr_r--);
            while (curr_l < l) remove(curr_l++);
            while (curr_l > l) add(--curr_l);
            answers[q.idx] = current_answer;
        }
        return answers;
    }

    void addQuery(int l, int r, int idx) {
        queries.push_back({l, r, idx});
    }

    void setArray(const vector<int> &input) {
        arr = input;
    }
};

int main() {
    int n, q;
    cin >> n >> q;
    MosAlgorithm mos(n, q);
    vector<int> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];
    mos.setArray(arr);
    for (int i = 0; i < q; i++) {
        int l, r;
        cin >> l >> r;
        l--, r--;
        mos.addQuery(l, r, i);
    }
    vector<int> results = mos.processQueries(n);
    for (int i = 0; i < q; i++) {
        cout << results[i] << endl;
    }
    return 0;
}
```

## 5.7 Ordered Set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, less<
     int>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
     tree_order_statistics_node_update> ordered_set;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    ordered_set s;
    long long cnt = 0;

    for (long long i = 1; i <= n; i++)
    {
        long long x;
        cin >> x;
        s.insert(x);

        long long index = s.order_of_key(x); //index of this
             number in a set
        cout << index << endl;
    }
}
```

## 5.8 Segement Tree

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

void build(long long *tree, long long *a, long long node,
     long long l, long long r)
{
    if (l == r)
    {
        tree[node] = a[l];
        return;
    }

    long long left = 2 * node, right = left + 1;
    long long mid = l + (r-l)/2;

    build(tree, a, left, l, mid);
    build(tree, a, right, mid + 1, r);

    tree[node] = tree[left] + tree[right];
}

long long query(long long *tree, long long *a, long long
     node, long long l, long long r, long long begin, long
     long end)
{
    if (r < begin || end < l)
        return 0;

    if (begin <= l && r <= end)
    {
        return tree[node];
    }

    long long left = 2 * node, right = left + 1;
    long long mid = l + (r-l)/2;

    long long left_value = query(tree, a, left, l, mid, begin
         , end);
    long long right_value = query(tree, a, right, mid + 1, r,
          begin, end);

    return left_value + right_value;
}

void update(long long *tree, long long *a, long long node,
     long long l, long long r, long long index, long long
     value)
{
    if (l == r)
    {
        a[l] = value;
        tree[node] = a[l];
        return;
    }

    long long left = 2 * node, right = left + 1;
    long long mid = l + (r-l)/2;

    if (index <= mid)
        update(tree, a, left, l, mid, index, value);
    else
        update(tree, a, right, mid + 1, r, index, value);

    tree[node] = tree[left] + tree[right];
}

int main()
{
    long long n;
    cin >> n;
    long long q;
    cin >> q;

    long long a[n + 2], tree[4 * n];
    for (long long i = 1; i <= n; i++)
        cin >> a[i];

    build(tree, a, 1, 1, n);

    while (q--)
    {
        long long tt, x, y;
        cin>>tt>>x>>y;
        if(tt==1){
            update(tree, a, 1, 1, n, x, y);
            continue;
        }
        long long desire_value = query(tree, a, 1, 1, n, x, y
            );
        cout << desire_value << endl;
    }
}
```

## 5.9 Segment Tree with Lazy

```
template <class T>
struct SegmentTree
{
    vector<T> tree, lazy, a;
    int n;

    SegmentTree(int size)
    {
        n = size;
        tree.resize(4 * n);
        lazy.resize(4 * n);
        a.resize(n + 1);
    }

    void build_tree(int node, int l, int r)
    {
        lazy[node] = 0;
        if (l == r)
        {
            tree[node] = a[l];
            return;
        }
        int left = 2 * node, right = left + 1, mid = l + (r -
             l) / 2;
        build_tree(left, l, mid);
        build_tree(right, mid + 1, r);
        tree[node] = tree[left] + tree[right];
    }

    void propagate(int node, int l, int r)
    {
        if (lazy[node])
        {
            tree[node] += (r - l + 1) * lazy[node];
            if (l != r)
            {
                lazy[2 * node] += lazy[node];
                lazy[2 * node + 1] += lazy[node];
            }
            lazy[node] = 0;
        }
    }

    T query(int node, int l, int r, int b, int e)
    {
        propagate(node, l, r);
        if (r < b || e < l)
            return 0;
        if (b <= l && r <= e)
            return tree[node];
        int mid = l + (r - l) / 2;
        return query(2 * node, l, mid, b, e) + query(2 * node
             + 1, mid + 1, r, b, e);
    }

    void update(int node, int l, int r, int b, int e, T val)
    {
        propagate(node, l, r);
        if (r < b || e < l)
            return;
        if (b <= l && r <= e)
        {
            tree[node] += (r - l + 1) * val;
            if (l != r)
            {
                lazy[2 * node] += val;
                lazy[2 * node + 1] += val;
            }
            return;
        }
        int mid = l + (r - l) / 2;
        update(2 * node, l, mid, b, e, val);
        update(2 * node + 1, mid + 1, r, b, e, val);
        tree[node] = tree[2 * node] + tree[2 * node + 1];
    }

    void update_range(int l, int r, T val)
    {
        update(1, 1, n, l, r, val);
    }

    T range_query(int l)
    {
        return query(1, 1, n, l, l);
    }

    T range_query(int l, int r)
    {
        return query(1, 1, n, l, r);
    }

    void set_array(const vector<T>& input)
    {
        for (int i = 1; i <= n; i++)
            a[i] = input[i - 1];
        build_tree(1, 1, n);
    }
};

int main()
{
    int n, q;
    cin >> n >> q;
    SegmentTree<long long> st(n);
    vector<long long> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];
    st.set_array(arr);
    while (q--)
    {
        int type;
        cin >> type;
        if (type == 1)
        {
            int l, r;
            long long val;
            cin >> l >> r >> val;
            st.update_range(l, r, val);
        }
        else
        {
            int l;
            cin >> l;
            cout << st.range_query(l) << endl;
        }
    }
}
```

## 5.10 Sliding Window Maximum

```
vector<int> maxSlidingWindow(vector<int> &nums, int k)
{
    int n = nums.size();
    deque<int> dq;
    vector<int> ans;
    for (int i = 0; i < n; i++)
    {
        while (dq.size() > 0 && nums[dq.back()] < nums[i])
            dq.pop_back();
        dq.push_back(i);
        if (i >= k - 1)
        {
            ans.push_back(dq.front());
            if (dq.front() < i - k + 2)
                dq.pop_front();
        }
    }
    for (int i = 0; i < ans.size(); i++)
    {
        ans[i] = nums[ans[i]];
    }
    return ans;
}
```

## 5.11 Sparse Table

```
const int N = 2e5 + 5, K = 20;

struct SparseTable {
    long long a[N], sparseAll[N][K][8];

    long long lcm(long long x, long long y) {
        return (x / __gcd(x, y)) * y;
    }

    void build(int n) {
        for (int i = 1; i <= n; i++)
            for (int j = 0; j < 8; j++)
                sparseAll[i][0][j] = a[i];
        for (int j = 1; j < K; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                sparseAll[i][j][0] = min(sparseAll[i][j - 1][0], sparseAll[i + (1 << (j - 1))][j - 1][0]);
                sparseAll[i][j][1] = max(sparseAll[i][j - 1][1], sparseAll[i + (1 << (j - 1))][j - 1][1]);
                sparseAll[i][j][2] = sparseAll[i][j - 1][2] + sparseAll[i + (1 << (j - 1))][j - 1][2];
                sparseAll[i][j][3] = __gcd(sparseAll[i][j - 1][3], sparseAll[i + (1 << (j - 1))][j - 1][3]);
                sparseAll[i][j][4] = lcm(sparseAll[i][j - 1][4], sparseAll[i + (1 << (j - 1))][j - 1][4]);
                sparseAll[i][j][5] = sparseAll[i][j - 1][5] ^ sparseAll[i + (1 << (j - 1))][j - 1][5];
                sparseAll[i][j][6] = sparseAll[i][j - 1][6] & sparseAll[i + (1 << (j - 1))][j - 1][6];
                sparseAll[i][j][7] = sparseAll[i][j - 1][7] | sparseAll[i + (1 << (j - 1))][j - 1][7];
            }
    }

    long long qMin(int l, int r) {
        int k = __lg(r - l + 1);
        return min(sparseAll[l][k][0], sparseAll[r - (1 << k) + 1][k][0]);
    }

    long long qMax(int l, int r) {
        int k = __lg(r - l + 1);
        return max(sparseAll[l][k][1], sparseAll[r - (1 << k) + 1][k][1]);
    }

    long long qSum(int l, int r) {
        int k = __lg(r - l + 1);
        return sparseAll[l][k][2] + sparseAll[r - (1 << k) + 1][k][2];
    }

    long long qGcd(int l, int r) {
        int k = __lg(r - l + 1);
        return __gcd(sparseAll[l][k][3], sparseAll[r - (1 << k) + 1][k][3]);
    }

