GIT DEPARTMENT OF COMPUTER ENGINEERING

CSE 222/505 - SPRING 2020

HOMEWORK 8

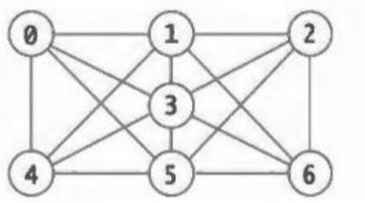
REPORT

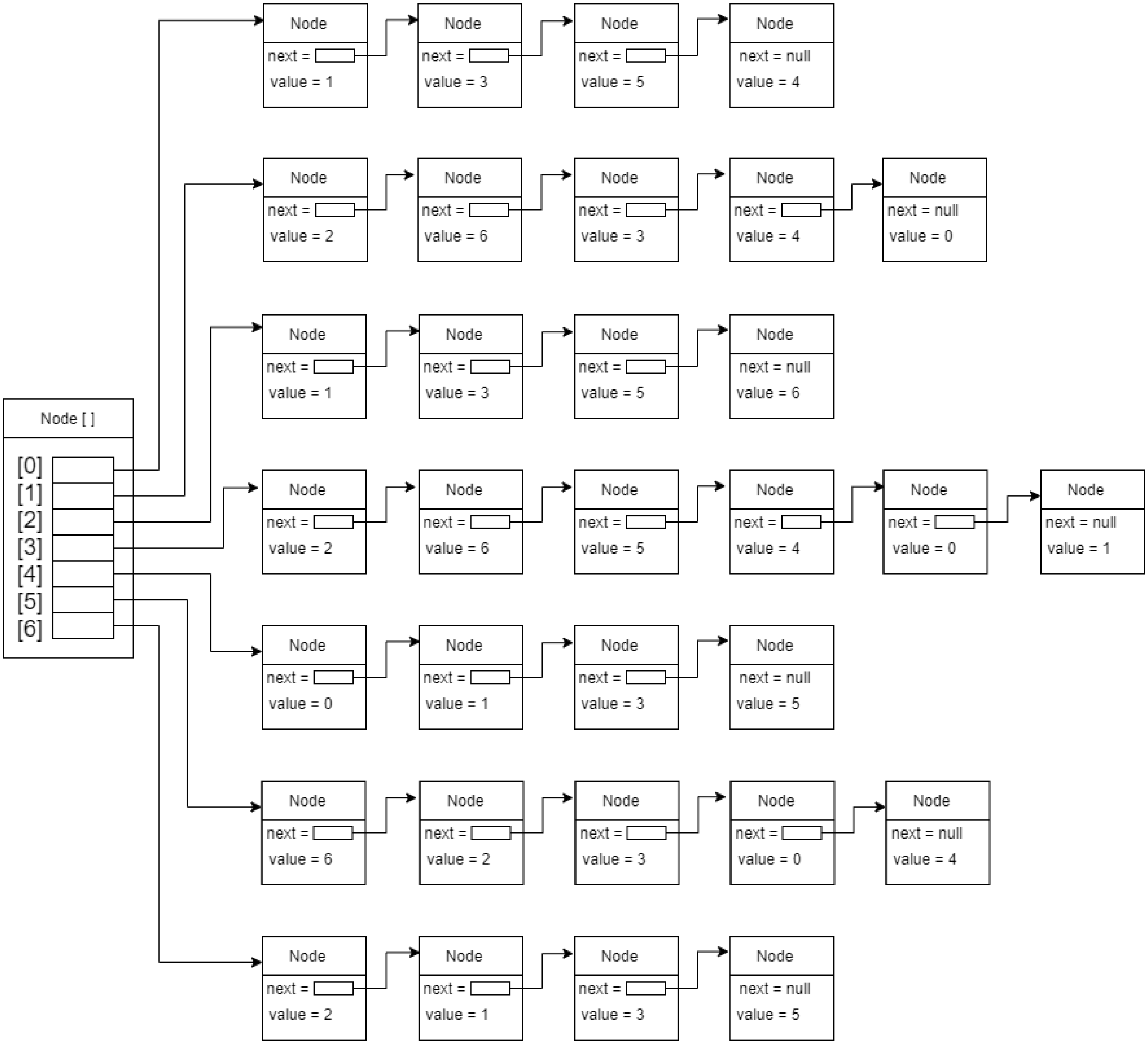
ŞEYDA ÖZER

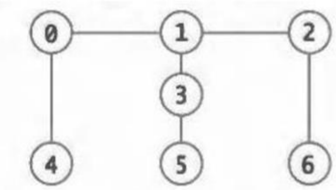
171044023

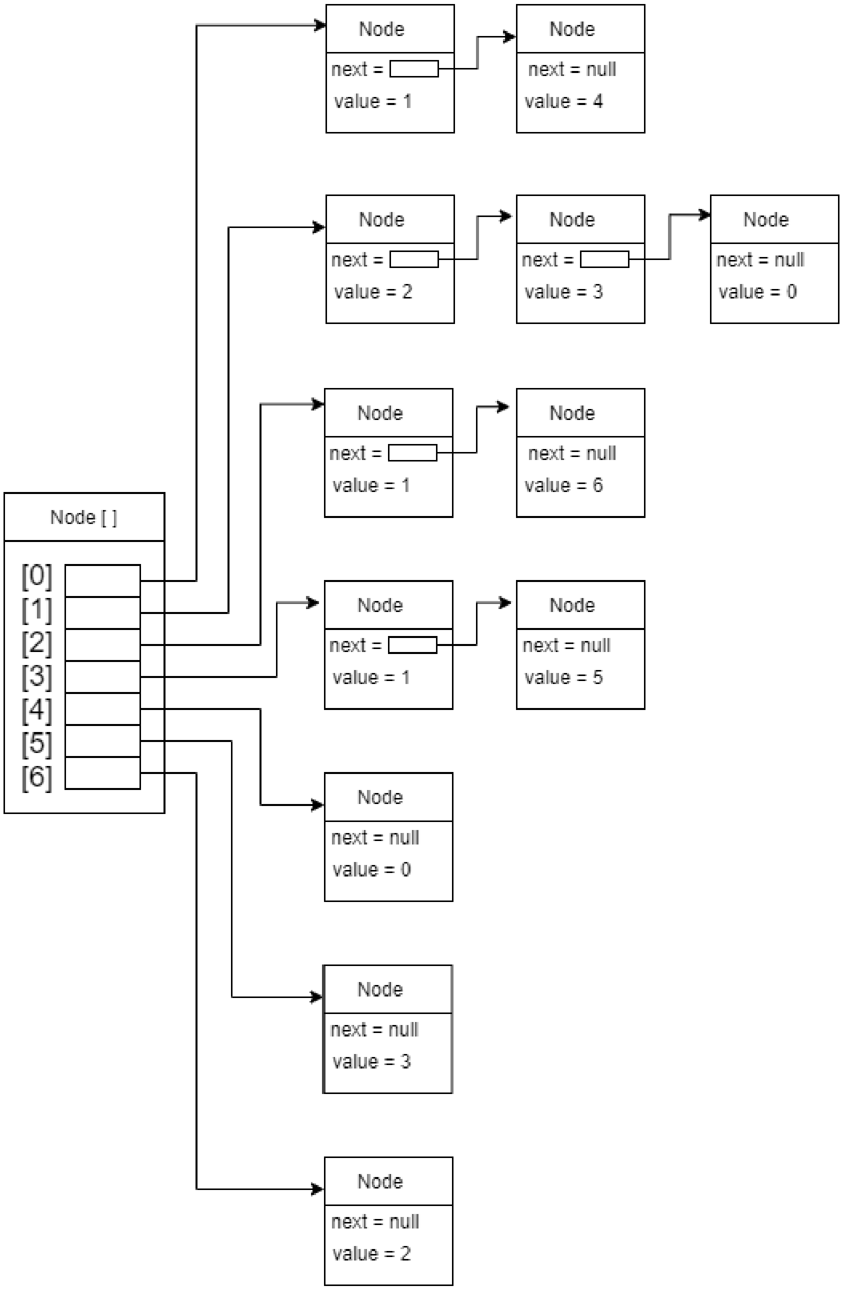
Q1:

- Represent the graphs above using adjacency lists. Draw the corresponding data structure.

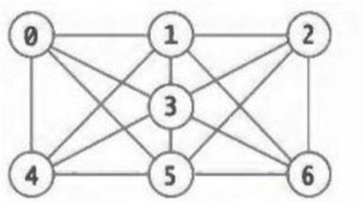




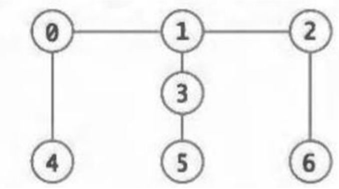

0
1
2
3
4
5
6


Node [ ]
[0]
[1]
[2]
[3]
[4]
[5]
[6]
Node
next =
value = 1
Node
next = null
value = 4
Node
next =
value = 2
Node
next =
value = 3
Node
next = null
value = 0
Node
next =
value = 1
Node
next = null
value = 6
Node
next =
value = 1
Node
next = null
value = 5
Node
next = null
value = 0
Node
next = null
value = 3
Node
next = null
value = 2

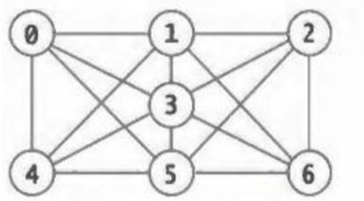
- Represent the graphs above using an adjacency matrix. Draw the corresponding data structure.

