COMPETITIVE PROGRAMMING NOTEBOOK

ISRAEL - Se der bom não pergunta como

March 18, 2022

HEADER

```cpp
#include <bits/stdc++.h>

using namespace std;

#define all(x) x.begin(), x.end()
#define sz(x) (int) x.size()
#define pb push_back
#define snd second
#define fst first

typedef long long int ll;
typedef vector <int> vi;
typedef pair <int,int> ii;
typedef pair<ii,int> iii;
const int mod = 1e9+7;
const ll INF = 1e18;
const int N = 2e5 + 5;
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL);
    return 0;
}

int dx[] = {-1, -1, -1, 0, 0, 1, 1, 1};
int dy[] = {-1, 0, 1, -1, 1, -1, 0, 1};
```

Listing 1: HEADER

1 TRICKS

- Sum-Xor property: $a + b = a \oplus b + 2(a\&b)$. Extended Version with two equations: $a + b = a|b + a\&b$ AND $a \oplus b = (a|b) - (a\&b)$
- $GCD(F(i), F(j)) = F(gcd(i, j))$ and $F(i)$ is the i'th fibonacci term
- Any even number greater than 2 can be split into two prime numbers
- Upto $(10)^{12}$ there can be at most 300 non-prime numbers between any two consecutive prime numbers

2 MATH

NCR + EXPMOD

```
int inv(int a){ // return the inverse modular multiplicative of a
    return pwr(a, mod - 2);
}

int add(int a, int b){
    a += b;
    if(a >= mod) a -= mod;
    return a;
}
int mul(int a, int b){
    return 1ll * a * b % mod;
}
int pwr(int a, int b){
    int r = 1;
    for(; b; b>>=1, a = mul(a,a))
        if(b&1)
            r = mul(r, a);
    return r;
}
int fact[N], ifact[N];
void init(){
    fact[0] = 1;
    for(int i = 1; i < N; i++)
        fact[i] = mul(i, fact[i-1]);
    ifact[N-1] = pwr(fact[N-1], mod-2);
    for(int i = N-2; i >= 0; i--)
        ifact[i] = mul(ifact[i+1], i+1);
}
int ncr(int n, int r){
    if(n < r) return 0;
    return mul(fact[n], mul(ifact[r], ifact[n-r]));
}
init();
//CATALAN NUMBERS
int catalan(int n){
    return mul(invi[n+1], C(2*n,n));
}
// where invi[n+1] is the inverse modular multiplicative
```

Listing 2: NCR + EXPMOD

isPrime

```cpp
bool isPrime (ll n){
    if (n < 0)
        return isPrime (-n);
    if (n < 5 || n % 2 == 0 || n % 3 == 0)
        return (n == 2 || n == 3);
    ll maxP = sqrt (n) + 2;
    for (ll p = 5; p < maxP; p += 6)
        if (n % p == 0 || n % (p + 2) == 0)
            return false;
    return true;
}
```

Listing 3: isPrime

Euler's totient function

```cpp
//phi[n] : counts the number of integers between 1 and n inclusive, which are coprime to n
vi EulerTot(int n){
    vi phi(n + 1);
    for(int i = 1; i <= n; i++)
        phi[i] = i;
    for(int i = 2; i <= n; i++){
        if(phi[i] == i){
            phi[i] = i - 1;
            for(int j = 2 * i; j <= n; j += i){
                phi[j] = ((phi[j]*(i-1)) / i);
            }
        }
    }
    return phi;
}
```

Listing 4: Euler

EULER DIVISOR SUM PROPERTY:

$$\sum_{d|n} \phi(d) = n$$

$\sum LCM(1, N), LCM(2, N)...LCM(N, N)$

```cpp
// final[i] = sum of all pairs of lcm up to i
EulerTot();
vi f(N, 0), final(N,0);
for(ll i = 1; i < N; i++){
    for(ll j = i; j < N; j += i){
        f[j] += i*phi[i];
    }
}
for(ll i = 1; i < N; i++){
    ll now = (((f[i] + 1)>> 1) * i);
    // now = sum of lcm(1,i) + lcm(2,i) + ... + lcm(i,i)
    final[i] = final[i-1] + now - i;
}
```

Listing 5: LCM SUM

All Prime Factors and His Power

```
vector<ii> p[N];
void __sieve(){
    for(int i = 2; i < N; i++){
        if(p[i].empty()){
            for(int j = i; j < N; j+=i){
                int q = j;
                ii temp = {i,0};
                while(q % i == 0){
                        q /= i, temp.snd++;
                }
                p[j].pb(temp);
            }
        }
    }
}
```

