EFFICIENT SHORTEST PATH ALGORITHM (AMESPA) IN DIRECTED GRAPH AND WEIGHTED GRAPH USING C++*

Anshul Mittal[1]

[1]amsbctech, Bengaluru, Karnataka
anshulmttl@gmail.com

ABSTRACT

This paper provides the shortest path algorithm for directed and undirected and weighted graph using C++.It is an efficient shortest path algorithm. The c++ code, Algorithm and Psuedo code is provided in this paper.

KEYWORDS

AMESPA, C++, Shortest Path, Algorithm, Graph, Data structures, Algorithm

1. INTRODUCTION

This paper provides the shortest path algorithm for directed and undirected and weighted graph in C++. This algorithm is fast and tested on small datasets. It combines the concepts of C++ and algorithms to provide an efficient algorithm for finding shortest path in a directed graph. The basic concepts of struct, inbuilt standard library, static variable are used. It provides shortest path algorithms for directed graph, undirected graph, and weighted graph. The same algorithm can be extended to other languages like java, c# and python and more which support concepts of objects and pointers/references. The algorithm is provided as step based, psuedo code and c++ implementation.

2. ALGORITHMS:

2.1 ALGORITHM, PSUEDO-CODE AND C++ CODE:

The following algorithm is for shortest path in directed / undirected weighted graph. The algorithm can be used for computing longest path as well.

2.1.1 ALGORITHM:

- **Step 1** : Create structure Edge with start node, end node and weight. Create structure Node and ShortestPaths. Extra code is added for creating edges, nodes and shortest paths using c++ structure.
- **Step 2** : Mark the start_node and end_node as integers for which shortest path has to be found.
- **Step 3**: Create vector<vector<Node*>> previous_iteration_pointers of nodes which maintains a vector<Node*> added at current iteration. Vector is a sequence here.

- **Step 4**: Create a Node* start_node_ with the marked start_node and add it to previous_iteration_pointers as vector of single element. Create all first edges from start node and add it to previous_iteration_pointers as vector<Node*>
- **Step 5**: Set the variables shortest_path_length as maximum integer value supported by that variable and boolean bNewNodeAdded as true. In c++ it is INT_MAX.
- **Step 6**: Run a do – while loop in C++ until bNewNodeAdded is false.
- **Step 7** : Set the bNewNodeAdded to false.
- **Step 8** : For each node in previous iteration do the following.
- **Step 8.1** : Create a ventor vector<Node*> pointers2;
- **Step 8.2** : If path already found for this node goto step 8.4.
- **Step 8.3** : For each edge in graph do the following operations.
- **Step 8.3.1** : if edge start node is equal to node id of step 8 node and path for current in step 8 not found and previous node of step 8 is not same as edge end node, . If given a directed graph don't need last one condition in step 8.2.1.
- **Step 8.3.2** : Set bNewNodeAdded to true;
- **Step 8.3.3** : Construct new Node* from weight of node, edge start node, edge end node as follows.
    - Node* new_node = new Node().
    - new_node->m_PreviousNode = node of step 2.
    - new_node->m_NodeId =edge end node
    - if edge end node is end node set new_node->bPathFound to true else set to false.
    - If edge end node is equal to set end node set the the new_node->bPathFound to true;
        - If current path length is less than shortest_path_length, set the shortest_path_length to current path length and set shortest_path_end_node to to new_node.
    - Add new_node to pointers2 created in step 8.1.
- **Step 8.4** : Exit loop from Step 8.

2.1.2 PSUEDO-CODE:

- SET start_node, end_node;
- SET shortest_path_length = INT_MAX;
- SET Node* start_node__ with m_NodeId = start_node, m_bPathFound = false, m_PreviousNode = NULL
- DECLARE VARIABLE vector<vector<Node*>> previous_iteration_pointers or reference typy in java / c#.

