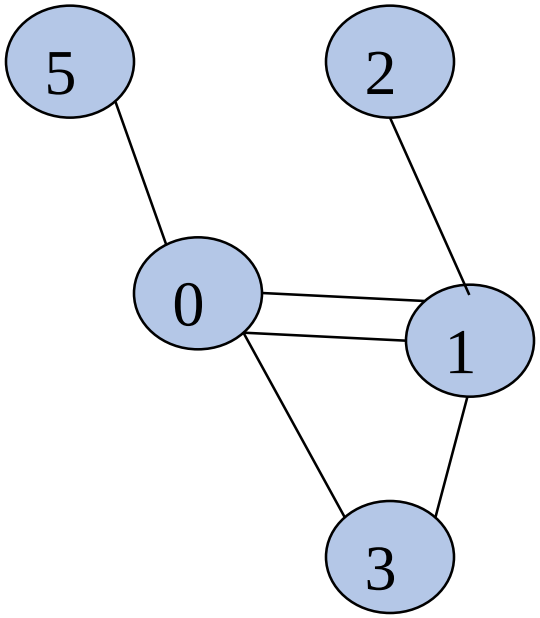
Graph Implementation

黃世強 (Sai-Keung Wong)

College of Computer Science
National Chiao Tung University, Taiwan

Using arrays to represent a graph

- We use integers for building up the relationships for nodes and edges.
- We use pointers to point to all the nodes and edges.
- All nodes have a unique ID.
- All nodes have a unique ID.
- We can obtain easily the adjacency information of nodes and edges.
- Good: deletion, addition



Classes

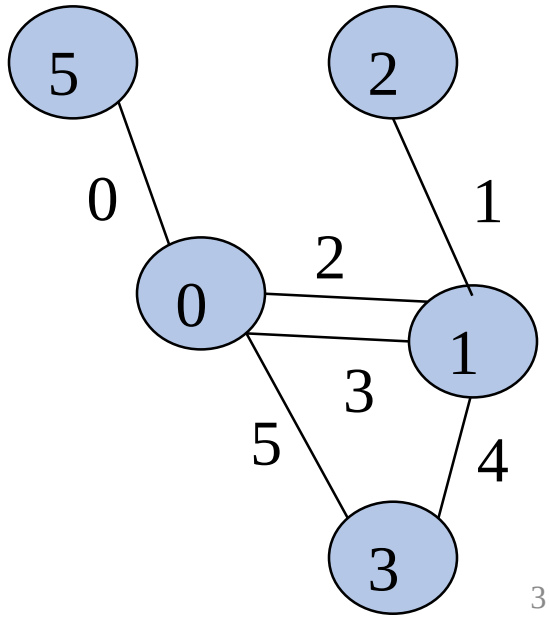
```
class GRAPH_NODE {
public:
    GRAPH_NODE( ) {
        r = 1.0;
        p = vector3(0.0, 0.0, 0.0);
    }
    vector3 p;  //position
    double r;   //radius
    int id;     //unique ID

    // in the active index array
    int dynamicID;

    vector<int> edgeID;

};
```

```
class GRAPH_EDGE {
public:
    int id; // unique
    int dynamicID;
    int nodeID[2];
};
```



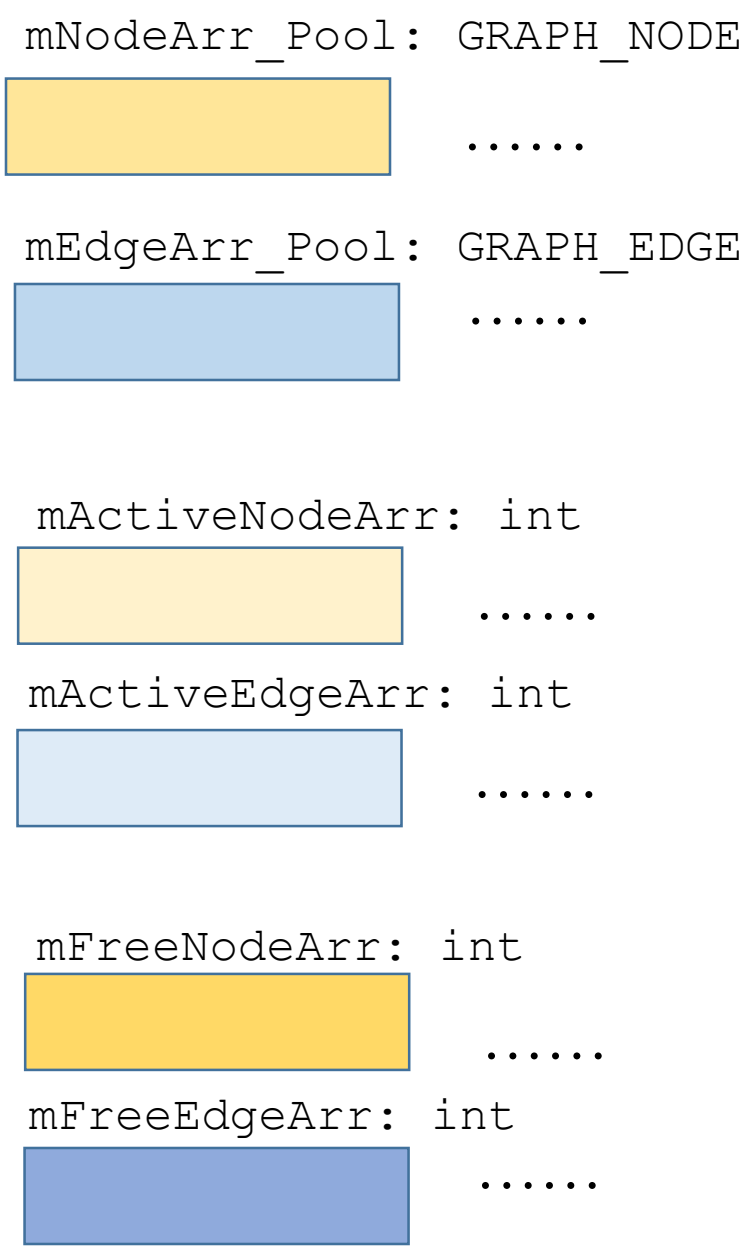
For node 0, edgeID[ ] {0, 2, 3, 5}

GRAPH class

```
class GRAPH_SYSTEM {
    ......
protected:
    GRAPH_NODE *mNodeArr_Pool;
    GRAPH_EDGE *mEdgeArr_Pool;
    //
    int *mActiveNodeArr;
    int mCurNumOfActiveNodes;
    int *mActiveEdgeArr;
    int mCurNumOfActiveEdges;
    //
    int *mFreeNodeArr;
    int *mFreeEdgeArr;
    int mCurNumOfFreeNodes;
    int mCurNumOfFreeEdges;
};
```

GRAPH class

```
class GRAPH_SYSTEM {
    ......
protected:
    GRAPH_NODE *mNodeArr_Pool;
    GRAPH_EDGE *mEdgeArr_Pool;
    //
    int *mActiveNodeArr;
    int mCurNumOfActiveNodes;
    int *mActiveEdgeArr;
    int mCurNumOfActiveEdges;
    //
    int *mFreeNodeArr;
    int *mFreeEdgeArr;
    int mCurNumOfFreeNodes;
    int mCurNumOfFreeEdges;
};
```



GRAPH class

The node pool stores all the nodes, including active and inactive ones. Similarly for the edge pool…

`mNodeArr_Pool: GRAPH_NODE`

……

`mEdgeArr_Pool: GRAPH_EDGE`

……

The active node array maintains the indices of all the active nodes.
These active nodes are used in a graph. Similarly for the active edge array…

`mActiveNodeArr: int`

……

`mActiveEdgeArr: int`

……

The free node array maintains the indices of all the inactive nodes.
These free nodes are not used in a graph. Similarly for the free edge array…

`mFreeNodeArr: int`

……

`mFreeEdgeArr: int`

……

```
void GRAPH_SYSTEM::initMemoryPool( )
{
    mNodeArr_Pool = new GRAPH_NODE[GRAPH_MAX_NUM_NODES];
    mEdgeArr_Pool = new GRAPH_EDGE[GRAPH_MAX_NUM_EDGES];

    mCurNumOfActiveNodes = 0;
    mCurNumOfActiveEdges = 0;
    mActiveNodeArr = new int[GRAPH_MAX_NUM_NODES];
    mActiveEdgeArr = new int[GRAPH_MAX_NUM_EDGES];

    mFreeNodeArr = new int[GRAPH_MAX_NUM_NODES];
    mFreeEdgeArr = new int[GRAPH_MAX_NUM_EDGES];
    //
    for ( int i = 0; i < GRAPH_MAX_NUM_NODES; ++i ) {
        mNodeArr_Pool[ i ].id = i; // assign a unique id
    }
    for ( int i = 0; i < GRAPH_MAX_NUM_EDGES; ++i ) {
        mEdgeArr_Pool[ i ].id = i; // assign a unique id
    }
    reset( );
}
```

```
void GRAPH_SYSTEM::reset( ) {

    mCurNumOfActiveNodes = 0;
    mCurNumOfActiveEdges = 0;

    mCurNumOfFreeNodes = GRAPH_MAX_NUM_NODES;
    mCurNumOfFreeEdges = GRAPH_MAX_NUM_EDGES;

    for ( int i = 0; i < mCurNumOfFreeNodes; ++i ) {
        mFreeNodeArr[ i ] = i; // index is not used
    }
    for ( int i = 0; i < mCurNumOfFreeEdges; ++i ) {
        mFreeEdgeArr[ i ] = i; // index is not used
    }
.......
}
```

Creation of a graph

- First, create nodes
- Second, create edges

- Before an edge is created, the nodes connected by the edge must be created first.

