Fardeen Ahmed
Cuny Id: 23826620
CSC 44800
Email: fahmed014@citymail.cuny.edu

Abstract:
Artificial intelligence is a newer and more interesting field in technology right now. We are seeing things invented every day, and these inventions are changing our surroundings even more. Artificial intelligence is a complex field, and to understand this field, we need to learn more about the theoretical and practical usage of computer science. Algorithms are an essential part of computer science. An algorithm is a process or set of rules to be followed to solve problems, usually by a computer. In the field of Artificial intelligence, there are many algorithms we can learn. These algorithms help us better the field of artificial intelligence. In this paper, I am going to talk about one such section of the algorithm known as the Blind Search algorithm.

Blind Search, also known as uninformed search, is a form of algorithm that explores a problem without knowing any specific knowledge about the problem. It is a brute-force way of solving a problem. Even though these searches sound ineffective, these algorithms are used a lot in computer science. This paper will cover two of the most used blind search algorithms, Depth-First Search and Depth-First Search. We will discuss what they are, how to implement them, pseudocode, examples, and advantages and disadvantages in this paper.

Depth-First Search(DFS):

Depth-first search is considered one of the most famous blind search algorithms. This is useful for traversing or searching tree or graph data structures. Depth-First Search starts at a node, often a root node of a tree data structure, and explores each branch as far as possible before it backtracks. This algorithm is often introduced early to computer science and engineering students due to its simplicity and effectiveness. It provides a foundational understanding of graph traversal and lays the groundwork for more advanced algorithms.

How does it work:

We will consider a tree data structure and explain how it works below:

1) We will start with the root of the tree and put it on top of the stack.
2) After that, we take the top item of the stack and add it to the visited list of the vertex.
3) For our next step, we create a list of that adjacent node of the vertex and add the ones that aren't visited to the visited list of vertexes on the top of the stack.
4) At the end, we will keep repeating steps 2 and 3 until our stack is empty.

Pseudocode:

Every algorithm needs a pseudocode, to understand what we need to do before we start coding. Now, to implement the Depth-First Search, we can follow different approaches. We can follow a recursive implementation or a non-recursive implementation. This section will go over both of them.

Recursive Approach:

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

Figure 1: the above picture is the pseudocode for the recursive implementation of Depth-First Search.[1]

Non-Recursive Approach:

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

Figure 2: The Non-Recursive or Iterative approach.[1]

In the figures above, I have given two pseudocodes for the Depth-First Search algorithm. The first one(Figure 1) is taking a recursive approach, meaning the function will keep calling itself until it has reached the base case or the stopping condition.

The pseudocode in Figure 2, uses an iterative approach. Here, we are using a stack data structure. A stack data structure is a linear data structure that follows the LIFO order or Last in First Out order. In the next part of this paper, I am giving an example using the iterative Depth-First Search algorithm.

Example:

Now I will take an example to go over the Non-Recursive approach to the Depth-First Search Algorithm.

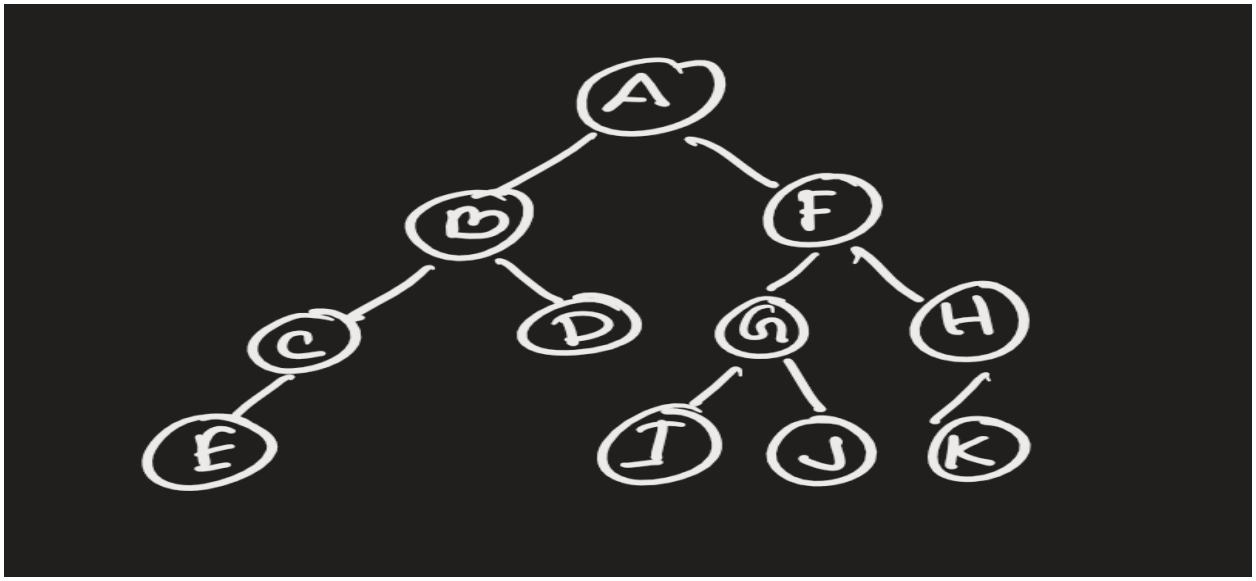

Figure 3: This image is a tree with 11 nodes, with A being the root node.

