{

    "": {

        "prefix": "all\_pre\_built\_algo\_fnc",

        "body": [

          "",

          "const ll INF = 1e18;",

          "const int INFI = 1e9;",

          "const ll PRIME1 = 1000000007;",

          "const int PRIME2 = 998244353;",

          "",

          "",

          "// Basic Functions ",

          "",

          "ll binpow(ll a,ll b,ll m = LLONG\_MAX){",

          "    ll res = 1;",

          "    while (b){",

          "        if (b&1) res = (res\*a)%m;",

          "        a = (a\*a)%m;",

          "        b>>=1;",

          "    }",

          "    return res;",

          "}",

          "",

          "bool cols (vll& v1,vll& v2){",

          "    return v1[1]<v2[1];    ",

          "}",

          "",

          "// Number Theory",

          "",

          "vll fac; //factorial of n ",

          "void factorial(ll n){",

          "    fac.clear();",

          "    fac.resize(n+1);",

          "    fac[0]=1;",

          "    fac[1]=1;",

          "    for (ll i=2;i<=n;i++) fac[i]=(fac[i-1]\*i)%PRIME2;",

          "}",

          "",

          "vector<bool> isprime; //true=prime",

          "void primecheck(ll n){ ",

          "    isprime.clear();",

          "    isprime.assign(n+1,true);",

          "    isprime[0]=false;",

          "    isprime[1]=false;",

          "    for (ll i=2;i\*i<=n;i++){",

          "        if (isprime[i]){",

          "            for (ll j=i\*i;j<=n;j+=i) isprime[j]=false;",

          "        }",

          "    }",

          "}",

          "",

          "ll extendedgcd(ll a,ll b,ll& x,ll& y){",

          "    // returns gcd and changes x and y ",

          "    // such that gcd = ax+by",

          "    x=1,y=0;",

          "    ll x1=0,y1=1,a1=a,b1=b;",

          "    while (b1){",

          "        ll q = a1/b1;",

          "        tie(x,x1) = make\_tuple(x1,x-q\*x1);",

          "        tie(y,y1) = make\_tuple(y1,y-q\*y1);",

          "        tie(a1,b1)= make\_tuple(b1,a1-q\*b1);",

          "    }",

          "    return a1;",

          "}",

          "",

          "vll phi; //totient function of n",

          "void totient\_sieve(ll n){",

          "    // totient function = no. of integers upto n co-prime to n",

          "    phi.clear();",

          "    phi.resize(n+1);",

          "    for (ll i=0;i<=n;i++) phi[i]=i;",

          "    for (ll i=2;i<=n;i++) {",

          "        if (phi[i]==i){",

          "            for (ll j=i;j<=n;j+=i) phi[j]=phi[j]-phi[j]/i;",

          "        }",

          "    }",

          "}",

          "",

          "vll lpf; //largest prime factor of n",

          "void lpf\_sieve(ll n){",

          "    // largest prime factors of numbers upto n",

          "    lpf.clear();",

          "    lpf.assign(n+1,0);",

          "    for (ll i=2;i<=n;i++) {",

          "        if (lpf[i]==0){",

          "            for (ll j=i;j<=n;j+=i) lpf[j]=i;",

          "        }",

          "    }",

          "}",

          "void pf(ll n){",

          "    // prints prime factorization of n",

          "    // requires lpf",

          "    while (n>1){",

          "        cout<<lpf[n]<<\" \";",

          "        n/=lpf[n];",

          "    }    ",

          "    cout<<endl;",

          "}",

          "",

          "// DSU",

          "",

          "vll dsp;",

          "vll si;",

          "void make(ll v){",

          "    // makes new node",

          "    dsp[v]=v;",

          "    si[v]=1;",

          "}",

          "ll find(ll v){",

          "    // finds parent",

          "    if(v==dsp[v]) return v;",

          "    return dsp[v]=find(dsp[v]);",

          "}",

          "void unio(ll a,ll b){",

          "    // merges two sets",

          "    a=find(a);",

          "    b=find(b);",

          "    if(a!=b){",

          "        if(si[a]<si[b]) swap(a,b);",

          "        dsp[b]=a;",

          "        si[a]+=si[b];",

          "    }",

          "}",

          "",

          "//Trees",

          "",

          "vector<vll> children; //children of each node",

          "vll parent; //parent od each node",

