KAZAKH BRITISH TECHNICAL UNIVERSITY

LECTURE 11 – DFS & TOPOLOGICAL SORT

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 11

KAZAKH KBTU BRITISH TECHNICAL UNIVERSITY

In current lecture, we'll continue to learn & implement algorithms upon graphs, and cover other traversing algorithm Depth-First Search (DFS), and Topological Sort.

TOPICS WE'LL COVER:

- Depth-First Search
- Use Cases of DFS
- Topological Sort
- Practice Problems

GOALS FOR THIS LECTURE:

- Understand and implement Depth-First Search (DFS) using recursion and stack.
- Understand the concept of Topological Sorting and when it applies.
- Implement Topological Sort using DFS on Directed Acyclic Graphs (DAGs).

DEPTH-FIRST SEARCH

Depth-First Search explores a graph **deeply**, going as far as possible along each branch before backtracking.

Key properties:

- Uses *stack behavior* (recursion or explicit stack);
- Visits nodes in *depth order*;

Algorithm:

1. Pick a starting vertex;
2. Visit it;
3. Recursively visit all unvisited neighbors.

Time complexity: O(V + E).

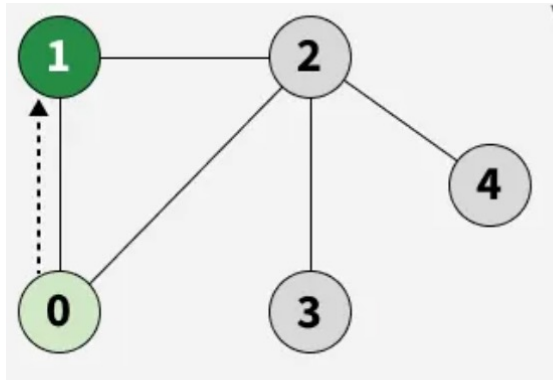
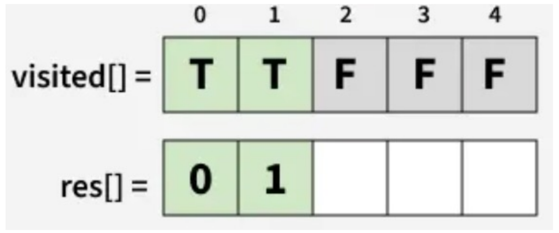


Figure 1 – Example graph [1]



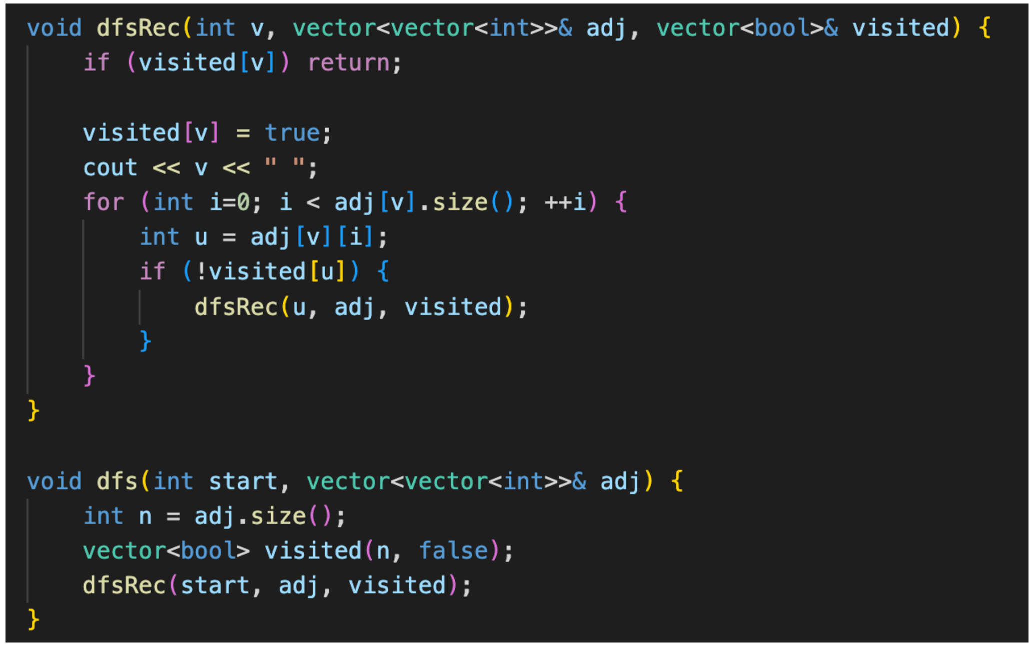
	0	1	2	3	4
visited[] =	T	T	F	F	F
res[] =	0	1			

Figure 2 – Marker of visited marker

[1] https://www.geeksforgeeks.org/dsa/depth-first-search-or-dfs-for-a-graph/

DEPTH-FIRST SEARCH (CONT.)