We are going to use the tree in Figure 3, as an example.

Iteration 1:

We take a list called visited and a stack. A stack is a data structure that follows the Last in First Out or LIFO method. We will start from root A and push it into the stack.

Visited =

Stack = A

The algorithm checks if the element in the stack is visited or not. So, we pop the element A from the stack, and add them to the list visited. After that, we push children B and F into the stack. After the first iteration, we have,

Visited = A

Stack = F B

Iteration 2:

Now we pop the last element of the stack, B, and insert it into the visited list. We take the children of B and push them into the stack. After iteration we have,

Visited = A B

Stack = F D C

Iteration 3:

Again we pop the last element of the stack, C, and insert it into the visited list. We push the children or in this case child E into the stack.

Visited = A B C

Stack = F D E

Iteration 4:

We pop E from the stack and insert it into the visited list. Notice that E doesn't have any child, so we don't push anything into the stack.

Visited = A B C E

Stack = F D

Iteration 5:

Now, we pop D and insert it into the stack. We don't push anything because D has no child.

Visited = A B C E D

Stack = F

Iteration 6:

Pop F, and insert its children G and H to the stack.

Visited = A B C E D F

Stack = H G

Iteration 7:

We pop G from the stack and insert it into the visited list. Push the children of G into the stack.

Visited = A B C E D F G

Stack = H J I

The rest of the iterations follow the same rules so after the final iterations we have,

Visited = A B C E D F G I J H K

Stack = empty

Advantages of DFS:

1) This algorithm requires a lot less time and space complexity than BFS.
2) The solutions can be found without doing much searching.

Disadvantages of DFS:

1) Not guaranteed that this algorithm will provide a solution.
2) The cut-off depth for this algorithm is smaller so time complexity is more.
3) As we go deeper into a tree, it takes more time to find the solution.

Applications:

1) It can be used to find the strongly connected components of the graph.
2) It can be used to see if the graph is bipartite.
3) It can detect cycles in a graph.
4) Often used for Network Analysis.
5) DFS can solve the puzzle with only one solution.
6) In Artificial Intelligence, we can develop neural networks and use them in deep learning.

Breath-First Search:

Breath-First Search is one of the popular blind search algorithms. It is also considered one of the most important algorithms in the field of artificial intelligence. BFS is used to traverse

a graph or a tree data structure. Like DFS it is taught to computer science and engineering students at the start of their academic careers.

How it works:

1) We start by putting the root of a tree at the front of the queue.
2) Then, we dequeue the node from the queue and add it to the visited list. We enqueue the children of the recently dequeued node into the queue.
3) We repeat steps 1 and 2 until the queue is empty.

Pseudocode:

```
procedure BFS(G, root) is
      let Q be a queue
      label root as explored
      Q.enqueue(root)
      while Q is not empty do
          v := Q.dequeue()
          if v is the goal then
              return v
          for all edges from v to w in G.adjacentEdges(v) do
             if w is not labeled as explored then
                 label w as explored
                 w.parent := v
                 Q.enqueue(w)
```

Figure 4: The pseudocode for Breath-First Search.[3]

In Figure 4, we can see the pseudocode for Breath-First Search. This code has similarities with the iterative approach of DFS. but there is a difference. Instead of using a stack, we use a queue data structure. The next part of this paper will give an example using the BFS algorithm.

Example:

\We are going to use the following tree in Figure 5 as an example.