          "vector<vll> ances; //2^ith ancestors of each node",

          "vll level; //distance from root of each node",

          "vll val; //spare vector",

          "",

          "void treeinit(ll n){",

          "    // Initialisation of tree structures ",

          "    // 1-indexed",

          "    ll k = log2(n)+1;",

          "    children.assign(n+1,vll(0));",

          "    parent.assign(n+1,-1);",

          "    ances.assign(n+1,vll(k,-1));",

          "    level.assign(n+1,0);",

          "    val.assign(n+1,0);",

          "}",

          "",

          "",

          "void Traversal(ll node){",

          "    // Traversing trees",

          "    for (auto x: children[node]){",

          "        if (x==parent[node]) continue;",

          "        parent[x]=node;",

          "        // add function here for Pre Order",

          "        Traversal(x);",

          "        // add function here for Post Order",

          "    }",

          "}",

          "",

          "void buildances(ll node){",

          "    // Build data structure for finding kth ancestor",

          "    // ances[node][i] = 2^ith ancestor of node",

          "    ll height = ances[0].size();",

          "    ances[node][0]=parent[node];",

          "    for (ll i=1;i<height;i++) {",

          "        if (ances[node][i-1]!=-1) {",

          "            ances[node][i]=ances[ances[node][i-1]][i-1];",

          "        }",

          "    }",

          "    for (auto x: children[node]){",

          "        if (x==parent[node]) continue;",

          "        parent[x]=node;",

          "        buildances(x);",

          "    }",

          "}",

          "",

          "ll findances(ll node,ll k){",

          "    // Returns kth ancestor of node",

          "    ll height=ances[0].size();",

          "    ll anc=node;",

          "    for (ll i=0;i<height;i++){",

          "        if (anc==-1) break;",

          "        if (k&1) anc=ances[anc][i];",

          "        k>>=1;",

          "    }",

          "    return anc;",

          "}",

          "",

          "ll lca(ll a,ll b){",

          "    // Returns lowest common ancestor of a and b",

          "    ll height=ances[0].size();",

          "    if (level[a]>level[b]) swap(a,b);",

          "    if (level[b]!=level[a]) b=findances(b,level[b]-level[a]);",

          "    for (ll i=height-1;i>=0;i--) {",

          "        if (ances[a][i]!=ances[b][i]) {",

          "            a=ances[a][i];",

          "            b=ances[b][i];",

          "        }",

          "    }",

          "    if (a!=b) return ances[a][0];",

          "    else return a;",

          "}",

          "",

          "//Graphs",

          "",

          "vector<vll> adj; //adjacent nodes of each node",

          "vector<vpll> adjw; //adjacent nodes of each node with weight of edge",

          "vector<bool> visited; //visited nodes check",

          "",

          "void bfs(ll s,vll& d,vll& p){",

          "    // Breadth First Search",

          "    ll n = adj.size();",

          "    d.assign(n,0);",

          "    p.assign(n,-1);",

          "    queue<ll> q;",

          "    vector<bool> used(n);",

          "    q.push(s);",

          "    used[s]=true;",

          "    p[s]=-1;",

          "    while (!q.empty()){",

          "        ll v = q.front();",

          "        q.pop();",

          "        for (ll u:adj[v]){",

          "            if (!used[u]){",

          "                used[u]=true;",

          "                q.push(u);",

          "                d[u]=d[v]+1; ",

          "                p[u]=v;",

          "            }",

          "        }",

          "    }",

          "}",

          "",

          "vii dfsarr; //dfs traversal array",

          "void dfs(ll v){",

          "    // Depth First Search",

          "    visited[v]=true;",

          "    for (auto u: adj[v]){",

          "        if (!visited[u]){",

          "            // enter pre function",

          "            dfs(u);",

          "            // enter post function",

          "        }",

