Data Structures

Unit II. Non Linear Data Structures

4. Trees and Graphs

Unit II. Objective

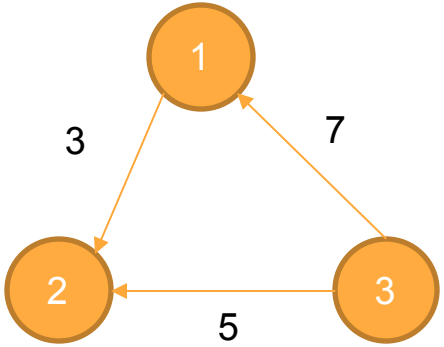
The student will build non linear structures for data storage and retrieval.

Learning Objectives

- Describe the basic characteristics of a Graph.

Implementation

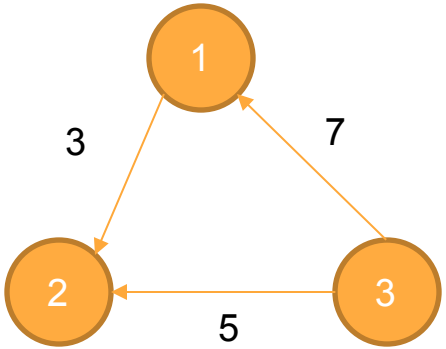
Static Implementation



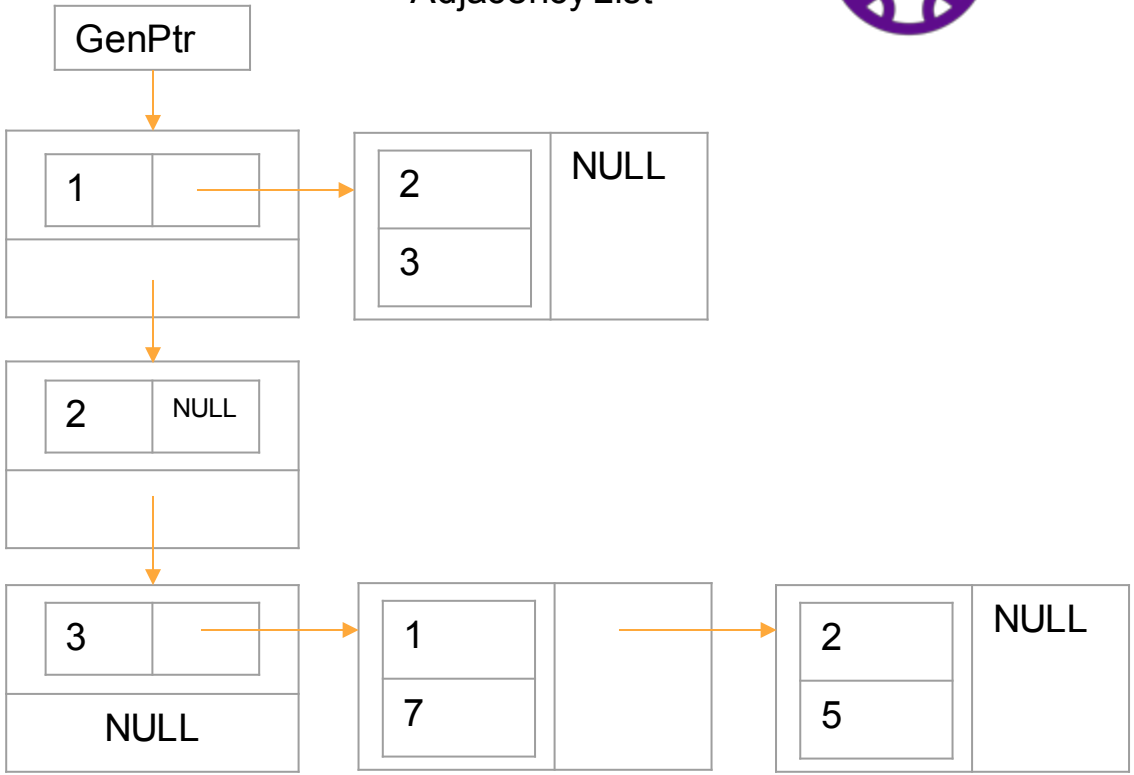
Adjacency Matrix

0	3	0
0	0	0
7	5	0

Dynamic Implementation

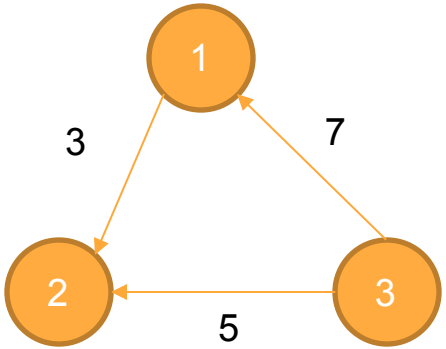


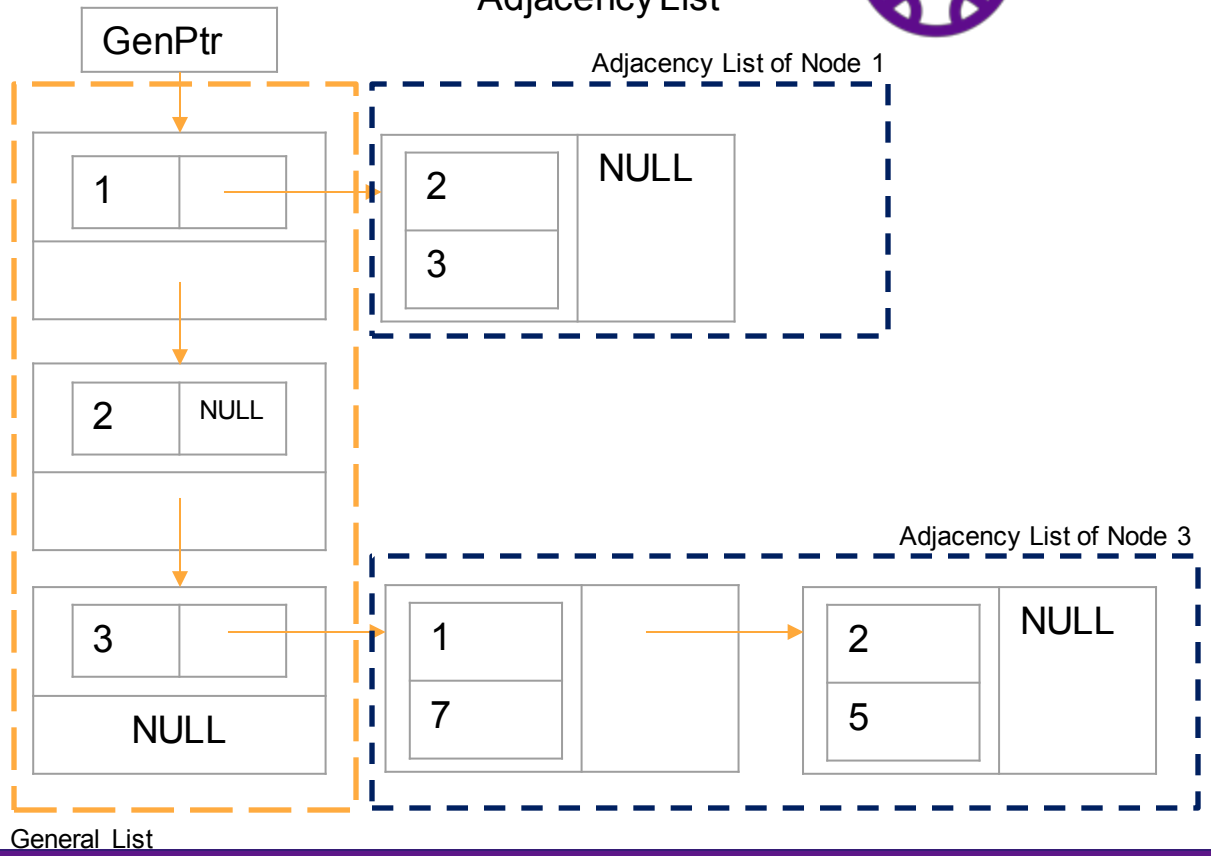
Adjacency List



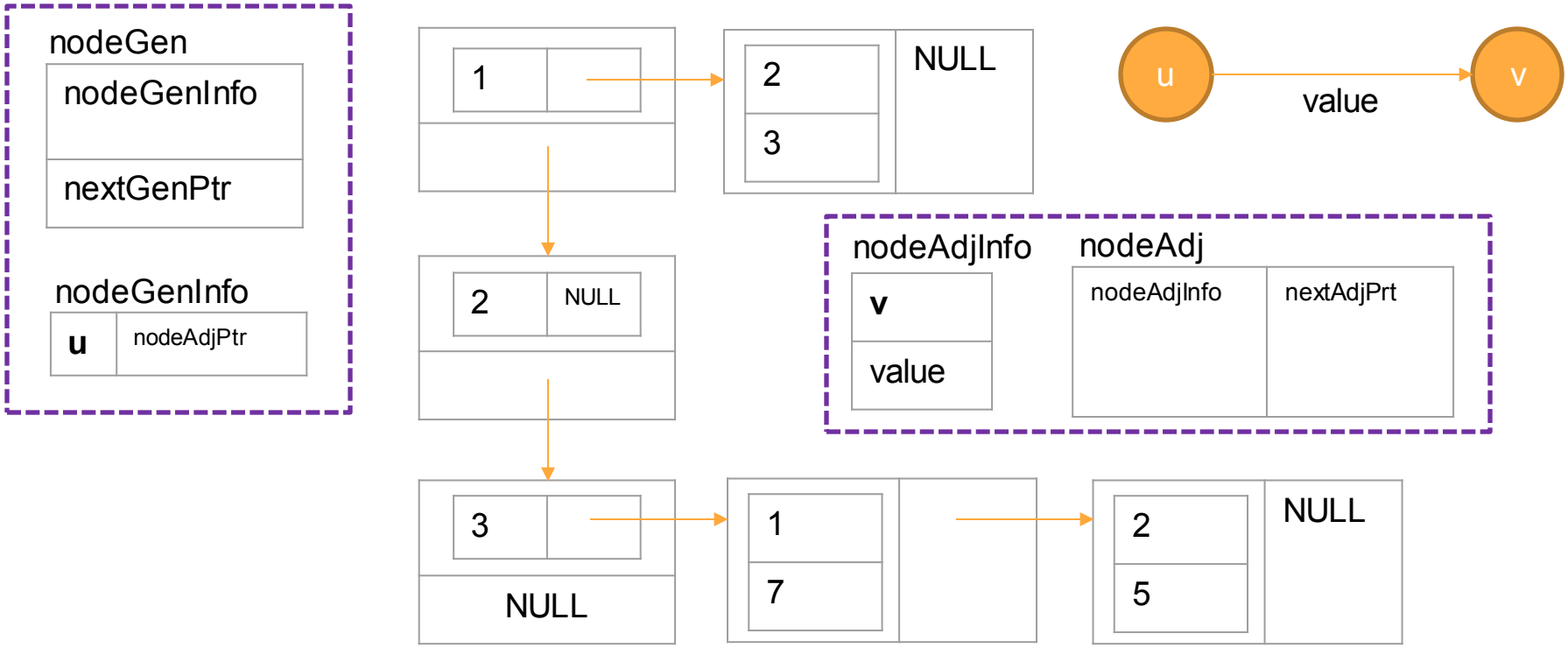
General List

Dynamic Implementation





Dynamic Implementation

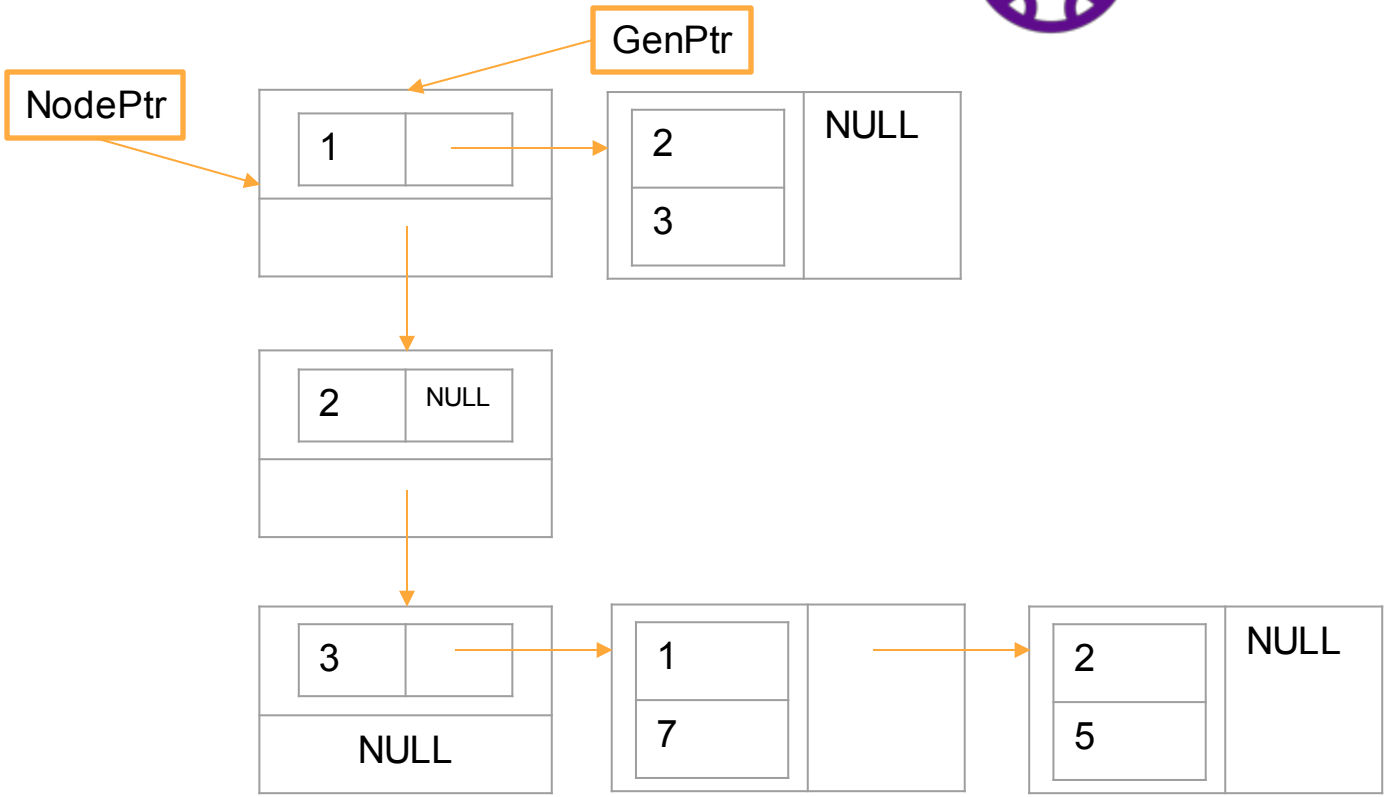


Searching a vertex

isVertex(GenPtr, 4)

1) Iterate over all nodes in General List until:

- NodePtr is NULL.
- Value is found.