    long long qLcm(int l, int r) {
        int k = __lg(r - l + 1);
        return lcm(sparseAll[l][k][4], sparseAll[r - (1 << k) + 1][k][4]);
    }

    long long qXor(int l, int r) {
        int k = __lg(r - l + 1);
        return sparseAll[l][k][5] ^ sparseAll[r - (1 << k) + 1][k][5];
    }

    long long qAnd(int l, int r) {
        int k = __lg(r - l + 1);
        return sparseAll[l][k][6] & sparseAll[r - (1 << k) + 1][k][6];
    }

    long long qOr(int l, int r) {
        int k = __lg(r - l + 1);
        return sparseAll[l][k][7] | sparseAll[r - (1 << k) + 1][k][7];
    }
};

int main() {
    int n, q;
    cin >> n >> q;
    SparseTable st;
    for (int i = 1; i <= n; i++) cin >> st.a[i];
    st.build(n);
    while (q--) {
        int l, r, type;
        cin >> l >> r >> type;
        //print what u want
    }
}
```

## 5.12 Sqrt Decomposition

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

struct SqrtDecomposition
{
    int n, block_size;
    vector<int> arr, blocks;

    SqrtDecomposition(const vector<int> &input)
    {
        n = input.size();
        block_size = sqrt(n); // Calculate the size of each block
        arr = input;
        blocks.resize((n + block_size - 1) / block_size, 0); // Adjust for the number of blocks

        // Build the initial blocks by summing over ranges
        for (int i = 0; i < n; i++)
        {
            blocks[i / block_size] += arr[i];
        }
    }

    // Query the sum in range [l, r]
    int query(int l, int r)
    {
        l--, r--; // Adjust for 0 indexing
        int sum = 0;

        int start_block = l / block_size;
        int end_block = r / block_size;

        if (start_block == end_block)
        {
            // When l and r are in the same block, sum directly
            for (int i = l; i <= r; i++) sum += arr[i];
        }
        else
        {
            // Sum the partial block from l to the end of its block
            for (int i = l; i < (start_block + 1) * block_size; i++) sum += arr[i];
            // Sum the full blocks in between
            for (int i = start_block + 1; i < end_block; i++)
                  sum += blocks[i];

            // Sum the partial block from the start of r's
                 block to r
            for (int i = end_block * block_size; i <= r; i++)
                  sum += arr[i];
        }

        return sum;
    }

    // Update the value at index idx to val
    void update(int idx, int val)
    {
        idx--; //Adjust for 0 indexing
        int block_idx = idx / block_size;

        // Adjust the block's sum by the difference in values
        blocks[block_idx] += (val - arr[idx]);

        // Update the original array
        arr[idx] = val;
    }
};

int main()
{
    int n, q;
    cin >> n >> q;

    vector<int> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];

    // Initialize sqrt decomposition
    SqrtDecomposition sqrtDecomp(a);

    while (q--)
    {
        int type;
        cin >> type;

        if (type == 1)
        {
            int l, r;
            cin >> l >> r;
            cout << sqrtDecomp.query(l, r) << endl; // 1
                 indexing
        }
        else
        {
            int x, val;
            cin >> x >> val;

            sqrtDecomp.update(x, val); //indexing
        }
    }
    return 0;
}
```

# 6 DP

## 6.1 Coin Changing

```
long long minNumberOfCoin(vector<long long> &v, long long
     val, long long n, vector<long long> &dp)
{
    cnt++;
    if (val == 0)
        return 0;
    if (val < 0)
        return LLONG_MAX;
    if (dp[val] != LLONG_MAX)
        return dp[val];

    long long ans = LLONG_MAX;

    for (long long i = 0; i < n; i++)
    {
        long long res = minNumberOfCoin(v, val - v[i], n, dp)
             ;
        if (res != LLONG_MAX)
            ans = min(ans, res + 1);
    }

    return dp[val] = ans;
}


int numberOfWays(int coins[], int n, int sum)
{
 int dp[sum + 1];
 memset(dp, 0, sizeof(dp));
 dp[0] = 1;
 for (int i = 0; i < n; i++)
  for (int j = coins[i]; j <= sum; j++)
   dp[j] += dp[j - coins[i]];
 return dp[sum];
}
```

## 6.2 Digit DP

```
vector<int> num;
int sz, k, n, m;
int dp[10][2][100][100];
int digitdp(int pos, int issmall, int sum, int val)
{
    if (pos == sz)
    {
        if (!sum && !val)
            return 1;
        return 0;
    }
    if (dp[pos][issmall][sum][val] != -1)
        return dp[pos][issmall][sum][val];
    int lim;
    if (issmall == 0)
        lim = num[pos];
    else
        lim = 9;
    int ans = 0;
    for (int digit = 0; digit <= lim; digit++)
    {
        int cur_issmall = issmall;
        if (issmall == 0 && digit < lim)
            cur_issmall = 1;
        int cur_sum = (sum + digit) % k;
        int cur_val = ((val * 10) + digit) % k;
        ans += digitdp(pos + 1, cur_issmall, cur_sum, cur_val
             );
    }
    return dp[pos][issmall][sum][val] = ans;
}

int solve(int n)
{
    num.clear();
    while (n > 0)
    {
        num.push_back(n % 10);
        n /= 10;
    }
    sz = num.size();
    reverse(num.begin(), num.end());
    memset(dp, -1, sizeof(dp));
    return digitdp(0, 0, 0, 0);
}
```

## 6.3 Knapsack

```
long long knapSack(long long w, long long i, long long *
     marks, long long *cap, vector<vector<long long>> &dp)
{
    if (i < 0)
        return 0;
    if (dp[i][w] != -1)
        return dp[i][w];

    if (cap[i] > w) dp[i][w] = knapSack(w, i - 1, marks, cap,
         dp);
    else dp[i][w] = max(marks[i] + knapSack(w - cap[i], i -
        1, marks, cap, dp), knapSack(w, i - 1, marks, cap,
        dp));

    return dp[i][w];
}
```

## 6.4 LIS

```
int LIS(vector<int>&v){
    int l = v.size(), dp[l];
    for(int i=0; i<l;i++){
        dp[i]=1;
        for(int j=0; j<i; j++){
            if(v[j]<v[i] and dp[i]<dp[j]+1) dp[i] = dp[j] +
                 1;
        }
    }
    return *max_element(dp, dp+l);
}
```

## 6.5 LNDS

```
int lnds[MAX];
int LNDS(int n) {
lnds[1]=arr[1]; //1 base index
int len = 1 ;
for(int i = 2; i<=n;i++) {
if(arr[i]>=lnds[len])
lnds [++ len] = arr [i];
else{
int j=upper_bound(lnds+1,lnds+len+ 1,arr[i])-lnds;
lnds [j] = arr [i]; } }
return len; }
```

## 6.6 SOS DP

```
for(int i = 0; i < (1 << N); ++i) {
F[i] = A[i]; }
for(int i = 0; i < N; ++i) {
for(int mask=0;mask<(1<<N); ++mask) {
if(mask & (1 << i)) {
F[mask] += F[mask ^ (1 << i)]; } }}
```

# 7 DSU

```
struct DSU
{
 int connected;
 vector<int> par, sz;
 void init(int n)
 {
  par = sz = vector<int> (n + 1, 0);
  for(int i = 1; i <= n; i++)
   par[i] = i, sz[i] = 1;
  connected = n;
 }
 int getPar(int u)
 {
  while(u != par[u])
  {
   par[u] = par[par[u]];
   u = par[u];
  }
  return u;
 }
 int getSize(int u)
 {
  return sz[getPar(u)];
 }
 void unite(int u, int v)
 {
  int par1 = getPar(u), par2 = getPar(v);

  if(par1 == par2)
   return;

  connected--;
  if(sz[par1] > sz[par2])
   swap(par1, par2);

  sz[par2] += sz[par1];
  sz[par1] = 0;
  par[par1] = par[par2];
 }
};
```

# 8 Formulas

## 8.1 Formula

```
Others:
Decider for a point located left or right of a line:
d=(x3-x2)*(y2-y1)-(y3-y2)*(x2-x1)
Number of digits: log10(n)+1
Depth of road water: (s^2-h^2)/2h

//sum of series n/1+n/2+n/3+....n/n
ll root=sqrt(n);
for(int i=1; i<=root; i++)
sum+=n/i;
sum=(2*sum)-(root*root);


count the numbers that are
divisible by given number in a
certain range:a=2,b=3,c=7;
low=(a+b-1)/a;
high=c/a;
total=high-low+1;

Euler Constant: 0 .5772156649

#Number of squares in a n*n grid:
S=(n*(n+ 1)*(2*n + 1))/6;

#Number of rectangle in a n*n grid:
R=(n+1)*n/2*(n+1)*n/2 - S;

#Total number of rectangle and square in a n*n grid:
ans=[(n^2 + n)^2]/4
```

```
#Number of squares in a n*m grid
exp:6*4
S=6*4+5*3+4*2+3*1=50

#Number of rectangles in n*m grid
R=m(m+1)n(n+1)/4

#Number of cubes in a n*n*n grid
formula:n^k-(n-2)^k
C=n*(n+1)/2*n*(n+1)/2;

#Number of boxes in a n*n*n grid:
B=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2-C;

#Number of hypercube in a n^4grid:
start a loop from 1 to <=n;
HC=0;
for(i=1;i<=n;i++)
HC+=i*i*i*i;

#Number of hyper box in a n^4 grid:
HB=(n+1)*n/2*(n+1)*n/2*(n+1)*n/2*(n+1)*n/2 - HC;
```