- Add start_node_ to previous_iterationPointers.
- Create all initial nodes directly connected in graph to start_node_ and add as vector to previous_iteration_pointers.
- SET boolean variable bNewNodeAdded to true.
- do

```
{
        SET bNewNodeAdded to false;
for(each pointer it in previous_iteration_pointers.back())
{
        if(it->m_bPathFound == true)
        {
                goto end;
        }

        for(each pointer it1 in edges)
        {
                Perform steps in 8.3 in algorithm b.
        }

        end:
}
}while(bNewNodeAdded);
```

2.1.3 C++ CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#include <fstream>
#include <sstream>
#include <chrono>
using namespace std;


// The node id for edge should start from 1 and not 0
struct Edge
{
   Edge()
   {
      m_StartNode = 0;
      m_EndNode = 0;
      m_Weight = 0;
   }

   ~Edge()
```

```
    {

    }

    Edge(unsigned int start_node, unsigned int end_node, unsigned int weight)
    {
        m_StartNode = start_node;
        m_EndNode = end_node;
        m_Weight = weight;
    }

    unsigned int m_StartNode;
unsigned int m_EndNode;
    unsigned int m_Weight;
};

struct Node
{
    vector<Node*> m_NextNodes;
    Node* m_PreviousNode;
    unsigned int m_PathLength = 0;
    bool m_bPathFound;
    unsigned int m_NodeId;
};

struct ShortestPaths
{
    Node* m_Path;
    unsigned int m_LengthShortestPath;
    bool m_bIsShortestPath;
};
using namespace chrono;
int main()
{
    vector<Edge> edges;

    ifstream ifs;
    ifs.open("/mnt/d/Work/DataSetShortestPath.txt");
    std::string line = "";
    while(std::getline(ifs,line))
    {
        stringstream check1(line);
        vector<string> tokens;
        string intermediate;
        while(std::getline(check1, intermediate, ' '))
        {
            tokens.push_back(intermediate);
        }
```

```
        edges.push_back(Edge(stoi(tokens[0].c_str()),stoi(tokens[1].c_str()),1));
        tokens.clear();
    }
    // keep track of paths while calculating shortest paths.
// Iterate through all edges and calculate shortest path.
    // Set a destination node and start node.
    //unsigned int start_node = 1;
    //unsigned int end_node = 5;
    unsigned int start_node = 1;
    unsigned int end_node = 255750;
    //unsigned int end_node = 13827;

    //vector<ShortestPaths*> paths;
    ShortestPaths paths;
    bool bFirstIteration = true;

    unsigned int path_length = INT_MAX;
    bool bShortestPathFound = false;

    Node* start_node_ = new Node();
    start_node_->m_NodeId = start_node;
    start_node_->m_bPathFound = false;
    start_node_->m_PreviousNode = NULL;
    paths.m_Path = start_node_;

    vector<vector<Node*>> previous_iteration_pointers;
    vector<Node*> pointers;
    pointers.push_back(start_node_);
    vector<Node*> pointers1;
    for(auto it : edges)
    {
        if(it.m_StartNode == start_node)
        {
            Node* new_node = new Node();
            new_node->m_PreviousNode = start_node_;
            new_node->m_NodeId = it.m_EndNode;
            new_node->m_bPathFound = false;
            if(it.m_EndNode == end_node)
            {
                new_node->m_bPathFound = true;
            }
            new_node->m_PathLength = it.m_Weight;
            start_node_->m_NextNodes.push_back(new_node);

            pointers1.push_back(new_node);
```

```
      }
   }
   previous_iteration_pointers.push_back(pointers);
   previous_iteration_pointers.push_back(pointers1);

   //unsigned int shortest_path_index = INT_MAX;
   //unsigned int shortest_path_distance = INT_MAX;
   unsigned int shortest_path_length = INT_MAX;

   Node* shortest_path_end_node = NULL;

   //for(unsigned int i = 0; i < edges.size(); i++)
   unsigned int i = 0;
   bool bNewNodeAdded = true;
   do
   {
      cout << "Iterations executed " << endl;
      bNewNodeAdded = false;
      for(auto it : previous_iteration_pointers.back())
      {
         vector<Node*> pointers2;
         if(it->m_bPathFound)
         {
            continue;
         }

