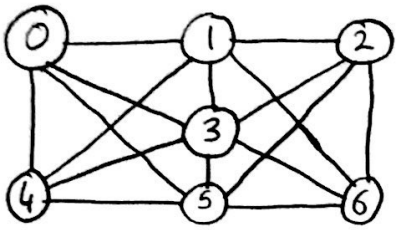
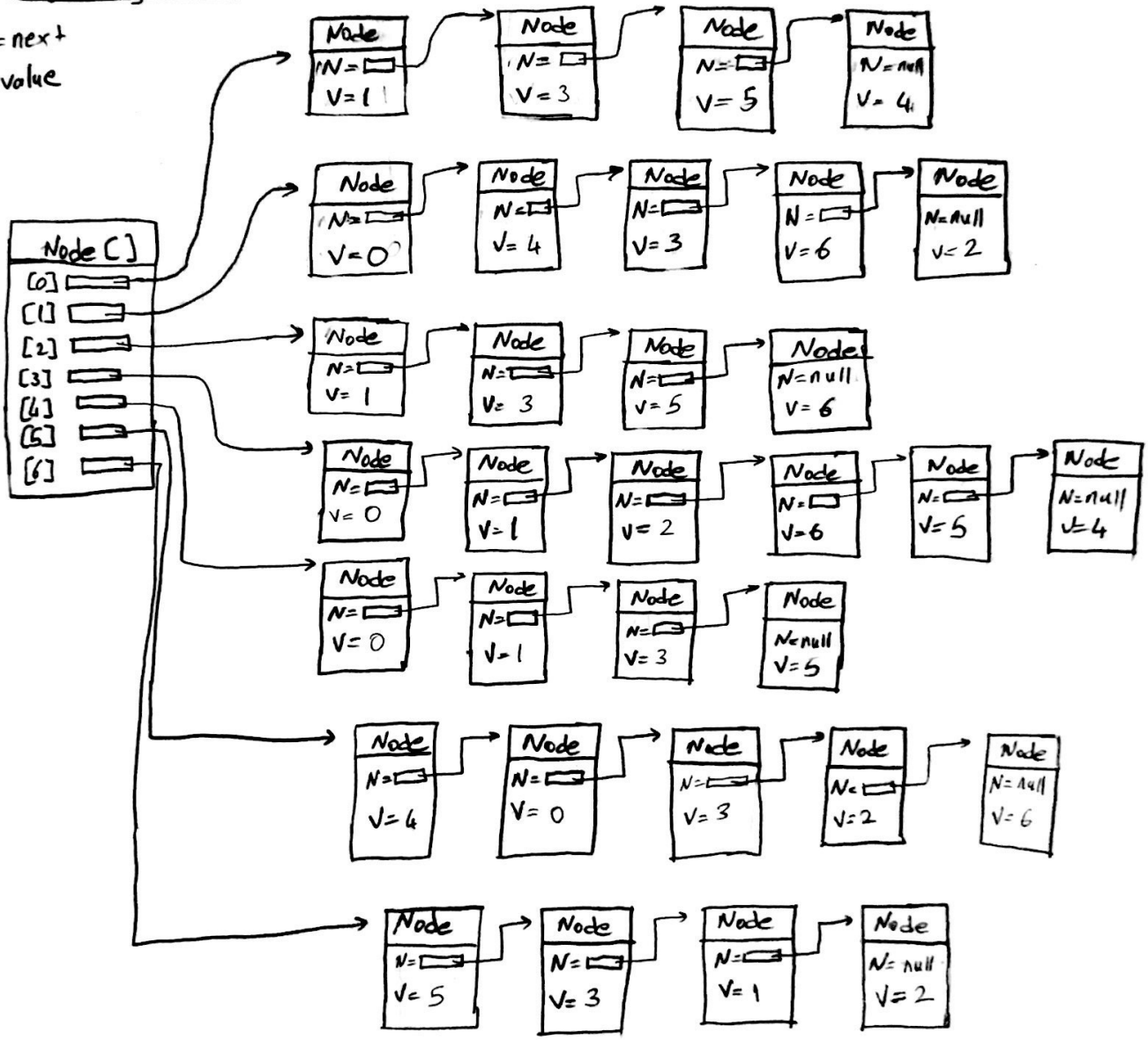


Adjacency List

N = next
V = value



Adjacency Matrix

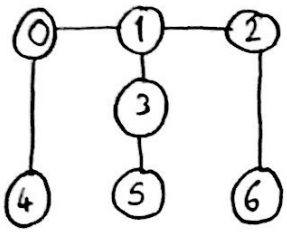
	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]		1.0		1.0	1.0	1.0	
[1]	1.0		1.0	1.0	1.0		1.0
[2]		1.0		1.0		1.0	1.0
[3]	1.0	1.0	1.0		1.0	1.0	1.0
[4]	1.0	1.0		1.0		1.0	
[5]	1.0		1.0	1.0	1.0		1.0
[6]		1.0	1.0	1.0		1.0	

$|V| = 7 \quad |E| = 14 \quad \text{Density} = \frac{2|E|}{|V|(|V|-1)} = \frac{2 \cdot 14}{7 \cdot 6} = \underline{\underline{0,66}}$

This is dense graph.

