**Breadth First Search (BFS) Algorithm**

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

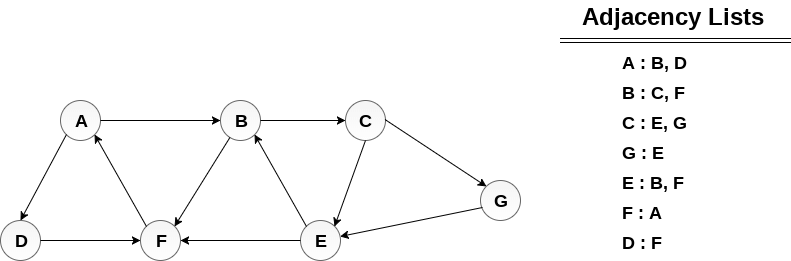
The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all of its neighbours. In the next step, the neighbours of the nearest node of A are explored and process continues in the further steps. The algorithm explores all neighbours of all the nodes and ensures that each node is visited exactly once and no node is visited twice.

**Algorithm**

* **Step 1:** SET STATUS = 1 (ready state)  
  for each node in G
* **Step 2:** Enqueue the starting node A  
  and set its STATUS = 2  
  (waiting state)
* **Step 3:** Repeat Steps 4 and 5 until  
  QUEUE is empty
* **Step 4:** Dequeue a node N. Process it  
  and set its STATUS = 3  
  (processed state).
* **Step 5:** Enqueue all the neighbours of  
  N that are in the ready state  
  (whose STATUS = 1) and set  
  their STATUS = 2  
  (waiting state)  
  [END OF LOOP]
* **Step 6:** EXIT

Example

Consider the graph G shown in the following image, calculate the minimum path p from node A to node E. Given that each edge has a length of 1.



Solution:

Minimum Path P can be found by applying breadth first search algorithm that will begin at node A and will end at E. the algorithm uses two queues, namely **QUEUE1** and **QUEUE2**. **QUEUE1** holds all the nodes that are to be processed while **QUEUE2** holds all the nodes that are processed and deleted from **QUEUE1**.

**Lets start examining the graph from Node A.**

1. Add A to QUEUE1 and NULL to QUEUE2.

QUEUE1 = {A}

QUEUE2 = {NULL}

2. Delete the Node A from QUEUE1 and insert all its neighbours. Insert Node A into QUEUE2

QUEUE1 = {B, D}

QUEUE2 = {A}

3. Delete the node B from QUEUE1 and insert all its neighbours. Insert node B into QUEUE2.

QUEUE1 = {D, C, F}

QUEUE2 = {A, B}

4. Delete the node D from QUEUE1 and insert all its neighbours. Since F is the only neighbour of it which has been inserted, we will not insert it again. Insert node D into QUEUE2.

QUEUE1 = {C, F}

QUEUE2 = { A, B, D}

5. Delete the node C from QUEUE1 and insert all its neighbours. Add node C to QUEUE2.

QUEUE1 = {F, E, G}

QUEUE2 = {A, B, D, C}

6. Remove F from QUEUE1 and add all its neighbours. Since all of its neighbours has already been added, we will not add them again. Add node F to QUEUE2.

QUEUE1 = {E, G}

QUEUE2 = {A, B, D, C, F}

7. Remove E from QUEUE1, all of E's neighbours has already been added to QUEUE1 therefore we will not add them again. All the nodes are visited and the target node i.e. E is encountered into QUEUE2.

QUEUE1 = {G}

QUEUE2 = {A, B, D, C, F,  E}

Now, backtrack from E to A, using the nodes available in QUEUE2.

The minimum path will be **A → B → C → E**.

## ****Topological Sort-****

Topological Sort is a linear ordering of the vertices in such a way that if there is an edge in the DAG going from vertex ‘u’ to vertex ‘v’, then ‘u’ comes before ‘v’ in the ordering.

* Topological Sorting is possible if and only if the graph is a [**Directed Acyclic Graph**](https://www.gatevidyalay.com/directed-acyclic-graphs/).
* There may exist multiple different topological orderings for a given directed acyclic graph.

**Algorithm**

**topoSort(u, visited, stack)**

**Input −** The start vertex u, An array to keep track of which node is visited or not. A stack to store nodes.  
**Output −** Sorting the vertices in topological sequence in the stack.

