ACM NOTEBOOK

MATHS

Contents

BINOMINAL COEFFICENT

```
struct binomial_coefficent{
const int MAXN = 1e6 + 5;
const int MOD = 1e9 + 7;
int64_t invs[MAXN + 12] , fact[MAXN + 12], catalan[MAXN + 12];

int64_t Power(int64_t A, int64_t B) {
        int64_t res = 1;
        while (B){
            if (B & 1) res = (res % MOD * a % MOD + MOD) % MOD;
            a = (a % MOD * a % MOD + MOD) % MOD;
            b >>= 1;
        }
        return res % MOD;
}

void prepare_fact(int N) {
        fact[0] = 1;
        for (int i = 1 ; i <= N ; i++)
                fact[i] = (1ll * fact[i-1] * i) % MOD;
        invs[N] = Power(fact[N], MOD - 2);
        for (int i = N - 1 ; i >= 0 ; i --)
                invs[i] = (1ll * invs[i + 1] * (i + 1)) % MOD;
}

int small_C(int n, int k) {
    int res = 1;
    for (int i = 1; i <= k; ++i){
        res *= (n - k + i);
        res /= i;
    }
    return res;
}

int C(int N, int K) {
        if (K > N) return 0;
        return (1ll * (1ll * fact[N] * invs[N - K]) % MOD * invs[K]) % MOD;
}

int P(int N, int K) {
        if (K > N || N < 0 || K < 0)
                return 0;
        return (1ll * fact[N] * invs[N - K]) % MOD;
}

};
```

CATALAN NUMBER

- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of rooted full binary trees with $n + 1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
- The number of ways to completely parenthesize $n + 1$ factors.
- The number of triangulations of a convex polygon with $n + 2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint chords.
- The number of non-isomorphic full binary trees with n internal nodes (i.e. nodes having at least one son).
- The number of monotonic lattice paths from point $(0, 0)$ to point (n, n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting $(0, 0)$ to (n, n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the rearrangement is stack sorted if and only if there is no such index $i < j < k$, such that $a_k < a_i < a_j$).
- The number of non-crossing partitions of a set of n elements.
- The number of ways to cover the ladder $1 \dots n$ using n rectangles (The ladder consists of n columns, where i^{th} column has a height i).

```
int Catalan(int N){
    //1, 1, 2, 5, 14, 42, 132, 429, 1430,...
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
    return catalan[N];
}
```

EXTENDED EUCLID (NOT A PRIME MOD)

```
struct extended_euclid{
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

int x, y;
int g = gcd(a, MOD, x, y);
if (g != 1) {
    cout << "No solution!";
}
else {
    x = (x % MOD + MOD) % MOD;
    cout << x << "\n";
}

};
```