## 8.2 Physics

```
Physics Formuas
Projectile motion
x =  utcos
y =  utsin    (1/2) gt*t
y = x  tan     g*x*x/( 2u*u*cos* cos )
T = 2u  sin /g
R = u*u* sin2 /g
H = u*u* sin * sin /2g
```

## 8.3 Trigonometry

```
//Trigonometry
 sin2 =2  sincos
 cos2 = cos * cos - sin * sin
 sin3 =3 sin -4* sin * sin * sin
 cos3 =4* cos * cos * cos -3 cos

//For triangle:
a=bcosC+ccosB
b=acosC+ccosA
c=bcosA+acosB
sin(A+ B )=sinAcosB +   cosAsinB
cos(A+ B )=cosAcosB   + sinAsinB
```

# 9 Geometry

## 9.1 Circle

```
Distance: sqrt((x2-x1)^2 + (y2-y1)^2)

Check if 3 points are in same line:
x1*(y2-y3)-x2(y1-y3)+x3(y1-y2) = 0

Find a circle that covers 2 given:
x3 = (x1+x2)/2 , y3 = (y1+y2)/2
r = dist(x1,y1,x2,y2)

Lattice Points:1+gcd(|x1-x2|,|y1-y2|)
Slope formed by 2 points:(y2-y1)/(x2-x1)
Area of sector of circle:  r^2*
Arc Length: r*theta
```

## 9.2 Convex Hull

```
vector<PT> convex_hull(vector<PT> &p) {
  if (p.size() <= 1) return p;
  vector<PT> v = p;
  sort(v.begin(), v.end());
  vector<PT> up, dn;
  for (auto& p : v) {
      while (up.size() > 1 && orientation(up[up.size() - 2],
          up.back(), p) >= 0) {
         up.pop_back();
      }
      while (dn.size() > 1 && orientation(dn[dn.size() - 2],
          dn.back(), p) <= 0) {
         dn.pop_back();
      }
      up.push_back(p);
      dn.push_back(p);
  }
  v = dn;
  if (v.size() > 1) v.pop_back();
  reverse(up.begin(), up.end());
  up.pop_back();
  for (auto& p : up) {
      v.push_back(p);
  }
 if (v.size() == 2 && v[0] == v[1]) v.pop_back();
  return v;
}
bool is_convex(vector<PT> &p) {
  bool s[3]; s[0] = s[1] = s[2] = 0;
  int n = p.size();
  for (int i = 0; i < n; i++) {
      int j = (i + 1) % n;
      int k = (j + 1) % n;
      s[sign(cross(p[j] - p[i], p[k] - p[i])) + 1] = 1;
      if (s[0] && s[2]) return 0;
  }
  return 1;
}
int is_point_in_convex(vector<PT> &p, const PT& x) {
  int n = p.size(); assert(n >= 3);
  int a = orientation(p[0], p[1], x), b = orientation(p[0],
       p[n - 1], x);
  if (a < 0 || b > 0) return 1;
  int l = 1, r = n - 1;
  while (l + 1 < r) {
      int mid = l + r >> 1;
      if (orientation(p[0], p[mid], x) >= 0) l = mid;
      else r = mid;
  }
  int k = orientation(p[l], p[r], x);
  if (k <= 0) return -k;
  if (l == 1 && a == 0) return 0;
  if (r == n - 1 && b == 0) return 0;
  return -1;
}
```

## 9.3 Parallelogram

```
Given 3 points find 4th point:
Dx = Ax + (Cx-Bx)
Dy = Ay + (Cy-By)

Area: | ((Ax*By+Bx*Cy+Cx*Dy+Dx*Ay)
-(Ay*Bx + By*Cx + Cx*Dx + Dy*Ax))|
```

## 9.4 Polygon

```
The sum of the interior angles: (2n  4)   90
*Area of the largest square inside
a pentagon->
```

```
s*(sin(108)/(sin(18)+sin(36)))

Area:(s^2*n)/4*tan(180/4)
Area:(r^2*n*sin(360/n))/2
Area:Apo^2*n*tan(180/n)
Area:  *sq(5*(5+2sq(5)))*s^2
Area:(Apo*s)/2
Area:pr/2
Area:  *n*sin(360/n)*s^2
Area:n*apo^2*tan(180/n)
Area:(n*r*sin(2*(180/n)))/2
Area:(  *n*s^2)/tan(180/n)
Perimeter:5*s
Diagonal:(s*(1+sq(5)))/2
Height:(s*sq(5+2sq(5)))/2

an:2*R*sin(180/n) ..here:an=side of
regular inscribed polygon,R=radius
of circumscribed circle.

Sum of interior angles of a
Convex polygon:180(n-2)

Exterior taken one at each
vertex:360

measurement of Exterior Ang:360/n
Measure Interior An:((n-2)*180)/n
No. Of Dia:(n*(n-3))/2
No. Of Tri:N-2
Side:2*R*sin(180/n)
Apo:R*cos(180/n)
Side:2*apo*tan(180/n)

Area of smallest tri:*apo*(s/2)
=   *Apo^2*tan(180/n)

Intersection points of diagonals of n(odd) sided regular
    polygon = nC4
```

## 9.5 Right Circular Cone

```
Volume:(pie*h/3)*(R^2+R*r+r^2)
Lateral surface Area:pie(r+R)*S
Area of the base:pie*r^2
Lateral area:pie*r*L
Total Surface A:pie*r^2+pie*r*s
Volume:  *pie*r^2*h
s=sq(r^2+h^2)
```

## 9.6 Trapezium

```
Area:(a+b)/(a-b) * sqrt((s-a)(s-b) (s-b-c)(s-b-d))
-> s = (a+b+c+d)/2
-> a = long parallel side
-> b = short parallel side
-> c,d = non-parallel side

Area:h*((b1+b2)/2)
H:sq(b^2-(b^2-d^2+(a-c)^2)/2(a-c))^2)
```

## 9.7 Triangle

```
To form: a+b>c, b+c>a, c+a>b
Check if 3 points form triangle: |(x2-x1)(y3-y1)-(y2-y1)(x3-
    x1)| >0
Perimeter: p = a+b+c
Area:    1) (a*b)/2
         2) (abSinC)/2
         3) sqrt(s(s-a)(s-b)(s-c)); ///s=(p/2)
         4) (sqrt(3)/4)*a*a; ///equi triangle
         5) (b*sqrt(4*a*a-b*b))/4; ///isosceles
Pythagoras: a*a+b*b = c*c
SineRule: a/SinA=b/SinB=c/SinC
CosineRule: CosA = (b*b+c*c-a*a)/2bc
Centre: x=(x1+x2+x3)/3, y=(y1+y2+y3)/3
Median: AD=sqrt((2*b*b+2*c*c-a*a)/4)
Centroid: AG=sqrt(2*b*b-2*c*c-a*a)/3
Inradius: A/s
Circumradius: a/(2*sinA)
r=abc/sq((a+b+c)(a+b-c)(a+c-b)(b+c-a))
```

## 9.8 Truncated Cone

```
z=(H*r2^2)(r1^2-r2^2)
R=(  *r1^2(z+h))/(H+z)
Volume:  *pie*h*(R^2+(R*r2)+r2^2)
Volume of a cylinder: pi*r*r*h
Volume of a triangular prism: 0.5*b*h*H
```

# 10 Graph

## 10.1 Articulation Point

```
void dfs(int v, int p = -1) {
visited[v] = true;
tin[v] = low[v] = timer++;
int children=0;
for (int to : adj[v]) {
if (to == p) continue;
if (visited[to]) {
low[v] = min(low[v], tin[to]); }
else { dfs(to, v);
low[v] = min(low[v], low[to]);
if (low[to] >= tin[v] && p!=-1)
IS_CUTPOINT(v);
++children; } }
if(p == -1 && children > 1)
IS_CUTPOINT(v);//v is the point print it
}

void find_cutpoints() {
timer = 0;
visited.assign(n, false);
tin.assign(n, -1);
low.assign(n, -1);
for (int i = 0; i < n; ++i) {
 if (!visited[i])
 dfs (i); } }
```

