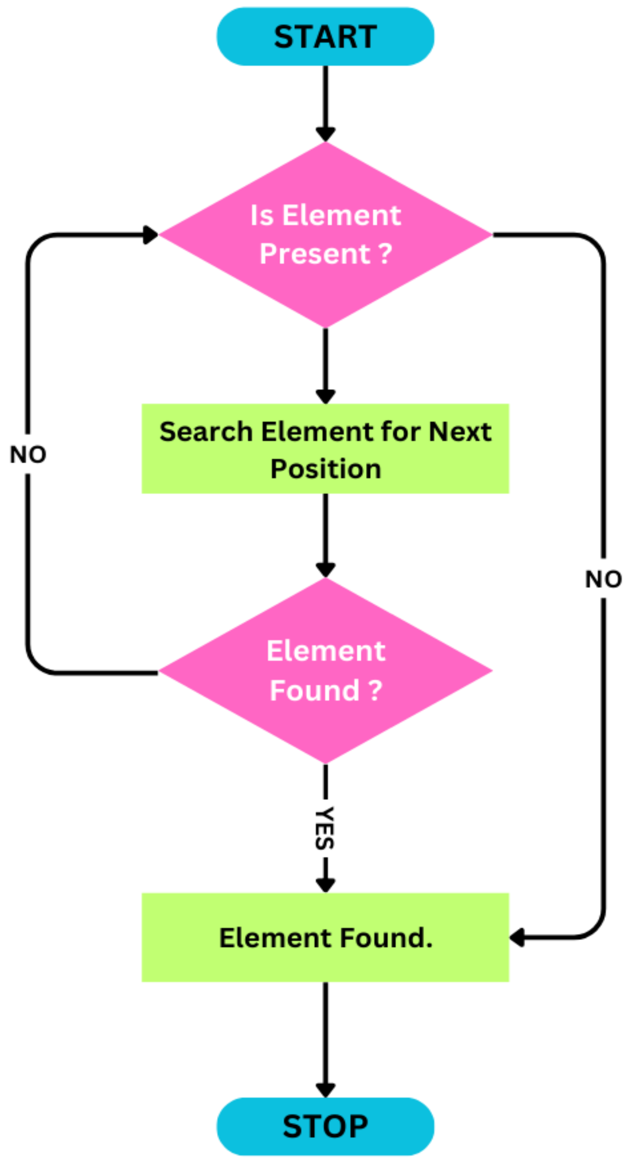
LINEAR SEARCH ALGORITHM TimeComp: O(n)

CODE : algorithms_learn/Searching Algorithms/Linear_search.cpp at main · Mayborg121/algorithms_learn



Applications of Linear Search Algorithm:

- **Unsorted Lists:** When we have an unsorted array or list, linear search is most commonly used to find any element in the collection.
- **Small Data Sets:** Linear Search is preferred over binary search when we have small data sets with
- **Searching Linked Lists:** In linked list implementations, linear search is commonly used to find elements within the list. Each node is checked sequentially until the desired element is found.
- **Simple Implementation:** Linear Search is much easier to understand and implement as compared to Binary Search or Ternary Search.

PSEUDO CODE :

```
1. Take input array from user.
2. Take element you want to search from user.
3. Start from 1st element in array to last.
4.  IF match found:
          then -> Print Message + index;
    ELSE :
          Move to next element in array;
 5. END.
```

Advantages of Linear Search Algorithm:

- Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Disadvantages of Linear Search Algorithm:

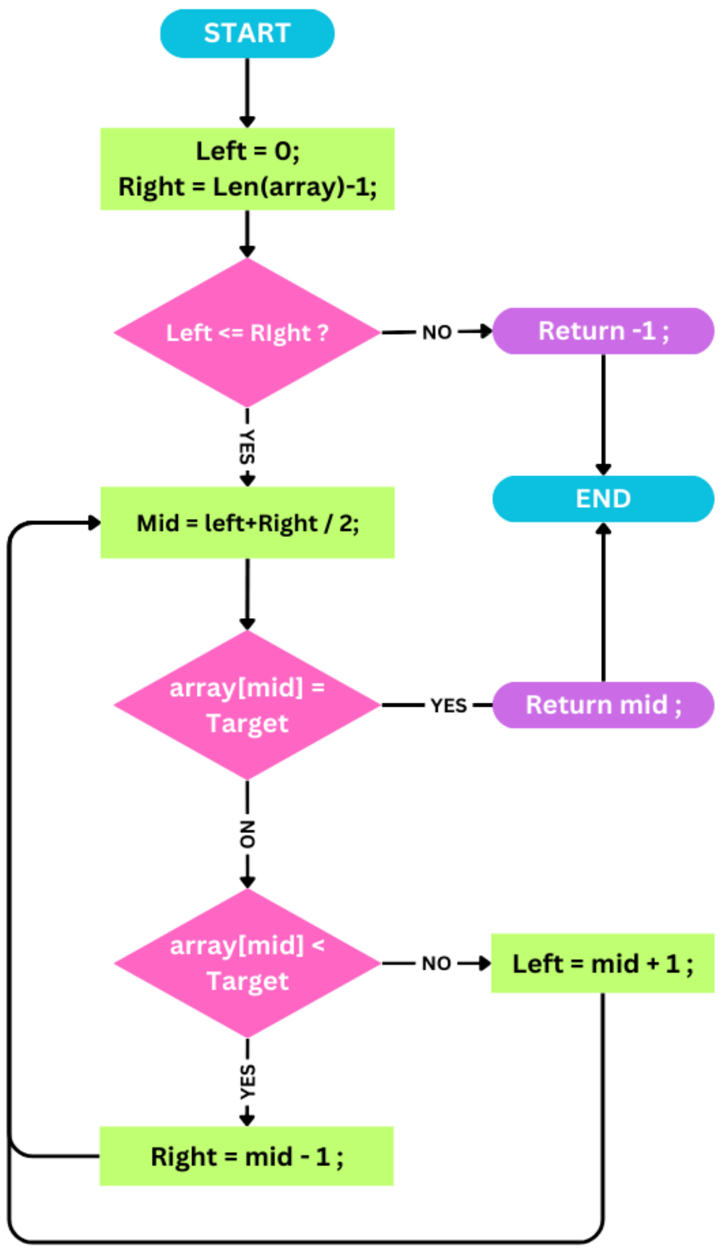
- Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
- Not suitable for large arrays.

When to use Linear Search Algorithm?

- When we are dealing with a small dataset.
- When you are searching for a dataset stored in contiguous memory.

BINARY SEARCH ALGORITHM TimeComp: O(log.n)

CODE : algorithms_learn/Searching Algorithms/Binary_Search.cpp at main · Mayborg121/algorithms_learn



Applications of Binary Search :

- Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
- It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
- It can be used for searching a database.

Advantages of Binary Search :

- Binary search is faster than linear search, especially for large arrays.
- More efficient than other searching algorithms with a similar time complexity, such as interpolation search or exponential search.
- Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.

Disadvantages of Binary Search :

