UNAL ICPC Team Notebook (2024)

Contents

hola mUNdo

1 Data Structures

1.1 DSU

```cpp
const int N = 1e5+5;
int dsu[N];
int cc;

int find (int node){
    if(dsu[node] == -1) return node;
    return dsu[node] = find(dsu[node]);
}

bool connected(int A, int B){
    return find(A)==find(B);
}

void join (int A, int B){
    A = find(A);
    B = find(B);
    dsu[A] = B;
    cc--;
}

memset(dsu, -1, sizeof dsu);
```

1.2 DSU Pesos

```cpp
int parent[MAX];
int rango[MAX];
int n;
void Init( int _n ){
   n = _n;
   for( int i = 0 ; i < n ; ++i ){
       parent[i] = i;
       rango[i] = 0;
   }
}

int Find( int x ){
   if( x == parent[ x ] )
       return x;
   else
       return parent[ x ] = Find( parent[ x ] );
}
void Union( int x , int y ){
   int xRoot = Find( x );
   int yRoot = Find( y );
   if( rango[ xRoot ] > rango[ yRoot ] )
       parent[ yRoot ] = xRoot;
   else{
       parent[ xRoot ] = yRoot;
       if( rango[ xRoot ] == rango[ yRoot ] )
           rango[ yRoot ]++;
   }
}
int countComponents(){
   int c = 0;
   for( int i=0; i<n; i++ )
       if( parent[i] == i )
           c++;
   return c++;
}
vector<int> getRoots(){
   vector<int> v;
   for( int i=0; i<n; i++ )
       if( i == parent[i] )
           v.push_back(i);
   return v;
}

int countNodesInComponent( int root ){
   int c = 0;
   for( int i=0; i<n; i++)
       if( Find(i) == root )
           c++;
   return c++;
}
bool sameComponent( int x, int y ){
   return Find(x) == Find(y);
}
```

2 Graphs

2.1 Strongest Connected components

```
vector<bool> visited; // keeps track of which vertices are already visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when dfs leaves it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongy connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root vertices)
void strongly_connected_components(vector<vector<int>> const& adj,
                                   vector<vector<int>> &components,
                                   vector<vector<int>> &adj_cond) {
    int n = adj.size();
    components.clear(), adj_cond.clear();

    vector<int> order; // will be a sorted list of G's vertices by exit time

    visited.assign(n, false);

    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);

    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);

    visited.assign(n, false);
    reverse(order.begin(), order.end());

    vector<int> roots(n, 0); // gives the root vertex of a vertex's SCC

    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            components.push_back(component);
            int root = *min_element(begin(component), end(component));
            for (auto u : component)
                roots[u] = root;
        }

    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (auto u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(roots[u]);
}
```

2.2 SCC Tarjan

```
struct TarjanScc{
   vector<bool> marked;
   vector<int> id;
   vector<int> low;
   int pre;
   int count;
   stack<int> stck;
   vector<vector<int> >G;

   TarjanScc( vector<vector<int> >g, int V ){
       G=g;
       marked = vector<bool>(V, false);
       stck = stack<int>();
       id= low = vector<int>(V, 0);
       pre=count=0;
       for(int u=0; u<V; u++)
           if( !marked[u] ) dfs(u);
   }

   void dfs( int u ){
       marked[ u ] = true;
       low[ u ] = pre++;
       int min = low[ u ];

       stck.push( u );
       for( int w=0; w<G[u].size(); w++){
           if(  !marked[G[u][w]] ) dfs( G[u][w] );
           if( low[ G[u][w] ] < min ) min = low[ G[u][w] ];
       }
       if( min<low[u] ){
           low[u] = min;
           return;
       }
       int w;
       do{
           w = stck.top();stck.pop();
           id[ w ] = count;
           low[ w ]= G.size();
       }while( w != u );
       count++;
   }

   int getCount() { return count; }


   // are v and w strongly connected?
   bool stronglyConnected(int v, int w) {
       return id[v] == id[w];
   }

   // in which strongly connected component is vertex v?
   int getId(int v) { return id[v]; }
};

//Ejemplo de Uso
int main( ){
   int u, v, N, M, cas, k=0;
   for(cin>>cas; k<cas; k++){
       scanf("%d %d", &N, &M);
       //cin>>N>>M;
       vector<vector<int> >G(N);

       for(int i=0; i < M; i++){
           scanf("%d %d", &u, &v);
           //cin>>u>>v;
           u--;v--;
           G[u].PB(v);
       }

       TarjanScc tscc(G, N);
       //Encontrar cuantos nodos tienen grado de entrada 0
       vector<int>indegree(tscc.getCount(), 0);
       int idu, idv;
       for( u = 0; u < N; u++){
           idu = tscc.getId( u );
           for( v = 0; v < G[u].size(); v++){
               idv = tscc.getId( G[u][v] );

               if( idu!=idv ){
                   indegree[idv]++;
               }
           }
       }
       int res=0;
       for(int i=0; i<indegree.size(); i++){
           if(indegree[i]==0)res++;
       }
       printf("Case %d: %d\n",k+1,res);
   }
return 0;
}
```