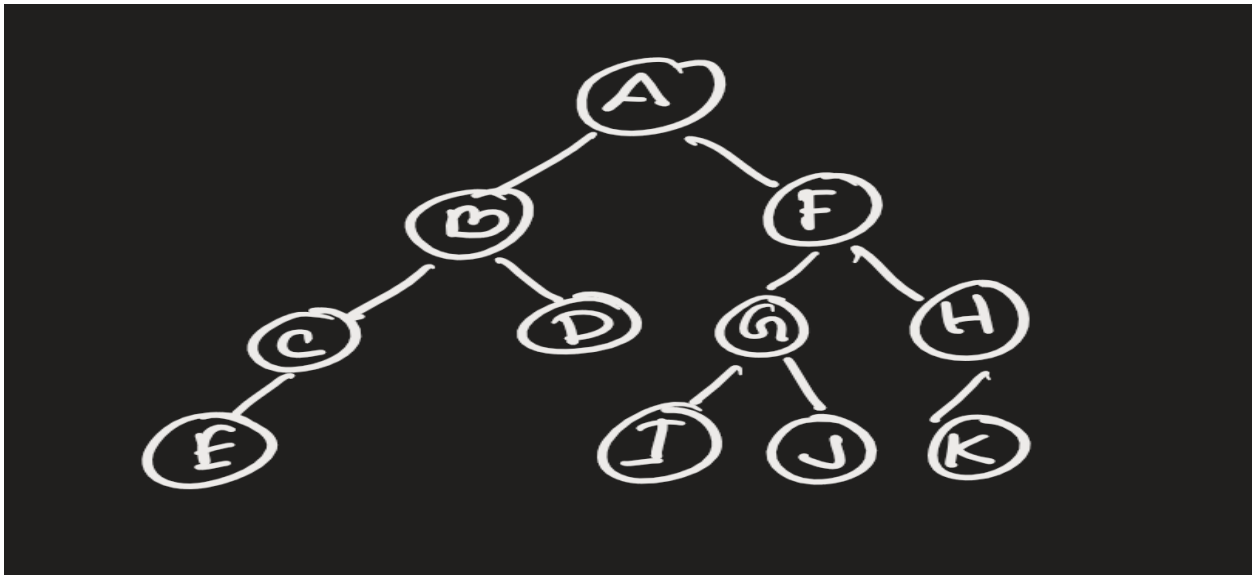

Figure 5: A tree for example.

Iteration 1:

We start from the root node A. Since we are using the queue data structure, we will enqueue A into the queue first. Then we will dequeue a from the queue and insert A into the visited list. Then we enqueue the children of A into the queue

Visited: A

Queue: B F

Iteration 2:

For the second iteration, we dequeue B from the queue since the queue uses the First In First Out, or FIFO method. We add B to the visited list and add the children of B, which are C and D into the queue.

Visited: A B

Queue: F C D

Iteration 3:

Next, we dequeue F from the queue and add it to the visited list. We enqueue the children of F into the queue.

Visited: A B F

Queue: C D G H

Iteration 4:

We dequeue C from the queue and add it to the visited list. Then we enqueue the children into the queue.

Visited: A B F C

Queue: D G H E

The rest of the iterations also follow the same path. In the end, we get,

Visited: A B F C D G H E I J K

Advantages of BFS:

1) BFS always finds the solution if it's available to the problem.
2) This algorithm never gets stuck into a blind alley, thanks to its ability to backtrack and keep track of neighboring nodes within a stack.
3) If there exists more than one solution to a problem, it finds a solution with minimal steps.
4) In its basic form, BFS can be implemented using a relatively small amount of memory. It only requires enough memory to store the path from the starting node to the current node.[2]

Disadvantages:

1) There are a lot of memory constraints as the algorithm stores all the nodes of the present level to go to the next level.
1) For bigger trees and graphs, it will take a lot of space and time to find a solution.

Applications of BFS:

1) This algorithm is used in GPS navigation, where it helps in finding the shortest path available.
2) This is used in path-finding algorithms,
3) This can be used in the minimum spanning tree.[4]

Conclusion:

This exploration into Blind Search algorithms, particularly Depth-First Search and Breadth-First Search, reveals the fundamental contributions these algorithms make to the field of artificial intelligence. Their applications in diverse areas, ranging from network analysis to puzzle-solving, underscore their versatility. The pseudocode and examples provided offer practical insights into the workings of these algorithms, illuminating their significance in problem-solving.

As artificial intelligence continues to advance, understanding foundational algorithms becomes increasingly crucial. Blind Search algorithms serve as pillars in this journey, navigating the uncharted territories of data structures and graphs. Through this exploration, we gain valuable insights into the capabilities and limitations of DFS and BFS, paving the way for further innovations in the dynamic field of artificial intelligence.

References:

Wikipedia. (n.d.). Depth-first search. Retrieved from https://en.wikipedia.org/wiki/Depth-first_search

Vatsal, U. (n.d.). Advantages and Disadvantages of AI Algorithms. Medium. Retrieved from https://medium.com/@vatsalunadkat/advantages-and-disadvantages-of-ai-algorithms-d8fb137f4df2

Wikipedia. (n.d.). Breadth-first search. Retrieved from https://en.wikipedia.org/wiki/Breadth-first_search

FavTutor. (n.d.). Breath First Search In Python. Retrieved from https://favtutor.com/blogs/breadth-first-search-python