

This block contains a large amount of code, likely from a competitive programming contest, spanning multiple files and sections. The code is organized into several main sections: Balance courses, Max-distance-subarray, Count solutions, Inversion, Bridges and articulations, Disjoint segments, and Max subarray. Each section contains multiple files (e.g., CBUS, CVRP, etc.) with their respective code snippets.

Bus-inter-city (BFS to build weighted directed graph and dijkstra)

```

int n, m;
vector<int> c, d, visited, dp;
vector<vector<int>> adj;
vector<long long> dis;
vector<vector<pair<int, int>> adj1;
void bfs(int i, int limit)
{
    fill(visited.begin(), visited.end(), 0);
    queue<pair<int, int>> q;
    visited[i] = 1;
    while (!q.empty())
    {
        int u = q.front().first;
        int depth = q.front().second;
        q.pop();
        if (u != i)
        {
            adj1[i].push_back({u, c[i]});
        }
        if (depth >= limit) continue;

        for (auto v : adj[u])
        {
            if (!visited[v])
            {
                visited[v] = 1;
                q.push({v, depth + 1});
            }
        }
    }
}
void dijkstra(int s)
{
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>> pq;
    dis[s] = 0;
    pq.push({0, s});

    for (int i = 0; i < n; i++)
    {
        int u = -1;

        while (!pq.empty())
        {
            u = pq.top().second;
            pq.pop();
            if (!visited[u]) break;
        }
        if (u == -1 || dis[u] == INT_MAX) break;
        for (auto edge : adj1[u])
        {
            int v = edge.first;
            int w = edge.second;

            if (dis[u] + w < dis[v])
            {
                dis[v] = dis[u] + w;
                pq.push({dis[v], v});
            }
        }
    }
    int main()
    {
        cin >> n >> m;
        c.resize(n + 1, 0);
        d.resize(n + 1, 0);
        dp.resize(n + 1, 0);
        visited.resize(n + 1, 0);
        adj1.resize(n + 1, vector<pair<int>>());
        adj1.resize(n + 1, vector<pair<int, int>>());
        dis.resize(n + 1, INT_MAX);
        for (int i = 1; i <= n; i++)
        {
            cin >> c[i] >> d[i];
        }
        for (int i = 1; i <= m; i++)
        {
            int u, v;
            cin >> u >> v;
            adj1[u].push_back(v);
            adj1[v].push_back(u);
        }
        for (int i = 1; i <= n; i++)
        {
            bfs(i, d[i]);
        }
        fill(visited.begin(), visited.end(), 0);
        dijkstra(1);
        cout << dis[n] << "\n";
        return 0;
    }
}

```

Make-span-schedule (topo sort and DP)

```

int n, m;
vector<vector<int>> adj;
vector<int> d;
vector<int> visited;
vector<int> order;
int hasCycle = 0;
int ans = 0;
vector<int> res;
void topo(int u)
{
    if (visited[u] == 2) return; // done
    if (visited[u] == 1)
    {
        hasCycle = 1;
        // cycle
        return;
    }
    visited[u] = 1;
    for (auto v : adj[u])
    {
        topo(v);
        if (hasCycle) return;
    }
    visited[u] = 2;
    order.push_back(u);
}
int main()
{
    cin >> n >> m;
    adj.resize(n + 1, vector<int>());
    visited.resize(n + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        cin >> d[i];
    }
    adj.resize(n + 1, vector<int>());
    visited.resize(n + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }
    for (int i = 1; i <= n; i++)
    {
        if (!visited[i]) topo(i);
    }
    reverse(order.begin(), order.end());
    for (auto u : order)
    {
        f[u] += d[u];
        for (auto v : adj[u])
        {
            f[v] = max(f[v], f[u]);
        }
        ans = max(ans, f[u]);
    }
    cout << ans << "\n";
    return 0;
}
int main()
{
    cin >> n >> m;
    vector<vector<int>> adj;
    // adj[task] = list of staffs
    // that can do this task
    vector<int> matchTask;
    // matchTask[staff] = task
    // assigned to this staff (0 if none)
    vector<bool> visited;
    // Kuhn's algorithm
    // (DFS to find augmenting path)
    bool dfs(int task)
    {
        for (int staff : adj[task])
        {
            if (visited[staff]) continue;
            visited[staff] = true;
            // If staff is free or we can
            // reassign their current task
            if (matchTask[staff] == 0 ||
                !dfs(matchTask[staff]))
            {
                matchTask[staff] = task;
                return true;
            }
        }
        return false;
    }
    int main()
    {
        cin >> n >> m;
        adj.resize(n + 1);
        matchTask.resize(m + 1, 0);
        for (int i = 1; i <= n; i++)
        {
            int k;
            cin >> k;
            for (int j = 0; j < k; j++)
            {
                int staff;
                cin >> staff;
                adj[i].push_back(staff);
            }
        }
        int result = 0;
        for (int task = 1; task <= n; task++)
        {
            visited.assign(m + 1, false);
            if (dfs(task))
            {
                result++;
            }
        }
        cout << result << "\n";
        return 0;
    }
}