2.3 Topological sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
```

```
        if (!visited[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

2.4 Floyd-Warshall

```
//O(n^3)

//inicializar todo en INF previo a la lectura

for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}

//Si se tienen pesos negativos:
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
        }
    }
}

//Pesos reales
if (d[i][k] + d[k][j] < d[i][j] - EPS)
    d[i][j] = d[i][k] + d[k][j];

/*Identificar ciclos negativos:
Si al final del algoritmo d[i][i] es negativo.*/
```

2.5 Dijkstra

```
//O(n^2+m)
for (int i = 1; i <= n; i++) distance[i] = INF;
distance[x] = 0;
q.push({0,x});
while (!q.empty()) {
    int a = q.top().second; q.pop();
    if (processed[a]) continue;
    processed[a] = true;
    for (auto u : adj[a]) {
        int b = u.first, w = u.second;
        if (distance[a]+w < distance[b]) {
            distance[b] = distance[a]+w;
            q.push({-distance[b],b});
        }
    }
}
```

2.6 Shortest Path Fast algorithm

```
//O(nm)
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;

        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false;  // negative cycle
                }
            }
        }
    }
    return true;
}
```

3 Dynamic Programming

3.1 Coin Exchange Problem

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

using namespace std;

// Returns total distinct ways to make sum using n coins of
// different denominations
int count(vector<int>& coins, int n, int sum)
{
    // 2d dp array where n is the number of coin
    // denominations and sum is the target sum
    vector<vector<int> > dp(n + 1, vector<int>(sum + 1, 0));

    // Represents the base case where the target sum is 0,
    // and there is only one way to make change: by not
    // selecting any coin
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= sum; j++) {

            // Add the number of ways to make change without
            // using the current coin,
            dp[i][j] += dp[i - 1][j];

            if ((j - coins[i - 1]) >= 0) {

                // Add the number of ways to make change
                // using the current coin
                dp[i][j] += dp[i][j - coins[i - 1]];
            }
        }
    }
    return dp[n][sum];
}

// Driver Code
int main()
{
    vector<int> coins{ 1, 2, 3 };
    int n = 3;
    int sum = 5;
    cout << count(coins, n, sum);
    return 0;
}
```

4 Flows

4.1 Dinic

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
```

```cpp
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

4.2 MinCost Flow

```cpp
struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

5 Math

5.1 Primes

```
/*
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      101     103     107     109     113     127     131     137     139     149     151     157
           163     167     173     179     181     191     193     197     199     211     223
227     229     233     239     241     251     257     263     269     271     277     281     283
           293     307     311     313     317     331     337     347     349     353     359
367     373     379     383     389     397     401     409     419     421     431     433     439
           443     449     457     461     463     467     479     487     491     499     503
509     521     523     541     547     557     563     569     571     577     587     593     599
           601     607     613     617     619     631     641     643     647     653     659
661     673     677     683     691     701     709     719     727     733     739     743     751
           757     761     769     773     787     797     809     811     821     823     827
```

```
829     839     853     857     859     863     877     881     883     887     907     911     919
            929     937     941     947     953     967     971     977     983     991     997


1009    1013    1019    1021    1031    1033    1039    1049    1051    1061    1063    1069    1087
            1091    1093    1097    1103    1109    1117    1123
1129    1151    1153    1163    1171    1181    1187    1193    1201    1213    1217    1223    1229
            1231    1237    1249    1259    1277    1279    1283
1289    1291    1297    1301    1303    1307    1319    1321    1327    1361    1367    1373    1381
            1399    1409    1423    1427    1429    1433    1439
1447    1451    1453    1459    1471    1481    1483    1487    1489    1493    1499    1511    1523
            1531    1543    1549    1553    1559    1567    1571
1579    1583    1597    1601    1607    1609    1613    1619    1621    1627    1637    1657    1663
            1667    1669    1693    1697    1699    1709    1721
1723    1733    1741    1747    1753    1759    1777    1783    1787    1789    1801    1811    1823
            1831    1847    1861    1867    1871    1873    1877
1879    1889    1901    1907    1913    1931    1933    1949    1951    1973    1979    1987    1993
            1997    1999

*/
```