- After a node is created, its unique id is returned. We can use its id to create an edge.
- After an edge is created, its unique id is returned. We can use its id to access its adjacent nodes.

First, create nodes

```
GRAPH_NODE *GRAPH_SYSTEM::getFreeNode( ) {
    if ( mCurNumOfFreeNodes == 0 ) return 0;
    --mCurNumOfFreeNodes;
    int id = mFreeNodeArr[ mCurNumOfFreeNodes ];
    GRAPH_NODE *n = &mNodeArr_Pool[ id ];
    mActiveNodeArr[ mCurNumOfActiveNodes ] = id;
    n->dynamicID = mCurNumOfActiveNodes;
    ++mCurNumOfActiveNodes;
    return n;
}
```

mNodeArr_Pool

mFreeNodeArr

```
int GRAPH_SYSTEM::addNode(
   float x, float y, float z, float r ) {
    GRAPH_NODE *g;
    g = getFreeNode( );
    if ( g == 0 ) return -1; // invalid id.
    g->p = vector3( x, y, z );
    g->r = r;
    g->edgeID.clear( );
    return g->id;
}
```

Second, create edges

```
GRAPH_EDGE *GRAPH_SYSTEM::getFreeEdge( ) {
    if ( mCurNumOfFreeEdges == 0 ) return 0;
    --mCurNumOfFreeEdges;
    int id = mFreeEdgeArr[ mCurNumOfFreeEdges ];
    GRAPH_EDGE *e = &mEdgeArr_Pool[ id ];
    mActiveEdgeArr[ mCurNumOfActiveEdges ] = id;
    e->dynamicID = mCurNumOfActiveEdges;
    ++mCurNumOfActiveEdges;
    return e;
}
```

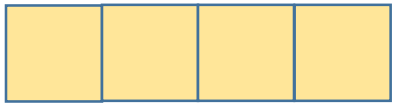
mEdgeArr_Pool

mFreeEdgeArr

```
int GRAPH_SYSTEM::addEdge( int nodeID_0, int nodeID_1 ) {
    GRAPH_EDGE *e;
    e = getFreeEdge( );
    if ( e == 0 ) return -1;
    e->nodeID[ 0 ] = nodeID_0;
    e->nodeID[ 1 ] = nodeID_1;
    mNodeArr_Pool[ nodeID_0 ].edgeID.push_back( e->id );
    mNodeArr_Pool[ nodeID_1 ].edgeID.push_back( e->id );
    return e->id;
}
```

An example

mNodeArr_Pool: GRAPH_NODE



mEdgeArr_Pool: GRAPH_EDGE

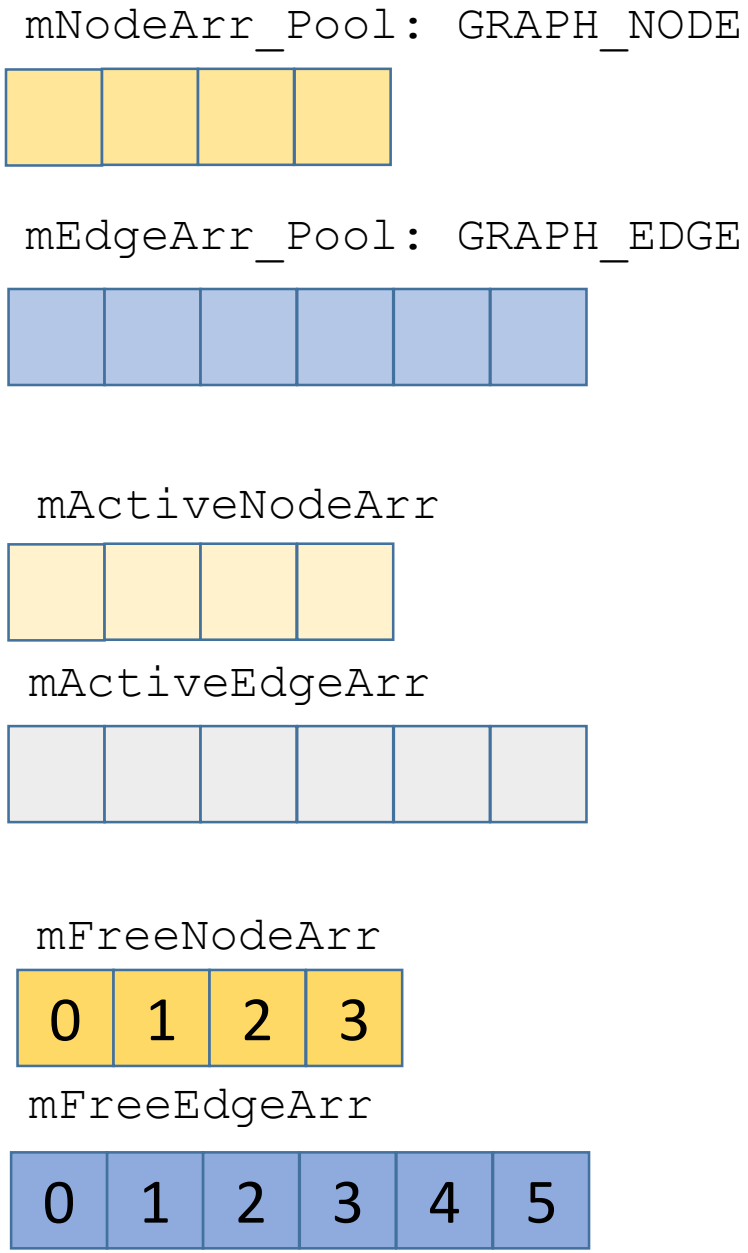
mActiveNodeArr: int

mActiveEdgeArr: int

mFreeNodeArr: int

mFreeEdgeArr: int

An example


mNodeArr_Pool: GRAPH_NODE
mEdgeArr_Pool: GRAPH_EDGE
mActiveNodeArr
mActiveEdgeArr
mFreeNodeArr
0
1
2
3
mFreeEdgeArr
0
1
2
3
4
5

An example

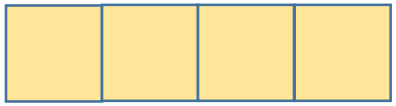
```
class GRAPH_NODE {
public:
    GRAPH_NODE( ) {
        r = 1.0;
        p = vector3(0.0, 0.0, 0.0);
    }
    vector3 p;   //position
    double r;    //radius
    int id;      //unique ID

    // in the active index array
    int dynamicID;

    vector<int> edgeID;

};
```

