# Template

#include <bits/stdc++.h>

using namespace std;

#define gettime printf("\nTime : %0.3lf\n",clock()\*1.0/CLOCKS\_PER\_SEC);

#define PB push\_back

#define MP make\_pair

#define rep(i,a,b) for(int i=a;i<=b;i++)

#define repp(i,a,b) for(int i=a;i>=b;i--)

#define Set(x,a) memset(x,a,sizeof(x));

#define vs vector<string>

#define vi vector<int>

#define ll long long

#define ff first

#define ss second

struct comp {

bool operator() (int a,int b) {

return a>b;

}

};

int main()

{

//freopen("input.txt","r",stdin); //freopen("output.txt","w",stdout);

return 0;

}

# Data Structure

## Segment Tree

int n = 100000;

int data[100005] = {};

int tree[400005] = {};

void build(int id = 1,int l = 1,int r = n){

if(l == r){ // l + 1 == r

tree[id] = data[l];

return ;

}

int mid = (l+r)/2;

buildSegmentTree(id \* 2, l, mid);

buildSegmentTree(id \* 2 + 1, mid+1, r);

tree[id] = tree[id \* 2] + tree[id \* 2 + 1];

}

void update(int p,int x,int id = 1,int l = 1,int r = n){

tree[id] += x - data[p]; //add to range the difference of current data with data to modified

if(l == r) {

data[p] = x;

return ;

}

int mid = (l + r)/2;

if(p <= mid)

updateSegmentTree(p, x, id \* 2, l, mid);

else

updateSegmentTree(p, x, id \* 2 + 1, mid+1, r);

}

int query(int x,int y,int id = 1,int l = 1,int r = n){

// out of range

// return depend on case

if(x > r || l > y) return 0;

if(x <= l && r <= y) return tree[id];

int mid = (l+r)/2;

return querySegmentTree(x, y, id \* 2, l, mid) +

querySegmentTree(x, y, id \* 2 + 1, mid+1, r);}

## Segment Tree Lazy Propogation

int n = 100000;

int data[100005] = {};

int tree[400005] = {};

int lazy[400005] = {};

void build(int id = 1,int l = 1,int r = n){

if(l == r){ // l + 1 == r

tree[id] = data[l];

return ;

}

int mid = (l+r)/2;

build(id \* 2, l, mid);

build(id \* 2 + 1, mid + 1, r);

tree[id] = tree[id \* 2] + tree[id \* 2 + 1];

}

void updateLazy(int id,int l,int r,int x){// increase all members in this interval by x

lazy[id] += x;

tree[id] += (r - l + 1) \* x;

}

void shiftToChild(int id,int l,int r){//pass update information to the children

int mid = (l+r)/2;

updateLazy(id \* 2, l, mid, lazy[id]);

updateLazy(id \* 2 + 1, mid + 1, r, lazy[id]);

lazy[id] = 0;// passing is done

}

void increaseInterval(int x,int y,int v,int id = 1,int l = 1,int r = n){

if(x > r || l > y) return ;

if(x <= l && r <= y){

updateLazy(id, l, r, v);

return ;

}

shiftToChild(id, l, r);

int mid = (l+r)/2;

increaseInterval(x, y, v, id \* 2, l, mid);

increaseInterval(x, y, v, id \* 2 + 1, mid + 1, r);

tree[id] = tree[id \* 2] + tree[id \* 2 + 1];

}

int query(int x,int y,int id = 1,int l = 1,int r = n){

if(x > r || l > y) return 0;

if(x <= l && r <= y) return tree[id];

shiftToChild(id, l, r);

int mid = (l+r)/2;

return query(x, y, id \* 2, l, mid) +

query(x, y, id \* 2 + 1, mid + 1, r);

}

## Persistent Segment Tree

int L[N\*2 + Q\*log(N)] : menyimpan indeks left dari segment tree

int R[N\*2 + Q\*log(N)] : menyimpan indeks right dari segment tree

int tree[N\*2 + Q\*log(N)] : menyimpan hasil segment tree dan sesudah query

int root[Q] - set 0 : start indeks dari setiap query

int data[N] : meyimpan data awal

int Next\_Indeks = 1 : untuk indexing dari segment tree dan query

int Segment\_Tree(int start,int end,int indeks) {

if (start == end) return tree[indeks] = data[start];

else {

int mid = (start + end) / 2;

L[indeks] = Next\_Indeks ++;

R[indeks] = Next\_Indeks ++;

return tree[indeks] = max (Segment\_Tree(start, mid, L[indeks]), Segment\_Tree(mid+1, end, R[indeks]));

}

}

int update(int start,int end,int change,int value,int indeks) {

int next = Next\_Indeks ++;

if (start == end) tree[next] = value;

else {

int mid = (start + end) / 2;

L[next] = L[indeks];

R[next] = R[indeks];

if (change <= mid) L[next] = update(start, mid, change, value, L[next]);

else R[next] = update(mid+1, end, change, value, R[next]);

tree[next] = max (tree[L[next]], tree[R[next]]);

}

return next;

}

int query(int start,int end,int l,int r,int indeks) {

if (l <= start && end <= r) return tree[indeks];

else {

int mid = (start + end) / 2;

if (r <= mid) return query(start, mid, l, r, L[indeks]);

else if (l >= mid + 1) return query(mid+1, end, l, r, R[indeks]);

else return max (query(start, mid, l, r, L[indeks]), query(mid+1, end, l, r, R[indeks]));

