Shortest Path in Directed Acyclic Graph

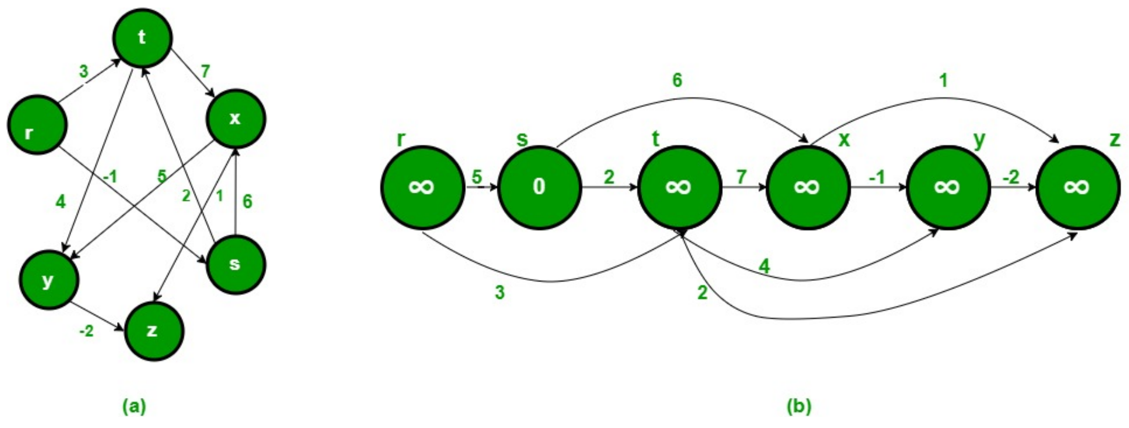
Given a Weighted Directed Acyclic Graph and a source vertex in the graph, find the shortest paths from given source to all other vertices.

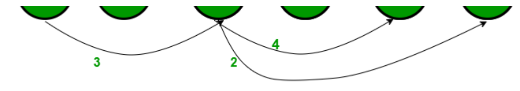
Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

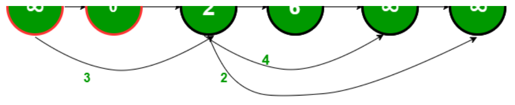
For a general weighted graph, we can calculate single source shortest distances in O(VE) time using Bellman–Ford Algorithm. For a graph with no negative weights, we can do better and calculate single source shortest distances in O(E + VLogV) time using Dijkstra's algorithm. Can we do even better for Directed Acyclic Graph (DAG)? We can calculate single source shortest distances in O(V+E) time for DAGs. The idea is to use Topological Sorting.

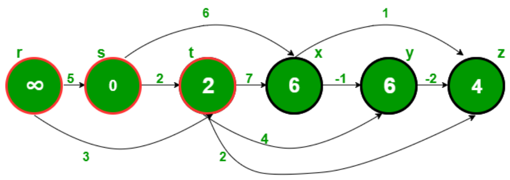
We initialize distances to all vertices as infinite and distance to source as 0, then we find a topological sorting of the graph. Topological Sorting of a graph represents a linear ordering of the graph (See below, figure (b) is a linear representation of figure (a)). Once we have topological order (or linear representation), we one by one process all vertices in topological order. For every vertex being processed, we update distances of its adjacent using distance of current vertex.

Following figure is taken from this source. It shows step by step process of finding shortest paths.