We can implement DFS algorithm with a recursive way, and a stack way as shown in Figures 3, and 4, respectively.

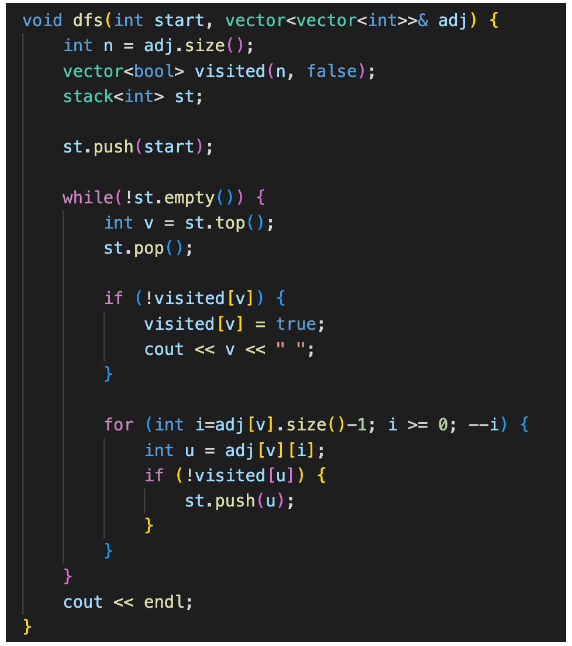


```
void dfsRec(int v, vector<vector<int>>& adj, vector<bool>& visited) {
    if (visited[v]) return;

    visited[v] = true;
    cout << v << " ";
    for (int i=0; i < adj[v].size(); ++i) {
        int u = adj[v][i];
        if (!visited[u]) {
            dfsRec(u, adj, visited);
        }
    }
}

void dfs(int start, vector<vector<int>>& adj) {
    int n = adj.size();
    vector<bool> visited(n, false);
    dfsRec(start, adj, visited);
}
```

Figure 3 – DFS with recursion



```
void dfs(int start, vector<vector<int>>& adj) {
    int n = adj.size();
    vector<bool> visited(n, false);
    stack<int> st;

    st.push(start);

    while(!st.empty()) {
        int v = st.top();
        st.pop();

        if (!visited[v]) {
            visited[v] = true;
            cout << v << " ";
        }

        for (int i=adj[v].size()-1; i >= 0; --i) {
            int u = adj[v][i];
            if (!visited[u]) {
                st.push(u);
            }
        }
    }
    cout << endl;
}
```

Figure 4 – DFS with stack

In Figure 4, the algorithm seems like BFS, but for the place of queue uses stack.

USE CASES OF DFS

The DFS algorithm can be implemented for solving many complex algorithms, like listed below.

Use cases:

- Cycle detection;
- Topological sorting;
- Strongly connected components;
- Solving puzzles (backtracking).

Practice problem:

- [207. Course Schedule]

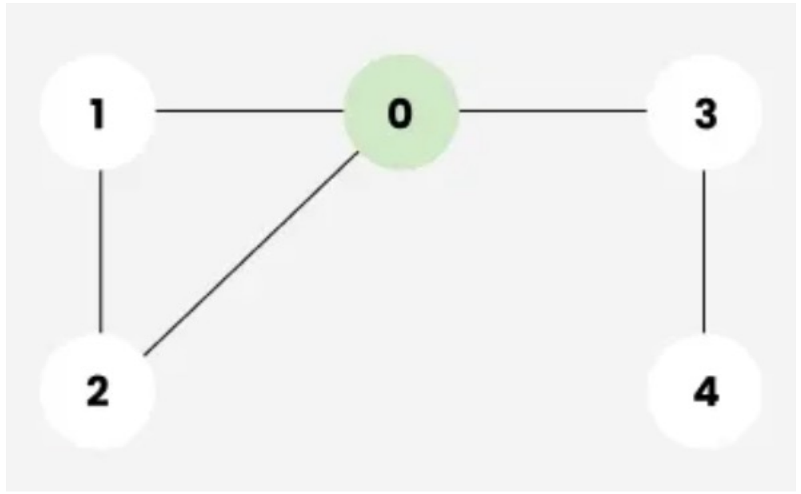


Figure 5 – Undirected graph with cycle [2]

[2] https://www.geeksforgeeks.org/dsa/detect-cycle-undirected-graph/

TOPOLOGICAL SORT

A **linear ordering of vertices** in a directed graph such that:

➢ *For every directed edge u → v, u appears before v in the ordering.*

✓ Works only for Directed Acyclic Graphs (DAGs).

Graph representation for Figure 6: {{1}, {2}, {}, {2, 4}, {}};

Applications:

- Ordering tasks with dependencies;
- Build systems (compilers);
- Course prerequisite problems;
- Scheduling.