}

}

## BIT (Fenwick Tree)

int fenwick[100005] - set 0 : menyimpan hasil fenwick

void updateFenwick(int indeks,int value) {

for ( ; indeks <= n; indeks += indeks & -indeks) fenwick[indeks] += value;

}

int sumFenwick(int L, int R) {

int sumL = 0, sumR = 0;

L--;

for ( ; L > 0 ; L -= L & -L) sumL += fenwick[L];

for ( ; R > 0 ; R -= R & -R) sumR += fenwick[R];

return sumR - sumL;

}

## BIT (Fenwick Tree Range Update)

int dataMul[MAX\_N],dataAdd[MAX\_N];

void update(int left,int right,int val){

internalUpdate(left, val, -val \* (left – 1);

internalUpdate(right, -val, val \* right);

}

void internalUpdate(int pos, int mul, int add){

while(pos < MAX\_N){

dataMul[pos] += mul;

dataAdd[pos] += add;

pos |= (pos + 1);

}

}

int query(int pos){

int mul = 0;

int add = 0;

int start = pos;

while (pos >= 0) {

mul += dataMul[pos];

add += dataAdd[pos];

pos = (pos & (pos + 1)) - 1;

}

return mul \* start + add;}

## Union-Find Disjoint Set

int parent[N] : menyimpan nilai parent

int rank[N] : menyimpan ranking suatu set

void init(int n) { for (int i = 1; i <= n; i++) parent[i] = i, rank[i] = 1; }

int find(int x) { return parent[x] == x ? x : parent[x] = find(parent[x]); }

void gabung(int a,int b) {

int x = find(a), y = find(b);

if (x != y) {

if (rank[x] < rank[y]) swap(x, y);

rank[x] += rank[y];

rank[y] = 0;

parent[y] = x;

}

}

## Tries

string data : data yang akan di buat trie

int n : data.size()

struct Tries {

bool end;

Tries \*next[26];

};

void buildTrie(Tries \*trie,int indeks = 0) {

if (indeks >= n) (\*trie).end = true;

else {

if ((\*trie).next[data[indeks]-'a'] == NULL) {

Tries \*node = new Tries();

(\*trie).next[data[indeks]-'a'] = node;

}

buildTrie((\*trie).next[data[indeks]-'a'], indeks+1);

}

}

bool query(Tries \*trie,int indeks = 0) {

if (indeks >= n) return (\*trie).end;

else {

if ((\*trie).next[data[indeks]-'a'] == NULL) return false;

else return query((\*trie).next[data[indeks]-'a'], indeks+1);

}

}

## Sparse Table

int n : jumlah data

int data[N] : data

int sp[N][log2(N) + 1] : tempat menyimpan sparse table

void SparseTable() {

for (int i = 0; i < n; i++) sp[i][0] = i;

for (int j = 1 ; 1 << j <= n; j++) {

for (int i = 0; i + (1 << j) - 1 < n; i++) {

if (data[sp[i][j - 1]] < data[sp[i + 1 << (j - 1)][j - 1]]) sp[i][j] = sp[i][j - 1];

else sp[i][j] = sp[i + 1 << (j - 1)][j - 1];

}

}

}

int query(int l,int r) {

int k = log2 (r - l + 1);

return min(data[sp[l][k]], data[sp[r - (int)pow(2.0, k) + 1][k]]);

}

## Heavy Light Decomposition

int n : jumlah vertex

vi vertex[N] : menyimpan adjacent vertex dari suatu vertex

vi weight[N] : menyimpan weight dari edge pada adjacent vertex

vi edgeIndeks[N] : menyimpan indeks dari edge yang di gunakan untuk melakukan update weight pada edge

int topParent[N][log(N) + 1] : menyimpan top parent untuk LCA

int parent[N] : menyimpan parent dari suatu vertex untuk LCA dan Query HLD

int level[N] : menyimpan level dari suatu vertex untuk menghitung LCA

bool visit[N] - set false : status vertex untuk dfs LCA

int chainNo = 1 : menyimpan chain nomor selanjutnya

int chainHead[N] - set 0 : menyimpan head dari suatu chain untuk Query

int chainNumber[N] : menyimpan nomor chain dari suatu vertex

int subsize[N] : menyimpan jumlah subsize dari suatu vertex untuk mencari chain pada HLD

int baseArray[N] : menyimpan weight ke suatu vertex yang akan di gunakan untuk membuat segment tree

int posInArray[N] : menyimpan posisi dari suatu vertex pada baseArray yang di gunakan untuk update weight

int pointer = 1 : pointer untuk menyimpan nilai pada baseArray

int edgeEnd[N] : menyimpan akhir dari suatu edge yang di gunakan untuk update baseArray

int tree[N\*4] : menyimpan nilai segment tree

void dfs(int u = 1,int ParentOfU = 1,int Lvl = 0) {

if (!visit[u]) {

visit[u] = true;

level[u] = Lvl;

parent[u] = ParentOfU;

subsize[u] = 1;

int m = vertex[u].size();

for (int v = 0; v < m; v++)

if (!visit[vertex[u][v]]) {

dfs(vertex[u][v], u, Lvl+1);

subsize[u] += subsize[vertex[u][v]];

edgeEnd[edgeIndeks[u][v]] = vertex[u][v];