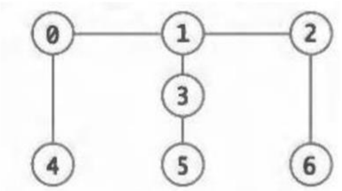


	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	∞	1.0	∞	1.0	1.0	1.0	∞
[1]	1.0	∞	1.0	1.0	1.0	∞	1.0
[2]	∞	1.0	∞	1.0	∞	1.0	1.0
[3]	1.0	1.0	1.0	∞	1.0	1.0	1.0
[4]	1.0	1.0	∞	1.0	∞	1.0	∞
[5]	1.0	∞	1.0	1.0	1.0	∞	1.0
[6]	∞	1.0	1.0	1.0	∞	1.0	∞



	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	∞	1.0	∞	∞	1.0	∞	∞
[1]	1.0	∞	1.0	1.0	∞	∞	∞
[2]	∞	1.0	∞	∞	∞	∞	1.0
[3]	∞	1.0	∞	∞	∞	1.0	∞
[4]	1.0	∞	∞	∞	∞	∞	∞
[5]	∞	∞	∞	1.0	∞	∞	∞
[6]	∞	∞	1.0	∞	∞	∞	∞





- For each graph above, what are the IVI=n, the IEI=m, and the density? Which representation is better for each graph? Explain your answers.

First graph:

IVI = 7, IEI = 16

Density = IEI / IVI^2 = 16 / 49

Second graph:

IVI = 7, IEI = 6

Density = IEI / IVI^2 = 6 / 49

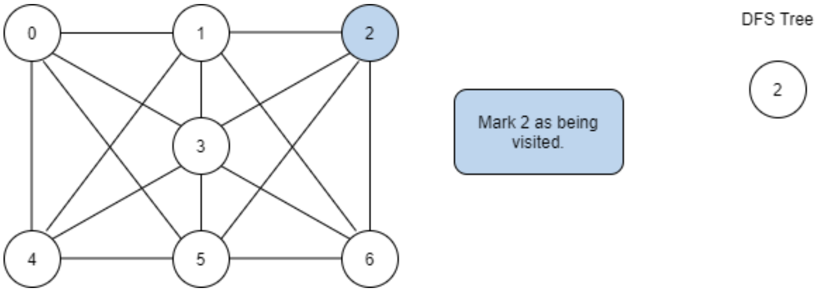
First graph density > second graph density

The first graph is a dense graph. For a dense graph, the run times of the adjacency matrix and adjacency list are approximately the same. However, the adjacency matrix uses less storage space. For this reason, the adjacency matrix should be used for dense graphs.

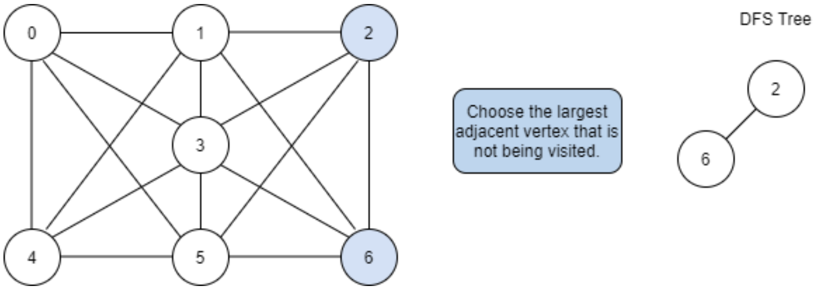
The second graph is a sparse graph. The adjacent list should be used for the sparse graph, because for sparse graphs there is no need to through the all vertices of the graph.

- Draw DFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest)

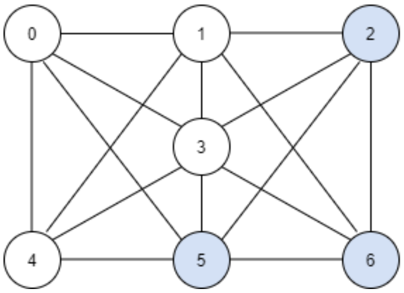
First graph:



Mark 2 as being visited.

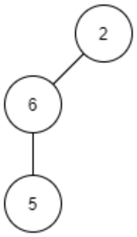


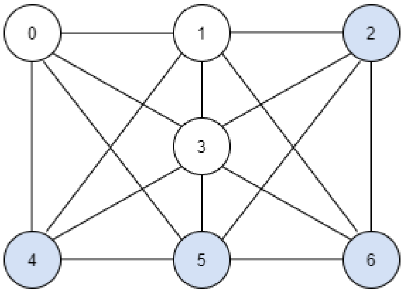
Choose the largest adjacent vertex that is not being visited.



Choose the largest adjacent vertex that is not being visited.

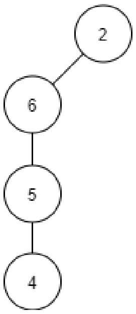
DFS Tree

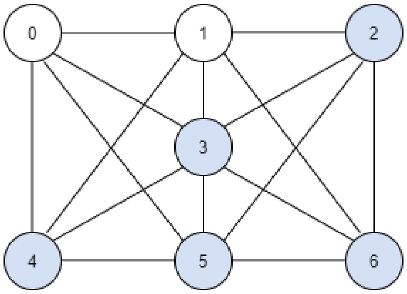




Choose the largest adjacent vertex that is not being visited.

