**Traversals of Graphs**

In tree we have 🡪 preorder, inorder, postorder, levelorder (heap) traversals

Most graph algorithms involve visiting each vertex in a systematic order

As with trees, there are different ways to do this

The two most common traversal algorithms are the breadth-first search (cautious) and the depth-first search (brave)

**Breadth-First Search (BFS)**

In bfs,

* visit the start node first
* then all nodes that are adjacent to it
* then all nodes that can be reached by a path from the start node containing two edges
* then all nodes that can be reached by a path from the start node containing three edges
* …and so on…

If we have visited a vertex, we don’t visit it again.

We must visit all nodes for which the shorthest path from the start node is length k before we visit any node for which the shortest path from the start node is length k+1

There is no special start vertex -- we arbitrarily choose the vertex with label 0

Chart

Description automatically generated

Chart, diagram

Description automatically generated with medium confidence

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with low confidence

When queue is empty, we have visited all the vertices IF THE GRAPH IS CONNECTED.

Here we can only visit vertices which are connected to 0. If there is a separate part, we cannot visit its vertices.

If graph is connected for adjacency list ----> |E| >= |V| - 1

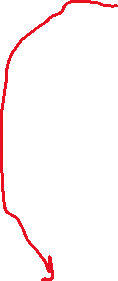
Algorithm

Text

Description automatically generated



*We assumed graph is connected in our analysis.*



while loop is executed |V| times (if graph is connected) 🡪 (|V|)

Step 3 runs (|V|) times.

In step 4, we can use iterator:

* We have to create iterator n times
* We use that iterator and use next()
* For some vertices, there could be 1 element or there could be n-1 elements
* We have to iterate different times for each u 🡪 |E| times in total



So running time of step 4 depends on whether graph is implemented as adjacency matrix or adjacency list.

* If it is adjacency list, it is gonna take (|E|) time
* If it is adjacency matrix, it is gonna take (|V|2) time

So the inner loop at step 4 is performed for ||, the number of edges that originate at that vertex

if statement (step 5) runs |E| times in total if it is directed (if graph is undirected, 2|E|). 🡪 (|E|)

Step 6 runs |V| times since if condition is true for only |V| times. Each vertex is changed its situation (unvisited-visited-identified) once 🡪 (|V|)

Step 7 runs |V| times. 🡪 (|V|)

Step 8 runs |V| times. 🡪 (|V|)

OVERALL RUNNING TIME FOR ADJACENCY LIST ---------> (|E| + |V|) 🡪 (|E|)

🡪 Since |E| >= |V| - 1 , it is better to not to write |V| in theta notation

Remember graph is connected so we can say |E| >= |V| - 1

OVERALL RUNNING TIME FOR ADJACENCY MATRIX ----> (|E| + |V|2) 🡪 (|V|2)

🡪 Since |E| is |V|2 at most, it is better to not to write in theta notation

Remember: |E| <= |V|2

Traversing non-connected graph: You should run the algorithm more than once to traverse all vertices.

A picture containing watch, scissors

Description automatically generatedWe can also build a tree that represents the order in which vertices will be visited in a bfs

The tree has all the vertices and some of the edges of the original graph (edge number is |V| - 1)

A path starting at the root to any vertex in the tree is the shortest (in terms of # of edges) path in the original graph to that vertex (considering all edges to have the same weight)

We can save the information we need to represent the tree by storing the parent of each vertex when we identify it

We can have an array that each index (representing vertex) has its parent

We can refine step 7 of the algorithm to accomplish this:

* 7.1 Insert vertex v into the queue
* 7.2 Set the parent of v to u

**Implementing BFS**

Text, letter

Description automatically generated

Diagram

Description automatically generated with medium confidenceThe method returns array parent which can be used to construct the breadth-first search tree

If we run the search on the graph we just traversed, parent will be filled with the values shown on the right

**Depth-First Search (DFS)**

In a dfs,

* start at a vertex
* visit it
* choose one adjacent vertex to visit
* then, choose a vertex adjacent to that vertex to visit
* and so on until you go no further (no adjacent vertex that is not visited)
* then back up and see whether a new vertex can be found

We put being visited vertices in a STACK.

Chart

Description automatically generated with medium confidence

We cannot stop visiting vertex 0 until all the vertices that can be reached from 0 are visited.

Chart

Description automatically generated with low confidence

Chart

Description automatically generated with medium confidence

Chart

Description automatically generated with low confidence

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

Chart

Description automatically generated

**Search Terms**

The discovery order is the order in which the vertices are discovered: 0, 1, 3, 4, 2, 5, 6

The finish order is the order in which the vertices are finished: 4, 3, 1, 6, 5, 2, 0

Back edges (dashed lines) connect a vertex with its ancestors in a dfs tree

Diagram

Description automatically generated

**Algorithm**

Text

Description automatically generated

🡪 for tree

Recursive call results in this loop being applied to each vertex. So if you ignore line 5 for now, we are doing lines 1, 2, 3, 4, 6 number of vertices (|V|) times.

So line 1 and line 6 are constant time and they are running |V| times 🡪

Lines 3 and 4 are constant time bc we keep “visited” array to check a vertex if it is visited or not

for loop:

* takes (|E|) time for adjacency list
* takes (|V|2) time for adjacency matrix

OVERALL RUNNING TIME FOR ADJACENCY LIST ---------> (|E| + |V|) 🡪 (|E|)

🡪 Since |E| >= |V| - 1 , it is better to not to write |V| in theta notation

Remember graph is connected so we can say |E| >= |V| - 1

OVERALL RUNNING TIME FOR ADJACENCY MATRIX ----> (|V|2)

2 arrays we use:

A picture containing chart

Description automatically generated

**Implementation**

class DepthFirstSearch performs a dfs on a graph and records the:

* start time 🡪 index (int)
* finish time 🡪 index (int)
* start order 🡪 int[ ]
* finish order 🡪 int[ ]

For an unconnected graph or for a directed graph, a dfs may not visit each vertex in the graph

Thus, once the recursive method returns, all vertices need to be examined to see if they have been visited -- if not the process repeats on the next unvisited vertex

Thus, a dfs may generate more than one tree

A collection of unconnected trees is called a forest

Text, table

Description automatically generated

In data field we have to have one more thing 🡪 int[ ] order

Text

Description automatically generated



If an order is specified, it is used these 2 places.

Text, letter

Description automatically generated