}

}

}

void HLD(int node,int prev,int cost) {

if (chainHead[chainNo] == 0) chainHead[chainNo] = node;

chainNumber[node] = chainNo;

baseArray[pointer] = cost;

posInArray[node] = pointer;

pointer++;

int nextnode = -1, nextcost = 0, m = vertex[node].size();

for (int i = 0; i < m; i++) if (vertex[node][i] != prev)

if (nextnode == -1 || subsize[nextnode] < subsize[vertex[node][i]])

nextnode = vertex[node][i], nextcost = weight[node][i];

if (nextnode != -1) HLD(nextnode, node, nextcost);

for (int i = 0; i < m; i++) {

int v = vertex[node][i];

if (v != nextnode && v != prev) chainNo++, HLD(v, node, weight[node][i]);

}

}

void buildLCA() {

for (int i = 1; i <= n; i++)

for (int j = 0; 1 << j < n; j++)

topParent[i][j] = (j == 0 ? parent[i] : -1);

for (int j = 1; 1 << j < n; j++)

for (int i = 1; i <= n; i++)

if (topParent[i][j - 1] != -1)

topParent[i][j] = topParent[topParent[i][j - 1]][j - 1];

}

int LCA(int u,int v) {

if (level[u] < level[v]) swap(u, v);

int k = log2 (level[u]);

for (int i = k; i >= 0; i--)

if (level[u] - (1 << i) >= level[v]) u = topParent[u][i];

if (u == v) return u;

for (int i = k; i >= 0; i--)

if (topParent[u][i]!= topParent[v][i]) u = topParent[u][i], v = topParent[v][i];

return parent[u];

}

int Segment\_Tree(int start,int end,int indeks) {

if (start == end - 1) return tree[indeks] = baseArray[end];

else {

int mid = (start + end) / 2;

return tree[indeks] = max (Segment\_Tree(start, mid, indeks\*2), Segment\_Tree(mid, end, indeks\*2 + 1));

}

}

void update\_tree(int start,int end,int change,int value,int indeks) {

if (start == end - 1) tree[indeks] = value;

else {

int mid = (start + end) / 2;

if (change <= mid) update\_tree(start, mid, change, value, indeks\*2);

else update\_tree(mid, end, change, value, indeks\*2 + 1);

tree[indeks] = max (tree[indeks\*2], tree[indeks\*2 + 1]);

}

}

void update(int edgeNo,int value) {

int u = edgeEnd[edgeNo];

update\_tree(1, pointer-1, u - 1, value, 1);

}

int query\_tree(int start,int end,int L,int R,int indeks) {

if (L <= start && end <= R) return tree[indeks];

else {

int mid = (start + end) / 2;

if (R <= mid) return query\_tree(start, mid, L, R, indeks\*2);

else if (L >= mid + 1) return query\_tree(mid+1, end, L, R, indeks\*2 + 1);

else return max (query\_tree(start, mid, L, R, indeks\*2), query\_tree(mid+1, end, L, R, indeks\*2 + 1));

}

}

int query\_chain(int u,int v) {

if (u == v) return 0;

int uChain, vChain = chainNumber[v], ans = 0;

while (true) {

uChain = chainNumber[u];

if (uChain == vChain) {

ans = max (ans, query\_tree(1, pointer-1, posInArray[v], posInArray[u], 1));

break;

}

ans = max (ans, query\_tree(1, pointer-1, posInArray[chainHead[uChain]], posInArray[u], 1));

u = chainHead[uChain];

u = parent[u];

}

return ans;

}

int query(int u,int v) {

int lca = LCA(u,v);

return max (query\_chain(u, lca), query\_chain(v, lca));

}

//cara penggunaan :

//rep (i,1,n-1) {

// scanf("%d %d %d",&a,&b,&c);

// vertex[a].PB(b);

// vertex[b].PB(a);

// weight[a].PB(c);

// weight[b].PB(c);

// edgeIndeks[a].PB(i);

// edgeIndeks[b].PB(i);

//}

//dfs();

//HLD(1,1,0);

//buildLCA();

//Segment\_Tree(1,pointer-1,1);

# Graph

## Dijkstra (SSSP)

struct edge { int v, w; };

int MAX : unlimited value

vector<edge> data[N] : data edge pada graph

long long dist[N] : jarak terpendek dari source ke sebuah verteks

priority\_queue<pii, vector<pii>, greater<pii> > next : queue berdasarkan jarak terpendek berikutnya

void dijkstra(int start,int n) {

for (int i = 1; i <= n; i++) dist[i] = MAX;

dist[start] = 0;

next.push(MP(0, start));

while (!next.empty()) {

int u = next.top().ss;

int n = data[u].size();

next.pop();

for (int i = 0; i < n; i++) {

int v = data[u][i].v;

int w = data[u][i].w;

if (w + dist[u] < dist[v]) {

dist[v] = w + dist[u];

next.push(MP(dist[v], v));

}

}

}

}

## Bellman Ford (SSSP, Negative Cycle)

//inside main

vi dist(V,INF); dist[s] = 0;

for(int i=0;i<V-1;i++)

for(int u=0;u<V;u++)

for(int j=0;j<AdjList[u].size();j++){

ii v = AdjList[u][j];

dist[v.first] = min(dist[v.first],dist[u]+v.second);

