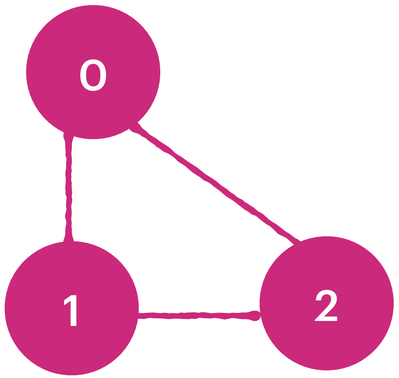
Design Graph :

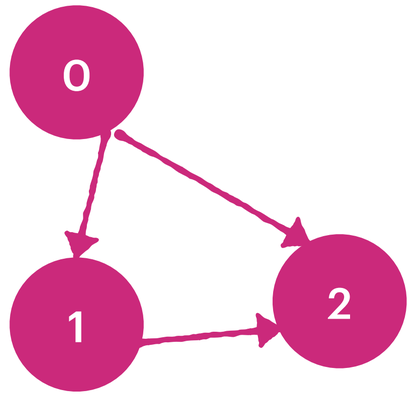
- AddVertex → O(1)
- AddEdge → O(1)
- RemoveEdge → O(1)
- Remove Vertex → O(V)
- Print → O(V+E)

n = 3, edges = [[0,1],[0,2],[1,2],[2,1],[2,0],[1,0]]