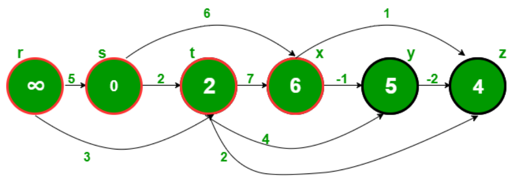



(c)

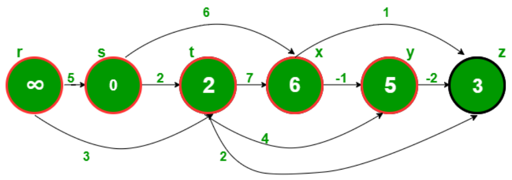

(d)



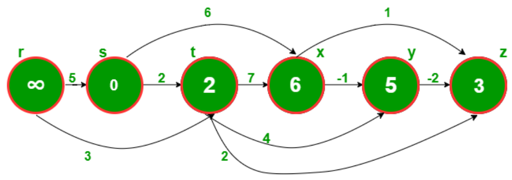
(e)



(f)



(g)



(h)

Following is complete algorithm for finding shortest distances.

1) Initialize dist[] = {INF, INF, ….} and dist[s] = 0 where s is the source vertex.

2) Create a toplogical order of all vertices.

3) Do following for every vertex u in topological order.

………..Do following for every adjacent vertex v of u

………………if (dist[v] > dist[u] + weight(u, v))

………………………dist[v] = dist[u] + weight(u, v)

C++

```
// C++ program to find single source shortest paths for Directed Acyclic Graphs
#include<iostream>
#include <list>
#include <stack>
#include <limits.h>
#define INF INT_MAX
using namespace std;

// Graph is represented using adjacency list. Every node of adjacency list
// contains vertex number of the vertex to which edge connects. It also
// contains weight of the edge
class AdjListNode
{
    int v;
    int weight;
public:
    AdjListNode(int _v, int _w)  { v = _v;  weight = _w;}
    int getV()       {  return v;  }
    int getWeight()  {  return weight; }
};
```

```
    // Pointer to an array containing adjacency lists
    list<AdjListNode> *adj;

    // A function used by shortestPath
    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);
public:
    Graph(int V);   // Constructor

    // function to add an edge to graph
    void addEdge(int u, int v, int weight);

    // Finds shortest paths from given source vertex
    void shortestPath(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<AdjListNode>[V];
}

void Graph::addEdge(int u, int v, int weight)
{
    AdjListNode node(v, weight);
    adj[u].push_back(node); // Add v to u's list
}

// A recursive function used by shortestPath. See below link for details
// https://www.geeksforgeeks.org/topological-sorting/
void Graph::topologicalSortUtil(int v, bool visited[], stack<int> &Stack)
{
    // Mark the current node as visited
    visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
    list<AdjListNode>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
    {
        AdjListNode node = *i;
        if (!visited[node.getV()])
            topologicalSortUtil(node.getV(), visited, Stack);
    }

    // Push current vertex to stack which stores topological sort
    Stack.push(v);
}

// The function to find shortest paths from given vertex. It uses recursive
// topologicalSortUtil() to get topological sorting of given graph.
void Graph::shortestPath(int s)
{
    stack<int> Stack;
    int dist[V];

    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to store Topological Sort
    // starting from all vertices one by one
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    // Initialize distances to all vertices as infinite and distance
    // to source as 0
    for (int i = 0; i < V; i++)
    while (Stack.empty() == false)
    {
        // Get the next vertex from topological order
        int u = Stack.top();
        Stack.pop();

        // Update distances of all adjacent vertices
        list<AdjListNode>::iterator i;
        if (dist[u] != INF)
        {
          for (i = adj[u].begin(); i != adj[u].end(); ++i)
             if (dist[i->getV()] > dist[u] + i->getWeight())
                dist[i->getV()] = dist[u] + i->getWeight();
        }
    }

    // Print the calculated shortest distances
    for (int i = 0; i < V; i++)
        (dist[i] == INF)? cout << "INF ": cout << dist[i] << " ";
}

// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram.  Here vertex numbers are
    // 0, 1, 2, 3, 4, 5 with following mappings:
    // 0=r, 1=s, 2=t, 3=x, 4=y, 5=z
    Graph g(6);
    g.addEdge(0, 1, 5);
    g.addEdge(0, 2, 3);
    g.addEdge(1, 3, 6);
    g.addEdge(1, 2, 2);
    g.addEdge(2, 4, 4);
    g.addEdge(2, 5, 2);
    g.addEdge(2, 3, 7);
    g.addEdge(3, 4, -1);
    g.addEdge(4, 5, -2);

    int s = 1;
    cout << "Following are shortest distances from source " << s <<" n";
    g.shortestPath(s);

    return 0;
}
```