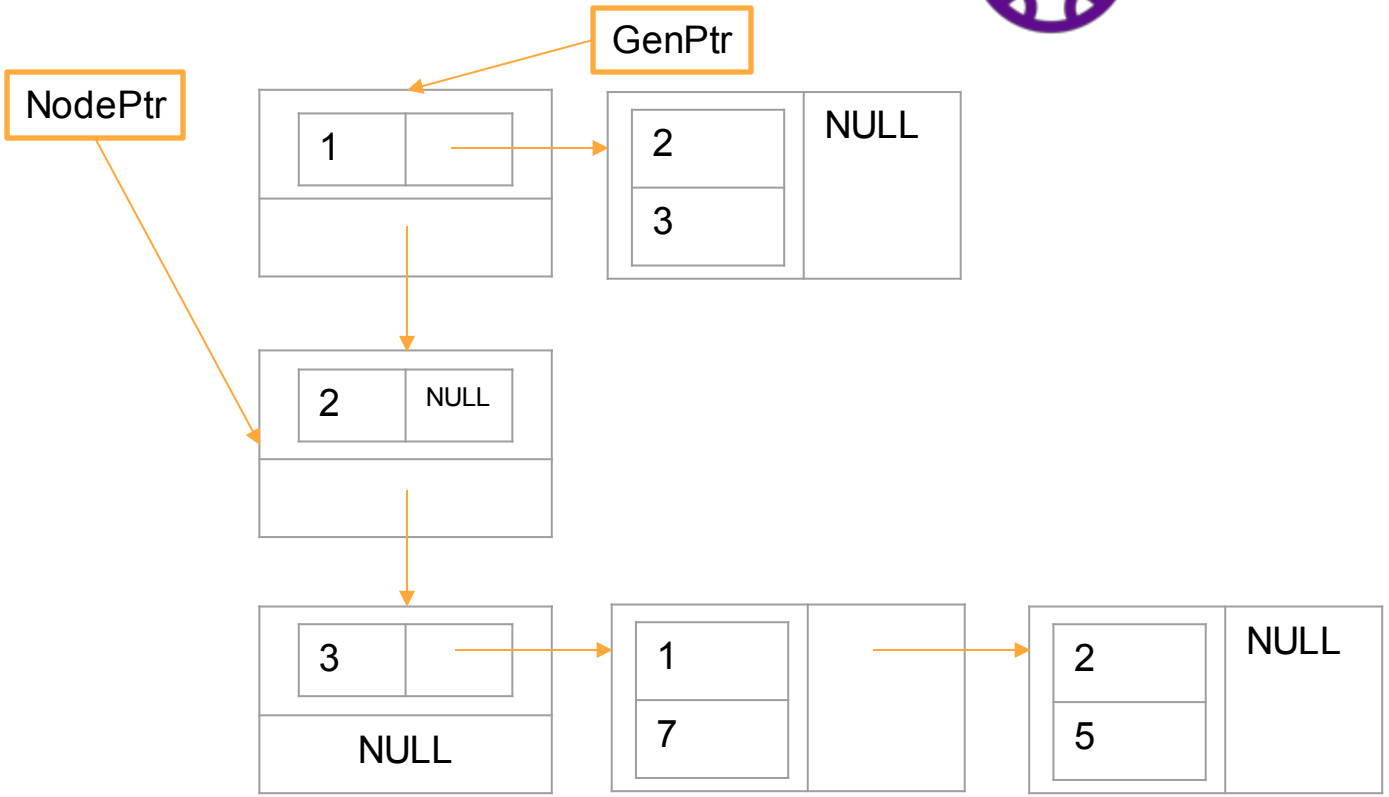


Searching a vertex

isVertex(GenPtr, 4)

1) Iterate over all nodes in General List until:

- NodePtr is NULL.
- Value is found.

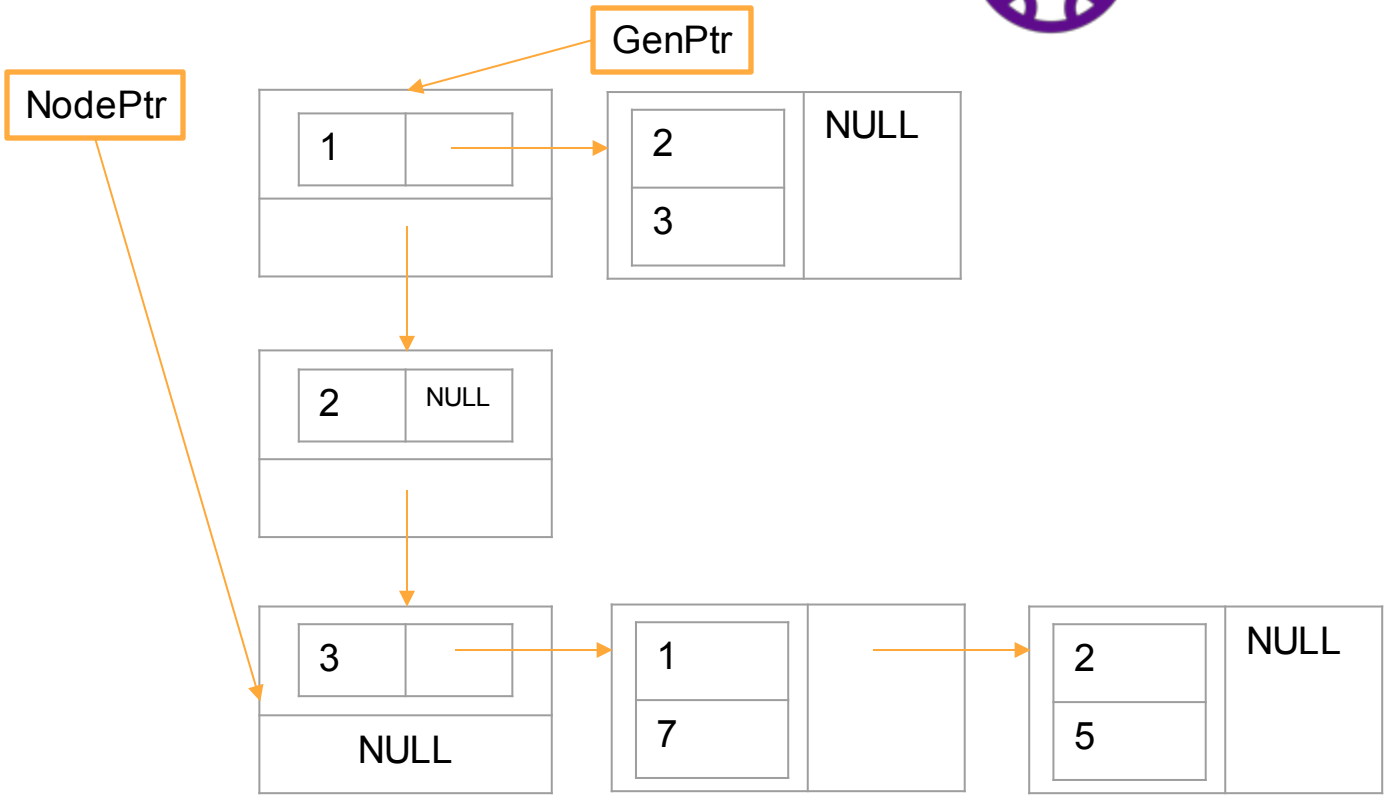


Searching a vertex

isVertex(GenPtr, 4)

1) Iterate over all nodes in General List until:

- NodePtr is NULL.
- Value is found.

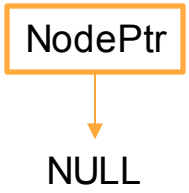


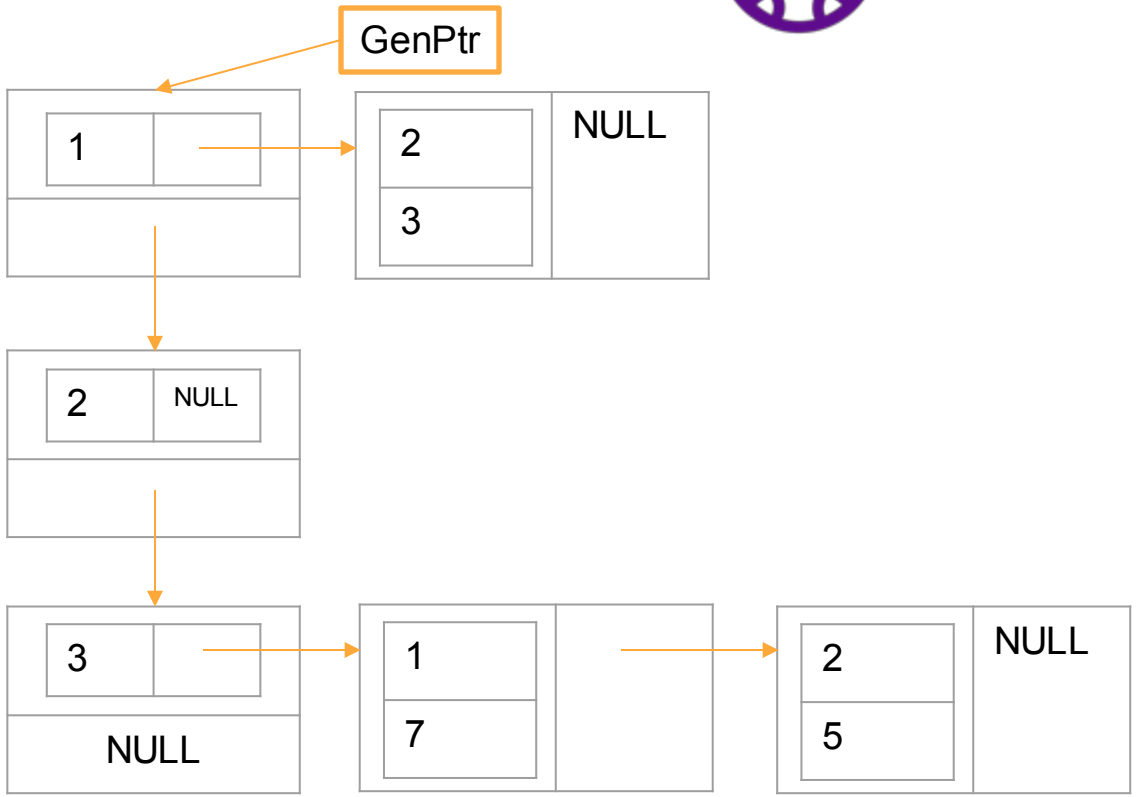
Searching a vertex

isVertex(GenPtr, 4)

1) Iterate over all nodes in General List until:

- NodePtr is NULL.
- Value is found.



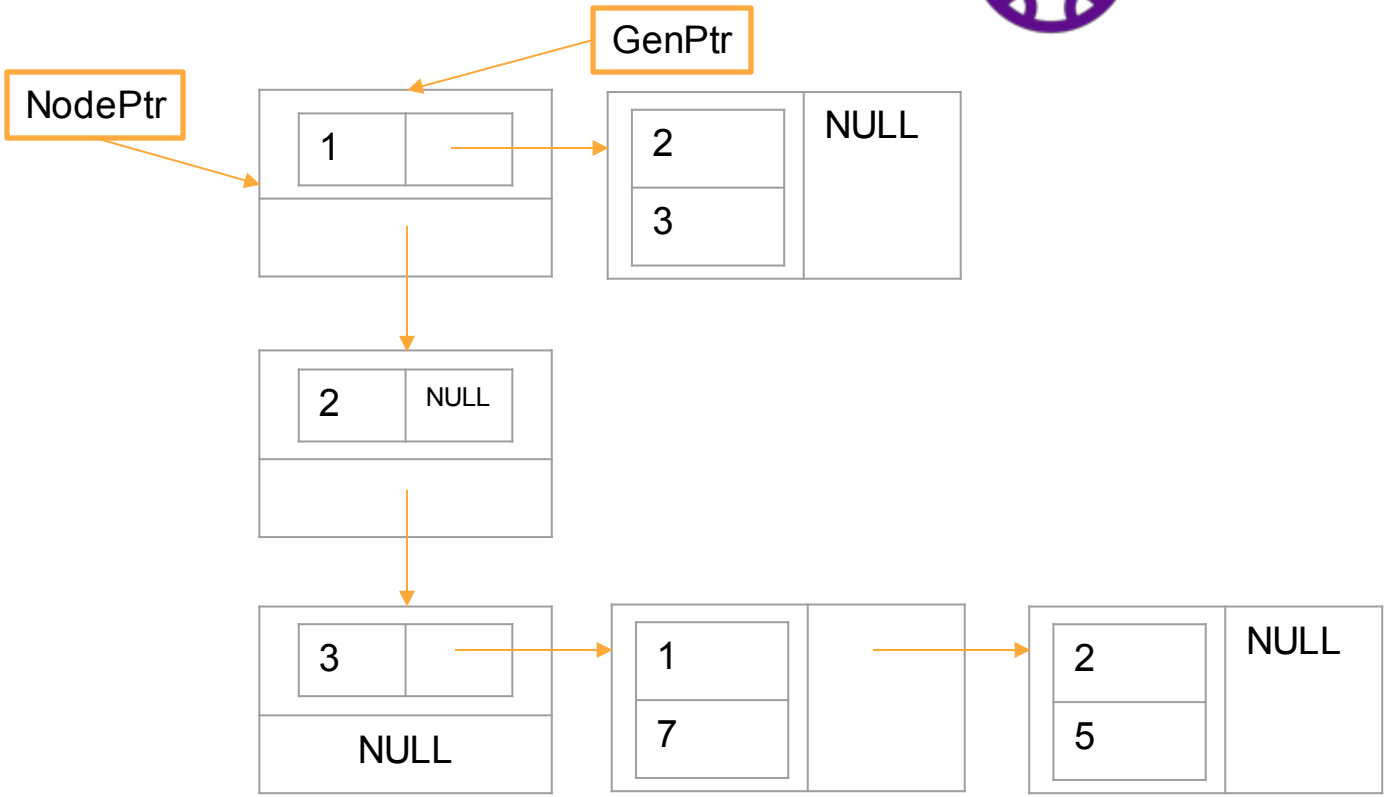


Searching a vertex

isVertex(GenPtr, 2)

1) Iterate over all nodes in General List until:

- NodePtr is NULL.
- Value is found.