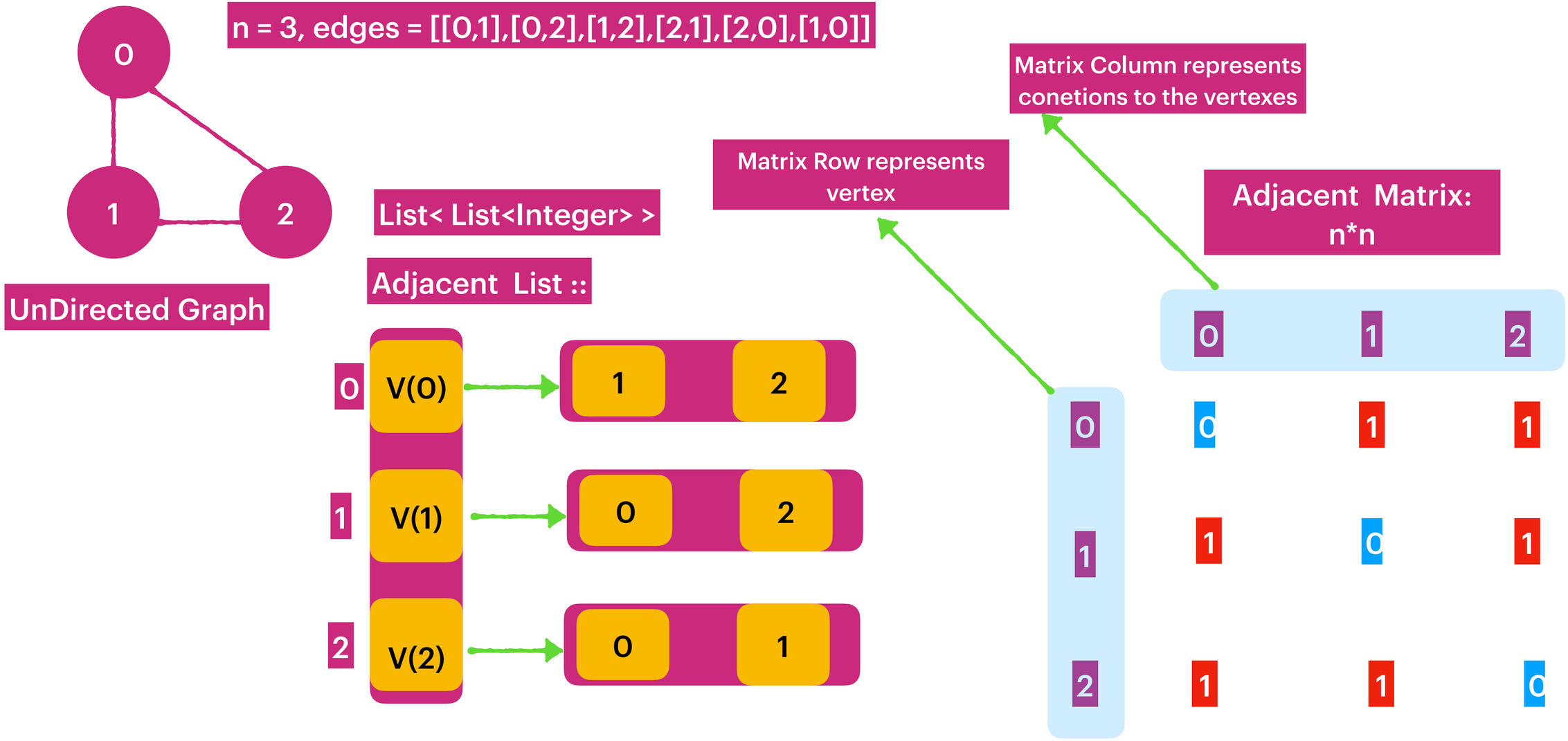


UnDirected Graph

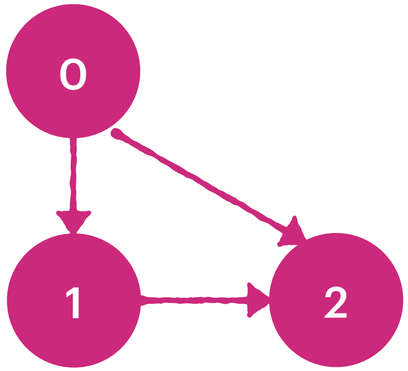
n = 3, edges = [[0,1],[1,2],[0,2]]



Directed Graph


n = 3, edges = [[0,1],[0,2],[1,2],[2,1],[2,0],[1,0]]
0
1
2
UnDirected Graph
List< List<Integer> >
Adjacent List ::
0
V(0)
1
2
1
V(1)
0
2
2
V(2)
0
1
Matrix Row represents vertex
Matrix Column represents conetions to the vertexes
Adjacent Matrix: n*n
0
1
2
0
0
1
1
1
1
0
1
2
1
1
0

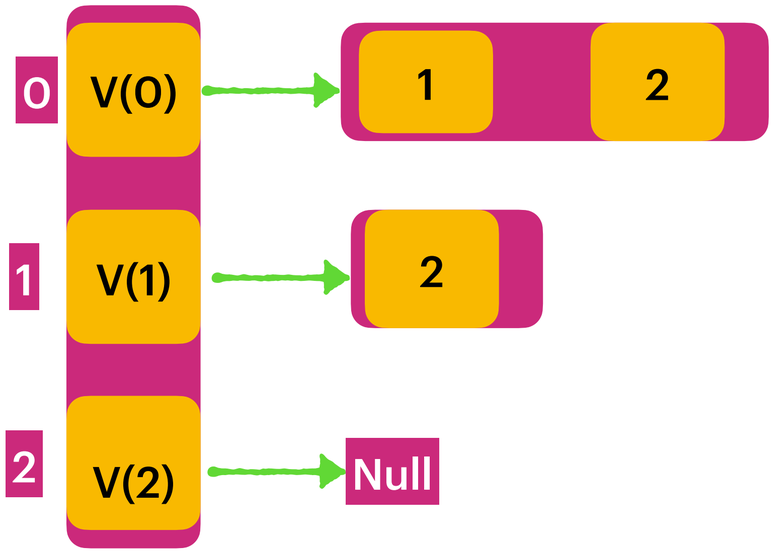
n = 3, edges = [[0,1],[1,2],[0,2]]



