**1 Some Useful Code**

**Max** priority\_queue<ll>

**Min** priority\_queue<ll,vector<ll>,greater<ll>>

**2 Number Theory**

**2.1 Prime number under 100**

// there are 25 numbers

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

**2.2 If prime number**

bool prime(ll n)

{

if (n<2) return false;

if (n<=3) return true;

if (!(n%2) || !(n%3)) return false;

for (ll i=5; i\*i<=n; i+=6)

if (!(n%i) || !(n%(i+2))) return false;

return true;

}

**2.3 Prime factorization**

**// smallest prime factor of a number.**

ll factor(ll n)

{

ll a;

if (n%2==0)

return 2;

for (a=3; a<=sqrt(n); a++++)

{

if (n%a==0)

return a;

}

return n;

}

**// complete factorization**

ll r;

while (n>1)

{

r = factor(n);

printf("%d", r);

n /= r;

}

**2.4 Leap year**

bool isLeap(ll n)

{

    if (n%100==0)

{

        if (n%400==0) return true;

        else return false;

}

    if (n%4==0) return true;

    else return false;

}

**2.5 Binary Exponentiation: (a^b)**

ll power(ll a, ll b) {

ll res = 1;

while (b > 0) {

if (b & 1)

res = res \* a;

a = a \* a;

b >>= 1;

}

return res;

}

**2.6 Binary Exponentiation: (a^b^c)**

ll binexp(ll base, ll power, ll modulo)

{

ll ans = 1;

while (power)

{

if (power % 2 == 1)

ans = (ans \* base) % modulo;

base = (base \* base) % modulo;

power /= 2;

}

return ans;

}

**//function call**  
binexp(a, binexp (b, c, mod - 1), mod);

**2.7 a^b mod p**

ll powmod(ll base, ll exp, ll modulus)

{

    base %= modulus;

    ll result = 1;

    while (exp > 0)

    {

        if (exp & 1) result = (result \* base) % modulus;

        base = (base \* base) % modulus;

        exp >>= 1;

    }

    return result;

}

**2.8 Factorial mod**

**//n! mod p**

ll factmod (ll n, ll p)

{

    ll res = 1;

    while (n > 1)

    {

        res = (res \* powmod (p-1, n/p, p)) % p;

        for (ll i=2; i<=n%p; ++i)

            res=(res\*i) %p;

        n /= p;

    }

    return ll (res % p);

}

**2.9 Next Greater Element :**

ll output[1000005];

void nextGreaterElement(ll x[], ll n) {

stack<ll> s;

s.push(0);

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

while (!s.empty() && x[s.top()] <= x[i]) {

output[s.top()] = i;

s.pop();

}

s.push(i);

}

while (!s.empty()) {

output[s.top()] = -1;

s.pop();

}

}

**2.10 Sieve:**

ll prime[20000005];

void sieve(ll n){

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

prime[i]=1;

}

for(ll i=4;i<=n;i+=2){

prime[i]=0;

}

for(ll i=3;i\*i<=n;i++){

if(prime[i]){

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

prime[j]=0;

}

}

}

}

**2.11 Segment Sieve**

void SegmentSieve(ll L, ll R){

if (L == 1)

L++;

ll maxN = R - L + 1;

ll a[maxN] = {0};

for (auto p : prime){

if (p \* p <= R)

{

ll x = (L / p) \* p;

if (x < L)

x += p;

for (ll i = x; i <= R; i += p)

{

if (i != p)

a[i - L] = 1;

}

}

else

break;}

for (ll i = 0; i < maxN; i++)

if (a[i] == 0)

cout << i + L << endl;

}

**2.12 Greatest common divisor — GCD**

int gcd(int a, int b)

{

if (b==0) return a;

else return gcd(b, a%b);

}

**2.13 Least common multiple — LCM**

int lcm(int a, int b)

{

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

}

**2.14 Num of trailing Zeros in factorial**

int res=0;

for(int i=5;i<=n;i=i\*5){

res=res+n/i;