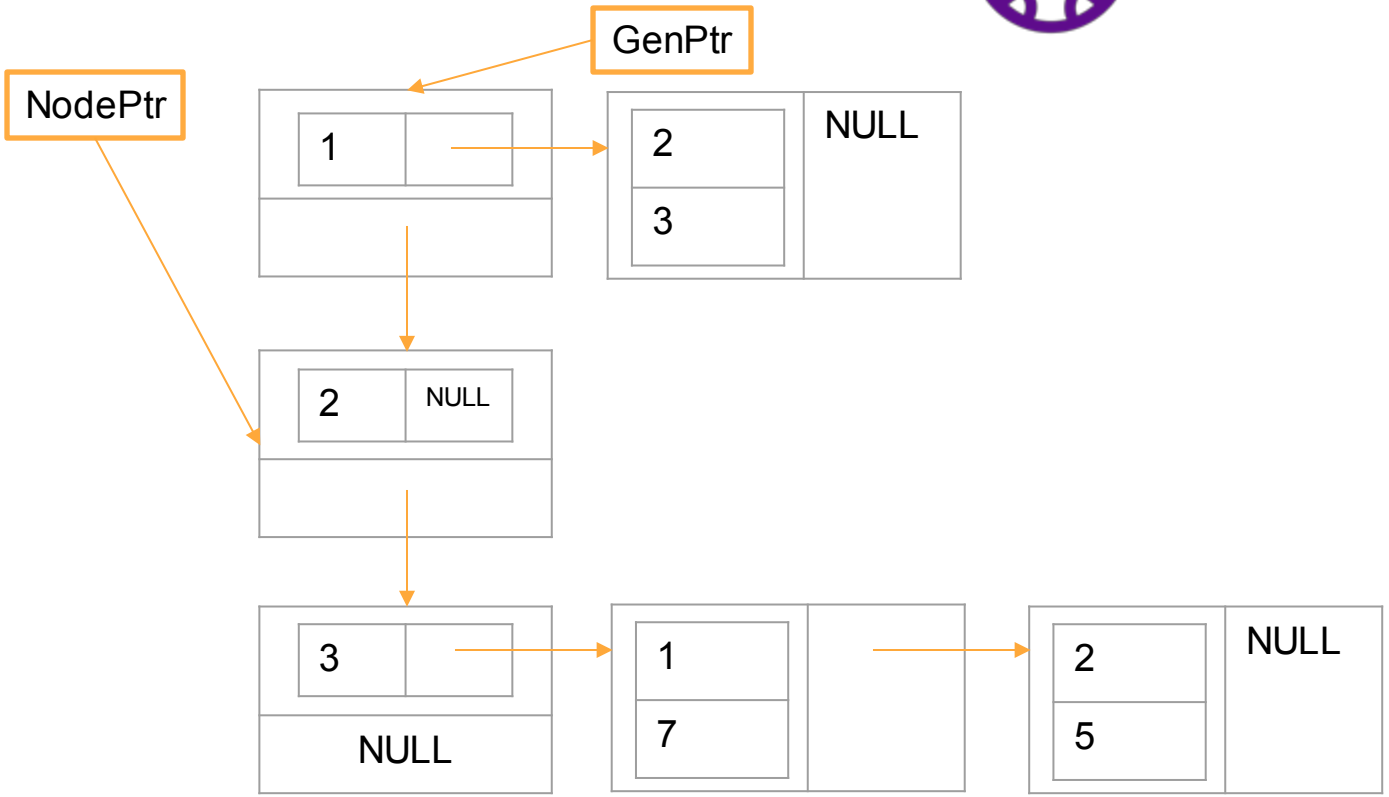


Searching a vertex

isVertex(GenPtr, 2)

1) Iterate over all nodes in General List until:

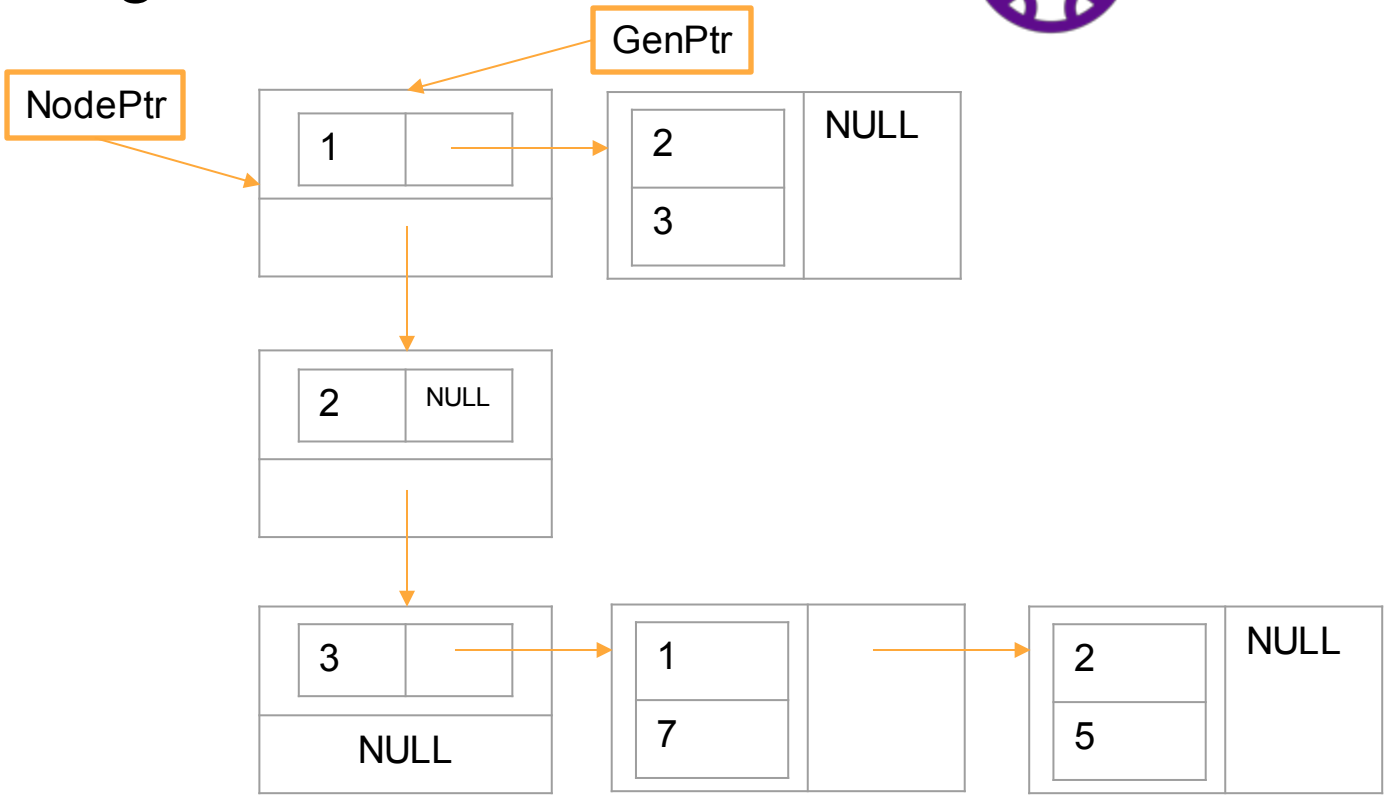
- NodePtr is NULL.
- Value is found.



Searching an edge

isEdge(GenPtr, 1, 4)

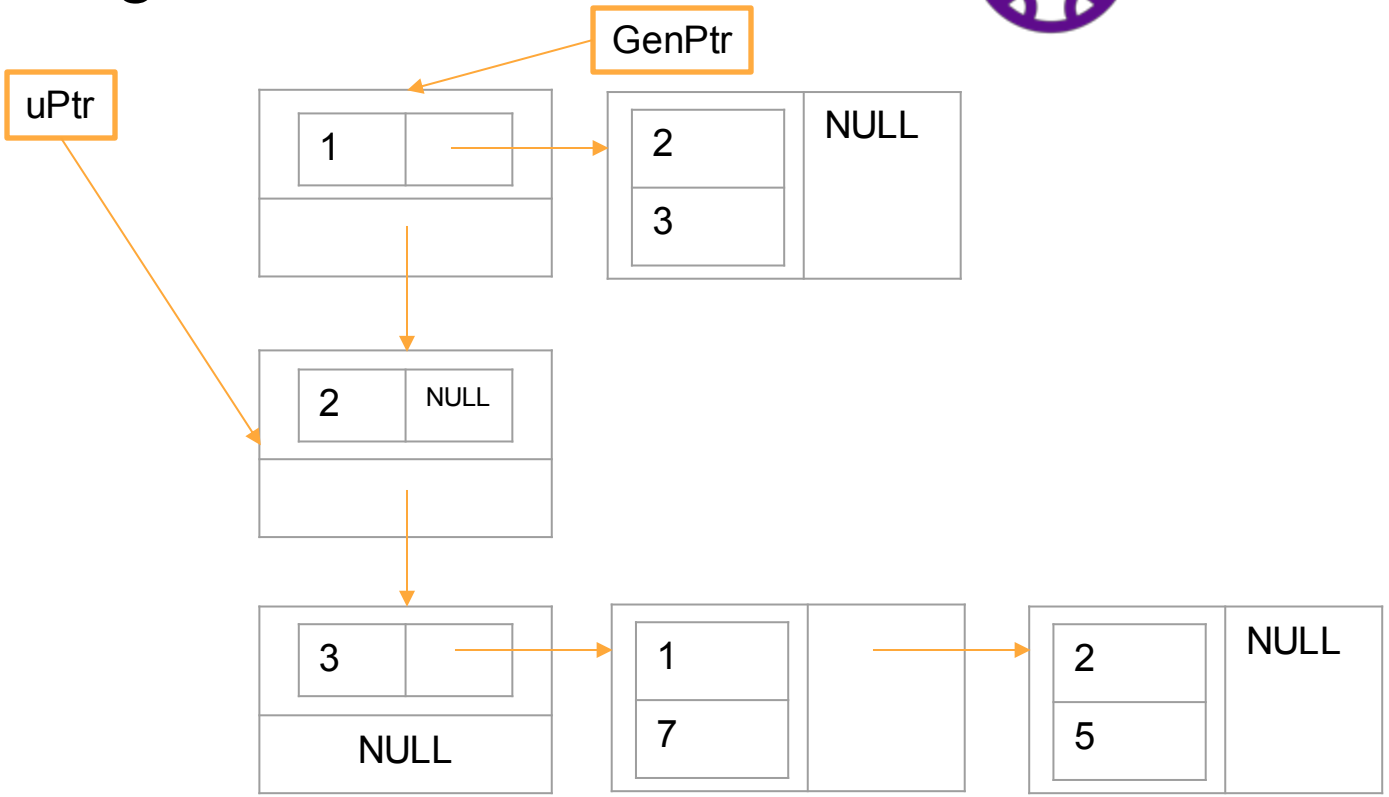
1) Check if u and v are in the general list.



Searching an edge

isEdge(GenPtr, 2, 1)

1) Check if u and v are in the general list.
2) Check if Adj List is not empty.