5.2 Log Utils

```
ln: log()
log base 10: log10()
e: exp()
primos aproximados hasta x: x/ln(x) o x/(ln(x)  1  ,08366)
```

5.3 Line Representation

```
//si b=0: es como si fuera oo o -oo
//si a=0: la fraccion es 0

struct frac{
    ll a,b;
    frac(ll_a, ll_b): a(_a), b(_b){
        if(b<0) a*=-1, b *=-1;
        if(b==0) a = 1;
    }

    bool operator < (frac other){
        return a * other.b < b * other.a;
    }
};

map<frac,frac> mp;
```

5.4 Cribe

```
int cribe[N];

for(int i=2; i < N; i++){
    if(!cribe[i]){
        for(int k=i+i; k<N; k+=i){
            cribe[k] = i;
        }
    }
}
```

5.5 FastCribe

```
#define forr(i,a,b) for(int i=(a); i<(b); i++)
typedef long long ll;
typedef pair<int,int> ii;

#define MAXP 100000     //no necesariamente primo
int criba[MAXP+1];
void crearcriba(){
        int w[] = {4,2,4,2,4,6,2,6};
        for(int p=25;p<=MAXP;p+=10) criba[p]=5;
        for(int p=9;p<=MAXP;p+=6) criba[p]=3;
        for(int p=4;p<=MAXP;p+=2) criba[p]=2;
        for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
                for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}
vector<int> primos;
void buscarprimos(){
        crearcriba();
        forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}
//~ Useful for bit trick: #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) ), #define INDEX(i) ( (criba[i
      >>5]>>((i)&31))&1 ), unsigned int criba[MAXP/32+1];


int main() {
        freopen("primos", "w", stdout);
        buscarprimos();
        cout << '{';
        bool first=true;
        forall(it, primos){
                if(first) first=false;
                else cout << ',';
                cout << *it;
    }
    cout << "};\n";
    return 0;
}
```

5.6 Fast Exp.

```
ll binpow(ll a, ll b) {
    /*si se necesita la potencia modulo m: aplicar el modulo a todas
    las multiplicaciones y a 'a' al antes del loop*/
    ll res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

5.7 Fast Matrix Exp.

```
#define forn(i,n) forr(i,0,n)
#define SIZE 350
int NN;
double tmp[SIZE][SIZE];
void mul(double a[SIZE][SIZE], double b[SIZE][SIZE]){ zero(tmp);
    forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
    forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
}
void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE]){
    forn(i, NN) forn(j, NN) res[i][j]=(i==j);
    while(n){
        if(n&1) mul(res, a), n--;
        else mul(a, a), n/=2;
    } }
```

5.8 Euclidean algorithm

```
//Iterative
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int 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;
}
```

5.9 GaussJordan

```
const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

5.10 Chinese Remainder Theorem

```
struct Congruence {
    long long a, m;
};

long long chinese_remainder_theorem(vector<Congruence> const& congruences) {
    long long M = 1;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }

    long long solution = 0;
    for (auto const& congruence : congruences) {
        long long a_i = congruence.a;
        long long M_i = M / congruence.m;
        long long N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

6 Range Queries

6.1 BIT