Listing 6: Prime fact Sieve

Karatsuba FAST MULTIPLICATION

```
// O(n^1.59)
int getSize(long num){
    int count = 0;
    while (num > 0){
        count++;
        num /= 10;
    }
    return count;
}
long karatsuba(long X, long Y){
    if (X < 10 && Y < 10) return X * Y;
    int size = fmax(getSize(X), getSize(Y));
    int n = (int)ceil(size / 2.0);
    long p = (long)pow(10, n);
    long a = (long)floor(X / (double)p);
    long b = X % p;
    long c = (long)floor(Y / (double)p);
    long d = Y % p;
    long ac = karatsuba(a, c);
    long bd = karatsuba(b, d);
    long e = karatsuba(a + b, c + d) - ac - bd;
    return (long)(pow(10 * 1L, 2 * n) * ac + pow(10 * 1L, n) * e +
    bd);
}
```

Listing 7: Karatsuba FAST MULTIPLICATION

PRIME FACTORIZATION - POLLARD RHO

```cpp
ull modmul(ull a, ull b, ull M) {
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
ull modpow(ull b, ull e, ull mod) {
    ull ans = 1;
    for (; e; b = modmul(b, b, mod), e /= 2)
        if (e & 1) ans = modmul(ans, b, mod);
    return ans;
}
bool isPrime(ull n) {
    if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
    ull A[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022},
        s = __builtin_ctzll(n-1), d = n >> s;
    for (ull a : A) {   // ^ count trailing zeroes
        ull p = modpow(a%n, d, n), i = s;
        while (p != 1 && p != n - 1 && a % n && i--)
            p = modmul(p, p, n);
        if (p != n-1 && i != s) return 0;
    }
    return 1;
}
ull pollard(ull n) {
    auto f = [n](ull x) { return modmul(x, x, n) + 1; };
    ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
    while (t++ % 40 || __gcd(prd, n) == 1) {
        if (x == y) x = ++i, y = f(x);
        if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
        x = f(x), y = f(f(y));
    }
    return __gcd(prd, n);
}
vector<ull> factor(ull n) {
    if (n == 1) return {};
    if (isPrime(n)) return {n};
    ull x = pollard(n);
    auto l = factor(x), r = factor(n / x);
    l.insert(l.end(), all(r));
    return l;
}
```

Listing 8: Prime factorization

MATRIX EXP - LINEAR RECURRENCE

```cpp

const int N = 3; // No of terms in the Recurrence Relation.
void multiply(ll A[N][N], ll B[N][N]){
    ll R[N][N];
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            R[i][j] = 0;
            for (int k = 0; k < N; k++){
                R[i][j] = (R[i][j] + A[i][k] * B[k][j]) % mod;
            }
        }
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            A[i][j] = R[i][j];
        }
    }
}
// Raise matrix A to the power of n in O(log n).
void power_matrix (ll A[N][N], int n){
    ll B[N][N];
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            B[i][j] = A[i][j];
        }
    }
    n = n - 1;
    while (n > 0){ // A = A * A ^ (n - 1).
        if (n & 1)
            multiply (A, B);
        multiply (B,B);
        n = n >> 1;
    }
}
// A = Coefficient Matrix, B = Base Matrix.
// It returns the nth term of the recurrence
ll solve_recurrence (ll A[N][N], ll B[N][1], int n){
    if (n < N) //Base Cases.
        return B[N - 1 - n][0];
    power_matrix (A, n - N + 1);      // A = A ^ (n - N + 1).
    ll result = 0;
    for(int i = 0; i < N; i++)
        result = (result + A[0][i] * B[i][0]) % mod;
    return result;
}
/*
    The recurrence relation used here is: -
    R(n) = 2 * R(n-1) + R(n-2) + 3 * R(n-3).
    Base Cases: R(0) = 1, R(1) = 2, R(2) = 3.
*/
ll A[N][N] = {{2, 1, 3}, {1, 0, 0}, {0, 1, 0}}; // Forming the Coefficient Matrix
ll B[N][1] = {{3}, {2}, {1}}; //Forming the Base Matrix
ll R_n = solve_recurrence (A, B, n); // n term
```

Listing 9: MATRIX EXP - LINEAR RECURRENCE

3 DATA STRUCTURE

BIT

```
int bit[N];
int query(int i){
    int sum = 0;
    for(i++; i > 0; i -= i&(-i))
        sum += bit[i];
    return sum;
}
void update(int i, int x){
    for(i++; i < N; i += i&(-i))
        bit[i] += x;
}
int query(int l, int r){
    return query(r) - query(l-1);
}
```