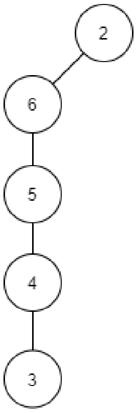
DFS Tree

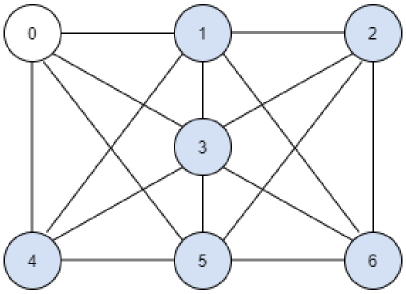




Choose the largest adjacent vertex that is not being visited.

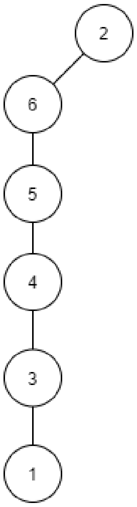
DFS Tree

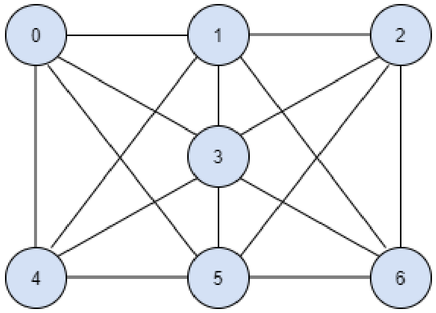




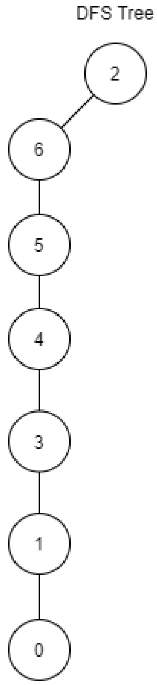
Choose the largest adjacent vertex that is not being visited.

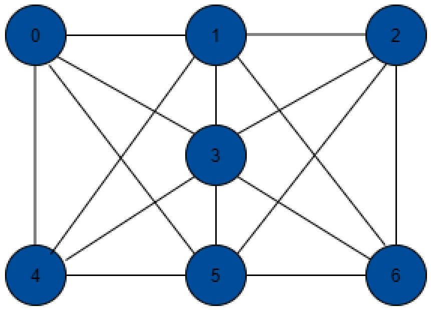
DFS Tree





Choose the largest adjacent vertex that is not being visited.





There are no vertices adjacent to 0 that are not being visited. Mark 0 as visited.

Return from the recursion to 1 All vertices adjacent to 1 are being visited. Mark 1 as visited.

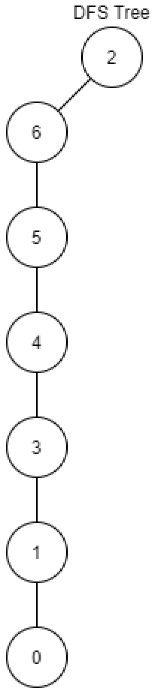
Return from the recursion to 3 All vertices adjacent to 3 are being visited. Mark 3 as visited.

Return from the recursion to 4 All vertices adjacent to 4 are being visited. Mark 4 as visited.

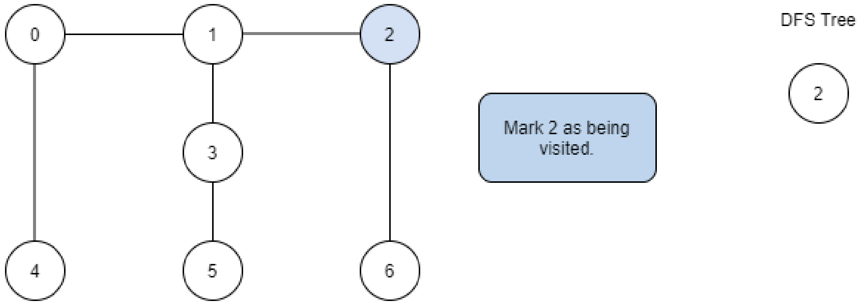
Return from the recursion to 5 All vertices adjacent to 5 are being visited. Mark 5 as visited.

Return from the recursion to 6 All vertices adjacent to 6 are being visited. Mark 6 as visited.

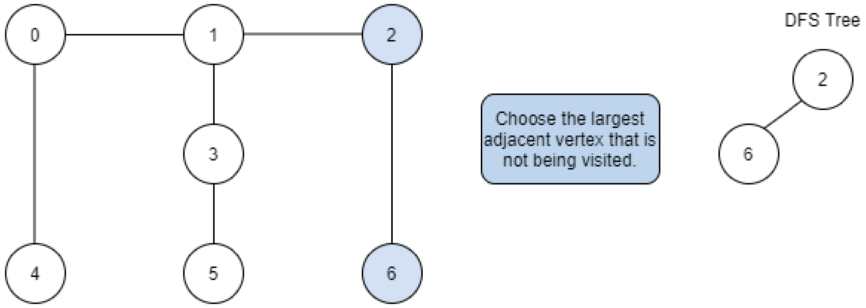
Return from the recursion to 2 All vertices adjacent to 2 are being visited. Mark 2 as visited.