```
const int N = 200005;
int BIT[N];

void update(int idx, int val){
    for(; idx < N; idx += idx&(-idx)){
        BIT[idx]+=val;
    }
}

int query (int idx){
    ll ret = 0;
    for(; idx > 0; idx-=idx&(-idx)){
        ret += BIT[idx];
    }
    return ret;
}

int query (int left, int right){
    return query(right) - query(left-1);
}

int lower_find(int val){
    int id = 0;
    for(int i = 31-__builtin_clz(n); i >= 0; --i){
        int nid = id | ( 1 << i);
        if(nid <= n && BIT[nid] <= val){
            val -= BIT[nid];
            id = nid;
        }
    }
    return id;
}


iota(idx+1, idx+n+1,1);
sort(idx+1, idx+n+1, [](int i_a, int i_b) { return arr[i_a] > arr[i_b];});

//Update range [l,r] to v
update(l,v);
update(r+1,-v);

//Update specific value at pos k to u
ll prev = query(k)-query(k-1);
update(k,u);
update(k, -prev);

//Inversions
for(int i=1; i <=n; i++){
    forward[i] = query(values[i]);
    update(1,1);
    update(values[i],-1);
}
memset(BIT, 0, sizeof BIT);

for(int i=n; i >0 ; i--){
    backward[i] = query(values[i]);
    update(values[i]+1, 1);
}


//Dimension change
sort(difval, difval+ind);
map<int,int> idx;
int cnt = 0;
idx[difval[0]] = cnt;
cnt++;
for(int i=1; i < ind;  i++){
    if(difval[i] != difval[i-1]){
        idx[difval[i]] = cnt;
        cnt++;
    }
}
```

6.2 Segment Tree Range Query

```
const int N = 1e5;  // limit for array size
int n;  // array size
int t[2 * N];

void build() {  // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}//O(n)

void modify(int p, int value) {  // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}//O(log(n))

int query(int l, int r) {  // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}//O(log(n))
```

```cpp
int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d", t + n + i);
  build();
  modify(0, 1);
  printf("%d\n", query(3, 11));
  return 0;
}
```

6.3 Segment Tree Range Update

```cpp
void modify(int l, int r, int value) {
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) t[l++] += value;
    if (r&1) t[--r] += value;
  }
}

int query(int p) {
  int res = 0;
  for (p += n; p > 0; p >>= 1) res += t[p];
  return res;
}

/*Push to inspect modifications*/

void push() {
  for (int i = 1; i < n; ++i) {
    t[i<<1] += t[i];
    t[i<<1|1] += t[i];
    t[i] = 0;
  }
} //
```

6.4 Segment Tree Lazy Propagation

```cpp
const int N = 500'005;
const int MOD = 998244353;

int add ( int A, int B ) { return A+B<MOD? A+B: A+B-MOD; }
int mul ( int A, int B ) { return ll(A)*B % ll(MOD); }
int sub ( int A, int B ) { return add ( A, MOD-B ); }

int n, q, h;

int sum[2*N];
pii lazy[2*N];
int lengths[2*N];

pii combine ( pii A, pii B ) {
    return {mul(A.ff, B.ff), add(mul(A.ss, B.ff), B.ss)};
}

void apply(int p, pii value) {
    sum[p] = add(mul(sum[p], value.ff), mul(lengths[p], value.ss));
    if (p < n) lazy[p] = combine(lazy[p], value);
}

void build_t() {  // build the tree
  for (int i = n - 1; i > 0; --i) {
    sum[i] = add(sum[i<<1], sum[i<<1|1]);
    lengths[i] = lengths[i<<1]+lengths[i<<1|1];
    lazy[i] = {1,0};
  }
}

void build(int p) {
  while (p > 1){
    p >>= 1;
    if(lazy[p] == pii(1,0)) sum[p] = add(sum[p<<1], sum[p<<1|1]);

  }
}

void push(int p) {
  for (int s = h-1; s > 0; --s) {
    int i = p >> s;
    if (lazy[i] != pii(1,0)) {
      apply(i<<1, lazy[i]);
      apply(i<<1|1, lazy[i]);
      lazy[i] = {1,0};
    }
  }
}

void modify(int l, int r, pii value) {
  l += n, r += n;
  int l0 = l, r0 = r;
  push(l0);
  push(r0 - 1);
  for (; l < r; l >>= 1, r >>= 1) {
    if (l&1) apply(l++, value);
    if (r&1) apply(--r, value);
  }
  build(l0);
  build(r0 - 1);
}

int query(int l, int r) {
  l += n, r += n;
  push(l);
  push(r - 1);
  int res = 0;
  for (; l < r; l >>= 1, r >>= 1) {
    if (l&1)res = add(res, sum[l++]);
    if (r&1)res = add(sum[--r], res);
  }
  return res;
}