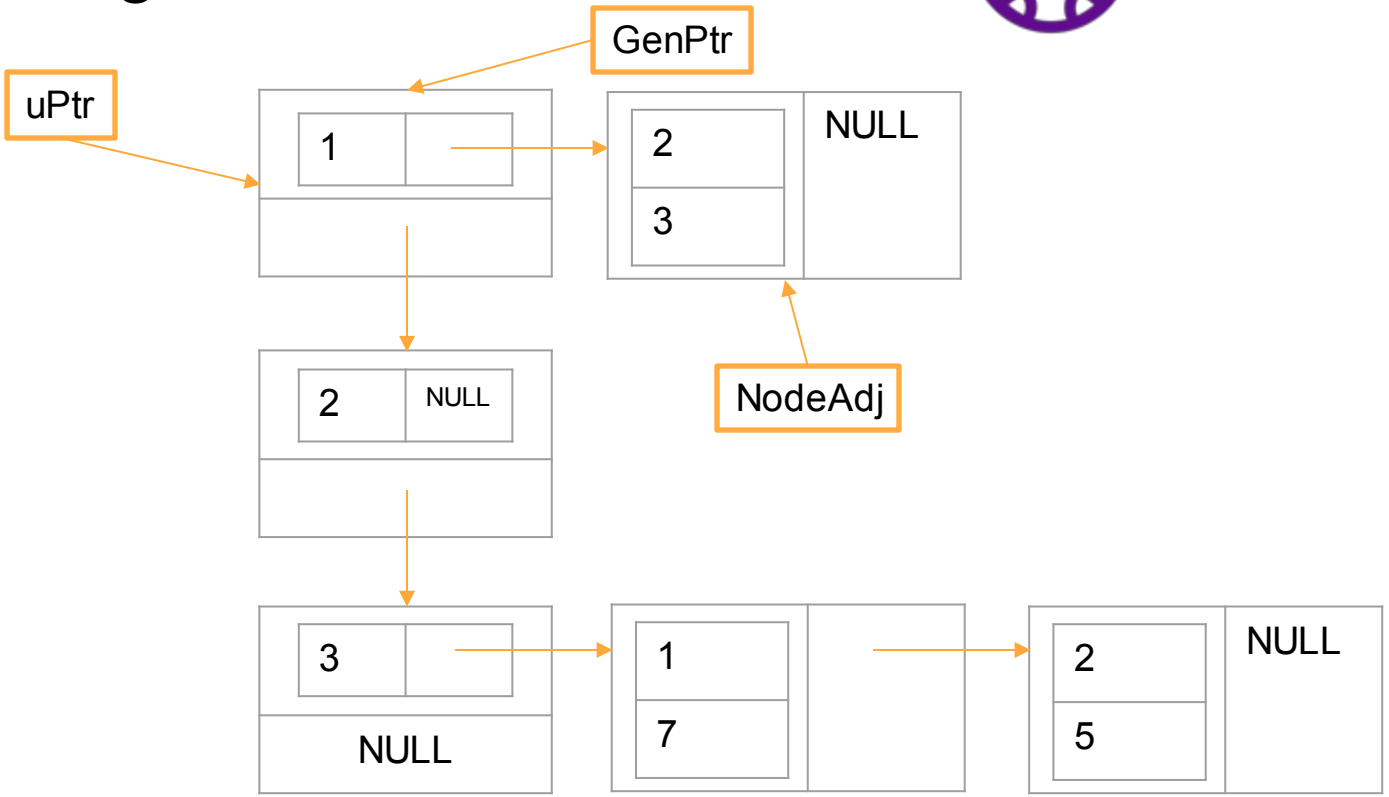


Searching an edge

isEdge(GenPtr, 1, 3)

1) Check if u and v are in the general list.
2) Check if Adj List is not empty.
3) Iterate over all element in Adj List until:

- AdjPtr is NULL.
- Value(v) is found.

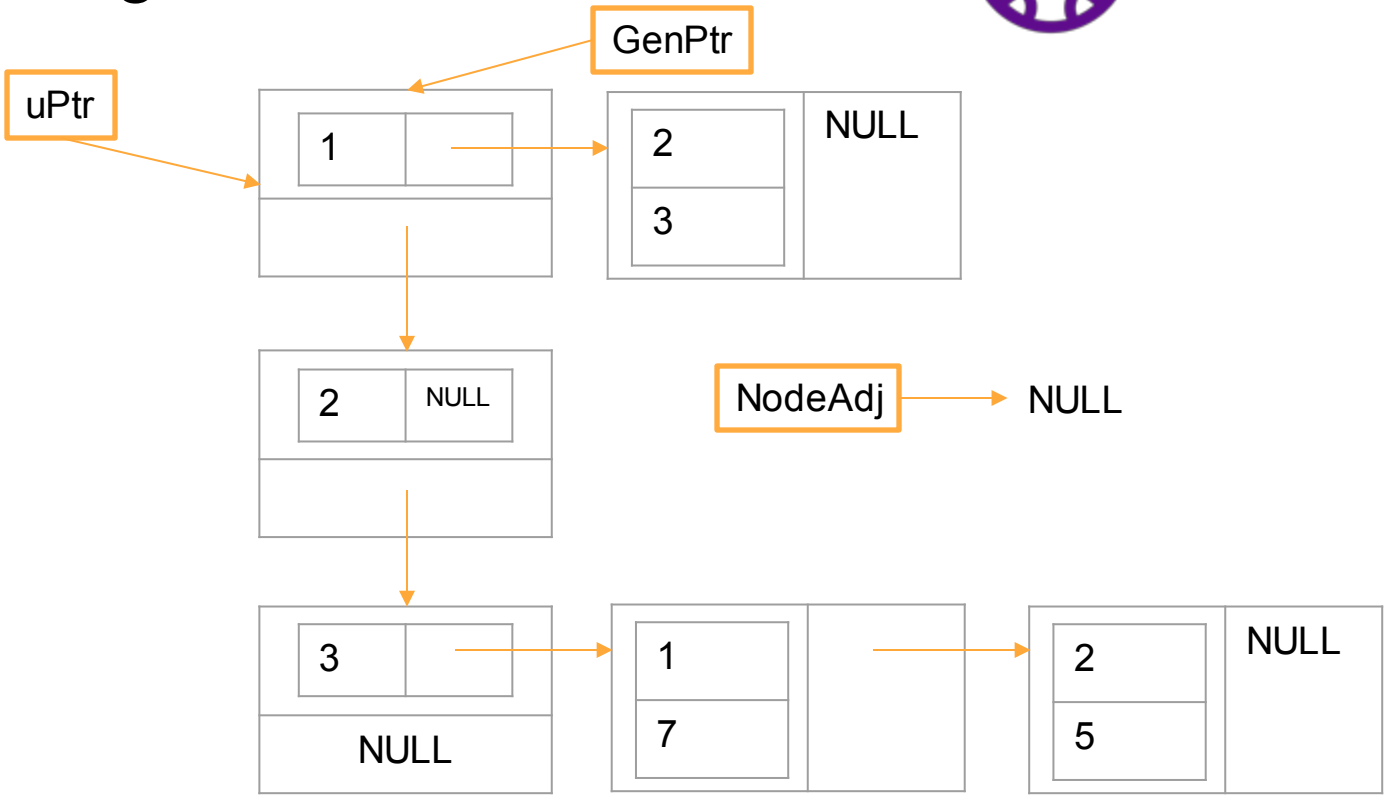


Searching an edge

isEdge(GenPtr, 1, 3)

1) Check if u and v are in the general list.
2) Check if Adj List is not empty.
3) Iterate over all element in Adj List until:

- AdjPtr is NULL.
- Value(v) is found.

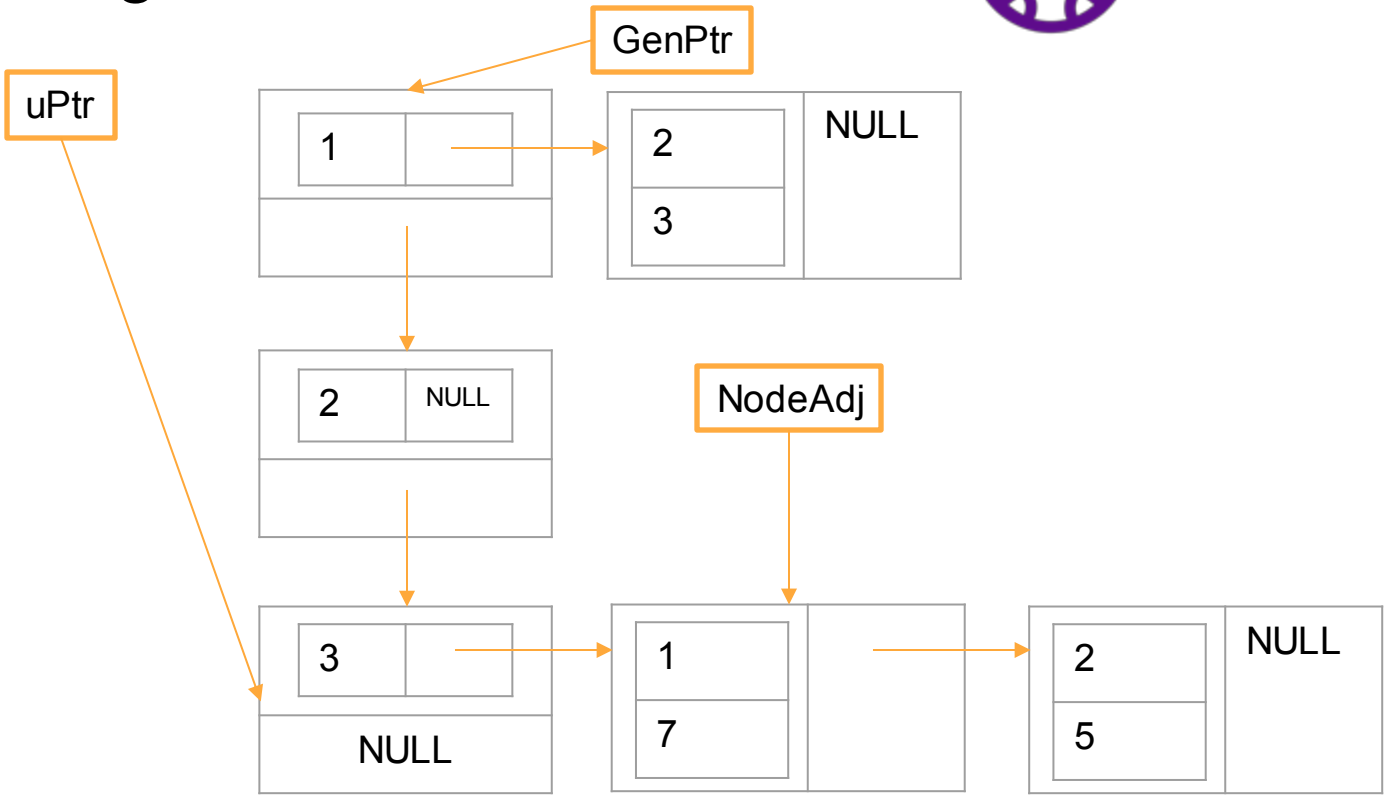


Searching an edge

isEdge(GenPtr, 3, 2)

1) Check if u and v are in the general list.
2) Check if Adj List is not empty.
3) Iterate over all element in Adj List until:

- AdjPtr is NULL.
- Value(v) is found.

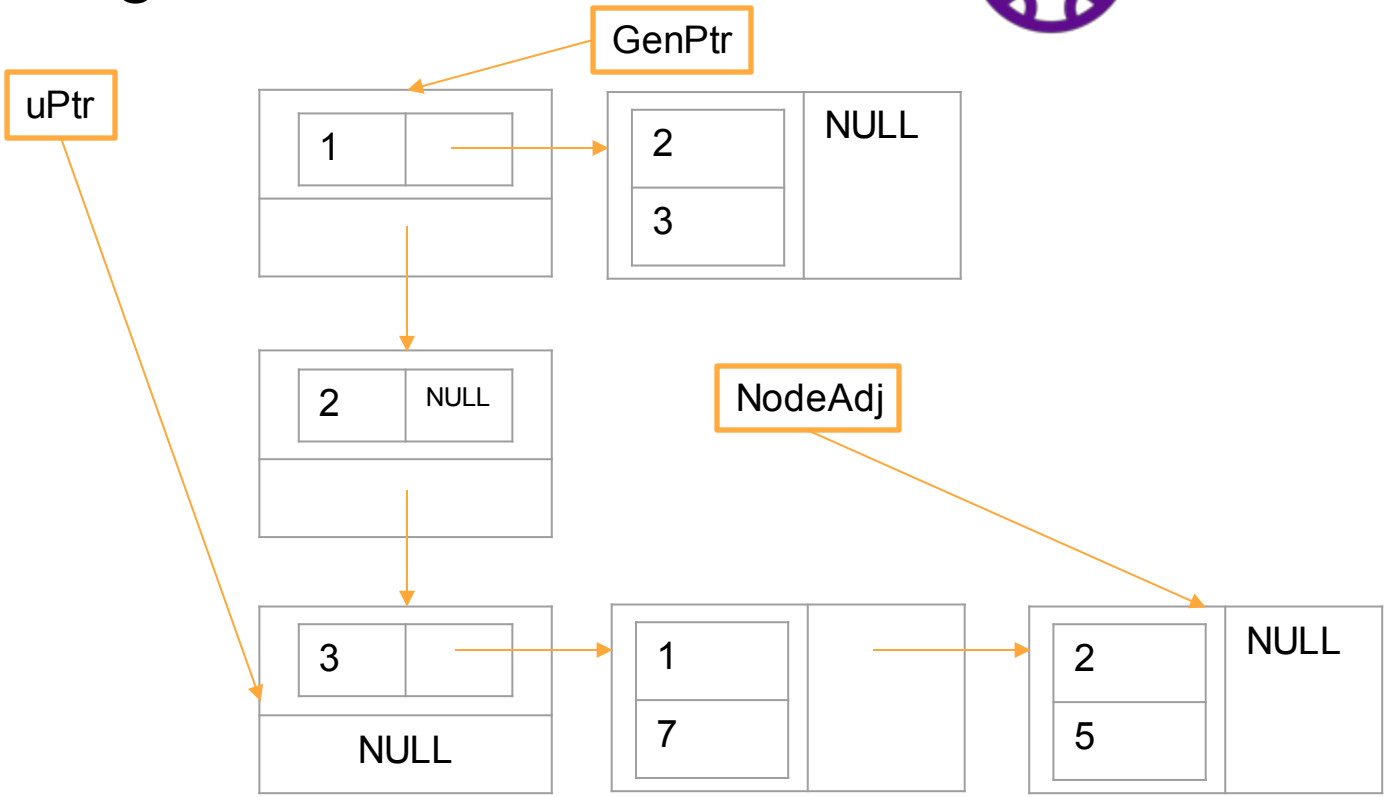


Searching an edge

isEdge(GenPtr, 3, 2)

1) Check if u and v are in the general list.
2) Check if Adj List is not empty.
3) Iterate over all element in Adj List until:

- AdjPtr is NULL.
- Value(v) is found.