mNodeArr_Pool: GRAPH_NODE



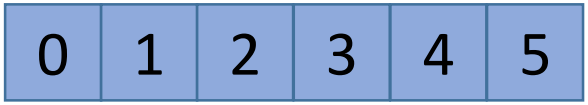
mEdgeArr_Pool: GRAPH_EDGE

mActiveNodeArr

mActiveEdgeArr

mFreeNodeArr

mFreeEdgeArr



An example

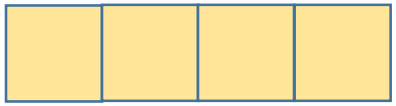
index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = ?;
 edgeID = { }
};
```

3

mNodeArr_Pool: GRAPH_NODE



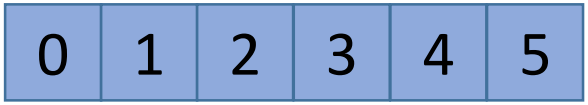
mEdgeArr_Pool: GRAPH_EDGE

mActiveNodeArr

mActiveEdgeArr

mFreeNodeArr

mFreeEdgeArr



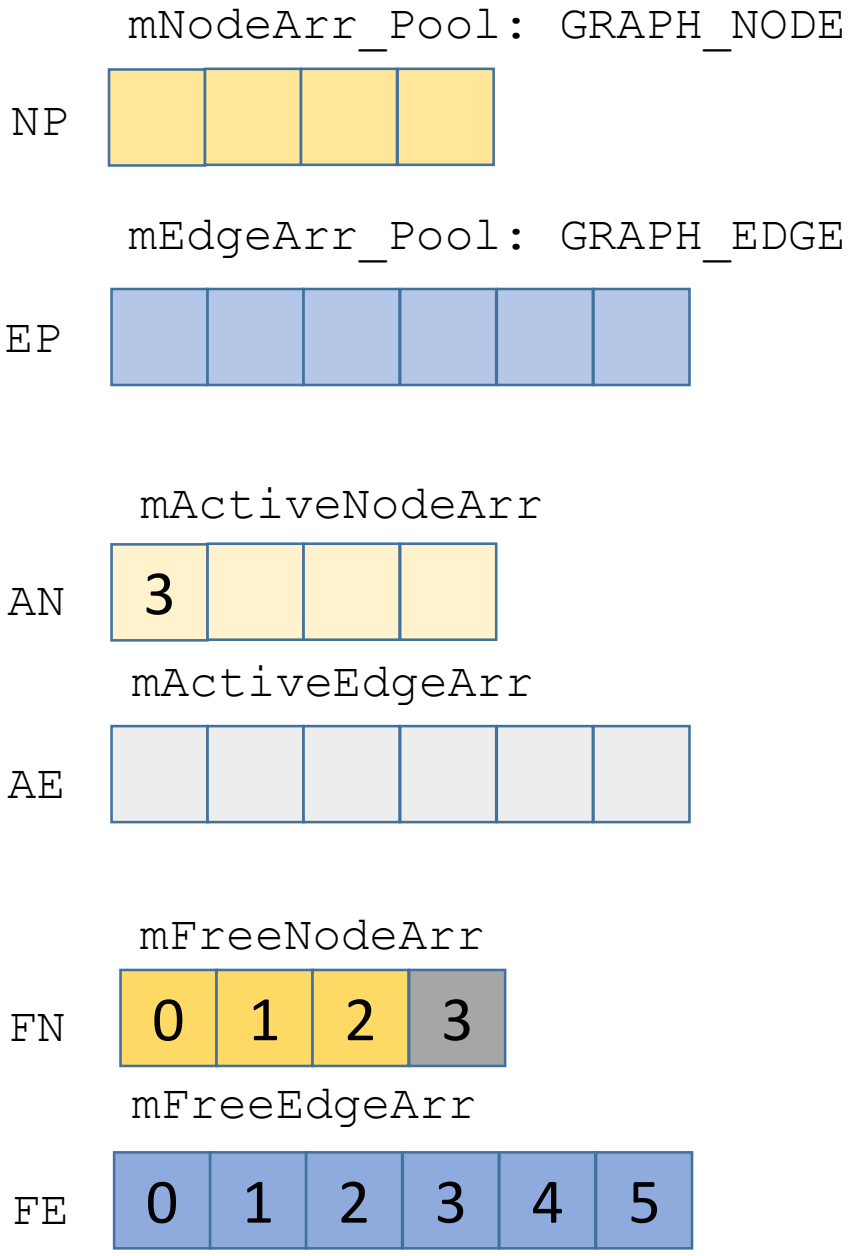
An example

index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = 7;
 edgeID = { }
};
```

3

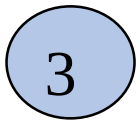


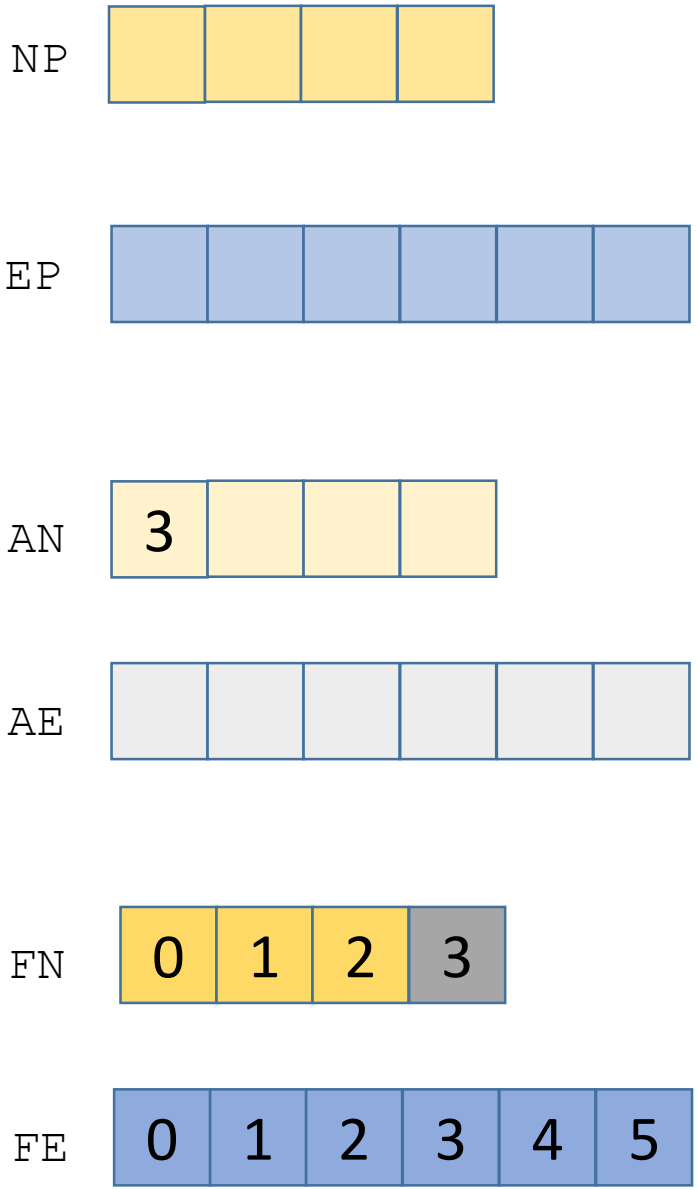
An example

index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = 7;
 edgeID = { }
};
```





An example
Try again

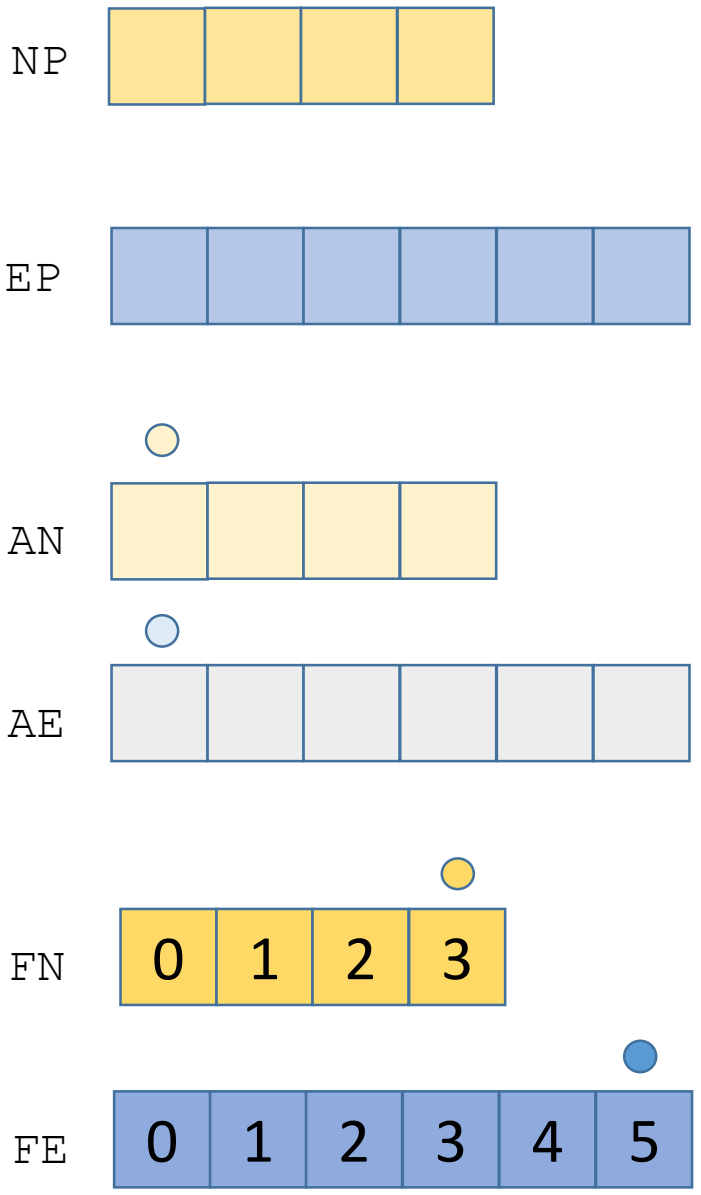
index = addNode(…)

```
freeNodeIndex = ?

