Table of contents

8. Knapsack
9. Coin Change
10. Knuth Opti
11. twonearpoint
12. Bit Field DP

0. Base

```
#include <bits/stdc++.h>
#define for1(s,n) for(int i = s; i < n; i++)
#define for1j(s,n) for(int j = s; j < n; j++)
#define foreach(k) for(auto i : k)
#define foreachj(k) for(auto j : k)
#define pb(a) push_back(a)
#define sz(a) a.size()

using namespace std;
typedef unsigned long long ull;
typedef long long ll;
typedef long long llint;
typedef vector <int> iv1;
typedef vector <vector<int>> iv2;
typedef vector <ll> llv1;
typedef vector <llv1> llv2;
typedef unsigned int uint;
typedef vector <ull> ullv1;
typedef vector <vector <ull>> ullv2;
typedef pair<int, int> pii;
typedef pair<ll, ll> ll;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}
```

1. Graph

1.1. Dijkstra

```
#define MAX 100000
#define INF (ll)1e18

struct edge {
    ll node;
    ll cost;
    bool operator<(const edge &to) const {
        return cost > to.cost;
    }
```

```
};

struct WGraph {
    ll n;
    vector<vector<edge>> adj;
    llv1 prev;
    WGraph(ll n) : n{n}, adj(n+1) {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({e, cost});
    }

    void input(ll m) { // 단방향
        ll a, b, c;
        while(m--) {
            cin >> a >> b >> c;
            addEdge(a,b,c);
        }
    }

    void inputD(ll m) { // 양방향
        ll a, b, c;
        while(m--){
            cin >> a >> b >> c;
            addEdge(a,b,c);
            addEdge(b,a,c);
        }
    }

    llv1 dijkstra(ll s) {
        llv1 dist(n+1, INF);
        prev.resize(n+1, -1);
        priority_queue<edge> pq;
        pq.push({ s, 0ll });
        dist[s] = 0;
        while (!pq.empty()) {
            edge cur = pq.top();
            pq.pop();
            if (cur.cost > dist[cur.node]) continue;
            for (auto &nxt : adj[cur.node])
                if (dist[cur.node] + nxt.cost < dist[nxt.node]) {
                    prev[nxt.node] = cur.node;
                    dist[nxt.node] = dist[cur.node] + nxt.cost;
                    pq.push({ nxt.node, dist[nxt.node] });
                }
        }
        return dist;
    }

    llv1 getPath(ll s, ll e) {
        llv1 ret;
        ll current = e;
        while(current != -1) {
            ret.push_back(current);
            current = prev[current];
```

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

1.2. Bellman-Ford

```
#define MAX 100010
#define INF (ll)1e18

struct edge {
    int to, cost;
};

int n;
vector<edge> v[MAX];
ll D[MAX];
bool bellman(ll start_point){
    fill(D,D+n+1, INF);
    D[start_point] = 0;

    bool isCycle = false;
    for1(1, n+1) {
        for1j(1, n+1) {
        for(int k=0; k<sz(v[j]); k++) {
            edge p = v[j][k];
            int end = p.to;
            ll dist = D[j] + p.cost;
                if (D[j] != INF && D[end] > dist) {
                    D[end] = dist;
                    if (i == n) isCycle = true;
                }
            }
        }
    }
    return isCycle;
}
```

1.3. Kruskal

```
#define MAXN 100010

int root[MAXN];
int level[MAXN];

class Edge{
public:
    int node[2];
```

```
    int distance;
    Edge(int a, int b, int distance){
        this->node[0] = a;
        this->node[1] = b;
        this->distance = distance;
    }

    bool operator<(Edge &edge){
        return this->distance < edge.distance;
    }
};

void init(int n) {
    for1(0, n){
        root[i] = i;
        level[i] = 1;
    }
}

int find(int x) {
    return root[x] == x ? x : root[x] = find(root[x]);
}

// merge와 동시에 cycle 여부 확인
bool merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x == y) return true;
    if (level[x] < level[y]) root[x] = y;
    else root[y] = x;
    if (level[x] == level[y]) level[x]++;
    return false;
}

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

    int n, m, start, end, cost;

    cin >> n >> m;
    vector<Edge> v;
    for1(0, m){
        cin >> start >> end >> cost;
        v.pb(Edge(start, end, cost));
    }
    sort(v.begin(), v.end());

    init(n+1);
    int sum = 0;
    for1(0, sz(v)){
        if(!merge(v[i].node[0], v[i].node[1])){
            sum += v[i].distance;
```

```
        }
    }
    cout << sum << endl;
}
```

1.4. Prim

```
struct edge {
    ll crt;
    ll node, cost;
};

struct WGraph {
    ll V;
    vector<edge> adj[MAX];
    vector<ll> prev;
    WGraph(ll V) : V{V} {}
    void addEdge(ll s, ll e, ll cost) {
        adj[s].push_back({s, e, cost});
        adj[e].push_back({e, s, cost});
    }

    ll prim(vector<edge> &selected) { // selected에 선택된 간선정보 vector 담김
        selected.clear();

        vector<bool> added(V, false);
        llv1 minWeight(V, INF), parent(V, -1);

        ll ret = 0;
        minWeight[0] = parent[0] = 0;
        for (int iter = 0; iter < V; iter++) {
            int u = -1;
            for (int v = 0; v < V; v++) {
                if (!added[v] && (u == -1 || minWeight[u]>minWeight[v]))
                    u = v;
            }

            if (parent[u] != u)
                selected.push_back({parent[u], u, minWeight[u]});

            ret += minWeight[u];
            added[u] = true;

            for1(0, sz(adj[u])) {
                int v = adj[u][i].node, weight = adj[u][i].cost;
                if (!added[v] && minWeight[v]>weight) {
                    parent[v] = u;
                    minWeight[v] = weight;
                }
            }
        }
```

```
        return ret;
    }
};
```