}

cout<<res<<endl;

**2.15 Common Divisors:**

**You are given an array of *n* positive integers. Your task is to find two integers such that their greatest common divisor is as large as possible.**

int main()

{

int n;

cin>>n;

vector<int> range(1e6+1,0);

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

{

int x;

cin>>x;

range[x]++;

}

for(int gcd=1e6; gcd >=1; gcd--)

{

int multiples=0;

for(int pointer=gcd,pointer<=1e6; pointer+=gcd)

{

multiples+=range[pointer];

}

if(multiples>1)

{

cout<<gcd<<endl;

return 0;

}

}

}

**2.16 nCr:**

const ll MOD = 1e9 + 7;

const ll MAX = 2e5 + 5;

vector<ll> fact(MAX), inv(MAX);

void factorial() {

fact[0] = 1;

for (ll i = 1; i < MAX; i++)

fact[i] = (i \* fact[i - 1]) % MOD;

}

ll bigmod(ll a, ll n, ll M = MOD) {

ll res = 1;

while (n) {

if (n & 1)

res = (res \* a) % M;

a = (a \* a) % M, n /= 2;

}

return res;

}

void inverse() {

for (ll i = 0; i < MAX; ++i)

inv[i] = bigmod(fact[i], MOD - 2);

}

ll C(ll a, ll b) {

if (a < b or a < 0 or b < 0)

return 0;

ll de = (inv[b] \* inv[a - b]) % MOD;

return (fact[a] \* de) % MOD;

}

// call factorial() and inverse() from main function

// end nCR

ll ModInv(ll a, ll M) { // M is prime

return bigmod(a, M - 2, M);

}

**2.17 Set Balancing:**

// return middle element of the set

void balance(multiset<ll> right, multiset<ll> &left){

while (true){

ll st = right.size();

ll sl = left.size();

if (st == sl || st == sl + 1)

break;

if (st < sl)

right.insert(left.begin()), left.erase(left.begin());

else

left.insert(right.rbegin()), right.erase(right.rbegin());

}

}

void insert\_in\_set(multiset<ll> &right, multiset<ll> &left, ll value)

{

if (right.emptleft())

right.insert(value);

else

{

auto it = right.end();

it--;

if (value < \*it)

right.insert(value);

else

left.insert(value);

}

}

**3 String Algorithm**

**3.1 KMP ALGORITHM: O(n + m)**

vector<ll> createLPS(string pattern)

{

vector<ll> lps(pattern.length());

ll index = 0;

for (ll i = 1; i < pattern.length();)

{

if (pattern[index] == pattern[i])

{

lps[i] = index + 1;

index++, i++;

}

else

{

if (index != 0)

index = lps[index - 1];

else

lps[i] = index, i++;

}

}

return lps;

}

ll kmp(string text, string pattern)

{

ll cnt\_of\_match = 0;

vector<ll> lps = createLPS(pattern);

debug(lps);

ll i = 0, j = 0;

// i -> text, j -> pattern

while (i < text.length())

{

if (text[i] == pattern[j])

i++, j++;

else

{

if (j != 0)

j = lps[j - 1];

else

i++;

}

if (j == pattern.length())

{

cnt\_of\_match++;

// the index where match found -> (i - pattern.length());

j = lps[j - 1];

}

}

return cnt\_of\_match;

}

**4 Dynamic Programming**

**4.1 0/1 Knapsack problems-O(n\*w)**

**//Top Down**

ll ks(ll W, ll i){

if(i==0 || W<=0) return 0;

if(weight[i]>W) return ks(W,i-1);

if(mem[W][i]==0) mem[W][i]=max(ks(W,i-1),value[i]+ks(W-weight[i],i-1));

return mem[W][i];

}

**//Bottom Up**

ll knapsack(ll capacity, ll ind){

for(ll i=1;i<=ind;i++){

for(ll c=1;c<=capacity;c++){

if(weight[i]>c){

mem[i][c]=mem[i-1][c];