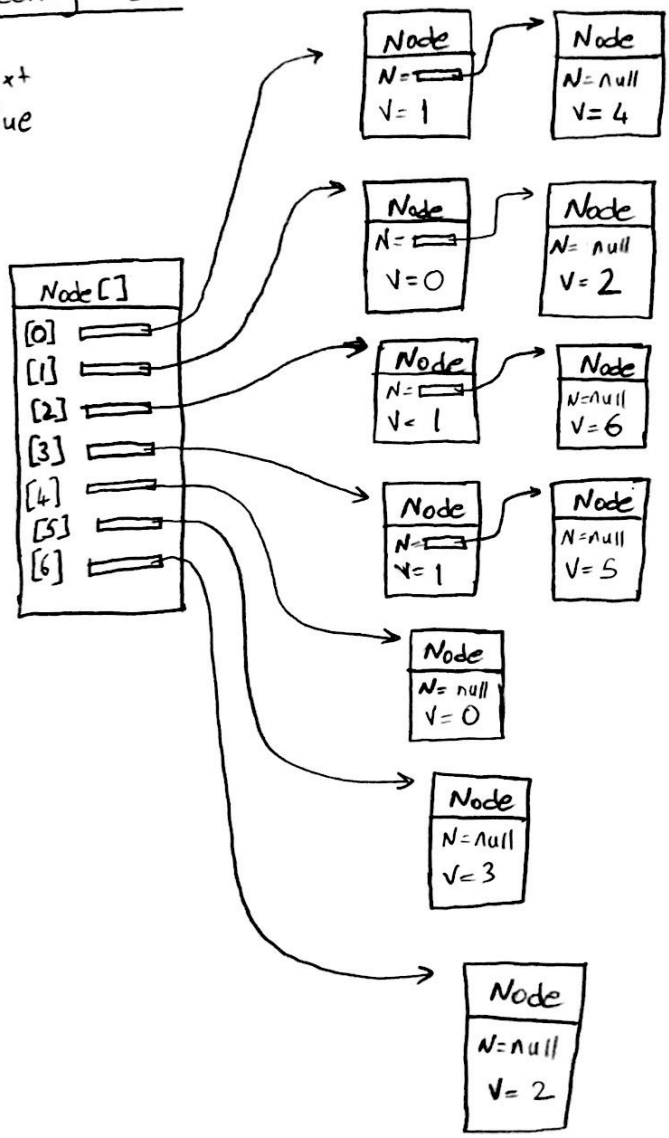
```
1. for each vertex u in some subset of the vertices
2.     for each vertex v in some subset of the vertices
3.         if (u,v) is an edge
4.             Do something with edge (u,v)
```

Step 3 for adjacency matrix rep. $O(1)$ and overall algorithm is $O(|V^2|)$
for adjacency list rep. $O(|E_u|)$ and the overall algorithm is $O(|V||E|)$.
So adjacency matrix is better for this graph.



Adjacency List

N = next
V = value



Adjacency Matrix

	[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				

$|V| = 7$ $|E| = 6$ $\text{Density} = \frac{2 \cdot |E|}{|V|(|V|-1)} = \frac{2 \cdot 6}{7 \cdot 6} = \underline{\underline{0.28}}$

This is sparse matrix

For time efficiency if the graph is sparse, the adjacency list representation is better.