//Initialization:
scanf ( "%d%d", &n, &q );
h = sizeof(int) * 8 - __builtin_clz(n);
//cout << "h: " << h << endl;

for ( int i = n; i < 2*n; ++i ){
    scanf ( "%d", &sum[i] );
    lengths[i] = 1;
}


build_t();
```

7 Strings

7.1 Borders

```cpp
const int N = 1e6+5;
int b[N];
int sz;

void borders(string p){
    b[0] = -1;
    p = '#'+p;
    for(int i=1; i <= sz; i++){
        int j=b[i-1];
        while(j>=0 && p[i] != p[j+1]) j = b[j];
        b[i] = j+1;
    }
}


//Encontrar periodos:
sz = s.size();
int aux = sz;
borders(s);

while(aux){
    cout << sz-b[aux] << " ";
    aux = b[aux];
}
```

7.2 Hashing

```cpp
const int p = 283;
const int M = 1e9+7;
const int N = 1e6+1;

int P[N], h[N];

ll binpow(ll a, ll b) {
    ll res = 1;
```

```
    a %= M;
    while (b > 0) {
        if (b & 1)
            res = (res * a)%M;
        a = (a * a)%M;
        b >>= 1;
    }
    return res;
}

void prepareP(int n){
    P[0] = 1;
    for(int i =1; i < n; ++i){
        P[i] = ((ll)P[i-1]*p) % M;
    }
}

void computeRollingHash(string T){
    for(int i=0; i < (int)T.size(); ++i){
        if(i!=0) h[i] = h[i-1];
        h[i] = (h[i]+((ll)(T[i]-'a'+1)*P[i])%M)%M;
    }
}

int hash_fast(int L, int R){
    if(L==0) return h[R];
    int ans = 0;
    ans = ((h[R]-h[L-1]) %M +M) %M;
    ans = ((ll)ans*binpow(P[L],M-2)) %M;
    return ans;
}
```

7.3 Manacher

```
//Find palindromes
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}
```

7.4 Z-Algorithm

```
vector<int> z(string s) {
    int n = s.size();
    vector<int> z(n);
    int x = 0, y = 0;
    for (int i = 1; i < n; i++) {
        z[i] = max(0,min(z[i-x],y-i+1));
        while (i+z[i] < n && s[z[i]] == s[i+z[i]]) {
            x = i; y = i+z[i]; z[i]++;
        }
    }
    return z;
}
```

8 Trees

8.1 LCA

```
/*
Binary lifting:
O(nlogn) para preprocesamiento
O(logn) para cada query
*/
```

```
int n, l;
vector<vector<int>> adj;

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

9 algorithm

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ $\in$[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

10 Math

10.1 Identidades

$\sum_{i=0}^{n} ni = 2^n$
$\sum_{i=0}^{n} ini = n * 2^{n-1}$
$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$
$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1)$ (doubles) $\rightarrow$ Sino ver caso impar y par
$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$
$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$
$r = e - v + k + 1$
Teorema de Pick: (Area, puntos interiores y puntos en el borde)
$A = I + \frac{B}{2} - 1$

10.2 Ec. Caracteristica

$a_0 T(n) + a_1 T(n-1) + ... + a_k T(n-k) = 0$
$p(x) = a_0 x^k + a_1 x^{k-1} + ... + a_k$
Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$
$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$

10.3 Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 (∈ `int`)
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 (∈ `tint`)
10! = 3.628.800	21! = 51.090.942.171.709.400.000

max signed tint = 9.223.372.036.854.775.807
max unsigned tint = 18.446.744.073.709.551.615

Primos cercanos a 10^n
9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n
$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

Divisores
Cantidad de divisores (σ_0) para *algunos* $n / \neg\exists n' < n, \sigma_0(n')\sigma_0(n)$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$

Suma de divisores (σ_1) para *algunos* $n / \neg\exists n' < n, \sigma_1(n')\sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$