}

else{

ll k1=mem[i-1][c];

ll k2=value[i]+mem[i-1][c-weight[i]];

mem[i][c]=max(k1,k2);

}

}

}

ll max\_profit=mem[ind][capacity];

return max\_profit;

}

**4.2 Complete Knapsack problems**

#include <iostream>

using namespace std;

ll f[1000] = {0};

ll n = 0, m = 0;

ll main(void)

{

cin >> n >> m;

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

{

ll price = 0, value = 0;

cin >> price >> value;

for (ll j = price; j <= m; j++)

if (f[j - price] + value > f[j])

f[j] = f[j - price] + value;

}

cout << f[m] << endl;

return 0;

}

**4.3 Longest common subsequence (LCS)-O(n\*m)**

ll dp[1001][1001];

ll lcs(const string &s, const string &t)

{

ll m = s.size(), n = t.size();

if (m == 0 || n == 0)

return 0;

for (ll i = 0; i <= m; ++i)

dp[i][0] = 0;

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

dp[0][j] = 0;

for (ll i = 0; i < m; ++i)

for (ll j = 0; j < n; ++j)

if (s[i] == t[j])

dp[i + 1][j + 1] = dp[i][j] + 1;

else

dp[i + 1][j + 1] = max(dp[i + 1][j], dp[i][j + 1]);

return dp[m][n];

}

**4.4 Longest increasing common sequence (LICS)**

#include <iostream>

using namespace std;

ll a[100] = {0};

ll b[100] = {0};

ll f[100] = {0};

ll n = 0, m = 0;

ll main(void)

{

cin >> n;

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

cin >> a[i];

cin >> m;

for (ll i = 1; i <= m; i++)

cin >> b[i];

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

{

ll k = 0;

for (ll j = 1; j <= m; j++)

{

if (a[i] > b[j] && f[j] > k)

k = f[j];

else if (a[i] == b[j] && k + 1 > f[j])

f[j] = k + 1;

}

}

ll ans = 0;

for (ll i = 1; i <= m; i++)

if (f[i] > ans)

ans = f[i];

cout << ans << endl;

return 0;

}

**4.5 Longest Increasing Subsequence (LIS)-O(n^2)**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll n = 0;

ll a[100] = {0}, f[100] = {0}, x[100] = {0};

ll main(void)

{

cin >> n;

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

{

cin >> a[i];

x[i] = LONG\_LONG\_MAX;

}

f[0] = 0;

ll ans = 0;

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

{

ll l = 0, r = i;

while (l + 1 < r)

{

ll m = (l + r) / 2;

if (x[m] < a[i])

l = m;

else

r = m;

// change to x[m]<=a[i] for non-decreasing case

}

f[i] = l + 1;

x[l + 1] = a[i];

if (f[i] > ans)

ans = f[i];

}

cout << ans << endl;

return 0;

}

**4.6 MCM**

const ll N = 1005;

ll d[N];

ll dp[N][N], mark[N][N];

ll MCM(ll i, ll j)

{

if (i == j)

return dp[i][j] = 0;

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

return dp[i][j];

ll mn = inf;

for (ll k = i; k < j; k++)

{

ll x = mn;

mn = min(mn, MCM(i, k) + MCM(k + 1, j) + d[i - 1] \* d[k] \* d[j]);

if (x != mn)

mark[i][j] = k;

}

return dp[i][j] = mn;

}

**5. Graph Theory**

**5.1 Knight Moves**

ll X[8]={2,1,-1,-2,-2,-1,1,2};

ll Y[8]={1,2,2,1,-1,-2,-2,-1};

**5.2 SPFA (Shortest Path) O(VxE)**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll q[3001] = {0}; // queue for node

ll d[1001] = {0}; // record shortest path from start to ith node

bool f[1001] = {0};

ll a[1001][1001] = {0}; // adjacency list

ll w[1001][1001] = {0}; // adjacency matrix

void SPFA(ll v0);

ll main(void)

{

ll n = 0, m = 0;

cin >> n >> m;

for (ll i = 1; i <= m; i++)