Run on IDE Copy Code

Java

```
// Java program to find single source shortest paths in Directed Acyclic Graphs
import java.io.*;
import java.util.*;

class ShortestPath
{
    static final int INF=Integer.MAX_VALUE;
    class AdjListNode
    {
        private int v;
        private int weight;
        AdjListNode(int _v, int _w) { v = _v;  weight = _w; }
        int getV() { return v; }
        int getWeight()  { return weight; }
    }

    // Class to represent graph as an adjcency list of
    // nodes of type AdjListNode
```

```
        Graph(int v)
        {
            V=v;
            adj = new LinkedList[V];
            for (int i=0; i<v; ++i)
                adj[i] = new LinkedList<AdjListNode>();
        }
        void addEdge(int u, int v, int weight)
        {
            AdjListNode node = new AdjListNode(v,weight);
            adj[u].add(node);// Add v to u's list
        }

        // A recursive function used by shortestPath.
        // See below link for details
        void topologicalSortUtil(int v, Boolean visited[], Stack stack)
        {
            // Mark the current node as visited.
            visited[v] = true;
            Integer i;

            // Recur for all the vertices adjacent to this vertex
            Iterator<AdjListNode> it = adj[v].iterator();
            while (it.hasNext())
            {
                AdjListNode node =it.next();
                if (!visited[node.getV()])
                    topologicalSortUtil(node.getV(), visited, stack);
            }
            // Push current vertex to stack which stores result
            stack.push(new Integer(v));
        }

        // The function to find shortest paths from given vertex. It
        // uses recursive topologicalSortUtil() to get topological
        // sorting of given graph.
        void shortestPath(int s)
        {
            Stack stack = new Stack();
            int dist[] = new int[V];

            // Mark all the vertices as not visited
            Boolean visited[] = new Boolean[V];
            for (int i = 0; i < V; i++)
                visited[i] = false;

            // Call the recursive helper function to store Topological
            // Sort starting from all vertices one by one
            for (int i = 0; i < V; i++)
                if (visited[i] == false)
                    topologicalSortUtil(i, visited, stack);

            // Initialize distances to all vertices as infinite and
            // distance to source as 0
            for (int i = 0; i < V; i++)
                dist[i] = INF;
            dist[s] = 0;

            // Process vertices in topological order
            while (stack.empty() == false)
            {
                // Get the next vertex from topological order
                int u = (int)stack.pop();

                // Update distances of all adjacent vertices
                Iterator<AdjListNode> it;
                if (dist[u] != INF)
                {
                    it = adj[u].iterator();
```

```
                        dist[i.getV()] = dist[u] + i.getWeight();
                }
            }
        }

        // Print the calculated shortest distances
        for (int i = 0; i < V; i++)
        {
            if (dist[i] == INF)
                System.out.print( "INF ");
            else
                System.out.print( dist[i] + " ");
        }
    }
}

// Method to create a new graph instance through an object
// of ShortestPath class.
Graph newGraph(int number)
{
    return new Graph(number);
}

public static void main(String args[])
{
    // Create a graph given in the above diagram.  Here vertex
    // numbers are 0, 1, 2, 3, 4, 5 with following mappings:
    // 0=r, 1=s, 2=t, 3=x, 4=y, 5=z
    ShortestPath t = new ShortestPath();
    Graph g = t.newGraph(6);
    g.addEdge(0, 1, 5);
    g.addEdge(0, 2, 3);
    g.addEdge(1, 3, 6);
    g.addEdge(1, 2, 2);
    g.addEdge(2, 4, 4);
    g.addEdge(2, 5, 2);
    g.addEdge(2, 3, 7);
    g.addEdge(3, 4, -1);
    g.addEdge(4, 5, -2);

    int s = 1;
    System.out.println("Following are shortest distances "+
                        "from source " + s );
    g.shortestPath(s);
}
}
//This code is contributed by Aakash Hasija
```