}

bool hasNegativeCycle = false;

for(int u=0;u<V;u++)

for(int j=0;j<Adjlist[u].size();j++){

ii v = AdjList[u][j];

if(dist[v.first]>dist[u] + v.second)

hasNegativeCycle = true;

}

## Floyd Warshall

//inside main

adjMat[i][j] ==> contain weight of i , j || no edge = INF

for(int k=0;k<V;k++)

for(int i=0;i<V;i++)

for(int j=0;j<V;j++)

aAdjMat[i][j] = min(AdjMat[i][j],AdjMat[i][k],AdjMat[k][j];

## Bipartite Graph Check

queue<int>q;q.push(s); // start vertex s

vi color(V,INF); color[s] =0;

bool isBipartite = true;

while(!q.empty() & isBipartite){

int u = q.front(); q.pop();

for(int j=0;j<AdjList[u].size();j++){

ii v = AdjList[u][j];

if(color[v.first] == INF){

color[v.first] = 1 – color[u];

q.push(v.first);}

else if (color[v.first] == color[u]){

isBipartite = false;break;}}}

## Tarjan (SCC)

vi dfs\_num, dfs\_low, S, visited;

void tarjanSCC(int u){

dfs\_low[u] = dfs\_num[u] = dfsNumberCounter++;

S.push\_back(u);

visited[u] = 1;

for(int j =0;j<AdjList[u].size();j++){

ii v = AdjList[u][j];

if (dfs\_num[v.first] == UNVISITED)

tarjanSCC(v.first);

if (visited[v.first])

dfs\_low[u] = min(dfs\_low[u],dfs\_low[v.first]);}

if(dfs\_low[u] == dfs\_num[u]){

printf(“SCC %d: “, ++numSCC);

while(1){

int v = S.back(); S.pop\_back(); visited[v]=0;

printf(“ %d”, v);

if( u == v ) break;}

pritnf(“\n”);

}}

//inside main

dfs\_num.assign(V, UNVISITED); dfs\_low.assign(V,0); visited.assign(V,0);

dfsNumberCounter = numSCC = 0;

for(int i=0;i<V;i++)

if(dfs\_num[i]==UNVISITED) tarjanSCC(i);

## Kruskal (MST)

vector< pair<int, ii> > EdgeList; //EdgeList (w,(u,v))

//input goes here

//sort edge first

sort(EdgeList.begin(), EdgeList.end());

int mst\_cost = 0; // init mst cost

UnionFind UF(V); // init UFDS

//each edge that doesnt make cycle, join it and add to mst\_cost

for (int i = 0; i < E; i++) {

pair<int, ii> front = EdgeList[i];

if (!UF.isSameSet(front.second.first, front.second.second)) {

mst\_cost += front.first;

UF.unionSet(front.second.first, front.second.second);

} }

## Prim (MST)

vector<vii> AdjList;

vi taken; //boolean to avoid cycle

priority\_queue<ii> pq; //priority\_queue to help choose shorter edge

void process(int vtx){

taken[vtx] = 1;

for(int j=0;j<AdjList[vtx].size();j++){

ii v = AdjList[vtx][j];

if(!taken[v.first]) pq.push(ii(-v.second,-v.first));

}

//inside main

taken.assign(V, 0);  
process(0); // prim start from vertex 0

int mst\_cost = 0;

while (!pq.empty()) {

ii front = pq.top(); pq.pop();

u = -front.second, w = -front.first;

if (!taken[u])

mst\_cost += w, process(u);

}

## Topo Sort

vector<int> data[N] : data graph

bool visit[N] - set false : menyimpan apakah suatu node telah di kunjungi

vector<int> ans : menyimpan hasil dari topologi sort

void topologi\_sort(int u) {

if (visit[u]) return;

visit[u] = true;

int n = data[u].size();

for (int i = 0; i < n; i++) topologi\_sort(data[u][i]);

ans.push\_back(u);

}

## LCA Square Decomposition

int n : jumlah node

vi data[N] : data graph

int topParent[N] : menyimpan top parent dari setiap node

int parent[N] : menyimpan parent dari setiap node

int level[N] : menyimpan level dari node

bool visit[N] - set false : menyimpan status visit node

void buildLCA(int u,int ParentOfU,int AkarN,int Lvl = 0) {

if (!visit[u]) {

visit[u] = true;

level[u] = Lvl;

parent[u] = ParentOfU;

if (Lvl < AkarN) topParent[u] = 1;

else if (level[u] % AkarN == 0) topParent[u] = parent[u];

else topParent[u] = topParent[parent[u]];

int m = data[u].size();

for (int v = 0; v < m; v++)

if (!visit[data[u][v]])

buildLCA(data[u][v], u, AkarN, Lvl + 1);

}

}

int LCA(int u,int v) {

while (topParent[u] != topParent[v]) {

if (level[u] > level[v]) u = topParent[u];

else v = topParent[v];

}

while (u != v) {

if (level[u] > level[v]) u = parent[u];

else v = parent[v];

}

return u;

}

## LCA Sparse Table

int n : jumlah node

vi data[N] : data graph

int topParent[N][log(N) + 1] : menyimpan top parent dari setiap node

int parent[N] : menyimpan parent dari setiap node

int level[N] : menyimpan level dari node

bool visit[N] - set false : menyimpan status visit node

void dfs(int u = 1,int ParentOfU = 1,int Lvl = 0) {

if (!visit[u]) {

visit[u] = true;

level[u] = Lvl;

parent[u] = ParentOfU;

int m = data[u].size();

for (int v = 0; v < m; v++)

if (!visit[data[u][v]])

dfs(data[u][v], u, Lvl+1);

