

Breadth-first search

Mikael Tylmad

March 24, 2008

1 Introduction

When learning about graph algorithms, the breadth-first search algorithm is a good starting point. It is one of the simplest algorithms for searching a graph and its ideas are used in many other important algorithms. The breadth-first algorithm works by systematically exploring the edges of a graph to find every vertex that is reachable from a specific source vertex. The result is a “breadth-first tree” that has the source vertex as root and sprouts out to all reachable vertices. This means that the shortest path (fewest number of edges) to all reachable vertices from the source is calculated and can be seen in the breadth-first tree.

The name, “breadth-first search”, comes from the fact that the algorithm works by first investigating vertices at distance k from the source vertex, before looking into any vertices at distance $k+1$. This means that the graph is “discovered” uniformly across the breadth of the frontier (the undiscovered vertices).

2 The algorithm

To ensure that the search throughout the graph proceeds in a breadth-first manner, all vertices are given a color. The algorithm starts with coloring all vertices white, which means “an undiscovered vertex”. It continues by coloring each encountered vertex gray and setting the distance (from the source) to these vertices. When a vertex is completely checked, it is colored black.

The algorithm explained in pseudo code:

```
BREADTH-FIRST SEARCH(Graph  $G$ , Vertex  $source$ )
(1)  foreach Vertex  $v$  in  $G$ 
(2)     $v.color \leftarrow WHITE$ 
(3)     $distToSource[v] \leftarrow \infty$ 
(4)     $predecessor[v] \leftarrow NULL$ 
(5)   $source.color \leftarrow GRAY$ 
(6)   $distToSource[source] \leftarrow 0$ 
(7)  FIFOSTORAGE.ADD( $source$ )
(8)  while FIFOSTORAGE.NOTEMPTY()
(9)    Vertex  $v \leftarrow$  FIFOSTORAGE.RETRIEVE()
(10) foreach Vertex  $a$  adjacent to  $v$ 
(11)   if  $a.color = WHITE$ 
(12)      $a.color \leftarrow GRAY$ 
(13)      $distToSource[a] \leftarrow distToSource[v] + 1$ 
(14)      $predecessor[a] \leftarrow v$ 
(15)     FIFOSTORAGE.ADD( $a$ )
(16)    $v.color \leftarrow BLACK$ 
```

Rows 1 through 7 are an initialization leading to the main loop. This loop processes the FiFo-type storage and checks discovered vertices. Newly discovered vertices are subsequently added to the FiFo storage. A graphical example of how breadth-first search traverses a graph is given in figure 1.

3 Beyond breadth-first

An important observation to make is that if the FiFo-type storage in the algorithm would be changed to a LiFo-type storage, this algorithm would become a “Depth-first search”. The ideas expressed in the algorithm can even apply in more advanced algorithms, such as in Prim’s algorithm. Changing the FiFo-type storage to a HpFo-type storage (highest priority, first out) would make it possible to examine a graph with weighted edges and create a minimum spanning tree.

4 Complexity analysis

When performing a breadth-first search of a graph $G(V,E)$ all the vertices, V , are checked at most once. This gives a running time of $\Theta(V)$. When each vertex is checked, the adjacency list of that vertex is checked once. This gives $\Theta(E)$ running time spent checking adjacency lists. The total running time of breadth-first search is therefore $\Theta(V + E)$, which means that the algorithm runs in time linear to the size of the graph.

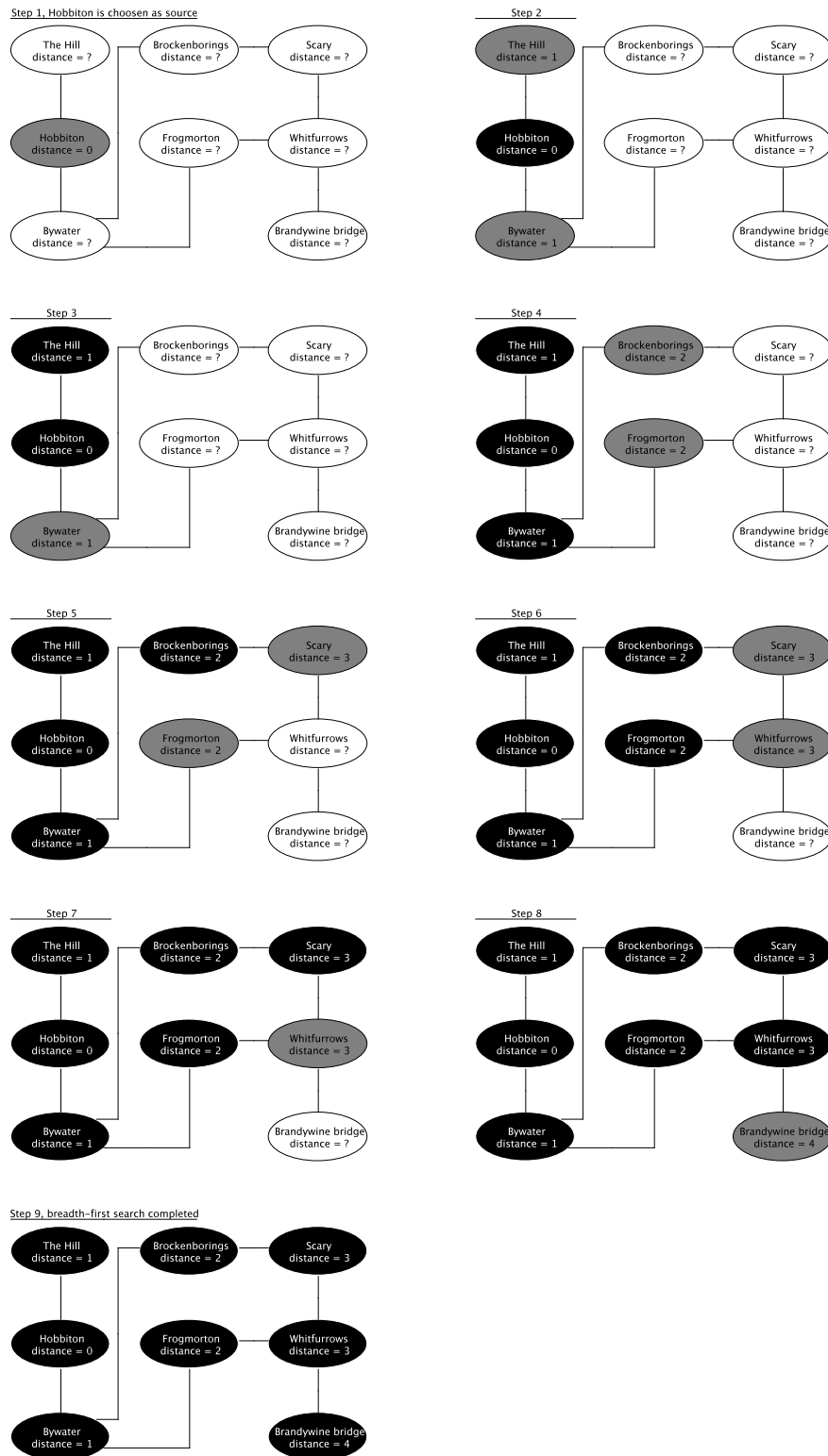


Figure 1: Breadth-first search, step-by-step example