Run on IDE Copy Code

Python

```
# Python program to find single source shortest paths
# for Directed Acyclic Graphs Complexity :OV(V+E)
from collections import defaultdict

# Graph is represented using adjacency list. Every
# node of adjacency list contains vertex number of
# the vertex to which edge connects. It also contains
# weight of the edge
class Graph:
    def __init__(self,vertices):

        self.V = vertices # No. of vertices

        # dictionary containing adjacency List
        self.graph = defaultdict(list)
    # A recursive function used by shortestPath
    def topologicalSortUtil(self,v,visited,stack):

        # Mark the current node as visited.
        visited[v] = True

        # Recur for all the vertices adjacent to this vertex
        if v in self.graph.keys():
            for node,weight in self.graph[v]:
                if visited[node] == False:
                    self.topologicalSortUtil(node,visited,stack)

        # Push current vertex to stack which stores topological sort
        stack.append(v)


    ''' The function to find shortest paths from given vertex.
        It uses recursive topologicalSortUtil() to get topological
        sorting of given graph.'''
    def shortestPath(self, s):

        # Mark all the vertices as not visited
        visited = [False]*self.V
        stack =[]

        # Call the recursive helper function to store Topological
        # Sort starting from source vertice
        for i in range(self.V):
            if visited[i] == False:
                self.topologicalSortUtil(s,visited,stack)

        # Initialize distances to all vertices as infinite and
        # distance to source as 0
        dist = [float("Inf")] * (self.V)
        dist[s] = 0

        # Process vertices in topological order
        while stack:

            # Get the next vertex from topological order
            i = stack.pop()

            # Update distances of all adjacent vertices
            for node,weight in self.graph[i]:
                if dist[node] > dist[i] + weight:
                    dist[node] = dist[i] + weight

        # Print the calculated shortest distances
        for i in range(self.V):
            print ("%d" %dist[i]) if dist[i] != float("Inf") else  "Inf" ,


g = Graph(6)
g.addEdge(0, 1, 5)
g.addEdge(0, 2, 3)
g.addEdge(1, 3, 6)
g.addEdge(1, 2, 2)
g.addEdge(2, 4, 4)
g.addEdge(2, 5, 2)
g.addEdge(2, 3, 7)
g.addEdge(3, 4, -1)
g.addEdge(4, 5, -2)

# source = 1
s = 1

print ("Following are shortest distances from source %d " % s)
```

Run on IDE Copy Code

Output:

```
Following are shortest distances from source 1
INF 0 2 6 5 3
```

Time Complexity: Time complexity of topological sorting is O(V+E). After finding topological order, the algorithm process all vertices and for every vertex, it runs a loop for all adjacent vertices. Total adjacent vertices in a graph is O(E). So the inner loop runs O(V+E) times. Therefore, overall time complexity of this algorithm is O(V+E).

References:
http://www.utdallas.edu/~sizheng/CS4349.d/l-notes.d/L17.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

- Find a Mother Vertex in a Graph
- Printing Paths in Dijkstra's Shortest Path Algorithm
- Shortest Path in a weighted Graph where weight of an edge is 1 or 2
- Shortest path with exactly k edges in a directed and weighted graph
- Some interesting shortest path questions | Set 1
- Longest Path in a Directed Acyclic Graph
- Strongly Connected Components
- Topological Sorting
- Bellman–Ford Algorithm | DP-23
- Dijkstra’s Algorithm for Adjacency List Representation | Greedy Algo-8
- Dijkstra's shortest path algorithm | Greedy Algo-7
- Prim’s Minimum Spanning Tree (MST) | Greedy Algo-5
- Floyd Warshall Algorithm | DP-16
- Find if there is a path between two vertices in a directed graph
- Detect Cycle in a Directed Graph

Article Tags : Graph Shortest Path Topological Sorting

1

3.5

☐ To-do ☐ Done

Based on **50** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Share this post!