Listing 10: BIT

SEGTREE

```
int seg[4*N], v[N];
void build(int cur, int l, int r){
    if(l == r)
        seg[cur] = v[l];
    else{
        int mid = (l+r)/2;
        build(2*cur, l, mid);
        build(2*cur+1, mid+1, r);
        seg[cur] = min(seg[2*cur], seg[2*cur+1]);
    }
}
int query(int cur, int l, int r, int a, int b){
    if(l > b or r < a)return inf;
    if(l >= a and r <= b)return seg[cur];
    int mid = (l+r)/2;
    return min(query(2*cur, l, mid, a, b),
               query(2*cur+1, mid+1, r, a, b));
}
void update(int cur, int l, int r, int j, int x){
    if(l > j or r < j)
        return;
    if(l == r)
        seg[cur] += x;
    else{
        int mid = (l+r)/2;
        update(2*cur, l, mid, j, x);
        update(2*cur+1, mid+1, r, j, x);
        seg[cur] = min(seg[2*cur], seg[2*cur+1]);
    }
}
```

Listing 11: SEGTREE

SEGTREE-LAZY PROPAGATION

```
int v[N], st[4*N], lz[4*N];
void push(int id, int l, int r){
    if(lz[id]){
        st[id] += lz[id]; // += (r - l + 1)*lz[id] ?
        if(l!=r){
            lz[2*id] += lz[id];
            lz[2*id+1] += lz[id];
        }
        lz[id] = 0;
    }
}
int query(int id, int l, int r, int i, int j){
    push(id, l, r);
    if(r < i or l > j) return 1e9;
    if(l >= i and r <= j)
        return st[id];
    return min(query(2*id, l, (l+r)/2, i, j) ,
            query(2*id+1, (l+r)/2+1, r, i, j));
}
void update(int id, int l, int r, int i, int j, int x){
    push(id, l, r);
    if(r < i or l > j)
        return ;
    if(l >= i and r <= j) {
        lz[id] += x;
        push(id, l, r);
    } else {
        update(2*id, l, (l+r)/2, i, j, x);
        update(2*id+1, (l+r)/2+1, r, i, j, x);
        st[id] = min(st[2*id], st[2*id+1]);
    }
}
void build(int id, int l, int r){
    if(l == r)
        st[id] = v[l];
    else{
        build(2*id, l, (l+r)/2);
        build(2*id+1, (l+r)/2+1, r);
        st[id] = min(st[2*id], st[2*id+1]);
    }
}
```

Listing 12: SEG + LAZY PROP

4 GRAPH

Ford Fulkerson

```
// O(|FLOW|*(n+m))
struct ed{
    int to, f, c;
}ed[N];
vi adj[N];
int cur = 0, tempo, seen[N];
void add_ed(int a, int b, int cp, int rc){ // rc = capacity of
    reverse edge (0 if normal graph)
    ed[cur].to = b, ed[cur].f = 0, ed[cur].c = cp;
    adj[a].pb(cur++);
    ed[cur].to = a, ed[cur].f = 0, ed[cur].c = rc;
    adj[b].pb(cur++);
}
int dfs(int s, int t, int f){
    if(s == t) return f;
    seen[s] = tempo;
    for(int e : adj[s]){
        if(seen[ed[e].to] < tempo and ed[e].c - ed[e].f > 0){
            if(int a = dfs(ed[e].to,t, min(f, ed[e].c - ed[e].f))){
                ed[e].f += a;
                ed[e ^ 1].f -= a;
                return a;
            }
        }
    }
    return 0;
}
int ford_fulkerson(int s, int t){
    int flow = 0;
    tempo = 1;
    while(int a = dfs(s, t, INT_MAX)){
        flow += a;
        tempo++;
    }
    return flow;
}
// main
add_ed(a, b, c, 0);
ford_fulkerson(s, t)
```