1.5. Topological Sort

```
int n;
int link[MAXN];
iv1 graph[MAXN];

iv1 topologySort() {
    iv1 result;
    queue<int> q;

    for1(1, n+1) {
        if(link[i] == 0) q.push(i);
    }

    while(!q.empty()) {
        int x = q.front();
        q.pop();
        result.pb(x);

        for1(0, sz(graph[x])) {
            int y = graph[x][i];
            if(--link[y]==0) q.push(y);
        }
    }

    return result;
}
```

1.6. Union-Find

```
struct UnionFind
{
    vector<int> parents;
    UnionFind(int n)
    {
        parents.resize(n);
        for (int i = 0; i < n; i++)
            parents[i] = i;
    }

    int find(int node)
    {
        int parent = parents[node];
        if (parent == node)
            return node;
```

```
        return parents[node] = find(parent);
    }

    bool merge(int a, int b)
    {
        int ra = find(a);
        int rb = find(b);
        if (ra == rb)
            return false;

        parents[ra] = rb;
        return true;
    }
};
```

1.7. SCC

```
vector<vector<int>> edges, reversed_edges, components;
vector<bool> visited;
stack<int> visit_log;
void dfs(int node)
{
    visited[node] = true;
    for (int next:edges[node])
        if (!visited[next])
            dfs(next);
    visit_log.push(node);
}

void scc(int node)
{
    visited[node] = true;
    for (int next:reversed_edges[node])
        if (!visited[next])
            scc(next);
    components.back().push_back(node);
}

int main(void)
{
    visited = vector<bool>(V, false);
    for (int node = 0; node < V; node++)
        if (!visited[node])
            dfs(node);

    visited = vector<bool>(V, false);
    while (!visit_log.empty())
    {
        int node = visit_log.top();
        visit_log.pop();
        if (!visited[node]) {
```

```
            components.push_back(vector<int>());
            scc(node);
        }
    }
}
```

1.8. 2-SAT

```
class Graph
{
public:
    int V;
    vector<bool> visited;
    stack<int> visit_stack;
    vector<int> component_number, source_components;
    vector<vector<int>> edges, reversed_edges, components, components_edges;

    Graph(int V): V(V)
    {
        edges.resize(V);
        reversed_edges.resize(V);
    }

    void append(int u, int v)
    {
        edges[u].push_back(v);
        reversed_edges[v].push_back(u);
    }

    void dfs(int node)
    {
        visited[node] = true;
        for (int next:edges[node])
            if (!visited[next])
                dfs(next);
        visit_stack.push(node);
    }

    void scc(int node)
    {
        visited[node] = true;
        for (int next:reversed_edges[node])
            if (!visited[next])
                scc(next);
        components.back().push_back(node);
    }

    void build_scc()
    {
        visited = vector<bool>(V, false);
        for (int node = 0; node < V; node++)
```

```
            if (!visited[node])
                dfs(node);

        visited = vector<bool>(V, false);
        while (!visit_stack.empty())
        {
            int node = visit_stack.top();
            visit_stack.pop();
            if (!visited[node]) {
                components.push_back(vector<int>());
                scc(node);
            }
        }

        component_number.resize(V);
        for (int i = 0; i < components.size(); i++)
            for (int node:components[i])
                component_number[node] = i;

        vector<bool> is_source = vector<bool>(components.size(), true);
        components_edges.resize(components.size());
        for (int u = 0; u < V; u++)
            for (int v:edges[u])
            {
                int cu = component_number[u];
                int cv = component_number[v];
                if (cu == cv)
                    continue;
                components_edges[cu].push_back(cv);
                is_source[cv] = false;
            }

        for (int component = 0; component < components.size(); component++) {
            if (is_source[component])
                source_components.push_back(component);
        }
    }
};

int main(void)
{
    int V, E;
    cin >> V >> E;

    Graph graph(2 * V + 1);
    for (int i = 0; i < E; i++)
    {
        int u, v;
        cin >> u >> v;
        graph.append(-u + V, v + V);
        graph.append(-v + V, u + V);
    }

    graph.build_scc();
```

```
    vector<int> last_component(2 * V + 1, -1);
    bool is_answer = true;
    for (int i = 0; i < graph.components.size(); i++)
    {
        for (int node:graph.components[i])
        {
            int negation = -(node - V) + V;
            if (last_component[negation] == i)
                is_answer = false;
            last_component[node] = i;
        }
    }

    if (is_answer) {
        vector<int> result(V);
        for (int i = 1; i <= V; i++)
        {
            int val = i + V;
            int negation = -i + V;
            result[i - 1] = graph.component_number[val] >
graph.component_number[negation];
        }

        for (int val:result)
            cout << val << " ";
        cout << "\n";
    }
}
```

1.9. Maximum flow(dinic)

```
struct FlowNetwork
{
    const llint INF = (1ll << 60ll);

    int n;
    vector<vector<llint>> capacities, flows;
    FlowNetwork(vector<vector<llint>> &capacities): capacities(capacities)
    {
        n = capacities.size();
        flows.assign(n, vector<llint>(n, 0));
    }

    vector<int> find_route(int source, int sink)
    {
        vector<int> parents(n, -1);
        parents[source] = source;
        queue<int> q;
        q.push(source);
        while (!q.empty())
```

```
        {
            int node = q.front();
            q.pop();

            for (int next = 0; next < n; next++)
            {
                int remain = capacities[node][next] - flows[node][next];
                if (remain > 0 && parents[next] == -1) {
                    parents[next] = node;
                    if (next == sink)
                        return parents;
                    q.push(next);
                }
            }
        }

        return parents;
    }

    llint calculate_max_flow(int source, int sink)
    {
        llint total = 0;
        while (true)
        {
            vector<int> parents = find_route(source, sink);
            if (parents[sink] == -1)
                break;

            llint min_remain = INF;
            for (int node = sink; node != source; node = parents[node])
            {
                int parent = parents[node];
                min_remain = min(min_remain, capacities[parent][node] -
flows[parent][node]);
            }

            for (int node = sink; node != source; node = parents[node])
            {
                int parent = parents[node];
                flows[parent][node] += min_remain;
                flows[node][parent] -= min_remain;
            }

            total += min_remain;
        }

        return total;
    }

    llint send_flow(int node, int sink, llint flow, vector<bool> &visited)
    {
        if (node == sink)
            return flow;
```

```
        visited[node] = true;
        for (int next = 0; next < n; next++)
        {
            llint remain = min(flow, capacities[node][next] - flows[node][next]);
            if (remain > 0ll && !visited[next]) {
                llint ret = send_flow(next, sink, remain, visited);
                if (ret > 0ll) {
                    flows[node][next] += ret;
                    flows[next][node] -= ret;
                    return ret;
                }
            }
        }
        return 0;
    }

    llint calculate_max_flow_dfs(int source, int sink)
    {
        llint total = 0;
        vector<bool> visited(n, false);
        while (llint flow = send_flow(source, sink, INF, visited) > 0)
        {
            total += flow;
            visited.assign(n, false);
        }

        return total;
    }
};
```