{

ll x = 0, y = 0, z = 0;

cin >> x >> y >> z; // node x to node y has weight z

a[x][0]++;

a[x][a[x][0]] = y;

w[x][y] = z;

// for undirected graph

a[x][0]++;

a[y][a[y][0]] = x;

w[y][x] = z;

}

ll s = 0, e = 0;

cin >> s >> e; // s: start, e: end

SPFA(s);

cout << d[e] << endl;

return 0;

}

void SPFA(ll v0)

{

ll t, h, u, v;

for (ll i = 0; i < 1001; i++)

d[i] = INT\_MAX;

for (ll i = 0; i < 1001; i++)

f[i] = false;

d[v0] = 0;

h = 0;

t = 1;

q[1] = v0;

f[v0] = true;

while (h != t)

{

h++;

if (h > 3000)

h = 1;

u = q[h];

for (ll j = 1; j <= a[u][0]; j++)

{

v = a[u][j];

if (d[u] + w[u][v] < d[v]) // change to > if calculating longest path

{

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

if (!f[v])

{

t++;

if (t > 3000)

t = 1;

q[t] = v;

f[v] = true;

}

}

}

f[u] = false;

}

}

**6.3 Floyd-Warshall algorithm – shortest path of all pairs O(n^3)**

// map[i][j]=infinity at start

void floyd()

{

for (ll k=1; k<=n; k++)

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

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

if (i!=j && j!=k && i!=k)

if (map[i][k]+map[k][j]<map[i][j])

map[i][j]=map[i][k]+map[k][j];}

**6.4 Prims- Hasnat**

typedef pair<ll,pair<ll,ll>> pairUV;

map<ll, bool> visited;

map<ll, vector<pair<ll, ll>>> adj;

void Prims() {

ll sum = 0, c = 0;

vector<pairUV> ans;

priority\_queue<pairUV, vector<pairUV>, greater<pairUV>> pq;

pq.push({0, {1, -1}});

while (!pq.empty()) {

pairUV k = pq.top();

pq.pop();

ll u = k.second.first;

ll v = k.second.second;

ll wt = k.first;

if (visited[u])

continue;

sum += wt;

visited[u] = 1;

if (v != -1)

ans.pb({wt, {u, v}});

for (auto it : adj[u]) {

ll adjNode = it.first;

ll adjwt = it.second;

if (!visited[adjNode])

pq.push({adjwt, {adjNode, u}});

}

}

}

**6.5 Prims — minimum spanning tree o(ElogV)-Rizu Bhai**

ll d[1001] = {0};

bool v[1001] = {0};

ll a[1001][1001] = {0};

ll main(void){

ll n = 0;

cin >> n;

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

{

ll x = 0, y = 0, z = 0;

cin >> x >> y >> z;

a[x][y] = z;

}

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

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

if (a[i][j] == 0)

a[i][j] =INT\_MAX;

cout << prim(1, n) << endl;}

ll prim(ll u, ll n){

ll mst = 0, k;

for (ll i = 0; i < d.length; i++)

d[i] =INT\_MAX;

for (ll i = 0; i < v.length; i++)

v[i] = false;

d[u] = 0;

ll i = u;

while (i != 0){

v[i] = true;

k = 0;

mst += d[i];

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

if (!v[j])

{

if (a[i][j] < d[j])

d[j] = a[i][j];

if (d[j] < d[k])

k = j;

}

i = k;}

return mst;}

**6.6 Kruskal**

#include<bits/stdc++.h>

#define ll long long int

using namespace std;

ll n,e;

class DSU{

ll\* parent;

ll\* \_size;

public:

DSU(ll n){

parent = new ll[n+1];

\_size = new ll[n+1];

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

parent[i]=i;

\_size[i]=1;

}

}

ll find\_set(ll x){

if(x==parent[x]) return x;

ll y=find\_set(parent[x]);

parent[x]=y;

return y;