EULER TOTIENT FUNCTION + SEGMENT SIEVE

```
long long eulerPhi(long long n) { // = n (1-1/p1) ... (1-1/pn)
      // Counting amount of prime numbers that <= n;
    if (n == 0) return 0;
    long long ans = n;
    for (int x = 2; x*x <= n; ++x) {
        if (n % x == 0) {
            ans -= ans / x;
            while (n % x == 0) n /= x;
        }
    }
    if (n > 1) ans -= ans / n;
    return ans;
```

```
struct segment_sieve(){
    vector<char> segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }

    vector<char> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i * i); j <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
   vector<char> segmentedSieveNoPreGen(long long L, long long R) {
    vector<char> isPrime(R - L + 1, true);
    long long lim = sqrt(R);
    for (long long i = 2; i <= lim; ++i)
        for (long long j = max(i * i, (L + i - 1) / i * i); j <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
```

```
struct inclusion_exclusion(){
    /*
    Problem: Task: given two numbers n and r,
    count the number of integers in the interval [1, r] that
    are relatively prime to n (their GCD is 1).

    Solution: Solve the inverse problem - count the numbers that aren't relatively prime to n;
    Using include - exclude principle to discard
    */
    int solve (int n, int r) {
    vector<int> p;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            p.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        p.push_back (n);

    int sum = 0;
    for (int msk=1; msk<(1<<p.size()); ++msk) {
        int mult = 1,
            bits = 0;
        for (int i=0; i<(int)p.size(); ++i)
            if (msk & (1<<i)) {
                ++bits;
                mult *= p[i];
            }

        int cur = r / mult;
        if (bits % 2 == 1)
            sum += cur;
        else
            sum -= cur;
    }
    return r - sum;
}
};
```

```
const int INF = 1e9;
const int MOD = 1e9 + 7;
const int MAX = 1e6 + 5;
const int BASE = 1e9 + 7;
using BigInt = vector <int>;

/*
94814663943294
94659453065882
7352110986964540053783128
*/
void Fix(BigInt &a){

    // REP(i, (int) a.size()) cout << a[i] << " ";
    while ((int) a.size() > 1 && a.back() == 0)  a.pop_back();
    int carry = 0;
    REP(i, (int) a.size()){
        a[i] += carry;
        carry = a[i] / 10;
        a[i] %= 10;
    }
    if (carry) a.push_back(carry);
}

void Print(BigInt a){
    Fix(a);
    PER(i, (int) a.size()) cout << a[i];
    cout << "\n";
}

void Scan(BigInt &a){
    string s;
    cin >> s;
    int n = (int) s.size();
    a.resize(n);
    REP(i, n) a[i] = s[n - 1 - i] - '0';
}
```

```
BigInt Integer(int x){
    string num = "";
    if (x == 0) num += '0';
    while (x){
        num = (char) (x % 10 + '0') + num;
        x /= 10;
    }
    reverse(all(num));
    BigInt ans((int) num.size());
    REP(i, (int) num.size()) ans[i] = num[i] - '0';
    return ans;
}
// Comparison
bool operator< (BigInt a, BigInt b){
    Fix(a);
    Fix(b);
    if ((int) a.size() != (int) b.size())
        return (a.size() < b.size());
    PER(i, (int) a.size()){
        if (a[i] != b[i])
            return (a[i] < b[i]);
    }
    return false;
}
bool operator> (BigInt a, BigInt b){
    return (b < a);
}

bool operator== (BigInt a, BigInt b){
    return (!(a < b) && !(a > b));
}

bool operator>= (BigInt a, BigInt b){
    return (a > b) || (a == b);
}

bool operator<= (BigInt a, BigInt b){
    return (a < b) || (a == b);
}
```

```
// Arithmetic
BigInt operator+ (BigInt a, BigInt b){
    Fix(a);
    Fix(b);
    BigInt ans;
    int carry = 0;
    REP(i, (int) max(a.size(), b.size())){
        if (i < (int) a.size()) carry += a[i];
        if (i < (int) b.size()) carry += b[i];
        ans.push_back(carry % BASE);
        carry /= BASE;
    }
    if (carry) ans.push_back(carry);
    Fix(ans);
    return ans;
}

BigInt operator- (BigInt a, BigInt b){
    Fix(a);
    Fix(b);
    BigInt ans(a.size());
    int carry = 0;

    REP(i, (int) a.size()){
        int cur = a[i] - (i < (int) b.size()? b[i] : 0) + carry;
        if (cur < 0)
            ans[i] = (cur + 10), carry = -1;
        else
            ans[i] = (cur), carry = 0;
    }
    Fix(ans);
    return ans;
}
BigInt operator* (BigInt a, BigInt b){
    Fix(a);
    Fix(b);
    BigInt ans;
    int n = (int) a.size();
    int m = (int) b.size();
    ans.assign(n + m + 1, 0);
    REP(i, n){
        REP(j, m) ans[i + j] += a[i] * b[j];
    }
    Fix(ans);
    return ans;
}
```

```
BigInt operator/ (BigInt a, BigInt b){
    Fix(a);
    Fix(b);
    BigInt ans, cur;
    if (b == Integer(0)) return Integer(-1);
    PER(i, (int) a.size()){
        cur.insert(cur.begin(), a[i]);
        int x = 0, l = 0, r = INF;
        while (l <= r){
            int m = (l + r) / 2;
            if (b * Integer(m) > cur){
                x = m;
                r = m - 1;
            }
            else l = m + 1;
        }
        cur = cur - b * Integer(x - 1);
        ans.insert(ans.begin(), (x - 1));
    }
    return ans;
}

void run_case(){
    BigInt A, B;
    Scan(A);
    Scan(B);
    Print(A / B);
}
```