}

}

void buildLCA() {

for (int i = 1; i <= n; i++)

for (int j = 0; 1 << j < n; j++)

topParent[i][j] = (j == 0 ? parent[i] : -1);

for (int j = 1; 1 << j < n; j++)

for (int i = 1; i <= n; i++)

if (topParent[i][j - 1] != -1)

topParent[i][j] = topParent[topParent[i][j - 1]][j - 1];

}

int LCA(int u,int v) {

if (level[u] < level[v]) swap(u, v);

int k = log2 (level[u]);

for (int i = k; i >= 0; i--)

if (level[u] - (1 << i) >= level[v]) u = topParent[u][i];

if (u == v) return u;

for (int i = k; i >= 0; i--)

if (topParent[u][i]!= topParent[v][i]) u = topParent[u][i], v = topParent[v][i];

return parent[u];

}

LCA Dynamic Programming

**void** process3(**int** N, **int** T[MAXN], **int** P[MAXN][LOGMAXN])

{

**int** i, j;

//we initialize every element in P with -1

**for** (i = 0; i < N; i++)

**for** (j = 0; 1 << j < N; j++)

     P[i][j] = -1;

//the first ancestor of every node i is T[i]

**for** (i = 0; i < N; i++)

        P[i][0] = T[i];

//bottom up dynamic programing

**for** (j = 1; 1 << j < N; j++)

**for** (i = 0; i < N; i++)

**if** (P[i][j - 1] != -1)

P[i][j] = P[P[i][j - 1]][j - 1];

}

**int** query(**int** N, **int** P[MAXN][LOGMAXN], **int** T[MAXN],

**int** L[MAXN], **int** p, **int** q)

{

**int** tmp, log, i;

//if p is situated on a higher level than q then we swap them

**if** (L[p] < L[q])

tmp = p, p = q, q = tmp;

//we compute the value of [log(L[p)]

**for** (log = 1; 1 << log <= L[p]; log++);

    log--;

//we find the ancestor of node p situated on the same level

//with q using the values in P

**for** (i = log; i >= 0; i--)

**if** (L[p] - (1 << i) >= L[q])

**p** = P[p][i];

**if** (p == q)

        return p;

//we compute LCA(p, q) using the values in P

**for** (i = log; i >= 0; i--)

**if** (P[p][i] != -1 && P[p][i] != P[q][i])

        p = P[p][i], q = P[q][i];

**return** T[p];

}

# Math

## Fast Modulo Multiplication

ll fast\_exp(int base, int exp) {

ll res=1;

while(exp>0) {

if(exp%2==1) res=(res\*base)%MOD;

base=(base\*base)%MOD;

exp/=2;

}

return res%MOD;

}

## Modular Multipicative Inverse

int Modular\_Inverse(int value,int mod) {

int tmp, x = 0, y = 1, times, modulus = mod;

if (mod <= 1 || \_\_gcd(value, mod) != 1) return -1;

while (value > 1) {

times = value / mod;

tmp = mod, mod = value % mod, value = tmp;

tmp = x, x = y - times \* x, y = tmp;

}

return (modulus + y) % modulus;

}

## Combination

int fact[N] : hasil factorial modulus mod

int nCk(int n,int k,int mod) {

if (n<k) return 0;

return ((ll)fact[n] \* Modular\_Inverse(fact[n-k], mod) % mod) \* Modular\_Inverse(fact[k], mod) % mod;

}

## NIM Game

n1^n2^n3 ^...^nk != 0 (winning state player 1)

## Misere Game

If n1 ... nk < 2 Then  
 n1^n2^n3^...^nk == 0 (winning state player 1)  
Else

n1^n2^n3^...^nk != 0 (winning state player 1)

## Fibonacci (Matrix Multiplication)

int mod : nilai modulo

map<ll,ll> data : menyimpan nilai fibo

ll fibo(ll indeks) {

if (data.find(indeks) != data.end()) return data[indeks];

if(indeks < 2) return 1LL;

int x = indeks / 2;

if (x\*2+1 == indeks) return (fibo(x) \* (fibo(x+1) + fibo(x-1))) % mod;

else return (fibo(x) \* fibo(x) + fibo(x-1) \* fibo(x-1)) % mod;

}

## Aritmetika

Un = a + (n – 1) \* b

Sn = n/2 (a + Un)

## Geometri

Un = a \* r ^ (n-1)

Sn = a(1 – r ^ n) / (1 – r)

## Number of Prime Factor of N

ll numPF(ll N) {

ll PF\_idx = 0, PF = primes[PF\_idx], ans = 0;

while ( PF \* PF <= N) {

while (N % PF == 0) N /= PF, ans++;

PF = primes[++PF\_idx];

}

if ( N != 1) ans++;

return ans;

}

## Number of Divisor

ll numDiv(ll N) {

ll PF\_idx = 0, PF = primes[PF\_idx], ans = 1;

while ( PF \* PF <= N) {

ll power = 0;

while ( N % PF == 0) N /= PF, power++;

ans \*= (power + 1);

PF = primes[++PF\_idx];

}

if (N != 1) ans \*= 2;

return ans;