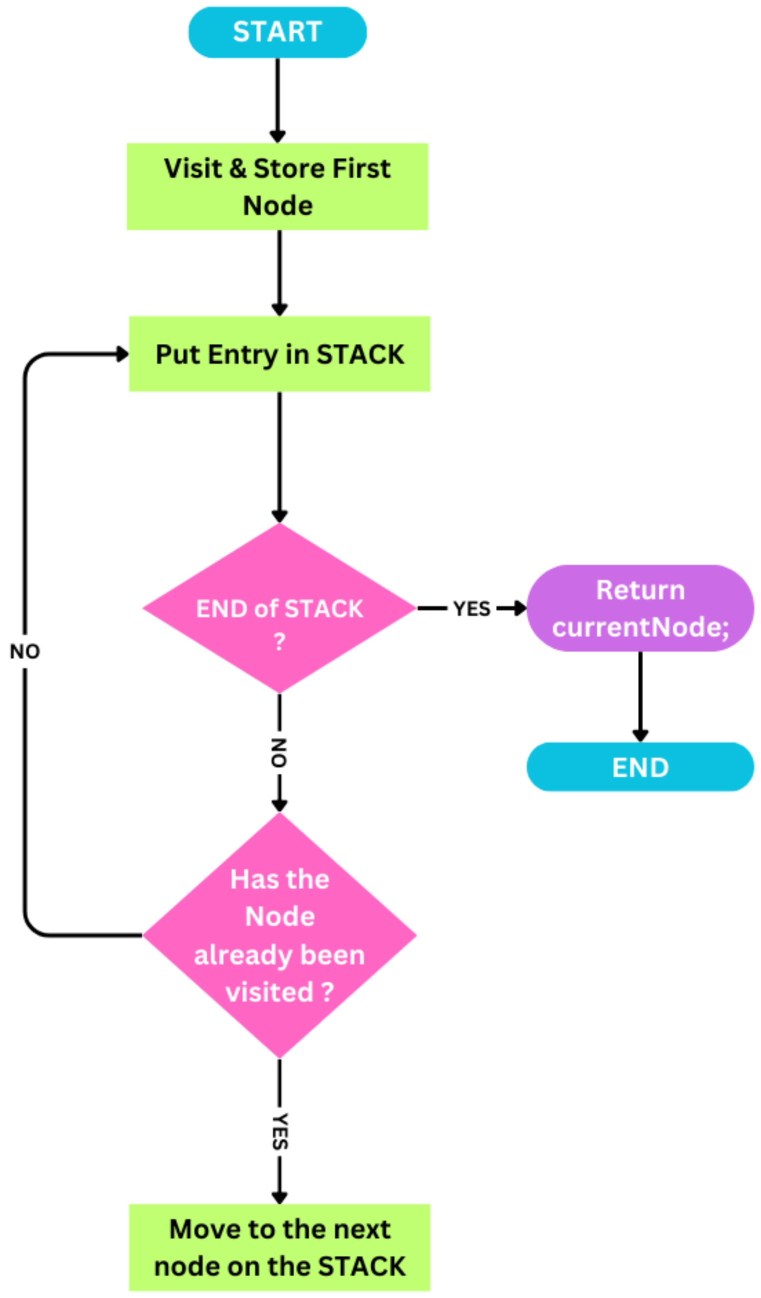
- The array should be sorted.
- Binary search requires that the data structure being searched be stored in contiguous memory locations.
- Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered.

PSEUDOCODE :

```
1. Take Input  (array[], target).
2. function binary_search(array, target):
3. left = 0
   right = length(array) - 1
        while left <= right:
               mid = (left + right) / 2
               if array[mid] == target:
                      return mid
               elseif array[mid] < target:
                      left = mid + 1
               else:
                      right = mid – 1
4. return -1
5. END
```

DEPTH FIRST SEARCH ALGORITHM TimeComp: O(|V|+|E|)

CODE : algorithms_learn/Searching Algorithms/Depth_First_Search.cpp at main · Mayborg121/algorithms_learn



Applications of Depth First Search:

- **Detecting cycle in a graph:** A graph has a cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.
- **Path Finding:** We can specialize the DFS algorithm to find a path between two given vertices u and z.
- **Topological Sorting:** Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs.
- **Backtracking:** Depth-first search can be used in backtracking algorithms.

Advantages of Depth First Search:

- Memory requirement is only linear with respect to the search graph. This is in contrast with breadth-first search which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.
- The time complexity of a depth-first Search to depth d and branching factor b (the number of children at each node, the outdegree) is O(bd)
- If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.
- DFS requires less memory since only the nodes on the current path are stored.

Disadvantages of Depth First Search:

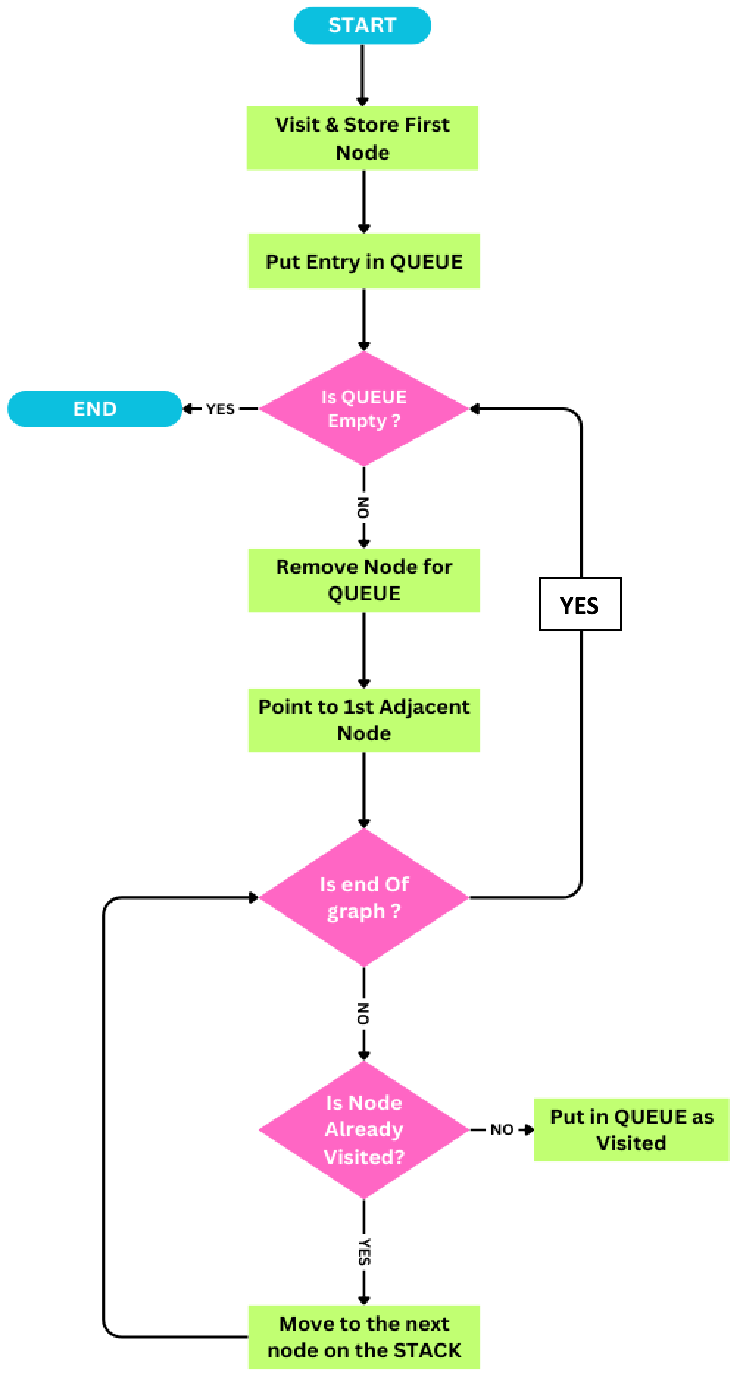
- The disadvantage of Depth-First Search is that there is a possibility that it may down the left-most path forever. Even a finite graph can generate an infinite solution to this problem is to impose a cutoff depth on the search. Depth-First Search is not guaranteed to find the solution.
- And there is no guarantee to find a minimal solution, if more than one solution.

PSEUDOCODE:

```
1. Push the starting node onto the stack.
2. While the stack is not empty, pop the top node.
3. If the node hasn't been visited, mark it as visited and print it.
4. Push all unvisited neighbors of the current node onto the stack.
```

BREADTH FIRST SEARCH ALGORITHM TimeComp: O(|V|+|E|)

CODE : algorithms_learn/Searching Algorithms/Breadth_First_Search.cpp at main · Mayborg121/algorithms_learn



PSEUDOCODE :

```
1. Start with a source node.
2. Mark the source node as visited and add it to a queue.
3. While the queue is not empty:
    a. Remove a node from the queue.
    b. Process the node (e.g., print or perform an operation).
    c. Add all unvisited neighbors to the queue and mark them as visited.
```

Applications :

1. **Shortest Path in Unweighted Graphs**: BFS finds the shortest path efficiently.
2. **Web Crawling**: Explores all pages level by level starting from a seed URL.
3. **Network Broadcasting**: Simulates spreading information in communication networks.
4. **Puzzle Solving**: Solves shortest-path problems like mazes or word ladders.

Advantages :

1. **Shortest Path Guarantee**: Always finds the shortest path in unweighted graphs.
2. **Level Order Traversal**: Naturally explores nodes layer by layer.
3. **Works for All Graphs**: Handles directed, undirected, and cyclic graphs.
4. **Simplicity**: Straightforward to implement using a queue.

Disadvantages :

1. **High Memory Usage**: Stores all visited nodes and neighbors in a queue.
2. **Inefficient for Dense Graphs**: Can be slow if many nodes have high degrees.
3. **Not Suitable for Weighted Graphs**: Does not handle weights (use Dijkstra for this).
4. **Depends on Connectivity**: Traversal is limited to connected components.

BFS	DFS
BFS stands for Breadth First Search.	DFS stands for Depth First Search.
BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.
BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).
BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.