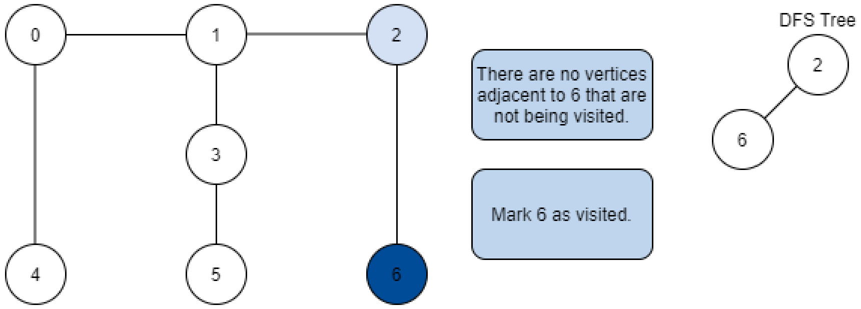
Second graph:



Mark 2 as being visited.

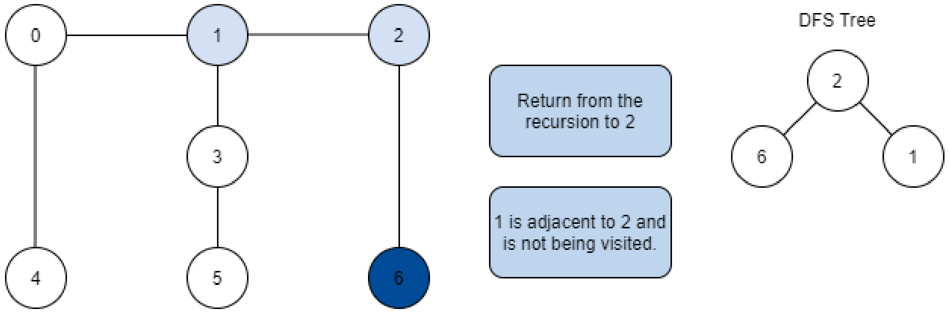


Choose the largest adjacent vertex that is not being visited.



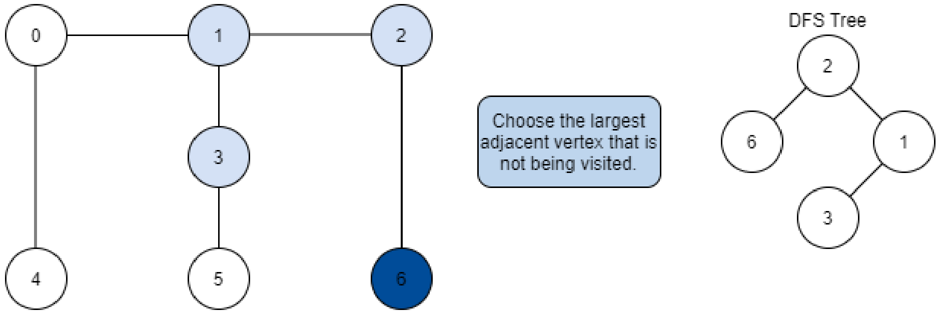
There are no vertices adjacent to 6 that are not being visited.

Mark 6 as visited.

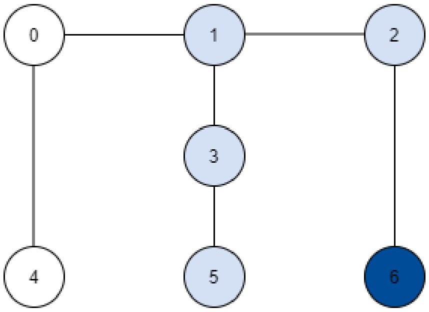


Return from the recursion to 2

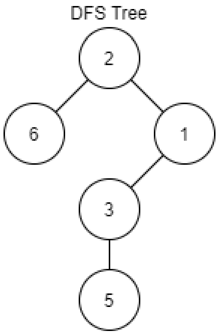
1 is adjacent to 2 and is not being visited.

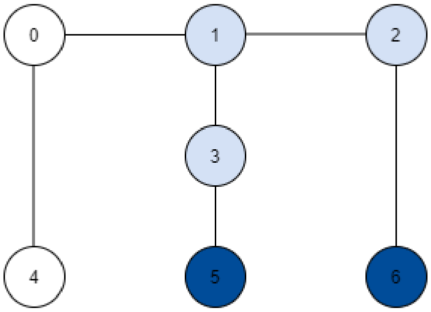


Choose the largest adjacent vertex that is not being visited.



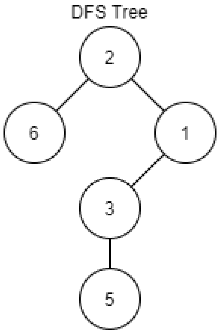
Choose the largest adjacent vertex that is not being visited.