}

## Sum of Divisor

ll sumDiv(ll N) {

ll PF\_idx = 0, PF = primes[PF\_\_idx], ans = 1;

while (PF \* PF <= N) {

ll power = 0;

while (N % PF == 0) N /= PF, power++;

ans \*= ((ll) pow((double)PF, power + 1.0) – 1) / (PF – 1);

PF = primes[++PF\_idx];

}

if (N != 1) ans \*= ((ll) pow((double) N, 2.0) – 1) / (N-1);

return ans;

}

## Euler Phi (untuk cari jumlah bilangan yang coprime dengan N)

ll EulerPhi(ll N) {

ll PF\_idx = 0, PF = primes[PF\_idx], ans = N;

while (PF \* PF <= N) {

if (N % PF == 0) ans -= ans / PF;

while(N%PF == 0) N /= PF;

PF = primes[++PF\_idx];

}

if (N != 1) ans -= ans / N;

return ans;

}

## Extended Euclid

Variable x, y, d : global variable

void extendedEuclid(int a, int b)

{

If (b == 0) { x = 1; y = 0; d = a; return; }

extendedEuclid(b, a % b);

int x1 = u;

int y1 = x – (a / b) \* y;

x = x1;

y = y1;

}

# Dynamic Programming

## Longest Increasing Subsequence

## answer = 0;

for (int i=1; i<=n; i++){

f[i]=lower\_bound(b+1, b+answer+1, a[i])-b;

maximize(answer, f[i]);

b[f[i]]=a[i];

}

printf("%d\n", answer);

//If you want to print the LIS:

vector<int> T;

int require = answer;

for (int i=n; i>=1; i--)

if (f[i]==require){

T.push\_back(a[i]);

require--;

}

// then print T with reversed order

int i=T.size();

while (i--) printf("%d ", T[i]);

printf("\n");

# String Processing

## Knuth Morris Pratt

string data1 : string pattern

string data2 : string teks yang akan di cek

int pat[N] : menyimpan pattern

int m : data1.size()

int n : data2.size()

void kmppattern() {

int i = 0, j = -1;

pat[0] = -1;

m = data1.size();

while (i < m) {

while (j >= 0 && data1[i] != data1[j]) j = pat[j];

i++; j++;

pat[i]=j;

}

}

int kmpsearch() {

int i = 0, j = 0, ans = 0;

n = data2.size();

while (i < n) {

while (j >= 0 && data2[i] != data1[j]) j = pat[j];

i++; j++;

if (j == m) {

ans++;

j = pat[j];

}

}

return ans;

}

## Z-Algorithm

int ans[N] : menyimpan hasil

string data : string yang akan di cek

int n : data.size()

void z\_algo() {

int L = 0, R = 0;

n = data.size();

ans[0] = 0;

for (int i = 1; i < n; i++) {

if (i > R) {

L = R = i;

while (R < n && data[R-L] == data[R]) R++;

ans[i] = R-L; R--;

}

else {

int k = i-L;

if (ans[k] < R-i+1) ans[i] = ans[k];

else {

L = i;

while (R < n && data[R-L] == data[R]) R++;

ans[i] = R-L; R--;

}

}

}

}

## Suffix Array & Longest Common Prefix

char T[100005];

int n;

int RA[100005],tempRA[100005];

int SA[100005],tempSA[100005];

int c[100005];

int PLCP[100005];

int Phi[100005];

int LCP[100005];

void countingSort(int k){

int i, sum, maxi = max(300, n);

memset(c, 0, sizeof c);

for (i=0; i < n;i++) c[i+k < n ?RA[i+k] : 0]++;

for (i=sum=0;i<maxi;i++){

int t = c[i];

c[i] = sum;

sum += t;

}

for(i=0;i<n;i++) tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];

for (i=0;i<n;i++) SA[i] = tempSA[i];

}

void constructSA(){

int i,k,r;

for(i=0;i<n;i++) RA[i] = T[i];

for(i=0;i<n;i++) SA[i] = i;

for(k=1;k<n;k<<=1){

countingSort(k);

countingSort(0);

tempRA[SA[0]] = r = 0;

for(i=1;i<n;i++){

tempRA[SA[i]] = (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;

}

for(i=0;i<n;i++)

RA[i] = tempRA[i];

if(RA[SA[n-1]] == n-1) break;

}

}

void computeLCP(){

int i, L;

Phi[SA[0]] = -1;

for(i=1;i<n;i++) Phi[SA[i]] = SA[i-1];

for(i=L=0;i<n;i++) {

if(Phi[i] == -1) {PLCP[i] = 0;continue;}

while(T[i+L] == T[Phi[i] + L]) L++;

PLCP[i] = L;

L = max(L-1,0);

}

for(i=0;i<n;i++) LCP[i] = PLCP[SA[i]];}

## Suffix Array

char data[N] : string yang akan di suffix

int n : strlen(data)

int range : jarak pengecekan pada saat compare

int sa[N] : hasil suffix array

int rank[N] : ranking pada saat compare suffix

int tmp[N] : untuk menampung sementara hasil ranking suffix

bool SufCmp(int x,int y) {

if (rank[x] != rank[y]) return rank[x] < rank[y];

x += range;

y += range;

return (x < n && y < n) ? rank[x] < rank[y] : x > y;