DEPTH – FIRST SEARCH

Algorithm for Depth-First Search

1. Mark the current vertex, u, visited (color it light orange), and enter it in the discovery order list.

2. for each vertex, v, adjacent to the current vertex, u

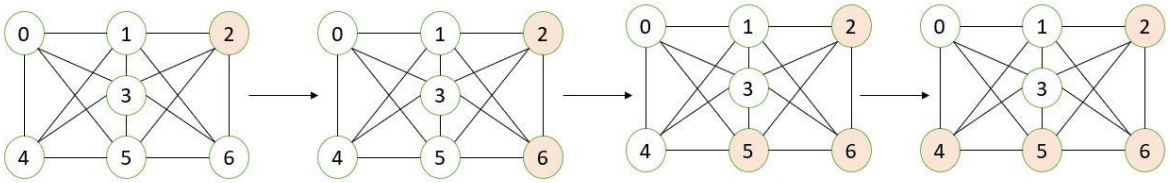
3. if v has not been visited

4. Set parent of v to u.

5. Recursively apply this algorithm starting at v.

6. Mark u finished (color it light blue) and enter u into the finish order list

GRAPH A)

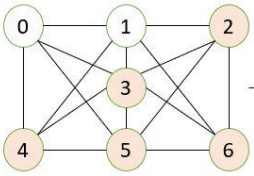


Discovery (Visit) order:
2
Finish order:

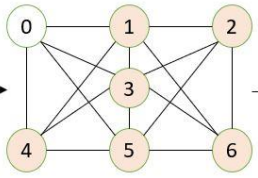
Discovery (Visit) order:
2,6
Finish order:

Discovery (Visit) order:
2,6,5
Finish order:

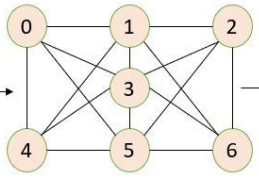
Discovery (Visit) order:
2,6,5,4
Finish order:



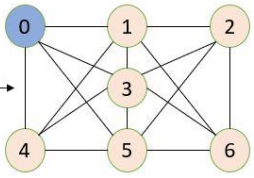
Discovery (Visit) order:
2,6,5,4,3
Finish order:



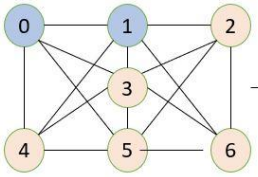
Discovery (Visit) order:
2,6,5,4,3,1
Finish order:



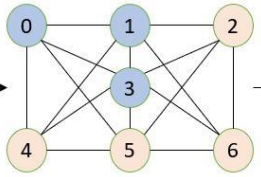
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:



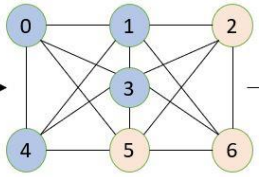
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0



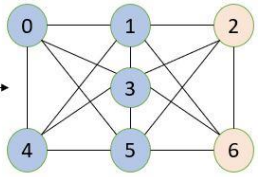
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1



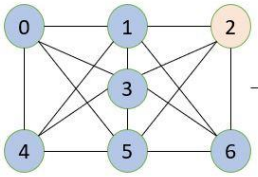
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1,3



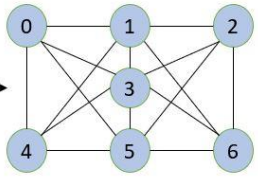
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1,3,4



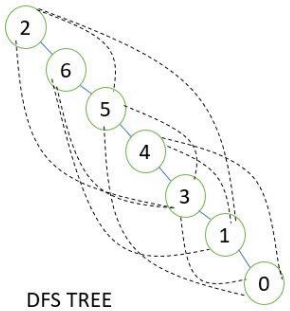
Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1,3,4,5



Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1,3,4,5,6

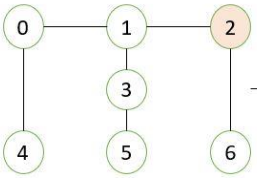


Discovery (Visit) order:
2,6,5,4,3,1,0
Finish order:
0,1,3,4,5,6,2

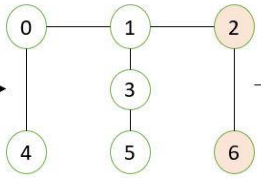


DFS TREE

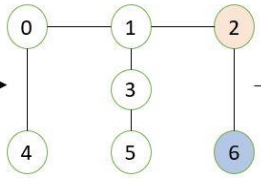
GRAPH B)



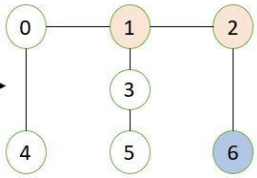
Discovery (Visit) order:
2
Finish order:



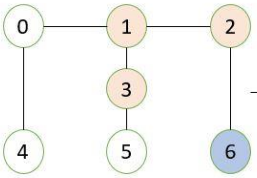
Discovery (Visit) order:
2,6
Finish order:



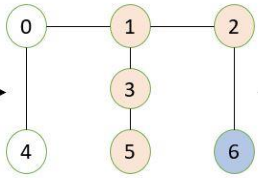
Discovery (Visit) order:
2
Finish order:
6