```

SCC

```

int n, m;
vector<vector<int>> adj;
vector<int> num;
vector<int> low;
stack<int> SCC;
vector<int> onStack;
int current_num = 0, cnt = 0;
vector<vector<int>> res;
void dfs(int u)
{
    current_num++;
    num[u] = current_num;
    low[u] = current_num;
    SCC.push(u);
    onStack[u] = 1;
    for (auto v : adj[u])
    {
        if (num[v] == 0)
        {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if (onStack[v] == 1)
        {
            low[u] = min(low[u], num[v]);
        }
    }
    if (low[u] == num[u])
    {
        cnt++;
        vector<int> res1;
        while (onStack[u])
        {
            res1.push_back(SCC.top());
            SCC.pop();
            onStack[u] = 0;
        }
        res.push_back(res1);
    }
}
int main()
{
    if (!(!cin >> m >> n >> k)) return 0;

    sz = max(n, m);
    a.assign(sz + 1, vector<long long>(sz + 1, 0));
    for (int i = 0; i < k; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        if (a[x][y] == 0) a[x][y] = -w;
        else a[x][y] = min(a[x][y], (long long)-w);
    }

    u.assign(sz + 1, 0);
    v.assign(sz + 1, 0);
    p.assign(sz + 1, 0);
    way.assign(sz + 1, 0);
    for (int i = 1; i <= sz; i++) You, 15 hours ago < 0
    {
        p[i] = i;
        int j0 = 0;
        minv.assign(sz + 1, INF);
        vector<bool> used(sz + 1, false);
        do
        {
            used[j0] = true;
            int i0 = p[j0];
            long long delta = INF;
            int j1;
            for (int j = 1; j <= sz; j++)
            {
                if (!used[j])
                {
                    if (v == u)
                    {
                        sort(res.begin(), res.end());
                        res.push_back(res1);
                        // cout << "\n";
                        break;
                    }
                }
            }
            int v = SCC.top();
            SCC.pop();
            onStack[v] = false;
            res1.push_back(v);
            // cout << " " >> v;
            if (v == u)
            {
                sort(res.begin(), res.end());
                res.push_back(res1);
                // cout << "\n";
                break;
            }
        }
        for (int i = 1; i <= m; i++)
        {
            cin >> n >> m;
            adj.resize(n + 1, vector<int>());
            num.resize(n + 1, 0);
            low.resize(n + 1, 0);
            onStack.resize(n + 1, 0);
            for (int i = 1; i <= m; i++)
            {
                int v, u;
                cin >> u >> v;
                adj[u].push_back(v);
            }
            for (int i = 1; i <= n; i++)
            {
                if (num[i] == 0) {
                    dfs(i);
                }
            }
            cout << cnt << "\n";
            sort(res.begin(), res.end());
            for (auto i : res)
            {
                for (auto j : i)
                {
                    cout << j << " ";
                }
                cout << "\n";
            }
            return 0;
        }
    }
}

