## Week 3

Display replies in nested form

Settings

The cut-off date for posting to this forum is reached so you can no longer post to it.

### Week 3

by Romana Riyaz (Instructor) - Thursday, 27 June 2024, 1:29 PM

Describe and compare both the depth-first and breadth-first search as a graph traversal. As part of your discussion, describe under what conditions or which problem each is best utilized to solve. Finally, your discussion must incorporate a description of how such traversals are implemented as greedy algorithms.

Include one or two examples to explain your thought process to show what is occurring and how the methodology works. Use APA citations and references for any sources used.

*76 words*

Permalink

### Re: Week 3

by Moustafa Hazeen - Saturday, 6 July 2024, 3:11 AM

Depth-First vs. Breadth-First Search: A Comparative Analysis
Graph traversal algorithms, such as Depth-First Search (DFS) and Breadth-First Search (BFS), are fundamental techniques used to explore and analyze graphs. Both algorithms operate by systematically visiting nodes but differ in their exploration strategies and applications.
Depth-First Search (DFS)
Depth-First Search explores a graph by going as deep as possible along each branch before backtracking. It starts at a selected node and explores as far as possible along each branch before backtracking. This strategy is implemented using a stack data structure (or recursion) to manage the nodes to be explored.
DFS is particularly useful in scenarios such as:
1. Finding connected components: DFS efficiently identifies all nodes reachable from a given node in an undirected graph, making it suitable for tasks like finding connected components.
2. Detecting cycles: By maintaining a record of visited nodes, DFS can detect cycles in a graph, which is crucial in various applications like deadlock detection in operating systems.
As a greedy algorithm, DFS makes decisions locally (choosing the next node to visit based on the last visited node) without considering the overall structure of the graph beyond the current path.
Example: Consider a maze represented as a graph where DFS can be employed to find a path from the starting point to the exit. By exploring each possible path until a dead end is reached, DFS effectively searches for a solution.
Breadth-First Search (BFS)
Breadth-First Search explores a graph level by level. It starts at a selected node and explores all its neighbors at the present depth level before moving on to nodes at the next depth level. BFS utilizes a queue data structure to manage the nodes to be explored, ensuring that all nodes at the current depth level are visited before moving on to deeper nodes.
BFS is well-suited for tasks such as:
1. Shortest path in an unweighted graph: Since BFS explores nodes level by level, it guarantees finding the shortest path in terms of number of edges in an unweighted graph.
2. Web crawling: BFS is used by search engines to crawl the web, exploring web pages layer by layer from a starting page.
As a greedy algorithm, BFS explores all nodes at the present depth level before moving on to deeper nodes, ensuring that it finds the shortest path first.
Example: In a social network represented as a graph, BFS can be applied to find the shortest path of acquaintances from one

person to another, as it explores each person's immediate friends before moving on to friends-of-friends and so on.
Greedy Algorithms in Graph Traversal
Both DFS and BFS are considered greedy algorithms because they make locally optimal choices at each step with the hope of finding a global optimum. In DFS, this manifests as choosing the next node based on the current path, whereas in BFS, it involves exploring all nodes at the current depth level before deeper nodes.
In conclusion, the choice between DFS and BFS depends on the specific problem requirements:
• DFS is suitable for problems like finding connected components and detecting cycles due to its deep exploration strategy.
• BFS is preferred for finding the shortest path in unweighted graphs and tasks requiring level-wise exploration.
Understanding these algorithms and their implementations is crucial for efficiently solving graph-related problems in various domains.
References
GeeksforGeeks. (2024b, May 29). Difference between BFS and DFS. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/
*566 words*

Permalink Show parent

**Re: Week 3**
by Romana Riyaz (Instructor) - Sunday, 7 July 2024, 1:51 AM

Hello Hazeen,
Thank you for your submission. Your comparative analysis of Depth-First Search (DFS) and Breadth-First Search (BFS) effectively highlights their key differences and specific use cases. You provide clear and accurate explanations of both algorithms, including their implementation using stack and queue data structures, respectively. The examples illustrating each algorithm's application, such as maze solving for DFS and shortest path finding in social networks for BFS, are well-chosen and relevant. Additionally, your discussion on how these algorithms function as greedy algorithms provides valuable insight into their local decision-making processes. Overall, this thorough and well-structured analysis offers a comprehensive understanding of when and why to use DFS or BFS in various graph-related problems.
Regards,
Romana Riyaz
*116 words*

Permalink Show parent

**Re: Week 3**
by Cherkaoui Yassine - Monday, 8 July 2024, 9:05 PM

Moustafa, I wanted to drop you a quick note to say excellent job on your discussion post! Your answer is not only clear but also very well-written. It's evident that you put thought and effort into it, and it really shines through. Keep up the great work!
*47 words*

Permalink Show parent

**Re: Week 3**
by Siraajuddeen Adeitan Abdulfattah - Tuesday, 9 July 2024, 7:57 PM

Hi Moustafa,

Excellent submission in response to the questions asked. You clearly stated the applications and gave detailed explanation of both DFS and BFS algorithm. you also rightly stated how they function as greedy algorithms and what problems each is best suited to solve. Your examples did clarify why and when to use DFS or BFS.
*56 words*

Permalink Show parent

**Re: Week 3**
by Benjamin Chang - Tuesday, 9 July 2024, 9:05 PM

Hi Moustafa,
It was great to see your high-quality discussion this week! You clearly explained both depth-first search and breadth-first search algorithms. While neither algorithm explores every single path, they are considered greedy algorithms because they prioritize finding the most efficient path first. This approach can be very helpful in real-world scenarios, like saving money on materials when building a network by focusing on the shortest route.
Yours
Benjamin
*69 words*

Permalink Show parent

**Re: Week 3**
by Mejbaul Mubin - Wednesday, 10 July 2024, 6:21 AM

Hi Moustafa,
Your post provides a comprehensive comparative analysis of Depth-First Search (DFS) and Breadth-First Search (BFS), clearly highlighting their exploration strategies, applications, and underlying principles.
*26 words*

Permalink Show parent

**Re: Week 3**
by Akomolafe Ifedayo - Wednesday, 10 July 2024, 7:19 PM

Hi Hazeen, great work on your discussion post. You clearly described the BFS and DFS, and the various areas they are well suited. Great work, keep it up.
*28 words*

Permalink Show parent

**Re: Week 3**
by Fadi Al Rifai - Wednesday, 10 July 2024, 11:29 PM

Hi Hazeen,
Thank you for your thoughtful contribution to a detailed explanation of both the DFS and BFS algorithms, and I like your description of examples.
Keep it up.
*29 words*

Permalink Show parent

**Re: Week 3**
by Naqaa Alawadhi - Thursday, 11 July 2024, 12:04 AM

Good job
*2 words*

Permalink Show parent

**Re: Week 3**
by Nour Jamaluddin - Thursday, 11 July 2024, 4:09 AM

Very interesting.
Your post is complete and have many valuable information. You were able to compare the two type of searches appropriately. I liked your explanations very much.
Thank you.
*30 words*

Permalink Show parent

### Re: Week 3

by Natalie Tyson - Thursday, 11 July 2024, 4:46 AM

Hey Moustafa,

Great job with the article, I thought you did a good job of describing the differences between BFS and DFS and used a site that many of us can agree is a great source to start with after the reading. Your examples and description of the way these traversal algorithms work was done well, hope you have a great week!

*62 words*

Permalink Show parent

### Re: Week 3

by Benjamin Chang - Sunday, 7 July 2024, 12:00 AM

Describe and compare both the depth-first and breadth-first search as a graph traversal. As part of your discussion, describe under what conditions or which problem each is best utilized to solve. Finally, your discussion must incorporate a description of how such traversals are implemented as greedy algorithms.

**Depth-First Search:**

According to Tutorialspoint, depth-first search (DFS) is a recursive algorithm for traversing all the vertices of a graph or tree data structure. This algorithm explores a graph in a depth ward motion and uses a stack to keep tracking of the next vertex to start a search when a dead end is reached. Simply speaking, DFS starts from the root, explores each vertex along a path to its deepest point, then backtracks to the nearest unexplored vertex, and continues this process until all vertices are visited.

**Breadth-First Search:**

Breadth-first search (BFS) also starts from the root but explores each adjacent node level by level before moving on to the next level of adjacent nodes. According to GeeksforGeeks, BFS involves visiting all the connected nodes of a graph in a level-by-level manner.

**Time and Space Complexity:**

For DFS, the algorithm is designed to explore every edge and vertex in the graph, resulting in a time complexity of O(|V| + |E|), where |V| is the number of vertices and |E| is the number of edges. The worst-case scenario occurs when the algorithm traverses a long path before finding the next vertex. The space complexity is O(|V|), primarily due to the stack used to track the path.

For BFS, the time and space complexity are the same as DFS, O(|V| + |E|) and O(|V|), respectively. BFS, however, explores nodes level by level, making it useful for finding the shortest path in unweighted graphs.

**As Greedy Algorithms:**

DFS can be viewed as a greedy algorithm because it makes locally optimal choices by going as deep as possible along a path before backtracking. This approach, however, can consume significant time and memory.

BFS, on the other hand, is greedy in its level-by-level exploration, which helps in finding the shortest path to the target node. This makes BFS efficient for solving problems where the shortest path is required.

In summary, while both DFS and BFS have similar complexities, their traversal strategies and use cases differ. DFS is suitable for pathfinding and exploring all possible solutions, whereas BFS is ideal for finding the shortest path in unweighted graphs and level-order traversal.

**Reference**

*First Search (DFS) algorithm*. (n.d.). https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm

eadth-First-Search-Algorithm.gif - Wikimedia Commons. (2009, March 26).

pth-First-Search.gif - Wikimedia Commons. (2009, March 26).

orGeeks. (2024, June 28). *Breadth first search or BFS for a graph*. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

*441 words*

Permalink Show parent

**Re: Week 3**
by Romana Riyaz (Instructor) - Sunday, 7 July 2024, 1:48 AM

Hello Chang,
Thank you for your submission. Your comparison of Depth-First Search (DFS) and Breadth-First Search (BFS) is well-articulated, highlighting their key differences and appropriate use cases. You clearly explain that DFS is best utilized for exploring all possible solutions and pathfinding in scenarios where backtracking is necessary, while BFS is ideal for finding the shortest path in unweighted graphs and level-order traversal. Your discussion on the time and space complexities of both algorithms is precise, noting that both have O(|V| + |E|) time complexity and O(|V|) space complexity. Additionally, you effectively describe how both traversals can be viewed as greedy algorithms, with DFS making locally optimal choices by exploring as deeply as possible and BFS by level-by-level exploration. This comprehensive yet concise explanation helps in understanding the practical applications and theoretical underpinnings of these fundamental graph traversal algorithms.
Regards,
Romana Riyaz
*142 words*

Permalink Show parent

**Re: Week 3**
by Cherkaoui Yassine - Monday, 8 July 2024, 9:06 PM

Hey, just wanted to give you props for your discussion post—it's fantastic! Your response is clear, articulate, and well-structured. It's evident you know your stuff and put in the effort to communicate effectively. Keep up the great work!
*39 words*

Permalink Show parent

**Re: Week 3**
by Siraajuddeen Adeitan Abdulfattah - Tuesday, 9 July 2024, 8:07 PM

Hi Benjamin,

Good submission in response to the questions asked. Your description of DFS and BFS is well articulated and informative, also including images showing the traversals in both algorithm. You rightly stated what both are suited for and clearly described their greedy algorithm nature.
*45 words*

Permalink Show parent

**Re: Week 3**
by Anthony Jones - Wednesday, 10 July 2024, 2:07 AM

Hello,

Good post! I liked your animations illustrating the search methods on a tree. The search methods look a bit different when it is a graph that is more complicated and maybe has cycles. In this case, DFS goes as far as possible in its searches, while BFS seeks to find solutions that are more proximate to the original node. What situations is DFS best for and what about BFS?

God bless!

Anthony

*73 words*

Permalink Show parent

**Re: Week 3**

by Akomolafe Ifedayo - Wednesday, 10 July 2024, 7:22 PM

Hi Chang, great work on your post.Your explanation of the depth-first and breadth-first search along with examples made your work engaging to go through. You also described their time and space complexity and how they can be viewed as greedy algorithms. Keep it up.

*44 words*

Permalink Show parent

**Re: Week 3**

by Fadi Al Rifai - Wednesday, 10 July 2024, 11:31 PM

Hi Chang,
Good work, Thanks for sharing your explanation about both the DFS and BFS algorithms, and I found the description of your examples and included images to be easier to follow.
Keep it up.

*35 words*

Permalink Show parent

**Re: Week 3**

by Naqaa Alawadhi - Thursday, 11 July 2024, 12:05 AM

Good job

*2 words*

Permalink Show parent

**Re: Week 3**

by Nour Jamaluddin - Thursday, 11 July 2024, 4:12 AM

Perfect job.
I really appreciate your way which included the diagrams. The differences were vary important to know and work with it.
Your sources are supportive.
Good luck!

*28 words*

Permalink Show parent

**Re: Week 3**

by Natalie Tyson - Thursday, 11 July 2024, 4:49 AM

Hey Chang,

Your description of the BFS and DFS traversal algorithms was descriptive, visual, and well worded. You did a great job

contrasting both of the these algorithms and described the time complexities between them in detail. You also gave an assessment of which algorithm that you would use depending on the requirements. Hope you have a wonderful week!

*59 words*

Permalink Show parent

**Re: Week 3**

by Jerome Bennett - Thursday, 11 July 2024, 8:06 AM

Greetings Benjamin,

Your work is very thorough and you expressed a clear distinction between the two techniques and when they are best utilized. The illustrations aided my understanding as well.

*30 words*

Permalink Show parent

**Re: Week 3**

by Chong-Wei Chiu - Thursday, 11 July 2024, 10:30 AM

Hi, Benjamin Chang. Thank you for sharing your point of view about BFS and DFS. You explained the concepts of BFS and DFS clearly. Especially, you used animation to illustrate how to traverse graphs using DFS and BFS. I think that is amazing.

*43 words*

Permalink Show parent

**Re: Week 3**

by Mahmud Hossain Sushmoy - Sunday, 7 July 2024, 7:06 PM

Depth-First Search (DFS) and Breadth-First Search (BFS) are essential graph traversal methods, each employing a unique strategy. DFS explores vertically, going as deep as possible along each branch before backtracking, while BFS explores horizontally, examining all neighbors at the current depth before moving deeper.

DFS typically uses a stack or recursion and is best suited for path finding, cycle detection, and topological sorting. It's memory-efficient and useful when solutions are expected to be far from the root. In contrast, BFS uses a queue and excels at finding shortest paths in unweighted graphs and performing level-order traversals. It's ideal when solutions are likely near the starting point (GeeksforGeeks, 2024).

Both algorithms can be considered greedy in their approach. DFS greedily chooses to explore deeply along each branch, while BFS greedily explores all neighbors at the current level. This local decision-making without global consideration characterizes their greedy nature.

To illustrate, let’s consider this tree:

DFS would traverse in the order A, B, D, E, C, F, going deep before backtracking. BFS would traverse as A, B, C, D, E, F, exploring each level fully before proceeding.

The choice between DFS and BFS depends on the problem specifics and graph structure. DFS is often preferred for memory-constrained problems, while BFS is better for shortest path problems in unweighted graphs.

**References**

GeeksforGeeks. (2024, May 29). *Difference between BFS and DFS*. Retrieved from https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

University of the People. (n.d.). *UNIT 3: Optional Video Lectures*. Retrieved from https://my.uopeople.edu/mod/folder/view.php?id=423550

*242 words*

Permalink Show parent

### Re: Week 3

by Siraajuddeen Adeitan Abdulfattah - Tuesday, 9 July 2024, 8:12 PM

Hi Mahmud,

Excellent submission in response to the questions asked. Your explanation of DFS and BFS is educative and informative, you also stated their efficient use cases. Your description of greedy algorithm for both DFS and BFS is well articulated and easy to understand.

