Notebook - Maratona de Programação

Z-girls

Contents

1 DP

1.1 Mochila

```
int val[MAXN], peso[MAXN], dp[MAXN][MAXS];

int knapsack(int n, int m){ // n Objetos | Peso max
    for(int i=0;i<=n;i++){
       for(int j=0;j<=m;j++){
            if(i==0 or j==0)
                dp[i][j] = 0;
            else if(peso[i-1]<=j)
                dp[i][j] = max(val[i-1]+dp[i-1][j-peso[i-1]], dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
       }
    }
    return dp[n][m];
}
```

1.2 Troco Minimo

```
int n;
vector<int> valores;

int tabela[1005];

int dp(int k){
    if(k == 0){
        return 0;
    }
    if(tabela[k] != -1)
        return tabela[k];
    int melhor = 1e9;
    for(int i = 0; i < n; i++){
        if(valores[i] <= k)
            melhor = min(melhor,1 + dp(k - valores[i]));
    }
    return tabela[k] = melhor;
}
```

1.3 Kadane

```
// achar uma subsequencia continua no array que a soma seja a maior possivel
// nesse caso vc precisa multiplicar exatamente 1 elemento da subsequencia
// e achar a maior soma com isso

int n, x, arr[MAX], tab[MAX][2]; // tab[maior resposta no intervalo][foi multiplicado ou ãno]

int dp(int i, bool mult) {
    if (i == n-1) {
        if (!mult) return arr[n-1]*x;
        return arr[n-1];
    }
    if (tab[i][mult] != -1) return tab[i][mult];

    int res;

    if (mult) {
        res = max(arr[i], arr[i] + dp(i+1, 1));
    }
    else {
        res = max({
            arr[i]*x,
            arr[i]*x + dp(i+1, 1),
            arr[i] + dp(i+1, 0)
        });
    }

    return tab[i][mult] = res;
}

int main() {

    memset(tab, -1, sizeof(tab));

    int ans = -oo;
    for (int i = 0; i < n; i++) {
        ans = max(ans, dp(i, 0));
    }

    return 0;
}
```

1.4 Substr Palindromo

```
// êvoc deve informar se a substring de S formada pelos elementos entre os indices i e j
// é um palindromo ou ãno.

char s[MAX];
int calculado[MAX][MAX]; // inciado com false, ou 0
int tabela[MAX][MAX];

int is_palin(int i, int j){
    if(calculado[i][j]){
        return tabela[i][j];
    }
    if(i == j) return true;
    if(i + 1 == j) return s[i] == s[j];

    int ans = false;
    if(s[i] == s[j]){
        if(is_palin(i+1, j-1)){
            ans = true;
        }
    }
    calculado[i][j] = true;
    tabela[i][j] = ans;
    return ans;
}
```

1.5 Moedas

```
int tb[1005];
int n;
vector<int> moedas;

int dp(int i){
    if(i >= n)
        return 0;
    if(tb[i] != -1)
        return tb[i];

    tb[i] = max(dp(i+1), dp(i+2) + moedas[i]);
    return tb[i];
}

int main(){
    memset(tb,-1,sizeof(tb));
}
```

1.6 Digitos

```
// achar a quantidade de numeros menores que R que possuem no maximo 3 digitos nao nulos
// a ideia eh utilizar da ordem lexicografica para checar isso pois se temos por exemplo
// o numero 8500, a gente sabe que se pegarmos o numero 7... qualquer digito depois do 7
// sera necessariamente menor q 8500

string r;
int tab[20][2][5];

// i - digito de R
// menor - ja pegou um numero menor que um digito de R
// qt - quantidade de digitos nao nulos
int dp(int i, bool menor, int qt){
    if(qt > 3) return 0;
    if(i >= r.size()) return 1;
    if(tab[i][menor][qt] != -1) return tab[i][menor][qt];

    int dr = r[i]-'0';
    int res = 0;

    for(int d = 0; d <= 9; d++) {
        int dnn = qt + (d > 0);
        if(menor == true) {
            res += dp(i+1, true, dnn);
        }
        else if(d < dr) {
            res += dp(i+1, true, dnn);
        }
        else if(d == dr) {
            res += dp(i+1, false, dnn);
        }
    }

    return tab[i][menor][qt] = res;
}
```

