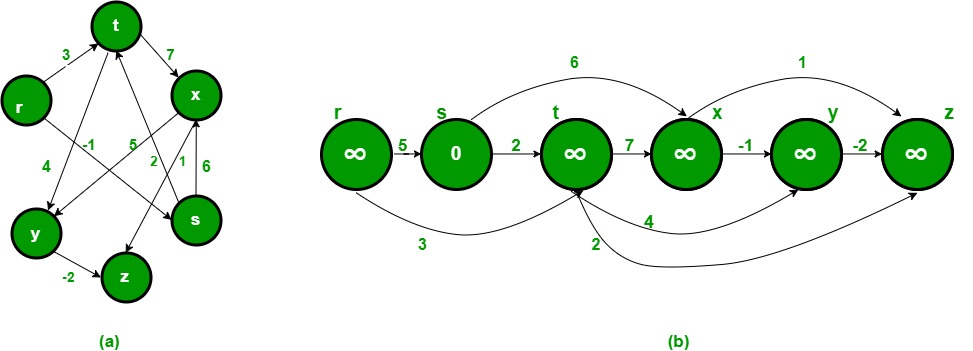
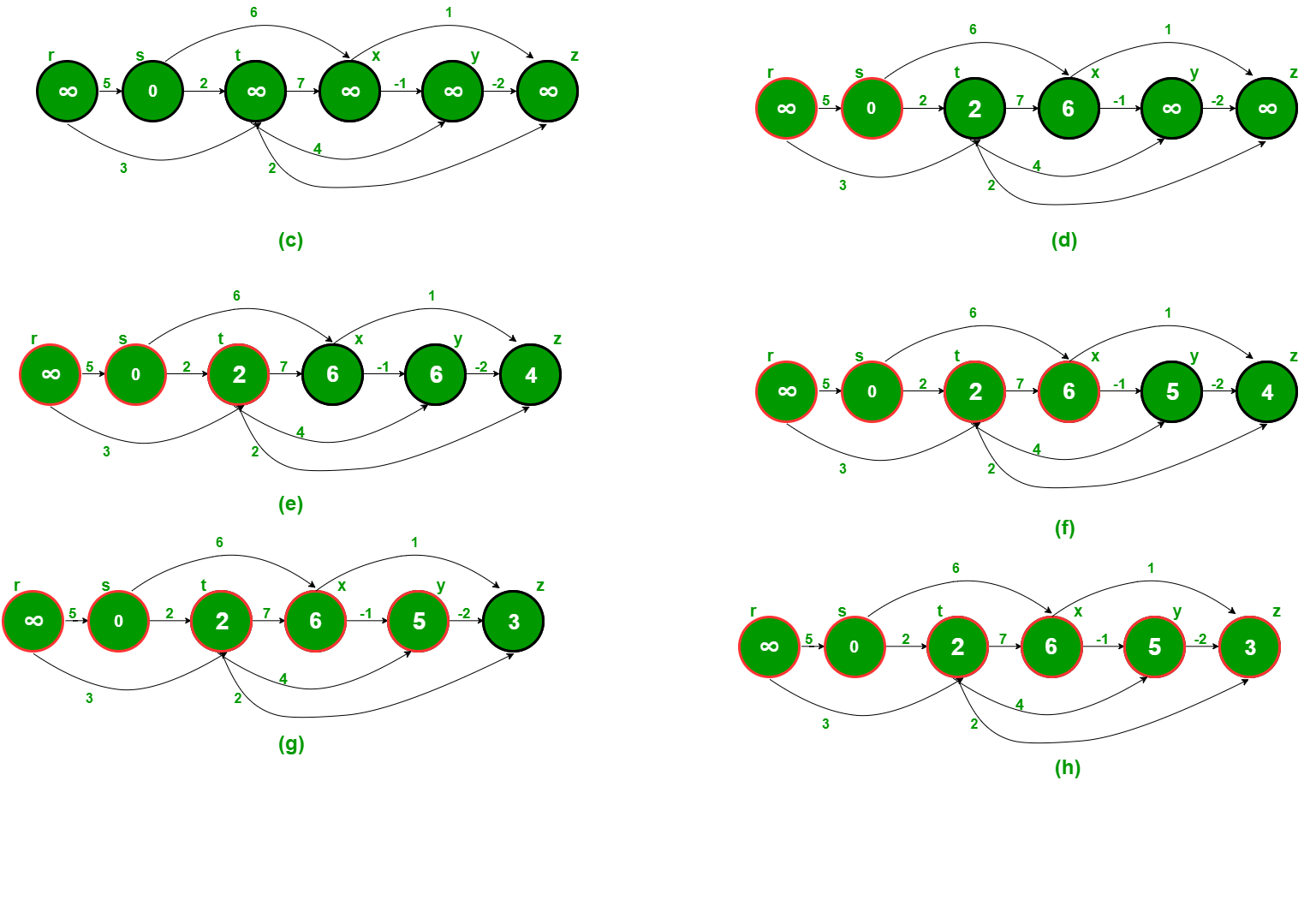
Shortest Path in Directed Acyclic Graph

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.**](https://practice.geeksforgeeks.org/problems/shortest-path-from-1-to-n/0)

For a general weighted graph, we can calculate single source shortest distances in O(VE) time using [Bellman–Ford Algorithm](https://www.geeksforgeeks.org/dynamic-programming-set-23-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](https://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/). 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](https://www.geeksforgeeks.org/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](https://www.geeksforgeeks.org/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](http://www.utdallas.edu/~sizheng/CS4349.d/l-notes.d/L17.pdf)source. It shows step by step process of finding shortest paths.  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/shortestpathstart.jpg)  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/shortestpathsteps.png)

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++ 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; }

};

// Class to represent a graph using adjacency list representation

class Graph

{

int V; // No. of vertices'

// 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++)

dist[i] = INF;

dist[s] = 0;

// Process vertices in topological order

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;

}