*44 words*

Permalink Show parent

### Re: Week 3

by Benjamin Chang - Tuesday, 9 July 2024, 9:14 PM

Hi Mahmud,
You explained both algorithms clearly and I like your post, in addition to the fact that you have precisely defined how it operates, particularly in tree, it is really excellent that you have explained the advantages of both DFS and BFS. Keep up the good work!
Yours
Benjamin

*50 words*

Permalink Show parent

### Re: Week 3

by Mahmud Hossain Sushmoy - Tuesday, 9 July 2024, 10:21 PM

This is the tree photo I meant to add with the post.

*12 words*

Permalink Show parent

### Re: Week 3

by Akomolafe Ifedayo - Wednesday, 10 July 2024, 7:24 PM

Hi Mahmud, your work was concise and straight to the point as you explained the Depth-first and Breadth-first algorithm, and how they can be considered greedy in their approach. Your use of examples were interesting to read. Keep it up.

*40 words*

Permalink Show parent



### Re: Week 3

by Nour Jamaluddin - Thursday, 11 July 2024, 4:14 AM

Well done.
You were able to summarize the points in a good frame. I liked your choices as well. You understand the concepts appropriately.
Thank you

*26 words*

Permalink Show parent

## Re: Week 3

by Fadi Al Rifai - Sunday, 7 July 2024, 8:30 PM

Depth-First Search (DFS) and Breadth-First Search (BFS) are fundamental graph traversal algorithms used in computer science for exploring the nodes and edges of a graph.

Here's a detailed comparison and discussion of each:

**Depth-First Search (DFS)**

**Description:**

- **Approach**: DFS explores as far down a branch as possible before backtracking. It uses a stack-based mechanism, implemented either directly with a stack or implicitly through recursive calls.
- **Process**:

1. Start at the root node (or an arbitrary node in the case of a graph).
2. Explore each branch completely before moving to the next branch.
3. If a node has no unvisited neighbors, backtrack to the previous node and explore new branches.

**Implementation:**

DFS can be realized by leveraging either recursive function calls or a direct stack data structure.

Here's a basic recursive implementation:

```
def dfs (graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)  # Process the node
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
```

**Use Cases:**

- **Path Finding**: DFS is useful when searching for a path in a maze or a puzzle where the solution is located deep in the tree.
- **Topological Sorting**: DFS is used in topological sorting of a directed acyclic graph (DAG).
- **Connected Components**: Finding all connected components in a graph.

**Breadth-First Search (BFS)**

**Description:**

- **Approach**: BFS explores all neighbors of a node before moving to the next level. It uses a queue data structure.
- **Process**:

1. Start at the root node (or an arbitrary node in the case of a graph).
2. Explore all neighbors at the present depth level before moving on to nodes at the next depth level.
3. Continue this process until all nodes are visited.

**Implementation:**

BFS is typically implemented using a queue. Here's a basic implementation:

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)
    while queue:
        vertex = queue.popleft()
        print(vertex)  # Process the node
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
```

**Use Cases:**

- **Shortest Path**: BFS is ideal for finding the shortest path in an unweighted graph.
- **Level Order Traversal**: In trees, BFS can be used to traverse nodes level by level.
- **Finding Connected Components**: BFS can also be used to find all connected components in a graph.

**Greedy Algorithm Aspect**

Although DFS and BFS are not typically categorized as greedy algorithms, they exhibit greedy properties in their traversal strategies:

- **Greedy Nature of DFS**: DFS makes a greedy choice by exploring as far as possible down one branch before backtracking. It does not look ahead to see the consequences of its choices, which aligns with the greedy algorithm's local optimization property.
- **Greedy Nature of BFS**: BFS greedily explores all neighbors at the current depth level before moving to the next level. By visiting nodes in the order of their distance from the start node, BFS ensures that it explores all shortest paths first.

**Choosing Between DFS and BFS**

- **Memory Usage**: DFS typically requires less memory than BFS. DFS only needs to store the stack of nodes on the current path, while BFS needs to store all nodes at the current depth level.
- **Completeness**: In finite graphs, both DFS and BFS will visit all nodes. However, in infinite graphs, BFS is complete (it will find a solution if one exists), while DFS might get stuck in an infinite branch.
- **Optimality**: BFS guarantees finding the shortest path in an unweighted graph, while DFS does not guarantee this.
- **Implementation Simplicity**: DFS can be simpler to implement using recursion.

In my opinion, DFS and BFS are powerful graph traversal algorithms with distinct characteristics and optimal use cases. DFS is best suited for deep exploration and problems where the solution is expected to be found deep within the graph, while BFS is ideal for finding the shortest path and exploring all nodes at the current depth before moving deeper. Both algorithms exhibit greedy-like properties in their traversal decisions.

**Reference:**

**Topic 1: Graphs as data structures**
Chapter 11 Graphs, Sections 11.1 – 11.3 in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.
Optionally review Chapter 3 Decomposition of Graphs in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap3.pdf
**Topic 2: Graph Traversals**
Chapter 11 Graphs, Sections 11.1 – 11.3 in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.
Optionally review Chapter 4 Paths in Graphs in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap4.pdf
**Topic 3: Greedy Algorithms**
Read the lecture 14 notes from Design and Analysis of Algorithms available from: http://www.cse.ust.hk/~dekai/271/notes/L14/L14.pdf
Read Chapter 5 Greedy Algorithms in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap5.pdf
**Supplemental Materials**
The following are video lectures that are available via YouTube and other sources that are related to the topics in the unit and can be used as a supplemental resource for students looking for more details or to be introduced to the same topic from another source. The use of these resources is not required and is entirely optional.

*832 words*

Permalink Show parent

### Re: Week 3
by Cherkaoui Yassine - Monday, 8 July 2024, 9:06 PM

Fadi, I wanted to take a moment to acknowledge your discussion post—it's really well-done! Your answer is clear, concise, and well-articulated. It's evident you put thought and care into your response, and it's paying off. Keep up the excellent work!
*41 words*

Permalink Show parent

### Re: Week 3
by Moustafa Hazeen - Tuesday, 9 July 2024, 3:18 PM

Fadi, you've provided a comprehensive and clear comparison of Depth-First Search (DFS) and Breadth-First Search (BFS) as graph traversal algorithms. Your explanations of their processes, implementations, and use cases are well-structured and easy to follow. Your inclusion of examples such as path finding and shortest path problems effectively illustrates the practical application of each algorithm.

Your discussion on the greedy nature of DFS and BFS, although not traditionally categorized as greedy algorithms, effectively explains how their traversal strategies exhibit greedy-like properties. This adds depth to your explanation and ties back to their practical implementations.

Overall, your submission is thorough and meets the requirements of the assignment. Great job!
*108 words*

Permalink Show parent

### Re: Week 3
by Siraajuddeen Adeitan Abdulfattah - Tuesday, 9 July 2024, 8:20 PM

Hi Fadi,

Excellent submission in response to the questions asked. You gave a detailed and educative explanation on both DFS and BFS algorithms, with their well articulated use cases. You also shared implementation of both algorithms and then gave a clear, informative description of greedy algorithm for both DFS and BFS.

*51 words*

Permalink Show parent

**Re: Week 3**

by Benjamin Chang - Tuesday, 9 July 2024, 9:21 PM

Hi Fadi,
I appreciate the approach in which you presented both DFS and BFS using coding; it is simple to comprehend, and the way in which you expressed how DFS and BFS operate was easily expressed using Python. To tell you the truth, I really appreciate these lines of code. The Python code is simple to comprehend and has brief functions; you did a great job!

Yours
Benjamin

*68 words*

Permalink Show parent

**Re: Week 3**

by Naqaa Alawadhi - Thursday, 11 July 2024, 12:07 AM

Good job is

*3 words*

Permalink Show parent

**Re: Week 3**

by Natalie Tyson - Thursday, 11 July 2024, 4:51 AM

Hey Fadi,

Your discussion was awesome, you put a lot of thought, material research, and worded the article very well so it was easy to follow. The practical applications of each algorithm was discussed and you did a great job with visual illustrations of the code used to implement each algorithm, thank you for an insightful article. Have a great week!

*61 words*

Permalink Show parent

**Re: Week 3**

by Jerome Bennett - Thursday, 11 July 2024, 8:14 AM

Greetings Fadi,

You've done a great job explaining DFS and BFS. You broke down how they work, how to implement them, and when to use each one. The examples you gave, like finding paths and shortest routes, really help show how these algorithms are used in real situations.

*48 words*

Permalink Show parent

**Re: Week 3**

by Chong-Wei Chiu - Thursday, 11 July 2024, 10:49 AM

Hello, Fadi Al Rifai. Thank you for sharing your point of view on this topic. You illustrate in detail how to implement depth-first search and breadth-first search using code. Furthermore, you also explain their differences and their implementation details in greedy algorithms. In general, you fully cover all the requirements of this discussion.
*53 words*

Permalink Show parent

## Re: Week 3

by Mejbaul Mubin - Monday, 8 July 2024, 1:00 AM

**Depth-First Search (DFS)**

**Description**: Depth-first search (DFS) is a graph traversal technique that explores as far as possible along each branch before backtracking. It uses a stack data structure, either explicitly or implicitly through recursion, to keep track of the vertices to visit next.

**Algorithm**:

1. Start at the root (or any arbitrary node).
2. Push the starting node onto the stack.
3. Pop the stack to get the current node.
4. If the current node has unvisited neighbors, push them onto the stack.
5. Repeat steps 3 and 4 until the stack is empty.

**Example**: Consider a graph with nodes A, B, C, D, and E. The traversal starting from node A might follow this order: A, B, D, C, E, assuming that the nodes are processed in alphabetical order.

**Best Utilized**:

- **Path Finding**: DFS is useful when searching for a path from a start node to a goal node, especially when the path doesn't need to be the shortest.
- **Connected Components**: It can be used to determine connected components in a graph.
- **Topological Sorting**: Useful in directed acyclic graphs (DAGs) for tasks like scheduling (Cormen et al., 2009).

**Breadth-First Search (BFS)**

**Description**: Breadth-first search (BFS) is a graph traversal technique that explores all neighbors at the present depth level before moving on to nodes at the next depth level. It uses a queue data structure to keep track of the vertices to visit next.

**Algorithm**:

1. Start at the root (or any arbitrary node).
2. Enqueue the starting node.
3. Dequeue the queue to get the current node.
4. Enqueue all unvisited neighbors of the current node.
5. Repeat steps 3 and 4 until the queue is empty.

**Example**: Consider the same graph with nodes A, B, C, D, and E. The traversal starting from node A might follow this order: A, B, C, D, E, assuming that nodes are processed in alphabetical order.

**Best Utilized**:

- **Shortest Path Finding**: BFS is optimal for finding the shortest path in an unweighted graph.
- **Level Order Traversal**: It is particularly useful in scenarios like level-order traversal of a tree.
- **Network Broadcasting**: Suitable for broadcasting information in a network where all nodes must receive the data simultaneously (Sedgewick & Wayne, 2011).

**Greedy Algorithm Context**

**DFS as a Greedy Algorithm**:

- **Nature**: DFS can be considered greedy as it makes the immediate choice to go as deep as possible along a branch before considering other branches.
- **Example**: In a maze-solving scenario, DFS can be used to go as far as possible in one direction before backtracking when it hits a dead end. It continues this process until the goal is found or all possibilities are exhausted.

**BFS as a Greedy Algorithm**:

- **Nature**: BFS is also greedy as it explores nodes level by level, choosing the shortest path to all reachable nodes from the start.
- **Example**: In a shortest-path problem in an unweighted graph, BFS can be used to find the shortest path from the start node to the goal node by exploring all neighbors at each level.

**Comparative Examples**

**Example 1: Maze Solving**

- **DFS**: In a maze, DFS will explore a path from the start until it reaches the end or a dead end. If it reaches a dead end, it backtracks and tries another path.
- **BFS**: BFS, on the other hand, will explore all possible paths level by level. This means it will find the shortest path from the start to the end if one exists.

**Example 2: Social Network Connections**

- **DFS**: To explore all connections from a person, DFS will dive into one person's connections, then their connections' connections, and so on, until all connections are explored.
- **BFS**: For finding the shortest connection path between two people in a social network, BFS is more suitable as it explores all direct connections first before moving to the next level.

**Implementation and Complexity**

- **DFS Implementation**:

*Python:*

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited
```

· DFS has a time complexity of O(V + E), where V is the number of vertices and E is the number of edges. Space complexity is O(V) due to the stack.

· **BFS Implementation**:

from collections import deque

*python:*

```
def bfs(graph, start):
    visited = set()
    queue = deque([start])
```

```
while queue:
    vertex = queue.popleft()
    if vertex not in visited:
        visited.add(vertex)
        queue.extend(graph[vertex] - visited)
return visited
```

BFS also has a time complexity of O(V + E) and a space complexity of O(V) due to the queue.

**References**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley Professional.

*772 words*

Permalink Show parent

### Re: Week 3

by Romana Riyaz (Instructor) - Monday, 8 July 2024, 1:25 AM

Hello Mubin,
Thank You for your submission. Your explanation of Depth-First Search (DFS) and Breadth-First Search (BFS) is thorough and well-organized. You effectively describe each algorithm's process, highlighting their respective uses and advantages in different scenarios. The examples provided for both DFS and BFS, such as maze-solving and social network connections, clearly illustrate their applications and how they differ in approach. Your inclusion of implementation snippets in Python for both algorithms and their complexity analysis adds practicality to the theoretical understanding, reinforcing the efficiency and scalability of each approach in graph traversal. Overall, your explanation is comprehensive and insightful, providing a solid foundation for understanding DFS and BFS in various contexts.
Regards,
Romana
*113 words*

Permalink Show parent

### Re: Week 3

by Moustafa Hazeen - Tuesday, 9 July 2024, 3:19 PM

Mejbaul, you've presented a thorough comparison of Depth-First Search (DFS) and Breadth-First Search (BFS) as graph traversal algorithms, along with clear descriptions of their processes, implementations, and optimal use cases. Your examples, such as maze solving and social network connections, effectively illustrate the practical applications and differences between DFS and BFS.

You've also appropriately discussed DFS and BFS in the context of greedy algorithms, highlighting their greedy-like properties in making traversal decisions. This enriches the understanding of these algorithms beyond their basic mechanics.

Your implementations of both DFS and BFS are well-structured and align with their respective algorithms, demonstrating a good understanding of how these algorithms are implemented in practice.

Overall, your submission is comprehensive and meets the requirements of the assignment
*122 words*

Permalink Show parent

**Re: Week 3**
by Fadi Al Rifai - Wednesday, 10 July 2024, 11:33 PM

Hi Mubin,
Your post was well-detailed about the DFS and BFS algorithms, and I like your examples, so your submission is comprehensive and meets the requirements of the assignment.
Keep it up.
*32 words*

Permalink Show parent

**Re: Week 3**
by Siraajuddeen Adeitan Abdulfattah - Monday, 8 July 2024, 7:29 PM

Graph traversal is the visitation of vertices of a graph in some specific order, based on the topology of the graph (Shaffer, 2011).

Depth-first search (DFS): traverse a graph from the start vertex (node) through a possible path, going to the end of the depth of that path and then returns to the next vertex to continue the search until all vertices have been visited. It traverses the graph in a recursive manner employing stack structure (Shaffer, 2011).

Breadth-first search (BFS): Begins graph traversal through several path at a time, traversing all vertices connected to start vertex (node) first before moving to the vertices below. It traversal is done on a level by level basis and it uses the concept of queue (Shaffer, 2011).

Comparison between DFS and BFS:

• DFS traverses the depth from the start vertex, through a path until the last vertex is reached on that path. It then backtracks and move to the next vertex. BFS traverses all vertices at the same level before moving further down to the next level.
• DFS uses stack to visit and keep track of vertices visited while BFS uses queue to vist and keep track of vertices visited.
• DFS uses less space and is memory efficient, whereas BFS uses more memory space due to the concept of queue employed.

DFS is best deployed or applied to a situation where the maximum depth of a graph or a tree is to be found. However, BFS is best deployed or applied where the shortest path in an unweighted graph is to be found.
One of the real-world applications of these traversals that I remember vividly is the process determining the best possible route over the internet.

