Dfs vs bfs when to use

ALGODAILY In simple terms: WIDTH The First Search (BFS) algorithm, called Width, finds all the node neighbors through the outer edges of the knot, then it finds the unused neighbors of the aforementioned neighbors through its outer edges and so on until all the knots reached from the origional source are visited (we can continue and take another origional source if there are any unused knots left and so on). Therefore, it can be used to find the shortest path (if any) from a node (origional source) to another node if the edge weights are uniform. The Depth First Search (DFS) algorithm, called Depth, finds the last found node x untapped neighbors through its outer edge. If there is no unused neighbor x, the algorithm backs down to find the unused neighbors of the node (through its outer edges) where the node x was found, and so on until all the knots reached from the origional source are visited (we can continue and take another origional source if there are unused nodes left and so on). Both BFS and DFS can be incomplete. For example, if the node branching factor is infinite or very high to support resources (memory) (for example, when the next nodes found are saved), the BFS is not complete, although the key you are searching for may be a few edges away from the origional source. This infinite branching can be caused by infinite selections (nodes next to each other) to find. If the depth is infinite or very high to support resources (memory) (for example, when the next nodes found are saved), DFS is not perfect, although the key sought may be the third neighbor of the origional source. This infinite depth can be caused by a situation where each node found by the algorithm has at least a new selection (an adjacent node) that has not previously been visited. Therefore, we can determine when BFS and DFS are used. For example, it is assumed to be a manageable limited branching factor and manageable limited depth. If the desired knot is low, i.e. reachable after some edges of the origional source, it is better to use the BFS. On the other hand, if the knot sought is deep, that is, reachable after many periphery from the origional source, it is better to use DFS. For example, on the social network, if we want to look for people with similar benefits from a particular person, we can apply the BFS from this person as an origional source, since mostly these people are their direct friends or friends of friends, that is, one or two edges far away. On the other hand, if we want to look for people with completely different interests in a particular person, we can apply DFS from this person as an origional source, because mostly these people are very far from him. friend of a friend.... That's too many edges to go far. Applications of BFS and DFS can also vary due to each search mechanism. For example, we can use either BFS (assumed that the branching factor is manageable) or DFS (assumed that the depth is manageable) when we just want to check accessibility from one node to another without any knowledge of where the node might be. Both can also solve the same tasks, such as topological sorting (if it is) in a chart. The BFS can be used to find the shortest path with unit weight edges, from one node (origional source) to another. DFS can be used to exhaust all selections because of its profound nature, such as finding the longest path between two nodes between an acyclic chart. DFS can also be used to detect cycle in the diagram. Eventually, if we have infinite depth and infinite branching, we can use iterative in-depth search (IDS). As a software engineer, there are moments when it is important to know certain algorithms, such as tree pass-through algorithms. The goal of this article is to view two main search algorithms for connected charts: depth-first search (DFS) and width-first search (BFS), both of which can be used to improve program efficiency. Depth-First Search Depth First Search (DFS) is a search algorithm that passes through nodes in a chart. It works by repeatedly expanding each node it locates (from the node to the alis nodes). When there are no more pedal nodes, it returns to the previous node and repeats the process with each adjacent node. It is important to note that if it finds the node it is looking for before passing through all the nodes, the search will finish. Depth First searches are used to determine whether one solution meets certain requirements among many; An example of a depth-first search could be the determination of the route a knight's chess song must use to pass all 64 boxes on the board. The following is a truncated chart with eight nodes. Orange arrows indicate the node path of the DFS algorithm. The following is the implementation of the Depth First search in C++, where the algorithm passes through the nodes in the diagram using an iterator that starts at the first node and continues until the last given node. Two variables visitedNode and checkedNode are used to organize the Depth First path. For this reason, a recursion passes through the nodes. void DFS(grafo &amp;g,int s){ visitedNode[s]=true; checkNode(s); list&lt;int&gt;::iterator it; for(it=g.sides[s].begin(); it!