2 Strings

2.1 Kmp

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> 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;
}
```

2.2 Z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    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;
}
```

3 Math

3.1 Mdc

```
long long gcd(long long a, long long b){
    return b ? gcd(b, a % b) : a;
}

// or just use __gcd(a,b)
```

3.2 Log

```
int intlog(double base, double x) {
    return (int)(log(x) / log(base));
}
```

3.3 Divisores

```
vector<long long> all_divisors(long long n) {
  vector<long long> ans;
  for(long long a = 1; a*a <= n; a++){
    if(n % a == 0) {
      long long b = n / a;
      ans.push_back(a);
      if(a != b) ans.push_back(b);
    }
  }
  sort(ans.begin(), ans.end());
  return ans;
}
```

3.4 Sieve Of Eratosthenes

```
int n;
vector<bool> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

3.5 Combinatoria

```
int comb(int k){
    if(k==1 or k==0)return 0;
    return (k*(k-1))/2;
}
```

3.6 Crt

```
ll crt(const vector<pair<ll, ll>> &vet){
    ll ans = 0, lcm = 1;
    ll a, b, g, x, y;
    for(const auto &p : vet) {
        tie(a, b) = p;
        tie(g, x, y) = gcd(lcm, b);
        if((a - ans) % g != 0) return -1; // no solution
        ans = ans + x * ((a - ans) / g) % (b / g) * lcm;
        lcm = lcm * (b / g);
        ans = (ans % lcm + lcm) % lcm;
    }
    return ans;
}
```

3.7 Prime Factors Sqrt

```
map<ll, ll> expo;

void primeFactors(ll n) {
    while (n % 2 == 0) {
        expo[2]++;
        n = n/2;
    }

    for (ll i = 3; i <= sqrt(n); i = i + 2) {
        while (n % i == 0) {
            expo[i]++;
            n = n/i;
        }
    }

    if (n > 2) expo[n]++;
}
```

3.8 Fatoracao Primos

```
vector<pair<int, int>> fatora(int x) {
  map<int, int> expoentes;
  while(x > 1) {
    expoentes[ lp[x] ]++; // aumentamos o expoente do primo lp[x] em 1 na resposta
    x /= lp[x];
  }
  vector<pair<int, int>> ans;
  for(pair<int, int> p : expoentes)
    ans.emplace_back(p);
  return ans;
}
```

3.9 Matrix Exponentiation

```
#include <bits/stdc++.h>
#define debug(x) cout << "[" << #x << " = " << x << "] "
#define ff first
#define ss second

using namespace std;
using ll = long long;
using ld = long double;
using pii = pair<int,int>;
using vi = vector<int>;

using tii = tuple<int,int,int>;
// auto [a,b,c] = ...
// .insert({a,b,c})

const int oo = (int)1e9; //INF to INT
const ll OO = 0x3f3f3f3f3f3f3f3fLL; //INF to LL

/*wa? coloca long long que passa;
testar casos, n = 0? n = 1? todos os numeros iguais?
Uma resposta ótima pode ter tamanho 2?
RELER O ENUNCIADO!*/

const int MOD = 1e9+7;

struct Mat{
    vector<vector<ll>> matriz;
    int l, c;

    Mat(vector<vector<ll>>& mat){
        matriz = mat;
        l = mat.size();
        c = mat[0].size();
    }