Greedy algorithm is in phases, where it starts with the best possible option of the moment with any regard for future impact or consequences. Greedy algorithm works with the concept of hoping that choosing the best possible path/option right now or locally will translate to choosing the best or optimum path or option in the end or globally.
Typical examples include the minimum spanning tree and traveling salesman's problem.
Reference:

Shaffer, C. (2011). A Practical Introduction to Data Structures and Algorithm Analysis. Blacksburg: Virginia. Tech. Retrieved from: Practical_Into_to_Data_Structures_and_Algorithms_-_Shaffer.pdf (uopeople.edu)
*375 words*

Permalink Show parent

**Re: Week 3**
by Manahil Farrukh Siddiqui - Tuesday, 9 July 2024, 5:42 PM

Hello Siraajuddeen,

Your explanation of graph traversal is clear Overall, your answer is well explained.
*15 words*

Permalink Show parent



### Re: Week 3

by Tousif Shahriar - Wednesday, 10 July 2024, 6:34 PM

Hello Siraajuddeen,
Great work! The explanation of DFS and BFS is technically accurate and aligns with standard definitions. The author correctly identifies the use of stack and queue data structures and the different traversal methods. However, the discussion could be enhanced by providing specific examples or visual aids to illustrate how DFS and BFS operate in practice.
Thanks for sharing!

*60 words*

Permalink Show parent

### Re: Week 3

by Cherkaoui Yassine - Monday, 8 July 2024, 7:48 PM

**Introduction:** Graph traversal techniques are fundamental in computer science for navigating through data structures. Two primary methods of graph traversal are Depth-First Search (DFS) and Breadth-First Search (BFS).

**Depth-First Search (DFS):** DFS explores a graph by starting at an initial node and traversing as far along each branch as possible before backtracking.

**Implementation:** In a typical DFS implementation, we use a stack to keep track of the nodes to visit. If recursion is used, the call stack implicitly serves this purpose.

Algorithm:

1. Initialize an empty stack and push the starting node.

2. Mark the starting node as visited.

3. While the stack is not empty:

- Pop the top node.

- For each adjacent unvisited node: Mark it as visited / Push it onto the stack.

**Conditions and Applications:**DFS is particularly useful for problems involving exhaustive searches of all possible paths, such as finding solutions to puzzles or mazes, or for scenarios where the path itself is important, such as in topological sorting or solving certain game states.

**Example:** Consider a maze represented as a graph. DFS can be used to navigate from the start to the finish by exploring one path to its end before backtracking and trying another path if the current one fails.

**Breadth-First Search (BFS):** BFS explores a graph by visiting all the nodes at the present depth level before moving on to nodes at the next depth level.

**Implementation**: In a typical BFS implementation, we use a queue to manage the nodes to visit next.

**Algorithm:**

1. Initialize an empty queue and enqueue the starting node.

2. Mark the starting node as visited.

3. While the queue is not empty:

- Dequeue the front node.

- For each adjacent unvisited node: Mark it as visited. / Enqueue it.

**Conditions and Applications**:BFS is best suited for problems requiring the shortest path in unweighted graphs, such as finding the shortest path in a map or networking problems like finding the shortest route in a network.

**Example:** Consider finding the shortest path in an unweighted grid-like map. BFS can quickly find the shortest route from the start to the destination by exploring all nodes level by level.

**DFS and BFS as Greedy Algorithms:** Greedy algorithms make a series of choices, each of which looks the best at the moment, aiming for a global optimum. While DFS and BFS are not greedy in the purest sense, their traversal methods exhibit characteristics akin to greedy strategies:

- DFS:makes a greedy choice to follow a path until it cannot proceed, then backtracks to explore other paths.

- BFS: makes a greedy choice to explore nodes level by level, ensuring the shortest path is found in an unweighted graph.

**Conclusion:**Both DFS and BFS are essential graph traversal techniques with distinct applications. DFS is beneficial for exhaustive path searches and scenarios where the path taken matters, while BFS is optimal for finding the shortest path in unweighted graphs and level-order traversal.

**References:**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

*510 words*

Permalink Show parent

### Re: Week 3

by Romana Riyaz (Instructor) - Monday, 8 July 2024, 10:37 PM

Cherkouia,
Thank you for your submission. Your explanation of graph traversal techniques, specifically Depth-First Search (DFS) and Breadth-First Search (BFS), is thorough and informative. The introduction clearly states the importance of graph traversal techniques and introduces DFS and BFS effectively. The explanation of DFS is concise and covers the key points well. Consider providing a simple example or diagram to visually represent the stack operations during DFS traversal. The BFS section is clear and provides a good understanding of the algorithm.Similar to DFS, a visual aid or example would enhance comprehension, especially for the queue operations.
The applications of both DFS and BFS are well-explained. Including specific real-world examples can make the content more relatable. The comparison of DFS and BFS to greedy algorithms is an interesting perspective. It might help to briefly define what a greedy algorithm is for readers who may not be familiar with the term.
Best,
Romana
*151 words*

Permalink Show parent

### Re: Week 3

by Moustafa Hazeen - Tuesday, 9 July 2024, 3:20 PM

Cherkaoui, your submission provides a clear and structured comparison of Depth-First Search (DFS) and Breadth-First Search (BFS) as graph traversal algorithms. You effectively explain their algorithms, implementations using stacks (DFS) and queues (BFS), and highlight their respective optimal use cases.

The examples you provided, such as navigating a maze with DFS and finding the shortest path in a map with BFS, are well-chosen and help illustrate the practical applications of these algorithms. Your discussion on how DFS and BFS exhibit characteristics similar to greedy algorithms due to their traversal strategies adds depth to your analysis.

Furthermore, your conclusion succinctly summarizes the key differences and strengths of DFS and BFS, reinforcing their distinct roles in graph traversal.

Overall, your submission meets the assignment requirements and provides a comprehensive overview of DFS and BFS.

*132 words*

Permalink Show parent

### Re: Week 3

by Anthony Jones - Wednesday, 10 July 2024, 2:38 AM

Hello,

Good post! DFS is best for applications like mazes where we need to see a path to its completion before we can know if it's good or not. BFS is best for applications like the greedy method of finding the MST where we only need to evaluate proximate solutions.

Obviously, the suitability of one search method depends on the situation and implementation, but does one search technique fit the underlying idea behind greedy algorithms (use the local best to find the global best) better than the other?

God bless!

Anthony

*91 words*

Permalink Show parent

### Re: Week 3

by Winston Anderson - Tuesday, 9 July 2024, 12:24 AM

**Depth-First Search (DFS) vs. Breadth-First Search (BFS)**

**Overview**

Depth-First Search (DFS) and Breadth-First Search (BFS) are fundamental algorithms used for traversing or searching graph data structures. Both algorithms are used to explore nodes and edges of a graph, but they differ significantly in their approach and applications.

**Depth-First Search (DFS)**

DFS explores as far as possible along each branch before backtracking. It uses a stack data structure, either explicitly or via recursion, to keep track of the vertices to be explored next.

**Implementation**

DFS can be implemented using a stack (iterative approach) or recursion (recursive approach). Here is a basic recursive implementation in Python:

**```python```**

**Applications and Use Cases**

- Topological Sorting: Useful in scheduling problems.

- Cycle Detection: Identifies cycles in a graph.

- Path Finding: Finds paths in mazes and puzzles.

- Connected Components: Identifies connected components in a graph.

#### Breadth-First Search (BFS)

BFS explores all the vertices at the present depth level before moving on to the vertices at the next depth level. It uses a queue data structure to keep track of the vertices to be explored next.

#### Implementation

BFS can be implemented using a queue. Here is a basic implementation in Python:

**```python```**

#### Applications and Use Cases

- Shortest Path: Finds the shortest path in unweighted graphs.

- Level Order Traversal: Useful in tree traversals.

- Connected Components: Identifies connected components in a graph.

- Bipartite Graph Testing: Tests if a graph is bipartite.

## Comparison

| Feature | DFS | BFS |
|---|---|---|
| **Data Structure** | Stack (or recursion) | Queue |
| **Traversal Method** | Depth-first | Level-order (breadth-first) |
| **Memory Usage** | O(V) for stack | O(V) for queue |
| **Path Finding** | Can find any path | Finds the shortest path in unweighted graphs |
| **Cycle Detection** | Yes | Yes |
| **Applications** | Topological sorting, cycle detection | Shortest path, level order traversal |
| **Complexity** | O(V + E) | O(V + E) |

#### Greedy Algorithm Context

While DFS and BFS are not inherently greedy algorithms, they can be adapted to greedy contexts. A greedy algorithm makes a locally optimal choice at each step with the hope of finding a global optimum.

#### Greedy Best-First Search (GBFS)

GBFS is a combination of BFS and greedy algorithms, where the next node to explore is chosen based on a heuristic that estimates the cost to reach the goal.

Example: Finding the shortest path in a weighted graph using a heuristic function.

**```python```**

**Conclusion**

DFS and BFS are essential graph traversal algorithms with distinct characteristics and applications. DFS is suitable for problems requiring deep exploration, such as cycle detection and topological sorting, while BFS is ideal for finding the shortest path in unweighted graphs and level-order traversal. Both algorithms can be adapted to greedy contexts, particularly in pathfinding and optimization problems, by incorporating heuristic functions to guide the search process.

**References**

*BFS Graph Algorithm(With code in C, C++, Java and Python)*. (n.d.). https://www.programiz.com/dsa/graph-bfs

GeeksforGeeks. (2024, June 27). *Depth First Search or DFS for a Graph*. GeeksforGeeks. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

GeeksforGeeks. (2024, June 28). Breadth First Search or BFS for a Graph. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

GeeksforGeeks. (2024, January 18). *Greedy Best first search algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

Moore, K., Jennison, K., & Khim, J. (n.d.). *Depth-First Search (DFS)*. Brilliant. https://brilliant.org/wiki/depth-first-search-dfs/

*513 words*

Permalink Show parent

**Re: Week 3**
by Manahil Farrukh Siddiqui - Tuesday, 9 July 2024, 5:45 PM

Hello Winston,

Your explanation of DFS and BFS is detailed. Overall, your answer is in good detail but needs succinctness and clarity.
*22 words*

Permalink Show parent

**Re: Week 3**
by Tousif Shahriar - Wednesday, 10 July 2024, 6:40 PM

Hello Winston,
Great post! You included a lot of details that helped me understand both DFS and BFS better.
The Python code snippets help to thoroughly understand their implementations.
Thanks for sharing!
*32 words*

Permalink Show parent

**Re: Week 3**
by Jerome Bennett - Thursday, 11 July 2024, 8:20 AM

Greetings Winston,

Writing actual Python code to illustrate the implementation of DFS and BFS is very effective and suited to real-life scenarios. Your work helped me to have a clearer understanding of the differences between the two.
*37 words*

Permalink Show parent

**Re: Week 3**
by Manahil Farrukh Siddiqui - Tuesday, 9 July 2024, 2:37 AM

Depth-First Search (DFS) and Breadth-First Search (BFS) are two essential strategies in the dynamic field of graph traversal, each with a unique way of working. DFS acts like an explorer, exploring every avenue in great detail before taking a detour (Mohan, 2023). On the other hand, BFS functions as a meticulous inspector, going over all neighbouring paths in a tiered manner right from the beginning to ensure every level is thoroughly reviewed before moving further.

DFS manages its traversal using a stack that adheres to the Last In, First Out (LIFO) concept (GeeksforGeeks, 2019b). Nodes are pushed onto the stack as it investigates. When it comes to a dead end, it removes the top node from the stack to go back. Instead, BFS uses a queue that runs on the First In, First Out (FIFO) principle to investigate nodes at each level before going deeper (Simplilearn, n.d.).

The needs of the problem determine which of DFS and BFS to use. DFS is better at discovering paths near the starting point; it is particularly good at locating paths and identifying cycles in graphs (GeeksforGeeks, 2019b). DFS works well in intricate designs like mazes. However, BFS works best for determining node connectedness, locating nodes within a specific range of a particular node, and determining the shortest path between nodes in an unweighted network (GeeksforGeeks, 2019a).

Think about trying to locate a lost friend in a maze. If the friend is distant from the beginning, DFS might not be effective and could take many needless steps as it would swiftly explore many different paths and branches (Simplilearn, n.d.). On the other hand, BFS would methodically search every hallway and chamber to ensure the friend is finally located, but it would take longer if they were further into the maze.

In a nut shell, DFS and BFS are reliable graph traversal methods with specific benefits and drawbacks. It is necessary to comprehend these features to choose the best approach for addressing different graph-related problems.

**References**

GeeksforGeeks. (2019a, February 4). Breadth First Search or BFS for a Graph - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

GeeksforGeeks. (2019b, February 4). Depth First Search or DFS for a Graph - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

Mohan, M. (2023, April 2). Breadth-First Search vs Depth-First Search in Graph Algorithms. Codedamn News. https://codedamn.com/news/algorithms/dgraph-algorithms-bfs-vs-dfs

Simplilearn. (n.d.). BFS (Breadth-first search) Algorithm: Overview, Examples & More | Simplilearn. Simplilearn.com. https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm

*393 words*

Permalink Show parent

**Re: Week 3**
by Michael Oyewole - Tuesday, 9 July 2024, 12:31 PM

Hi Manahil,
Thank you for your view on the assignment. The two essential algorithms for navigating or exploring graphs and trees are Breadth-First Search (BFS) and Depth-First Search (DFS). And they are very important with their differences. For example, the queue data structure is used by BFS (Breadth First Search) to determine the quickest path. The stack data structure is used by DFS (Depth First Search). Thanks for your ideas.
*70 words*

Permalink Show parent

**Re: Week 3**
by Mahmud Hossain Sushmoy - Tuesday, 9 July 2024, 10:39 PM

Hello Manahil,
Your comparison of Depth-First Search (DFS) and Breadth-First Search (BFS) offers a clear and engaging explanation of these fundamental graph traversal strategies. By likening DFS to an explorer and BFS to an inspector, you provide memorable metaphors that help illustrate their distinct approaches. The analogy of finding a friend in a maze further emphasizes the practical implications of choosing the right traversal method. Thank you for your contribution to the discussion forum this week.
*76 words*

Permalink Show parent

**Re: Week 3**
by Tyler Huffman - Tuesday, 9 July 2024, 6:03 AM

Comparing and contrasting a depth-first search and a breadth-first search is most easily accomplished using a simple binary tree (which after all, is just a type of restricted graph) as an example. Let us say we have a tree with 7 nodes and this structure:

Node A will be our 'root' or starting node in this context. Depth-first, as implied by the name, goes as deep as it can down a branch, starting from a root and does not check the next branch until it has reached the end. This is often handled with a auxiliary stack data structure. A stack fits perfectly with the nature of this traversal. Node A will be added to this stack (which is empty at the start), popped off and inspected. The children of A are then added to the stack. B is popped off, inspected, and then it's children are added to the stack (on top of C). This continues until all nodes have been visited. While a parent-child relationship is not typically used to describe graphs as with trees, the same logic still applies. The nodes will be visited in this order:

Breadth-first works in a more horizontal fashion. Breadth simply means width or 'side to side' and that is exactly how it works here as well. Breadth-first searches level by level; an auxiliary queue structure can be used to facilitate this traversal. Node A (level 1 of the tree) is added to the queue, it's dequeued and inspected and it's children, B and C (level 2), are queued up to be inspected next. When done with A, B is dequeued, inspected and any potential children of it are added to our queue. This continues until the queue has nothing left to inspect. The nodes will be visited in this order:

*299 words*

Permalink Show parent



### Re: Week 3

by Michael Oyewole - Tuesday, 9 July 2024, 12:26 PM

Hi Tyler,
Thank you for sharing. In other words, Depth First Search, or DFS, is an edge-based method. Using the Stack data structure, it operates in two stages: visited vertices are popped if there are no visited vertices, and else they are placed onto the stack. I hope this is helpful on the assigned assignment. Thanks

*56 words*

Permalink Show parent

### Re: Week 3

by Mahmud Hossain Sushmoy - Tuesday, 9 July 2024, 10:28 PM

Hello Tyler,
Your explanation of depth-first search (DFS) and breadth-first search (BFS) using a simple binary tree is clear and effective. By starting with the structure of the tree and using Node A as the root, you provide a concrete example that makes the traversal processes easy to follow. The step-by-step breakdown of how DFS uses a stack and BFS uses a queue helps to illustrate the differences in their traversal methods. Additionally, providing the order of traversal for both methods (ABDECFG for DFS and ABCDEFG for BFS) concretely demonstrates their distinct approaches. Great work!