1.10. Maximum flow(adj)

```
const int INF = (1 << 30) - 1;

struct Edge
{
    Edge *reverse_edge = NULL;
    int dest, capacity, flow = 0;
    Edge(int dest, int capacity)
        : dest(dest), capacity(capacity) { }
    void add_flow(int value)
    {
        flow += value;
        reverse_edge->flow -= value;
    }
};
typedef Edge* EdgePtr;

struct FlowNetwork
{
    int n;
```

```
    vector<vector<EdgePtr>> edges;
    FlowNetwork(vector<vector<pair<int, int>>> &adj)
    {
        n = adj.size();
        edges.resize(n);
        for (int node = 0; node < n; node++)
        {
            for (auto &it:adj[node])
            {
                Edge *edge = new Edge(it.first, it.second);
                Edge *reverse_edge = new Edge(node, 0);
                edge->reverse_edge = reverse_edge;
                reverse_edge->reverse_edge = edge;

                edges[node].push_back(edge);
                edges[it.first].push_back(reverse_edge);
            }
        }
    }

    pair<vector<int>, vector<int>> find_route(int source, int sink)
    {
        vector<int> parents(n, -1);
        vector<int> indicies(n, -1);
        parents[source] = source;
        indicies[source] = source;
        queue<int> q;
        q.push(source);
        while (!q.empty())
        {
            int node = q.front();
            q.pop();

            for (int i = 0; i < edges[node].size(); i++)
            {
                auto &edge = *edges[node][i];
                int remain = edge.capacity - edge.flow;
                int next = edge.dest;
                if (remain > 0 && parents[next] == -1) {
                    parents[next] = node;
                    indicies[next] = i;
                    if (next == sink)
                        return {parents, indicies};
                    q.push(next);
                }
            }
        }

        return {parents, indicies};
    }

    int calculate_max_flow(int source, int sink)
    {
        int total = 0;
```

```
        while (true)
        {
            auto [parents, indicies] = find_route(source, sink);
            if (parents[sink] == -1)
                break;

            int min_remain = INF;
            for (int node = sink; node != source; node = parents[node])
            {
                int parent = parents[node];
                auto &edge = *edges[parent][indicies[node]];
                min_remain = min(min_remain, edge.capacity - edge.flow);
            }

            for (int node = sink; node != source; node = parents[node])
            {
                int parent = parents[node];
                auto &edge = *edges[parent][indicies[node]];
                edge.add_flow(min_remain);
            }
            total += min_remain;
        }
        return total;
    }
};
```

2. Tree

2.1. segment tree

```
struct SegmentTree
{
    int n;
    vector<llint> nodes;

    SegmentTree(int n): n(n)
    {
        nodes.resize(4 * n + 1);
    }

    void update(int idx, llint val)
    {
        sub_update(idx, val, 1, 0, n - 1);
    }

    void sub_update(int idx, llint val, int node, int l, int r)
    {
        if (l >= r) {
            nodes[node] = val;
            return;
        }
```

```
        int mid = (l + r) / 2;
        int left_node = node * 2;
        int right_node = node * 2 + 1;
        if (idx <= mid)
            sub_update(idx, val, left_node, l, mid);
        else
            sub_update(idx, val, right_node, mid + 1, r);
        nodes[node] = max(nodes[left_node], nodes[right_node]);
    }

    llint query(int left, int right)
    {
        return sub_query(left, right, 1, 0, n - 1);
    }

    llint sub_query(int left, int right, int node, int l, int r)
    {
        if (left <= l && r <= right)
            return nodes[node];

        if (r < left || right < l)
            return 0;

        int mid = (l + r) / 2;
        return max(sub_query(left, right, node * 2, l, mid)
                , sub_query(left, right, node * 2 + 1, mid + 1, r));
    }
};
```

2.2. segment tree with lazy propagation

```
struct SegmentTree
{
    vector<llint> vec, lazy;
    SegmentTree()
    {
        vec.resize(4 * N + 1);
        lazy.resize(4 * N + 1);
    }

    void update(int idx, llint val, int node = 1, int l = 0, int r = N - 1)
    {
        vec[node] += val;
        if (l >= r)
            return;

        int mid = (l + r) / 2;
        if (idx <= mid)
            update(idx, val, node * 2, l, mid);
        else
```

```
            update(idx, val, node * 2 + 1, mid + 1, r);
    }

    void update_range(int left, int right, llint val, int node = 1, int l = 0, int
r = N - 1)
    {
        if (r < left || l > right)
            return;

        if (left <= l && r <= right) {
            lazy[node] += val;
            return;
        }

        int mid = (l + r) / 2;
        update_range(left, right, val, node * 2, l, mid);
        update_range(left, right, val, node * 2 + 1, mid + 1, r);
    }

    llint query(int left, int right, int node = 1, int l = 0, int r = N - 1)
    {
        if (r < left || l > right)
            return 0;

        vec[node] += lazy[node] * (r - l + 1);
        if (left <= l && r <= right) {
            lazy[node] = 0;
            return vec[node];
        }

        lazy[node * 2] += lazy[node];
        lazy[node * 2 + 1] += lazy[node];
        lazy[node] = 0;

        int mid = (l + r) / 2;
        llint lval = query(left, right, node * 2, l, mid);
        llint rval = query(left, right, node * 2 + 1, mid + 1, r);
        return lval + rval;
    }
};
```