    Mat(int r, int col, bool identidade=false){
        //qnt linhas, qnt colunas, identidade
        l = r;  c = col;
        matriz.assign(l, vector<ll>(col, 0));
        if(identidade){
            for(int i = 0; i < min(l,col); i++)
                matriz[i][i] = 1;
        }
    }

    Mat operator * (const Mat& a) const{
        assert(c == a.l); //qnt lcolunas mat deve ser igual qnt linhas a
        vector<vector<ll>> resp(l, vector<ll>(a.c, 0));
        //multiplica. Algoritmo úcbico.
        for(int i = 0; i < l; i++){
            for(int j = 0; j < a.c; j++){
                for(int k = 0; k < a.l; k++){
                    resp[i][j] = (resp[i][j] + (matriz[i][k]*a.matriz[k][j]) % MOD) % MOD;
                }
            }
        }
        return Mat(resp);
    }

    Mat operator + (const Mat& a) const{
        assert(l == a.l && c == a.c); //dimensoes iguais
        vector<vector<ll>> resp(l, vector<ll>(c,0));
        for(int i = 0; i < l; i++){
            for(int j = 0; j < c; j++){
                resp[i][j] = (resp[i][j] + matriz[i][j] + a.matriz[i][j]) % MOD;
            }
        }
        return Mat(resp);
    }
};

Mat fexp(Mat& base, ll expoente, ll sz){
    Mat result = Mat(sz, sz, 1);
    while(expoente > 0){
        if(expoente & 1) result = result * base;
        base = base * base;
        expoente /= 2;
    }
    return result;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    ll n, a, b;
    cin >> a >> b >> n;

    Mat X(2,2);

    //f_i = c1 * f_(i-1) + c2 * f(i-2) + ... + ck * f(i-k)
    // monta a matriz X
    //  A °2 diagonal (todas as çõposies acima dos elementos q pertecem a diagonal principal) = 1
    //  A ultima linha é composta por c_k, c_(k-1), c_(k-2), .... , c_2, c_1
    //Para se ter o pé-simo elemento é ós fazer X^(P-1) pq indexa em 0
    //e multiplicar pela matriz coluna, onde os elementos ãso: [f(0)
    /*
              f(1)

```

```
                f(2)

                ....

                f(k-1)

                ]
    */

    //nessa ãquesto a gente tem que f_i = f_(i-1) - f(i-2), sendo que f_0 = a e f_1 = b, a matriz fica

    // 0 1
    //-1 1
    X.matriz[0][1] = 1;
    X.matriz[1][0] = -1;
    X.matriz[1][1] = 1;

    Mat y = fexp(X,n-1,2);

    ll ans = y.matriz[0][0] * a + y.matriz[0][1] * b;

    while(ans < 0)
        ans += MOD;

    cout << ans % MOD << endl;
}
```

3.10 Fast Exponentiation

```
ll fexp(ll b, ll e, ll mod) {
    ll res = 1;
    b %= mod;
    while(e){
        if(e & 1LL)
            res = (res * b) % mod;
        e = e >> 1LL;
        b = (b * b) % mod;
    }
    return res;
}
```

3.11 Mmc

```
long long lcm(long long a, long long b){
    return (a/__gcd(a,b)*b);
}
```

4 Misc

4.1 Int128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

5 ED

5.1 Seg Tree

```
class SegTree{
    vector<int> seg;
    vector<int> v;
    int size;
    int el_neutro = INT_MAX;

    int f(int a, int b){
        return min(a,b);
    }

    void update(int pos, int ini, int fim, int i, int val){
        if(i < ini or i > fim) return;
        if(ini == fim){
            seg[pos] = val; return;
        }

        int m = (ini+fim)/2;
        int e = 2*pos, d = 2*pos+1;
        update(e, ini, m, i, val);
        update(d, m+1, fim, i, val);

        seg[pos] = f(seg[e], seg[d]);
    }

    int query(int pos, int ini, int fim, int p, int q){
        if(q < ini or p > fim) return el_neutro;
        if(p <= ini and fim <= q) return seg[pos];