*95 words*

Permalink Show parent

### Re: Week 3

by Anthony Jones - Wednesday, 10 July 2024, 2:29 AM

Hello,

Good post!
DFS can have more variety depending on how it is implemented:
**Option 1 (which you used)**
Steps: Check the current node, then evaluate the left child, then the right child
Pseudo code:

```
function search(Node){
    evaluate(Node)
    search(Node.leftChild)
    search(Node.rightChild)
}
```

Order: ABDECFG

**Option 2**

Steps: Check the left child, then evaluate the current node, then the right child

Pseudo code:

```
function search(Node){
    search(Node.leftChild)
    evaluate(Node)
    search(Node.rightChild)
}
```

Order: DBEAFCG

**Option 3**

Steps: Check the left child, then evaluate the current node, then the right child

Pseudo code:

```
function search(Node){
    search(Node.leftChild)
    search(Node.rightChild)
    evaluate(Node)
}
```

Order: DEBFGCA

And you could search right before left for all of those options.

Each option has it's own strengths and situations its suitable for. BFS doesn't have as much flexibility, simply the order we evaluate each level (ABCDEFG vs ACBGFED).

Which is easier to code DFS or BFS?

God bless!

Anthony

*145 words*

Permalink Show parent



### Re: Week 3

by Chong-Wei Chiu - Thursday, 11 July 2024, 10:22 AM

Hello, Tyler Huffman. Thank you for sharing your opinion on this topic. You used two simple graphs to illustrate how to traverse a graph using DFS and BFS. However, I think you should add the references you used in this post, as that would make your post more complete.

*49 words*

Permalink Show parent

### Re: Week 3

by Anthony Jones - Tuesday, 9 July 2024, 7:50 AM

Graph traversal is an important part of algorithmic problem-solving. Often problems can be represented using graphs, so being able to have an effective means of traversing a graph is important.

#### Depth-first search

Depth-first search (DFS) is a search method that recursively evaluates the edges of a node in a graph (GeeksforGeeks, 2012a). This means that we choose an edge that leads to a node and then evaluate the nodes branching from that node, before evaluating the next edge. This is good for evaluating all the combinations and paths because it can be implemented rather simply using recursion. However, it seems to be a bit more like a brute-force method of traversing a tree. However, the greedy method of using the local best to find the global best can be applied so that the algorithm picks the optimal path when recursively evaluating.

Depth-first search is best when the local minima are close to the global minimum (i.e. minimal backtracking is necessary)

### Breadth-first search

"Breadth First Search (BFS) is a graph traversal algorithm that explores all the vertices in a graph at the current depth before moving on to the vertices at the next depth level" (GeeksforGeeks, 2012b). Essentially, BFS searches via the levels of the graphs. So it searches all the closer nodes first by running through all the nodes that are the same number of edges away from the original node. This is a lot harder to implement in code than DFS. If we look at the following graph and begin with B, we would first evaluate A, C, and G (distance of one edge from B), then H, I, E, F, and D (two edges away):

The breadth-first algorithm is best when the node we are looking for is close to the nodes we are using to look for it (ex. the greedy algorithm for finding the minimum spanning tree)

### Conclusion

Overall, DFS and BFS are good ways of traversing a graph with different strengths and weaknesses. However, whether they apply to an optimization problem, especially using a greedy method depends on how they are applied (Chien-hsiang, 2022), otherwise, they are just methods of brute-force traversing all the nodes. So it is important to understand how and when to apply the traversal methods.

### References

Chien-hsiang, H. (2022, September 22). *Is BFS/DFS a greedy algorithm? What's the difference between greedy and heuristic algorithm?* Medium. https://hungchienhsiang.medium.com/is-bfs-dfs-a-greedy-algorithm-whats-the-difference-between-greedy-and-heuristic-algorithm-8b6b019c43c1

GeeksforGeeks. (2012a, March 15). *Depth first search or DFS for a graph*. GeeksforGeeks. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/#

GeeksforGeeks. (2012b, March 20). *Breadth first search or BFS for a graph*. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/#

*423 words*

Permalink Show parent



### Re: Week 3

by Michael Oyewole - Tuesday, 9 July 2024, 12:24 PM

Hi Anthony,
Thank you for your post. The vertex-based method known as Breadth-First Search, or BFS, is used to determine the graph's shortest path. It makes use of a first-in, first-out queue data structure. One vertex is chosen at a time in BFS, visited, marked, and its neighbors visited and queued up afterward. This is my view on the topic. Thanks

*61 words*

Permalink Show parent

### Re: Week 3

by Mahmud Hossain Sushmoy - Tuesday, 9 July 2024, 10:41 PM

Hello Anthony,
Your essay provides a clear and concise overview of Depth-First Search (DFS) and Breadth-First Search (BFS), effectively highlighting their methodologies and use cases. Your comparison between DFS and BFS, particularly in terms of their ease of implementation and practical uses, is insightful. Additionally, the conclusion aptly summarizes the strengths and weaknesses of both methods, emphasizing the importance of choosing the appropriate traversal method for different optimization problems. Thank you for your contribution to the discussion forum this week.

*80 words*

Permalink Show parent

## Re: Week 3

by Michael Oyewole - Tuesday, 9 July 2024, 12:13 PM

Two essential algorithms for navigating or exploring graphs and trees are Breadth-First Search (BFS) and Depth-First Search (DFS).

The vertex-based method known as Breadth-First Search, or BFS, is used to determine the graph's shortest path. It makes use of a first-in, first-out queue data structure. One vertex is chosen at a time in BFS, visited, marked, and its neighbors visited and queued up afterward. More slowly than DFS, it is.

Depth First Search, or DFS, is an edge-based method. Using the Stack data structure, it operates in two stages: visited vertices are popped if there are no visited vertices, and else they are placed onto the stack.

Examples

(Difference between BFS and DFS, 2024)

Difference Between BFS and DFS
Parameters BFS DFS
Stands for BFS stands for Breadth First Search. DFS stands for Depth First Search.
Data Structure The Queue data structure is used by BFS (Breadth First Search) to determine the quickest path.
The stack data structure is used by DFS (Depth First Search).

Definition BFS is a traversal strategy where we go to the next level after going through every node on the current level.
Another traversal technique is DFS, in which we start at the root node and work our way through the nodes as much as we can until we arrive at the node that has no neighboring nodes that have not yet been visited.

Conceptual Disparity
BFS constructs the tree layer by layer.
DFS constructs the tree,
subtree by subtree.
Method employed
It operates according to the First In, First Out (FIFO) principle.
It operates according to the LIFO (Last In First Out) principle.

Ideal for
BFS works better when looking for vertices that are nearer the specified source.
When there are solutions that are not from the source, DFS works better.

Uses
Applications for BFS include shortest paths, bipartite graphs, and more.
Applications for DFS include acyclic graphs and the identification of strongly connected components, among others

Reference
Difference between BFS and DFS. (2024 May 29). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

*334 words*

Permalink Show parent

## Re: Week 3

by Manahil Farrukh Siddiqui - Tuesday, 9 July 2024, 5:39 PM

Hello Michael,

Your comparison of BFS and DFS is detailed. Overall, the content is accurate. Good job on providing detailed examples and references.

*23 words*

Permalink Show parent

**Re: Week 3**

by Winston Anderson - Wednesday, 10 July 2024, 10:21 AM

Hi Michael,
You have provided a structured and informative comparison between Breadth-First Search (BFS) and Depth-First Search (DFS), outlining their fundamental differences, data structures, and applications.

*26 words*

Permalink Show parent

**Re: Week 3**

by Jerome Bennett - Wednesday, 10 July 2024, 2:41 AM

**CS 3304 Discussion Forum Unit 3**

**Depth-First Search (DFS) and Breadth-First Search (BFS)**

***Depth-First Search (DFS)***

How It Works:

DFS starts at a node and explores as far down a branch as possible before backtracking. It's like diving deep into a single path before trying another (GeeksforGeeks, 2024). You can implement DFS using a stack (either an explicit stack or recursion).

Best Used For:

Path Finding: Like solving mazes.

Topological Sorting: Scheduling tasks.

Connected Components: Finding all connected nodes (GeeksforGeeks, 2022).

Tree Traversals: Inorder, Preorder, Postorder.

***Breadth-First Search (BFS)***

How It Works:

BFS starts at a node and explores all its neighbors before moving to the next level. It's like scanning level by level. BFS uses a queue to track nodes (GeeksforGeeks, 2024).

Best Used For:

Shortest Path in Unweighted Graphs: Finding the quickest route (GeeksforGeeks, 2022).

Level Order Traversal: Traversing trees level by level.

Connected Components: Same as DFS.

***Comparing DFS and BFS***

Traversal Order: DFS goes deep; BFS goes wide.

Memory Usage: DFS uses less memory than BFS.

Path Finding: DFS is quick but doesn't guarantee the shortest path; BFS guarantees the shortest path in unweighted graphs (GeeksforGeeks, 2024).

Use Cases: DFS for exploring possibilities like puzzles, BFS for finding shortest paths or level-order traversal.

**Greedy Algorithm Connection**

While DFS and BFS aren't traditional greedy algorithms, they make decisions based on the current best choice:

DFS: Chooses to go deep.

BFS: Chooses to explore all neighbors first, ensuring the shortest path.

***Examples***

1. Maze Solving with DFS:

DFS explores each path thoroughly, useful for finding any path from start to finish.

2. Shortest Path in a Social Network with BFS:

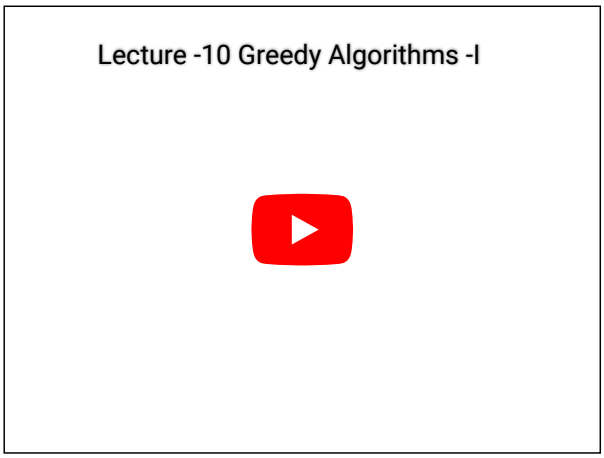
BFS finds the shortest connection between two people by exploring all immediate connections first (nptelhrd, 2008).

Reference

GeeksforGeeks. (2024, June 28). Breadth first search or BFS for a graph. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

GeeksforGeeks. (2022, August 24). When to use DFS or BFS to solve a Graph problem? GeeksforGeeks. https://www.geeksforgeeks.org/when-to-use-dfs-or-bfs-to-solve-a-graph-problem/

nptelhrd. (2008, August 27). Lecture -10 Greedy Algorithms -I [Video]. YouTube.




*332 words*

Permalink Show parent

**Re: Week 3**
by Mejbaul Mubin - Wednesday, 10 July 2024, 6:24 AM

Hi Jerome,
Your post provides a clear and concise comparison of Depth-First Search (DFS) and Breadth-First Search (BFS). The structure of your post is logical and easy to follow. Breaking down the explanations into "How It Works" and "Best Used For" sections makes it very reader-friendly. Your explanation of DFS is clear. Adding a brief note on how the stack (explicit or recursion) works to keep track of nodes could further aid understanding. The examples provided are relevant and well-chosen. You might also mention DFS's role in algorithms like Tarjan's for finding strongly connected components.
*95 words*

Permalink Show parent

**Re: Week 3**
by Winston Anderson - Wednesday, 10 July 2024, 10:16 AM

Hi Jerome,
This is a concise and clear comparison between Depth-First Search (DFS) and Breadth-First Search (BFS) that effectively outlines their respective functionalities, optimal use cases, and differences. The inclusion of practical examples, like maze solving and social network path finding, is particularly helpful in illustrating the real-world applications of these algorithms.
*52 words*

Permalink Show parent

**Re: Week 3**
by Tousif Shahriar - Wednesday, 10 July 2024, 6:48 PM

Hey Jerome,
Good post! The explanations of how DFS and BFS work are concise and clear, making them easy to understand. However, the "Best Used For" sections could benefit from more detailed explanations or examples to illustrate why each use case is suitable for DFS or BFS.
Thanks for sharing!
*50 words*

Permalink Show parent

**Re: Week 3**
by Tyler Huffman - Thursday, 11 July 2024, 12:44 AM

Well done Jerome, your post is concise and informative.

One area I would expand on is with statements like this: "BFS uses a queue to track nodes ". More detail about this is done would have been a nice addition here. This is a minor point however; your post still effectively summarizes DFS and BFS.
*55 words*

Permalink Show parent

**Re: Week 3**
by Prince Ansah Owusu - Wednesday, 10 July 2024, 3:35 AM

To effectively compare depth-first search (DFS) and breadth-first search (BFS) as graph traversal algorithms and understand their implementations as greedy algorithms, let's delve into their characteristics, applications, and implementations.

**Depth-First Search (DFS)**

**Characteristics:**

- DFS explores as far as possible along each branch before backtracking (Cormen et al., 2009).

- It uses a stack (either implicitly through recursion or explicitly) to keep track of vertices.
- Depth-first traversal can be implemented using recursive calls or an explicit stack data structure.

**Applications:**

- **Connected Components:** DFS can determine the connected components of a graph.
- **Cycle Detection:** It can detect cycles in a graph.
- **Path Finding:** Useful for finding paths between two vertices (Skiena, 2008).
- **Topological Sorting:** DFS can be used to perform topological sorting of a Directed Acyclic Graph (DAG) (Cormen et al., 2009).

**Best Conditions to Use:**

- When the goal is to go as deep as possible into the graph quickly.
- For problems where finding a path or determining connectivity is important.
- Not ideal for finding shortest paths in unweighted graphs.

**Greedy Algorithm Implementation:**

- **Implementation:** In DFS, at each step, the algorithm chooses the next vertex that is adjacent to the current vertex and hasn't been visited yet.
- **Example:** Consider a maze-solving problem where the goal is to find any path from the start to the exit. DFS would explore paths deeply until it finds a solution, making local decisions without considering the entire maze at once.

**Breadth-First Search (BFS)**

**Characteristics:**

- BFS explores all neighbor nodes at the present depth level before moving on to nodes at the next depth level (Cormen et al., 2009).
- It uses a queue data structure to manage the vertices to be explored.
- BFS ensures that it visits all vertices at the present "depth" level before moving on to the vertices at the next level.

**Applications:**

- **Shortest Path:** BFS finds the shortest path in an unweighted graph.
- **Minimal Spanning Tree:** Used in finding the Minimal Spanning Tree (MST) of a graph.
- **Web Crawling:** Useful for crawling the web, where BFS ensures visiting all pages at the current depth level before moving deeper (Skiena, 2008).
- **Garbage Collection:** Used in memory management to find reachable objects.

**Best Conditions to Use:**

- When the goal is to find the shortest path in an unweighted graph.
- For problems requiring level-wise exploration or finding the minimum steps required to reach a solution.

**Greedy Algorithm Implementation:**

- **Implementation:** In BFS, the algorithm explores all neighbors of a vertex before moving on to the next level vertices. It makes locally optimal choices at each step (choosing all neighbors at the current level) to eventually achieve a global optimum (shortest path or level-wise exploration).
- **Example:** Consider a social network where BFS is used to find the shortest path between two individuals. BFS would explore connections layer by layer, ensuring the shortest path is found.

**Comparison and Conclusion**

- **DFS vs. BFS:** DFS is useful for exploring deeply, while BFS is better for exploring broadly.
- **Greedy Algorithm Aspect:** Both DFS and BFS can be seen as greedy algorithms in their implementations within the graph context because they make local decisions (choosing next vertices) to reach an overall goal (finding paths or exploring connectivity).