NODE {
 id = ?
 dynamicID = ?;
 edgeID = ?
};
```

Need counters

3



Add Node Step 1:0

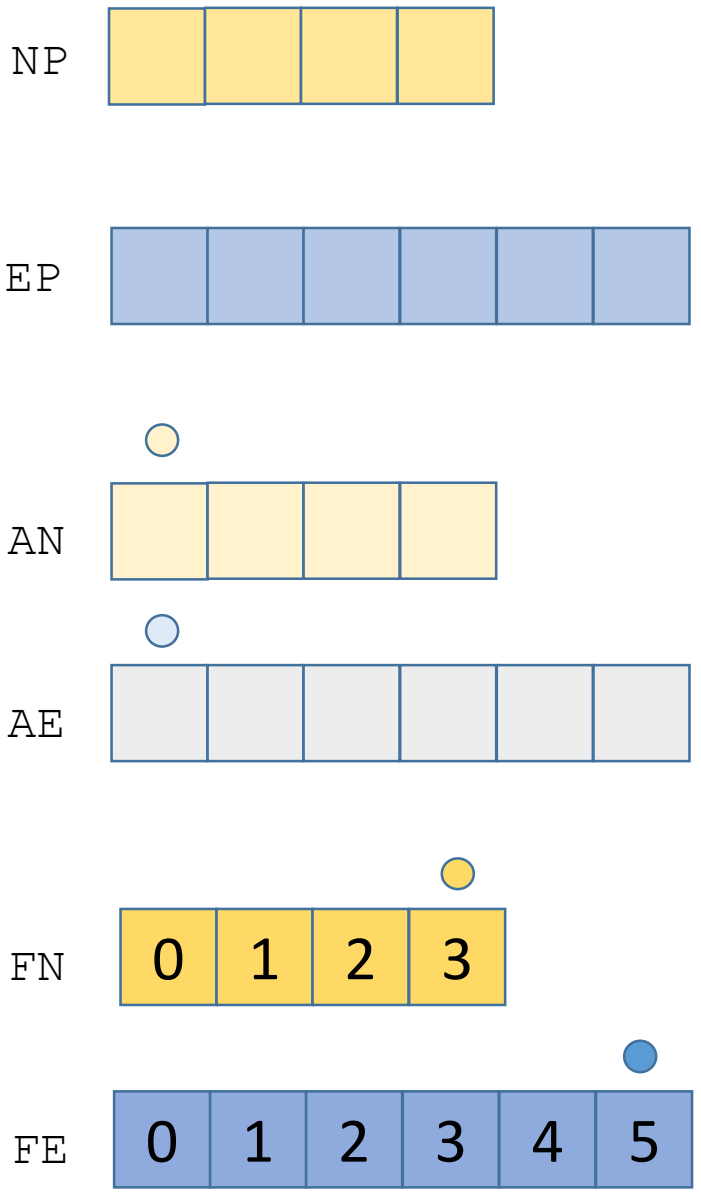
index = addNode(…)

```
freeNodeIndex = ?

NODE {
 id = ?
 dynamicID = ?;
 edgeID = ?
};
```

Need counters

3



Add Node Step 1:1

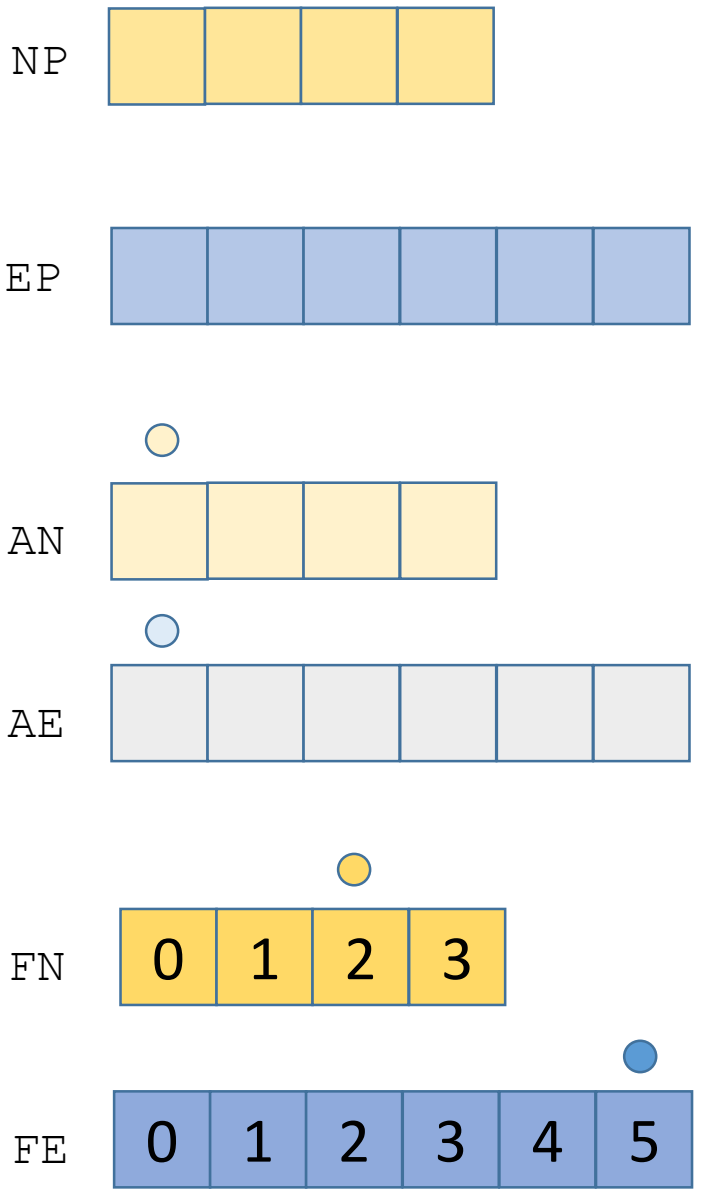
index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = ?
 dynamicID = ?;
 edgeID = ?
};
```

Need counters

3



Add Node Step 1:2

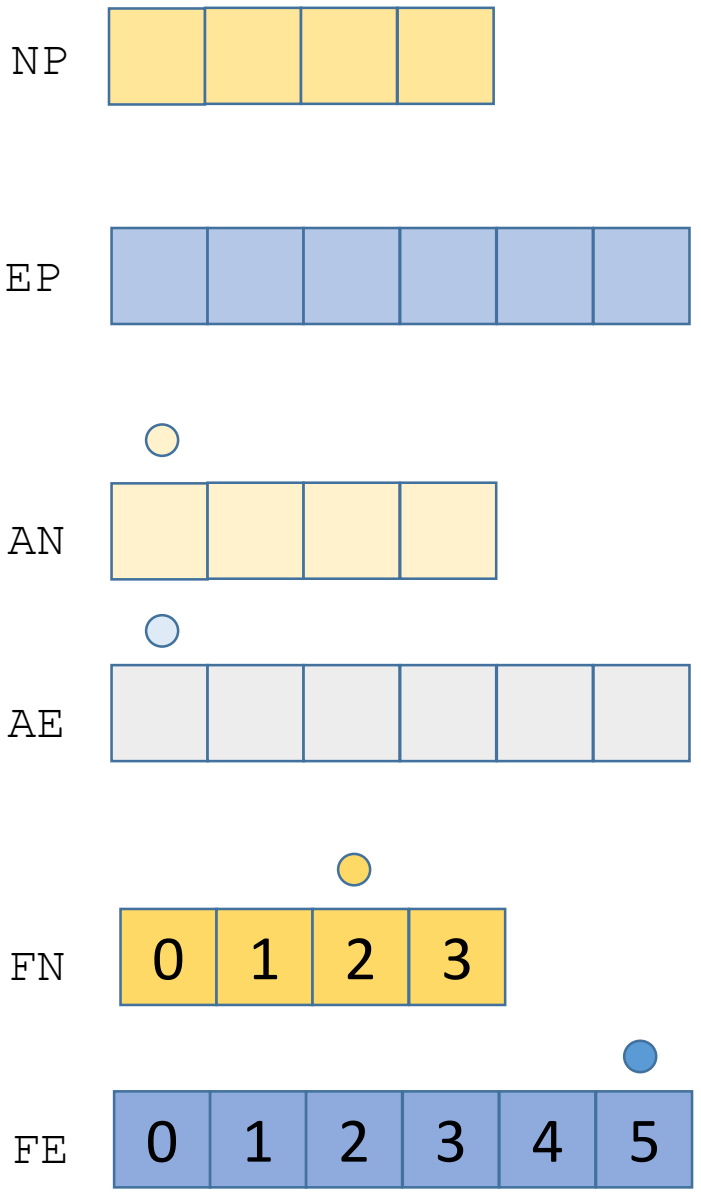
index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = ?;
 edgeID = ?
};
```

Need counters

3

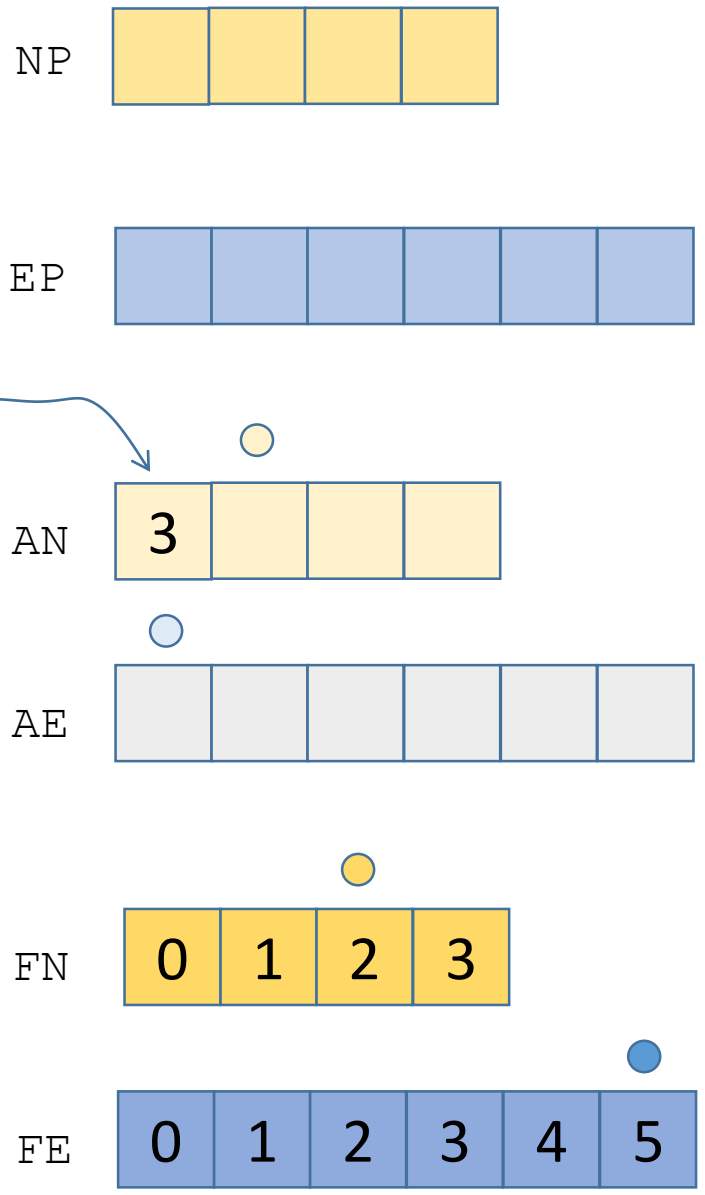


Add Node Step 1:3

index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = 0;
 edgeID = ?
};
```



