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

Table of Contents

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. From each unvisited vertex, start the DFS, mark it gray while entering and mark it black 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.

```cpp
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;
```

```cpp
    for (int v = 0; v < n; v++) {
        if (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_sta
            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;
    }
}
```