DIJKSTRA

```
// single source shortest path (no negative)
vector<<pair<int, int>>> adj[MAXN];

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;
    using pii = pair<int, int>;
    priority_queue<pii, vector<pii>, greater<pii>> q;
    q.push({0, s});
    while (!q.empty()) {
        int v = q.top().second;
        int d_v = q.top().first;
        q.pop();
        if (d_v != d[v])
            continue;

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

            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
                q.push({d[to], to});
            }
        }
    }
vector<int> restore_path(int s, int t, vector<int> const& p) {
    vector<int> path;
    for (int v = t; v != s; v = p[v])
        path.push_back(v);
    path.push_back(s);

    reverse(path.begin(), path.end());
    return path;
}
```

0 – 1 BFS

```
// based on dijkstra algorithm, faster.
vector<int> d(n, INF);
d[s] = 0;
deque<int> q;
q.push_front(s);
while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    for (auto edge : adj[v]) {
        int u = edge.first;
        int w = edge.second;
        if (d[v] + w < d[u]) {
            d[u] = d[v] + w;
            if (w == 1)
                q.push_back(u);
            else
                q.push_front(u);
        }
    }
}
```

```
// detect negative cycle
void bellman_ford()
{
    vector<int> d (n, INF);
    d[v] = 0;
    vector<int> p (n, - 1);
    int x;
    for (int i=0; i<n; ++i)
    {
        x = -1;
        for (int j=0; j<m; ++j)
            if (d[e[j].a] < INF)
                if (d[e[j].b] > d[e[j].a] + e[j].cost)
                {
                    d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);
                    p[e[j].b] = e[j].a;
                    x = e[j].b;
                }
    }

    if (x == -1)
        cout << "No negative cycle from " << v;
    else
    {
        int y = x;
        for (int i=0; i<n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur=y; ; cur=p[cur])
        {
            path.push_back (cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse (path.begin(), path.end());

        cout << "Negative cycle: ";
        for (size_t i=0; i<path.size(); ++i)
            cout << path[i] << ' ';
    }
}
```

BELLMAN – FORD QUEUE

```
// faster than normal? but not safe in some tests
// detect negative cycle
const int INF = 1000000000;
vector<vector<pair<int, int>>> adj;

bool bellman_ford_queue(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;
}
```

FLOYD

```
// Tested:
// - https://cses.fi/problemset/task/1672/
// - (trace) https://oj.vnoi.info/problem/floyd
using ll = long long;
const ll INF = 4e18;
struct Floyd {
    Floyd(int _n, const std::vector<std::vector<ll>> _c) : n(_n), c(_c), trace(n) {
        for (int i = 0; i < n; i++) {
            trace[i] = std::vector<int> (n, -1);
            for (int j = 0; j < n; j++) {
                if (c[i][j] == INF) trace[i][j] = -1;
                else trace[i][j] = i;
            }
        }

        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (c[i][k] != INF && c[k][j] != INF && c[i][j] > c[i][k] + c[k][j])
{
                        c[i][j] = c[i][k] + c[k][j];
                        trace[i][j] = trace[k][j];
                    }
                }
            }
        }
    }

    // Return {distance, path}
    // If no path -> returns {-1, {}}
    std::pair<ll, std::vector<int>> get_path(int start, int target) {
        if (trace[start][target] == -1) return {-1, {}};

        std::vector<int> path;
        for (int u = target; u != start; u = trace[start][u]) {
            path.push_back(u);
        }
        path.push_back(start);
        reverse(path.begin(), path.end());
        return {c[start][target], path};
    }

    int n;
    std::vector<std::vector<ll>> c;
    std::vector<std::vector<int>> trace;
};
```

PRIM

```
const int INF = 1000000000;

struct Edge {
    int w = INF, to = -1;
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
};

int n;
vector<vector<Edge>> adj;

void prim() {
    int total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    for (int i = 0; i < n; ++i) {
        if (q.empty()) {
            cout << "No MST!" << endl;
            exit(0);
        }

        int v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());

        if (min_e[v].to != -1)
            cout << v << " " << min_e[v].to << endl;

        for (Edge e : adj[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }

    cout << total_weight << endl;
}
```

```
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);
    }
}
```

