Contents

1 Breadth-First Search (BFS) 2
   1.1 The breadth-first search algorithm ........................................ 2
   1.2 Time complexity of breadth-first search ............................... 3

2 Depth-First Search (DFS) 3
   2.1 The depth-first search algorithm ........................................ 3
   2.2 Classification of edges ..................................................... 4
   2.3 Time complexity of depth-first search ................................ 4
1  Breadth-First Search (BFS)

1.1  The breadth-first search algorithm

The goal of breadth-first search is to explore a graph. The technique used is to start at an arbitrary vertex and to visit its neighbors. The neighbor’s neighbors are then examined in turn, and so on until all vertices have been visited.

In breadth-first search, we color vertices we have not yet visited white, vertices we have visited black, and vertices we are in the process of visiting gray. As well, we keep track of the in-process (gray) vertices in a queue. Thus the algorithm must ensure that any vertex colored gray is also on the queue and vice versa.

Below is the algorithm of BFS of a graph $G$ starting at the vertex $s \in V[G]$:

function BFS($G, s$)
1:   for all $u \in V[G] - \{s\}$ do
2:      $color[u] \leftarrow$ WHITE
3:      $d[u] \leftarrow \infty$
4:      $\pi[u] \leftarrow$ NIL
5:   end for
6:   $color[s] \leftarrow$ GRAY
7:   $d[s] \leftarrow 0$
8:   $\pi[s] \leftarrow$ NIL
9:   $Q \leftarrow \{s\}$
10:  while $Q \neq \emptyset$ do
11:    $u \leftarrow$ DEQUEUE ($Q$)
12:    for all $v \in Adj[u]$ do
13:       if $color[v] =$ WHITE then
14:          $color[v] \leftarrow$ GRAY
15:          $d[v] \leftarrow d[u] + 1$
16:          $\pi[v] \leftarrow u$
17:    end if
18:    end for
19:    $enqueue (Q, v)$
20:   end while

This algorithm maintains two additional variables associated with each vertex. One of these is a time stamp, $d$, that is incremented each time a vertex is visited. The other is $\pi$, which identifies the predecessor to the vertex—that is, the vertex from which the current vertex was found. These variables are useful in some of the applications of breadth-first search.

The edges are of the following types:

- Black vertex $\rightarrow$ black vertex  Completely explored territory
- Black vertex $\rightarrow$ gray vertex  Haven’t yet finished with latter vertex
- Gray vertex $\rightarrow$ gray vertex  Haven’t yet finished with either vertex
- Gray vertex $\rightarrow$ white vertex  Haven’t yet finished with former, haven’t seen latter
- White vertex $\rightarrow$ white vertex  Haven’t seen either vertex

\[1\] If the graph is not connected (that is, if the vertices of the graph can be divided into two partitions with no edges between vertices in one partition and vertices in the other), it is actually quite relevant which vertex the algorithm begins with, as only one partition of the graph will be reached by breadth-first search.
1.2 Time complexity of breadth-first search

What is the time complexity of BFS? Each vertex is added to and removed from the queue at most once, when it is colored gray and when it is colored black, respectively. These queue operations take constant time, so the total work involved is \(O(|V|)\).

Each edge is examined no more than twice, once for each vertex. Again, this involves constant time for each vertex, \(O(|E|)\) in total.

Thus BFS requires \(O(|V| + |E|)\) time.

2 Depth-First Search (DFS)

As we have seen, breadth-first search offers us a method to visit the vertices of a graph. In particular, given a starting vertex \(s\), BFS first visits \(s\), then visits all vertices adjacent to \(s\), then visits all vertices adjacent to those vertices, and so on. An alternative approach, and that used by depth-first search, is, intuitively, to “plunge in” — as each node is visited, visit its children before continuing the search at the same depth, and only back out when no unvisited children remain.

2.1 The depth-first search algorithm

If we modify the breadth-first search algorithm by changing the queue \(Q\) to a stack, we have derived depth-first search. Alternatively, we may use recursion to implement the stack; this is the approach typically taken. One small but useful change in the algorithm is a different treatment of time stamps: in DFS, each vertex has two associated time stamps, one, \(d\), holding the discovery time, and the other, \(f\), holding the completion time. The time is incremented at each \(d\) or \(f\) assignment. Here is the recursive version:

```plaintext
function DFS(G)
1: for all \(u \in V[G]\) do
2: \(color[u] \leftarrow\) WHITE
3: \(\pi[u] \leftarrow\) NIL
4: end for
5: \(time \leftarrow 0\)
6: for all \(u \in V[G]\) do
7: if \(color[u] =\) WHITE then
8: DFS-visit\((u)\)
9: end if
10: end for

function DFS-visit\((u)\)
1: \(color[u] \leftarrow\) GRAY
2: \(d[u] \leftarrow time \leftarrow time + 1\)
3: for all \(v \in Adj[u]\) do
4: if \(color[v] =\) WHITE then
5: \(\pi[v] \leftarrow u\)
6: DFS-visit\((v)\)
7: end if
8: end for
9: \(color[u] \leftarrow\) BLACK
10: \(f[u] \leftarrow time \leftarrow time + 1\)
```
2.2 Classification of edges

We can classify the edges in the graph $G$ based on when the DFS algorithm traverses them:

A **tree edge** is an edge from a gray vertex to a white vertex. A tree edge brings the algorithm into deeper territory, as of yet undiscovered.

A **back edge** is an edge from a gray vertex to another gray vertex. Back edges form cycles in the graph, as they indicate that the algorithm has discovered a vertex further back in the path it is exploring.

Edges from gray vertices to black vertices fall into two classes. A **forward edge** connects the current vertex to a vertex in the subtree rooted at the current vertex (its “descendant”). A **cross edge** connects the current vertex to a vertex in another subtree (for example, it’s “cousin”, “uncle” or “nephew”).

2.3 Time complexity of depth-first search

Depth-first search examines each vertex exactly once. However, it must consider each edge to determine if it leads to an undiscovered (white) vertex. This leads to a running time of $\Theta(|V| + |E|)$. 