}

void Union(ll x, ll y){

ll rx=find\_set(x);

ll ry=find\_set(y);

if(rx==ry) return;

if(\_size[rx]<=\_size[ry]){

parent[rx]=parent[ry];

\_size[ry]+=\_size[rx];

}

else{

parent[ry]=parent[rx];

\_size[rx]+=\_size[ry];

}

}

~DSU(){

delete parent;

delete \_size;

}

};

ll Kruskal(pair<ll,pair<ll,ll>>edges[]){

DSU d(n);

sort(edges,edges+n+1);

ll weight=0;

for(ll i=0;i<e;i++){

ll w=edges[i].first;

ll u=edges[i].second.first;

ll v=edges[i].second.second;

if(d.find\_set(u)!=d.find\_set(v)){

weight+=w;

d.Union(u,v);

}

}

return weight;

}

int main(){

cin>>n>>e;

pair<ll,pair<ll,ll>>edges[e];

for(ll i=0;i<e;i++){

ll u,v,w; cin>>u>>v>>w;

edges[i].first=w;

edges[i].second.first=u;

edges[i].second.second=v;

}

ll ans=Kruskal(edges);

cout<<ans<<"\n";

}

**6.7 DSU:**

#For every i, set parent[i]=I ans size[i]=1

ll find\_set(ll x){

if(parent[x]==x) return x;

ll y=find\_set(parent[x]);

parent[x]=y;

return y;

}

void Union(ll x, ll y){

x=find\_set(x); y=find\_set(y);

if(x==y) return;

if(Size[x]>Size[y]) swap(x,y);

parent[x]=y;

Size[y]+=Size[x];

}

**6.7 Topological sort:**

// Find any solution of topological sort.

#include <iostream>

using namespace std;

ll f[100] = {0}, ans[100] = {0};

bool g[100][100] = {0}, v[100] = {0};

ll n = 0, m = 0;

void dfs(ll k){

ll i = 0;

v[k] = true;

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

if (g[k][i] && !v[i])

dfs(i);

m++;

ans[m] = k;}

ll main(void){

cin >> n >> m;

for (ll i = 1; i <= m; i++)

{

ll x = 0, y = 0;

cin >> x >> y;

g[y][x] = true;

}

m = 0;

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

if (!v[i])

dfs(i);

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

cout << ans[i] << endl;

return 0;}

**6.8 Dijkstra**

map<ll, vector<pair<ll, ll>>> m;

map<ll, ll> dist;

#define pairi pair<ll, ll>

void dijkstra(ll src, ll n) {

priority\_queue<pairi, vector<pairi>, greater<pairi>> pq;

pq.push({0, src});

dist[src] = 0;

vector<ll> dis(n, inf);

dis[src] = 0;

while (!pq.empty()) {

ll u = pq.top().second;

pq.pop();

for (ll i = 0; i < m[u].size(); i++) {

ll wt = m[u][i].second;

ll v = m[u][i].first;

if (dis[v] > dis[u] + wt) {

dis[v] = dis[u] + wt;

pq.push({dis[v], v});

dist[v] = dis[u] + wt;

}

}

}

}

**6.9 Rerooting:**

map<ll, vector<ll>> m;

ll dp[1000001], dp1[1000001], sub[1000001], n;

void dfs(ll x, ll parent) {

dp[x] = 0;

sub[x] = 1;

for (ll i = 0; i < m[x].size(); i++) {

if (m[x][i] != parent) {

dfs(m[x][i], x);

sub[x] += sub[m[x][i]];

dp[x] += dp[m[x][i]] + sub[m[x][i]];

}

}

}

void dfs1(ll x, ll parent, ll carry) {

dp1[x] = dp[x] + carry;

sub[x] = 1;

for (ll i = 0; i < m[x].size(); i++) {

if (m[x][i] != parent) {

ll parent\_dp = dp1[x];

parent\_dp = dp[m[x][i]] + sub[m[x][i]];

ll parent\_sub = (n - sub[m[x][i]]);

ll new\_carry = parent\_dp + parent\_sub;

dfs1(m[x][i], x, new\_carry);

