Teambook Sindicato de Transporte 2880

February 12, 2023

Contents

Chapter 1

Graphs

This chapter shows some of the basec algorithms and implementations required to solve problems that include graphs.

1.1 Depth First Search (DFS)

The DFS algorithm is a recursive algorithm that visits all the nodes of a graph. It is used to find connected components, topological sorting, and to find bridges and articulation points. The algorithm is as follows:

The implementation can be done as follows:

1.2 Breadth First Search (BFS)

The BFS algorithm is a non-recursive algorithm that visits all the nodes of a graph. It is used to find connected components, topological sorting, and to find bridges and articulation points, to better understand it, a propagating fire can be imagined. The algorithm is as follows:

The implementation can be done as follows:

```cpp
#include <bits/stdc++.h>

using namespace std;
signed main()
{
    vector<vector<int>> adj;  // adjacency list representation
    int n; // number of nodes
    int s; // source vertex

    queue<int> q;
    vector<bool> used(n);
    vector<int> d(n), p(n);

    q.push(s);
    used[s] = true;
    p[s] = -1;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int u : adj[v]) {
            if (!used[u]) {
                used[u] = true;
                q.push(u);
                d[u] = d[v] + 1;
                p[u] = v;
            }
        }
    }
    return 0;
}
```

Graphs/BFS.cpp

Algorithm 1 Depth First Search (DFS)

```
1: procedure DFS(G)
2:     visited ← ∅
3:     time ← 0
4:     parent ← ∅
5:     low ← ∅
6:     disc ← ∅
7:     AP ← ∅
8:     bridge ← ∅
9:     for all v ∈ V do
10:        visited[v] ← false
11:        parent[v] ← −1
12:        low[v] ← ∞
13:        disc[v] ← ∞
14:    end for
15:    for all v ∈ V do
16:        if visited[v] = false then
17:            DFSUtil(G, v, visited, time, parent, low, disc, AP, bridge)
18:        end if
19:    end for
20: end procedure
21: procedure DFSUtil(G, v, visited, time, parent, low, disc, AP, bridge)
22:    visited[v] ← true
23:    disc[v] ← time
24:    low[v] ← time
25:    time ← time + 1
26:    children ← 0
27:    for all u ∈ Adj(v) do
28:        if visited[u] = false then
29:            parent[u] ← v
30:            children ← children + 1
31:            DFSUtil(G, u, visited, time, parent, low, disc, AP, bridge)
32:            low[v] ← min(low[v], low[u])
33:            if parent[v] = −1 and children > 1 then
34:                AP[v] ← true
35:            end if
36:            if parent[v]! = −1 and low[u] ≥ disc[v] then
37:                AP[v] ← true
38:            end if
39:            if low[u] > disc[v] then
40:                bridge[v][u] ← true
41:            end if
42:        else
43:            low[v] ← min(low[v], disc[u])
44:        end if
45:    end for
```

Algorithm 2 Breadth First Search (BFS)

```
1: procedure BFS(G)
2:     visited ← ∅
3:     time ← 0
4:     parent ← ∅
5:     low ← ∅
6:     disc ← ∅
7:     AP ← ∅
8:     bridge ← ∅
9:     for all v ∈ V do
10:        visited[v] ← false
11:        parent[v] ← −1
12:        low[v] ← ∞
13:        disc[v] ← ∞
14:    end for
15:    for all v ∈ V do
16:        if visited[v] = false then
17:            BFSUtil(G, v, visited, time, parent, low, disc, AP, bridge)
18:        end if
19:    end for
20: end procedure
21: procedure BFSUtil(G, v, visited, time, parent, low, disc, AP, bridge)
22:    visited[v] ← true
23:    disc[v] ← time
24:    low[v] ← time
25:    time ← time + 1
26:    children ← 0
27:    for all u ∈ Adj(v) do
28:        if visited[u] = false then
29:            parent[u] ← v
30:            children ← children + 1
31:            BFSUtil(G, u, visited, time, parent, low, disc, AP, bridge)
32:            low[v] ← min(low[v], low[u])
33:            if parent[v] = −1 and children > 1 then
34:                AP[v] ← true
35:            end if
36:            if parent[v]! = −1 and low[u] ≥ disc[v] then
37:                AP[v] ← true
38:            end if
39:            if low[u] > disc[v] then
40:                bridge[v][u] ← true
41:            end if
42:        else
43:            low[v] ← min(low[v], disc[u])
44:        end if
45:    end for
```