TARJAN

```
// Counting strongly component connected
const int N = 100005;
const int oo = 0x3c3c3c3c;
int n, m, Num[N], Low[N], cnt = 0;
vector<int> a[N];
stack<int> st;
int Count = 0;

void visit(int u) {
    Low[u] = Num[u] = ++cnt;
    st.push(u);
    for (int v : a[u])
        if (Num[v])
            Low[u] = min(Low[u], Num[v]);
        else {
            visit(v);
            Low[u] = min(Low[u], Low[v]);
        }
    if (Num[u] == Low[u]) {  // found one
        Count++;
        int v;
        do {
            v = st.top();
            st.pop();
            Num[v] = Low[v] = oo;  // remove v from graph
        } while (v != u);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        a[x].push_back(y);
    }

    for (int i = 1; i <= n; i++)
        if (!Num[i]) visit(i);

    cout << Count << endl;
}
```

```cpp
const int maxN = 10010;

int n, m;
bool joint[maxN];
int timeDfs = 0, bridge = 0;
int low[maxN], num[maxN];
vector <int> g[maxN];

void dfs(int u, int pre) {
    int child = 0; // Số lượng con trực tiếp của đỉnh u trong cây DFS
    num[u] = low[u] = ++timeDfs;
    for (int v : g[u]) {
        if (v == pre) continue;
        if (!num[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] == num[v]) bridge++;
            child++;
            if (u == pre) { // Nếu u là đỉnh gốc của cây DFS
                if (child > 1) joint[u] = true;
            }
            else if (low[v] >= num[u]) joint[u] = true;
        }
        else low[u] = min(low[u], num[v]);
    }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for (int i = 1; i <= n; i++)
        if (!num[i]) dfs(i, i);

    int cntJoint = 0;
    for (int i = 1; i <= n; i++) cntJoint += joint[i];

    cout << cntJoint << ' ' << bridge;
}
```

BIPARTITE GRAPH CHECK

```
int n, m;
vector <int> adj[MAX];
bool color[MAX], vis[MAX];
bool is_bipartite = true;

void dfs(int i, bool x){
    vis[i] = true;
    color[i] = x;
    for (int j : adj[i]){
        if (!vis[j])
            dfs(j, !x);
        else if (color[j] == color[i])
            is_bipartite = false;
    }
}

int main(){
    cin >> n >> m;
    FOR(i, 1, m){
        int x, y;
        cin >> x >> y;
        adj[x].pb(y);
        adj[y].pb(x);
    }
    FOR(i, 1, n){
        if (vis[i])
            continue;
        dfs(i, 0);
    }
    if (is_bipartite)
        FOR(i, 1, n) cout << color[i] + 1 << ' ';
    else cout << "IMPOSSIBLE";
    cout << '\n';
}
```