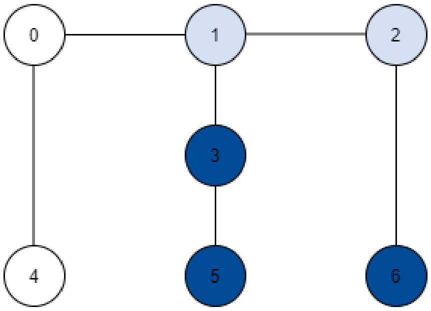




There are no vertices adjacent to 5 that are not being visited.

Mark 5 as visited.

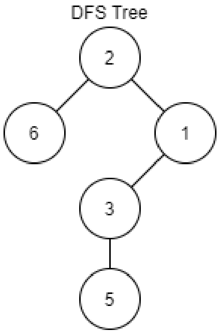


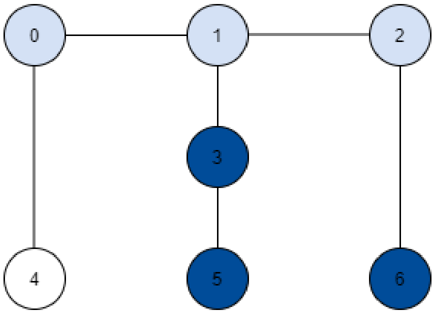


Return from the recursion to 3

All vertices adjacent to 3 are being visited.

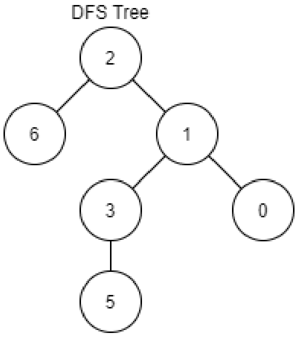
Mark 3 as visited.

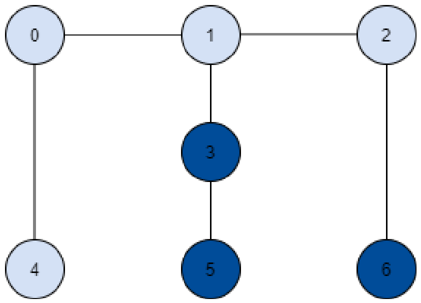




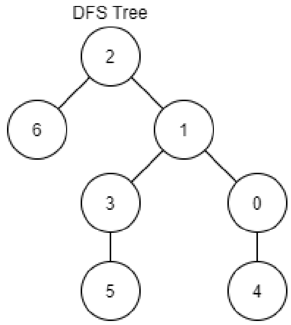
Return from the recursion to 1

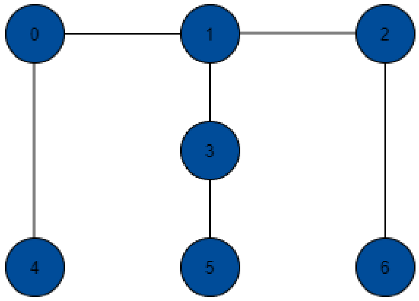
0 is adjacent to 1 and is not being visited.





Choose the largest adjacent vertex that is not being visited.



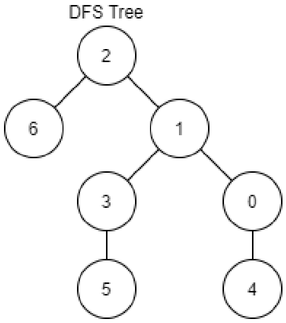


There are no vertices adjacent to 4 that are not being visited. Mark 4 as visited.

Return from the recursion to 0 All vertices adjacent to 0 are being visited. Mark 0 as visited.

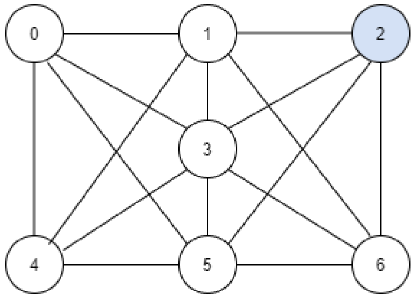
Return from the recursion to 1 All vertices adjacent to 1 are being visited. Mark 1 as visited.

Return from the recursion to 2 All vertices adjacent to 2 are being visited. Mark 2 as visited.



- Draw BFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).

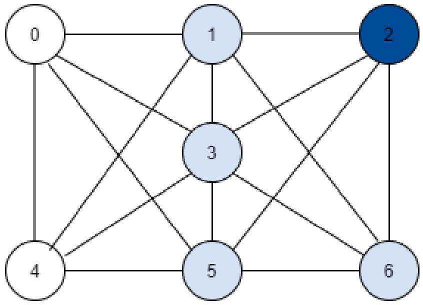
First graph:



While visiting 2, we identify its adjacent nodes and add them to a queue.

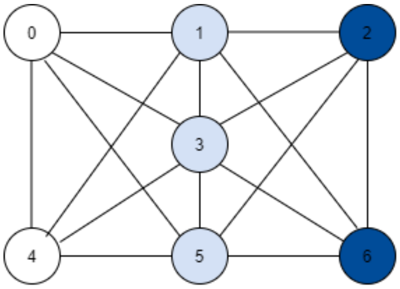
We color 2 as visited.