2.3. merge sort tree

```
llv1 a;
llv1 mTree[Mx];
void makeTree(ll idx, ll ss, ll se) {
    if(ss == se) {
        mTree[idx].push_back(a[ss]);
        return;
    }
```

```
    ll mid = (ss+se)/2;

    makeTree(2*idx+1, ss, mid);
    makeTree(2*idx+2, mid+1, se);

    merge(mTree[2*idx+1].begin(), mTree[2*idx+1].end(), mTree[2*idx+2].begin(),
mTree[2*idx+2].end(), back_inserter(mTree[idx]));
}

ll query(ll node, ll start, ll end, ll q_s, ll q_e, ll k) {
    //i j k: Ai, Ai+1, ..., Aj로 이루어진 부분 수열 중에서 k보다 큰 원소의 개수를 출력
한다.
    if (q_s > end || start > q_e) return 0;

    if (q_s <= start && q_e >= end) {
        return mTree[node].size() - (upper_bound(mTree[node].begin(),
mTree[node].end(), k) - mTree[node].begin());
    }

    ll mid = (start+end)/2;
    ll p1 = query(2*node+1, start, mid, q_s, q_e, k);
    ll p2 = query(2*node+2, mid+1, end, q_s, q_e, k);
    return p1 + p2;
}
```

2.4. LCA

```
vector<vector<int>> get_sparse_table(vector<int> &vec, int size)
{
    vector<vector<int>> table(size, vector<int>(vec.size()));
    for (int i = 0; i < vec.size(); i++)
        table[0][i] = vec[i];

    for (int i = 1; i < size; i++)
        for (int j = 0; j < vec.size(); j++)
            table[i][j] = table[i - 1][table[i - 1][j]];
    return table;
}

void dfs(int node, int depth, vector<vector<int>> &edges, vector<bool> &visited,
vector<int> &parents, vector<int> &depths)
{
    visited[node] = true;
    for (int next:edges[node])
        if (!visited[next]) {
            parents[next] = node;
            depths[next] = depth + 1;
            dfs(next, depth + 1, edges, visited, parents, depths);
        }
}
```

```
int get_lca(int u, int v, vector<vector<int>> &table, vector<int> &depths)
{
    if (depths[v] > depths[u])
        swap(v, u);

    int du = depths[u];
    int dv = depths[v];
    if (du > dv) {
        int gap = du - dv;
        for (int i = 0; i < table.size(); i++)
            if (gap & (1 << i))
                u = table[i][u];
    }
    if (u == v)
        return u;

    for (int i = table.size() - 1; i >= 0; i--)
    {
        if (table[i][u] != table[i][v]) {
            u = table[i][u];
            v = table[i][v];
        }
    }

    return table[0][u];
}

int main(void)
{
    vector<bool> visited(N, false);
    vector<int> parents(N, 0);
    vector<int> depths(N, 0);
    dfs(0, 0, edges, visited, parents, depths);

    vector<vector<int>> table = get_sparse_table(parents, 20);

    int u, v;
    cin >> u >> v;
    get_lca(u, v, table, depths);
}
```

2.5. Fenwick Tree 2D

```
struct FenwickTree2D {
    ll size;
    llv2 data;

    FenwickTree2D(ll N) {
        size = N;
        data = llv2(size+1, llv1(size+1));
    }
```

```
    void update(int x, int y, ll val) {
        ll dv = val - sum(x, y, x, y);
        while(x <= size) {
            int y2 = y;
            while(y2 <= size) {
                data[x][y2] += dv;
                y2 += y2 & -y2;
            }
            x += x & -x;
        }
    }

    ll sum(int x, int y) {
        ll ret = 0;
        while(x) {
            int y2 = y;
            while(y2) {
                ret += data[x][y2];
                y2 -= y2 & -y2;
            }
            x -= x & -x;
        }
        return ret;
    }

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

ll N, M;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    cin >> N >> M;

    FenwickTree2D F = FenwickTree2D(N);

    for1(1, N+1) {
        for1j(1, N+1) {
            ll a;
            cin >> a;
            F.update(i,j,a);
        }
    }

    while(M--) {
        ll w,a,b,c,d;
        cin >> w;
        if(w == 0) {
```

```
            cin >> a >> b >> c;
            F.update(a,b,c);
        }
        else {
            cin >> a >> b >> c >> d;
            cout << F.sum(a,b,c,d) << "\n";
        }
    }

}
```

3. String

3.1. KMP

```
vector<int> build_lps(string str)
{
    vector<int> lps(str.size());
    lps[0] = 0;

    int current = 0;
    for (int i = 1; i < str.size(); i++)
    {
        while (current > 0 && str[i] != str[current])
            current = lps[current - 1];

        if (str[i] == str[current])
            lps[i] = ++current;
    }
    return lps;
}

int kmp_search(string H, string N)
{
    vector<int> lps = build_lps(N);
    int cnt = 0, current = 0;
    for (int i = 0; i < H.size(); i++)
    {
        if (current > 0 && H[i] != N[current])
            current = lps[current - 1];

        if (H[i] == N[current]) {
            if (++current == N.size()) {
                cnt++;
                current = lps[current - 1];
            }
        }
    }
    return cnt;
}
```

