Last update: June 10, 2024 Translated From: e-maxx.ru

Checking a graph for acyclicity and finding a cycle in $O(M)$

Consider a directed or undirected graph without loops and multiple edges. We have to check whether it is acyclic, and if it is not, then find any cycle.

We can solve this problem by using Depth First Search in $O(M)$ where M is number of edges.

Algorithm

We will run a series of DFS in the graph. Initially all vertices are colored white (0). From each unvisited (white) vertex, start the DFS, mark it gray (1) while entering and mark it black (2) on exit. If DFS moves to a gray vertex, then we have found a cycle (if the graph is undirected, the edge to parent is not considered). The cycle itself can be reconstructed using parent array.

Implementation

Here is an implementation for directed graph.

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

Here is an implementation for undirected graph. Note that in the undirected version, if a vertex `v` gets colored black, it will never be visited again by the DFS. This is because we already explored all connected edges of `v` when we first visited it. The connected component containing `v` (after removing the edge between `v` and its parent) must be a tree, if the DFS has completed processing `v` without finding a cycle. So we don't even need to distinguish between gray and black states. Thus we can turn the char vector `color` into a boolean vector `visited`.

```
int n;
vector<vector<int>> adj;
vector<bool> visited;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v, int par) { // passing vertex and its parent vertex
    visited[v] = true;
    for (int u : adj[v]) {
        if(u == par) continue; // skipping edge to parent vertex
        if (visited[u]) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
        parent[u] = v;
        if (dfs(u, parent[u]))
```

```
            return true;
    }
    return false;
}

void find_cycle() {
    visited.assign(n, false);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (!visited[v] && dfs(v, parent[v]))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}
```

Practice problems:

- AtCoder : Reachability in Functional Graph
- CSES : Round Trip
- CSES : Round Trip II

Contributors:

ankitsangwan1999 (32.84%) jakobkogler (25.37%) praran26 (24.63%) mtan429 (9.7%) Aryamn (2.99%) adamant-pwn (2.24%) roll-no-1 (1.49%) ladamalina (0.75%)