}

}

}

ll main() {

ll x, y, n;

cin >> n;

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

cin >> x >> y;

m[x].pb(y);

m[y].pb(x);

}

dfs(1, -1);

dfs1(1, -1, 0);

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

cout << i + 1 << " " << dp[i + 1] << "\n";

}

m.clear();

return 0;

}

**6.10 Bipartite Graph Test:**

bool dfs(int v, int c)

{

vis[v]=1;

col[v]=c;

for(int child : ar[v]){

if(vis[child]==0){

if(dfs(child,c^1)==false)

return false;

}

else

if(col[v]==col[child])

return false;

}

return true;

}

**7. Range Quarey:**

**7.1 Segment Tree:**

vector<ll> v(2\*1e5 +5),seg(4\*1e5 + 5);

void build(ll ti, ll low, ll high){

if (high == low){

seg[ti] = v[low];

return;

}

ll mid = (low + high) / 2;

build(2 \* ti + 1, low, mid);

build(2 \* ti + 2, mid + 1, high);

seg[ti] = seg[2\*ti+1]+seg[ti\*2+2];

}

//tree left, tree right, query left, query right, index

ll findValue(ll tl, ll tr, ll ql, ll qr, ll ti){

if (tl > qr or tr < ql)

return 0;(sum, xor)

// return INT\_MAX;(min)

// return INT\_MIN;(max)

if (tl >= ql and tr <= qr)

return seg[ti];

ll mid = (tl + tr) / 2;

ll l = findValue(tl, mid, ql, qr, 2 \* ti + 1);

ll r = findValue(mid + 1, tr, ql, qr, 2 \* ti + 2);

return l + r;(sum)

// return min(l,r);

// return max(l,r);

}

void update(ll ti, ll low, ll high, ll id, ll val){

if (id > high or id < low)

return;

if (id == high and high == low){

seg[ti] = val;

return;

}

ll mid = (low + high) / 2;

update(2 \* ti + 1, low, mid, id, val);

update(2 \* ti + 2, mid + 1, high, id, val);

seg[ti] = (seg[2 \* ti + 1] + seg[ti \* 2 + 2]);

}

**8 Game Theory:**

**8.1 Nim Game:**

The current player has a winning strategy if and only if the xor-sum of the pile sizes is non-zero.

**8.2 Miser Nim:**

-Last player to remove stones loses.

-Winning state if xor-sum of pile sizes is non-zero.

-Exception: Each pile has one stone only.

-Winning strategy: If there is only one pile of size greater than one,take all or all but one from that pile leaving an odd number one-size piles. Otherwise, same as normal nim.

**8.3 Grundy’s Game:**

The starting configuration is a single heap of objects. The two players take turn splitting a single heap into two heaps of different sizes. The player who can't make a move loses./ **In each turn, a player can pick any pile and divide it into two unequal piles.**

**If a player cannot do so, he/she loses the game.**

int mex(vector<int> v) {

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

int ret = 0;

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

if(v[i] == ret) ++ret;

else if(v[i] > ret) break;

}

return ret;

}

const int N = 1e3 + 7;

int dp[N];

int g(int n) {

if(n == 0) return 0;

if(dp[n] != -1) return dp[n];

vector<int> gsub;

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

int cur = g(i) xor g(n-i);

gsub.push\_back(cur);

}

dp[n] = mex(gsub);

return dp[n];

}

int main() {

memset(dp, -1, sizeof dp);

int n;

while(cin >> n) {

if(g(n) > 0) cout << "First\n";

else cout << "Second\n";

}

}

**8.4 Again Stone Game:**

**Alice and Bob are playing a stone game. Initially there are n piles of stones and each pile contains some stone. Alice stars the game and they alternate moves. In each move, a player has to select any pile and should remove at least one and no more than half stones from that pile. So, for example if a pile contains 10 stones, then a player can take at least 1 and at most 5 stones from that pile. If a pile contains 7 stones; at most 3 stones from that pile can be removed.**

bool t[N];

ll mex(const vector<ll> &grd)