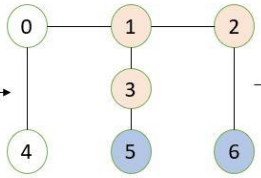
Discovery (Visit) order:
2,1
Finish order:
6



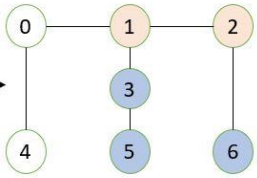
Discovery (Visit) order:
2,1,3
Finish order:
6



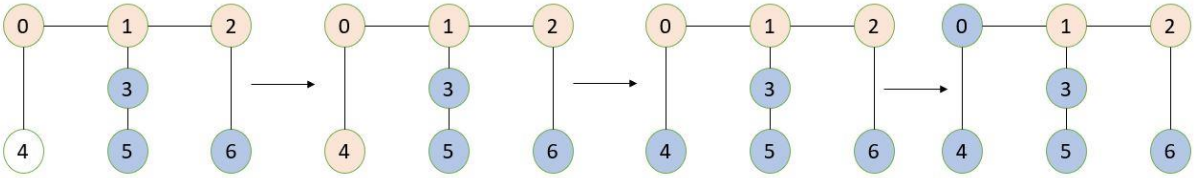
Discovery (Visit) order:
2,1,3,5
Finish order:
6



Discovery (Visit) order:
2,1,3
Finish order:
6,5



Discovery (Visit) order:
2,1
Finish order:
6,5,3

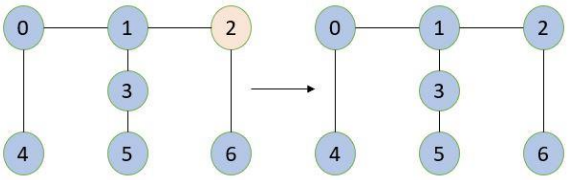


Discovery (Visit) order:
2,1,0
Finish order:
6,5,3

Discovery (Visit) order:
2,1,0,4
Finish order:
6,5,3

Discovery (Visit) order:
2,1,0
Finish order:
6,5,3,4

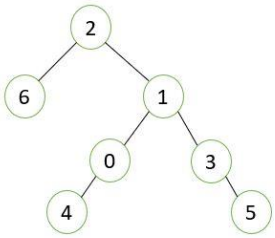
Discovery (Visit) order:
2,1
Finish order:
6,5,3,4,0



Discovery (Visit) order:
2
Finish order:
6,5,3,4,0

Discovery (Visit) order:

Finish order:
6,5,3,4,0,2



DFS TREE

BREADTH – FIRST SEARCH

Algorithm for Breadth-First Search

1. Take an arbitrary start vertex, mark it identified (color it light gray), and place it in a queue.

2. while the queue is not empty

3. Take a vertex, u, out of the queue and visit u.

4. for all vertices, v, adjacent to this vertex, u

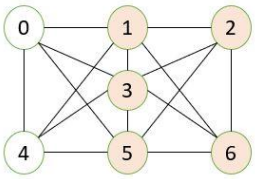
5. if v has not been identified or visited

6. Mark it identified (color it light gray).

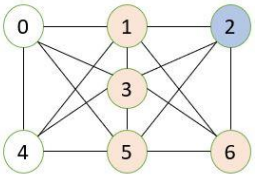
7. Insert vertex v into the queue.

8. We are now finished visiting u (color it dark gray).

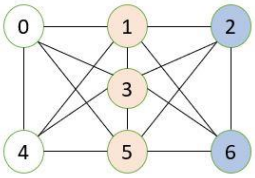
GRAPH A)