Listing 13: Ford Fulkerson

Edmonds Karp

```
// O(V*E^2)
vi adj[N];
int capacity[N][N], flowpassed[N][N], parent[N], pathcap[N], n;
int bfs(int s, int t){
    memset(parent, -1, sizeof(parent));
    memset(pathcap, 0, sizeof(pathcap));
    queue<int> q;
    q.push(s);
    parent[s] = -2;
    pathcap[s] = mod;
    while(!q.empty()){
        int now = q.front(); q.pop();
        for(auto i:adj[now]){
            if(parent[i] == -1 and capacity[now][i] > flowpassed[now][i]){
                parent[i] = now;
                pathcap[i] = min(pathcap[now], capacity[now][i] - flowpassed[now][i]);
                if(i == t){
                    return pathcap[t];
                }
                q.push(i);
            }
        }
    }
    return 0;
}
int maxflow(int s, int t) {
    int maxflow = 0;
    while(true){
        int flow = bfs(s, t);
        if(flow == 0) break;
        maxflow += flow;
        int now = t;
        while(now != s){
            int prev = parent[now];
            flowpassed[prev][now] += flow;
            flowpassed[now][prev] -= flow;
            now = prev;
        }
    }
    return maxflow;
}
// MAIN
memset(capacity, 0, sizeof(capacity));
memset(flowpassed, 0, sizeof(flowpassed));
for(int i = 1; i <= n; i++)adj[i].clear();
capacity[x][y] += w;
capacity[y][x] += w;
adj[x].pb(y);
adj[y].pb(x);
maxflow(s, t)
```

Listing 14: Edmonds Karp

LCA

```
int l, timer;
vi adj[N];
vector<int> tin, tout;
vector<vector<int>> up;
void dfs(int v, int p){
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++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) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

Listing 15: LCA

LCA + RMQ

```
// weight[i] : edge going from i the father of i from root
vector<ii> adj[N];
int up[N][22], maxi[N][22], level[N], l = 21, weight[N];
void dfs(int v, int pp, int h){
    level[v] = h;
    up[v][0] = pp;
    if(pp != -1){
        //maxi[v][0] = max(weight[v], weight[pp]);
        maxi[v][0] = weight[v];
    }
    for(int i = 1; i < l; i++){
        up[v][i] = up[up[v][i-1]][i-1];
        maxi[v][i] = max(maxi[v][i-1], maxi[up[v][i-1]][i-1]);
    }
    for(auto u : adj[v]){
        if(u.fst != pp)
            dfs(u.fst, v, h + 1);
    }
}
int get_max(int u, int v){ // lca
    int ans = INT_MIN;
    if(level[u] > level[v]) swap(u,v);
    for(int i = l-1; i >= 0; i--){
        if(up[v][i] != -1 and level[up[v][i]] >= level[u]){
            ans = max(ans, maxi[v][i]);
            v = up[v][i];
        }
    }
    if(u != v){
        for(int i = l-1; i >= 0; i--){
            if(up[u][i] != up[v][i]){
                ans = max({ans, maxi[v][i], maxi[u][i]});
                u = up[u][i];
                v = up[v][i];
            }
        }
        ans = max({ans, maxi[v][0], maxi[u][0]});
    }
    return ans;
}
```

Listing 16: LCA + RMQ

Tarjan

```
int foundat = 1, disc[N], low[N];
vi adj[N];
int comp = 0;
bool onstack[N];
vector<vi> scc;

void tarjan(int u){
    static stack<int> st;
    disc[u] = low[u] = foundat++;
    st.push(u);
    onstack[u] = true;
    for(auto i:adj[u]){
        if(disc[i] == -1){
            tarjan(i);
            low[u] = min(low[u], low[i]);
        }
        else if(onstack[i])
            low[u] = min(low[u], disc[i]);
    }
    if(disc[u] == low[u]){
        vi scctem;
        while(true){
            int v = st.top();
            st.pop();
            onstack[v] = false;
            scctem.pb(v);
            if(u == v)
                break;
        }
        comp++;
        scc.pb(scctem);
    }
}
// main
memset(disc, -1, sizeof(disc));
for(int i = 1; i <= n; i++){
    if(disc[i] == -1)
        tarjan(i);
}
```

Listing 17: Tarjan

PRIM

```
ll prim(){
    int see[MAX];
    memset(see, 0, sizeof(see));
    see[0] = true;
    priority_queue<ii> pq;
    ll ans = 0;
    for(auto j : adj[0]) pq.push({-j.snd , -j.fst});
    while(!pq.empty()){
        int u = -pq.top().snd , w = -pq.top().fst;
        pq.pop();
        if(!see[u]){
            ans += w;
            see[u] = 1;
            for(auto j : adj[u])
                if(!see[j.fst])
                    pq.push({-j.snd , -j.fst});
        }
    }
    return ans;
}
```

Listing 18: PRIM

FLOYD WARSHALL

```
void floyd(){
    for(int k = 1; k <= n; k++){
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= n; j++){
                if(adj[i][k] + adj[k][j] < adj[i][j])
                    adj[i][j] = adj[i][k] + adj[k][j];
            }
        }
    }
}
```

Listing 19: FLOYD WARSHALL

MAXIMUM BIPARTITE MATCHING