Need counters

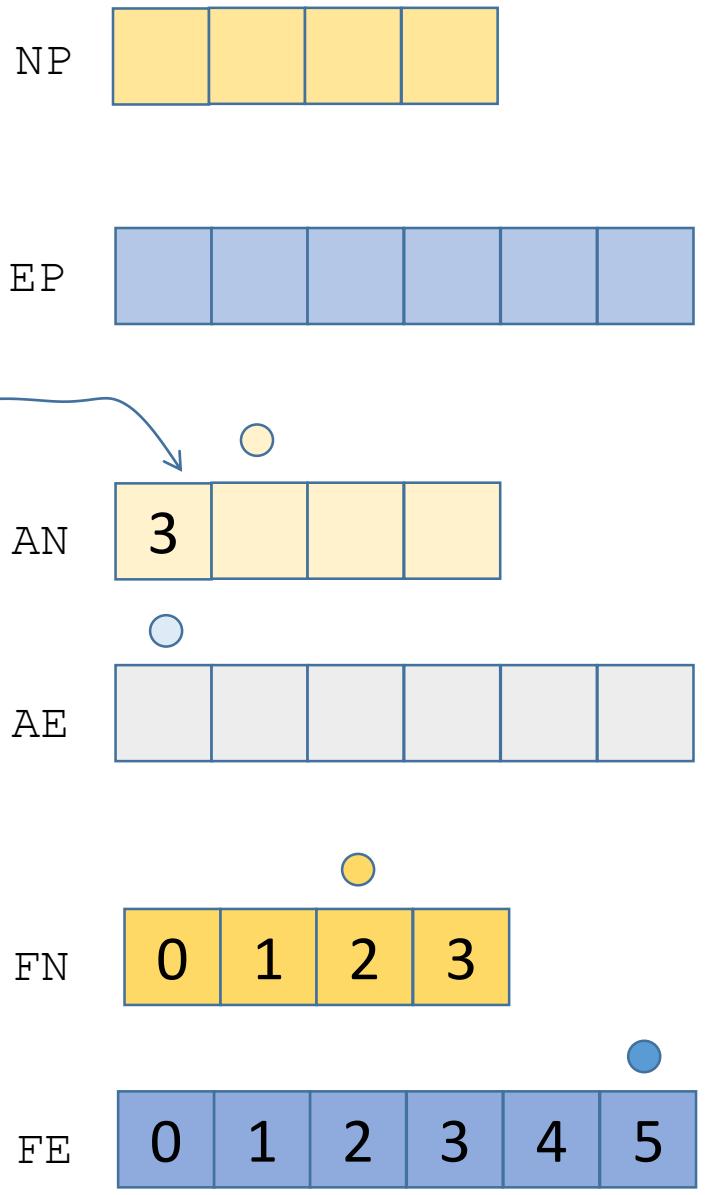
3

Add Node Step 1:4

index = addNode(…)

```
freeNodeIndex = 3

NODE {
 id = 3
 dynamicID = 0;
 edgeID = { }
};
```