39 Comments **GeeksforGeeks** 1 **Login**

Recommend 4 Tweet Share **Sort by Newest**

Join the discussion…

LOG IN WITH **OR SIGN UP WITH DISQUS** ?

Name

tao yangyang • 5 days ago

From Figure (a) t to y should be -1 and x to y should be 5, though it doesn't affect the correctness of algorithm!

• Reply • Share ›

Jaideep Kaiwart • a month ago

I think simple dfs will work here. We start from source and initialize its dist[s] = 0 and for others dist = INF. Then we take its adjacent nodes one by one and recurse. We update distance of a node if we find a shorter path and then call dfs with this node again.

Please correct me if I am wrong.

1 • Reply • Share ›

Lin Paul • 2 months ago

• Reply • Share ›

da.cr0w • 4 months ago

using array, simple c++
https://ide.geeksforgeeks.o...

• Reply • Share ›

neruwa • 6 months ago

Rather than using links, it would be better to make some inline templates

• Reply • Share ›

Pradeep Poudel • 6 months ago

Video rather make me understand better.

• Reply • Share ›

Rehmanali Momin • 7 months ago

https://ide.geeksforgeeks.o...

A simple solution using Queue. O(V+E)

• Reply • Share ›

Akhand Singh • 10 months ago

In the place of stack.append(v), stack.insert(0,v) would be correct. Append will add the vertices in the last but we have to put it on first place. So my suggestion is use insert instead of append.

• Reply • Share ›

deku • 10 months ago

https://ide.geeksforgeeks.o...
using kahn's algorithm

• Reply • Share ›

pulkit gupta • a year ago

can someone explain why does it work ?
whats the role of topological sorting?

• Reply • Share ›

Duplex • 2 years ago

Similar principal to Q. of "Longest path in a DAG",
Here C++ code: http://ideone.com/D98Fce

1 • Reply • Share ›

flashmozzg • 2 years ago

Bad example. You don't even need topsort for it. you can just go from 0 to V - 1 and it'd still work.

• Reply • Share ›

So which if you don't use topological sort, it still works.

Reply • Share ›

siddharth patel • 2 years ago

I think the inner loop runs for O(E) times not O(V+E),please correct me if am wrong.

Reply • Share ›

Code@Geeks • 2 years ago

Without using classes in C++:
http://ideone.com/Mvry6s

Reply • Share ›

varun9 • 2 years ago

Same approach as mentioned in "Longest Path in a Directed Acyclic Graph".
http://www.geeksforgeeks.or...

1 • Reply • Share ›

geeks13 • 2 years ago

Is "dist[u]!=INF" necessary?

Reply • Share ›

Avishek Dutta geeks13 • 2 years ago

Yes.

Because here INF is defined as INT_MAX. If you add anything to INT_MAX, this operation invokes undefined behavior. Signed integer overflow is undefined behavior in C/C++. It can result to INT_MIN or the implementation can consider
this expression to be positive or the program can crash. Do not let a
portable program compute this expression.

So according to the above diagram, dist[r]=INF, dist[s]=0

Now if you try to relax the egde(r,s) i.e.

if(dist[s] > dist[r]+5)

this makes dist[r]+5 undefined making dist[s]>dist[r]+5 and a garbage value gets stored in dist[s].

Reference: http://stackoverflow.com/a/...

Reply • Share ›

bhavik gujarati geeks13 • 2 years ago

Yes, it is. Otherwise, for the source, the distance will not remain 0 because of previous vertices of topological order and it will create chaos.