3.2. Trie

```
int chToIdx(char ch) { return ch - 'a'; }
struct Trie {
    int terminal = -1;
    Trie* fail; // fail, output은 아호 코라식에 사용
    vector<int> output;
    Trie* chil[ALPHABETS];
    Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            chil[i] = NULL;
    }
    ~Trie() {
        for (int i = 0; i < ALPHABETS; i++)
            if (chil[i])
                delete chil[i];
    }
    // number -> 문자열 번호(ith string)
    void insert(string& s, int number, int idx) {
        if (idx == s.size()) {
            terminal = number;
            return;
        }
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            chil[next] = new Trie();
        chil[next]->insert(s, number, idx + 1);
    }
    int find(string& s, int idx = 0) {
        if (idx == s.size())
            return terminal;
        int next = chToIdx(s[idx]);
        if (chil[next] == NULL)
            return false;
        return chil[next]->find(s, idx + 1);
    }
};
```

3.3 Aho-Corasick

```
void computeFail(Trie* root) {
    queue<Trie*> q;
    root->fail = root;
    q.push(root);
    while (!q.empty()) {
        Trie* here = q.front();
        q.pop();
        for (int i = 0; i < ALPHABETS; i++) {
            Trie* child = here->chil[i];
            if (!child) continue;
```

```
            if (here == root)
                child->fail = root;
            else {
                Trie* t = here->fail;
                while (t != root && t->chil[i] == NULL)
                    t = t->fail;
                if (t->chil[i]) t = t->chil[i];
                child->fail = t;
            }
            child->output = child->fail->output;
            if (child->terminal != -1)
                child->output.push_back(child->terminal);
            q.push(child);
        }
    }
}
vector<pair<int, int>> ahoCorasick(string& s, Trie* root) {
    vector<pair<int, int>> ret;
    Trie* state = root;
    for (int i = 0; i < s.size(); i++) {
        int idx = chToIdx(s[i]);
        while (state != root && state->chil[idx] == NULL)
            state = state->fail;
        if (state->chil[idx])
            state = state->chil[idx];
        for (int j = 0; j < state->output.size(); j++)
            ret.push_back({ i, state->output[j] });
    }
    return ret;
}
```

3.4 SuffixArray

```
struct SuffixComparator {
    const vector<int>& group;
    int t;
    SuffixComparator(const vector<int>& _group, int _t) :group(_group), t(_t) { }
    bool operator() (int a, int b) {
        if (group[a] != group[b])
            return group[a] < group[b];
        return group[a + t] < group[b + t];
    }
};
vector<int> getSuffixArr(const string& s) {
    int n = s.size();
    int t = 1;
    vector<int> group(n + 1);
    for (int i = 0; i < n; i++) group[i] = s[i];
    group[n] = -1;
    vector<int> perm(n);
    for (int i = 0; i < n; i++) perm[i] = i;
```

```
    while (t < n) {
        SuffixComparator compare(group, t);
        sort(perm.begin(), perm.end(), compare);
        t *= 2;
        if (t >= n) break;
        vector<int> new_group(n + 1);
        new_group[n] = -1;
        new_group[perm[0]] = 0;
        for (int i = 1; i < n; i++)
            if (compare(perm[i - 1], perm[i]))
                new_group[perm[i]] = new_group[perm[i - 1]] + 1;
            else
                new_group[perm[i]] = new_group[perm[i - 1]];
        group = new_group;
    }
    return perm;
}
int getHeight(const string& s, vector<int>& pos) // 최장 중복 부분 문자열의 길이
{
    const int n = pos.size();
    vector<int> rank(n);
    for (int i = 0; i < n; i++)
        rank[pos[i]] = i;
    int h = 0, ret = 0;
    for (int i = 0; i < n; i++)
    {
        if (rank[i] > 0) {
            int j = pos[rank[i] - 1];
            while (s[i + h] == s[j + h])
                h++;
            ret = max(ret, h);
            if (h > 0)
                h--;
        }
    }
    return ret;
}
```

3.5 Manacher

```
// Use space to insert space between each character
// To get even length palindromes!

vector<int> manacher(string& s){
    int n = s.size(), R = -1, p = -1;
    vector<int> A(n);
    for(int i=0; i<n; i++){
        if(i <= R) A[i] = min(A[2*p-i], R-i);
        while(i-A[i]-1 >= 0 && i+A[i]+1 < n && s[i-A[i]-1] == s[i+A[i]+1]) A[i]++;
        if(i+A[i] > R) R = i+A[i], p = i;
    }
```

```
    return A;
}

string space(string& s){
    string t;
    for(char c: s) t+= c, t+= ' ';
    t.pop_back();
    return t;
}

int maxpalin(vector<int>& M, int i){
    if(i%2) return (M[i]+1)/2*2;
    return M[i]/2*2 + 1;
}
```