}

void SuffixArray() {

n = strlen(data);

for (int i = 0; i < n; i++) sa[i] = i, rank[i] = data[i];

tmp[0] = 0;

for (range = 1; ; range \*= 2) {

sort(sa, sa+n, SufCmp);

for (int i = 0; i < n-1; i++) tmp[i+1] = tmp[i] + SufCmp(sa[i], sa[i + 1]);

for (int i = 0; i < n; i++) rank[sa[i]] = tmp[i];

if (tmp[n-1] == n-1) break;

}

}

## LCP

int lcp[N] : hasil LCP

void LCP() {

SuffixArray();

for (int i = 0, k = 0; i < n; i++) if (rank[i] != n-1) {

for (int j = sa[rank[i] + 1]; data[i + k] == data[j + k]; ) ++k;

lcp[rank[i]] = k;

if (k) --k;

}

}

# Dan Lain-Lain

## DateToInt

String

day[8]={"Senin","Selasa","Rabu","Kamis","Jumat","Sabtu","Minggu"};

int DateToInt (int m, int d, int y){

return

1461 \* (y + 4800 + (m - 14) / 12) / 4 +

367 \* (m - 2 - (m - 14) / 12 \* 12) / 12 -

3 \* ((y + 4900 + (m - 14) / 12) / 100) / 4 +

d - 32075;

}

void IntToDate (int jd, int &m, int &d, int &y){

int x, n, i, j;

x = jd + 68569;

n = 4 \* x / 146097;

x -= (146097 \* n + 3) / 4;

i = (4000 \* (x + 1)) / 1461001;

x -= 1461 \* i / 4 - 31;

j = 80 \* x / 2447;

d = x - 2447 \* j / 80;

x = j / 11;

m = j + 2 - 12 \* x;

y = 100 \* (n - 49) + i + x;

}

string IntToDay (int jd){

return day[jd % 7

}

* 1. **Bigint**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<ll,ll> lll;

typedef pair<ll,int> lli;

typedef pair<int,int> ii;

#define EL printf("\n")

#define OK printf("OK")

#define pb push\_back

#define mp make\_pair

#define ep emplace\_back

#define X first

#define Y second

#define fillchar(a,x) memset(a, x, sizeof(a))

#define FOR(i,l,r) for (int i=l;i<=r;i++)

#define FORD(i,r,l) for (int i=r;i>=l;i--)

const int base = 1e9;

typedef vector<int> BigInt;

void Set(BigInt &a) {

while (a.size() > 1 && a.back() == 0) a.pop\_back();

}

void Print(BigInt a) {

Set(a);

printf("%d", (a.size() == 0) ? 0 : a.back());

FORD(i,a.size()-2,0) printf("%09d", a[i]); EL;

}

BigInt Integer(string s) {

BigInt ans;

if (s[0] == '-') return ans;

if (s.size() == 0) {ans.pb(0); return ans;}

while (s.size()%9 != 0) s = '0'+s;

for (int i=0;i<s.size();i+=9) {

int v = 0;

for (int j=i;j<i+9;j++) v = v\*10+(s[j]-'0');

ans.insert(ans.begin(),v);

}

Set(ans);

return ans;

}

BigInt Integer(char c[]) {

string s = "";

FOR(i,0,strlen(c)-1) s = s + c[i];

return Integer(s);

}

BigInt Integer(ll x) {

string s = "";

while (x > 0) s = char(x%10+'0') + s, x /= 10;

return Integer(s);

}

BigInt Integer(int x) {

return Integer((ll) x);

}

void operator >> (istream &in, BigInt &a) {

string s;

getline(cin, s);

a = Integer(s);

}

void operator << (ostream &out, BigInt a) {

Print(a);

}

bool operator < (BigInt a, BigInt b) {

Set(a);

Set(b);

if (a.size() != b.size()) return (a.size() < b.size());

FORD(i,a.size()-1,0)

if (a[i] != b[i]) return (a[i] < b[i]);

return false;

}

bool operator > (BigInt a, BigInt b) {

return (b < a);

}

bool operator == (BigInt a, BigInt b) {

return (!(a < b) && !(b < a));

}

bool operator <= (BigInt a, BigInt b) {

return (a < b || a == b);

}

bool operator >= (BigInt a, BigInt b) {

return (b < a || b == a);

}

bool operator < (BigInt a, int b) {

return (a < Integer(b));

}

bool operator > (BigInt a, int b) {

return (a > Integer(b));

}

bool operator == (BigInt a, int b) {

return (a == Integer(b));

}

bool operator >= (BigInt a, int b) {

return (a >= Integer(b));

}

bool operator <= (BigInt a, int b) {

return (a <= Integer(b));

}

BigInt max(BigInt a, BigInt b) {

if (a > b) return a;

return b;

}

BigInt min(BigInt a, BigInt b) {

if (a < b) return a;

return b;

}

BigInt operator + (BigInt a, BigInt b) {

Set(a);

Set(b);

BigInt ans;

int carry = 0;

FOR(i,0,max(a.size(), b.size())-1) {

if (i < a.size()) carry += a[i];

if (i < b.size()) carry += b[i];

ans.pb(carry%base);

carry /= base;

}

if (carry) ans.pb(carry);

Set(ans);

return ans;

}

BigInt operator + (BigInt a, int b) {

return a + Integer(b);

}

BigInt operator ++ (BigInt &a) { // ++a

a = a + 1;

return a;

}

void operator += (BigInt &a, BigInt b) {

a = a + b;