TOPO SORT

```cpp
// Topo sort
// returns: <has topo sort?, topo order>
// Notes:
// - To find lexicographically min -> change queue<int> qu to priority_queue (greater
<int>)
//
std::pair<bool, std::vector<int>> topo_sort(const std::vector<std::vector<int>>& g) {
    int n = g.size();
    // init in_deg
    std::vector<int> in_deg(n, 0);
    for (int u = 0; u < n; u++) {
        for (int v : g[u]) {
            in_deg[v]++;
        }
    }

    // find topo order
    std::vector<int> res;
    std::queue<int> qu;
    for (int u = 0; u < n; u++) {
        if (in_deg[u] == 0) {
            qu.push(u);
        }
    }

    while (!qu.empty()) {
        int u = qu.front(); qu.pop();
        res.push_back(u);
        for (int v : g[u]) {
            in_deg[v]--;
            if (in_deg[v] == 0) {
                qu.push(v);
            }
        }
    }

    if ((int) res.size() < n) {
        return {false, {}};
    }
    return {true, res};
}
```

```cpp
// DisjointSet {{{
struct DSU {
    vector<int> lab;

    DSU(int n) : lab(n+1, -1) {}

    int getRoot(int u) {
        if (lab[u] < 0) return u;
        return lab[u] = getRoot(lab[u]);
    }

    bool merge(int u, int v) {
        u = getRoot(u); v = getRoot(v);
        if (u == v) return false;
        if (lab[u] > lab[v]) swap(u, v);
        lab[u] += lab[v];
        lab[v] = u;
        return true;
    }

    bool same_component(int u, int v) {
        return getRoot(u) == getRoot(v);
    }

    int component_size(int u) {
        return -lab[getRoot(u)];
    }
};
```

```cpp
// 1D Fenwick
// 0 based index
//
// Tested:
// - https://judge.yosupo.jp/problem/static_range_sum
// - https://judge.yosupo.jp/problem/point_add_range_sum
template<
    typename T  // need to support operators + -
> struct Fenwick {
    Fenwick(int _n) : n(_n), f(_n + 1) {}

    // a[u] += val
    void update(int u, T val) {
        assert(0 <= u && u < n);
        ++u;
        for (; u <= n; u += u & -u) {
            f[u] += val;
        }
    }

    // return a[0] + .. + a[u-1]
    T get(int u) const {
        assert(0 <= u && u <= n);
        T res = 0;
        for (; u > 0; u -= u & -u) {
            res += f[u];
        }
        return res;
    }

    // return a[l] + .. + a[r-1]
    T get(int l, int r) const {
        assert(0 <= l && l <= r && r <= n);
        if (l == r) return 0;  // empty
        return get(r) - get(l);
    }

    void reset() {
        std::fill(f.begin(), f.end(), T(0));
    }

    int n;
    vector<T> f;
};
```

```
struct FenwickTree2D {
    //zero index
    vector<vector<int>> bit;
    int n, m;

    int sum(int x, int y) {
        int ret = 0;
        for (int i = x; i >= 0; i -= (i & -i))
            for (int j = y; j >= 0; j -= (j & -j))
                ret += bit[i][j];
        return ret;
    }

    void add(int x, int y, int val) {
        for (int i = x; i < n; i += (i & -i)))
            for (int j = y; j < m; j += (j & -j))
                bit[i][j] += val;
    }

    int query(int x1, int y1, int x2, int y2){
        return (sum(x2, y2) - sum(x1 - 1, y2) - sum(x2, y1 - 1) + sum(x1 - 1, y1 - 1));
    }

    FenwickTree2D(std::vector<std::vector<int>> matrix){
        int n = matrix.size();
        // matrix must not be empty.
        int m = matrix[0].size();
        // Initialize matrix ft
        ft.assign(m + 1, std::vector<int> (n + 1, 0));
        for(int i = 0; i < m; ++i){
            for(int j = 0; j < n; ++j)
               add(i + 1, j + 1, matrix[i][j]);
        }
    }
};
```

```
// RMQ {{{
// Sparse table
// get minimum element in range [l, r]
// Usage:
// RMQ<int, _min> st(v);
// Note:
// - doesn't work for empty range
// Tested:
// - https://judge.yosupo.jp/problem/staticrmq

// precompute log2
int lg[MAXN+1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++)
    lg[i] = lg[i/2] + 1;
// f(x, y) = min(x, y)
int st[MAXN][K + 1];
for (int i = 0; i < N; i++)
    st[i][0] = array[i];

for (int j = 1; j <= K; j++)
    for (int i = 0; i + (1 << j) <= N; i++)