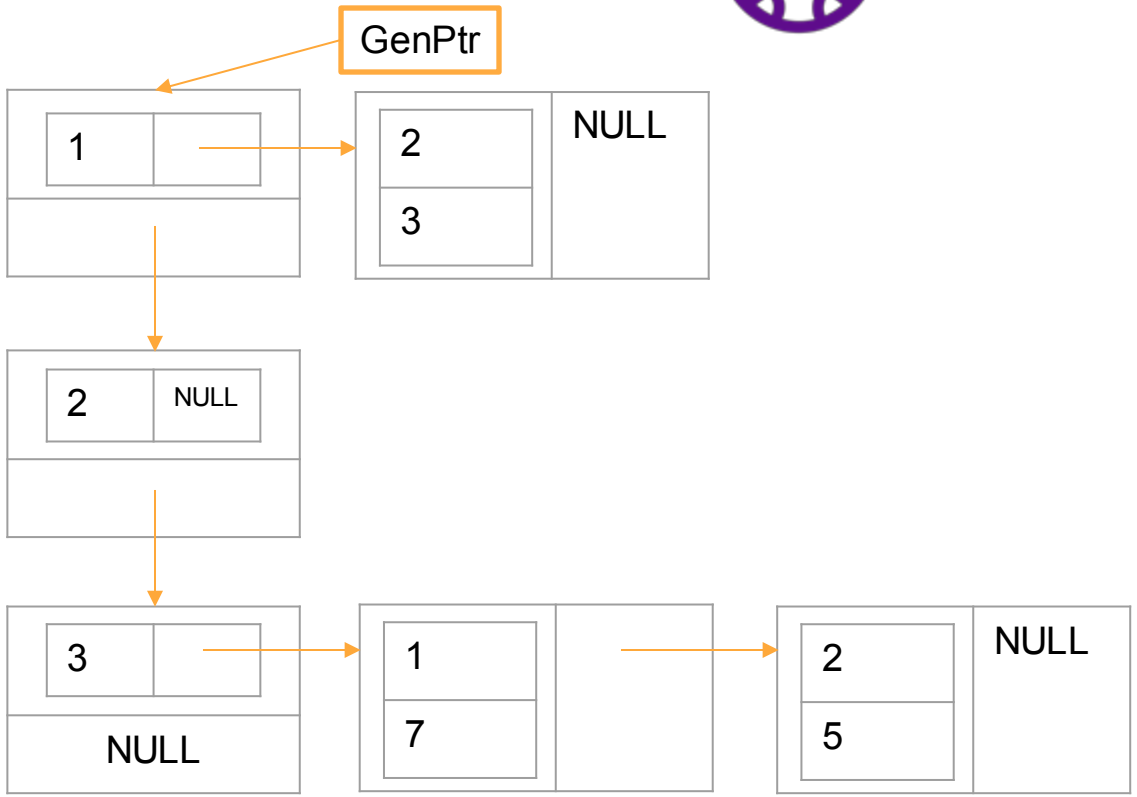


Add vertex

addVertex(&GenPtr, 3)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr



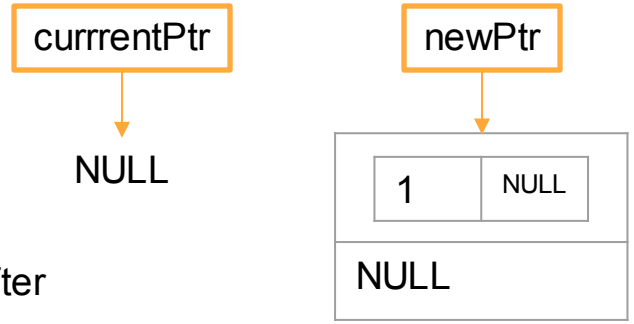
Add vertex

addVertex(&GenPtr, 1)

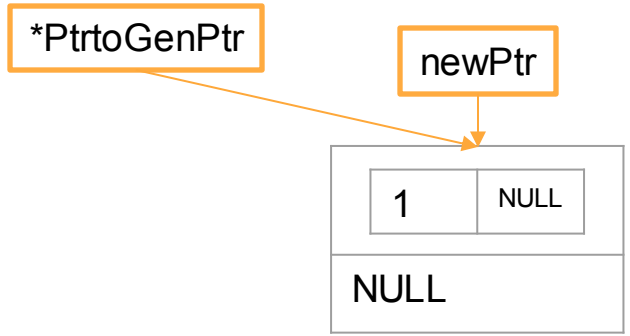
1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

