Theory: Breadth-first search

27 minutes 9 / 9 problems solved

Start practicing

When working with graphs, sometimes we need to know how far some two nodes are from each other. For example, recall our graph with edges representing roads in a city and nodes corresponding to the roads' intersections. If we need to get from one point of the city to another, we probably want to go for the shortest route. Another example is a graph with each node representing an airport and edges corresponding to flights between two airports. If we have to get from one airport to another and there is no direct flight between them, then we'd prefer to use the route with fewer transfers.

In terms of graphs, the described problems can be formulated as follows: given two nodes of a graph named *source* and *target*, find a path between them consisting of the smallest possible number of intermediate nodes.

One of the algorithms that allows us to solve this problem is the **breadth-first search**. It is a simple and efficient algorithm that can be used either as a standalone procedure or a subroutine for other graph processing algorithms. In this topic, we will learn the procedure in more details and see how it works on an example.

Current topic:

✓ Breadth-first search

Topic depends on:

✓ Representation of graphs

✓ Queue

Topic is required for:

✓ Dijkstra's algorithm

✓ Lee's algorithm

§1. Breadth-first search procedure

The breadth-first search (BFS) traversal begins with a single node of a graph. Then, the algorithm visits all its neighbors. After that, it considers all the neighbors of the nodes visited at the previous step and so on until all the nodes are processed. In other words, BFS gradually visits all nodes of a graph in the order of increasing distance from the start node.

The algorithm goes like this:

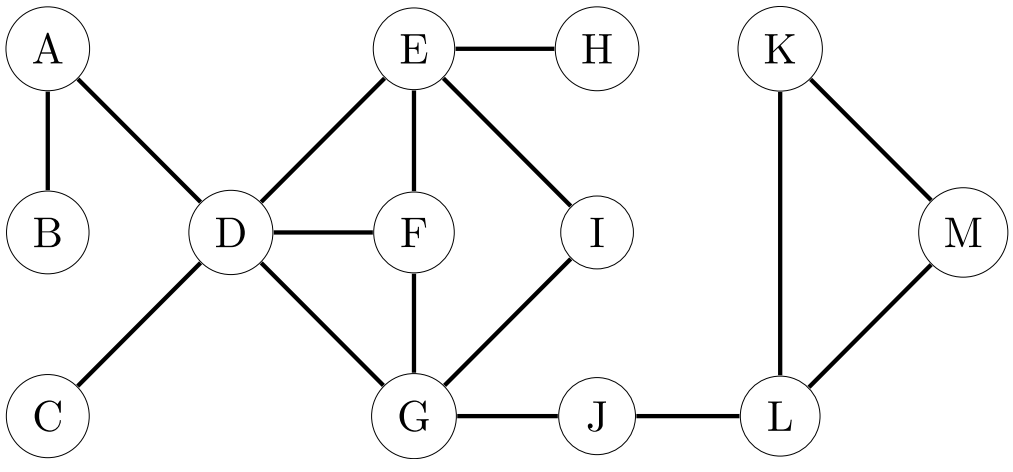
1. Choose an initial node of a graph, mark it as visited and set its distance to 0.
2. Consider all unvisited neighbors of the nodes visited at the previous step and mark all of them as visited. Set the distance to each of these nodes to $d + 1$, where d is the distance to the nodes processed at the previous step.
3. Repeat step 2 until all the nodes are visited.

After the procedure is completed, the distance to each node v will correspond to the shortest distance between the initial node and v.

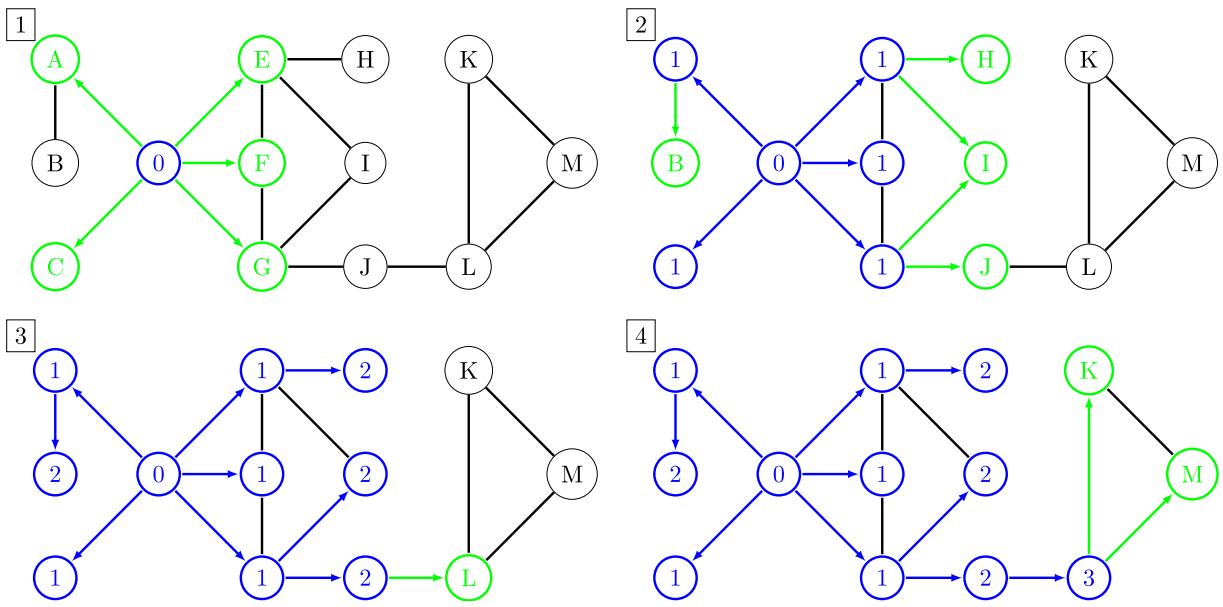
To implement BFS using a particular programming language, you will need to use the **queue** data structure. The idea is to put a starting node to the queue, then at each step extract the current node from it, consider all the neighbors and put them to the queue, and so on. This allows us to traverse the nodes of a graph exactly in the order described above.