        int m = (ini + fim)/2;
        int e = 2*pos, d = 2*pos+1;
        return f(query(e,ini,m,p,q), query(d,m+1,fim,p,q));
    }

    void build(int pos, int ini, int fim){
        if(ini == fim){
            seg[pos] = v[ini]; return;
        }

        int m = (ini+fim)/2;
        int e = 2*pos, d=2*pos+1;

        build(e,ini,m);
        build(d,m+1,fim);

        seg[pos] = f(seg[e], seg[d]);
    }

    public:
        SegTree(int n, vector<int> source): seg(4*size), v(size){
            size = n;
            for(int i=0; i<size; i++) v[i] = source[i];
        }

        void update(int i, int val){ return update(1,1,size,i,val); }

        int query(int p, int q){ return query(1,1,size,p,q); }

        void build(){ return build(1,1,size); }
```

```cpp
};
```

5.2 Seg Lazy

```cpp
using ll = long long;

struct segTree {
    int size;
    vector<ll> tree, lazy;

    ll modify_op(ll a, ll b, ll len) {
        if (b == -1) return a;
        return b * len;
    }

    void apply_mod_op(ll &a, ll b, ll len) {
        a = modify_op(a, b, len);
    }

    ll merge(ll a, ll b) {
        return a + b;
    }

    void init(int n) {
        size = 1;
        while (size < n) size *= 2;
        tree.assign(2 * size, 0LL);
        lazy.assign(2 * size, -1);
    }

    void propagate(int x, int lx, int rx) {
        if (rx - lx == 1) return;

        int m = (lx + rx) / 2;
        apply_mod_op(lazy[2 * x + 1], lazy[x], 1);
        apply_mod_op(tree[2 * x + 1], lazy[x], m - lx);

        apply_mod_op(lazy[2 * x + 2], lazy[x], 1);
        apply_mod_op(tree[2 * x + 2], lazy[x], rx - m);

        lazy[x] = -1;
    }

    void build(vector<int> &arr, int x, int lx, int rx) {
        if (rx - lx == 1) {
            if (lx < (int)arr.size())
                tree[x] = arr[lx];

            return;
        }

        int m = (lx + rx) / 2;
        build(arr, 2 * x + 1, lx, m);
        build(arr, 2 * x + 2, m, rx);

        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
    }

    void build(vector<int> &arr) {
        build(arr, 0, 0, size);
    }

    void update(int l, int r, int v, int x, int lx, int rx) {
        propagate(x, lx, rx);

        if (lx >= r || l >= rx) return;
        if (lx >= l && rx <= r) {
            apply_mod_op(lazy[x], v, 1);
            apply_mod_op(tree[x], v, rx - lx);
            return;
        }

        int m = (lx + rx) / 2;
        update(l, r, v, 2 * x + 1, lx, m);
        update(l, r, v, 2 * x + 2, m, rx);

        tree[x] = merge(tree[2 * x + 1], tree[2 * x + 2]);
    }

    void update(int l, int r, int v) {
        update(l, r, v, 0, 0, size);
    }

    ll query(int l, int r, int x, int lx, int rx) {
        propagate(x, lx, rx);

        if (lx >= r || l >= rx) return 0;
        if (lx >= l && rx <= r) return tree[x];

        int m = (lx + rx) / 2;
        ll s1 = query(l, r, 2 * x + 1, lx, m);
        ll s2 = query(l, r, 2 * x + 2, m, rx);

        return merge(s1, s2);
    }

    ll query(int l, int r) {
        return query(l, r, 0, 0, size);
    }

    void debug() {
        for (auto e : tree)
            cout << e << ' ';
        cout << endl;