        st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
// minimum in range [L, R]
int j = lg[R - L + 1];
int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
```

```
int n, q;
int a[maxN];
int st[4 * maxN]; // Lí do sử dụng kích thước mảng là 4 * maxN sẽ được giải thích ở phần
sau

// Thủ tục xây dựng cây phân đoạn
void build(int id, int l, int r) {
    // Đoạn chỉ gồm 1 phần tử, không có nút con
    if (l == r) {
        st[id] = a[l];
        return;
    }

    // Gọi đệ quy để xử lý các nút con của nút id
    int mid = l + r >> 1; // (l + r) / 2
    build(2 * id, l, mid);
    build(2 * id + 1, mid + 1, r);

    // Cập nhật lại giá trị min của đoạn [l, r] theo 2 nút con
    st[id] = min(st[2 * id], st[2 * id + 1]);
}

// Thủ tục cập nhật
void update(int id, int l, int r, int i, int val) {
    // i nằm ngoài đoạn [l, r], ta bỏ qua nút id
    if (l > i || r < i) return;

    // Đoạn chỉ gồm 1 phần tử, không có nút con
    if (l == r) {
        st[id] = val;
        return;
    }

    // Gọi đệ quy để xử lý các nút con của nút id
    int mid = l + r >> 1; // (l + r) / 2
    update(2 * id, l, mid, i, val);
    update(2 * id + 1, mid + 1, r, i, val);

    // Cập nhật lại giá trị min của đoạn [l, r] theo 2 nút con
    st[id] = min(st[2 * id], st[2 * id + 1]);
}
```

```
// Hàm lấy giá trị
int get(int id, int l, int r, int u, int v) {
    // Đoạn [u, v] không giao với đoạn [l, r], ta bỏ qua đoạn này
    if (l >  v || r <  u) return inf;

    /* Đoạn [l, r] nằm hoàn toàn trong đoạn [u, v] mà ta đang truy vấn,
        ta trả lại thông tin lưu ở nút id */
    if (l >= u && r <= v) return st[id];

    // Gọi đệ quy với các nút con của nút id
    int mid = l + r >> 1; // (l + r) / 2
    int get1 = get(2 * id, l, mid, u, v);
    int get2 = get(2 * id + 1, mid + 1, r, u, v);

    // Trả ra giá trị nhỏ nhất theo 2 nút con
    return min(get1, get2);
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> a[i];
    build(1, 1, n);

    cin >> q;
    while (q--) {
        int type, x, y;
        cin >> type >> x >> y;
        if (type == 1) update(1, 1, n, x, y); // Gán giá trị y cho phần tử ở vị trí x
        else cout << get(1, 1, n, x, y) << '\n'; // In ra giá trị nhỏ nhất trong đoạn [x, y]
    }
}
```

```
int n, q;
int a[maxN];
long long st[4 * maxN], lazy[4 * maxN];

void build(int id, int l, int r) {
    if (l == r) {
        st[id] = a[l];
        return;
    }
    int mid = l + r >> 1;
    build(2 * id, l, mid);
    build(2 * id + 1, mid + 1, r);
    st[id] = max(st[2 * id], st[2 * id + 1]);
}

// Cập nhật nút đang xét và đẩy giá trị lazy xuống các nút con
void fix(int id, int l, int r) {
    if (!lazy[id]) return;
    st[id] += lazy[id];

    // Nếu id không phải là nút lá thì đẩy giá trị xuống các nút con
    if (l != r){
        lazy[2 * id] += lazy[id];
        lazy[2 * id + 1] += lazy[id];
    }

    lazy[id] = 0;
}

void update(int id, int l, int r, int u, int v, int val) {
    fix(id, l, r);
    if (l >  v || r <  u) return;
    if (l >= u && r <= v) {
        /* Khi cài đặt, ta LUÔN ĐẢM BẢO giá trị của nút được cập nhật ĐỒNG THỜI với
        giá trị Lazy Propagation. Như vậy sẽ tránh sai sót. */
        lazy[id] += val;
        fix(id, l, r);
        return;
    }
    int mid = l + r >> 1;
    update(2 * id, l, mid, u, v, val);
    update(2 * id + 1, mid + 1, r, u, v, val);
    st[id] = max(st[2 * id], st[2 * id + 1]);
}
```

```
long long get(int id, int l, int r, int u, int v) {
    fix(id, l, r);
    if (l >  v || r <  u) return -inf;
    if (l >= u && r <= v) return st[id];

    int mid  = l + r >> 1;
    long long get1 = get(2 * id, l, mid, u, v);
    long long get2 = get(2 * id + 1, mid + 1, r, u, v);
    return max(get1, get2);
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> a[i];
    build(1, 1, n);

    cin >> q;
    while (q--){
        int type, l, r, val;
        cin >> type >> l >> r;
        if (type == 1) {
            cin >> val;
            update(1, 1, n, l, r, val);
        }
        else cout << get(1, 1, n, l, r) << '\n';
    }
}
```