4. Geometry

4.1 Vector2

```
const double EPSILON = 1e-10;

struct Vector2
{
    double x, y;
    Vector2(): x(0), y(0) {}
    Vector2(double x, double y): x(x), y(y) { }
    Vector2(const Vector2 &other): x(other.x), y(other.y) { }
    double norm()
    {
        return sqrt(x * x + y * y);
    }
    Vector2 normalized()
    {
        Vector2 result = *this / norm();
        return result;
    }
    double dot(Vector2 rhs)
    {
        return x * rhs.x + y * rhs.y;
    }
    double cross(Vector2 rhs)
    {
        return x * rhs.y - y * rhs.x;
    }
    Vector2 operator+(Vector2 rhs)
    {
        Vector2 result(x + rhs.x, y + rhs.y);
        return result;
    }
    Vector2 operator+=(Vector2 rhs)
    {
```

```
        x += rhs.x;
        y += rhs.y;
        return *this;
    }
    Vector2 operator-(Vector2 rhs)
    {
        Vector2 result(x - rhs.x, y - rhs.y);
        return result;
    }
    Vector2 operator-=(Vector2 rhs)
    {
        x -= rhs.x;
        y -= rhs.y;
        return *this;
    }
    Vector2 operator-()
    {
        Vector2 result(-x, -y);
        return result;
    }
    Vector2 operator*(double scalar)
    {
        Vector2 result(x * scalar, y * scalar);
        return result;
    }
    Vector2 operator/(double scalar)
    {
        Vector2 result(x / scalar, y / scalar);
        return result;
    }
    bool operator==(Vector2 rhs)
    {
        return x == rhs.x && y == rhs.y;
    }
    bool operator<(Vector2 rhs)
    {
        if (x == rhs.x)
            return y < rhs.y;
        return x < rhs.x;
    }
    bool operator<=(Vector2 rhs)
    {
        if (x == rhs.x)
            return y <= rhs.y;
        return x <= rhs.x;
    }
    bool operator>(Vector2 rhs)
    {
        return rhs < *this;
    }
    bool operator>=(Vector2 rhs)
    {
        return rhs <= *this;
    }
```

```
};

bool is_intersect(Vector2 a, Vector2 b, Vector2 c, Vector2 d)
{
    double ret1 = (b - a).cross(c - a) * (b - a).cross(d - a);
    double ret2 = (d - c).cross(a - c) * (d - c).cross(b - c);

    if (ret1 == 0 && ret2 == 0) {
        if (a > b)
            swap(a, b);
        if (c > d)
            swap(c, d);
        return a <= d && c <= b;
    }

    return ret1 <= 0 && ret2 <= 0;
}

pair<bool, Vector2> get_intersection(Vector2 a, Vector2 b, Vector2 c, Vector2 d)
{
    if (a > b)
        swap(a, b);
    if (c > d)
        swap(c, d);
    if (a > c) {
        swap(a, c);
        swap(b, d);
    }

    if (!is_intersect(a, b, c, d))
        return {false, Vector2(NAN, NAN)};

    Vector2 dir1 = (b - a).normalized();
    Vector2 dir2 = (d - c).normalized();
    double den = dir1.cross(dir2);
    if (-EPSILON <= den && den <= EPSILON) {
        if (b == c)
            return {true, b};
        return {true, Vector2(NAN, NAN)};
    }
    else {
        double l = (c - a).cross(dir1) / den;
        Vector2 intersection = c + dir2 * l;
        return {true, intersection};
    }
}
```

4.2 Convenx Hull

```
vector<Vector2> get_convex_hull(vector<Vector2> points)
{
```

```
    sort(points.begin(), points.end());
    Vector2 start_point = points[0];
    for (Vector2 &point:points)
        point -= start_point;

    stable_sort(points.begin(), points.end(), [](Vector2 A, Vector2 B) {
        return A.cross(B) > 0;
    });

    vector<Vector2> lasts;
    for (Vector2 &point:points)
    {
        while (lasts.size() > 1)
        {
            Vector2 current = lasts.back();
            Vector2 last = lasts[lasts.size() - 2];
            if ((current - last).cross(point - last) > 0.0)
                break;
            lasts.pop_back();
        }
        lasts.push_back(point);
    }
    for (Vector2 &point:lasts)
        point += start_point;

    return lasts;
}
```

4.3 Separating Axis Theorem

```
pair<double, double> get_projection(vector<Vector2> &points, Vector2 &axis)
{
    double min_val = axis.dot(points[0]);
    double max_val = min_val;
    for (int i = 1; i < points.size(); i++)
    {
        double projected = axis.dot(points[i]);
        min_val = min(min_val, projected);
        max_val = max(max_val, projected);
    }
    return {min_val, max_val};
}

vector<Vector2> get_normals(vector<Vector2> &points)
{
    vector<Vector2> ret;
    if (points.size() == 1)
        return ret;

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

```
        Vector2 &a = points[i];
        Vector2 &b = points[(i + 1) % points.size()];
        ret.push_back(Vector2((b - a).y, -(b - a).x));
    }
    return ret;
}

bool can_separate(vector<Vector2> &A, vector<Vector2> &B)
{
    if (A.size() == 1 && B.size() == 1)
        return true;

    auto c_a = get_convex_hull(A);
    auto c_b = get_convex_hull(B);
    auto n_a = get_normals(c_a);
    auto n_b = get_normals(c_b);
    n_a.insert(n_a.end(), n_b.begin(), n_b.end());
    if (c_a.size() > 1)
        n_a.push_back(Vector2(c_a[1] - c_a[0]));
    if (c_b.size() > 1)
        n_a.push_back(Vector2(c_b[1] - c_b[0]));

    for (Vector2 &axis:n_a)
    {
        auto p_a = get_projection(c_a, axis);
        auto p_b = get_projection(c_b, axis);
        if (!((p_a.second >= p_b.first) && (p_b.second >= p_a.first)))
            return true;
    }
    return false;
}
```

4.4 Two Far Points

```
pair<Vector2, Vector2> get_max_points(vector<Vector2> &points)
{
    int left = 0, right = max_element(points.begin(), points.end()) -
points.begin();
    int ret1 = left, ret2 = right;
    double max_len = (points[right] - points[left]).norm();
    int end = right;

    Vector2 left_dir = Vector2(0, -1.0);

    vector<Vector2> edges;
    for (int i = 0; i < points.size(); i++)
        edges.push_back((points[(i + 1) % points.size()] -
points[i]).normalized());

    while (right != 0 || left != end)
    {
```

```
        double next1 = left_dir.dot(edges[left]);
        double next2 = -left_dir.dot(edges[right]);
        if (left != end && (right == 0 || next1 > next2)) {
            left_dir = edges[left];
            left = (left + 1) % points.size();
        }
        else {
            left_dir = -edges[right];
            right = (right + 1) % points.size();
        }

        double len = (points[right] - points[left]).norm();
        if (len > max_len) {
            ret1 = left;
            ret2 = right;
            max_len = len;
        }
    }
    return { points[ret1], points[ret2] };
}
```

5. Extra

5.1. Treap

```
// Treap* root = NULL;
// root = insert(root, new Treap(3));
typedef int type;
struct Treap {
    Treap* left = NULL, * right = NULL;
    int size = 1, prio = rand();
    type key;
    Treap(type key) : key(key) { }
    void calcSize() {
        size = 1;
        if (left != NULL) size += left->size;
        if (right != NULL) size += right->size;
    }
    void setLeft(Treap* l) { left = l, calcSize(); }
    void setRight(Treap* r) { right = r, calcSize(); }
};
typedef pair<Treap*, Treap*> TPair;
TPair split(Treap* root, type key) {
    if (root == NULL) return TPair(NULL, NULL);
    if (root->key < key) {
        TPair rs = split(root->right, key);
        root->setRight(rs.first);
        return TPair(root, rs.second);
    }
    TPair ls = split(root->left, key);
    root->setLeft(ls.second);
```