        for (auto e : lazy)
            cout << e << ' ';
        cout << endl;
    }
};
```

5.3 Dsu

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

using namespace std;

const int MAX = 1e6+17;

struct DSU {
    int n;
    vector<int> link, sizes;

    DSU(int n) {
        this->n = n;
        link.assign(n+1, 0);
        sizes.assign(n+1, 1);

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

    int find(int x) {
        while (x != link[x])
            x = link[x];

        return x;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);

        if (a == b) return;

        if (sizes[a] < sizes[b])
            swap(a, b);

        sizes[a] += sizes[b];
        link[b] = a;
    }

    int size(int x) {
        return sizes[x];
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int cities, roads; cin >> cities >> roads;
    vector<int> final_roads;
    int ans = 0;
    DSU dsu = DSU(cities);
    for (int i = 0, a, b; i < roads; i++) {
        cin >> a >> b;
        dsu.unite(a, b);
    }

    for (int i = 2; i <= cities; i++) {
        if (!dsu.same(1, i)) {
            ans++;
            final_roads.push_back(i);
            dsu.unite(1,i);
        }
    }

    cout << ans << '\n';
    for (auto e : final_roads) {
        cout << "1 " << e << '\n';
    }

}
```

5.4 Seg Pqru

```cpp
#include <bits/stdc++.h>
using namespace std;

class SegTree{
    vector<int> seg;
    vector<int> v;
    int size;
    int el_neutro = INT_MAX;

    int f(int a, int b){
        return min(a,b);
    }

    void update_range(int pos, int ini, int fim, int l, int r, int val){
        if(r < ini or l > fim) return;
        if(l <= ini and fim <= r){
            seg[pos] += val;
        }

        int mid = (ini+fim)/2;

        update_range(2*pos, ini, mid, l, r, val);
        update_range(2*pos+1, mid+1, fim, l, r, val);
    }

    int query_point(int pos, int ini, int fim, int i){
        if(ini == fim) return seg[pos];

        int mid = (ini + fim)/2;
        if(i<=mid)
            return query_point(2*pos, ini, mid, i);
        else
            return query_point(2*pos+1, mid+1, fim, i);
    }

    void build(int pos, int ini, int fim){
        if(ini == fim){
            seg[pos] = v[ini]; return;
        }

        int m = (ini+fim)/2;
        int e = 2*pos, d=2*pos+1;

        build(e,ini,m);
        build(d,m+1,fim);

        seg[pos] = f(seg[e], seg[d]);
    }

public:
    SegTree(int n, vector<int> source): seg(4*size), v(size){
        size = n;
        for(int i=0; i<size; i++) v[i] = source[i];
    }

    void update(int l, int r, int val){ return update_range(1,1,size,l, r,val); }

    int query(int i){ return query_point(1,1,size,i); }

    void build(){ return build(1,1,size); }
};
```

6 Grafos

6.1 Kruskall

```cpp
vector<int> parent, rank;

void make_set(int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

int n;
vector<Edge> edges;

int cost = 0;
vector<Edge> result;
parent.resize(n);
rank.resize(n);
for (int i = 0; i < n; i++)
    make_set(i);

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

for (Edge e : edges) {
    if (find_set(e.u) != find_set(e.v)) {
        cost += e.weight;
        result.push_back(e);
        union_sets(e.u, e.v);
    }
}
```

6.2 Dijkstra

```
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

void dijkstra(int s, vector<int> & d, vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

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

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }
}
```

6.3 Dfs

```
vector<vector<int>> graph;
vector<bool> visited;

void dfs(int vertex){
    visited[vertex] = true;

    for(int w: graph[vertex]){
        if(!visited[w]){
            dfs(w);
        }
    }
}
```

6.4 Bellman Ford

```
struct edge
{
    int a, b, cost;
};

int n, m, v;
vector<edge> e;
const int INF = 1000000000;

void solve()
{
    vector<int> d (n, INF);
    d[v] = 0;
    for (int i=0; i<n-1; ++i)
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                d[e[j].b] = min (d[e[j].b], d[e[j].a] + e[j].cost);
}
```