Queue:
6, 5, 3, 1



The queue determines which nodes to visit next

Queue:
6, 5, 3, 1



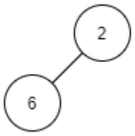
Visit the first node in the queue, 6

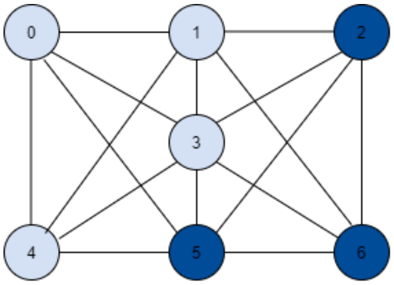
Select all its adjacent nodes that have not been visited or identified

6 is done, we color it as visited

Queue:
5, 3, 1

BFS Tree





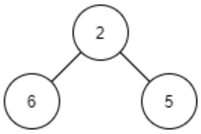
Visit the first node in the queue, 5

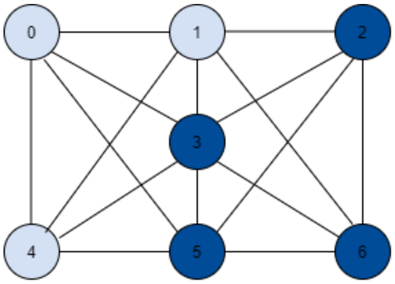
Select all its adjacent nodes that have not been visited or identified

5 is done, we color it as visited

Queue:
3, 1, 4, 0

BFS Tree





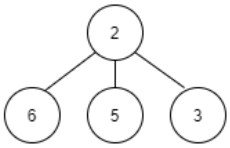
Visit the first node in the queue, 3

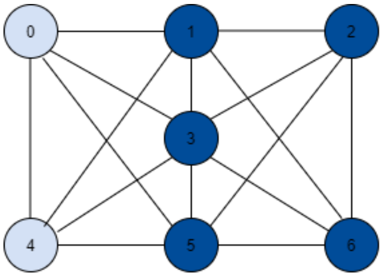
Select all its adjacent nodes that have not been visited or identified

3 is done, we color it as visited

Queue:
1, 4, 0

BFS Tree





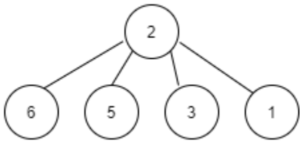
Visit the first node in the queue, 1

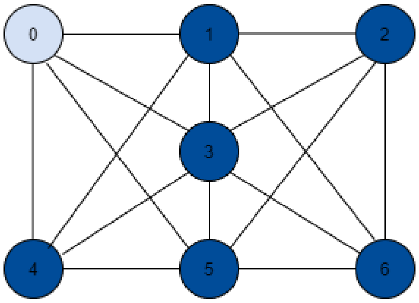
Select all its adjacent nodes that have not been visited or identified

1 is done, we color it as visited

Queue:
4, 0

BFS Tree



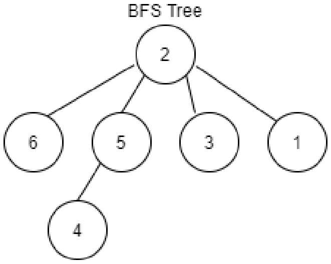


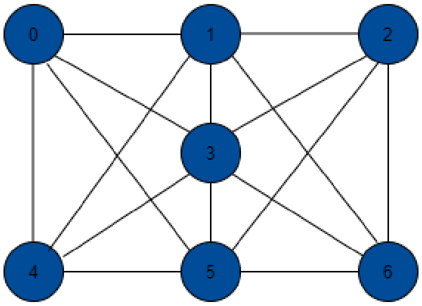
Visit the first node in the queue, 4

Select all its adjacent nodes that have not been visited or identified

4 is done, we color it as visited

Queue:
0





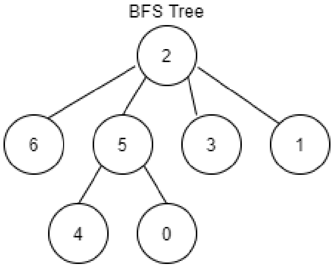
Visit the first node in the queue, 0

Select all its adjacent nodes that have not been visited or identified

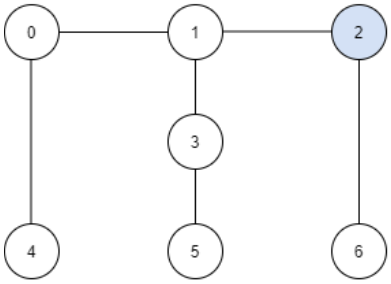
0 is done, we color it as visited

Queue:
empty

The queue is empty; all vertices have been visited.



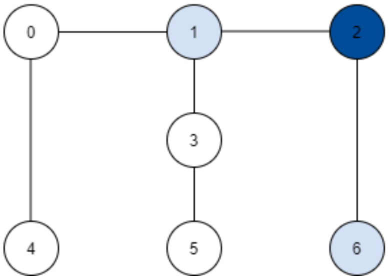
Second graph:



While visiting 2, we identify its adjacent nodes and add them to a queue.