## 10.2 Detect Cycle in Tree

```
void dfs(int u, vector<bool> &vis, vector<vector<int>>&p,
    int prev)
{
    if(vis[u]){
        cycle = true;
        return;
    }

    vis[u] = true;

    for (int i = 0; i < p[u].size(); i++)
    {
        int v = p[u][i];
        if(prev!=v){
            dfs(v, vis, p, u);
        }
    }
}
```

## 10.3 DFS on Tree

```
vector<vector<int>> p;
vector<long long> depth;
vector<long long> reverse_depth;
void dfs(int u, int par)
{
    if (p[u].size() == 1 && p[u][0] == par){
        depth[u] = depth[par]+1;
        reverse_depth[u] = 1;
    }
    else
    {
        for (auto v : p[u])
        {
            if (v != par)
            {
                depth[v] = 1 + depth[u];
                dfs(v, u);
                reverse_depth[u] = 1 + reverse_depth[v];
            }
        }
    }
}
void solve(int n)
{
    p.assign(n + 2, vector<int>());
    depth.assign(n + 2, 0);
    reverse_depth.assign(n + 2, 0);
    for (int i = 1; i < n; i++)
    {
        int x, y;
        cin >> x >> y;
        p[x].push_back(y);
        p[y].push_back(x);
    }
    depth[1]=1;
    dfs(1, -1);
    int x = 2;
    cout<<depth[x]<<" "<<reverse_depth[x]<<endl;
}
```

## 10.4 Dijkstra

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

const long long N = 2e5 + 3;
const long long inf = 1e18;

vector<pair<long long, long long>> edges[N];
vector<long long> dist(N, inf);

int main()
{
    long long n, m;
    cin >> n >> m;

    while (m--)
    {
        long long x, y, w;
        cin >> x >> y >> w;
        edges[x].push_back({y, w});
    }

    dist[1] = 0;

    priority_queue<pair<long long, long long>, vector<pair<
         long long, long long>>, greater<pair<long long, long
          long>>> pq;

    pq.push({0, 1});

    while (!pq.empty())
    {
        long long u = pq.top().second, d = pq.top().first;
        pq.pop();
        if (dist[u] < d)
            continue;
        for (auto e : edges[u])
        {
            long long w = e.second, v = e.first;
            if (dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }

    for (long long i = 1; i <= n; i++)
        cout << dist[i] << " ";
}
```

## 10.5 Dikstra K shortest

```
const ll mx = 1e6;
ll n, m, k;
map<pair<ll, ll>, ll> mp;
void dijkstra(ll src, vector<pair<ll, ll>> adj[])
{
    vector<vector<ll>> dist(n + 10, vector<ll>(k,
         LONG_LONG_MAX));
    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
         greater<pair<ll, ll>>> pq;
    pq.push({0, src});
    while (pq.size() > 0)
    {
        pair<ll, ll> cur = pq.top();
        pq.pop();
        ll u = cur.second;
        ll d = cur.first;

        if (d > dist[u][k - 1])
        {
            continue;
        }
        for (ll i = 0; i < adj[u].size(); i++)
        {
            ll v = adj[u][i].first;
            ll d1 = adj[u][i].second;
            if (d + d1 < dist[v][k - 1])
            {
                dist[v][k - 1] = d + d1;
                pq.push({dist[v][k - 1], v});
                srt(dist[v]);
            }
        }
    }
    for (ll i = 0; i < k; i++)
    {
        cout << dist[n][i] << " ";
    }
    cout << endl;
}
```

## 10.6 Distance of Leaf from root

```
#include<bits/stdc++.h>
using namespace std;
//K= root, n=node
void find(vector<long long>a[], long long n, long long k)
{
  queue<long long>q;
  long long vis[n+2]={};
  long long dis[n+2]={},maxx=0ll;
  vis[k]=0;
  dis[k]=1;
  q.push(k);
  while(!q.empty())
  {
    long long x=q.front();
    q.pop();
    long long l=a[x].size();
    for(long long i=0;i<l;i++)
    {
      long long y=a[x][i];
      if(!vis[y])
      {
        q.push(y);
        vis[y]=1;
        dis[y]=dis[x]+1;
        maxx=max(maxx,dis[y]);
      }
    }
  }
  cout<<maxx<<endl;
}
```

## 10.7 Euler Tour

```
void dfs_euler(int src, int par)
{
    euler_path.push_back(src);
    int f = 0;
    for (auto i : graph[src])
    {
        if (i == par)
            continue;
        f = 1;
        dfs_euler(i, src);
    }
    if (f)
        euler_path.push_back(src);
}

void init_euler_path()
{
    dfs_euler(1, 0);
    int idx = 1;
    for (auto i : euler_path)
    {
        if (st[i] == 0)
            st[i] = idx;
        en[i] = idx;
        idx++;
    }
}
```

## 10.8 Floyed Warshal

```
const int INF = 1e9; // Replace with a suitable large value
     for infinity

void floydWarshall(vector<vector<int>>& graph, int nodes) {
    for (int k = 1; k <= nodes; k++) {
        for (int i = 1; i <= nodes; i++) {
            for (int j = 1; j <= nodes; j++) {
                if (graph[i][k] != INF && graph[k][j] != INF)
                     {
                    graph[i][j] = min(graph[i][j], graph[i][k]
                         + graph[k][j]);
                }
            }
        }
    }
}

void addEdge(vector<vector<int>>& dist, int u, int v, int c,
      int N) {
    for (int x = 0; x < N; ++x) {
        for (int y = 0; y < N; ++y) {
            if (dist[x][u] != INF && dist[v][y] != INF) {
                dist[x][y] = min(dist[x][y], dist[x][u] + c +
                     dist[v][y]);
            }
            if (dist[x][v] != INF && dist[u][y] != INF) {
                dist[x][y] = min(dist[x][y], dist[x][v] + c +
                     dist[u][y]);
            }
        }
    }
}

int main() {
    int nodes = 4; // Example: Number of nodes
    vector<vector<int>> graph(nodes + 1, vector<int>(nodes +
         1, INF));

    // Initialize graph distances
    for (int i = 1; i <= nodes; i++) {
        graph[i][i] = 0; // Distance to self is 0
    }

    // Add some edges (example)
    graph[1][2] = 4;
    graph[2][3] = 5;
    graph[3][4] = 6;

    floydWarshall(graph, nodes);

    int N = nodes; // Example: Graph size for ‘dist‘
    vector<vector<int>> dist = graph; // Copy ‘graph‘ to ‘
        dist‘
    int u = 1, v = 4, c = 3; // Example edge
    addEdge(dist, u, v, c, N);

    // Output the result
    for (int i = 1; i <= nodes; i++) {
        for (int j = 1; j <= nodes; j++) {
            cout << (dist[i][j] == INF ? "INF" : to_string(
                dist[i][j])) << " ";
        }
        cout << endl;
    }

    return 0;
}
```

## 10.9 Krushkal;s + DSU

```
int par[N];
int sz[N];
void make_set(int v){
    par[v] = v;
}
int find_set(int v){
    if(v == par[v]) return v;
    return par[v] = find_set(par[v]);
}
void union_sets(int a, int b){
    a = find_set(a);
    b = find_set(b);
    if(a != b){
        if(sz[a] > sz[b]) swap(a, b);
        par[a] = b;
        sz[b] += sz[a];
    }
}
void reset(int n){
    for(int i = 0;i <= n;i++){
        make_set(i);
    }
    for(int i = 0;i <= n;i++){
        sz[i] = 0;
    }
}
int mst(vector<pair<int, pair<int, int>>> &edge){
    int ret = 0;
    for(int i = 0;i < edge.size();i++){
        int w = edge[i].first;
        int v1 = edge[i].second.first;
        int v2 = edge[i].second.second;
        if(find_set(v1) == find_set(v2)) continue;
        ret += w;
        union_sets(v1, v2);
    }
    return ret;
}
```

## 10.10 Longest Path in dag

```
const int N = 1e5;
vector<pair<int, int>> adj[N];
int dp[N];
bool visited[N];

void topologicalSort(int v, stack<int>& Stack) {
  visited[v] = true;
  for (auto& neighbor : adj[v]) {
    int next = neighbor.first;
    if (!visited[next]) {
        topologicalSort(next, Stack);
    }
  }
  Stack.push(v);
}
void longestPath(int src, int n) {
  fill(dp, dp + n, INT_MIN);
  dp[src] = 0;
  stack<int> Stack;
  for (int i = 1; i <= n; i++) {
    if (!visited[i]) {
        topologicalSort(i, Stack);
    }
  }
  while (!Stack.empty()) {
    int u = Stack.top();
    Stack.pop();
    if (dp[u] != INT_MIN) {
      for (auto& neighbor : adj[u]) {
        int v = neighbor.first;
        int weight = neighbor.second;
        if (dp[v] < dp[u] + weight) {
          dp[v] = dp[u] + weight;
        }
      }
    }
  }
  //now dp[i] = longest path
  //INT_MIN mane impossible
}
```

## 10.11 Lowest Common Ancestor