         // iterate the nodes of previous iteration
         for(auto it1 : edges)
         {
            if(it1.m_StartNode == it->m_NodeId && (it->m_PreviousNode->m_NodeId
!= it1.m_EndNode) && !it->m_bPathFound)
            {
               bNewNodeAdded = true;
               Node* new_node = new Node();
               new_node->m_PreviousNode = it;
               new_node->m_NodeId = it1.m_EndNode;
               new_node->m_bPathFound = false;
               if(it1.m_EndNode == end_node)
               {
                  new_node->m_bPathFound = true;
if(it->m_PathLength + it1.m_Weight < shortest_path_length)
                  {
                     shortest_path_length = it->m_PathLength + it1.m_Weight;
                     shortest_path_end_node = new_node;
                  }
               }
               new_node->m_PathLength = it->m_PathLength + it1.m_Weight;
               it->m_NextNodes.push_back(new_node);
```

```
                pointers2.push_back(new_node);
              }
            }
            previous_iteration_pointers.push_back(pointers2);
        }
    }while(bNewNodeAdded);
    milliseconds etms = duration_cast< milliseconds >(system_clock::now().time_since_epoch());
    cout << "End time " << etms.count() << endl;

    if(shortest_path_end_node != NULL)
    {
        cout << "Shortest path length is " << shortest_path_length << endl;
        cout << "Shortest path is " << endl;
        do
        {
            cout << shortest_path_end_node->m_NodeId << " ";
            shortest_path_end_node = shortest_path_end_node->m_PreviousNode;
        }while(shortest_path_end_node->m_PreviousNode);
    }
}
```

2.2 C++ CODE FOR SHORTEST PATH IN UN-WEIGHTED GRAPH DIRECTED/UNDIRECTED GRAPH.

The following c++ code is for shortest path in directed and undirected graph from which algorithm can be derived.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Level
{
    void AddElementCurrentLevel(int element)
    {
        if(current_level.empty())
        {
            pathLength++;
        }
        current_level.push_back(element);
    }
    int GetPathLength()
    {
        return pathLength;
    }
    vector<int> current_level;
    static int pathLength;
```

```
};
int Level::pathLength = 0;
int main()
{
   vector<std::pair<int,int>> edges;
   edges.push_back(make_pair<int,int>(1,2));
   edges.push_back(make_pair<int,int>(1,5));
   edges.push_back(make_pair<int,int>(1,4));
   edges.push_back(make_pair<int,int>(4,1));
   edges.push_back(make_pair<int,int>(2,3));
   edges.push_back(make_pair<int,int>(2,1));
   edges.push_back(make_pair<int,int>(3,4));
   edges.push_back(make_pair<int,int>(3,2));
   edges.push_back(make_pair<int,int>(4,6));
   edges.push_back(make_pair<int,int>(4,3));
   edges.push_back(make_pair<int,int>(6,5));
   edges.push_back(make_pair<int,int>(6,4));
   edges.push_back(make_pair<int,int>(5,1));
   edges.push_back(make_pair<int,int>(5,6));
   vector<Level> current_level;
   Level level1;
   level1.AddElementCurrentLevel(1);
   current_level.push_back(level1);
   int startNode = 1;
   int endNode = 4;
   int shortestPathLength = 1000000;
   bool bShortestPathFound = false;
   while(true)
   {
      Level level;
      vector<int> previousNodes = current_level.back().current_level;
      for(auto it : edges)
      {
         if(std::find(previousNodes.begin(), previousNodes.end(), it.first) !=
previousNodes.end())
         {
            level.AddElementCurrentLevel(it.second);

            if(it.second == endNode)
            {
               shortestPathLength = level.GetPathLength();
               bShortestPathFound = true;
               break;
            }
         }

      }
      current_level.push_back(level);
```

```
        if(bShortestPathFound == true)
        {
            break;
        }
    }
    cout << "Shortest path from 2 to 4 is " << shortestPathLength - 1 << endl;
}
```

3. TESTS AND RESULTS:

The algorithm was tested using 4 core processor with 8 threads using the dataset provided in [1]. The average time taken to searching graph for a path which does not exist was 832 milliseconds in which maximum length scanned was 11136.

4. REFERENCES

[1] Test Graph database : https://snap.stanford.edu/data/email-EuAll.html

Authors

Anshul Mittal : I am owner of amsbctech
located at Bengaluru india.