Need counters

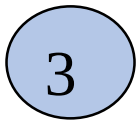
3

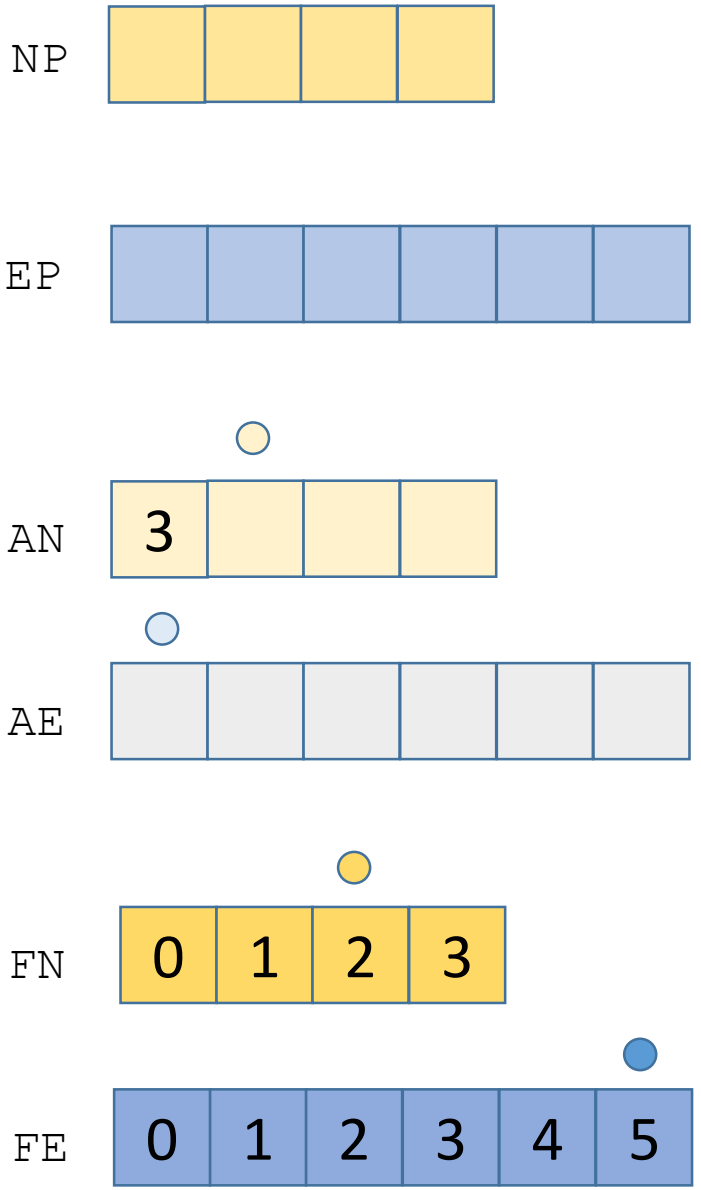
Add 2nd Node Step 2:0

index = addNode(…)

```
freeNodeIndex = ?

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



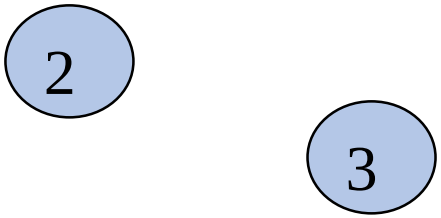


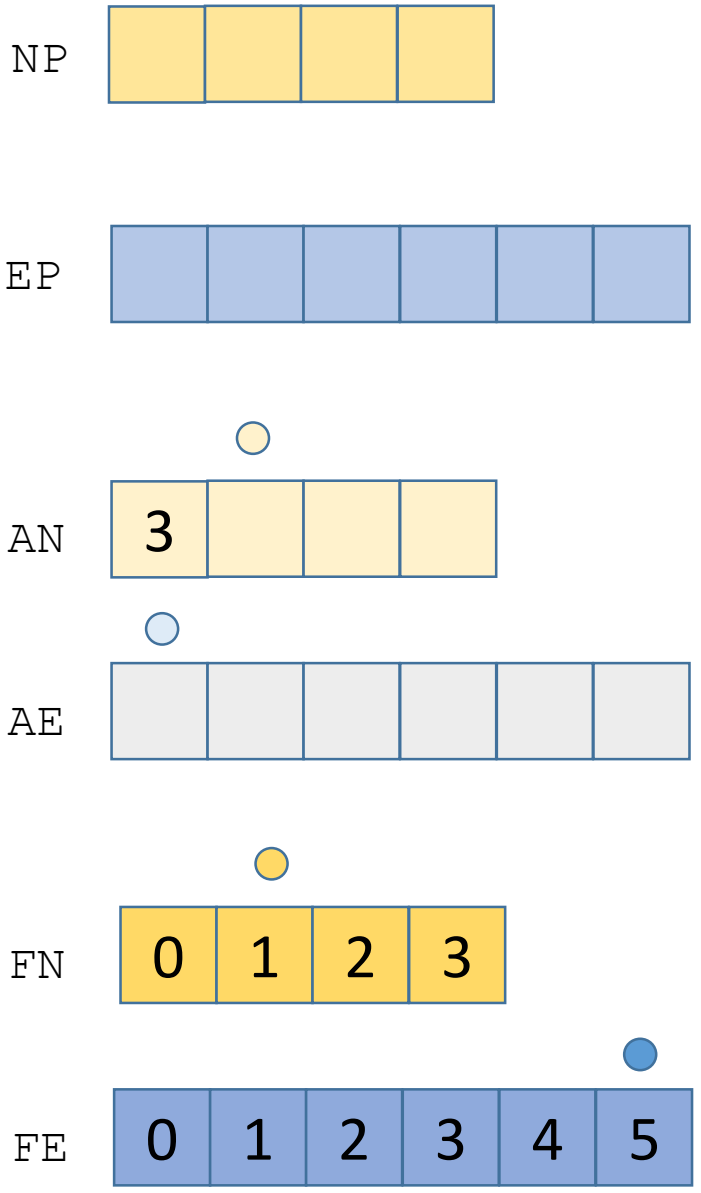
Add 2nd Node Step 2:1

index = addNode(…)

```
freeNodeIndex = 2

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



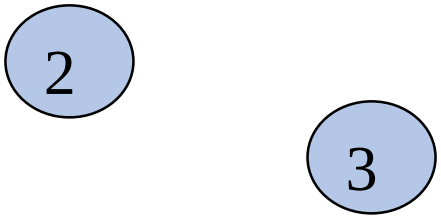


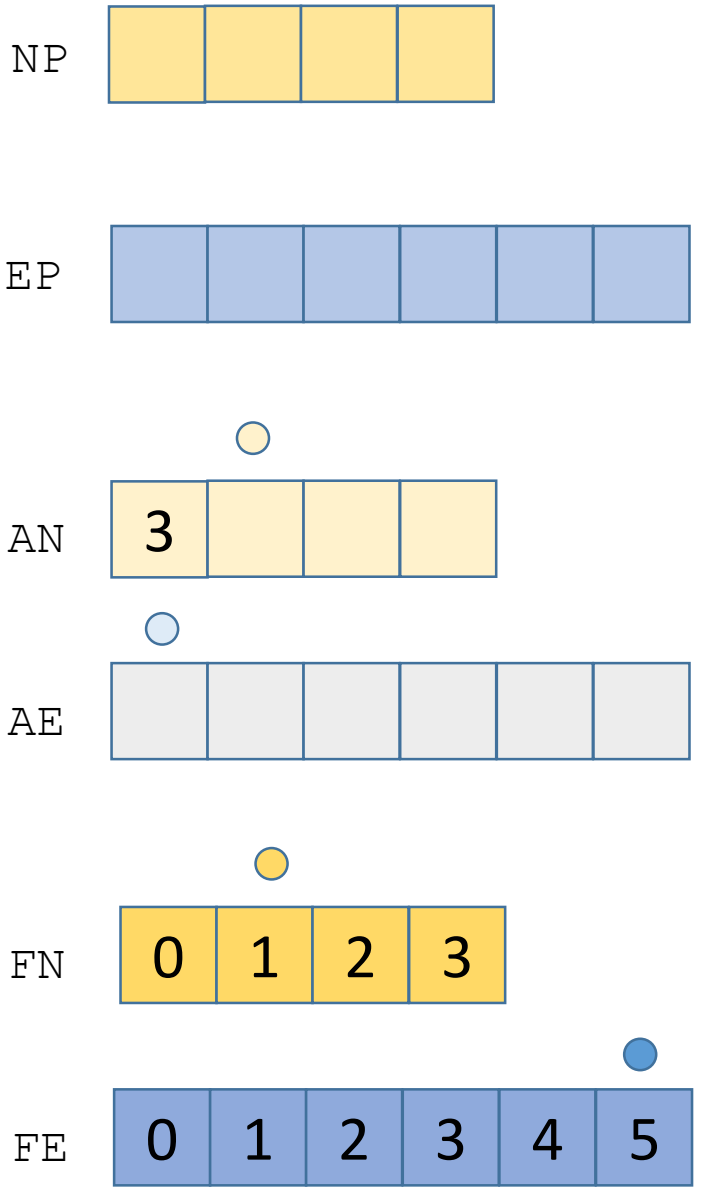
Add 2nd Node Step 2:2

index = addNode(…)

```
freeNodeIndex = 2

NODE {
 id = 2
 dynamicID = ?
 edgeID = ?
};
```



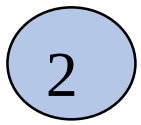


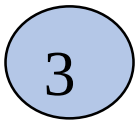
Add 2nd Node Step 2:3

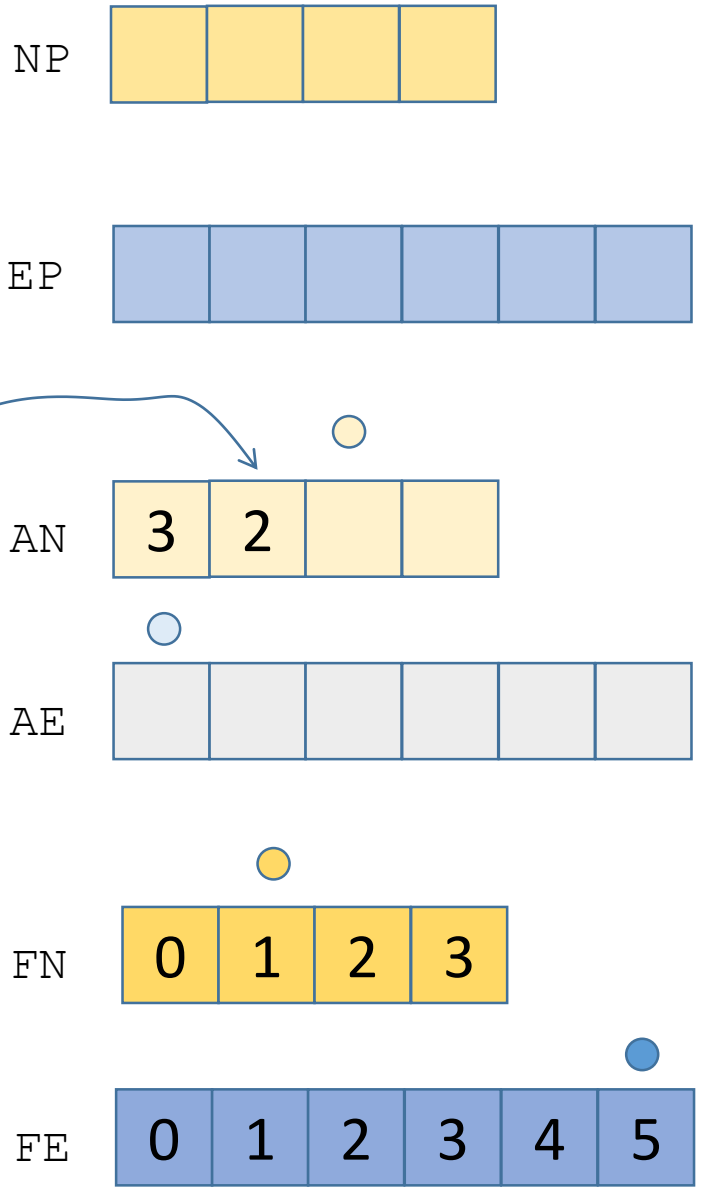
index = addNode(…)

```
freeNodeIndex = 2

NODE {
 id = 2
 dynamicID = 1
 edgeID = ?
};
```