```
    return TPair(ls.first, root);
}
Treap* insert(Treap* root, Treap* node) {
    if (root == NULL) return node;
    if (root->prio < node->prio) {
        TPair s = split(root, node->key);
        node->setLeft(s.first);
        node->setRight(s.second);
        return node;
    }
    else if (node->key < root->key)
        root->setLeft(insert(root->left, node));
    else
        root->setRight(insert(root->right, node));
    return root;
}
Treap* merge(Treap* a, Treap* b) {
    if (a == NULL) return b;
    if (b == NULL) return a;
    if (a->prio < b->prio) {
        b->setLeft(merge(a, b->left));
        return b;
    }
    a->setRight(merge(a->right, b));
    return a;
}
Treap* erase(Treap* root, type key) {
    if (root == NULL) return root;
    if (root->key == key) {
        Treap* ret = merge(root->left, root->right);
        delete root;
        return ret;
    }
    if (key < root->key)
        root->setLeft(erase(root->left, key));
    else
        root->setRight(erase(root->right, key));
    return root;
}
Treap* kth(Treap* root, int k) { // kth key
    int l_size = 0;
    if (root->left != NULL) l_size += root->left->size;
    if (k <= l_size) return kth(root->left, k);
    if (k == l_size + 1) return root;
    return kth(root->right, k - l_size - 1);
}
int countLess(Treap* root, type key) { // count less than key
    if (root == NULL) return 0;
    if (root->key >= key)
        return countLess(root->left, key);
    int ls = (root->left ? root->left->size : 0);
    return ls + 1 + countLess(root->right, key);
}
```

5.2. MCC

```
double getR(double x, double y){
    return x*x + y*y;
}

double avg(vector<double> x){
    double ans=0;
    for(int i=0; i<sz(x); i++) ans+=x[i];
    return ans/sz(x);
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    double inputx, inputy, rx, ry, distance, lr=1;
    int n, index;
    vector<double> x, y;

    cin >> n;
    for1(0, n){
        cin >> inputx >> inputy;
        x.pb(inputx);
        y.pb(inputy);
    }

    rx = avg(x);
    ry = avg(y);

    for1(0, 100000){
        distance = -1; index = -1;
        for1j(0, n){
            if(distance < getR(x[j] - rx, y[j] - ry)){
                distance = getR(x[j] - rx, y[j] - ry);
                index = j;
            }
        }
        rx = rx + (x[index] - rx) * lr;
        ry = ry + (y[index] - ry) * lr;
        lr *= 0.999;
    }

    cout << fixed;
    cout.precision(2);
    cout << sqrt(distance) << endl;

    return 0;
}
```

5.3. ExtendEuclid

```
int gcd(int a, int b){
    if(b==0) return a;
    return gcd(b, a%b);
}

// ax+by=gcd(a,b)
pii ext_gcd(int a, int b){
    if(b==0) return pii(1, 0);
    pii tmp = ext_gcd(b, a%b);
    return pii(tmp.second, tmp.first - (a/b) * tmp.second);
}

// ax = 1 (mod b)
ll mod_inv(int a, int b){
    return (ext_gcd(a, b).first + b) % b;
}

/*
Ax + By = C일때
x0 = s * C/D
y0 = t * C/D
x = x0 + k * B/D
y = y0 - k * A/D
*/
```

5.4. Fermat

```
// p는 무조건 소수
ll pow(ll a, ll b){
    if(b == 0) return 1;
    ll n = pow(a, b/2)%p;
    ll temp = (n * n)%p;

    if(b%2==0) return temp;
    return (a * temp)%p;
}

ll fermat(ll a, ll b){
    return a%p*pow(b, p-2)%p;
}
```

5.5. FFT

```
const double PI = acos(-1);
typedef complex<double> cpx;
```

```
void FFT(vector<cpx> &f, cpx w){
    int n = f.size();
    if(n == 1) return;

    vector<cpx> even(n/2), odd(n/2);
    for(int i = 0; i < n; ++i)
        (i%2 ? odd : even)[i/2] = f[i];

    FFT(even, w*w);
    FFT(odd, w*w);

    cpx wp(1, 0);
    for(int i = 0; i < n/2; ++i){
        f[i] = even[i] + wp*odd[i];
        f[i + n/2] = even[i] - wp*odd[i];
        wp *= w;
    }
}

vector<cpx> multiply(vector<cpx> a, vector<cpx> b){
    int n = 1;
    while(n < a.size()+1 || n < b.size()+1) n *= 2;
    n *= 2;
    a.resize(n);
    b.resize(n);
    vector<cpx> c(n);

    cpx w(cos(2*PI/n), sin(2*PI/n));

    FFT(a, w);
    FFT(b, w);

    for(int i = 0; i < n; ++i)
        c[i] = a[i]*b[i];

    FFT(c, cpx(1, 0)/w);
    for(int i = 0; i < n; ++i){
        c[i] /= cpx(n, 0);
        c[i] = cpx(round(c[i].real()), round(c[i].imag()));
    }
    return c;
}
```

5.6. ConvexHullTrick

```
struct linear{
    ll a, b;
    double s;
};

ll dp[MAX], top=0;
```

```
linear f[MAX];

double cross(linear &f, linear &g){
    return (g.b-f.b)/(f.a-g.a);
}

void addLine(ll a, ll b){ // y = ax + b
    linear g({a, b, 0});
    while(top > 0){
        g.s = cross(f[top-1], g);
        if(f[top-1].s < g.s) break;
        top--;
    }
    f[top++] = g;
}

ll searchLine(ll x){
    ll pos = top-1;
    if(x < f[top-1].s){
        ll lo = 0, hi = top-1;
        while(lo+1 < hi){
            ll mid = (lo+hi)/2;
            (x < f[mid].s ? hi:lo) = mid;
        }
        pos = lo;
    }
    return pos;
}
```

5.7. LIS