Before



After

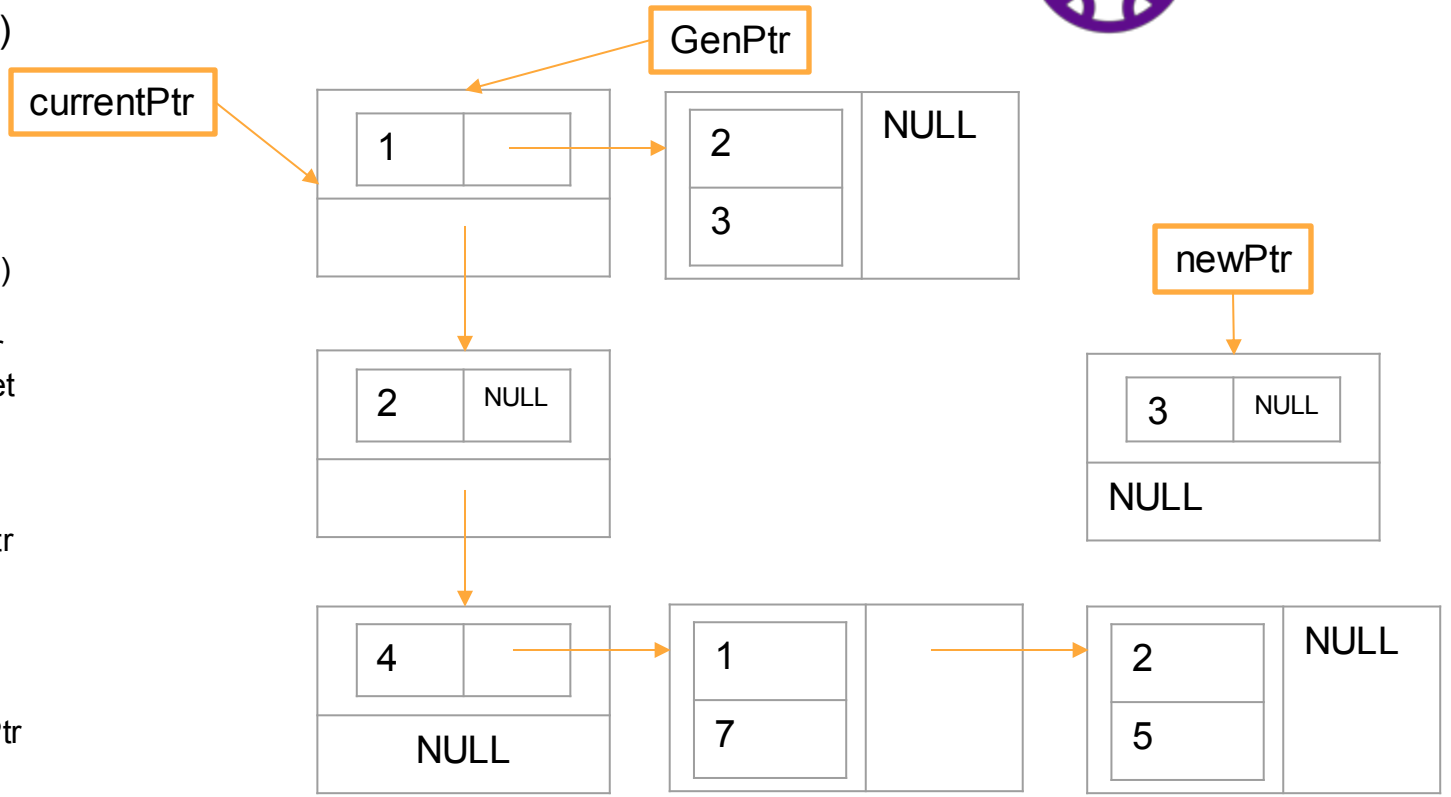


Add vertex

addVertex(&GenPtr, 3)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

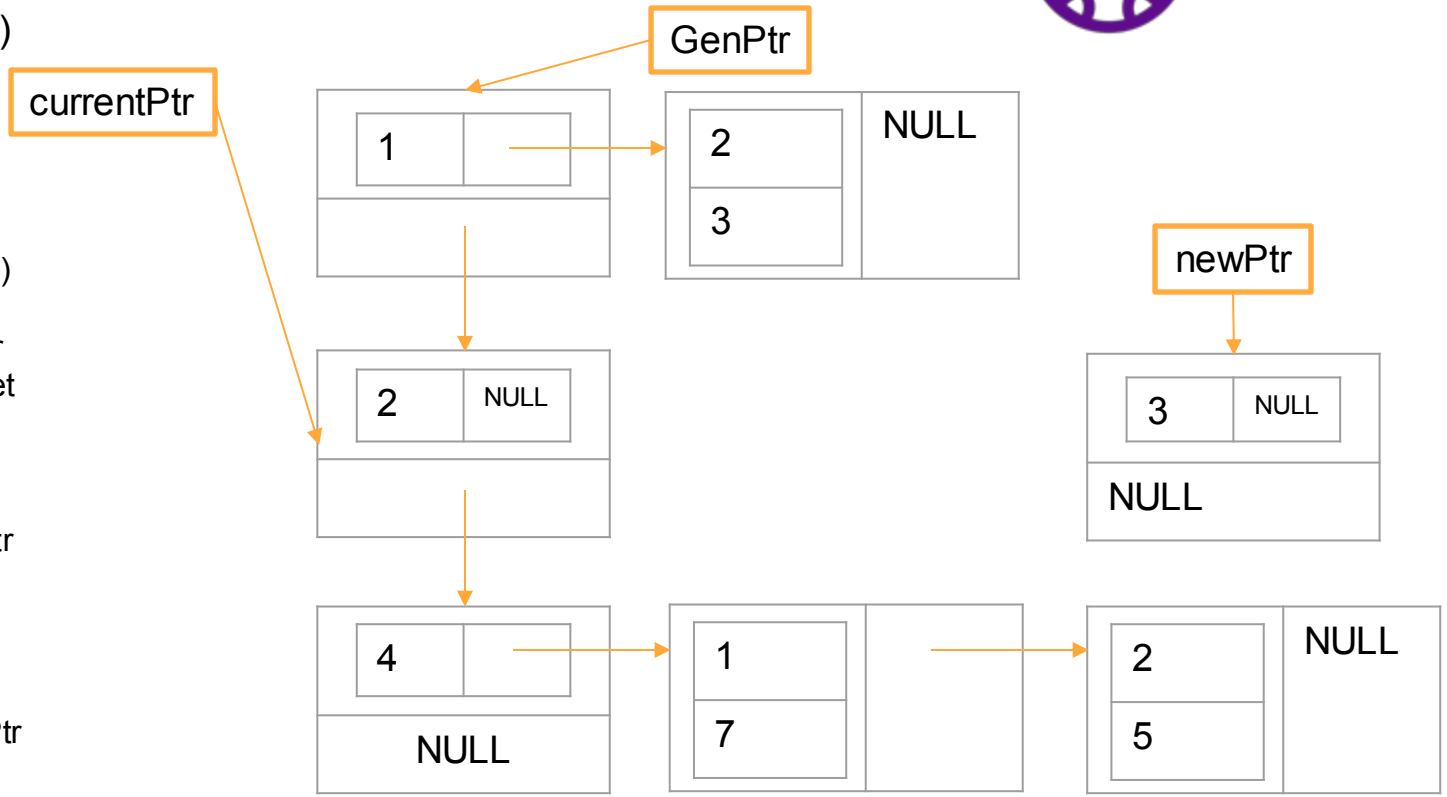


Add vertex

addVertex(&GenPtr, 3)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

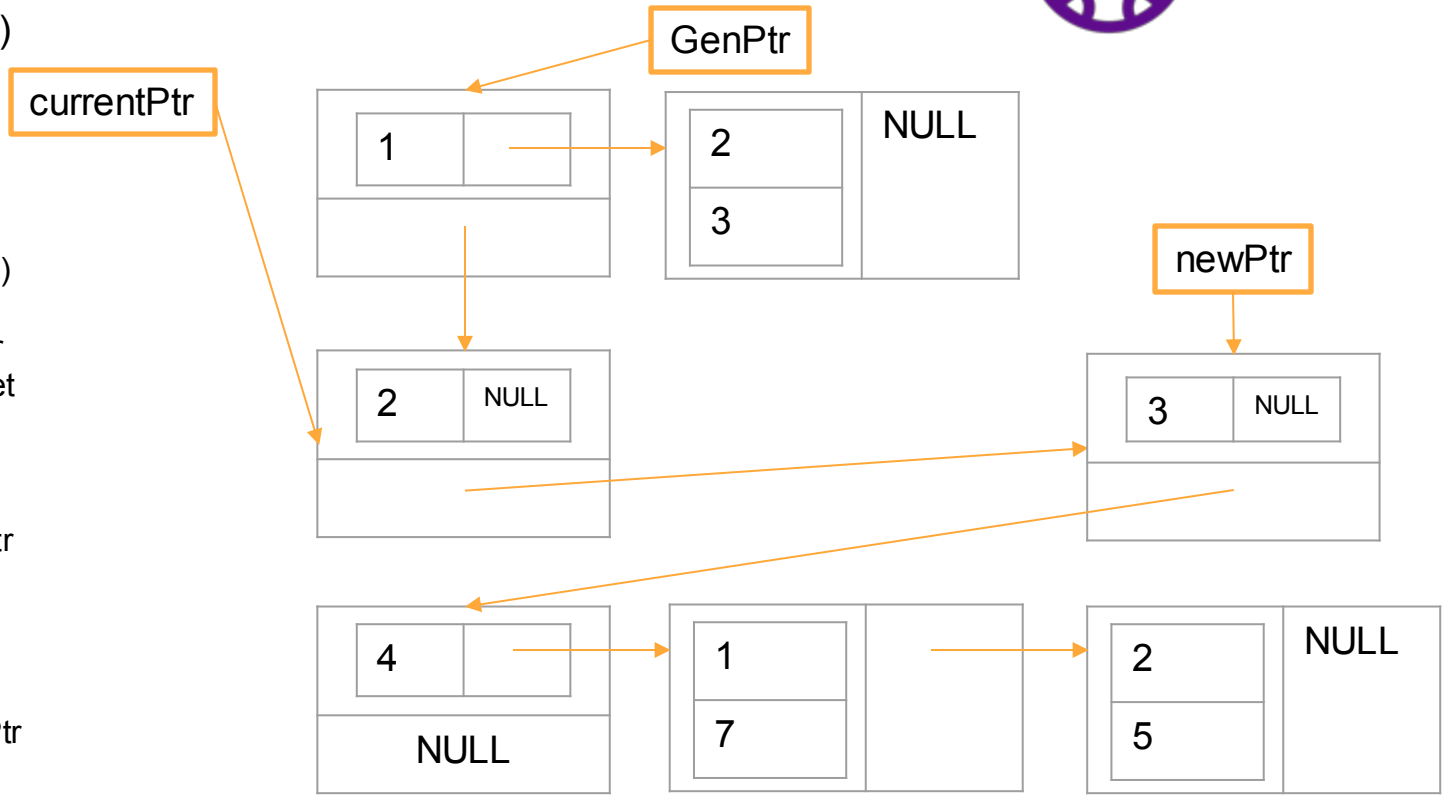


Add vertex

addVertex(&GenPtr, 3)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

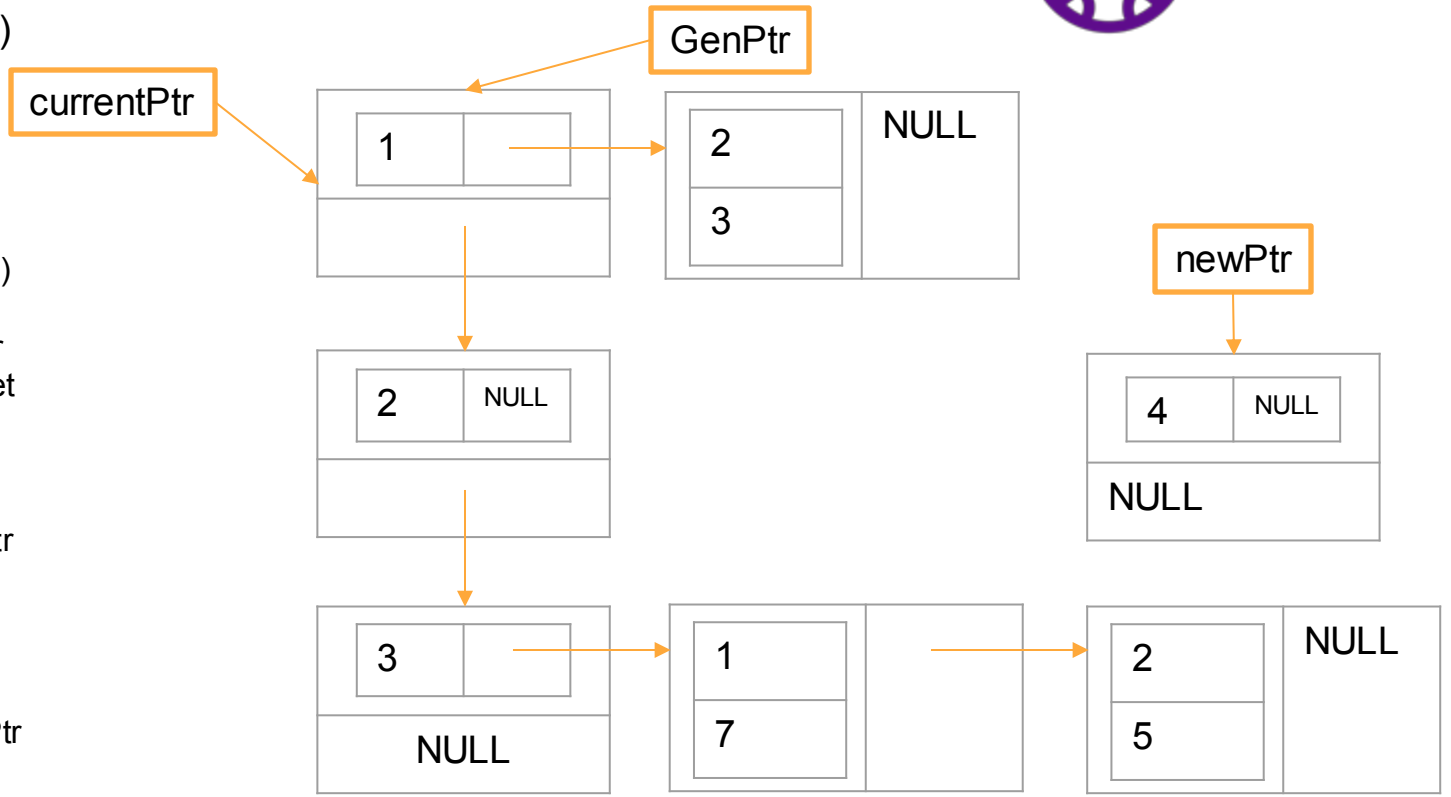


Add vertex

addVertex(&GenPtr, 4)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

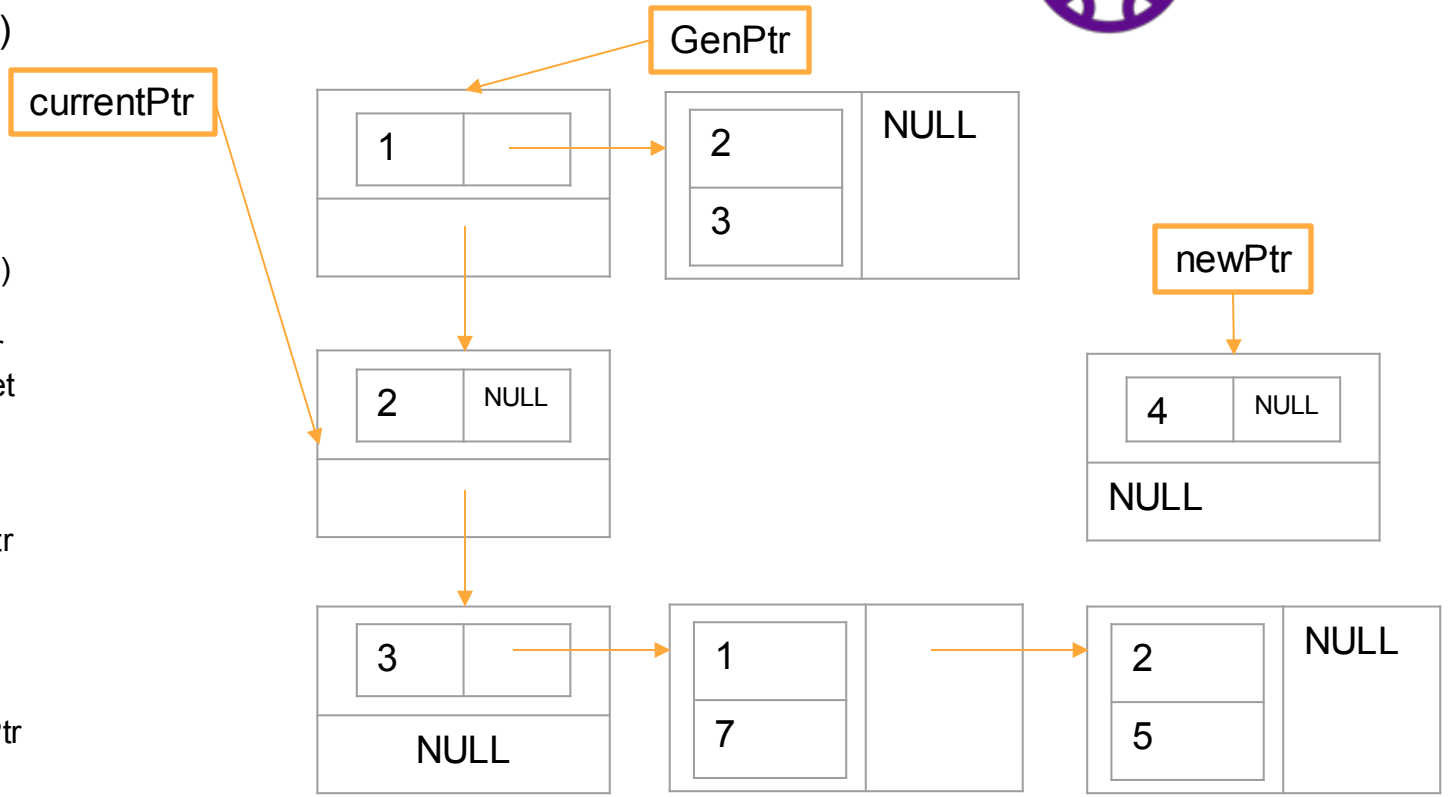


Add vertex

addVertex(&GenPtr, 4)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

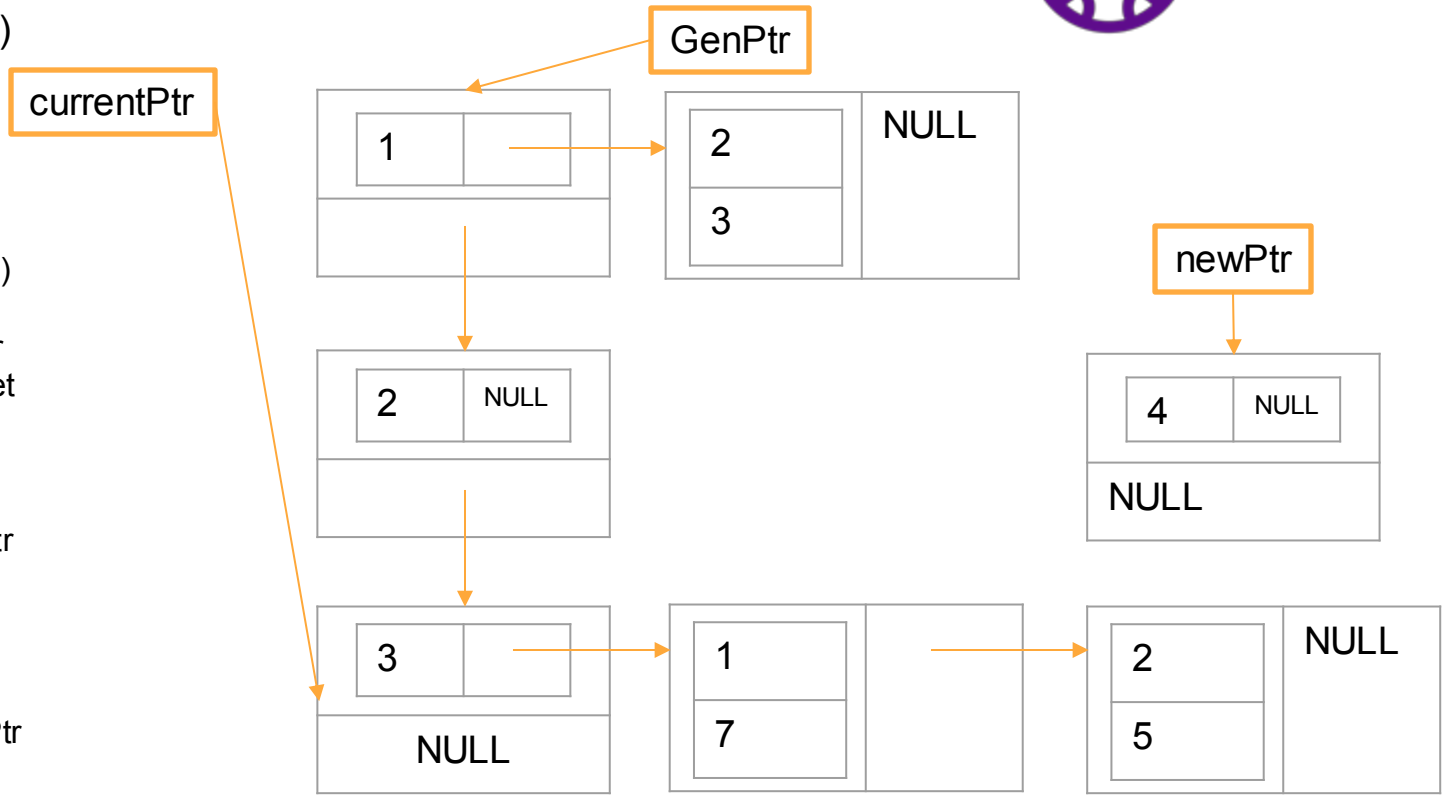


Add vertex

addVertex(&GenPtr, 4)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