          "    }",

          "    dfsarr.pb(v);",

          "}",

          "",

          "void dijkstra(ll s,vll& d,vll& p) {",

          "    // Dijkstra: shortest distance from source",

          "    ll n = adjw.size();",

          "    d.assign(n, INF);",

          "    p.assign(n, -1);",

          "    d[s] = 0;",

          "    set<pll> q;",

          "    q.insert({0, s});",

          "    while (!q.empty()) {",

          "        ll v = q.begin()->sec;",

          "        q.erase(q.begin());",

          "        for (auto edge : adjw[v]) {",

          "            ll to = edge.first;",

          "            ll len = edge.second;",

          "            if (d[v] + len < d[to]) {",

          "                q.erase({d[to],to});",

          "                d[to] = d[v] + len;",

          "                p[to] = v;",

          "                q.insert({d[to], to});",

          "            }",

          "        }",

          "    }",

          "}",

          "",

          "",

          "vector<vll> edges (0,vll(3)); //weight,node1,node2",

          "vpii minedges;",

          "ll kruskal(ll n){",

          "    //Kruskal: minimum spanning tree",

          "    ll minwt = 0;",

          "    minedges.clear();",

          "    sort(all(edges));",

          "    dsp.assign(n,0);",

          "    si.assign(n,0);",

          "    for (ll i=0;i<n;i++) dsp[i]=i;",

          "    for (auto x:edges){",

          "        ll u = find(x[1]);",

          "        ll v = find(x[2]);",

          "        if (u!=v){",

          "            minwt+=x[0];",

          "            minedges.pb({x[1],x[2]});",

          "            unio(u,v);",

          "        }",

          "    }",

          "    return minwt;",

          "}",

          "",

          "// Range Queries ",

          "",

          "vector<vll> st; ",

          "void constst(vll& v){",

          "    //Constructing Sparse Table ",

          "    ll n = v.size();",

          "    ll k = log2(n) + 1;",

          "    st.assign(n,vll (k,0));",

          "    for (ll i=0;i<n;i++) st[i][0]=v[i];",

          "    for (ll j=1;j<=k;j++){",

          "        for (ll i=0;i+(1<<j)<=n;i++){",

          "            st[i][j]=min(st[i][j-1],st[i+(1<<(j-1))][j-1]); // change function here",

          "        }",

          "    }",

          "}",

          "",

          "ll queryst(ll l,ll r){",

          "    // Query for [l,r] 0-indexed",

          "    ll j = log2(r-l+1);",

          "    return min(st[l][j],st[r-(1<<j)+1][j]); //change function here",

          "}",

          "",

          "vll seg;",

          "vll lazy;",

          "void buildseg(vll& a,ll v,ll tl,ll tr) {",

          "    // First resize seg,lazy to 4\*n and assign 0",

          "    // seg[v] gives ans for [tl,tr] 0-indexed",

          "    // Start : v=1,tl=0,tr=n-1",

          "    if (tl == tr) seg[v] = a[tl];",

          "    else {",

          "        ll tm = (tl + tr) / 2;",

          "        buildseg(a, v\*2, tl, tm);",

          "        buildseg(a, v\*2+1, tm+1, tr);",

          "        seg[v] = seg[v\*2] + seg[v\*2+1]; // change function here",

          "    }",

          "}",

          "",

          "void push(ll v,ll tl,ll tr){",

          "    // Default : addition operation",

          "    ll tm = (tl+tr)/2;",

          "    seg[2\*v]+=(tm-tl+1)\*lazy[v];",

          "    lazy[2\*v]+=lazy[v];",

          "    seg[2\*v+1]+=(tr-tm)\*lazy[v];",

          "    lazy[2\*v+1]+=lazy[v];",

          "    lazy[v]=0; // change identity here",

          "}",

          "",

          "ll queryseg(ll v,ll tl,ll tr,ll l,ll r) {",

          "    // Query for [l,r] 0-indexed",

          "    // seg[v] gives ans for [tl,tr]",

          "    // Start : v=1,tl=0,tr=n-1",

          "    if (l > r) return 0; //change identity here",

          "    if (l == tl && r == tr) return seg[v];",

          "    push(v,tl,tr);",

          "    ll tm = (tl + tr) / 2;",

          "    return queryseg(v\*2, tl, tm, l, min(r, tm)) + queryseg(v\*2+1, tm+1, tr, max(l, tm+1), r); //change function here",