```
void lis(){
    int n, i, x;
    iv1 v, buffer;
    iv1::iterator vv;
    vector<pair<int, int> > print;
    v.pb(2000000000);

    cin >> n;
    for1(0, n){
        cin >> x;
        if(x > *v.rbegin()) {
            v.pb(x);
            print.push_back({v.size()-1, x});
        }
        else{
            vv = lower_bound(v.begin(), v.end(), x);
            *vv = x;
            print.push_back({vv-v.begin(), x});
        }
    }
```

```
    cout << sz(v) << endl;

    for(i=sz(print)-1;i>-1;i--){
        if(print[i].first == sz(v)-1){
            buffer.pb(print[i].second);
            v.pop_back();
        }
    }
    for(i=sz(buffer)-1;i>-1;i--) cout << buffer[i] << " ";
}
```

5.8. Knapsack

```
ll N, maxWeight,ans;
ll D[2][11000];
ll weight[110], cost[110];

void knapsack() {
    for(int x=1; x<=N; x++) {
        for(int y=0; y<=maxWeight; y++) {
            if(y>=weight[x]) {
                D[x%2][y] = max(D[(x+1)%2][y],D[(x+1)%2][y-weight[x]]+cost[x]);
            }
            else {
                D[x%2][y] = D[(x+1)%2][y];
            }
            ans = max(ans, D[x%2][y]);
        }
    }
}

void input() {
    cin >> N >> maxWeight;
    for(int x=1; x<=N; x++) {
        cin >> weight[x] >> cost[x];
    }
}
```

5.9. Coin Change

```
// 경우의 수
ll CC(llv1& coin, ll money, ll MX) {
    ll D[MX];
    fill(D, D+MX, 0);
    D[0] = 1;
    for(int i = coin.size()-1; i >=0; i--) {
        for(int j = coin[i]; j <= money; j++) {
            D[j] += D[j - coin[i]];
            D[j] %= MOD;
```

```
            }
        }
        return D[money] % MOD;
    }
```

5.10. Knuth Opti

```
int solve(int n) {
    for (int m = 2; m <= n; m++) {
        for (int i = 0; m + i <= n; i++) {
            int j = i + m;
            for (int k = K[i][j - 1]; k <= K[i + 1][j]; k++) {
                int now = dp[i][k] + dp[k][j] + sum[j] - sum[i];
                if (dp[i][j] > now)
                    dp[i][j] = now, K[i][j] = k;
            }
        }
    }
    return dp[0][n];
}

int main() {
    int n;
    cin >> n;
    fill(&dp[0][0], &dp[MAX-1][MAX-1], INF);
    for (int i = 1; i <= n; i++){
        cin >> arr[i];
        sum[i] = sum[i - 1] + arr[i];
        K[i - 1][i] = i;
        dp[i - 1][i] = 0;
    }
    cout << solve(n) << "\n";
}

/*
if
C[a][c] + C[b][d] <= C[a][d] + C[b][c] (a<=b<=c<=d)
C[b][c] <= C[a][d] (a<=b<=c<=d)

then
dp[i][j] = min(dp[i][k] + dp[k][j]) + C[i][j]
range of k: A[i, j-1] <= A[i][j]=k <= A[i+1][j]
*/
```

5.11. twonearpoint

```
struct Point {
    int x, y;
};
```

```
int dist(Point &p, Point &q) {
    return (p.x-q.x)*(p.x-q.x)+(p.y-q.y)*(p.y-q.y);
}

struct Comp {
    bool comp_in_x;
    Comp(bool b) : comp_in_x(b) {}
    bool operator()(Point &p, Point &q) {
        return (this->comp_in_x? p.x < q.x : p.y < q.y);
    }
};

int nearest(vector<Point>::iterator it, int n) {
    if (n == 2)
        return dist(it[0], it[1]);
    if (n == 3)
        return min({dist(it[0], it[1]), dist(it[1], it[2]), dist(it[2], it[0])});

    int line = (it[n/2 - 1].x + it[n/2].x) / 2;
    int d = min(nearest(it, n/2), nearest(it + n/2, n - n/2));

    vector<Point> mid;

    for (int i = 0; i < n; i++) {
        int t = line - it[i].x;
        if (t*t < d)
            mid.push_back(it[i]);
    }

    sort(mid.begin(), mid.end(), Comp(false));

    int mid_sz = mid.size();
    for (int i = 0; i < mid_sz - 1; i++)
        for (int j = i + 1; j < mid_sz && (mid[j].y - mid[i].y)*(mid[j].y -
mid[i].y) < d; j++)
            d = min(d, dist(mid[i], mid[j]));

    return d;
}
```

5.12. Bit Field DP

```
#define MOD 9901;
using namespace std;

int dp[1<<14+1][200];
int n, m;
int solve(int pos,int check,int dep)
{
    if(dp[check][pos]!=0)return dp[check][pos];
```

```
    int &ret = dp[check][pos];

    if(dep == n*m)return ret = 1;

    if((check&1)) return ret = solve(pos-1,check>>1,dep)%MOD;

    int sum = 0;
    if(!(check&1) && (pos-1) / m > 0)
    sum += solve(pos-1,(check>>1)|(1<<(m-1)),dep+2)%MOD;
    if(!(check&1) && pos%m!=1 && !(check&2)&& pos>=2 && m>1)
    sum += solve(pos-2,check>>2,dep+2)%MOD;

    //cout<<pos<<" "<<check<<" "<<dep<<" "<<sum<<endl;
    return ret = sum%MOD;
}
int main() {
    cin>>n>>m;
    if(n*m % 2 == 1)cout<<0;
    else
    cout<<solve(n*m,0,0)%MOD;
    return 0;
}
```