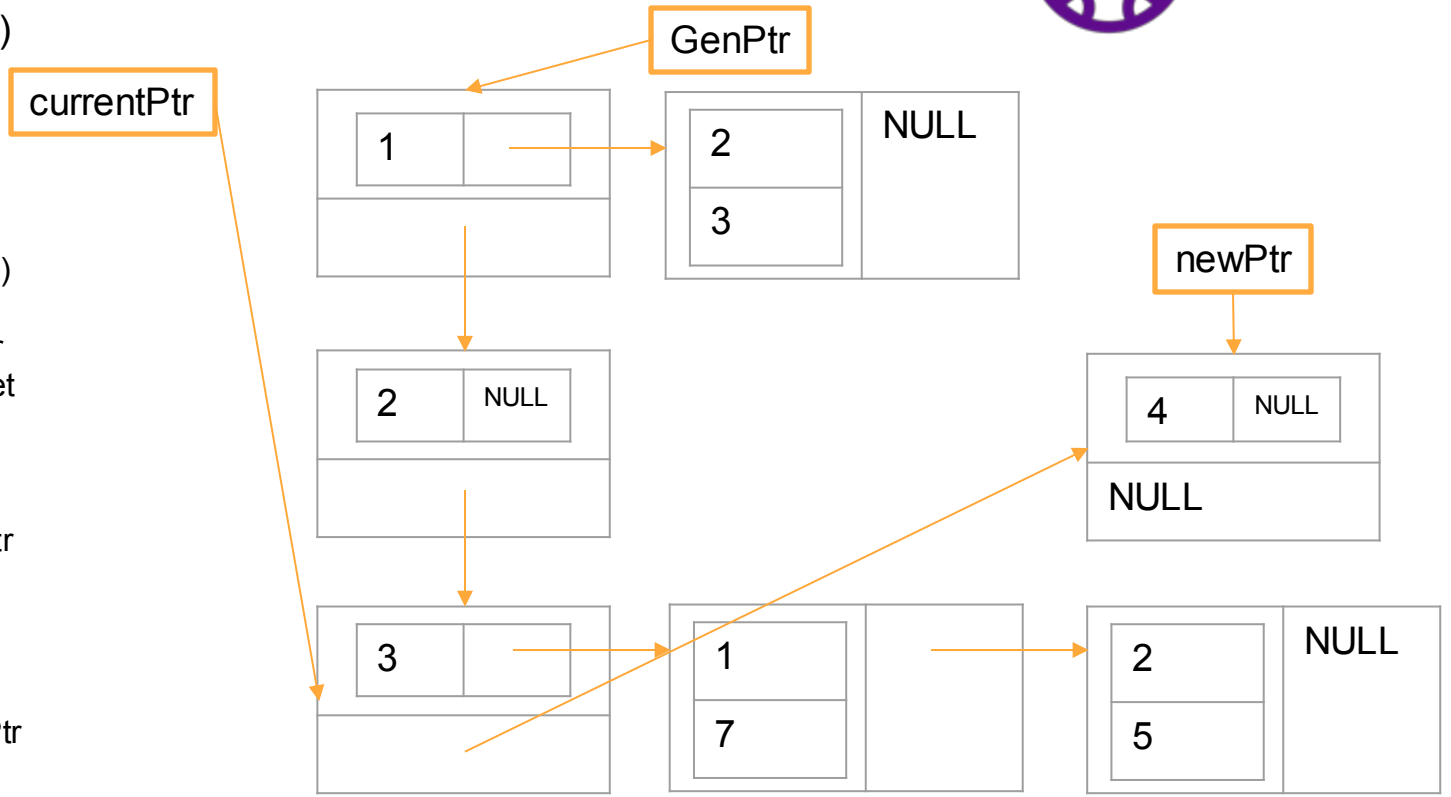


Add vertex

addVertex(&GenPtr, 4)

1) Check if u is in the general list.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
currentPtr == *PtrtoGenPtr
5) If currentPtr==NULL, set newPtr as the first node.
6) Iterate until:
- currentPtr is NULL.
- currentPtr->nextGenPtr has a vertex value greater than u.

7) Update:
- newPtr->nextGenPtr
- currentPtr->nextGentPtr

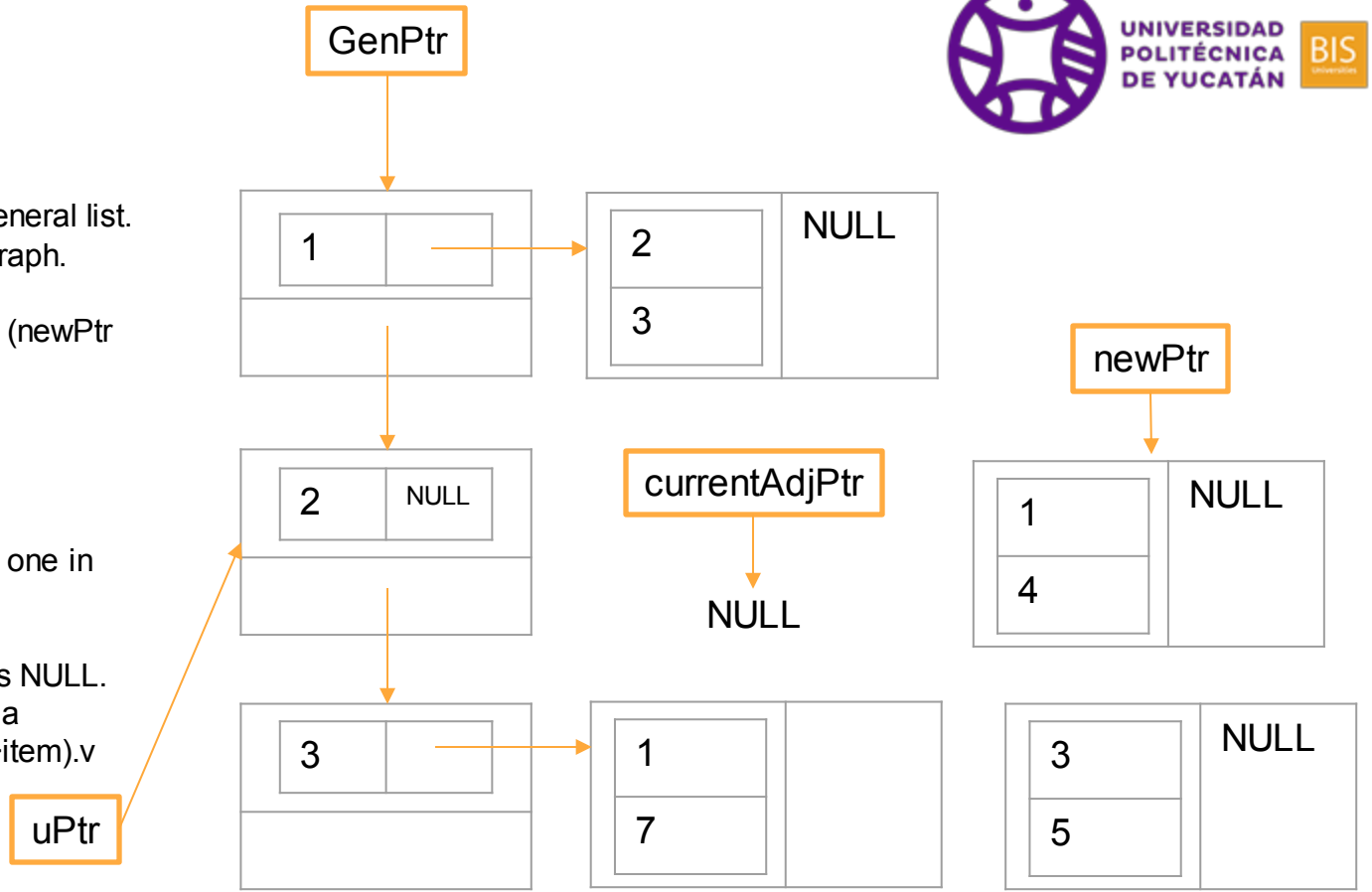


Add edge

addEdge(GenPtr, 2, 1, 4)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.

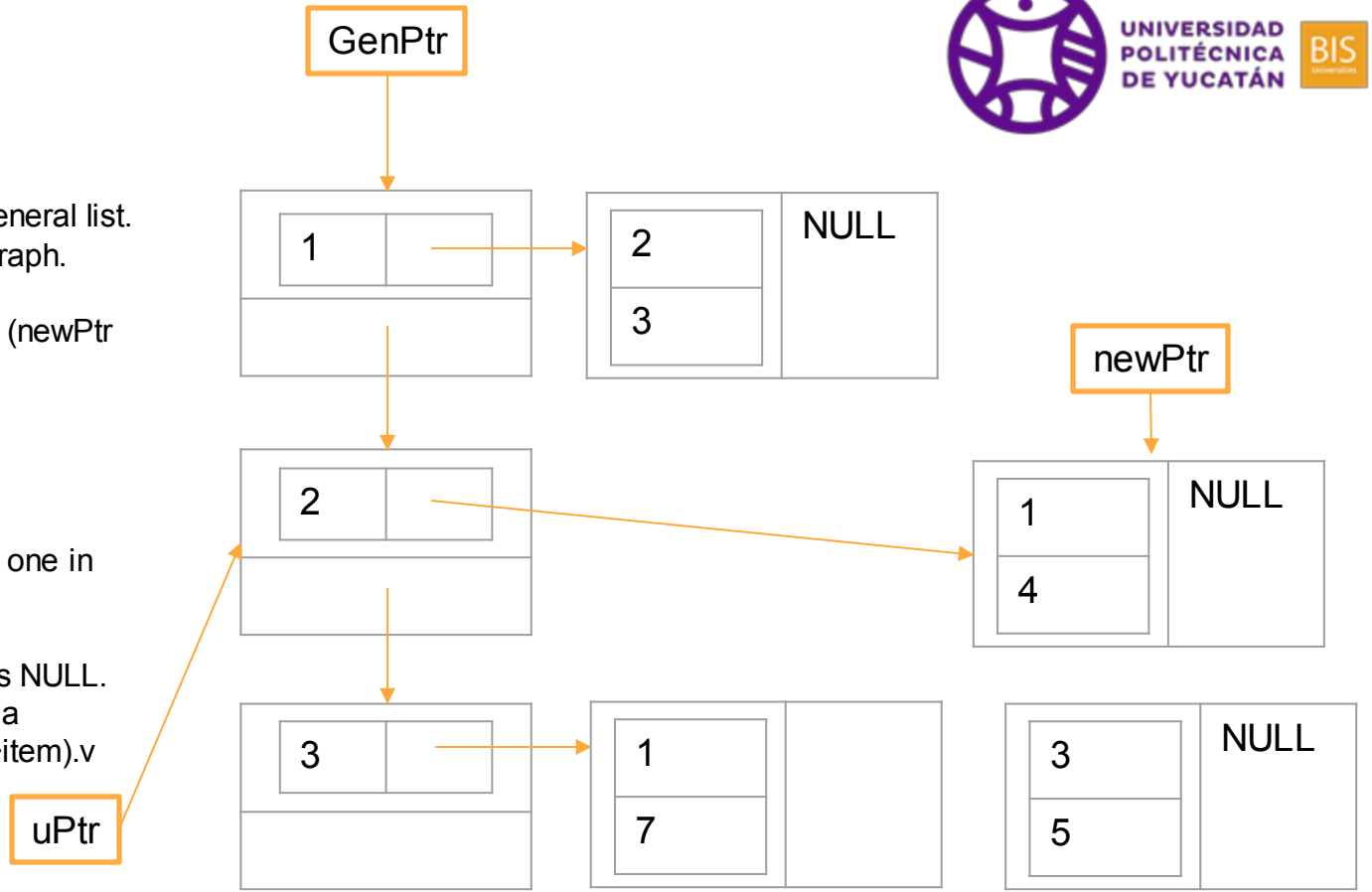


Add edge

addEdge(GenPtr, 2, 1, 4)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.

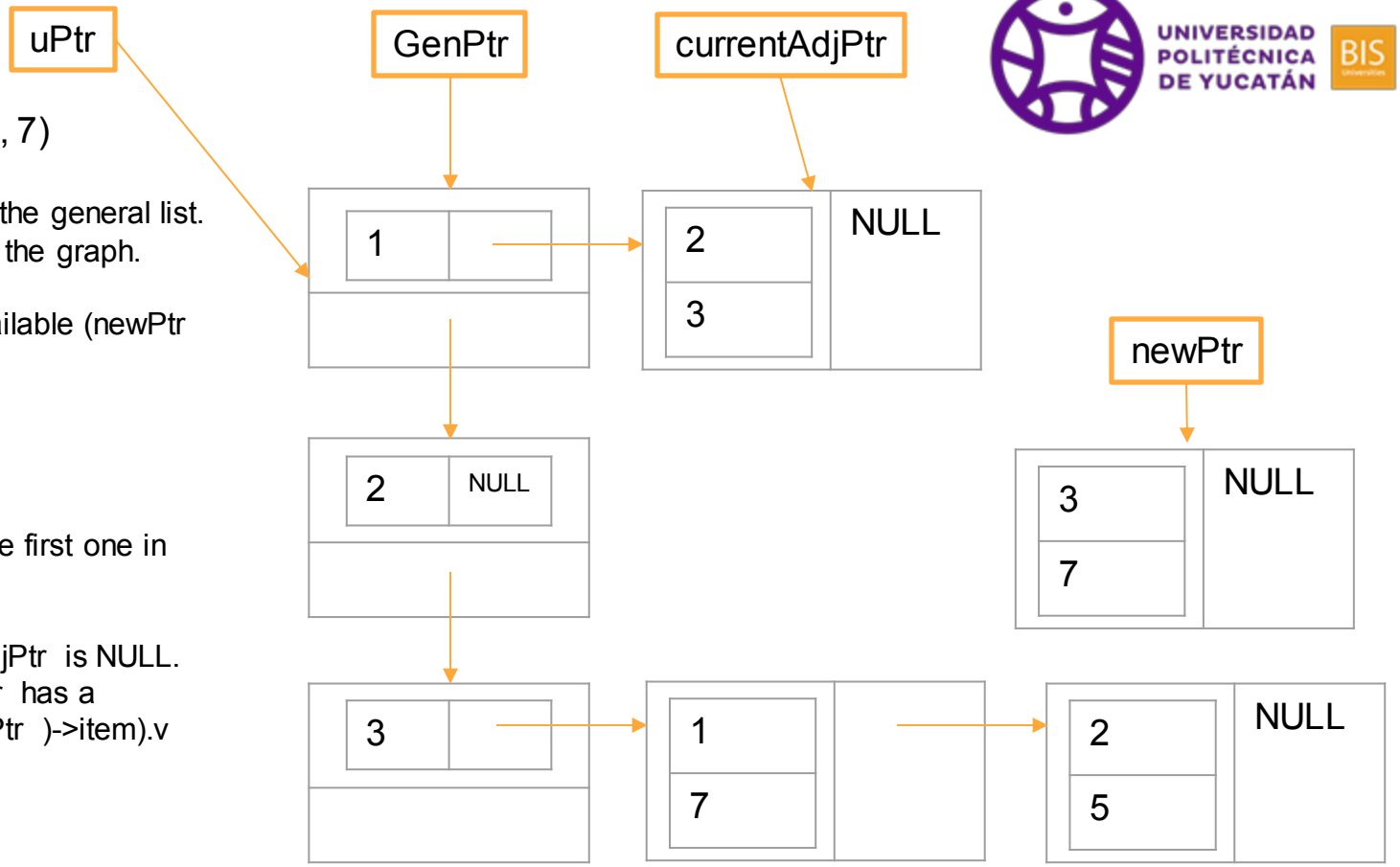


Add edge

addEdge(GenPtr, 1, 3, 7)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.