```

Hungary

```

const long long INF = 1e18; // Giá trị vô cùng lớn cho long long
int n, m, k, sz;
vector<vector<long long>> a; // Ma trận chỉ phai (dâ dào đầu)
vector<long long> u, v, minv;
vector<int> p, way; // p: hung matching, way: mang truy vết
int main()
{
    if (!(!cin >> n >> m >> k)) return 0;

    sz = max(n, m);
    a.assign(sz + 1, vector<long long>(sz + 1, 0));
    for (int i = 0; i < k; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        if (a[x][y] == 0) a[x][y] = -w;
        else a[x][y] = min(a[x][y], (long long)-w);
    }

    u.assign(sz + 1, 0);
    v.assign(sz + 1, 0);
    p.assign(sz + 1, 0);
    way.assign(sz + 1, 0);
    for (int i = 1; i <= sz; i++) You, 15 hours ago < 0
    {
        p[i] = i;
        int j0 = 0;
        minv.assign(sz + 1, INF);
        vector<bool> used(sz + 1, false);
        do
        {
            used[j0] = true;
            int i0 = p[j0];
            long long delta = INF;
            int j1;
            for (int j = 1; j <= sz; j++)
            {
                if (!used[j])
                {
                    if (v == u)
                    {
                        sort(res.begin(), res.end());
                        res.push_back(res1);
                        // cout << "\n";
                        break;
                    }
                }
            }
            int v = SCC.top();
            SCC.pop();
            onStack[v] = false;
            res1.push_back(v);
            // cout << " " >> v;
            if (v == u)
            {
                sort(res.begin(), res.end());
                res.push_back(res1);
                // cout << "\n";
                break;
            }
        }
        for (int i = 1; i <= m; i++)
        {
            cin >> n >> m;
            adj.resize(n + 1, vector<int>());
            num.resize(n + 1, 0);
            low.resize(n + 1, 0);
            onStack.resize(n + 1, 0);
            for (int i = 1; i <= m; i++)
            {
                int v, u;
                cin >> u >> v;
                adj[u].push_back(v);
            }
            for (int i = 1; i <= n; i++)
            {
                if (num[i] == 0) {
                    dfs(i);
                }
            }
            cout << cnt << "\n";
            sort(res.begin(), res.end());
            for (auto i : res)
            {
                for (auto j : i)
                {
                    cout << j << " ";
                }
                cout << "\n";
            }
            return 0;
        }
    }
}

```

Edmond-karp (max flow)

```

int n, m;
vector<vector<int>> capacity;
vector<vector<int>> adj;
int bfs(int source, int sink, vector<int>& parent)
{
    fill(parent.begin(), parent.end(), -1);
    parent[source] = source;
    queue<pair<int, int>> q;
    q.push({source, INT_MAX});
    while (!q.empty())
    {
        int u = q.front().first;
        int flow = q.front().second;
        q.pop();
        for (int v : adj[u])
        {
            if (parent[v] == -1 && capacity[u][v] > 0)
            {
                parent[v] = u;
                int new_flow = min(flow, capacity[u][v]);
                q.push({v, new_flow});
            }
        }
    }
    return 0;
}
int edmond_karp(int source, int sink)
{
    int max_flow = 0;
    vector<int> parent(n + 1);
    int new_flow;
    while ((new_flow = bfs(source, sink, parent)) > 0)
    {
        max_flow += new_flow;
        int cur = sink;
        while (cur != source)
        {
            int prev = parent[cur];
            capacity[prev][cur] -= new_flow;
            capacity[cur][prev] += new_flow;
            cur = prev;
        }
    }
    return max_flow;
}
int main()
{
    cin >> n >> m;
    capacity.resize(n + 1, vector<int>(n + 1, 0));
    adj.resize(n + 1);
    for (int i = 1; i <= m; i++)
    {
        int u, v, c;
        cin >> u >> v >> c;
        capacity[u][v] += c;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    int source, sink;
    cin >> source >> sink;
    cout << edmond_karp(source, sink) << "\n";
    return 0;
}