```
const int N = 1e5 + 10;
vector<int> adj[N];
int timer, l;
int tin[N], tout[N];
int up[N][18];
int dist[N];
void dfs(int src, int par){
    tin[src] = ++timer;
    up[src][0] = par;
    for (int i = 1; i <= l; ++i)
        up[src][i] = up[up[src][i-1]][i-1];
    for (int child : adj[src]) {
        if(child != par){
            dist[child] = dist[src] + 1;
            dfs(child, src);
        }
    }
    tout[src] = ++timer;
}
bool is_ancestor(int u, int v){
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}
int lca(int u, int v){
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
void preprocess(int root, int n) {
    timer = 0;
    l = ceil(log2(n));
    dfs(root, root);
}
int getKthAncestor(int node, int k) {
    for(int i = 0; i < 18; i++){
        if(k & (1 << i)){
            node = up[node][i];
            if(node == -1) return -1;
        }
    }
    return node;
}
```

## 10.12 Topological Sort

```
void topologicalSortUtil(
    int v,
    vector<vector<int>> &adj,
    vector<bool> &visited,
    stack<int> &Stack)
{
    visited[v] = true;
    for (int i : adj[v])
    {
        if (!visited[i])
            topologicalSortUtil(i, adj, visited, Stack);
    }
    Stack.push(v);
}

void topologicalSort(vector<vector<int>> &adj, int N)
{
    stack<int> Stack;
    vector<bool> visited(N, false);
    for (int i = 0; i < N; i++)
    {
        if (!visited[i])
            topologicalSortUtil(i, adj, visited, Stack);
    }
    while (!Stack.empty())
    {
        cout << Stack.top() << " ";
        Stack.pop();
    }
}
```

# 11 Math

## 11.1 String and int multiply

```
string multyply(string a, int b)
{
    int carry = 0, l = a.size();

    string ans = "";

    for (int i = l - 1; i >= 0; i--)
    {
        carry = ((a[i] - '0') * b + carry);
        ans += carry % 10 + '0';
        carry /= 10;
    }
    while (carry != 0)
    {
        ans += carry % 10 + '0';
        carry /= 10;
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

## 11.2 Sum of Absolute Diff of All Pairs

```
int
     sum_of_absolute_differences_of_all_pairs_in_a_given_array
     (int a[], int n)
{
    int ans = 0;
    sort(a, a + n);
    for (long long i = 0; i < n; i++)
        ans += a[i] * (2 * i - n + 1);
    return ans;
}
```

# 12 Miscellaneous

## 12.1 repeat

```
long long doRepeat(int cur, int plus, int n, bool clockWise=
     true){
    long long pos;
    if(clockWise) pos = (cur+plus)%n;
    else pos = (cur-plus+n)%n;
    return pos?pos:n;
}
```

# 13 Number Theory

## 13.1 Big Mul

```
long long bigMul(long long n, long long m, long long p)
{
  if(m<=0) return 0;
    long long res = bigMul(n, m/2, p);
    long long ans = (2*res)%p;
    if(m%2) ans = (ans+n)%p;
    return ans;
}
```

## 13.2 Bigmod , Inverse MOD, ncr

```
#define MOD 1000000007
long long bigMod(long long a, long long b)
{
  a %= MOD;
  if (!b)
    return 1;
  long long res = bigMod(a, b / 2);
  long long ans = (res * res) % MOD;
  if (b % 2)
    ans = (ans * a) % MOD;
  return ans;
}

long long inverseMod(long long a)
{
  return bigMod(a, MOD - 2);
}


long long fact[MOD];
void factorial()
{
  fact[0] = 1;
  for (long long i = 1; i < MOD; i++)
    fact[i] = (((i % MOD) * (fact[i - 1] % MOD)) % MOD);
}
long long nCr(long long n, long long r)
{
  return ((fact[n] % MOD) * (inverseMod((fact[r] * fact[n -
       r]) % MOD) % MOD)) % MOD;
}
```

## 13.3 Bigmod with Loop

```
#define MOD 1000000007
long long Big(long long x, long long n)
{
  long long ans=1;
  while(n>0){
    x%=MOD;
    if(n&1) ans*=x;
    ans%=MOD;
    x*=x;
    n>>=1;
  }
  return ans;
}
```

## 13.4 Check if a subset sum exists

```
bitset<MAX>bs;
bool check(int sum) { bs.reset(); bs[0]=1;
for(int i=1;i<=n;i++) bs |= bs << arr[i]; return bs[sum]; }
```

## 13.5 Euler Phi Sieve

```
// P1k1    1(P1    1).P2k2    1(P2    1)....Prkr - 1(Pr - 1)
void computeTotient(int n)
{
    for (int i = 1; i <= n; i++) phi[i] = i;
    for (int j = 2; j <= n; j++)
    {
        if (phi[j] == j)
        {
            phi[j] = j - 1;
            for (int i = 2 * j; i <= n; i += j)
            {
                phi[i] = (phi[i] / j) * (j - 1);
            }
        }
    }
}
```

## 13.6 Factorial

### 13.6.1 Digit Count

```
int factDigitCnt(int n)
{
    if (n <= 1)
        return n;
    double digits = 0;
    for (int i = 2; i <= n; i++)
    {
        digits += log10(i);
    }
    return floor(digits) + 1;
}
```

### 13.6.2 Divisor Count

```
long long factDivisorsCnt(long long n)
{
    long long res = 1;
    for (int i = 0; primes[i] <= n; i++)
    {
        long long exp = 0;
        long long p = primes[i];
        while (p <= n)
        {
            exp += (n / p);
            p *= primes[i];
        }
        res *= (exp + 1);
    }
    return res;
}
```

### 13.6.3 Tailing Zeros

```
int trailingZeroes(int n)
{
    int c = 0, f = 5;
    while (f <= n)
    {
        c += n / f;
        f *= 5;
    }
    return c;
}
```

## 13.7 Generate Number of Divisor 1 to N

```
vector<int>generateNumberOfDivisor(int n= 1e6){
    vector<int>divisor(n+1, 1);
    for(int i=2;i<=n;i++){
        if(divisor[i]==1){
            for(int j=i;j<=n;j+=i){
                int num = j, primeFactor = 0;
                while(num%i==0){
                    num/=i;
                    primeFactor++;
                }
                divisor[j] *= (primeFactor+1);
            }
        }
    }
    return divisor;
}
```

## 13.8 Get Prime

```
#define INF 1000005
int prime[INF];
bool vis[INF];

void getPrime()
{
    int k = 1;
    prime[k++] = 2;
    for (long long i = 3; i <INF; i += 2)
    {
        if (!vis[i] && i % 2)
            prime[k++] = i;
        for (long long j = i * i; j < INF; j += i)
        {
            vis[j] = true;
        }
    }
}
```

## 13.9 Is Prime

```
vector<bool> isPrime(long long n = 1e6)
{
    vector<bool> vis(n + 5);
    vis[1] = true;
    for (long long i = 3; i <= n; i+=2)
    {
        if (!vis[i])
            for (long long j = i * i; j <= n; j += i)
                vis[j] = true;
    }
    return vis;
}
```

## 13.10 MOD Jog Gun

```
#define MOD 1000000007

long long modGunKoro(long long a, long long b){
    return ((a%MOD)*(b%MOD))%MOD;
}

long long modJogKoro(long long a, long long b){
    return ((a%MOD)+(b%MOD))%MOD;
}
```

## 13.11 Number of Divisor

```
#define nn 1000010115.
long long int notprime[nn] = {}, prime[nn];
long long numberofDivisor(long long n)
{
    long long int c = 1, i, j, ans = 1;
    for (i = 3; i * i <= nn; i += 2)
    {

        if (!notprime[i])
        {
            for (j = i * i; j <= nn; j += i)
                notprime[j] = 1;
        }
    }
    prime[c++] = 2;
    for (i = 3; i <= nn; i += 2)
    {
        if (!notprime[i])
        {

            prime[c++] = i;
        }
    }

    for (i = 1; i <= nn && prime[i] * prime[i] <= n; i++)
    {
        if (n % prime[i] == 0)
        {
            int cnt = 1;
            while (n > 1 && n % prime[i] == 0)
            {
                n /= prime[i];
                cnt++;
            }
            ans *= cnt;
        }
    }
    if (n != 1)
        ans *= 2;
}
```

## 13.12 Number of Prime Divisor

```
vector<int>generateNumberOfPrimeDivisor(int n = 1e6){
    vector<int>primeDivisor(n+1, 0);
    for(int i=2;i<=n;i++){
        if(primeDivisor[i]==0){
            for(int j=i;j<=n;j+=i){
                primeDivisor[j] ++;
            }
        }
    }
    return primeDivisor;
}
```

## 13.13 Pascal Tree

```
void generatePascalsTriangle(int maxN)
{
    for (int n = 0; n <= maxN; n++)
    {
        pascalTriangle[n][0] = 1;
        for (int r = 1; r <= n; r++)
        {
            pascalTriangle[n][r] =
                pascalTriangle[n - 1][r - 1] +
                pascalTriangle[n - 1][r];
        }
    }
}
ll nCr(int n, int r) { return pascalTriangle[n][r]; }
```