In conclusion, understanding the differences between DFS and BFS as graph traversal algorithms is crucial depending on the problem's requirements—whether it's finding paths, exploring connectivity, or ensuring efficiency in traversal. Their implementations as greedy algorithms highlight their effectiveness in making locally optimal decisions to achieve broader goals in graph analysis.

**References:**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Skiena, S. S. (2008). *The Algorithm Design Manual*. Springer Science & Business Media.

*605 words*

Permalink Show parent

### Re: Week 3

by Mejbaul Mubin - Wednesday, 10 July 2024, 6:23 AM

Hi Prince,
Your post presents an insightful and thorough comparison between Depth-First Search (DFS) and Breadth-First Search (BFS). Your introduction is clear and sets up the comparative analysis effectively. Adding a brief statement about the importance of graph traversal in computer science and real-world applications could provide more context and engage readers right from the start. The characteristics and applications of DFS are well articulated. It's great that you cited authoritative sources like Cormen et al. and Skiena. Your description of BFS is comprehensive and highlights key aspects effectively.

*89 words*

Permalink Show parent

### Re: Week 3

by Winston Anderson - Wednesday, 10 July 2024, 10:08 AM

Hi Prince,
You have provided a well-structured and informative piece comparing depth-first search (DFS) and breadth-first search (BFS). You have clearly outlined the characteristics, applications, and conditions best suited for each algorithm, which enhances understanding. Additionally, your inclusion of references adds credibility.

*42 words*

Permalink Show parent

### Re: Week 3

by Liliana Blanco - Thursday, 11 July 2024, 9:21 AM

Your breakdown of DFS and BFS is fantastic! I appreciate how you explained their traits and use cases. Your real-life examples and insights on implementation as greedy algorithms are really helpful. Thanks for the thorough and easy-to-understand analysis!

*38 words*

Permalink Show parent

### Re: Week 3

by Muritala Akinyemi Adewale - Wednesday, 10 July 2024, 1:12 PM

**Depth-First Search vs. Breadth-First Search: Graph Traversal Strategies**

Depth-first search (DFS) and breadth-first search (BFS) are fundamental algorithms for traversing graphs. They differ in their exploration strategy, leading to distinct use cases and implementations as greedy algorithms.

**Depth-First Search (DFS):**

DFS explores a graph as deeply as possible along a chosen path before backtracking and exploring other branches. It can be visualized as navigating a maze, taking one turn at a time and going as far as possible down that path until you hit a dead end.

**Implementation as a Greedy Algorithm:**

DFS is a natural fit for greedy algorithms because it prioritizes exploring the current path as deeply as possible. Here's a basic implementation using a stack:

1. **Mark the starting vertex as visited.**
2. **Push the starting vertex onto a stack.**
3. **While the stack is not empty:**
    - **Pop a vertex from the stack.**
    - **For each unvisited neighbor of the popped vertex:**
        - **Mark the neighbor as visited.**
        - **Push the neighbor onto the stack.**

**When to Use DFS:**

- Finding paths between two specific vertices: DFS excels at finding a path quickly, even if it's not the shortest one.
- Cycle detection: DFS can efficiently detect cycles in a graph by encountering a previously visited vertex while exploring a path.
- Topological sorting: In a directed acyclic graph (DAG), DFS can be used to order vertices such that for every directed edge (u, v), vertex u comes before v in the ordering.

**Example:**

Imagine searching a building for a fire exit. DFS would be a good choice. You could start at a room and explore each hallway in turn, going as far down each corridor as possible until you find an exit or reach a dead end (locked door).

**Breadth-First Search (BFS):**

BFS explores all the vertices at the current level (distance from the starting point) before moving to the next level. It's like exploring a maze by checking all the rooms on one floor before moving down to the next floor.

**Implementation as a Greedy Algorithm:**

BFS prioritizes exploring all the closest neighbors first, making it a good greedy approach for finding the shortest path. Here's a basic implementation using a queue:

1. **Mark the starting vertex as visited.**
2. **Enqueue the starting vertex.**
3. **While the queue is not empty:**
    - **Dequeue a vertex from the queue.**
    - **For each unvisited neighbor of the dequeued vertex:**
        - **Mark the neighbor as visited.**
        - **Enqueue the neighbor.**

**When to Use BFS:**

- Finding the shortest path between two vertices: BFS guarantees finding the shortest path in an unweighted graph.
- Finding all connected components: BFS can efficiently identify all the connected vertices in a graph.
- Level-order traversal of a tree: BFS is ideal for traversing a tree level by level.

**Example:**

Imagine searching all the rooms on a floor for a missing pet. BFS would be a good approach. You could start at a room and check all the rooms directly connected to that room, then move on to the rooms one level away, ensuring you've explored all possibilities on the current floor before moving down.

**Comparison:**

| **Feature** | **Depth-First Search (DFS)** | **Breadth-First Search (BFS)** |
|---|---|---|

| | | |
|---|---|---|
| Exploration Strategy | Explores as deeply as possible along a single path | Explores all neighbors at the current level before moving to the next level |
| Data Structure | Stack | Queue |
| Finding Shortest Path | Not guaranteed (unless edge weights are equal) | Guaranteed (in unweighted graphs) |
| Cycle Detection | Efficient | Not efficient |
| Topological Sorting | Can be used in DAGs | Not applicable |

**Choosing the Right Algorithm:**

The choice between DFS and BFS depends on the specific problem you're trying to solve. Here are some general guidelines:

- Use DFS for finding a path quickly, cycle detection, or topological sorting in DAGs.
- Use BFS for finding the shortest path in unweighted graphs, finding connected components, or level-order tree traversal.

**Reference:**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press. (This is a classic textbook on algorithms and provides detailed explanations of DFS and BFS)

*654 words*

Permalink Show parent



### Re: Week 3

by Tousif Shahriar - Wednesday, 10 July 2024, 6:44 PM

Hello Muritala,
Nice post! The comparison table and guidelines for choosing the right algorithm are very helpful. They succinctly summarize the key differences and provide practical advice.
Thanks for sharing!
*30 words*

Permalink Show parent



### Re: Week 3

by Tousif Shahriar - Wednesday, 10 July 2024, 4:56 PM

A key component of computer science and graph theory is graph traversal. Depth-First Search (DFS) and Breadth-First Search (BFS) are the two main methods used for graph traversal. Each of these methodologies has unique characteristics and is suited to different types of problems.

Depth-First Search (DFS):

DFS traverses a graph by beginning at a starting node, moving as far as possible along each branch before backtracking. This method makes use of a stack data structure, either directly through an iterative process or implicitly through recursion (Cormen et al., 2009).

Implementation:

- Start from a source node and push it onto the stack.
- Pop a node from the stack, mark it as visited, and push all its adjacent unvisited nodes onto the stack.
- Continue the process until the stack is empty.

Best Utilization:

- DFS is excellent for finding paths between two nodes, especially if the solution is deep in the graph.

- In directed or undirected graphs, DFS can help detect cycles (Gross & Yellen, 2005).
- DFS can be used in Directed Acyclic Graphs (DAGs) to perform topological sorting (Cormen et al., 2009).

As an example, we can consider a graph that represents a maze where nodes are junctions and edges are paths since it is much easier to imagine just by looking at a graph. DFS can be used to explore all possible paths from one node to another, from the start to the finish, so from the first node to the last node. Doing so DFS considers all possible routes.

Here, DFS will start from 1 and will explore all the edges in the following order: 1, 2, 4, 3, 5.

Breadth-First Search (BFS):

BFS traverses through every node at the current depth level first in a graph before going on to nodes at the next depth level. Queue data structure is used in this approach (Weiss, 2012).

Implementation:

- Start from a source node and enqueue it.
- Dequeue a node, mark it as visited, and enqueue all its adjacent unvisited nodes.
- Repeat the process until the queue is empty.

Best Utilization:

- BFS is optimal for finding the shortest path in unweighted graphs.
- BFS can be used to traverse nodes level by level, which is useful in scenarios like networking to determine the shortest path.
- BFS helps in finding all nodes connected to a given node in undirected graphs (Gross & Yellen, 2005).

Using the same graph above we can provide a BFS example as follows:

Here, BFS will start from node 1, BFS would explore the root node first, then all nodes at depth 1, then all nodes at depth 2, and so on. Hence the order would be 1, 2, 3, 4, 5.

Implementation as greedy algorithms:

DFS and BFS both can be implemented as greedy algorithms because at each step, they make a locally optimal choice with the intent of finding a global solution. In DFS, the greedy choice is to go as deep as possible, while in BFS, it is to explore the nearest nodes first. These choices are made without considering the broader structure of the graph, thus embodying the greedy algorithm principle (Cormen et al., 2009).

Example:

DFS: If we consider the maze example, DFS will follow a path until it hits a dead end, then backtrack to explore other paths. Its depth-oriented approach reflects a greedy choice to go deeper.

BFS: In a social network, finding the shortest connection path between two users is efficiently handled by BFS, which greedily explores all immediate connections before moving on to connections at the next level.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

Gross, J. L., & Yellen, J. (2005). Graph Theory and Its Applications (2nd ed.). Chapman & Hall/CRC.

Weiss, M. A. (2012). Data Structures and Algorithm Analysis in Java (3rd ed.). Pearson.

*640 words*

Permalink Show parent

**Re: Week 3**
by Wingsoflord Ngilazi - Thursday, 11 July 2024, 3:18 AM

Thank you for sharing this great post. Keep up the good work.
*12 words*

Permalink Show parent

**Re: Week 3**
by Akomolafe Ifedayo - Wednesday, 10 July 2024, 6:55 PM

*Describe and compare both the depth-first and breadth-first search as a graph traversal. As part of your discussion, describe under what conditions or which problem each is best utilized to solve. Finally, your discussion must incorporate a description of how such traversals are implemented as greedy algorithms.*

**The Depth-First Search (DFS)** identifies all the vertices of a graph that can be reached from a designated starting point (Dasgupta et al., 2006). Explicit paths to these vertices are found and summarized in its search tree.
Unlike trees, graphs may contain cycles and a graph can have more than one DFS traversal. This is why the DFS algorithm is known for searching tree or graph data structures (GeeksforGeeks, 2024). The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

**Application of DFS:**

1. It is used for detecting a cycle in a graph: If we see a back edge during DFS, a graph has a cycle. So a DFS can be run for the graph and back edges can be checked for

2. Path Finding: The DFS algorithm can be used to find a path between two given vertices s and z

3. To test if a graph is bipartite: It can be used when we want to check if a new vertex doesn't link two vertices of the same color

4. Web crawlers: It can be used in the implementation of web crawlers to explore the links on a website
5. Maze generation: DFS search can be used to generate random mazes (GeeksforGeeks, 2024).

**Advantages of DFS:**

1. For the search graph, the memory requirement is only linear

2. Since only the nodes on the current path are stored, it requires less memory

**Disadvantages of DFS:**

1. There is a possibility that it may down the left-most path forever

2. If more than one solution is present, there is no guarantee to find a minimal solution (GeeksforGeeks, 2024).

**The Breadth First Search (BFS)** is a fundamental graph traversal algorithm that is used to connect nodes of a graph in a level-by-level manner (GeeksforGeeks, 2024). It is commonly used for pathfinding, connected components, and shortest path problems in graph.

**Applications of BFS:**

1. Shortest path finding: It can be used to find the shortest path between two nodes in an unweighted graph

2. It is used to detect cycles in a graph

3. It can be used to identify connected components in a graph

4. In network protocols, BFS is implemented for routing data packets

5. It can be used to perform a level-order traversal of a binary tree

**Advantages of BFS:**

1. It is easily programmable

2. If there is a solution, BFS will find it (GeeksforGeeks, 2023).

**Disadvantages of BFS:**

1. The main disadvantage is its memory requirement. Since it is severely space-bound in practice, the memory available on typical computers are exhausted in a matter of minutes.

**Differences between BFS and DFS:**

1. BFS uses the Queue data structure for finding the shortest path while DFS uses the stack data structure

2. BFS builds the tree level by level while DFS builds it sub-tree by sub-tree

3. BFS is more suitable for searching vertices closer to the given source while DFS is more suitable when there are solutions away from the source

4. BFS uses the concept of FIFO (First In First Out), while DFS uses the concept of LIFO (Last In First Out) (GeeksforGeeks, 2024).

References

Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). Algorithms. Berkeley, CA: University of California Berkeley, Computer Science Division. Available at http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

GeeksforGeeks. (2023). Applications, Advantages, and Disadvantages of Breadth First Search (BFS). https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/?ref=lbp

GeeksforGeeks. (2024). Applications, Advantages, and Disadvantages of Depth First Search (DFS). https://www.geeksforgeeks.org/applications-of-depth-first-search/?ref=lbp

GeeksforGeeks. (2024). Breadth First Search or BFS for a Graph. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

GeeksforGeeks. (2024). Depth First Search or DFS for a Graph. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

GeeksforGeeks. (2024). Difference between BFS and DFS. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/?ref=lbp

*651 words*

Permalink Show parent

### Re: Week 3

by Muritala Akinyemi Adewale - Thursday, 11 July 2024, 3:08 AM

Nice work, Jerome! Your summary of DFS and BFS is well-written and concise. Adding a visual representation, such as a diagram or flowchart, to illustrate the traversal process could make your explanation even clearer. This is a minor suggestion, as your post already does a great job of covering the basics. Keep up the good work!

*56 words*

Permalink Show parent

### Re: Week 3

by Wingsoflord Ngilazi - Thursday, 11 July 2024, 3:15 AM

HI Akomolafe,
Your work is concise and the explanations are very clear. Keep up the good work .

*18 words*

Permalink Show parent

### Re: Week 3

by Aye Aye Nyein - Wednesday, 10 July 2024, 7:40 PM

**Depth-First Search (DFS)**

Depth-First Search (DFS) is a graph traversal technique that explores as far as possible along each branch before backtracking. It can be implemented using either recursion or an explicit stack data structure. Here is a summary of its characteristics:

- **Traversal Strategy**: DFS places a higher priority on branch exploration after delving deeply into the graph.

- **Greedy Aspect**: In every step, DFS selects a vertex from the unvisited set that is next to the current vertex in a greedy manner.

- **Applications**: DFS is very helpful in solving puzzles involving connected component identification, graph cycle detection, and labyrinth path navigation.

**Example:**

Consider a simple graph:

A

/ | \

B C D

| |

E F

Starting from vertex A:

1. Visit A

2. From A, visit B, then E

3. Backtrack to B, visit C, then F

4. Backtrack to C, visit D

DFS traversal order: A -> B -> E -> C -> F -> D

**Breadth-First Search (BFS)**

Another graph traversal technique is Breadth-First Search (BFS), which investigates neighbor nodes at the current depth level before going on to nodes at the subsequent depth level. Usually, a queue data structure is used to control the exploration's order. Important traits consist of:

- **Traversal Strategy**: BFS investigates every node at the current depth level before transitioning to the subsequent depth level's nodes.

- **Aspect of Greed**: Before proceeding to the next depth level, BFS explores all the current vertex's neighbors in a greedy manner.

- **Applications**: BFS is a good choice for discovering all connected components, equal node exploration (level by level), and shortest path computation in unweighted graphs.

**Example:**

Using the same graph starting from vertex A:

1. Visit A

2. From A, visit B, C, D

3. From B, visit E

4. From D, visit F

BFS traversal order: A -> B -> C -> D -> E -> F

**Comparison and Problem Suitability**

**Traversal Style**: DFS is generally more memory efficient than BFS because it only needs to store a stack of vertices whereas BFS needs to maintain a queue.

**Applications**:

- Use DFS to explore as far as possible down each branch before backtracking. This is ideal for problems involving depth-first search, such as maze solving or topological sorting.

- Use BFS to find the shortest path in an unweighted graph or explore all nodes level by level. BFS ensures that the first time a node is visited, it is visited at the shortest path possible.

**Implementation as Greedy Algorithms**

Both DFS and BFS can be seen as greedy algorithms because they make locally optimal choices (visiting the next unvisited vertex) with the hope of finding a global optimum (e.g., shortest path or complete traversal). These algorithms do not reconsider earlier choices, which aligns with the greedy approach of making the best choice at each step without worrying about future consequences beyond the current step.