6.5 Bipartite

```
const int NONE = 0, BLUE = 1, RED = 2;
vector<vector<int>> graph(100005);
vector<bool> visited(100005);
int color[100005];

bool bfs(int s = 1){

    queue<int> q;
    q.push(s);
    color[s] = BLUE;

    while (not q.empty()){
        auto u = q.front(); q.pop();

        for (auto v : graph[u]){
            if (color[v] == NONE){
                color[v] = 3 - color[u];
                q.push(v);
            }
            else if (color[v] == color[u]){
                return false;
            }
        }
    }

    return true;
}

bool is_bipartite(int n){

    for (int i = 1; i<=n; i++)
        if (color[i] == NONE and not bfs(i))
            return false;

    return true;
}
```

6.6 Floyd Warshall

```
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]);
        }
    }
}
```

6.7 Bfs

```
void bfs(int start){

    queue<int> q;
    q.push(start);

    vector<bool> visited(GRAPH_MAX_SIZE,false);
    visited[start] = true;
    while(q.size()){
        int u = q.front();
        q.pop();
        for(int w: graph[u]){
            if(not visited[w]){
                q.push(w);
                visited[w] = true;
            }
        }
    }

}
```

6.8 Lca

```
const int MAX = 2e5+17;

int n, l;
vector<vector<int>> adj;
// vector<pair<int, int>> adj[MAX];
// int dist[MAX];

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);
    }

    /*for (auto [u, peso] : adj[v]) {
        if (u != p) {
            dist[u] = dist[v] + peso;
            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(MAX);
    tout.resize(MAX);
    timer = 0;
    up.assign(MAX, vector<int>(32));
    dfs(root, root);
}

//distance between a and b
// dist[a] + dist[b] - 2*dist[lca(a, b)]
```

6.9 Dinic

```
const int N = 300;

struct Dinic {
    struct Edge{
        int from, to; ll flow, cap;
    };
    vector<Edge> edge;

    vector<int> g[N];
    int ne = 0;
    int lvl[N], vis[N], pass;
    int qu[N], px[N], qt;

    ll run(int s, int sink, ll minE) {
        if(s == sink) return minE;

        ll ans = 0;

        for(; px[s] < (int)g[s].size(); px[s]++) {
            int e = g[s][ px[s] ];
            auto &v = edge[e], &rev = edge[e^1];
            if(lvl[v.to] != lvl[s]+1 || v.flow >= v.cap)
                continue;           // v.cap - v.flow < lim
            ll tmp = run(v.to, sink,min(minE, v.cap-v.flow));
            v.flow += tmp, rev.flow -= tmp;
            ans += tmp, minE -= tmp;
            if(minE == 0) break;
        }
        return ans;
    }
    bool bfs(int source, int sink) {
        qt = 0;
        qu[qt++] = source;
        lvl[source] = 1;
        vis[source] = ++pass;
        for(int i = 0; i < qt; i++) {
            int u = qu[i];
            px[u] = 0;
            if(u == sink) return true;
            for(auto& ed : g[u]) {
                auto v = edge[ed];
                if(v.flow >= v.cap || vis[v.to] == pass)
                    continue; // v.cap - v.flow < lim
                vis[v.to] = pass;
                lvl[v.to] = lvl[u]+1;
                qu[qt++] = v.to;
            }
```

```cpp
        }
        return false;
    }
    ll flow(int source, int sink) {
        reset_flow();
        ll ans = 0;
        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
        while(bfs(source, sink))
            ans += run(source, sink, LLINF);
        return ans;
    }
    void addEdge(int u, int v, ll c, ll rc) {
        Edge e = {u, v, 0, c};
        edge.pb(e);
        g[u].push_back(ne++);

        e = {v, u, 0, rc};
        edge.pb(e);
        g[v].push_back(ne++);
    }
    void reset_flow() {
        for(int i = 0; i < ne; i++)
            edge[i].flow = 0;
        memset(lvl, 0, sizeof(lvl));
        memset(vis, 0, sizeof(vis));
        memset(qu, 0, sizeof(qu));
        memset(px, 0, sizeof(px));
        qt = 0; pass = 0;
    }
};
```