{

for(auto it : grd)

{

t[it]=true;

}

ll res=0;

while(t[res]) res++;

for(auto it : grd)

{

t[it]=false;

}

return res;

}

ll dp[N];

ll g(ll n)

{

if(n<=1) return 0;

ll &ret=dp[n];

if(ret!=-1) return ret;

vector<ll> grd;

for(int i=1;i<=n/2;i++)

{

ll x=g(n-i);

// dbg3(i,n-i,x);

grd.push\_back(x);

}

ll ans=mex(grd);

return ret=ans;

}

ll get\_g(ll n)

{

if(n<2) return 0;

if(n%2==0) return n/2;

return get\_g(n/2);

}

void solve()

{

ll n;

cin>>n;

ll ans=0;

loop(i,0,n)

{

ll x;

cin>>x;

ll p=get\_g(x);

ans^=p;

// dbg1(p)

}

if(ans)

{

cout<<"Alice"<<endl;

}

else

cout<<"Bob"<<endl;

}

**9 Extra**

**9.1 Ordered Set:**

#include<ext/pb\_ds/assoc\_container.hpp>

#include<ext/pb\_ds/tree\_policy.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

typedef tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> pbds; // find\_by\_order, order\_of\_key

// finding kth element - 4th query

\*A.find\_by\_order(0)--- index 0 er value

// finding number of elements smaller than X

A.order\_of\_key(6) --- 6 er smaller kotogulo elements

**9.2 Multiply Large Numbers represented as Strings**

#include <bits/stdc++.h>

using namespace std;

// Multiplies str1 and str2, and prints result.

string multiply(string num1, string num2){

int len1 = num1.size();

int len2 = num2.size();

if (len1 == 0 || len2 == 0)

return "0";

// will keep the result number in vector

// in reverse order

vector<int> result(len1 + len2, 0);

// Below two indexes are used to find positions

// in result.

int i\_n1 = 0;

int i\_n2 = 0;

// Go from right to left in num1

for (int i = len1 - 1; i >= 0; i--){

int carry = 0;

int n1 = num1[i] - '0';

// To shift position to left after every

// multiplication of a digit in num2

i\_n2 = 0;

// Go from right to left in num2

for (int j = len2 - 1; j >= 0; j--){

// Take current digit of second number

int n2 = num2[j] - '0';

// Multiply with current digit of first number

// and add result to previously stored result

// at current position.

int sum = n1 \* n2 + result[i\_n1 + i\_n2] + carry;

// Carry for next iteration

carry = sum / 10;

// Store result

result[i\_n1 + i\_n2] = sum % 10;

i\_n2++;

}

// store carry in next cell

if (carry > 0)

result[i\_n1 + i\_n2] += carry;

// To shift position to left after every

// multiplication of a digit in num1.

i\_n1++;

}

// ignore '0's from the right

int i = result.size() - 1;

while (i >= 0 && result[i] == 0)

i--;

// If all were '0's - means either both or

// one of num1 or num2 were '0'

if (i == -1)

return "0";

// generate the result string

string s = "";

while (i >= 0)

s += std::to\_string(result[i--]);

return s;

}

// Driver code

int main(){

string str1 = "1235421415454545454545454544";

string str2 = "1714546546546545454544548544544545";

cin >> str1 >> str2;

if ((str1.at(0) == '-' || str2.at(0) == '-') &&

(str1.at(0) != '-' || str2.at(0) != '-'))

cout << "-";

if (str1.at(0) == '-')

str1 = str1.substr(1);

if (str2.at(0) == '-')

str2 = str2.substr(1);

cout << multiply(str1, str2);

return 0;

}

**9.3 Sum of two large numbers:**

#include <bits/stdc++.h>

using namespace std;

// Function for finding sum of larger numbers

string findSum(string str1, string str2){

// Before proceeding further, make sure length

// of str2 is larger.

if (str1.length() > str2.length())

swap(str1, str2);

// Take an empty string for storing result

string str = "";