Note: There may be several topological orderings for a graph.

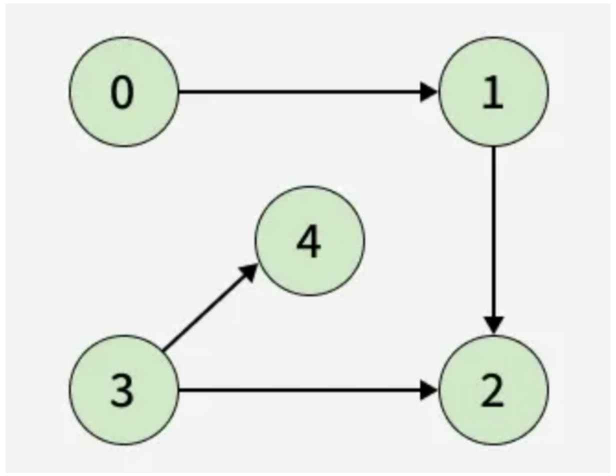


Figure 6 – Directed Acyclic Graph [3]

[3] https://www.geeksforgeeks.org/dsa/topological-sorting/

TOPOLOGICAL SORT (CONT.)

Depth-First Search

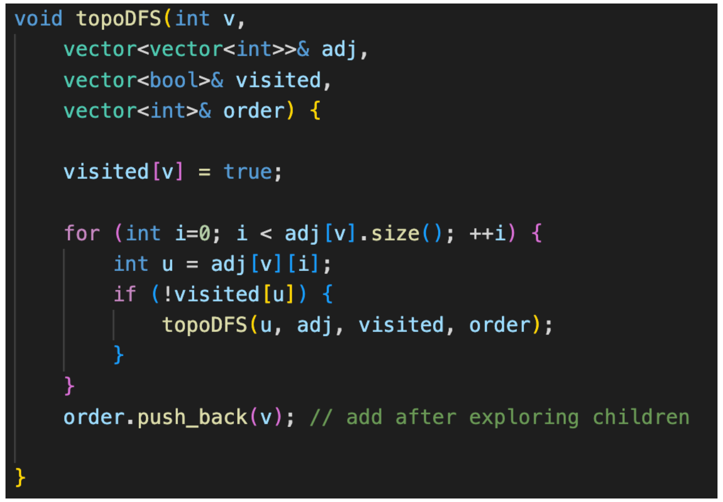
Use Cases of DFS

Topological Sort

Practice Problems

Idea behind Topological sort:

1. Perform DFS;
2. After exploring all neighbors of node u, push u into a stack;
3. Reverse stack → topological order.



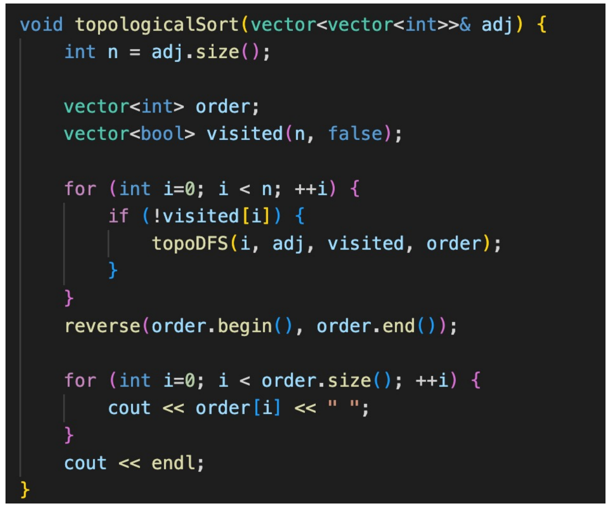
```cpp
void topoDFS(int v,
    vector<vector<int>>& adj,
    vector<bool>& visited,
    vector<int>& order) {

    visited[v] = true;

    for (int i=0; i < adj[v].size(); ++i) {
        int u = adj[v][i];
        if (!visited[u]) {
            topoDFS(u, adj, visited, order);
        }
    }
    order.push_back(v); // add after exploring children

}
```

Figure 7 – Topological Sort (1st function)



```cpp
void topologicalSort(vector<vector<int>>& adj) {
    int n = adj.size();

    vector<int> order;
    vector<bool> visited(n, false);

    for (int i=0; i < n; ++i) {
        if (!visited[i]) {
            topoDFS(i, adj, visited, order);
        }
    }
    reverse(order.begin(), order.end());

    for (int i=0; i < order.size(); ++i) {
        cout << order[i] << " ";
    }
    cout << endl;
}
```

Figure 8 – Topological Sort (main function)

Time Complexity: O(V+E)

PRACTICE PROBLEMS

We have two tasks to solve:

- 112. Path Sum;
- 802. Find Eventual Safe States.

Q & A