          "}",

          "",

          "void updateseg(ll v,ll tl,ll tr,ll pos,ll new\_val) {",

          "    // Update at index pos (0-indexed)",

          "    // seg[v] gives ans for [tl,tr]",

          "    // Start : v=1,tl=0,tr=n-1",

          "    if (tl == tr) seg[v] = new\_val;",

          "    else {",

          "        ll tm = (tl + tr) / 2;",

          "        if (pos <= tm) updateseg(v\*2, tl, tm, pos, new\_val);",

          "        else updateseg(v\*2+1, tm+1, tr, pos, new\_val);",

          "        seg[v] = seg[v\*2] + seg[v\*2+1]; // change function here",

          "    }",

          "}",

          "",

          "void upranseg(ll v,ll tl,ll tr,ll l,ll r,ll addend) {",

          "    // Range Update for [l,r] 0-indexed",

          "    // Default: addition update operation, sum query operation",

          "    // seg[v] gives ans for [tl,tr]",

          "    // Start : v=1,tl=0,tr=n-1",

          "    if (l > r) return;",

          "    if (l == tl && tr == r) {",

          "        seg[v] += (tr-tl+1)\*addend; // change function here",

          "        lazy[v] += addend; // change function here",

          "    } ",

          "    else {",

          "        push(v,tl,tr);",

          "        ll tm = (tl + tr) / 2;",

          "        upranseg(v\*2, tl, tm, l, min(r, tm), addend);",

          "        upranseg(v\*2+1, tm+1, tr, max(l, tm+1), r, addend);",

          "        seg[v] = seg[v\*2] + seg[v\*2+1]; // change function here",

          "    }",

          "}",

          "",

          "// SQRT Algos ",

          "",

          "vector<vll> sqal;",

          "const ll block = 1000;",

          "",

          "bool compmomo(vll& v1,vll& v2){",

          "    // comparator to be used with momo()",

          "    if (v1[0]==v2[0]){",

          "        if (v1[0]&1) return v1[1]>v2[1];",

          "        return v1[1]<v2[1];",

          "    }",

          "    else return v1[0]<v2[0];",

          "}",

          "",

          "void momo(vpll& v){",

          "    // requires compmomo()",

          "    // root-N Optimization for Offline Queries (Mo's Algo)",

          "    // Element of vector = {l's set,r,l,query index}",

          "    ll n = v.size();",

          "    sqal.assign(n,vll(4,0));",

          "    for (ll i=0;i<n;i++){",

          "        sqal[i][0]=v[i].f/block;",

          "        sqal[i][1]=v[i].sec;",

          "        sqal[i][2]=v[i].f;",

          "        sqal[i][3]=i;",

          "    }",

          "    sort(all(sqal),compmomo);",

          "}",

          "",

          "// Strings ",

          "",

          "const int alpha = 26;",

          "",

          "struct triv{",

          "    ll next[alpha];",

          "    bool output=false;",

          "    triv(){",

          "        for (ll i=0;i<alpha;i++) next[i]=-1;",

          "    }",

          "};",

          "",

          "vector<triv> tri;",

          "",

          "void add\_string(string const& s){",

          "    ll v=0;",

          "    for (char c:s){",

          "        if (tri[v].next[c-'a']==-1){",

          "            tri[v].next[c-'a']=tri.size();",

          "            tri.emplace\_back();",

          "        }",

          "        v=tri[v].next[c-'a'];",

          "    }",

          "    tri[v].output=true;",

          "}",

          "",

          "bool search(const string& s){",

          "    ll v=0;",

          "    for (ll i=0;i<s.length();i++){",

          "        if (tri[v].next[s[i]-'a']==-1) return false;",

          "        v=tri[v].next[s[i]-'a'];",

          "    }",

          "    if (tri[v].output) return true;",

          "    return false;",

          "}",

          ""

        ],

        "description": ""

      }

}