```

Max-matching-bipartite

```

int n, m;
vector<vector<int>> adj;
// adj[task] = list of staffs
// that can do this task
vector<int> matchTask;
// matchTask[staff] = task
// assigned to this staff (0 if none)
vector<bool> visited;
// Kuhn's algorithm
// (DFS to find augmenting path)
bool dfs(int task)
{
    for (int staff : adj[task])
    {
        if (visited[staff]) continue;
        visited[staff] = true;
        // If staff is free or we can
        // reassign their current task
        if (matchTask[staff] == 0 ||
            !dfs(matchTask[staff]))
        {
            matchTask[staff] = task;
            return true;
        }
    }
    return false;
}
int main()
{
    cin >> n >> m;
    adj.resize(n + 1);
    matchTask.resize(m + 1, 0);
    for (int i = 1; i <= n; i++)
    {
        int k;
        cin >> k;
        for (int j = 0; j < k; j++)
        {
            int staff;
            cin >> staff;
            adj[i].push_back(staff);
        }
    }
    int result = 0;
    for (int task = 1; task <= n; task++)
    {
        visited.assign(m + 1, false);
        if (dfs(task))
        {
            result++;
        }
    }
    cout << result << "\n";
    return 0;
}

```

Total-path-lengths

```

int n;
vector<vector<pair<int, int>>> a(MAX + 1);
vector<int> subtree_size(MAX + 1, 1);
vector<int> visited(MAX + 1, 0);
vector<long long> answer(MAX + 1, 0);

void dfs(int u, int depth)
{
    visited[u] = 1;
    answer[1] += depth;
    for (auto path : a[u])
    {
        int v = path.first;
        int w = path.second;
        if (visited[v] == 0)
        {
            dfs(v, depth + subtree_size[v] * w); // or +1
            subtree_size[u] += subtree_size[v];
        }
    }
}

void dfs2(int u)
{
    visited[u] = 1;
    for (auto path : a[u])
    {
        int v = path.first;
        int w = path.second;
        if (visited[v] == 0)
        {
            // Cân bằng answer[v] TRUE khi gọi đệ quy xuống v
            answer[v] = answer[u] - w * subtree_size[v] + w * (n - subtree_size[v]); // 1 : weight
            dfs2(v);
        }
    }
}

int main()
{
    cin >> n;
    for (int i = 1; i < n; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        a[u].push_back(make_pair(v, w));
        a[v].push_back(make_pair(u, w));
    }
    dfs(1, 0);
    for (int i = 1; i <= n; i++) visited[i] = 0;
    dfs2(1);
    cout << *max_element(answer.begin(), answer.end()) << "\n";
    return 0;
}

```

Longest-path

```

int n;
vector<int> visited(MAX + 1, 0);
vector<vector<int>> a(MAX + 1);
vector<int> distances(MAX + 1);

void bfs(int i)
{
    queue<int> q;
    if (visited[i] == 1) return;
    q.push(i);
    visited[i] = 1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (auto path : a[u])
        {
            int v = path.first;
            int w = path.second;
            if (visited[v] == 0)
            {
                q.push(v);
                visited[v] = 1;
                distances[v] = distances[u] + w;
            }
        }
    }
}

int main()
{
    cin >> n;
    for (int i = 1; i < n; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        a[u].push_back(make_pair(v, w));
        a[v].push_back(make_pair(u, w));
    }
    bfs(1);
    int Max = 0;
    int start = 1;
    for (int i = 1; i <= n; i++)
    {
        if (Max < distances[i])
        {
            Max = distances[i];
            start = i;
        }
    }
    bfs(start);
    Max = 0;
    for (int i = 1; i <= n; i++)
    {
        if (Max < distances[i])
        {
            Max = distances[i];
        }
    }
    cout << Max << "\n";
    return 0;
}

```