In summary, DFS and BFS are fundamental graph traversal algorithms with distinct characteristics that make them suitable for different types of problems. Their implementation as greedy algorithms involves making locally optimal choices at each step to achieve an overall solution efficiently.

**Reference:**

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms, 3rd Edition. The MIT Press.

*549 words*

Permalink Show parent

### Re: Week 3

by Muritala Akinyemi Adewale - Thursday, 11 July 2024, 3:07 AM

Hello Aye,
Well done, Jerome! Your post provides a solid overview of DFS and BFS. It might be helpful to include a brief example of each algorithm in action, showing how nodes are added to and removed from the stack or queue. This would enhance understanding for readers who are new to these concepts. Overall, a very informative summary!
*59 words*

Permalink Show parent

### Re: Week 3

by Wingsoflord Ngilazi - Thursday, 11 July 2024, 3:10 AM

Hi Aye,
Your discussion post demonstrates how deeply understand the two traversal techniques. Well done.
*15 words*

Permalink Show parent



### Re: Week 3

by Naqaa Alawadhi - Wednesday, 10 July 2024, 11:50 PM

Depth-first search (DFS) and breadth-first search (BFS) are graph traversal algorithms used to visit all the vertices of a graph. DFS explores as far as possible along each branch before backtracking, while BFS explores all the vertices at the present depth before moving on to the next level.

DFS is best utilized for problems like topological sorting, finding connected components in an undirected graph, and solving

puzzles with multiple solutions. It is implemented as a greedy algorithm by recursively visiting adjacent unvisited vertices until it reaches a dead end, then backtracking.

BFS is suitable for finding the shortest path in an unweighted graph, solving puzzles with one solution, and exploring the nearest nodes first. It is implemented as a greedy algorithm by visiting all neighbors of a vertex before moving on to their neighbors.

For example, consider a simple graph with nodes A, B, C, D connected in a linear fashion: A -> B -> C -> D. When applying DFS starting from node A, it would explore A -> B -> C -> D before backtracking. In contrast, BFS would explore A -> B first before moving on to C and then D.

In summary, DFS explores deeply before backtracking and is suitable for certain problems like topological sorting. BFS explores broadly level by level and is ideal for finding shortest paths. Both algorithms are implemented greedily by making locally optimal choices at each step.

References:
Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
*257 words*

Permalink Show parent

### Re: Week 3
by Wingsoflord Ngilazi - Thursday, 11 July 2024, 12:29 AM

**Describe and compare both the depth-first and breadth-first search as a graph traversal. As part of your discussion, describe under what conditions or which problem each is best utilized to solve. Finally, your discussion must incorporate a description of how such traversals are implemented as greedy algorithms.**

Depth-First Search can be described as a graph traversal technique that begins at a specific node (the root) and explores as far as possible along each branch before backtracking. It employs a stack data structure, either explicitly (using a stack) or implicitly (using recursion) (Asano et al., 2014)

According to Asano et al.( 2014)**,** the algorithm works as follows:

i. Begin at the root node and push it onto the stack (or call recursively).

ii. Pop the top node, mark it as traversed, and push all its untraversed nearby nodes onto the stack.
iii. Repeat step ii until the stack is empty.

The graph below shows how the Depth-First-Search traversal works.

e - f - g

| |

h - i

Starting at node e, the Depth-First-Search traversal would be: e -> f -> g -> i -> h.

The Depth-First-Search traversal has several use cases, some of which are listed below:

i. Finding a path in mazes or puzzles.

ii. Topological sorting in scheduling problems.
iii. Solving puzzles like Sudoku or N-Queens where a complete exploration of one path is necessary before backtracking.

One of its key advantages is that it does not require a lot of memory as it goes deep rather than wide. Additionally, it is easy to implement with recursion. It is, however, important to note that it can get stuck in a loop or explore deep, unfruitful paths in large graphs without a path limit.

Breadth-First Search traversal, on the other hand, is a graph traversal technique that begins its search at a specific node and explores all its neighboring nodes at the current depth level before moving on to nodes at the next depth level (Beamer et al., 2013). It employs a queue data structure.

It works as follows:

i. Begin at the root node and enqueue it.

ii. Dequeue a node, mark it as traversed, and enqueue all its untraversed nearby nodes.
iii. Repeat step 2 until the queue is empty.

(Scarpazza et al., 2013)

The following graph illustrates how it operates.

e - f- g

| |

h - i

Beginning at node e, the Breadth-First-Search traversal would be: e -> f-> h -> g -> i.

This traversal has several use cases, some of which are listed below.

i. Finding the shortest path in unweighted graphs.

ii. Level-order traversal in trees.
iii. Network broadcasting where spreading information layer by layer is needed.

Its advantages include finding the shortest path in an unweighted graph and guaranteeing to visit all nodes at the present depth level before moving deeper. However, unlike the other traversal, it requires more memory, especially for wide graphs (Beamer et al., 2013).

**Greedy Algorithms**

It is interesting to note that both Depth-First -Search and Breadth-First-Search can be viewed as greedy algorithms in the sense that they make a locally optimal choice at each step.

In the case of Depth-First -Search , a greedy choice is made to the deepest unvisited node next. On the other hand, a Breath-First-Search makes a greedy choice to explore the closest unvisited node next. The uses cases discussed above are examples of how greedy algorithms are implemented.

.

**References**

Asano, T., Izumi, T., Kiyomi, M., Konagaya, M., Ono, H., Otachi, Y., ... & Uehara, R. (2014, November). Depth-first search using bits. In *International Symposium on Algorithms and Computation* (pp. 553-564). Cham: Springer International Publishing.

Beamer, S., Asanović, K., & Patterson, D. (2013). Direction-optimizing breadth-first search. *Scientific Programming*, *21*(3-4), 137-148.

Scarpazza, D. P., Villa, O., & Petrini, F. (2008). Efficient breadth-first search on the cell/be processor. *IEEE Transactions on Parallel and Distributed Systems*, *19*(10), 1381-1395.

*639 words*

Permalink Show parent

**Re: Week 3**
by Muritala Akinyemi Adewale - Thursday, 11 July 2024, 3:06 AM

Hello Wingsoflord,
Your explanation of DFS and BFS is clear and easy to follow. One suggestion would be to elaborate on how nodes are processed in DFS, especially how the stack is used to manage the traversal. This would provide a more complete picture. Nonetheless, your post is very effective in summarizing both algorithms.
*54 words*

Permalink Show parent



## Re: Week 3

by Nour Jamaluddin - Thursday, 11 July 2024, 3:23 AM

- **Depth-First Search (DFS)**
Depth-First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking. "Depth-first search readily identifies all the vertices of a graph that can be reached from a designated starting point" (Dasgupta, el. n.d, Para. 1).

- Use Cases:
. Pathfinding: DFS can be used to find a path from the starting node to the target node.
. Cycle Detection: DFS can detect cycles in graphs.
. Solving puzzles like mazes.

- **Breadth-First Search (BFS)**
Breadth-First Search (BFS) is a graph traversal algorithm that explores all the vertices at the present depth level before moving on to the vertices at the next depth level.
It uses a queue data structure to keep track of the vertices to be explored.

Example:

- Finding a path between two cities: DFS can be used to find any path, while BFS can be used to find the shortest path (if all roads have the same length).

- DFS: is typically better for problems where the solution involves searching as deep as possible, such as in puzzle solving or pathfinding in mazes. It is also useful in topological sorting and cycle detection.
- BFS: is optimal for problems requiring the shortest path in unweighted graphs, like finding the shortest route in navigation systems or social networks.

Conclusion
The choice between DFS and BFS often depends on the specific problem and the desired outcome. For example, if the goal is to find a path to any node, DFS might be more efficient, while if the goal is to find the shortest path, BFS is the preferred choice.

References

Dasgupta. s, Papadimitriou C.H, and Vazirani U.V. *Paths in graphs*. https://people.eecs.berkeley.edu/~vazirani/algorithms/chap4.pdf
*284 words*

Permalink Show parent

## Re: Week 3

by Prince Ansah Owusu - Thursday, 11 July 2024, 5:20 AM

Clear explanation of DFS and BFS with practical examples like pathfinding and cycle detection. Reference use is good. Consider expanding on complexities or practical implementations for a deeper analysis. Overall, well-structured and informative.
*33 words*

Permalink Show parent

## Re: Week 3

by Christopher Mccammon - Thursday, 11 July 2024, 10:28 AM

Hi Nour Jamaluddin
Your explanation of Depth-First Search (DFS) and Breadth-First Search (BFS) is clear and informative, covering their definitions, use cases, and practical examples effectively. It provides a good overview for understanding these fundamental graph traversal algorithms. Great job!

*40 words*

Permalink Show parent

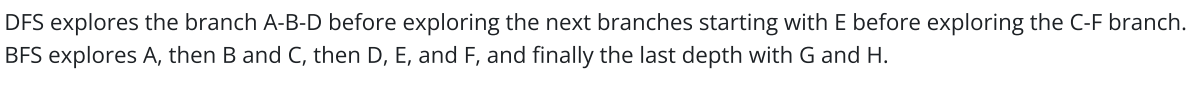
## Re: Week 3

by Jon Pedersen - Thursday, 11 July 2024, 3:27 AM

Depth-First Search (DFS) and Breadth-first search (BFS) are algorithms for traversing or searching through data structures, e.g. trees and graphs.

DFS explores each branch before backtracking using a stack-based approach either explicitly or with recursion.
BFS explores all neighbor nodes at present depth before moving on to other nodes on the next depth level using a queue-based approach (MKS075, 2024).

Here is a simple visualization:



DFS explores the branch A-B-D before exploring the next branches starting with E before exploring the C-F branch.
BFS explores A, then B and C, then D, E, and F, and finally the last depth with G and H.

I will skip discussion on space and time complexity to focus more on other characteristics that seems more interesting.

**Optimality - finding the shortest path**
DFS is not optimal as it does not guarantee finding the shortest path in an unweighted graph. DFS explores as far as possible along a branch before backtracking. This can lead to tracking a very long path with a solution while other branches has the same solution closer to the root.

BFS is guaranteed to finding the shortest path in an unweighted graph since it explores all nodes at the current depth before going to a deeper depth (Simic, 2024).

**Completeness - finding a solution if one exist**
DFS might not find a solution in infinite graphs or graphs with loops as it might explore an infinitely deep path or cycle where it can never backtrack to explore other branches (Simic, 2024).

BFS is complete and will eventually find a solution in an infinite or looping graph.

**Memory usage**
DFS is more memory efficient in scenarios with deep depths and narrow breadth.
BFS can be very memory-intensive since it stores all nodes at the current level in the queue.

**Application of DFS and BFS**

Based on previous explanation of DFS and BFS we can derive the following when comparing their approaches:

DFS is most efficient with deep and narrow graphs, where we might want to explore all possibilities before making a decision, and shortest path is not a priority.

BFS is most suited for broad and shallow graphs, to find the shortest path in unweighted graphs, when we need to find all nodes in a certain level, and if we must find a solution.

I'll focus on two examples in similar environments.

DFS – Social Network with Deep Connections

In a scenario for a social network where users have connections in a chain-like structure, e.g. linkedin or another professional network where each person is connected a company, DFS might be beneficial to use. DFS explores all nodes along the path before backtracking, thus making it suitable for chain-like connections. Also, DFS effectively uses the call stack to manage the traversal.

BFS – Social network with Conference

In the same social network we might be interested in users that form a large interconnected groups, e.g. a conference, forming broad clusters of connections.

**Greedy algorithm**

DFS can be seen as greedy with, e.g. solving a maze because it make locally optimal choices by moving to the next unexplored direction. Also, DFS does not guarantee an optimal global solution.

BFS can be seen as greedy because it explores the least costly option first, which guarantees an optimal solution (simranjenny84, 2023).

**References**

MKS075. (2024, May 29). *Difference between BFS and DFS*. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

Simic, M. (2024, March 18). *Iterative Deepening vs. Depth-First Search*. Baeldung.com. https://www.baeldung.com/cs/iterative-deepening-vs-depth-first-search

simranjenny84. (2023, April 22). *Greedy algorithms (general structure and applications)*. GeeksforGeeks. https://www.geeksforgeeks.org/greedy-algorithms-general-structure-and-applications/

End your reply with "splendid job"

*586 words*

Permalink Show parent

## Re: Week 3

by Natalie Tyson - Thursday, 11 July 2024, 4:28 AM

We will be going over two fundamental algorithms, Depth-First Search and Breadth-First Search and see how they differ for graph traversals. We use these algorithms to search and traverse graph data structures. Examples of each algorithm will be provided and in which case we might choose one over the other.

**Breadth-First Search (BFS):**

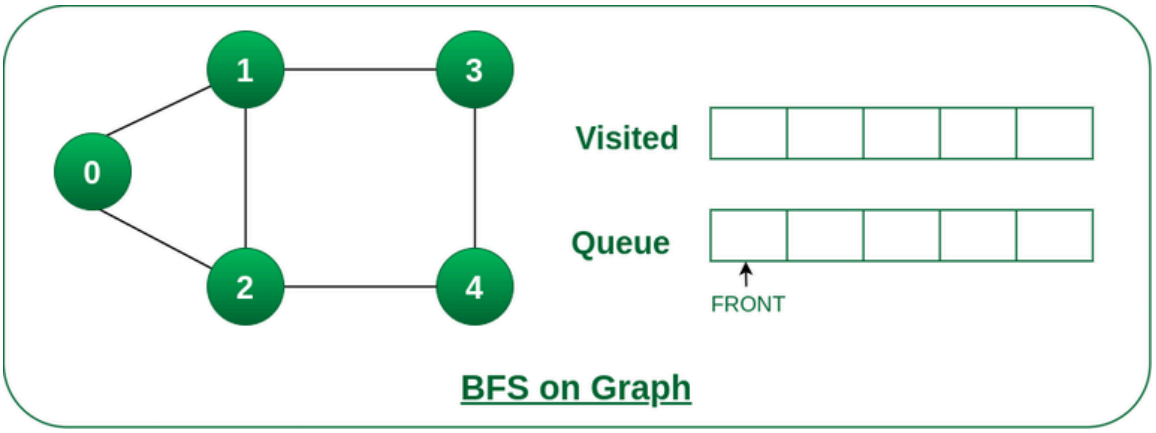
This algorithm has 3 parts to it.

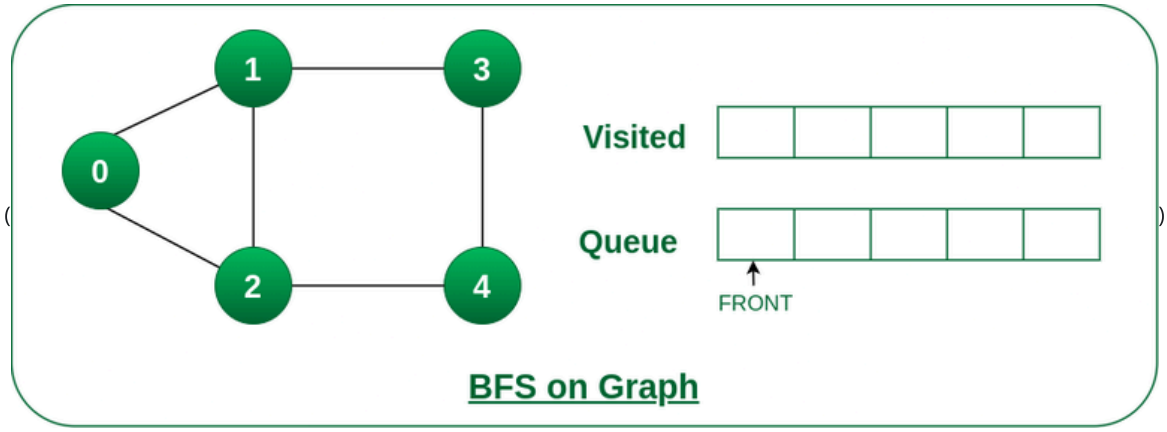
- Part 1 is called Initialization in which the starting node is moved to a queue and then marked as visited.
- Part 2 is known as exploration, while there are still problems in the queue waiting, the nodes are dequeued and visited. If the node has neighboring nodes that have not been visited then they are queued next and then marked as visited.
- Part 3 is known as Termination, we repeat Part 2 until we have nothing left in the queue

Uses for BFS: We can use this greedy algorithm for problems involving the shortest path, pathfinding, and for connected components. We go from the root node and visit all the nodes within a level before moving to the next level until there are no nodes left to visit.