## 13.14 Phi 4 large number

```
int phi(int n)
{
    double res = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            while (n % i == 0)
                n /= i;
            res *= (1.0 - (1.0 / i))
        }
    }
    if (n > 1)
        res *= (1.0 - (1.0 / n));
    return (int)(res);
}
```

## 13.15 Phi for a large number

```
int phi(int n)
{
    double res = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
        {
            while (n % i == 0)
                n /= i;
            res *= (1.0 - (1.0 / i))
        }
    }
    if (n > 1)
        res *= (1.0 - (1.0 / n));
    return (int)(res);
}
```

## 13.16 Pollard Rho

```
ll modular_pow(ll base, int exponent, ll modulus)
{
    ll result = 1;
    while (exponent > 0)
    {
        if (exponent & 1)
            result = (result * base) % modulus;
        exponent = exponent >> 1;
        base = (base * base) % modulus;
    }
    return result;
}
ll PollardRho(ll n)
{
    srand(time(NULL));
    if (n == 1)
        return n;
    if (n % 2 == 0)
        return 2;
    ll x = (rand() % (n - 2)) + 2;
    ll y = x;
    ll c = (rand() % (n - 1)) + 1;
    ll d = 1;
    while (d == 1)
    {
        x = (modular_pow(x, 2, n) + c + n) % n;
        y = (modular_pow(y, 2, n) + c + n) % n;
        y = (modular_pow(y, 2, n) + c + n) % n;
        d = __gcd(abs(x - y), n);
        if (d == n)
            return PollardRho(n);
    }
    return d;
}
```

## 13.17 SOD

```
// SOD of n^m:
//(P1e1 +1    1 / P1    1). (P2e2 +1    1 / P2    1)..... (
    Prer +1    1 / Pr    1)
ll primeFact(ll n, int m)
{
    ll sum = 1;
    for (int i = 0; i < primes.size() && primes[i] <= n; i++)
    {
        ll cnt = 0, p = primes[i];
        if (n % p == 0)
        {
            while (n % p == 0)
                cnt++, n /= p;
            cnt = cnt * m + 1;
            ll calc = (bigMod(p, cnt, MOD) + MOD - 1) % MOD;
            calc *= bigMod(p - 1, MOD - 2, MOD);
            calc %= MOD;
            sum = (sum * calc) % MOD;
        }
    }
    if (n > 1)
    {
        ll calc = (bigMod(n, 1 + m, MOD) + MOD - 1) % MOD;
        calc *= bigMod(n - 1, MOD - 2, MOD);
        calc %= MOD;
        sum = (sum * calc) % MOD;
    }
    return sum;
}

// Sod
sod(n) = (p1 ^ (e1 + 1) - 1) / (p1 - 1)
```

## 13.18 Sum of Divisor

```
vector<int> generateSumOfDivisor(int n = 2e6)
{
    vector<int> divSum(n + 5, 1);
    divSum[1]=0;
    for (int i = 2; i <= n; i++)
    {
        for(int j=i+i;j<=n;j+=i){
            divSum[j]+=i;
        }
    }
    return divSum;
}
```

# 14 String

## 14.1 Hashing

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

const int N = 1e6 + 9;
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
pair<int, int> pw[N], ipw[N];
int ip1, ip2;

int power(long long n, long long k, int mod)
{
    int ans = 1 % mod;
    n %= mod;
    if (n < 0)
        n + mod;
    while (k)
    {
        if (k & 1)
            ans = (long long)ans * n % mod;
        n = (long long)n * n % mod;
        k >>= 1;
    }
    return ans;
}

void prec()
{
    pw[0] = {1, 1};
    for (int i = 1; i < N; i++)
    {
        pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
        pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
    }
    ip1 = power(p1, MOD1 - 2, MOD1);
    ip2 = power(p2, MOD2 - 2, MOD2);
    ipw[0] = {
        1,
        1,
    };
    for (int i = 1; i < N; i++)
    {
        ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
        ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
    }
}

struct Hashing
{
    int n;
    string s;
    vector<pair<int, int>> hs;
    Hashing() {}
    Hashing(string _s)
    {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (int i = 0; i < n; i++)
        {
            pair<int, int> p;
            p.first = (hs[i].first + 1ll * pw[i].first * s[i]
                  % MOD1) % MOD1;
            p.second = (hs[i].second + 1ll * pw[i].second * s
                  [i] % MOD2) % MOD2;
            hs.push_back(p);
        }
    }

    pair<int, int> get_hash(int l, int r)
    {
        pair<int, int> ans;
        ans.first = (hs[r].first - hs[l - 1].first + MOD1) *
             1ll * ipw[l - 1].first % MOD1;
        ans.second = (hs[r].second - hs[l - 1].second + MOD2)
              * 1ll * ipw[l - 1].second % MOD2;
        return ans;
    }
};

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    prec();

    string s;
    cin >> s;
    int n = s.size();

    Hashing h(s);

    int ans = -1;
    for (int i = 1; i < n; i++)
    {
        auto p1 = h.get_hash(1, i), p2 = h.get_hash(n - i +
             1, n);
        if (p1 == p2) ans = i;
    }

    if (ans == -1) cout << "Not found!" << endl;
    else
    {
        for (int i = 0; i < ans; i++)
            cout << s[i];
    }

    //abxaxbxab
    //pre: ab
    //suff: ab
}
```

## 14.2 KMP

```
vector<int> lps(string pattern)
{
    int n = pattern.size();
    vector<int> v(n);
    int index = 0;

    for (int i = 1; i < n;)
    {
        if (pattern[i] == pattern[index])
        {
            v[i] = index + 1;
            i++;
            index++;
        }
        else
        {
            if (index)
                index = v[index - 1];
            else
            {
                v[i] = 0;
                i++;
            }
        }
    }
    return v;
}

int kmp(string s, string pattern)
{
    int n = s.size(), m = pattern.size();
    int i = 0, j = 0;
    int ans = 0;

    vector<int> v = lps(pattern);

    while (i < n)
    {
        if (s[i] == pattern[j])
        {
            i++;
            j++;
        }
        else
        {
            if (j)
                j = v[j - 1];
            else
                i++;
        }

        if (j == m) //to count how many pattern match
        {
            ans++;
            j = v[j - 1];
        }
    }

    return ans;
}
```

## 14.3 KthLexiSmallestSubstring

```
const int N = 1e5 + 5;
struct state {
int len, link, next[26];
ll cnt = 0, cnt2 = -1;
};
string s; state st[2 * N];
int n, k, sz, last;
vector<pair<int, int>> order;

void st_init() {
st[0].len = 0;
st[0].link = -1;
sz++; last = 0; }

void dfs(int u) {
if (st[u].cnt2 != -1) return;
st[u].cnt2 = st[u].cnt;
for (int i = 0; i < 26; ++i) {
if(st[u].next[i]) {
dfs(st[u].next[i]);
st[u].cnt += st[st[u].next[i]].cnt; } } }

void st_extend(int c) {
int cur = sz++;
st[cur].len = st[last].len + 1;
st[cur].cnt = 1;
order.emplace_back(st[cur].len, cur);
int p = last;
while (p != -1 && !st[p].next[c]) {
st[p].next[c] = cur;
p = st[p].link; }
if (p == -1) st[cur].link = 0;
else {
int q = st[p].next[c];
if (st[p].len + 1 == st[q].len) {
st[cur].link = q;}
else {
int clone = sz++;
st[clone].len = st[p].len + 1;
st[clone].link = st[q].link;
memcpy(st[clone].next, st[q].next, sizeof(st[q].next));
order.emplace_back(st[clone].len, clone);
while (p != -1 && st[p].next[c] == q) {
st[p].next[c] = clone;
p = st[p].link; }
st[cur].link = st[q].link = clone; } }
last = cur; }

int main() {
cin >> s >> k;
n = s.length(); k += n;
st_init();
for (int i = 0; i < n; ++i) st_extend(s[i] - ’a’);
sort(order.begin(), order.end());
reverse(order.begin(), order.end());
for (auto &p: order) {
st[st[p.second].link].cnt += st[p.second].cnt; }

dfs(0);
if (st[0].cnt < k) {
cout << "No such line.";
return 0; }

int cur = 0;
while (k > st[cur].cnt2) {
k -= st[cur].cnt2;
for (int i = 0; i < 26; ++i) {
if (st[cur].next[i]) {
int j = st[cur].next[i];
if (st[j].cnt < k) k -= st[j].cnt;
else {
cout << (char)(i + ’a’);
cur = j;
break; } } } } }
```

## 14.4 lcs

```
int lcs(string X, string Y, int m, int n, vector<vector<int
    >> &dp)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1)
    {
        return dp[m][n];
    }
    return dp[m][n] = max(lcs(X, Y, m, n - 1, dp), lcs(X, Y,
        m - 1, n, dp));
}
```

## 14.5 Mancher