6.10 Find Cycle

```cpp
bitset<MAX> visited;
vector<int> path;
vector<int> adj[MAX];

bool dfs(int u, int p){

    if (visited[u]) return false;

    path.pb(u);
    visited[u] = true;

    for (auto v : adj[u]){
        if (visited[v] and u != v and p != v){
            path.pb(v); return true;
        }

        if (dfs(v, u)) return true;
    }

    path.pop_back();
    return false;
}

bool has_cycle(int N){

    visited.reset();

    for (int u = 1; u <= N; ++u){
        path.clear();
        if (not visited[u] and dfs(u,-1))
            return true;

    }

    return false;
}
```

7 Template

7.1 Template Clean

```cpp
#include <bits/stdc++.h>
using namespace std;

// g++ -std=c++20 main.cpp

// g++ -std=c++17 -Wshadow -Wall -Wextra -Wformat=2 -Wconversion -fsanitize=address,undefined -fno-sanitize-recover -Wfatal-errors

// cout << fixed << setprecision(12) << value << endl;

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);



    return 0;
}
```

7.2 Template

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define optimize std::ios::sync_with_stdio(false); cin.tie(NULL);
#define vi vector<int>
#define ll long long
#define pb push_back
#define mp make_pair
#define ff first
#define ss second
#define pii pair<int, int>
#define MOD 1000000007
#define sqr(x) ((x) * (x))
#define all(x) (x).begin(), (x).end()
#define FOR(i, j, n) for (int i = j; i < n; i++)
#define qle(i, n) (i == n ? "\n" : " ")
#define endl "\n"
const int oo = 1e9;
const int MAX = 1e6;

int32_t main(){ optimize;

    return 0;
}
```

8 Algoritmos

8.1 Ceil

```cpp
long long division_ceil(long long a, long long b) {
    return 1 + ((a - 1) / b); // if a != 0
}
```

8.2 Binary Search Last True

```cpp
int last_true(int lo, int hi, function<bool(int)> f) {
    lo--;
    while (lo < hi) {
        int mid = lo + (hi - lo + 1) / 2;
        if (f(mid)) {
            lo = mid;
        } else {
            hi = mid - 1;
        }
    }
    return lo;
}
```

8.3 Kadane

```cpp
int ans = a[0], ans_l = 0, ans_r = 0;
int sum = 0, minus_pos = -1;

for (int r = 0; r < n; ++r) {
    sum += a[r];
    if (sum > ans) {
        ans = sum;
        ans_l = minus_pos + 1;
        ans_r = r;
    }
    if (sum < 0) {
        sum = 0;
        minus_pos = r;
    }
}
```

8.4 Binary Exponentiation

```cpp
long long power(long long a, long long b) {
    long long res = 1;
    while (b > 0) {
        if (b & 1)
            res = res * a;
        a = a * a;
        b >>= 1;
    }
    return res;
}
```

8.5 Delta-encoding

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n, q;
    cin >> n >> q;
    int [n];
    int delta[n+2];

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        delta[l] += x;
        delta[r+1] -= x;
    }

    int curr = 0;
    for(int i=0; i < n; i++){
        curr += delta[i];
        v[i] = curr;
    }

    for(int i=0; i< n; i++){
        cout << v[i] << ' ';
    }
    cout << '\n';

    return 0;
}
```

8.6 Binary Search First True

```cpp
int first_true(int lo, int hi, function<bool(int)> f) {
    hi++;
    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        if (f(mid)) {
            hi = mid;
        } else {
            lo = mid + 1;
        }
    }
    return lo;
}
```