1 • Reply • Share ›

Dima Kravtsov • 3 years ago

if (visited[i] == false)
topologicalSortUtil(i, visited, Stack);

Can we do just for a source vertex?
topologicalSortUtil(s, visited, Stack);
• Reply • Share ›

Devi • 3 years ago
1. How Can you visit every node and get a minimum path?
2. How can you get 2 directed paths starting at a node 's' such that every node in G lies on atleast one of those paths, and such that the sum of the weights of all edges in the paths is minimized?
• Reply • Share ›

Kartik Devi • 3 years ago
See problem number 10 here (http://courses.csail.mit.ed...)
• Reply • Share ›

Raffaele Intorcia • 3 years ago
How can i get the node list for the shortest path?
• Reply • Share ›

Billionaire • 3 years ago
Here is my summary:

For single source shortest path problem:

This method(topological sort) only applies for a DAG, (need to be both directed and acyclic). -> O(V+E)
If the graph is undirected, and without negative weight -> Dijkstra's algorithm -> O(ElogV)
If the graph has negative weight -> Bellman-Ford algorithm -> O(VE)
5 • Reply • Share ›

Code@Geeks Billionaire • 2 years ago
Wrong. Dijsktra works for negative weights too, but there should be no negative weight cycle.
• Reply • Share ›

Ratul Banerjee Code@Geeks • 2 years ago
Dijkstra does not work for negative weights even if there is no negative weight cycle.
• Reply • Share ›

saurabh singh Ratul Banerjee • 9 months ago
dijikstra does work on negative weights its the negative weight cycle that messes it up therefore it is not considered wise to use it at all in presence of negative weights
• Reply • Share ›

Zaid Haq Code@Geeks • 2 years ago
Are you sure... I guess Dijkstra doesn't work for negative weight cycle...Cuz it is greedy and once you add to the set, you cant change the distance, but if there is a negative weight

Ajay kumar ➦ Zaid Haq • 2 years ago

I think Dijkstra does not work only in case when the Source vertex is reachable from the vertex with the negative edge, Creating negative cycle.

Correct me if i am wrong.

• Reply • Share ›

Rajnish Kumar • 3 years ago

Why Cant we use BFS ?

2 • Reply • Share ›

Geek ➦ Rajnish Kumar • 3 years ago

I hope u want to say dfs as bfs wont do anything. We cant use dfs Coz if we do, we cant use visited array. That wud make it exponential.

• Reply • Share ›

numid • 3 years ago

Can someone explain why topological sort?The proofs in the references are not easy to understand.

2 • Reply • Share ›

Andy Toh • 3 years ago

Here I revised the above program so that it outputs the shortest paths themselves (for each possible destination) as well. The printout could be made a bit prettier, but I only concentrated on getting the correct nodes in the shortest paths.

http://ideone.com/ZgC7ms

The algorithm actually requires proof for its correctness. There I have also typed out a formal proof of the algorithm's correctness using mathematical induction.

• Reply • Share ›

WP • 4 years ago

The link for topological sorting is broken.

1 • Reply • Share ›

AllergicToBitches • 4 years ago

Implementation without using stacks -
http://ideone.com/GU7r

• Reply • Share ›

Hector • 4 years ago

Hello,
I am not sure what version of c you are using, but normally a list does not have a [] operator (line adj[u]) you probably want to be using a vector or a hash (not really standard) for the adj type.

• Reply • Share ›

Anurag Singh Hector • 4 years ago

adj is not a list here.
adj is an array of list.
It is declared as:
list<adjlistnode> *adj;
adj = new list<adjlistnode>[V];

so adj[0] will have 1st list, adj[1] will have 2nd list and so on.

1 • Reply • Share ›

jinzhi chen • 5 years ago

wow, thanks

1 • Reply • Share ›

Allanian • 5 years ago

Explanation as great as longest in path for DAG. Thank you.