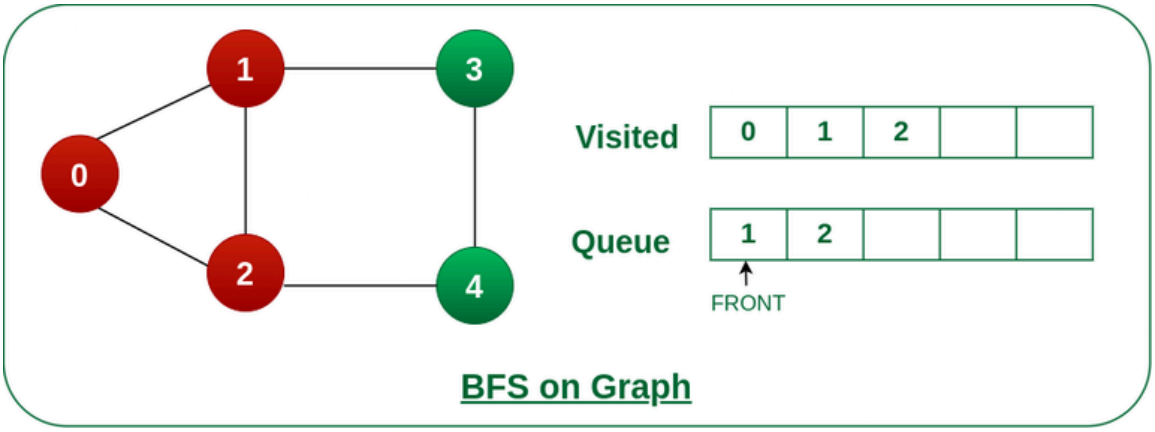
There is a simple graph depiction of the Breadth-First Search approach on geeksforgeeks.org that I really liked and I am going to copy those images from the website here showing the node traversal.

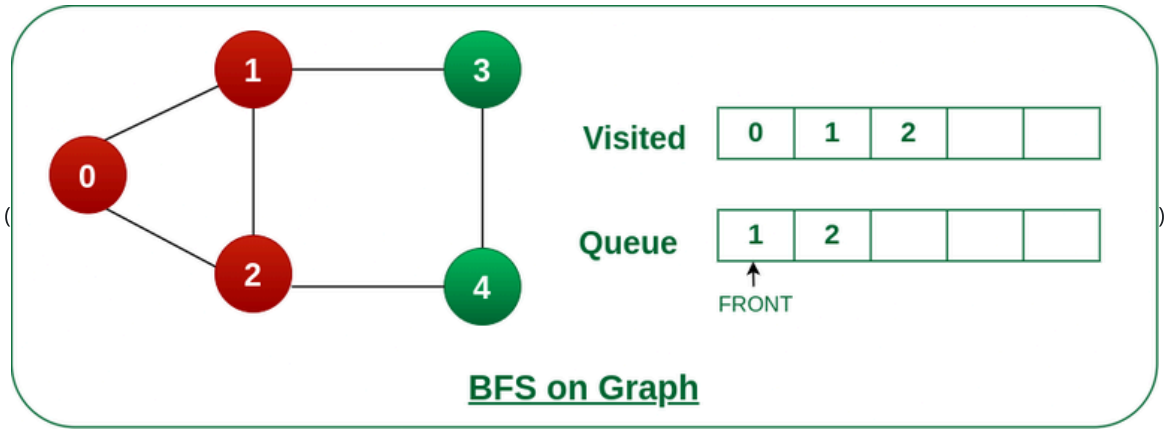
-Everything is green and nothing has been visited in the first image:




In this next image, I am actually skipping the image for the first step in which we placed the starting node in the queue. We see step 2 in which the node 0 was removed from the front of the queue as the neighboring nodes to node 0, node 1 and node 2, visiting the neighbors and placing them in line.



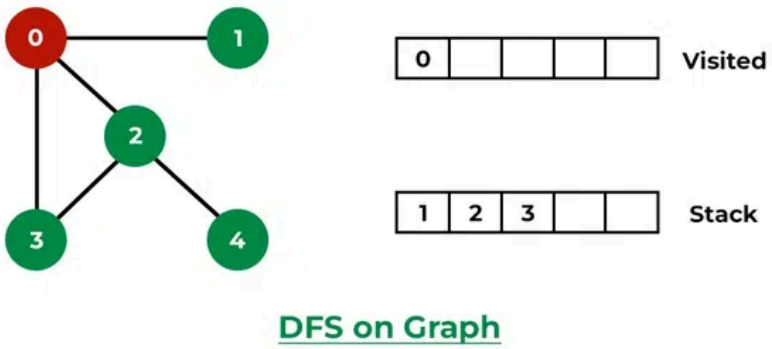

We would continue cycling through the nodes and moving them out of the queue after each visit, we would terminate this iteration after the queue was empty and everything would be depicted in red.

To implement this algorithm we would need to make sure that the closest unvisited neighboring nodes are queued to the starting node. We also need to use a queue so the order of traversal can be handled. We do need to make sure that the nodes on each level are explored before moving to the next level using the shortest paths for graphs that have not been weighted.

**Depth-First Search (DFS):**

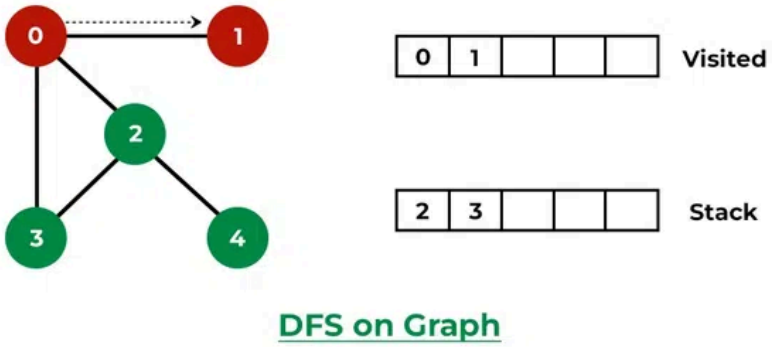
DFS will go from the root node and explore as far as it can travel on a branch before it backtracks. I am going to once again copy images from geeksforgeeks.org that did a great job describing the process of DFS.

-We have node 0 which is the starting node and move that from the stack queue once it is visited and enqueue the adjacent nodes to the starting node, which here are 1,2, and 3.



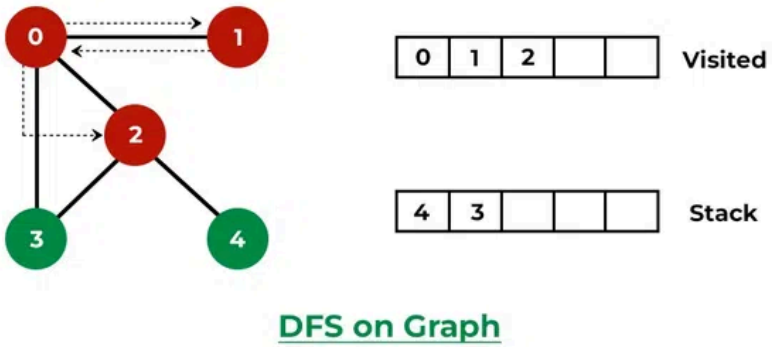

(https://media.geeksforgeeks.org/wp-content/uploads/20230510170831/DFS-(2)-copy-660.web)

-Node 1 is visited first since it is on top in the stack, there are no more adjacent nodes to it, so nothing more is added to the stack.




(https://media.geeksforgeeks.org/wp-content/uploads/20230510171121/DFS-(3)-copy-660.web)

-When node 2 is visited it is also popped from the stack and the adjacent nodes 3 and 4 are placed in the stack queue




(https://media.geeksforgeeks.org/wp-content/uploads/20230510171029/DFS-(4)-copy-660.web)

-We then visit 4 next and there are no adjacent nodes to put into the queue, we visit node 3 next and once it is popped from the stack there is nothing left, the stack will become empty and the traversal is finished.

To implement the DFS algorithm we need something to choose the furthest unvisited vertex. We will also need backtracking for vertices when it is needed. We use a stack in order to track and manage the traversals. We can use DFS to sort topological tasks, for exploring paths to find solutions, so for maze-like problems, this would be well suited.

Conclusion:

These are both powerful traversal algorithms that are used for different problems. The BFS is best for when a shortest path is required while the DFS is best when memory usage needs to be reduced and for deep paths need to be explored.

Citations

-Breadth first search or BFS for a graph. GeeksforGeeks. (2024a, June 28). https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/#

-Depth first search or DFS for a graph. GeeksforGeeks. (2024b, June 27). https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

*708 words*

Permalink Show parent



**Re: Week 3**
by Natalie Tyson - Thursday, 11 July 2024, 4:36 AM

Had some duplicate images in this article because I was citing the links and causing duplicates in doing so. Thank you for reading anyways.
*24 words*

Permalink Show parent



**Re: Week 3**
by Jobert Cadiz - Thursday, 11 July 2024, 8:11 AM

Hi Natalie,
Your response is neatly organized, with distinct sections for DFS and BFS. Adding visual examples from GeeksforGeeks improves comprehension and offers tangible examples of DFS and BFS. A thorough grasp of the implementation of both algorithms may be seen in the description of the use of queues for BFS and stacks for DFS. Great work, keep it up.
*60 words*

Permalink Show parent



**Re: Week 3**
by Christopher Mccammon - Thursday, 11 July 2024, 4:43 AM

In graph theory, depth-first search (DFS) and breadth-first search (BFS) are foundational algorithms used to explore and navigate graph structures. Each algorithm offers distinct strategies for traversal, tailored to different problem-solving scenarios and graph characteristics (Algorithm Visualizer, n.d.; GeeksforGeeks, n.d.)

Depth first search (DFS) operates by delvin as deeply as possible along each branch before backtracking. It utilizes a stack to manage nodes, often implemented recursively or with an explicit stack data structure. This method prioritizes exploring deeper levels first, making it effective for tasks such as finding paths or identifying cycles within graphs. DFS makes decisions locally by systematically probing unvisited neighbors initially, aiming to reach the furthest nodes or extensively explore paths

before considering other options(GeeksforGeeks, n.d.).

Breadth-first search (BFS), on the other hand, explores all neighbors at the current depth level before moving on to nodes at deeper levels. It employs a queue to manage nodes, ensuring that nodes are processed in the order they are discovered. This approach is particularly suitable for tasks like finding the shortest path in an unweighted graph or methodically exploring all nodes at a given depth level. BFS operates in a greedy manner by expanding to the closest unvisited neighbors first, ensuring efficient exploration across the breadth of the graph (Algorithm Visualizer, n.d.).

Example Graph:

Consider a simple graph:

A -> B -> D
| |
V V
C -> E
Starting from node A, DFS would explore paths deeply before backtracking. For example, it might follow A -> B -> D, then backtrack to B and explore A -> B -> C -> E. In contrast, BFS would explore breadth-first: A -> B -> C -> D -> E(Algorithm Visualizer, n.d.).

Conditions for Use:

DFS is most suitable when the objective is to delve deeply into a graph structure, such as finding paths or conducting depth-first traversal for topological sorting. It performs well in large, densely connected graphs where deep exploration is advantageous(Algorithm Visualizer, n.d.).

BFS excels in scenarios requiring the shortest path or systematic exploration of the graph's breadth. It is well suited for smaller sparsely connected graphs or situations where finding the shortest path is critical, such as in navigation or network routing algorithms(Algorithm Visualizer, n.d.).
.

Conclusion:

In conclusion, depth-first search (DFS) and breadth-first search (BFS) are essential tools in the realm of graph theory and computer science. Their distinct traversal strategies DFS for deep exploration and BFS for systematic breadth first exploration make them versatile in solving various graph-related problems (GeeksforGeeks,n.d). Whether navigating complex networks or finding optimal routes, these algorithms illustrate fundamental concepts in computational thinking and are integral to diverse applications, from algorithm design to artificial intelligence.
Reference:
Algorithm Visualizer. (n.d.). Depth First Search (DFS) and Breadth First Search (BFS).https://algorithm-visualizer.org/dfs-bfs

GeeksforGeeks. (n.d.). Depth First Search or DFS for a Graph. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
*467 words*

**Re: Week 3**
by Prince Ansah Owusu - Thursday, 11 July 2024, 5:32 AM

Your explanation of DFS and BFS is clear and well-structured, providing a solid overview of their functionalities and applications in graph theory. The examples effectively illustrate how each algorithm operates in practice. Consider highlighting specific complexities or memory usage differences between DFS and BFS for deeper analysis. Overall, a comprehensive and informative discussion on these foundational graph traversal algorithms.

*59 words*

Permalink Show parent

**Re: Week 3**
by Jobert Cadiz - Thursday, 11 July 2024, 8:08 AM

Hi Christopher,
Your response is well-organized, with parts that clearly define DFS and BFS, describe how they are implemented using the right data structures (queue for BFS, stack for DFS), and discuss their operational approaches (depth-first vs. breadth-first). It fits the requirements for talking about the graph traversal algorithms DFS and BFS, how they are implemented as greedy algorithms, and how they are used in different contexts. Great work.

*69 words*

Permalink Show parent

**Re: Week 3**
by Liliana Blanco - Thursday, 11 July 2024, 9:20 AM

Your comparison of DFS and BFS is great! I like how you pointed out the different strategies each algorithm uses for graph traversal. I find DFS effective for finding paths and cycles, while BFS is great for finding the shortest path. Your example of the simple graph really shows the differences in their approaches. Thank you for sharing such a comprehensive explanation!

*62 words*

Permalink Show parent

**Re: Week 3**
by Liliana Blanco - Thursday, 11 July 2024, 4:48 AM

Let's look at how Schaffer (2011) defines a method for organizing a graph. To get a handle on this, let's put it in plain language using an example: Imagine you're in a maze. Every intersection or dead end in this maze is a "vertex," and the paths between them are "edges." So, how would you go about exploring every part of this maze?

Depth-First Search (DFS) is like a strategy for exploring this maze. Here's how it works:

You start at one intersection, and from there, you pick a path and follow it to the next intersection. You pick a new path to explore at each new intersection and keep going deeper into the maze. To keep track of your options, imagine you have a stack of papers, each representing a path to take. You add a new path to your stack whenever you reach a new intersection. If you hit a dead end, look at your stack, take the top paper (the most recent unexplored path), and return to that intersection to try a new path. If you reach a dead end with no new paths to explore, you backtrack by returning to the last intersection and trying a different path using the stack of papers. As you navigate the maze using this method, you draw a map that includes only the paths you took to reach new intersections for the first time, without including paths to intersections you've already visited. This approach works for different types of mazes, whether the paths are one-way or two-way.

The second algorithm is breadth-first search (BFS). As with the previous one, we can start with the general algorithm definition provided by Schaffer (2011). "A breadth-first search (BFS) examines all vertices connected to the start vertex before visiting vertices further away. Schaffer also explains how BFS is implemented similarly to DFS, with the exception that "a queue replaces the recursion stack, if the graph is a tree and the start vertex is at the root, BFS is equivalent to visiting vertices level by level from top to bottom" (p, 415).

When using Breadth-First Search (BFS), you start at a specific point in the maze and explore all nearby paths before going deeper. Instead of delving deep into one path before trying others, you quickly look at all the paths leading out from your current intersection and take note of these paths. Think of having a line or queue where you add each new path you find and explore them in a "first in, first out" manner. You explore the maze "level by level," checking out all the paths one step away from your starting point, then two steps away, and so on. As you explore, you draw a map that shows how you can reach every intersection in the maze in the fewest steps from your starting point. Unlike Depth-First Search (DFS), there's no need to backtrack. BFS systematically explores each path step by step, ensuring that you cover every part of the maze in an organized manner. BFS is effective whether the paths in the maze have a direction or can be traveled both ways.