=g.sides[s].end(); ++it){ if(!visitedNode[*it]){ parentNode[*it]=s; DFS(g,*it); }else if(!checkedNode[*it]){ checkSide(s,*it); } } checkedNode[s]=true; } In addition, 2 checkNode and checkSide functions are called to monitor the path of the node, just like &lt;/int&gt;in the following illustration: In the figure above, the first two digits (8.8) represent the number of nodes and pages, followed by 8 pairs of numbers representing the relationships between the nodes. Finally, the path to the DFS algorithm is displayed from node 1 to node 8. DFS Applied Depth-first search algorithms have several applications, like: Finding connected nodes in a chart Topological organization of a directed acyclic chart Finding bridges between charts Solve puzzles with a single solution, such as labyrinths Finding heavily connected nodes A directed acyclic chart (a set of nodes where each node has one direction that is not in itself) can be arranged topologically using the DFS algorithm. This allows you to organize activities that are some kind of interdependent and aim to implement the list as efficiently as possible. Chart bridges are 2 knots that are connected so that it is necessary for one of these nodes to reach their ends. In other words, if you remove one of the nodes, you cannot use another node because it has been completely truncaged. This allows you to prioritize activities represented by nodes. If you want to solve a puzzle or labyrinth, this can be done with DFS; the next steps in the solution can be presented with nodes where each node depends on the previous node. In short, each stage of the puzzle solution depends on the previous step. In some cases, it is important to know how certain activities or components are interconnected in order to reduce activities or components. The aim is to organise the activity or component group in the best possible way to make the system easier to understand; This can also be done using the DFS algorithm. Width-first search The first search for width (BFS) is an algorithm that passes through the nodes in the diagram. It starts with a root node (one of the nodes in the chart is selected as the root) and then examines everyone for its adjacent nodes. The following example examines each adjacent adjacent node until the entire chart is exceeded. It is important to note that if the algorithm finds the node it is looking for before passing through all the nodes, the search will finish. For algorithms, the first search for width is used, for which it is important to always choose the best possible path. The following is a detached diagram in which orange arrows indicate the node path of the BFS algorithm. The following is a width-first search in C++, where the algorithm passes through the nodes of the chart using an iterator that starts at the first node and continues until the last given node. it passes through all adjacent nodes before moving to alis nodes. Width-first path is using two variables: visitedNode and checkedNode. For this reason, the queue data structure is used for all nodes in order of arrival. In other words, the algorithm processes the nodes that arrived first in the queue. void BFS(graph &amp;g,int nodeStart){ int solde;&lt;int&gt; queue nodeQueue; list&lt;int&gt;:iterator it;visitedNode[nodeStart]=true; nodeQueue.push(nodeStart); while(!nodeQueue.empty(){ node=nodeQueue.front(),nodeQueue.pop(); checkedNode[node]=true; checkNode(solde); for(it=g.sides[node].begin(); it!=g.sides[node].end(); ++it){ if(!visitedNode[*it]))){nodeQueue.push(*it]) { checkSide(node ,*it); } } } } } } In addition, the two functions are called (checkNode and checkSide) to monitor the path of the node, just as in the following illustration: It is important to note that in the previous picture, the first two digits (8.8) represent the number of nodes and pages, followed by 8 pairs of numbers representing the relationships between nodes. Finally, the path of the BFS algorithm from node 1 to node 8 is displayed. BFS Applied Width's first search algorithm has several applications, What are: Finding the shortest path between two nodes measured by the number of nodes connected Prove if the chart is two-part (it can be divided into two parts) Finding a minimum extension tree in an unadserved diagram Creating Web crawler GPS navigation systems to find neighboring locations in unscensored charts (where nodes have no cost or weight), it is possible to use BFS algorithms to find the shortest path. If you have a set of activities represented by nodes and want to achieve a specific task by performing as few activities as possible, you can use this algorithm. in the case of a weighted chart, you can use the previous example, where actions have a specified time interval, modify the BFS algorithm, and obtain the Dijkstra algorithm (also known as the Shortest Path First algorithm). According to code theory, to decode word codes, it is possible to use the BFS to test whether the node chart is two-part. This result, in turn, checks whether the data is corrupted or not. An example is a Tanner diagram in which all word code numbers representing nodes on one side and a combination of nodes on the other side are numbers that are expected to rise to zero in a word code with no errors. Finding a minimum extension tree in an unadpared chart is another way to use the BFS algorithm. If you want to go through multiple locations without going through the same place twice, you can use this algorithm. If you want to create a website search engine, such as Google, you can use the Document Object Model (DOM) with a web crawler, an algorithm that usually uses wide first searches through HTML tags; in this way, you can: &lt;/int&gt; &lt;/int&gt; searches at DOM. In GPS navigation systems, you can use BFS algorithms for nearby points of interest. For example, if you have a set of attractions and services represented by nodes connected by geographical proximity, you can create a list of nearby attractions that can be used to plan a trip or tour. Conclusion The algorithms mentioned in this article can be used to find paths between two or more nodes, based on the nature of the problem. BFS algorithms can be used on Google Maps to find optimal routes between tourist destinations or to search for texts on a website using a web crawler. DFS algorithms can be used to solve rids or games such as labyrinth and chess. About Avantica If you are looking for a software partner who works for business goals and success, Avantica is your solution. We offer our customers dedicated teams, team additions and individual projects. We are constantly looking for the best methods to get the best results. Do you like what you read? Contact!   About the author