```cpp
//Kuhn's Algorithm for Maximum Bipartite Matching
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;
bool try_kuhn(int v) {
    if (used[v])
        return false;
    used[v] = true;
    for (int to : g[v]) {
        if (mt[to] == -1 || try_kuhn(mt[to])) {
            mt[to] = v;
            return true;
        }
    }
    return false;
}
int main(){
    // read the graph
    mt.assign(k, -1);
    vector<bool> used1(n, false);
    for (int v = 0; v < n; ++v) {
        for (int to : g[v]) {
            if (mt[to] == -1) {
                mt[to] = v;
                used1[v] = true;
                break;
            }
        }
    }
    for (int v = 0; v < n; ++v) {
        if (used1[v])
            continue;
        used.assign(n, false);
        try_kuhn(v);
    }

    for (int i = 0; i < k; ++i)
        if (mt[i] != -1)
            printf("%d %d\n", mt[i] + 1, i + 1);
}
```

Listing 20: Kuhn's Algorithm

5 STRING

PREFIX FUNCTION + KMP

```
vi prefix_function(string s) {
    int n = (int)s.length();
    vi pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
void KMP(string pattern, string text){
    int n = sz(text), m = sz(pattern);
    vi Lps = prefix_function(pattern);
    int i=0, j=0;
    while(i < n){
        if(pattern[j]==text[i]){i++;j++;}
        if(j == m){
            cout<< i - m << "\n";// found pattern
            j = Lps[j - 1];
        }
        else if(i < n && pattern[j] != text[i]){
            if(j == 0)
                i++;
            else
                j = Lps[j - 1];
        }
    }
}
```

Listing 21: PREFIX FUNCTION + KMP

Boths

```
// K = Lexicographically minimal string rotation needed
int rotate(string s){
    s += s;
    int n = sz(s);
    vi f(n,-1);
    int k = 0;
    for(int j = 1; j < n; j++){
        char c = s[j];
        int i = f[j - k - 1];
        while( i != -1 and c != s[k + i + 1]){
            if(c < s[k + i + 1])
                k = j - i - 1;
            i = f[i];
        }
        if(c != s[k + i + 1]){
            if(c < s[k])
                k = j;
            f[j - k] = -1;
        }else{
            f[j - k] = i + 1;
        }
    }
    return k;
}
```

Listing 22: Lexicographically minimal

Zfunction

```
// z[i] = greatest number of characters starting from the position
     that coincide with the first characters of string s
vi zFunction(string s) {
    int n = sz(s);
    vi z(n, 0);
    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;
}
```

Listing 23: Zfunction

6 GEOMETRY

7 MISC

Native function

```
struct Compare{
    bool operator()(ii const& a, ii const& b){
        return 0;
    }
};
priority_queue<ii, vector<ii>, Compare> pq;
```

Listing 24: Native function

Native sort

```
struct Node{
    int x, y, idx;
    Node(int xx, int yy, int ii){x = xx; y = yy; idx = ii;}
    bool operator < (const Node& other){
        if(x == other.x)
            return y > other.y;
        else
            return x < other.x;
    }
};
vector<Node> v;
v.push_back(Node(x,y,i));
sort(v.begin(), v.end());
```

Listing 25: Native sort

STIRLING NUMBERS OF THE FIRST KIND

Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets and is denoted by $S(n, k)$

Figure 1: Stirling table

$s(n,k)$	0	1	2	3	4	5
0	1					
1	0	1				
2	0	1	1			
3	0	2	3	1		
4	0	6	11	6	1	
5	0	24	50	35	10	1
6	0	120	274	225	85	15
7	0	720	1764	1624	735	175
8	0	540	13068	13132	6769	1960
9	0	40320	109584	118124	67284	22449
10	0	362880	1026576	1172700	723680	269325

RECURRENCE RELATION O(n*k):

$$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1)$$

Figure 2: Stirling in nlogn

If you want to compute just one row $s(n,k)$ for all $k \in \{1, 2, \ldots, n\}$, then exploit the generating function:

$s(n,k)$ is the coefficient of x^k in the falling factorial $x(x-1)(x-2)\ldots(x-(n-1))$

So you compute the product $x(x-1)(x-2)\ldots(x-(n-1))$ by recursively computing the first and last $\frac{n}{2}$ products and multiplying them using FFT in $O(n \log n)$. This gives the recurrence:

$T(n) = 2T(\frac{n}{2}) + O(n \log n)$

So, $T(n) = O(n \log^2 n)$

Listings