```
struct Manacher {
  vector<int> p[2];
    p[0][2] = 2
  Manacher(string s) {
    int n = s.size();
    p[0].resize(n + 1);
    p[1].resize(n);
    for (int z = 0; z < 2; z++) {
      for (int i = 0, l = 0, r = 0; i < n; i++) {
        int t = r - i + !z;
        if (i < r) p[z][i] = min(t, p[z][l + t]);
        int L = i - p[z][i], R = i + p[z][i] - !z;
        while (L >= 1 && R + 1 < n && s[L - 1] == s[R + 1])
          p[z][i]++, L--, R++;
        if (R > r) l = L, r = R;
      }
    }
  }
  bool is_palindrome(int l, int r) {
    int mid = (l + r + 1) / 2, len = r - l + 1;
    return 2 * p[len % 2][mid] + len % 2 >= len;
  }
};
string s; cin >> s;
Manacher M(s);
int n = s.size();
for (int i = 0; i < n; i++) {
    cout << 2 * M.p[1][i] + 1 << ' ';
    if (i + 1 < n) cout << 2 * M.p[0][i + 1] << ' ';
}
```

## 14.6 String Matching with Bitset

```
const int N = 1e5 + 9;
vector<int> v;
bitset<N>bs[26], oc;
int i, j, k, n, q, l, r;
string s, p;
cin >> s;
for(i = 0; s[i]; i++) bs[s[i] - 'a'][i] = 1;
cin >> q;
while(q--) {
  cin >> p;
  oc.set();
  for(i = 0; p[i]; i++) oc &= (bs[p[i] - 'a'] >> i);
  cout << oc.count() << endl; // number of occurences
  int ans = N, sz = p.size();
  int pos = oc._Find_first();
  v.push_back(pos);
  pos = oc._Find_next(pos);
  while(pos < N) {
    v.push_back(pos);
    pos = oc._Find_next(pos);
  }
  for(auto x : v) cout << x << ' '; // position of
       occurences
  cout << endl;
  v.clear();
  cin >> l >> r; // number of occurences from l to r,where l
       and r is 1-indexed
  if(sz > r - l + 1) cout << 0 << endl;
  else cout << (oc >> (l - 1)).count() - (oc >> (r - sz + 1)
       ).count() << endl;
}
```

## 14.7 Suffix Array

```
const int mxN = 1e+5,K = 20;
int sa[mxN], pos[mxN], tmp[mxN],st[mxN][K+1],lcp[mxN],pre[
    mxN], gap, N;

bool comp(int x, int y) {
if(pos[x] != pos[y])return pos[x] < pos[y];
x += gap;
y += gap;
return (x < N && y < N) ? pos[x] < pos[y] : x > y; }

void suffix(string &s) {
for (int i = 0; i < N; i++) sa[i] = i, pos[i] = s[i];
for (gap = 1;; gap <<= 1) {
sort(sa, sa+N, comp);
for (int i = 0; i < N-1; i++)
tmp[i+1] = tmp[i] + comp(sa[i], sa[i+1]);
for (int i = 0; i < N; i++)
pos[sa[i]] = tmp[i];
if (tmp[N - 1] == N - 1) break; } }

int check(int m, string &s, string &x) {
int f = -1, k = x.size(), j = sa[m];
if (N - j >= k)f = 0;
for (int i = 0; i < min(N - j, k); i++) {
if (s[j+i] < x[i]) return -1;
if (s[j+i] > x[i]) return 1; }
return f; }

int patternExistsLB(int l,int r, string &s, string &x) {
int ans=-1;
while (l <= r) {
int m = l + (r-l)/2;
int v = check(m, s, x);
if (v == 0){
ans = m; r = m - 1; }
else if (v == 1) r = m - 1;
else l = m + 1; }
return ans; }
int patternExistsRB(int l,int r, string &s, string &x) {
int ans=-1;
while (l <= r) {
int m = l + (r-l)/2;
int v = check(m,s, x);
if (v == 0) {
ans = m; l = m + 1;}
else if (v == -1) l = m + 1;
else r = m - 1; }
return ans; }

int patternCount(int l,int r, string s,string &x) {
int L=patternExistsLB(l,r,s,x);
if(L==-1)return 0;
int R=patternExistsRB(L,r,s,x);
return (R-L)+1; }

int query(int l, int r) {
int j = log2(r-l+1);
return min(st[l][j], st[r-(1<<j)+1][j]) + 1; }
```

```
void buildIdx() {
for (int i = 0; i < N; i++)
st[i][0] = sa[i];

for (int j = 1; j <= K; j++) {
for (int i = 0; i + (1<<j) <= N; i++) {
st[i][j] = min(st[i][j-1], st[i + (1<<j-1)][j-1]); } } }

int findFirstIdx(int l,int r, string s,string &x) {
int L=patternExistsLB(l,r,s,x);
if(L==-1)return -1;
int R=patternExistsRB(L,r,s,x);
return query(L,R); }

void build_lcp(string &s){
for (int i = 0, k = 0; i < N; i++) if (pos[i] != N-1) {
int j = sa[pos[i] + 1];
while (s[i + k] == s[j + k]) k++;
lcp[pos[i]] = k;
if (k) k--; } }

long long int getUniqueSubCnt() {
long long int sm = accumulate(lcp, lcp+N, 0LL);
long long int tot = ((long long)1*(N)*(N+1))/2;
tot-=sm;
return tot; }

void printEveryUnqiueSubStirngCnt() {
int prev = 0;
for (int i = 0; i < N; i++) {
pre[prev + 1]++;
pre[N - sa[i] + 1]--;
prev = lcp[i]; }
for (int i = 1; i <= N; i++) {
cout << pre[i] << ’ ’ ;
pre[i+1] += pre[i];} }

string kthDistinctSubstring(string s,long long k){
long long prev = 0;
long long cur = 0;
for (int i = 0; i < N; i++){
if (cur + (N-sa[i]) - prev >= k) {
long long j = prev;
string ans = s.substr(sa[i], j);
while (cur < k){
ans += s[sa[i]+j];
cur++; j++; }
return ans; }
cur += (N-sa[i]) - prev;
prev = lcp[i]; }
```

```
return ""; }
```

## 15 templates

```
struct Point
{
   double x, y;
   Point() {}
   Point(double x, double y) : x(x), y(y) {}
   Point(const Point &p) : x(p.x), y(p.y) {}
   void input()
   {
      scanf("%lf%lf", &x, &y);
   }
   Point operator+(const Point &p)
      const
   {
      return Point(x + p.x, y + p.y);
   }
   Point operator-(const Point &p)
      const
   {
      return Point(x - p.x, y - p.y);
   }
   Point operator*(double c) const
   {
      return Point(x * c, y * c);
   }
   Point operator/(double c) const
   {
      return Point(x / c, y / c);
   }
};
vector<Point> polygon;
double getClockwiseAngle(Point p)
{
   return -1 * atan2(p.x, -1 * p.y);
}
// compare function to compare clockwise
bool comparePoints(Point p1, Point p2)
{
   return getClockwiseAngle(p1) < getClockwiseAngle(p2);
}
// rotate 90 degree counter clockwise
Point RotateCCW90(Point p)
{
   return Point(-p.y, p.x);
}
// rotate 90 degree clockwise
Point RotateCW90(Point p)
{
   return Point(p.y, -p.x);
}

Point RotateCCW(Point p, double t)
{
   return Point(p.x * cos(t) - p.y * sin(t),
               p.x * sin(t) + p.y * cos(t));
}
Point RotateCW(Point p, double t)
{
   return Point(p.x * cos(t) + p.y * sin(t),
               -p.x * sin(t) + p.y * cos(t));
}
double dot(Point A, Point B)
{
   return A.x * B.x + A.y * B.y;
}
double cross(Point A, Point B)
{
   return A.x * B.y - A.y * B.x;
}
double dist2(Point A, Point B)
{
   return dot(A - B, A - B);
}
// returns distance between two point
double dist(Point A, Point B)
{
   return sqrt(dot(A - B, A - B));
}
// Distance between point A and B
double distBetweenPoint(Point A, Point B)
{
   return sqrt(dot(A - B, A - B));
}
// project point c onto line AB (A != B)
Point ProjectPointLine(Point A, Point B, Point C)
{
   return A + (B - A) *
                dot(C - A, B - A) / dot(B - A, B - A);
}
// Determine if Line AB and CD are parallel or collinear
bool LinesParallel(Point A, Point B, Point C, Point D)
{
   return fabs(cross(B - A, C - D)) < EPS;
}
// Determine if Line AB and CD are collinear
```

```
bool LinesCollinear(Point A, Point B, Point C, Point D)
{
    return LinesParallel(A, B, C, D) && fabs(cross(A - B, A - C)) < EPS && fabs(cross(C - D, C - A)) < EPS;
}

