Detect Negative Cycle using Bellman-Ford Algorithm Implementation

Roshan Tiwari

July 13, 2024

Introduction

The Bellman-Ford algorithm is used for finding the shortest path from a single source vertex to all other vertices in a weighted graph. Unlike Dijkstra's algorithm, Bellman-Ford can handle graphs with negative weight edges. Here, we provide an implementation of the Bellman-Ford algorithm in C++.

Implementation

The following code demonstrates the implementation of the Bellman-Ford algorithm:

```cpp
#include<bits/stdc++.h>
using namespace std;

struct node {
    int u, v, wt;
    node(int first, int second, int weight) {
        u = first;
        v = second;
        wt = weight;
    }
};

```

```cpp
int main() {
    int N = 6, m = 7;
    vector<node> edges;
    edges.push_back(node(0, 1, 5));
    edges.push_back(node(1, 2, -2));
    edges.push_back(node(1, 5, -3));
    edges.push_back(node(2, 4, 3));
    edges.push_back(node(3, 2, 6));
    edges.push_back(node(3, 4, -2));
    edges.push_back(node(5, 3, 1));

    int src = 0;
    int inf = 10000000;
    vector<int> dist(N, inf);
    dist[src] = 0;

    // Relax all edges N-1 times
    for(int i = 1; i <= N-1; i++) {
        for(auto it: edges) {
            if(dist[it.u] + it.wt < dist[it.v]) {
                dist[it.v] = dist[it.u] + it.wt;
            }
        }
        // Print distances after each iteration
        for(int i = 0; i < N; i++) {
            cout << dist[i] << " ";
        }
        cout << endl;
    }

    // Check for negative weight cycle
    int fl = 0;
    for(auto it: edges) {
        if(dist[it.u] + it.wt < dist[it.v]) {
```

```cpp
                cout << -1;
                fl = 1;
                break;
            }
        }

    if(!fl) {
        for(int i = 0; i < N; i++) {
            cout << dist[i] << " ";
        }
    }

    return 0;
}
```

Listing 1: Bellman-Ford Algorithm in C++

Explanation

- **Struct Definition**: Defines the structure `node` to store the edges of the graph, each represented by a source node `u`, a destination node `v`, and a weight `wt`.
- **Graph Initialization**: Initializes the graph with 6 nodes and 7 edges.
- **Distance Initialization**: Initializes the distance vector with a large value (infinity), except for the source node which is set to 0.
- **Relaxation Process**: Relaxes all edges `N-1` times to ensure the shortest paths are found.
- **Negative Cycle Detection**: Checks for the presence of negative weight cycles by attempting to relax the edges one more time.
- **Output**: Prints the shortest distances from the source node to all other nodes, or `-1` if a negative weight cycle is detected.

Iteration Table

The table below shows the values of the distance vector `dist` after each iteration of the relaxation process:

Iteration	Node 0	Node 1	Node 2	Node 3	Node 4	Node 5
0 (Initial)	0	∞	∞	∞	∞	∞
1	0	5	3	∞	∞	2
2	0	5	3	3	∞	2
3	0	5	3	3	1	2
4	0	5	3	3	1	2
5	0	5	3	3	1	2

Output

Given the provided graph and source node, the output of the algorithm is as follows:

```
0 5 3 3 1 2
```

This output represents the shortest distances from the source node 0 to all other nodes in the graph. If a negative weight cycle is detected, the output would be `-1`.

Conclusion

The Bellman-Ford algorithm is a powerful tool for finding the shortest paths in a weighted graph, particularly when negative weights are involved. Its ability to detect negative weight cycles makes it a valuable algorithm in various applications.