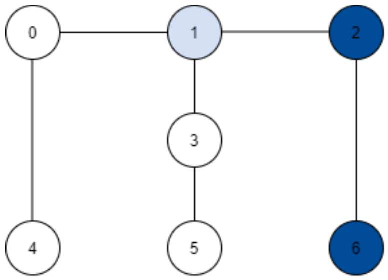
We color 2 as visited.

Queue:
6, 1



The queue determines which nodes to visit next

Queue:
6, 1

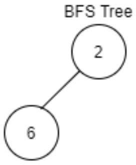


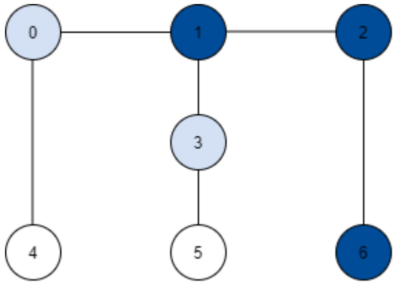
Visit the first node in the queue, 6

Select all its adjacent nodes that have not been visited or identified

6 is done, we color it as visited

Queue:
1



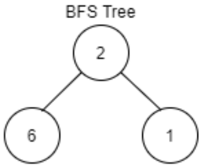


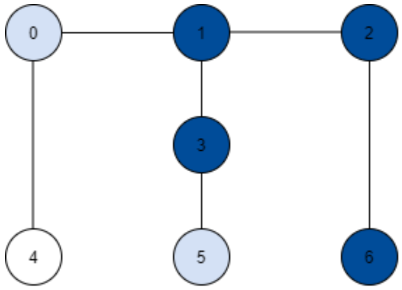
Visit the first node in the queue, 1

Select all its adjacent nodes that have not been visited or identified

1 is done, we color it as visited

Queue:
3, 0



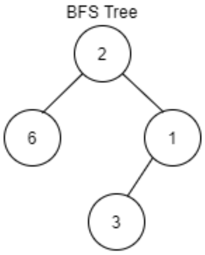


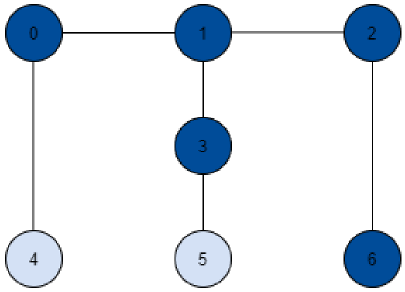
Visit the first node in the queue, 3

Select all its adjacent nodes that have not been visited or identified

3 is done, we color it as visited

Queue:
0, 5



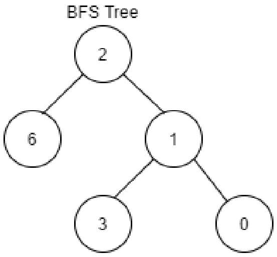


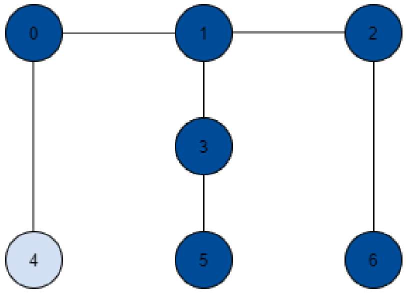
Visit the first node in the queue, 0

Select all its adjacent nodes that have not been visited or identified

0 is done, we color it as visited

Queue:
5, 4



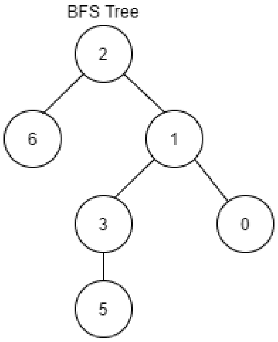


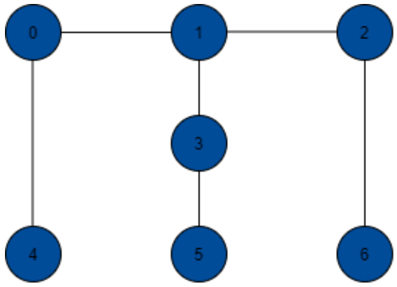
Visit the first node in the queue, 5

Select all its adjacent nodes that have not been visited or identified

5 is done, we color it as visited

Queue:
4





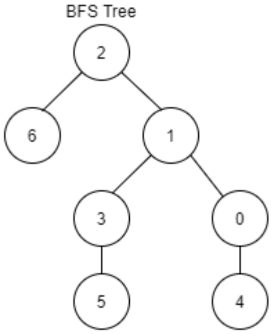
Visit the first node in the queue, 4

Select all its adjacent nodes that have not been visited or identified

4 is done, we color it as visited

Queue:
empty

The queue is empty; all vertices have been visited.


BFS Tree
2
6
1
3
0
5
4