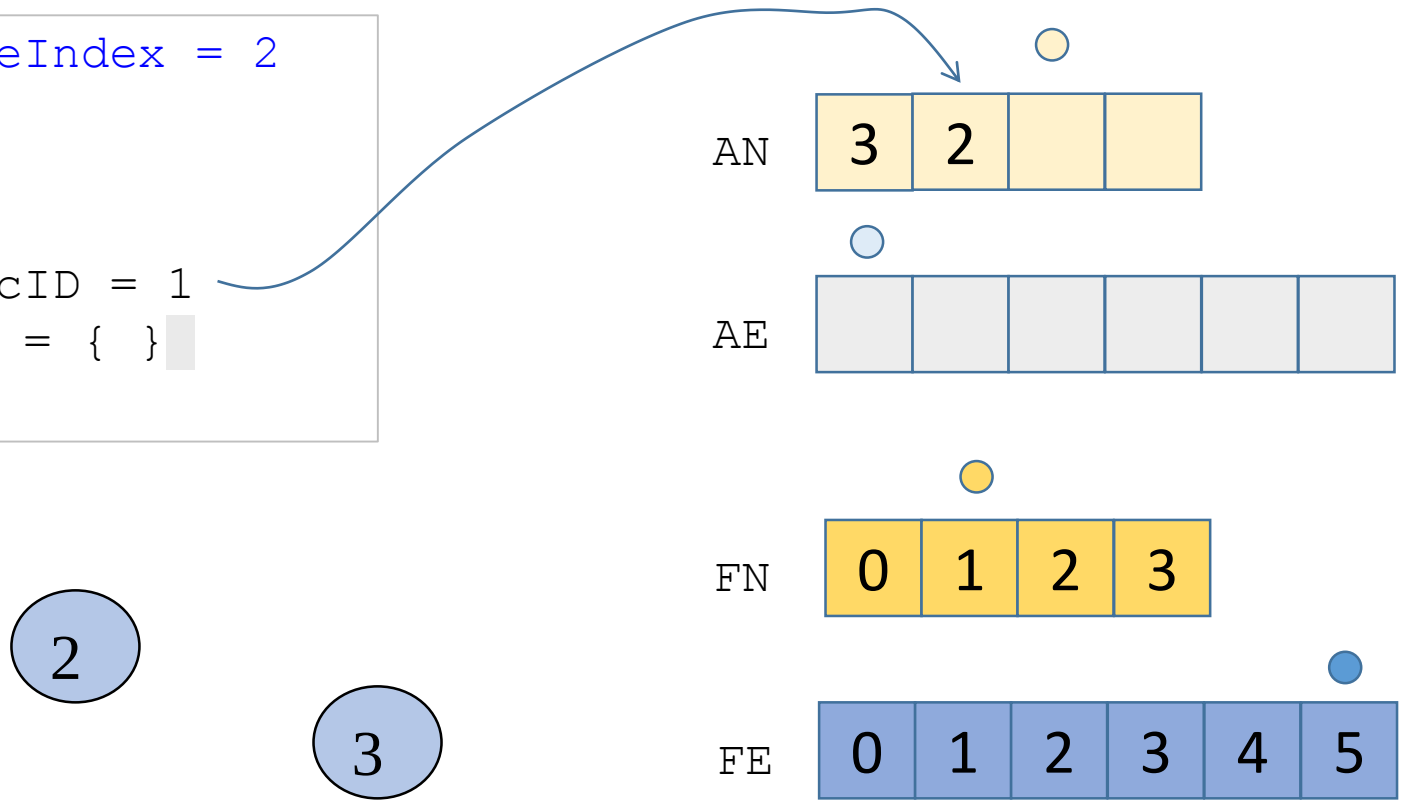


Add 2nd Node Step 2:4

index = addNode(…)

```
freeNodeIndex = 2

NODE {
 id = 2
 dynamicID = 1
 edgeID = { }
};
```

