Path with Maximum Probability Using Dijkstra's Algorithm

Roshan Tiwari

July 13, 2024

Problem Statement

The problem requires us to find the path with the maximum probability of success to go from a given starting node to an ending node in an undirected weighted graph. The graph is represented as an edge list, where each edge has a probability of success associated with it. We need to return the success probability of the path with the maximum probability.

Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2

Output: 0.25000

Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.

Approach

To solve this problem, we will use a graph traversal algorithm called Dijkstra's algorithm. The algorithm finds the shortest path in a weighted graph, but in our case, we will modify it to find the path with the maximum probability.

1. Initialize a distance array `dist` with the starting node's index set to 1 and all other nodes set to 0. This array will store the maximum probability of reaching each node.
2. Create an adjacency list to represent the graph. Each element of the adjacency list will

be a pair `(neighbor, probability)`. This will help us traverse the graph efficiently.

3. Use a priority queue to implement Dijkstra's algorithm. The priority queue will store nodes based on their probabilities. We will start with the starting node and its probability of 1.

4. While the priority queue is not empty:

   (a) Extract the node with the maximum probability from the priority queue.

   (b) Update the maximum probability of reaching its neighboring nodes.

   (c) Add the neighboring nodes to the priority queue if their probabilities have been updated.

5. Finally, the maximum probability of reaching the ending node will be stored in `dist[end]`. Return this value as the result.

Full Solution

C++ Implementation

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

class Solution {
public:
    double maxProbability(int n, vector<vector<int>>& edges,
       vector<double>& succProb, int start, int end) {
        vector<vector<pair<int, double>>> graph(n);  // {a: [(b,
           prob_ab)]}
        priority_queue<pair<double, int>> maxHeap;   // (prob to
           reach u, u)
        maxHeap.emplace(1.0, start);
        vector<bool> seen(n);

        for (int i = 0; i < edges.size(); ++i) {
            const int u = edges[i][0];
```

```
            const int v = edges[i][1];
            const double prob = succProb[i];
            graph[u].emplace_back(v, prob);
            graph[v].emplace_back(u, prob);
        }

        while (!maxHeap.empty()) {
            const auto [prob, u] = maxHeap.top();
            maxHeap.pop();
            if (u == end)
                return prob;
            if (seen[u])
                continue;
            seen[u] = true;
            for (const auto& [nextNode, edgeProb] : graph[u]) {
                if (seen[nextNode])
                    continue;
                maxHeap.emplace(prob * edgeProb, nextNode);
            }
        }

        return 0;
    }
};

int main() {
    Solution solution;
    int n = 3;
    vector<vector<int>> edges = {{0, 1}, {1, 2}, {0, 2}};
    vector<double> succProb = {0.5, 0.5, 0.2};
    int start = 0, end = 2;
    double result = solution.maxProbability(n, edges, succProb,
       start, end);
    cout << fixed << setprecision(5) << result << endl;
```

```
47      return 0;
48  }
```

Listing 1: Path with Maximum Probability in C++

Explanation

- **Graph Initialization**: Initializes the graph with n nodes and edges with their respective success probabilities.
- **Priority Queue**: Uses a priority queue to store and extract nodes based on the maximum probability.
- **Traversal and Relaxation**: Implements the modified Dijkstra's algorithm to update the maximum probability of reaching each node.
- **Output**: Prints the maximum probability of reaching the ending node from the starting node.

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
0 (Initial)	1.0	0.0	0.0
1	1.0	0.5	0.0
2	1.0	0.5	0.25

Output

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

```
0.25000
```

This output represents the maximum probability of reaching the ending node 2 from the starting node 0.

Conclusion

Here, we explored the problem of finding the path with the maximum probability in a graph. We discussed the problem statement, explained the approach using a modified Dijkstra's algorithm, and implemented the solution in C++.