// checks if AB intersect with CD
bool SegmentIntersect(Point A, Point B, Point C, Point D)
{
    if (LinesCollinear(A, B, C, D))
    {
        if (dist2(A, C) < EPS || dist2(A, D) < EPS || dist2(B, C) < EPS || dist2(B, D) < EPS)
            return true;
        if (dot(C - A, C - B) > 0 && dot(D - A, D - B) > 0 && dot(C - B, D - B) > 0)
            return false;
        return true;
    }
    if (cross(D - A, B - A) * cross(C - A, B - A) > 0)
        return false;
    if (cross(A - C, D - C) * cross(B - C, D - C) > 0)
        return false;
    return true;
}
// Compute the coordinates where AB and CD intersect
Point ComputeLineIntersection(Point A, Point B, Point C, Point D)
{
    double a1, b1, c1, a2, b2, c2;
    a1 = A.y - B.y;
    b1 = B.x - A.x;
    c1 = cross(A, B);
    a2 = C.y - D.y;
    b2 = D.x - C.x;
    c2 = cross(C, D);
    double Dist = a1 * b2 - a2 * b1;
    return Point((b1 * c2 - b2 * c1) / Dist, (c1 * a2 - c2 * a1) / Dist);
}
// Project point C onto line segment AB -- return the Point from AB which is the closest to C --
Point ProjectPointSegment(Point A, Point B, Point C)
{
    double r = dot(B - A, B - A);
    if (fabs(r) < EPS)
        return A;
    r = dot(C - A, B - A) / r;
    if (r < 0)
        return A;
    if (r > 1)
        return B;
    return A + (B - A) * r;
}
// return the minimum distance from a point C to a line AB
double DistancePointSegment(Point A, Point B, Point C)
{
    return distBetweenPoint(C, ProjectPointSegment(A, B, C));
}

// return distance between P and a
// point where p is perpendicular on
// AB.AB er upore p jei point e lombo
// shei point theke p er distance

double distToLine(Point p, Point a, Point b)
{
    pair<double, double> c;
    double scale = (double)(dot(p - a, b - a)) / (dot(b - a, b - a));
    c.first = a.x + scale * (b.x - a.x);
    c.second = a.y + scale * (b.y - a.y);
    double dx = (double)p.x - c.first;
    double dy = (double)p.y - c.second;
    return sqrt(dx * dx + dy * dy);
}
long long orientation(Point p, Point q, Point r)
{
    long long val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
    if (val > 0)
        return 1;
    if (val < 0)
        return 2;
    else
        return val;
}

// Given three colinear points p,
// q, r, the function checks if
// point q lies on line segment
// 'pr'

bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}
// checks if Point P is inside of
// polygon or not
bool isInside(int n, Point p)
{
    if (n < 3)
        return false;
    Point extreme = Point(INF, p.y); // here INF=1e4
    int count = 0, i = 0;
    do
    {
        int next = (i + 1) % n;
        if (SegmentIntersect(polygon[i], polygon[next], p, extreme))
        {
            if (orientation(polygon[i], p, polygon[next]) == 0)
                return onSegment(polygon[i], p, polygon[next]);
            count++;
        }
        i = next;
    } while (i != 0);
    return count & 1;
}
// returns the perimeter of a
// polygon
double polygonPerimeter(int n)
{
    double perimeter = 0.0;
    for (int i = 0; i < n - 1; i++) // polygon vector holds the corner points of the given polygon
        perimeter += dist(polygon[i], polygon[i + 1]);
    perimeter += dist(polygon[0], polygon[n - 1]);
    return perimeter;
}
// returns the area of a polygon
double polygonArea(int n)
{
    double area = 0.0;
    int j = n - 1;
    for (int i = 0; i < n; i++)
    {
        area += (polygon[j].x + polygon[i].x) * (polygon[j].y - polygon[i].y);
        j = i;
    }
    return fabs(area) * 0.5;
}
double getTriangleArea(Point a, Point b, Point c)
{
    return fabs(cross(b - a, c - a));
}
bool compareConvex(Point X, Point Y)
{
    long long ret =
        orientation(points[0], X, Y);
    if (ret == 0)
    {
        ll dist11 = dist2(points[0], X);
        ll dist22 = dist2(points[0], Y);
        return dist11 < dist22;
    }
    else if (ret == 2)
        return true;
    else
        return false;
}
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// make a minimum area polygon
stack<Point> convexHull(int N)
{
    int ymin = points[0].y, index = 0;
    for (int i = 1; i < N; i++)
    {
        if (points[i].y < ymin || (points[i].y == ymin &&
             points[i].x < points[index].x))
        {
            ymin = points[i].y;
            index = i;
        }
    }
    stack<Point> S;
    swap(points[0], points[index]);
    sort(&points[1], &points[N], compareConvex);
    S.push(points[0]);
    for (int i = 1; i < N; i++)
    {
        while (S.size() > 1 && orientation(nextToTop(S), S.
             top(), points[i]) != 2)
        {
            S.pop();
        }
```

```
            S.push(points[i]);
        }
    return S;
}
// Angle between Line AB and AC in degree
double angle(Point B, Point A, Point C)
{
    double c = dist(A, B);
    double a = dist(B, C);
    double b = dist(A, C);
    double ans = acos((b * b + c * c - a * a) / (2 * b * c));
    return (ans * 180) / acos(-1);
}
// returns number of vertices on boundary of a polygon
ll picks_theorem_boundary_count()
{
    int sz = polygon.size(), i;
    long long res = __gcd((long long)abs(polygon[0].x -
         polygon[sz - 1].x), (long long)abs(polygon[0].y -
         polygon[sz - 1].y));
    for (i = 0; i < sz - 1; i++)
    {
        res += __gcd((long long)abs(polygon[i].x - polygon[i
             + 1].x), (long long)abs(polygon[i].y - polygon[i
              + 1].y));
    }
    return res;
}

// picks theorem
// Polygon area= inside points + boundary points/2 -1
// return inside points counts
ll lattice_points_inside_polygon()
{
    ll ar = polygonArea(n);
    ll b =
        picks_theorem_boundary_count();
    long long tot = ar + 1 - b / 2;
    return tot;
}

// Physics Formuas
// motion
v = u + at
s = ut +(1/2) at*t ,
v*v    u*u = 2*a*s

// Projectile motion
x = utcos
y = utsin    (1/2) gt*t
```

```
y = x  tan     g*x*x/( 2u*u*cos* cos )
T = 2u  sin /g
R = u*u* sin2 /g
H = u*u* sin * sin /2g

// others:
p=mv
v*v/r*g = tan (Banking angle)
W = F S cos
K = (  )mv*v = p*p/ 2m
T = 2*  *sqrt(l/g)

// Trigonometry
 sin2 =2  sincos
 cos2 = cos * cos - sin * sin
 sin3 =3 sin -4* sin * sin * sin
 cos3 =4* cos * cos * cos -3 cos
For triangle:
a=bcosC+ccosB
b=acosC+ccosA
c=bcosA+acosB
sin(A+ B )=sinAcosB +   cosAsinB
cos(A+ B )=cosAcosB   + sinAsinB

// Circle Line intersection
double r, a, b, c; // given as input //ax+by+c=0//EPS=1e-9
double x0 = -a*c/(a*a+b*b), 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';}
```

## 16 Trie

```
struct Trie
{
    bool lastLetter;
    Trie *children[10];

    Trie()
    {
        for (int i = 0; i < 10; i++)
        {
            lastLetter = false;
            children[i] = nullptr;
        }
    }
};

void insert(string &s, Trie *root)
{
    int n = s.size();

    for (char c : s)
    {
        int index = c - '0';
        if (root->children[index] == nullptr)
            root->children[index] = new Trie();
        root = root->children[index];
    }
    root->lastLetter = true;
}

bool isPrefix(Trie *node)
{
    for (int i = 0; i < 10; i++)
    {
        if (node->children[i] != nullptr)
        {
            if (node->lastLetter)
                return true;
            if (isPrefix(node->children[i]))
                return true;
        }
    }
    return false;
}

void clear(Trie *node)
{
    for (int i = 0; i < 10; i++)
    {
        if (node->children[i] != nullptr)
        {
            clear(node->children[i]);
            node->children[i] = nullptr;
        }
    }
    delete (node);
}

int main()
{
    long long t, k = 0;
    cin >> t;
    while (t--)
    {
        long long n;
        cin >> n;

        Trie *root = new Trie();

        while (n--)
        {
            string s;
            cin >> s;
            insert(s, root);
        }

        bool f = !isPrefix(root);
        cout << loj(++k, con) << endl;
        clear(root);
    }
}
```

# 17 Z Function

```
#include <bits/stdc++.h>
using namespace std;
#define endl  \ n
vector<int> z_function(string s)
{
    int n = (int)s.size();
    vector<int> z(n);
    z[0] = n;
    for (int 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;
}
int main()
{
    // Simple is BEAST
    int n;
    cin >> n;
    cout << n * n << endl;
    return 0;
}
```