SQRT DECOMPOSITION

```
//return number of elements equal to K
const int BLOCK_SIZE = 320;
const int N = 1e5 + 2;

int n;
int cnt[N / BLOCK_SIZE + 2][N];
int a[N];

void preprocess()
{
    for (int i = 0; i < n; ++i)
        ++cnt[i / BLOCK_SIZE][a[i]];
}

int query(int l, int r, int k)
{
    int blockL = (l + BLOCK_SIZE - 1) / BLOCK_SIZE;
    int blockR = r / BLOCK_SIZE;
    if (blockL >= blockR)
        return count(a + l, a + r + 1, k); // using stl

    int sum = 0;
    for (int i = blockL; i < blockR; ++i)
        sum += cnt[i][k];

    for (int i = l, lim = blockL * BLOCK_SIZE; i < lim; ++i)
        if (a[i] == k) ++sum;

    for (int i = blockR * BLOCK_SIZE; i <= r; ++i)
        if (a[i] == k) ++sum;

    return sum;
}

void update(int u, int v)
{
    int block = u / BLOCK_SIZE;
    --cnt[block][a[u]];
    a[u] = v;
    ++cnt[block][a[u]];
}
```

```
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure
int block_size;
struct Query {
    int l, r, idx;
    bool operator<(Query other) const{
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};
vector<int> mo_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
    /*Optimize 1
     sort(queries.begin(), queries.end(), [](node a, node b){
         return (a.l / blockSize < b.l / blockSize || (a.l / blockSize == b.l / blockSize
&& a.r < b.r));
     });
     bool cmp(pair<int, int> p, pair<int, int> q) {
            if (p.first / BLOCK_SIZE != q.first / BLOCK_SIZE)
                return p < q;
            return (p.first / BLOCK_SIZE & 1) ? (p.second < q.second) : (p.second >
q.second);
        }
    */
    int cur_l = 0, cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
```

```cpp
#include <stdio.h>
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 100005;
int n, Root, l[N], P[20][N];

int level(int u) {
    if (u == Root)
        return l[u] = 1;
    if (l[u] == 0)
        l[u] = level(P[0][u]) + 1;
    return l[u];
}

int lca(int x, int y) {
    for (int k = 19; k >= 0; k--)
        if (l[P[k][x]] >= l[y])
            x = P[k][x];
    for (int k = 19; k >= 0; k--)
        if (l[P[k][y]] >= l[x])
            y = P[k][y];
    for (int k = 19; k >= 0; k--)
        if (P[k][x] != P[k][y]) {
            x = P[k][x];
            y = P[k][y];
        }
    while (x != y) {
        x = P[0][x];
        y = P[0][y];
    }
    return x;
}
/*
Truy vấn tổ tiên chung của cặp x, y
l[u] là độ sâu của nút u, nút gốc có
độ sâu bằng 1.
P[k][u] là tổ tiên thứ 2^k của u. Nói
riêng, P[0][u] là cha trực tiếp của
nút u.
*/
```

```cpp
void solve() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        int p;
        scanf("%d", &p);
        while (p-- > 0) {
            int q;
            scanf("%d", &q);
            P[0][q] = i;
        }
    }
    for (int i = 1; i <= n; i++)
        if (P[0][i] == 0)
            Root = i;
    for (int i = 1; i <= n; i++)
        level(i); // done l

    for (int k = 1; k <= 19; k++)
        for (int i = 1; i <= n; i++)
            P[k][i] = P[k - 1][P[k - 1][i]];

    int m;
    scanf("%d", &m);
    while (m-- > 0) {
        int x, y;
        scanf("%d%d", &x, &y);
        printf("%d\n", lca(x, y));
    }
}

int main() {
    int t;
    scanf("%d", &t);
    for (int i = 1; i <= t; i++) {
        printf("Case %d:\n", i);
        solve();
        for (int j = 1; j <= n; j++) {
            l[j] = 0;
            P[0][j] = 0;
        }
    }
}
```