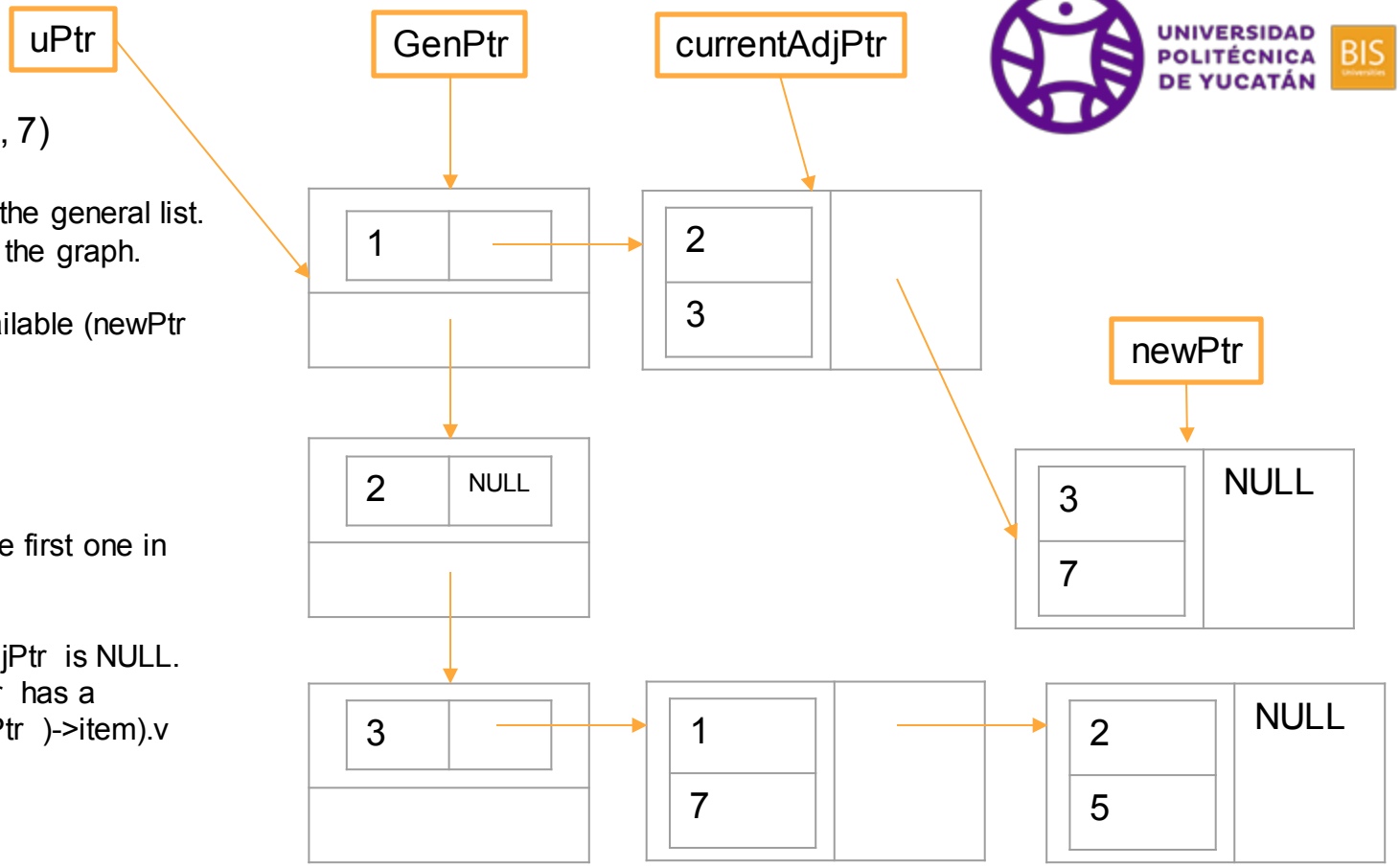


Add edge

addEdge(GenPtr, 1, 3, 7)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.

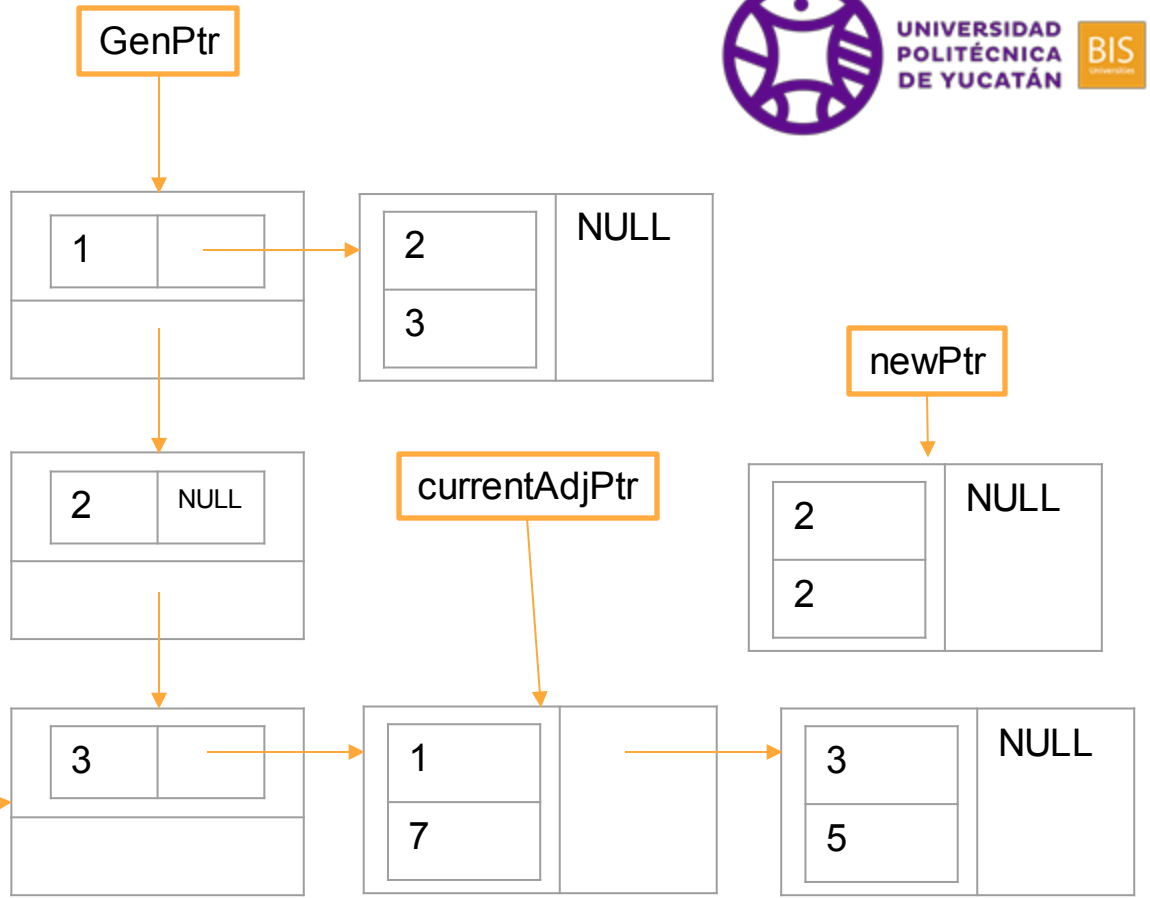


Add edge

addEdge(GenPtr, 3, 2, 2)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.

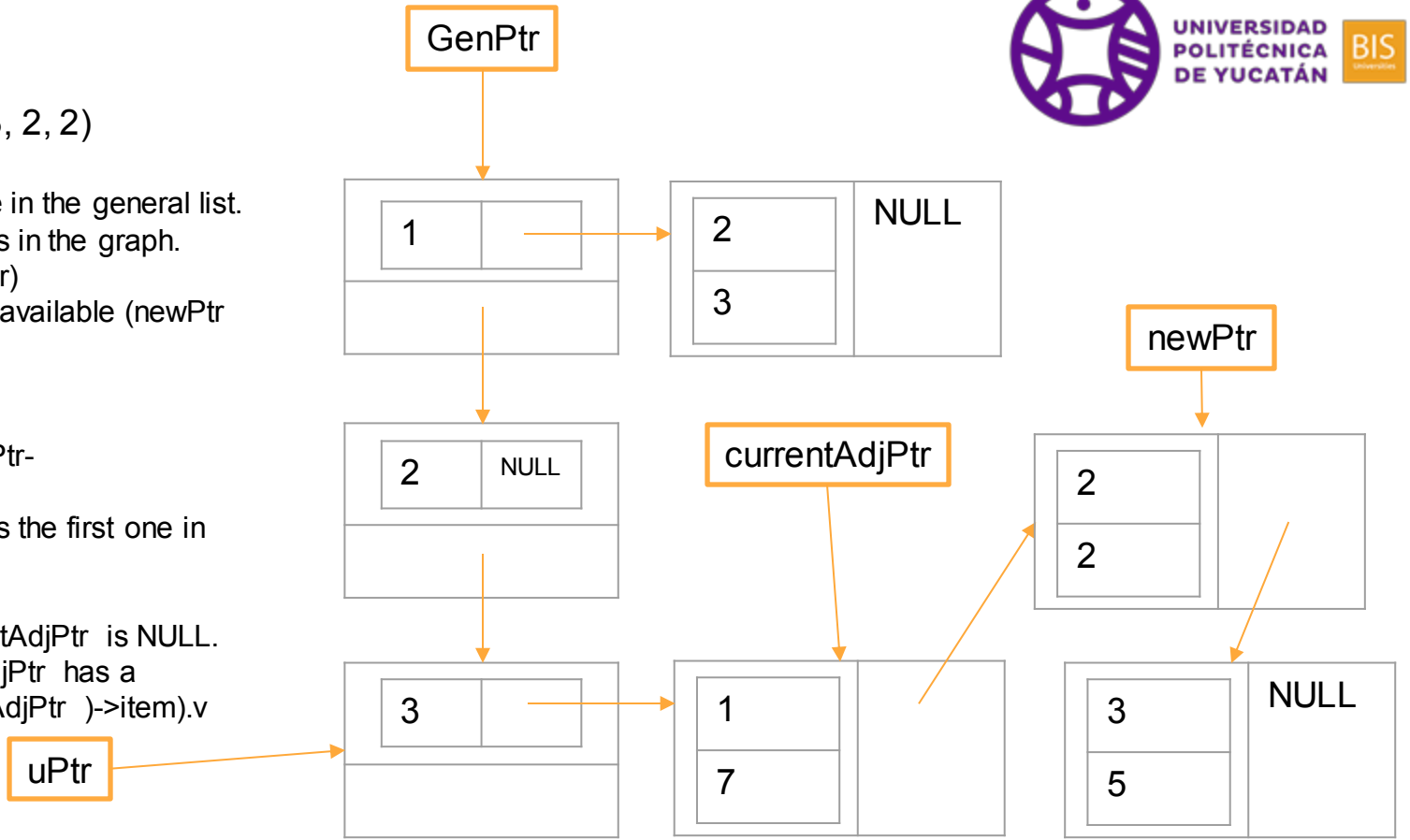


Add edge

addEdge(GenPtr, 3, 2, 2)

1) Check if u and v are in the general list.
2) Check if edge(u,v) is in the graph.
2) Create node (newPtr)
3) Check if memory is available (newPtr != NULL)
4) Initialize:
- newPtr
- currentAdjPtr == (uPtr->item).nodeAdjPtr

5) Check if edge(u,v) is the first one in the Adj List.
6) Iterate until:
- currentAdjPtr->nextAdjPtr is NULL.
- currentPtr->nextAdjPtr has a ((currentPtr->nextAdjPtr)->item).v greater than v.



Suggested References

- Barabási, Albert-László (2016) Network Science. USA.
- Menczer, Fortunato, Davis (2020) A First Course in Network Science.
- Mark Newman (2010). Networks: An introduction. UK.
- Reza Zafarani, Mohammad Ali Abbasi, Huan Liu (2014) Social Media Mining: An Introduction. UK.

- L. Joyanes-Aguilar et al. (2005) Estructuras de datos en C. Serie Shaum. McGraw-Hill.
- A. N. Kamthane (2012) Data Structures Using C. Pearson Education, Inc.
- L. Joyanes-Aguilar (2008) Fundamentos de Programación. Algorithmos, Estructura de Datos y Objetos.