}

void operator += (BigInt &a, int b) {

a = a + b;

}

BigInt operator - (BigInt a, BigInt b) {

Set(a);

Set(b);

BigInt ans;

int carry = 0;

FOR(i,0,a.size()-1) {

carry += a[i] - (i < b.size() ? b[i] : 0);

if (carry < 0) ans.pb(carry+base), carry = -1;

else ans.pb(carry), carry = 0;

}

Set(ans);

return ans;

}

BigInt operator - (BigInt a, int b) {

return a - Integer(b);

}

void operator -- (BigInt &a) { // --a

a = a - 1;

}

void operator -= (BigInt &a, BigInt b) {

a = a + b;

}

void operator -= (BigInt &a, int b) {

a = a - b;

}

BigInt operator \* (BigInt a, BigInt b) {

Set(a);

Set(b);

BigInt ans;

ans.assign(a.size()+b.size(), 0);

FOR(i,0,a.size()-1) {

ll carry = 0ll;

for (int j=0;j<b.size() || carry > 0;j++) {

ll s = ans[i+j] + carry + (ll)a[i]\*(j<b.size()?(ll)b[j]:0ll);

ans[i+j] = s%base;

carry = s/base;

}

}

Set(ans);

return ans;

}

BigInt operator \* (BigInt a, int b) {

return a \* Integer(b);

}

void operator \*= (BigInt &a, BigInt b) {

a = a \* b;

}

void operator \*= (BigInt &a, int b) {

a = a \* b;

}

BigInt operator / (BigInt a, BigInt b) {

Set(a);

Set(b);

if (b == Integer(0)) return Integer("-1");

BigInt ans, cur;

FORD(i,a.size()-1,0) {

cur.insert(cur.begin(), a[i]);

int x = 0, L = 0, R = base;

while (L <= R) {

int mid = (L+R)>>1;

if (b\*Integer(mid) > cur) {

x = mid;

R = mid-1;

}

else

L = mid+1;

}

cur = cur - Integer(x-1)\*b;

ans.insert(ans.begin(),x-1);

}

Set(ans);

return ans;

}

BigInt operator / (BigInt a, int b) {

Set(a);

BigInt ans;

ll cur = 0ll;

FORD(i,a.size()-1,0) {

cur = (cur\*(ll)base + (ll)a[i]);

ans.insert(ans.begin(),cur/b);

cur %= b;

}

Set(ans);

return ans;

}

void operator /= (BigInt &a, BigInt b) {

a = a / b;

}

void operator /= (BigInt &a, int b) {

a = a / b;

}

BigInt operator % (BigInt a, BigInt b) {

Set(a);

Set(b);

if (b == Integer(0)) return Integer("-1");

BigInt ans;

FORD(i,a.size()-1,0) {

ans.insert(ans.begin(), a[i]);

int x = 0, L = 0, R = base;

while (L <= R) {

int mid = (L+R)>>1;

if (b\*Integer(mid) > ans) {

x = mid;

R = mid-1;

}

else

L = mid+1;

}

ans = ans - Integer(x-1)\*b;

}

Set(ans);

return ans;

}

int operator % (BigInt a, int b) {

Set(a);

if (b == 0) return -1;

int ans = 0;

FORD(i,a.size()-1,0)

ans = (ans\*(base%b) + a[i]%b)%b;

return ans;

}

void operator %= (BigInt &a, BigInt b) {

a = a % b;

}

void operator %= (BigInt &a, int b) {

a = a % Integer(b);

}

BigInt gcd(BigInt a, BigInt b) {

Set(a);

Set(b);

while (b > Integer(0)) {

BigInt r = a%b;

a = b;

b = r;

}

Set(a);

return a;

}

BigInt lcm(BigInt a, BigInt b) {

return (a\*b/gcd(a,b));

}

BigInt sqrt(BigInt a) {

BigInt x0 = a, x1 = (a+1)/2;

while (x1 < x0) {

x0 = x1;

x1 = (x1+a/x1)/2;

}

return x0;

}

BigInt pow(BigInt a, BigInt b) {

if (b == Integer(0)) return Integer(1);

BigInt tmp = pow(a, b/2);

if (b%2 == 0) return tmp \* tmp;

return tmp \* tmp \* a;

}

BigInt pow(BigInt a, int b) {

return pow(a,(Integer(b)));

}

int log(int n, BigInt a) { // log\_n(a)

Set(a);

int ans = 0;

while (a > Integer(1)) {

ans++;

a /= n;

}

return ans;

}

int main()

{

int a,b,c,tot;

cin>>a>>b>>c;

tot = a+b+c;

int maxx;

if(a>=b && a>=c)maxx = a,a = 1;

else if(b>=a && b>=c)maxx = b,b = 1;

else if(c>=a && c>=b)maxx = c,c =1;

BigInt hasilTot =Integer("1");

BigInt hasilA = Integer("1");

BigInt hasilB = Integer("1");

BigInt hasilC = Integer("1");

for(int i = tot;i>maxx;i--)hasilTot\*=i;

for(int i = 1;i<=a;i++)hasilA \*=i;

for(int i = 1;i<=b;i++)hasilB \*=i;

for(int i = 1;i<=c;i++)hasilC \*=i;

hasilTot/=hasilA;

hasilTot/=hasilB;

hasilTot/=hasilC;

Print(hasilTot);

}