// Calculate length of both string

int n1 = str1.length(), n2 = str2.length();

int diff = n2 - n1;

// Initially take carry zero

int carry = 0;

// Traverse from end of both strings

for (int i = n1 - 1; i >= 0; i--){

// Do school mathematics, compute sum of

// current digits and carry

int sum = ((str1[i] - '0') + (str2[i + diff] - '0') + carry);

str.push\_back(sum % 10 + '0');

carry = sum / 10;}

// Add remaining digits of str2[]

for (int i = n2 - n1 - 1; i >= 0; i--){

int sum = ((str2[i] - '0') + carry);

str.push\_back(sum % 10 + '0');

carry = sum / 10;}

// Add remaining carry

if (carry)

str.push\_back(carry + '0');

// reverse resultant string

reverse(str.begin(), str.end());

return str;}

// Driver code

int main(){

string str1 = "12";

string str2 = "198111";

cin >> str1 >> str2;

cout << findSum(str1, str2);

return 0;}

**9.4 Divide large number represented as string:**

#include <bits/stdc++.h>

using namespace std;

// A function to perform division of large numbers

string longDivision(string number, int divisor){

    // As result can be very large store it in string

    string ans;

    // Find prefix of number that is larger

    // than divisor.

    int idx = 0;

    int temp = number[idx] - '0';

    while (temp < divisor)

        temp = temp \* 10 + (number[++idx] - '0');

    // Repeatedly divide divisor with temp. After

    // every division, update temp to include one

    // more digit.

    while (number.size() > idx) {

        // Store result in answer i.e. temp / divisor

        ans += (temp / divisor) + '0';

        // Take next digit of number

        temp = (temp % divisor) \* 10 + number[++idx] - '0';}

    // If divisor is greater than number

    if (ans.length() == 0)

        return "0";

    // else return ans

    return ans;}

// Driver program to test longDivision()

int main(){

    string number = "1248163264128256512";

    int divisor = 125;

    cout << longDivision(number, divisor);

    return 0;}

**9.5 Subtraction of two large numbers:**

#include <bits/stdc++.h>

using namespace std;

// Returns true if str1 is smaller than str2,

// else false.

bool isSmaller(string str1, string str2){

    // Calculate lengths of both string

    int n1 = str1.length(), n2 = str2.length();

    if (n1 < n2)

        return true;

    if (n2 < n1)

        return false;

    for (int i = 0; i < n1; i++) {

        if (str1[i] < str2[i])

            return true;

        else if (str1[i] > str2[i])

            return false;}

    return false;}

// Function for finding difference of larger numbers

string findDiff(string str1, string str2){

    // Before proceeding further, make sure str1

    // is not smaller

    if (isSmaller(str1, str2))

        swap(str1, str2);

    // Take an empty string for storing result

    string str = "";

    // Calculate lengths of both string

    int n1 = str1.length(), n2 = str2.length();

    int diff = n1 - n2;

    // Initially take carry zero

    int carry = 0;

    // Traverse from end of both strings

    for (int i = n2 - 1; i >= 0; i--) {

        // Do school mathematics, compute difference of

        // current digits and carry

        int sub = ((str1[i + diff] - '0') - (str2[i] - '0')

                   - carry);

        if (sub < 0) {

            sub = sub + 10;

            carry = 1;}

        else

            carry = 0;

        str.push\_back(sub + '0');}

    // subtract remaining digits of str1[]

    for (int i = n1 - n2 - 1; i >= 0; i--) {

        if (str1[i] == '0' && carry) {

            str.push\_back('9');

            continue;}

        int sub = ((str1[i] - '0') - carry);

        if (i > 0 || sub > 0) // remove preceding 0's

            str.push\_back(sub + '0');

        carry = 0;}

    // reverse resultant string

    reverse(str.begin(), str.end());

    return str;}

// Driver code

int main(){

    string str1 = "88";

    string str2 = "1079";

    // Function call

    cout << findDiff(str1, str2);

    return 0;}