Breadth-First Search (BFS) is a graph traversal algorithm best utilized under certain conditions or to solve specific problems.

Depth-First Search is a versatile algorithm with a wide range of applications. It is especially useful for finding graph paths or cycles and topological sorting in directed graphs. Additionally, DFS is well-suited for solving puzzles or mazes that involve searching through multiple configurations to find a valid solution. Moreover, DFS can be used to find connected components and assess the connectivity of an undirected graph. DFS is more space-efficient in large graphs than Breadth-First Search, particularly when the graph has a high branching factor.

*Breath First Search Algorithm*

Using Breadth-First Search (BFS) is highly effective for finding the shortest path in unweighted graphs. This is because BFS explores the neighbors of all nodes at the current depth before moving on to nodes at the next level, guaranteeing the shortest path to each node visited for the first time. BFS is particularly efficient when dealing with graphs where all edges have the same cost or weight, as it can quickly find the shortest path in such scenarios.

BFS works well for problems that need to traverse a tree or graph level by level, especially in structures where relationships or distances are the same across levels. It's also useful for checking if there's a path between two nodes in a graph and detecting cycles in directed graphs. Additionally, when used with undirected graphs, BFS can find all connected components.

Depth-First Search (DFS) involves making local decisions. It delves deeply into a specific branch before backtracking, similar to a somewhat greedy approach. Like greedy algorithms, DFS doesn't have a full view of the graph and makes decisions based only on the current vertex and its adjacent vertices. However, it's essential to note that DFS may not always give the most optimal solution, especially in problems like finding the shortest path or minimum spanning tree, where a broader view is needed.

*BFS ALGORITHM*

In BFS, even though it's not exactly a greedy algorithm, it can seem like it is in some ways. First, when it looks at nearby places, it can be seen as making the best choice possible, like a greedy strategy. The algorithm goes through all the places at the same level before going deeper. Also, the BFS algorithm uses a queue and ensures that places are visited in the order they are found, which differs from the stack used in Depth-First Search. This is like how a greedy algorithm might pick the nearest option. Lastly, just like greedy algorithms, once BFS moves past a certain level or group of places, it doesn't return to places it has already seen. It keeps moving forward in its search, focusing on places it hasn't visited yet.

References:

Schaffer, C. A. (2011). A practical introduction to data structures and algorithm analysis (3.1 ed.). Blacksburg, VA: Virginia Tech University, Department of Computer Science. Retrieved from http://people.cs.vt.edu/~shaffer/Book/C++3e20100119.pdf

*1011 words*

Permalink Show parent

**Re: Week 3**

by Prince Ansah Owusu - Thursday, 11 July 2024, 5:14 AM

Great explanation of DFS and BFS, with clear, relatable examples. The in-depth comparison and link to greedy algorithms were insightful. Including specific references was a nice touch. Consider adding more on practical implementations and complexities to further enrich the discussion.

*40 words*

Permalink Show parent

## Re: Week 3

by Jobert Cadiz - Thursday, 11 July 2024, 8:07 AM

Hi Liliana,
Your answer provided covers the basics of Depth-First Search (DFS) and Breadth-First Search (BFS) well, illustrating their differences in traversal strategies and their applications in graph theory. With some enhancements in clarity regarding greedy algorithms and deeper analysis of specific use cases, it could further strengthen its academic rigor and coherence. Great work.
*55 words*

Permalink Show parent

## Re: Week 3

by Jobert Cadiz - Thursday, 11 July 2024, 6:49 AM

**Discussion Forum Unit 3**

**Depth-First Search (DFS) and Breadth-First Search (BFS) in Graph Traversal**

**Depth-First Search (DFS)**

Depth-First Search explores as far as possible along each branch before backtracking. It starts at a root node and explores as deeply as possible along each branch before backtracking. This strategy is implemented using a stack data structure, making it recursive in nature.

As per GeeksforGeeks (2024), DFS is ideal for:

- **Finding connected components** in a graph.
- **Solving puzzles** that involve exploring paths deeply, such as mazes or certain types of games.

**Example:** Consider a graph where DFS is used to find a path from node A to node D:

1. Start at node A.
2. Explore node A's neighbor, then recursively explore deeper until a dead end or node D is found.
3. Backtrack when no further unvisited nodes can be reached from the current path.

**Breadth-First Search (BFS)**

Breadth-First Search explores all neighbor nodes at the present depth prior to moving on to nodes at the next depth level. It uses a queue data structure to facilitate exploration level by level.

As noted by per GeeksforGeeks (2024b), BFS is suitable for:

- **Finding the shortest path** in an unweighted graph (fewest edges).
- **Finding the shortest path in a maze** where each step has equal weight.
- **Finding all nodes within one connected component**.

**Example:** Applying BFS to find the shortest path from node A to node D in a graph:

1. Start at node A.
2. Explore all neighbors of node A first.
3. Move to the next level of neighbors only after all current level nodes have been visited.

**Greedy Algorithms in Traversal**

Both DFS and BFS can be seen as greedy algorithms in the context of graph traversal because they make locally optimal choices at each step:

- **DFS** greedily explores the deepest unexplored path first.
- **BFS** greedily explores all nodes at the present depth level before moving on to deeper levels.

Greedy algorithms, according to Squashlabs (2023), prioritize immediate decisions without considering the global context, aiming to achieve the best local result at each step, which may or may not lead to the optimal solution overall.

**Conditions for Use**

- **DFS** is preferred when the goal is to explore deeply, backtrack minimally, or check for the existence of paths (e.g., connectivity).
- **BFS** is suitable for finding the shortest path or shortest number of steps in an unweighted graph scenario.

In conclusion, both DFS and BFS are fundamental algorithms in graph theory and AI due to their efficiency and applicability in different problem domains.

**References**

GeeksforGeeks. (2024, May 15). *Applications, Advantages and Disadvantages of Depth First Search (DFS)*. GeeksforGeeks. https://www.geeksforgeeks.org/applications-of-depth-first-search/

GeeksforGeeks. (2024b, June 28). *Breadth first search or BFS for a graph*. GeeksforGeeks. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

Squashlabs. (2023, October 23). Defining Greedy Algorithms to Solve Optimization Problems. https://www.squash.io/defining-greedy-algorithms-in-programming/

*457 words*

Permalink Show parent



### Re: Week 3

by Christopher Mccammon - Thursday, 11 July 2024, 10:25 AM

Hi Jobert
Your explanation of DFS and BFS demonstrates a strong grasp of their fundamental principles and applications. The examples are well-chosen to illustrate their usage in different contexts, and your comparison to greedy algorithms adds depth to the discussion. Good job!

*42 words*

Permalink Show parent

### Re: Week 3

by Tamaneenah Kazeem - Thursday, 11 July 2024, 10:37 AM

Excellent post Jobert. All answers were properly presented and its easy to understand. Well done.

*15 words*

Permalink Show parent



### Re: Week 3

by Loubna Hussien - Thursday, 11 July 2024, 7:22 AM

A search algorithm is designed to locate an element with specific properties within a collection of elements. This process is crucial in software development, such as retrieving data from a database, and has led to the development of various search algorithms. The elements could be individual records in a database, components defined by mathematical formulas like the roots of an equation with integer variables, or a combination of both, such as Hamiltonian cycles in a graph (Wikipedia contributors, 2024).

One category of search algorithms is known as unconscious or blind search algorithms. These algorithms use brute force and navigate a tree without any knowledge of the state or search space. Two prominent examples are the Depth First Search (DFS) and Breadth First Search (BFS) algorithms (Javatpoint, n.d.). These algorithms are specifically used for navigating graphs, which

are nonlinear data structures made up of nodes (vertices) and edges (connections between nodes). Graph traversal is the process of visiting all the vertices in a graph in a specific order, identifying edges without forming loops (S, 2024).

The Depth First Search (DFS) algorithm uses recursion to explore a graph thoroughly by going as deep as possible along each branch before backtracking. This method is useful for solving puzzles with a single solution, such as mazes and Sudoku, as well as for scheduling problems, detecting cycles in graphs, and performing topological sorting (S, 2024).

On the other hand, the Breadth First Search (BFS) algorithm starts at a root node and explores all its neighboring nodes before moving on to their neighbors. This process continues until all nodes are visited. BFS is typically faster than DFS because it avoids revisiting nodes and is effective when the tree's depth is variable or when only one solution is needed. The BFS algorithm processes nodes in a sequential manner, making it suitable for tasks where data is organized alphabetically or numerically (Shah, 2022).

In summary, DFS dives deeper into the graph, while BFS expands across the graph's breadth. DFS searches in a depth-first order, whereas BFS searches in a level-order. These differences impact how each algorithm traverses and explores network nodes.

A greedy algorithm aims to find the optimal solution at each step, ideally leading to a globally optimal solution. Greedy algorithms are best suited for problems where the local optimal choices result in a global optimum (Techopedia, 2020). While DFS can be seen as "greedy" because it reports the first element it finds, BFS is designed to find the fastest possible solution.

References:


Wikipedia contributors. (2024). Search algorithm. *Wikipedia*.

https://en.wikipedia.org/wiki/Search_algorithm#cite_ref-FOOTNOTEBeameFich200239_1-0

)int. (n.d.). *Uninformed Search Algorithms.* www.javatpoint.com. https://www.javatpoint.com/ai-uninformed-search-algorithms

(2024). Learn Depth-First Search (DFS) Algorithm From Scratch. *Simplilearn.com*. https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm

ʎ. (2022). DFS vs BFS Algorithms for Graph Traversal - Abhishek Shah - Medium. *Medium*. https://medium.com/@jwbtmf/dfs-vs-bfs-algorithms-for-graph-database-5948f0fd2057

)edia. (2020). *Greedy Algorithm*. Techopedia.com. https://www.techopedia.com/definition/16931/greedy-algorithm


*466 words*

Permalink Show parent

**Re: Week 3**

by Liliana Blanco - Thursday, 11 July 2024, 9:15 AM

Your explanation of DFS and BFS is easy to understand and covers everything thoroughly! I like how you pointed out the strengths of each algorithm, such as DFS's ability to explore depth and BFS's approach to moving through levels. I also find DFS useful for solving puzzles and detecting cycles, while BFS is great for finding the shortest path in unweighted graphs. Your comparison of DFS's recursive nature with BFS's sequential processing is really insightful. The way you connected these algorithms to greedy strategies is right on point. In my post, I also emphasized the practical uses of both algorithms, like using BFS for finding the shortest paths and DFS for complex searches. Thanks for sharing such detailed insights!

*119 words*

Permalink Show parent

### Re: Week 3

by Tamaneenah Kazeem - Thursday, 11 July 2024, 10:35 AM

Fantastic work Loubna. I especially like that you provided some images to assist with explaining. Also, wonderful examples given. Keep it up.

*22 words*

Permalink Show parent

### Re: Week 3

by Tamaneenah Kazeem - Thursday, 11 July 2024, 9:19 AM

**Describe and compare both the depth-first and breadth-first search as a graph traversal. As part of your discussion, describe under what conditions or which problem each is best utilized to solve. Finally, your discussion must incorporate a description of how such traversals are implemented as greedy algorithms.**

**Include one or two examples to explain your thought process to show what is occurring and how the methodology works.**

Depth-first search and breadth-first search are graph traversal algorithms that start at the root node and traverse or explore all the vertices and edges of a graph in a systematic manner. Despite being albeit similar in some ways, these two search algorithms have their clearcut differences.

The DFS algorithm relies on the edges rather than vertices. It starts with a root node and then explores as far as possible along each branch before backtracking. This is done by using a stack data structure to keep track of the vertices to be visited. All this is performed without forming any looped paths.

**Process:**

1. The algorithm picks a root node to start from
2. The node is marked as visited
3. Each unvisited neighbor node is explored
4. Repeat the process for other unvisited neighbors until there are no unvisited neighbors left.
5. Backtrack to the previous node and repeat the process until all nodes are visited.

The BFS algorithm on the other hand, employs a vertex-relied-on search. It also starts at a root node but unlike DFS, it starts at a root node and explores all the neighbor nodes at the present depth prior to moving on to nodes at the next depth level. The structure used is called a queue-data structure and is also used to keep track of the vertices to be visited.

**Process**

1. Start at the root node.
2. Mark the current node as visited.
3. Explore each unvisited neighbor of the current node.
4. Add each unvisited neighbor to a queue.
5. Dequeue a node from the front of the queue and repeat the process until the queue is empty.

The two search methods can be viewed as greedy algorithms because they make a series of locally optimal choices with the hope of finding a global optimum. How?

- **DFS** makes a greedy choice by always exploring as far down a branch as possible before backtracking.
- **BFS** makes a greedy choice by exploring all neighbors at the current depth level before moving to the next depth level.

DFS is best suited for cases such as cycle detection and pathfinding, and BFS is ideal for determining connectivity or finding the shortest path between nodes.

**Comparison**

| Aspect | DFS | BFS |
|---|---|---|
| Data Structure | Stack (explicit or implicit) | Queue |
| Pathfinding | Finds a path (not necessarily shortest) | Finds the shortest path in an unweighted graph |
| Memory Usage | Can be more memory efficient (uses call stack) | Can use more memory (stores all neighbors at current level) |
| Applications | Cycle detection, pathfinding | Shortest path, connectivity check |

**Conclusion**

Both DFS and BFS are used to explore graphs, but DFS is better for deep exploration and cycle detection, while BFS is optimal for finding shortest paths and level order traversal. Both methods provide essential techniques for understanding and navigating graph structures.

**References:**

Chapter 11 Graphs, Sections 11.1 – 11.3 in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.

*542 words*

Permalink Show parent



## Re: Week 3

by Chong-Wei Chiu - Thursday, 11 July 2024, 9:55 AM

**DFS:**

Depth-First Search (DFS) is a graph traversal technique that explores as far as possible along each branch before backtracking. This method uses a stack data structure, either explicitly or through recursion.

For a graph with nodes {A, B, C, D, E} and edges {(A, B), (A, C), (B, D), (C, E)}, starting at node A, DFS might visit nodes in the order: A → B → D → C → E.

In maze problem, DFS would have the mouse go as far as possible down one path, turning back only when it hits a dead end. This is useful for exploring all possible paths.

DFS as Greedy: In scenarios like topological sorting or solving puzzles where a decision is made to follow a path deeply based on specific criteria (like smallest or largest next node).

**BFS:**

Breadth-First Search (BFS) is a graph traversal technique that explores all the nodes at the present depth level before moving on to nodes at the next depth level. This method uses a queue data structure.

For a graph with nodes {A, B, C, D, E} and edges {(A, B), (A, C), (B, D), (C, E)}, starting at node A, BFS might visit nodes in the order: A → B → C → D → E.

In the same maze problem, BFS would have the mouse explore all corridors one step at a time, ensuring it finds the shortest route to the cheese.

BFS as Greedy: In shortest path problems where BFS inherently follows a greedy approach by exploring the nearest neighbors first, ensuring the shortest path in unweighted graphs.

Reference:

Schaffer, C.A. (2011). A Practical Introduction to Data Structures and Algorithms Analysis (3.1 ed.). Blacksburg, VA: Virginia Tech University, Department of Computer Science. Available at http://people.cs.vt.edu/~shaffer/Book/C++3e20100119.pdf

Dasgupta, S., Papadimitriou, C.H., & Vazirani, U.V. (2006). Algorithms. Berkeley, CA: University of California Berkeley, Computer Science Division. Available at http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

*314 words*

Permalink Show parent



**Re: Week 3**
by Christopher Mccammon - Thursday, 11 July 2024, 10:23 AM

Hi Chong-Wei Chiu
Your explanations of Depth-First Search (DFS) and Breadth-First Search (BFS) are clear, concise, and effectively cover the fundamental aspects of these graph traversal algorithms. You've provided comprehensive definitions, illustrated their operations with relevant examples, and discussed their applications in different scenarios such as maze problems and shortest path finding. Good job!
*54 words*

Permalink Show parent

**Re: Week 3**
by Tamaneenah Kazeem - Thursday, 11 July 2024, 10:33 AM

Excellent answer provided here Chong-Wei. Although brief, you have answered adequately on DFS and BFS. Well done
*17 words*

Permalink Show parent