§2. An example

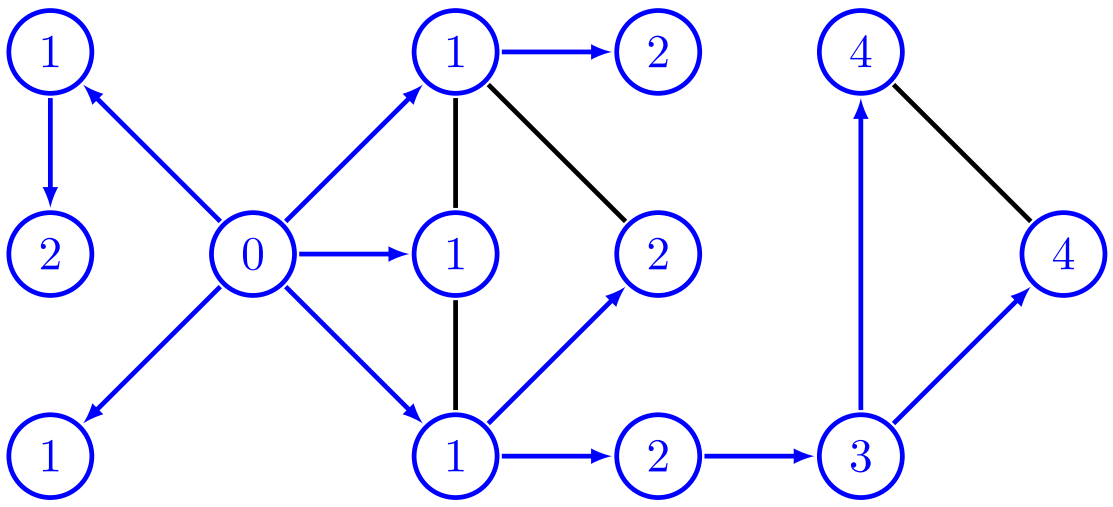
Let's apply the algorithm for the following undirected graph using D as a starting node:



The figures below illustrate the procedure step by step. The visited nodes are shown in blue, and the unvisited neighbors of the visited nodes are green. The remaining nodes are shown in black. The label of each visited node v corresponds to the shortest distance between the initial node D and v.



We begin the procedure with the node D setting its label to 0. Then, we consider all its unvisited neighbors. There are five of them shown in green in the first figure. We set the distances to each of the nodes to 1 and mark them as visited. At the second step, we consider all the unvisited neighbors of the nodes labeled 1. There are four of them shown in green in the second figure. We set the distance to each of the nodes to 2 and move to the next unvisited neighbors. After all the nodes are processed, we get the following graph:



The label of each node v of that graph corresponds to the shortest distance between the starting node and v. The starting node itself has the label 0. As you can see, the blue nodes and edges of the graph form a tree. Such a tree is called a **spanning tree** since it connects (spans) all the nodes of the graph. So, finding a spanning tree is one of the applications of the BFS procedure.

Although the graph above is undirected, BFS works the same way for directed graphs.

If you want to see another example of how the algorithm works, you may take a look at a visualization of the process.

§3. Complexity analysis

While applying the BFS procedure for a graph $G = (V, E)$, each node is added to a queue and extracted from it exactly once. Each edge $\{x, y\}$ is considered exactly twice: while visiting x and then while visiting y (or vice versa). So, the overall running time of the algorithm is $2 \cdot |V| + 2 \cdot |E| = O(|V| + |E|)$.

Report a typo

26 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (1) Hints (0) Useful links (1) Show discussion