Begin

   mark u as visited

   for all vertices v which is adjacent with u, do

      if v is not visited, then

         topoSort(c, visited, stack)

   done

   push u into a stack

End

**performTopologicalSorting(Graph)**

**Input −** The given directed acyclic graph.  
**Output −** Sequence of nodes.

Begin

   initially mark all nodes as unvisited

   for all nodes v of the graph, do

      if v is not visited, then

         topoSort(i, visited, stack)

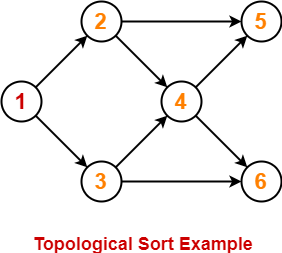
   done

   pop and print all elements from the stack

End.

**Topological Sort Example-**

 Consider the following directed acyclic graph-



For this graph, following 4 different topological orderings are possible-

* **1 2 3 4 5 6**
* **1 2 3 4 6 5**
* **1 3 2 4 5 6**
* **1 3 2 4 6 5**

**Applications of Topological Sort-**

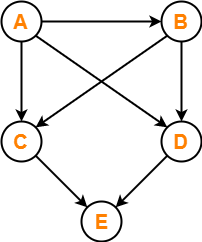
Few important applications of topological sort are-

* Scheduling jobs from the given dependencies among jobs
* Instruction Scheduling
* Determining the order of compilation tasks to perform in makefiles
* Data Serialization

**PRACTICE PROBLEMS BASED ON TOPOLOGICAL SORT-**

**Problem-01:**

Find the number of different topological orderings possible for the given graph-

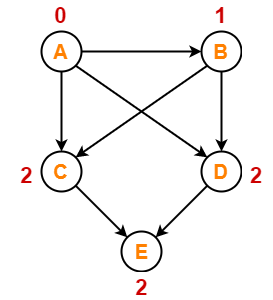


**Solution-**

The topological orderings of the above graph are found in the following steps-

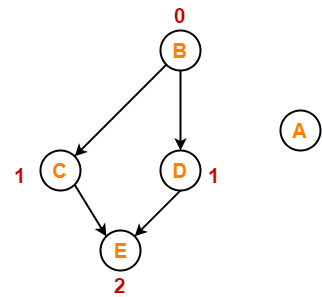
**Step-01:**

Write in-degree of each vertex-



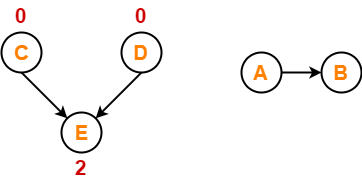
**Step-02:**

* Vertex-A has the least in-degree.
* So, remove vertex-A and its associated edges.
* Now, update the in-degree of other vertices.



**Step-03:**

* Vertex-B has the least in-degree.
* So, remove vertex-B and its associated edges.
* Now, update the in-degree of other vertices.



**Step-04:**

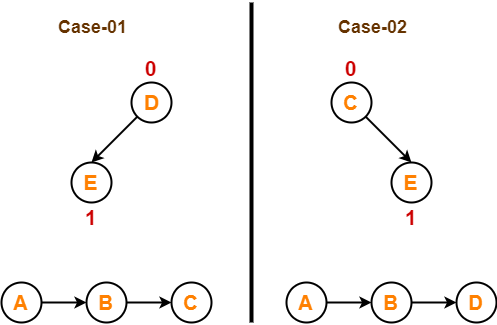
There are two vertices with the least in-degree. So, following 2 cases are possible-

In case-01,

* Remove vertex-C and its associated edges.
* Then, update the in-degree of other vertices.

In case-02,

* Remove vertex-D and its associated edges.
* Then, update the in-degree of other vertices.



**Step-05:**

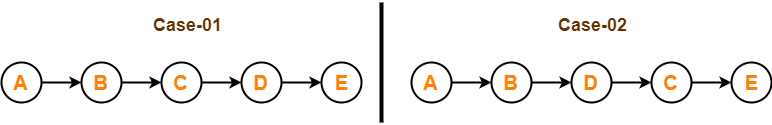
Now, the above two cases are continued separately in the similar manner.

In case-01,

* Remove vertex-D since it has the least in-degree.
* Then, remove the remaining vertex-E.

In case-02,

* Remove vertex-C since it has the least in-degree.
* Then, remove the remaining vertex-E.



**Conclusion-**

For the given graph, following **2** different topological orderings are possible-

* **A B C D E**
* **A B D C E**

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.javatpoint.com/breadth-first-search-algorithm>
2. <https://www.gatevidyalay.com/topological-sort-topological-sorting/>

**Lecture Video:**

1. <https://youtu.be/vf-cxgUXcMk>
2. <https://youtu.be/dis_c84ejhQ>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**