Directed Graph

List< List<Integer> >

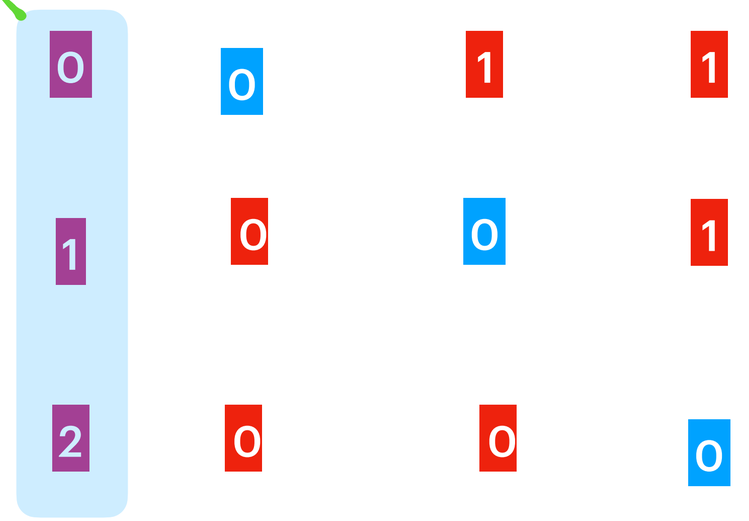
Adjacent List ::



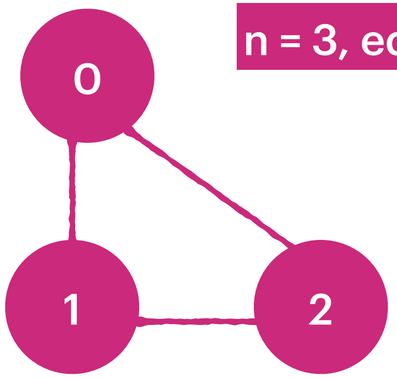
Matrix Row represents vertex

Matrix Column represents conetions to the vertexes

Adjacent Matrix: n*n



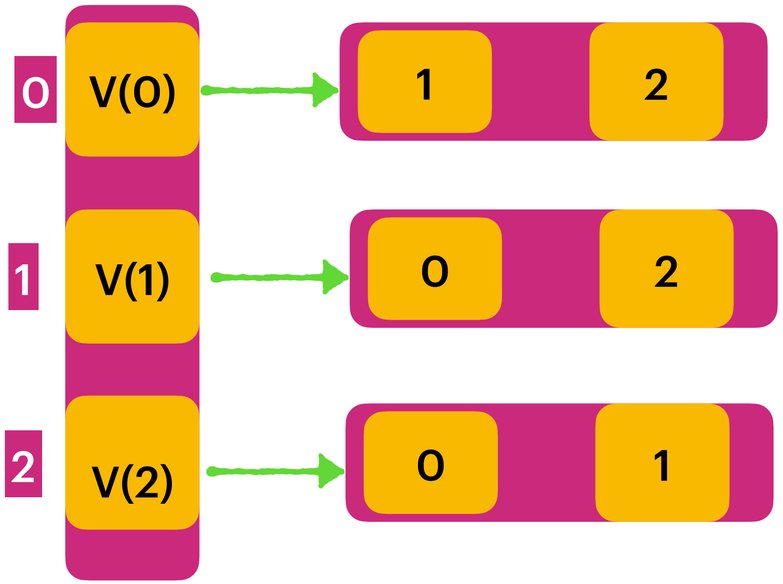
	Adjacent List	Adjacent Matrix
Add Vertex	O(1)	O(n^2)
Remove Vertex	O(V+E)	O(n^2
Add Edge	O(1)	O(1)
Remove Edge	O(E)	O(1)



n = 3, edges = [[0,1],[0,2],[1,2],[2,1],[2,0],[1,0]]

List< LinkedHashSet >

Adjecent Hashing




```
removeVertex(2) :
for(v:o to  Vn )
{
if(list.get(v).contains(2))
{
list.get(v).remove(2)
}
}
list.remove(2);
```

Hashing

Add Vertex	O(1)
Remove Vertex	O(V)
Add Edge	O(1)
Remove Edge	O(1)

Print O(V+E)