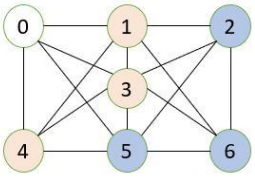
Queue:
6,5,3,1
Visit sequence:
2



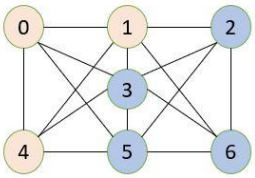
Queue:
6,5,3,1
Visit sequence:
2



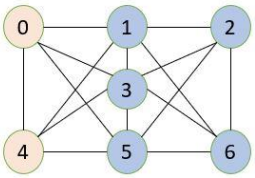
Queue:
5,3,1
Visit sequence:
2,6



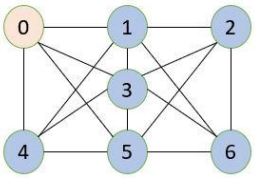
Queue:
3,1,4
Visit sequence:
2,6,5



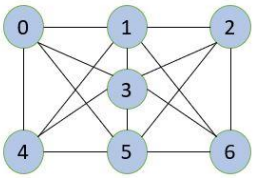
Queue:
1,4,0
Visit sequence:
2,6,5,3



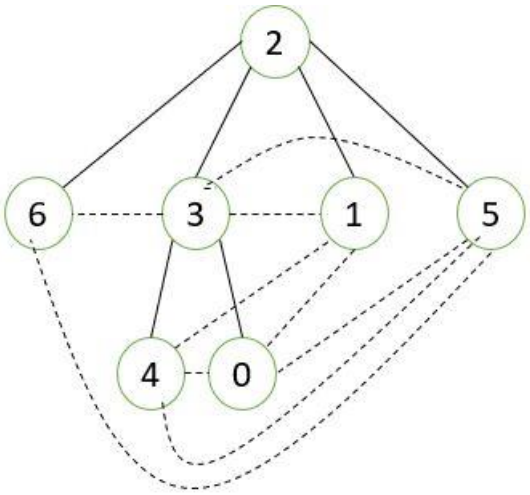
Queue:
4,0
Visit sequence:
2,6,3,1



Queue:
0
Visit sequence:
2,6,3,1,4

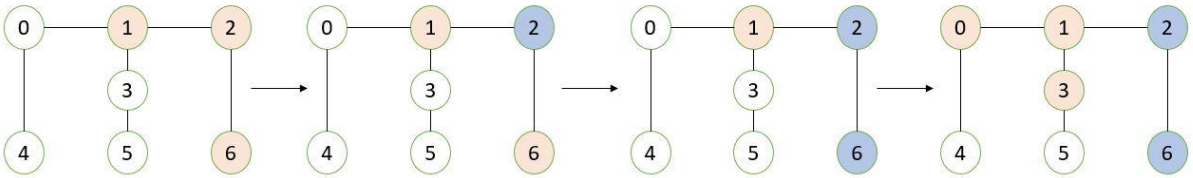


Queue:
null
Visit sequence:
2,6,3,1,4,0



BFS TREE

GRAPH B)

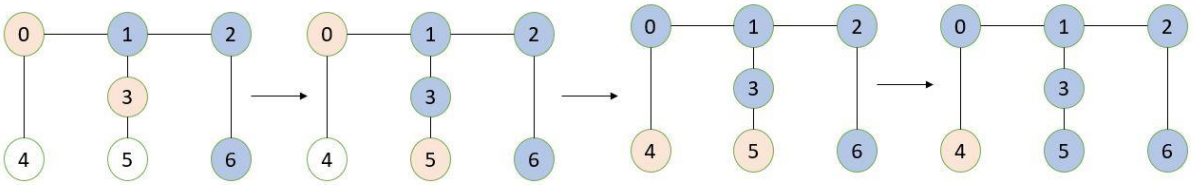


Queue:
6,1
Visit sequence:

Queue:
6,1
Visit sequence:
2

Queue:
1
Visit sequence:
2,6

Queue:
1,3,0
Visit sequence:
2,6

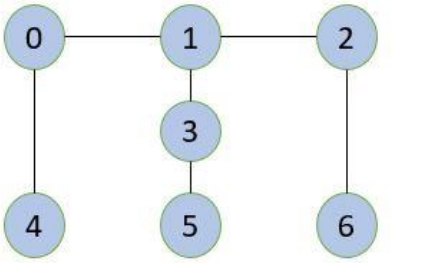


Queue:
3,0
Visit sequence:
2,6,1

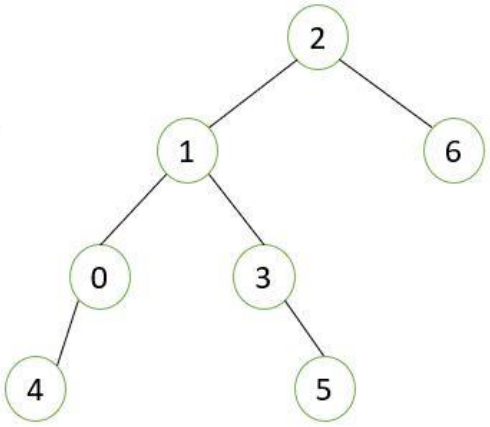
Queue:
0,5
Visit sequence:
2,6,1,3

Queue:
5,4
Visit sequence:
2,6,1,3

Queue:
4
Visit sequence:
2,6,1,3,5



Queue:
null
Visit sequence:
2,6,1,3,5,4



BFS TREE