STRING

HASHING

```
/* String matching */
const int N = 1000006, BASE = 1000000007;
int m, n;
char a[N], b[N];
long A[N], B[N], M[N];

void hash_build(char a[], int n, long H[]) {
    for (int i = 1; i <= n; i++)
        H[i] = (H[i - 1] * M[1] + a[i]) % BASE;
}

long hash_range(long H[], int L, int R) {
    return (H[R] - H[L - 1] * M[R - L + 1] + 1LL * BASE * BASE) % BASE;
}

int main() {
    M[0] = 1;
    M[1] = 2309;
    for (int i = 2; i < N; i++)
        M[i] = M[i - 1] * M[1] % BASE;

    scanf("%s", a + 1);
    m = strlen(a + 1);
    scanf("%s", b + 1);
    n = strlen(b + 1);
    hash_build(a, m, A);
    hash_build(b, n, B);
    for (int i = 1; i <= m - n + 1; i++) {
        if (hash_range(A, i, i + n - 1) == B[n])
            printf("%d ", i);
    }
    printf("\n");
}
```

```
const int MN = 1000111;

char s[MN], t[MN];
int nxt[MN];
// String matching t occurring in s (more precise than hashing)
// O(m + n);
int main() {
    scanf("%s\n", &t[1]);
    scanf("%s\n", &s[1]);

    int j;
    j = nxt[1] = 0;
    for (int i = 2; s[i]; ++i) {
        while (j > 0 && s[j + 1] != s[i]) j = nxt[j];
        if (s[j + 1] == s[i]) ++j;
        nxt[i] = j;
    }

    j = 0;
    for (int i = 1; t[i]; ++i) {
        while (j > 0 && s[j + 1] != t[i]) j = nxt[j];
        if (s[j + 1] == t[i]) ++j;

        if (s[j + 1] == 0) {  // Het xau s
            printf("%d ", i - j + 1);
            j = nxt[j];
        }
    }
    puts("");
    return 0;
}
```

```
// http://codeforces.com/blog/entry/8219 (DnC optimization for DP)
// Original Recurrence
//   dp[i][j] = min(dp[i-1][k] + C[k][j]) for k < j
// Sufficient condition:
//   A[i][j] <= A[i][j+1]
//   where A[i][j] = smallest k that gives optimal answer
// How to use:
//   // compute i-th row of dp from L to R. optL <= A[i][L] <= A[i][R] <= optR
//   compute(i, L, R, optL, optR)
//       1. special case L == R
//       2. let M = (L + R) / 2. Calculate dp[i][M] and opt[i][M] using O(optR - optL + 1)
//       3. compute(i, L, M-1, optL, opt[i][M])
//       4. compute(i, M+1, R, opt[i][M], optR)
// Example: http://codeforces.com/contest/321/problem/E
const int MN = 4011;
const int inf = 1000111000;
int n, k, cost[MN][MN], dp[811][MN];
inline int getCost(int i, int j) {
    return cost[j][j] - cost[j][i-1] - cost[i-1][j] + cost[i-1][i-1];}
void compute(int i, int L, int R, int optL, int optR) {
    if (L > R) return ;
    int mid = (L + R) >> 1, savek = optL;
    dp[i][mid] = inf;
    FOR(k,optL,min(mid-1, optR)) {
        int cur = dp[i-1][k] + getCost(k+1, mid);
        if (cur < dp[i][mid]) {
            dp[i][mid] = cur;
            savek = k;
        }
    }
    compute(i, L, mid-1, optL, savek);
    compute(i, mid+1, R, savek, optR);
}
void solve() {
    cin >> n >> k;
    FOR(i,1,n) FOR(j,1,n) {
        cin >> cost[i][j];
        cost[i][j] = cost[i-1][j] + cost[i][j-1] - cost[i-1][j-1] + cost[i][j];
    }
    dp[0][0] = 0;
    FOR(i,1,n) dp[0][i] = inf;
    FOR(i,1,k) compute(i, 1, n, 0, n);
    cout << dp[k][n] / 2 << "\n";
}
```

TERNARY SEARCH

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

BITMASK

```
void bitmask (){
    // Iterating over subsets.
    // Note that constrain of the problem requiring bitmask usually small, around n = ~20;

    #define MASK(i) (1 << (i)) // make mask
    #define COUNT_BIT(x) __builtin_popcount(x) // counting set-on bits of x
    #define STATE(x, i) ((x) & (1 << (i))) // state of ith bit of x
    #define SET_ON(x, i) ((x) | (1 << (i))) // set the ith bit of x on
    #define SET_OFF(x, i) ((x) & ~(1 << (i))) //  set the ith bit of x off

}
```