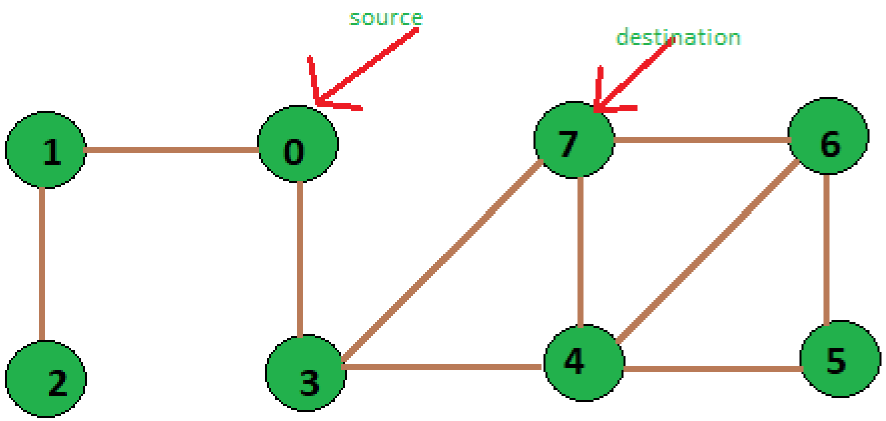
Dash
All
Articles
Videos
Problems
Quiz
Contest
« Prev
Next »

Shortest path in an Unweighted graph

Given an unweighted graph, a source, and a destination, we need to find the shortest path from source to destination in the graph in the most optimal way.



unweighted graph of 8 vertices

```
Input: source vertex = 0 and destination vertex is = 7.
Output: Shortest path length is:2
        Path is::
        0 3 7

Input: source vertex is = 2 and destination vertex is = 6.
Output: Shortest path length is:5
```

```
Path is::
2 1 0 3 4 6
```

Dash
All
Articles
Videos
Problems
Quiz
Contest
«
»

One solution is to solve in O(VE) time using Bellman–Ford. If there are no negative weight cycles, then we can solve in O(E + VLogV) time using Dijkstra's algorithm.
Since the graph is unweighted, we can solve this problem in O(V + E) time. The idea is to use a modified version of Breadth-first search in which we keep storing the predecessor of a given vertex while doing the breadth-first search.

We first initialize an array dist[0, 1, …., v-1] such that dist[i] stores the distance of vertex i from the source vertex and array pred[0, 1, ….., v-1] such that pred[i] represents the immediate predecessor of the vertex i in the breadth-first search starting from the source.

Now we get the length of the path from source to any other vertex in O(1) time from array d, and for printing the path from source to any vertex we can use array p and that will take O(V) time in worst case as V is the size of array P. So most of the time of the algorithm is spent in doing the Breadth-first search from a given source which we know takes O(V+E) time. Thus the time complexity of our algorithm is O(V+E).

Take the following unweighted graph as an example:

Following is the complete algorithm for finding the shortest path:

Implementation:

C++ Java

```java
// Java program to find shortest path in an undirected
// graph
import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;

public class pathUnweighted {
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
// Driver Program
public static void main(String args[])
{
    // No of vertices
    int v = 8;

    // Adjacency list for storing which vertices are connected
    ArrayList<ArrayList<Integer>> adj =
                        new ArrayList<ArrayList<Integer>>(v);
    for (int i = 0; i < v; i++) {
        adj.add(new ArrayList<Integer>());
    }

    // Creating graph given in the above diagram.
    // add_edge function takes adjacency list, source
    // and destination vertex as argument and forms
    // an edge between them.
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 3);
    addEdge(adj, 1, 2);
    addEdge(adj, 3, 4);
    addEdge(adj, 3, 7);
    addEdge(adj, 4, 5);
    addEdge(adj, 4, 6);
    addEdge(adj, 4, 7);
    addEdge(adj, 5, 6);
    addEdge(adj, 6, 7);
    int source = 0, dest = 7;
    printShortestDistance(adj, source, dest, v);
}

// function to form edge between two vertices
// source and dest
private static void addEdge(ArrayList<ArrayList<Integer>> adj, int i, int j)
{
    adj.get(i).add(j);
    adj.get(j).add(i);
}

// function to print the shortest distance and path
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
// between source vertex and destination vertex
private static void printShortestDistance(
                 ArrayList<ArrayList<Integer>> adj,
                         int s, int dest, int v)
{
    // predecessor[i] array stores predecessor of
    // i and distance array stores distance of i
    // from s
    int pred[] = new int[v];
    int dist[] = new int[v];

    if (BFS(adj, s, dest, v, pred, dist) == false) {
        System.out.println("Given source and destination" +
                                     "are not connected");
        return;
    }

    // LinkedList to store path
    LinkedList<Integer> path = new LinkedList<Integer>();
    int crawl = dest;
    path.add(crawl);
    while (pred[crawl] != -1) {
        path.add(pred[crawl]);
        crawl = pred[crawl];
    }

    // Print distance
    System.out.println("Shortest path length is: " + dist[dest]);

    // Print path
    System.out.println("Path is ::");
    for (int i = path.size() - 1; i >= 0; i--) {
        System.out.print(path.get(i) + " ");
    }
}

// a modified version of BFS that stores predecessor
// of each vertex in array pred
// and its distance from source in array dist
private static boolean BFS(ArrayList<ArrayList<Integer>> adj, int src,
                              int dest, int v, int pred[], int dist[])
```

```
{
    // a queue to maintain queue of vertices whose
    // adjacency list is to be scanned as per normal
    // BFS algorithm using LinkedList of Integer type
    LinkedList<Integer> queue = new LinkedList<Integer>();

    // boolean array visited[] which stores the
    // information whether ith vertex is reached
    // at least once in the Breadth first search
    boolean visited[] = new boolean[v];

    // dist[i] for all i set to infinity
    for (int i = 0; i < v; i++) {
        visited[i] = false;
        dist[i] = Integer.MAX_VALUE;
        pred[i] = -1;
    }

    // now source is first to be visited and
    // distance from source to itself should be 0
    visited[src] = true;
    dist[src] = 0;
    queue.add(src);

    // bfs Algorithm
    while (!queue.isEmpty()) {
        int u = queue.remove();
        for (int i = 0; i < adj.get(u).size(); i++) {
            if (visited[adj.get(u).get(i)] == false) {
                visited[adj.get(u).get(i)] = true;
                dist[adj.get(u).get(i)] = dist[u] + 1;
                pred[adj.get(u).get(i)] = u;
                queue.add(adj.get(u).get(i));

                // stopping condition (when we find
                // our destination)
                if (adj.get(u).get(i) == dest)
                    return true;
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
                }
            }
        }
        return false;
    }
}
```

Output

```
Shortest path length is : 2
Path is::
0 3 7
```

Time Complexity : O(V + E)
Auxiliary Space: O(V)

Mark as Read

Report An Issue
If you are facing any issue on this page. Please let us know.