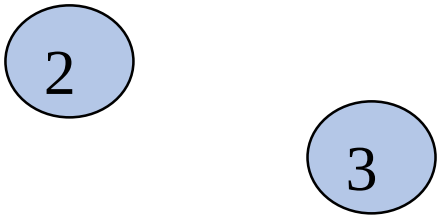


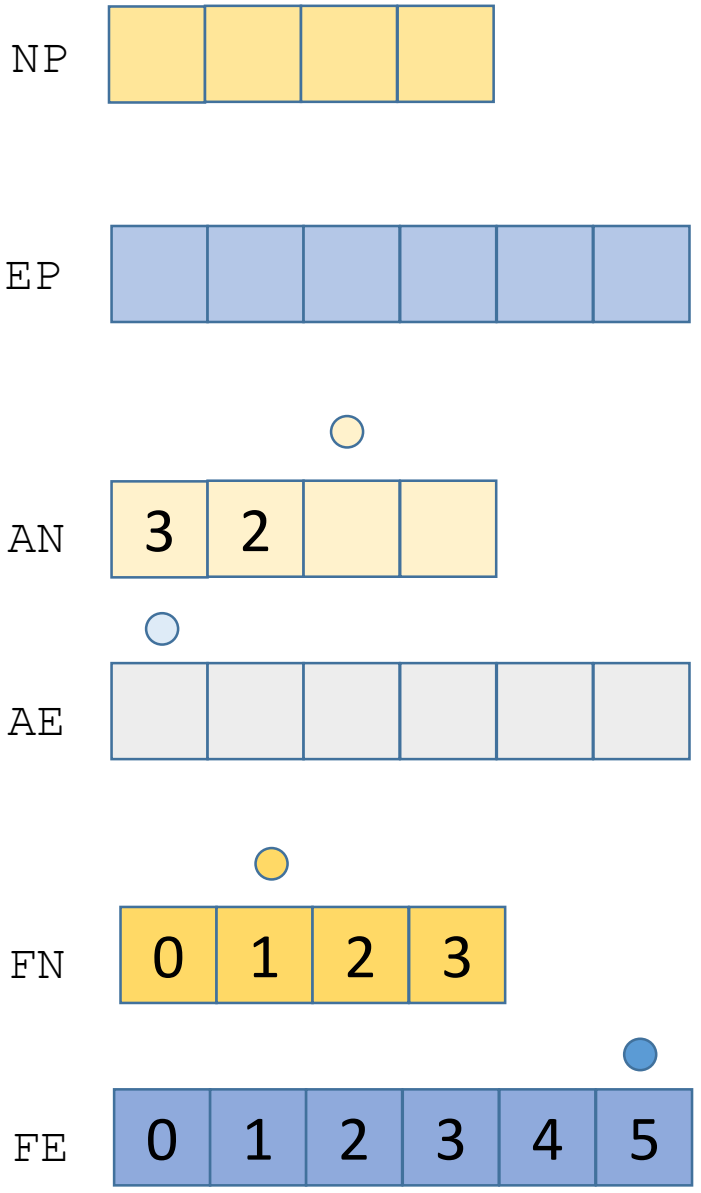
Add 3rd Node Step 3:0

index = addNode(…)

```
freeNodeIndex = ?

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



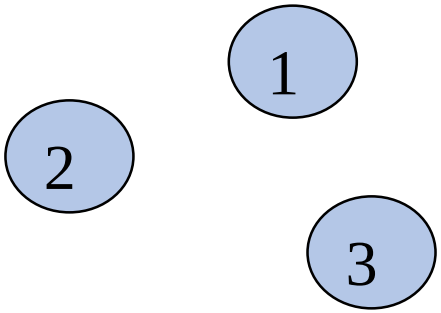


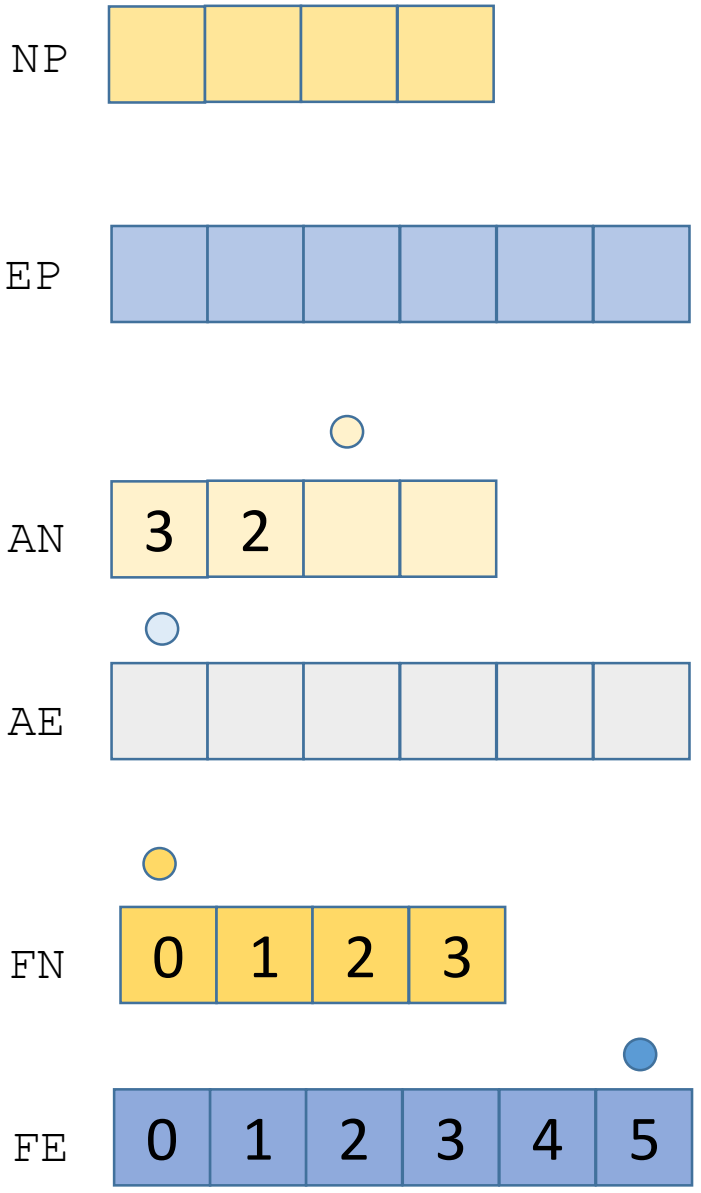
Add 3rd Node Step 3:1

index = addNode(…)

```
freeNodeIndex = 1

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



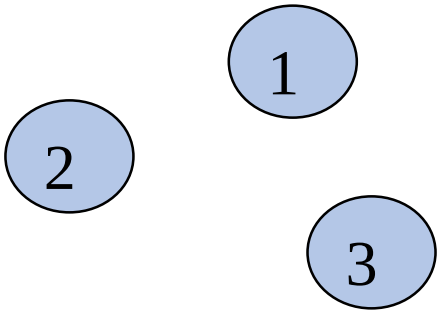


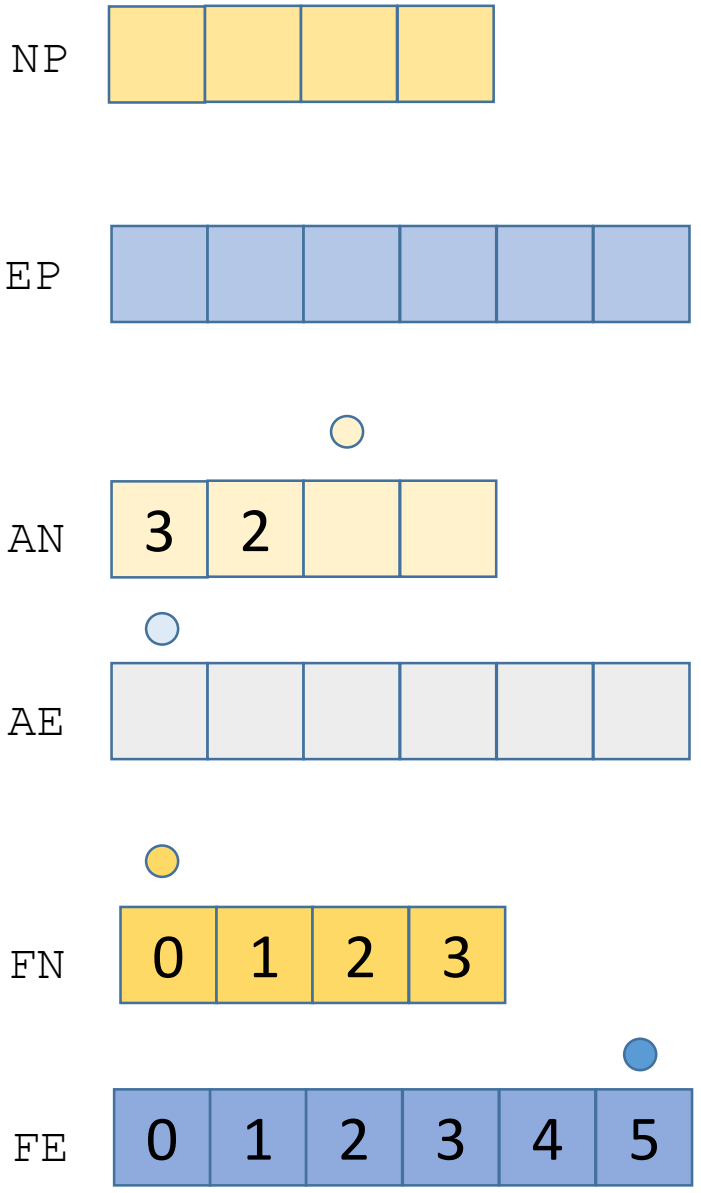
Add 3rd Node Step 3:2

index = addNode(…)

```
freeNodeIndex = 1

NODE {
 id = 1
 dynamicID = ?
 edgeID = ?
};
```



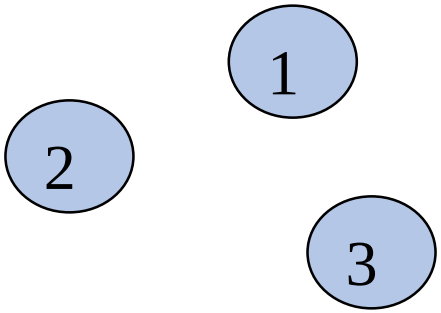


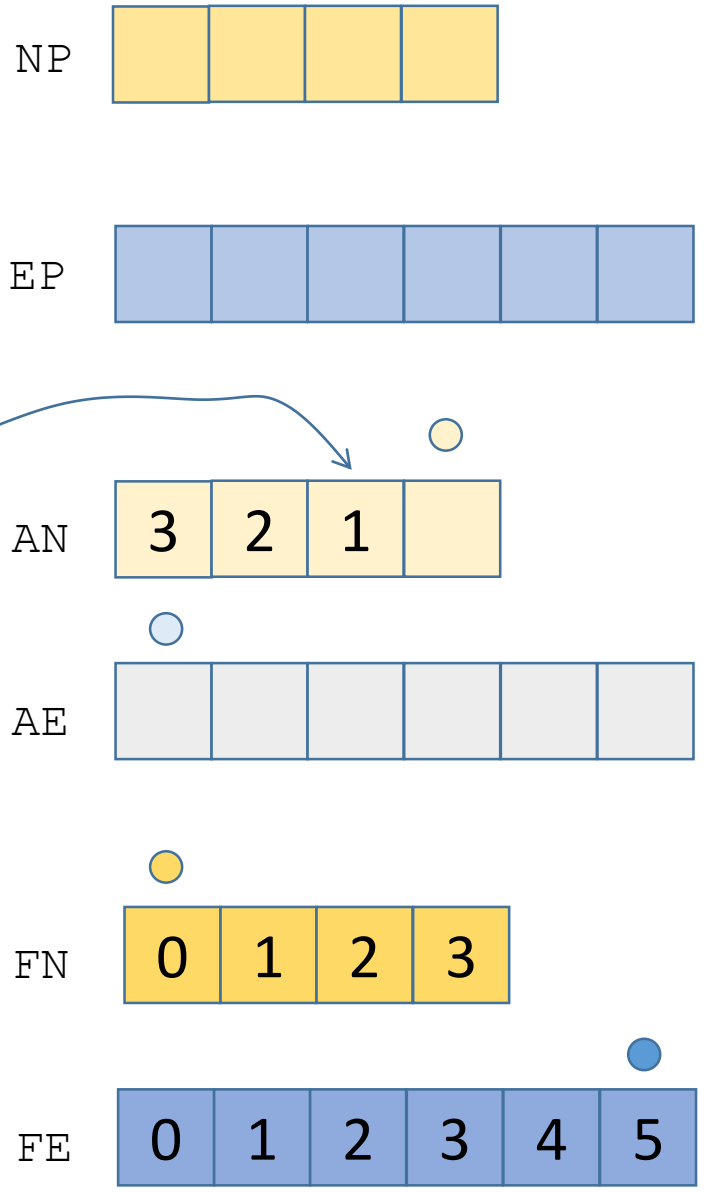
Add 3rd Node Step 3:3

index = addNode(…)

```
freeNodeIndex = 1

NODE {
 id = 1
 dynamicID = 2
 edgeID = ?
};
```



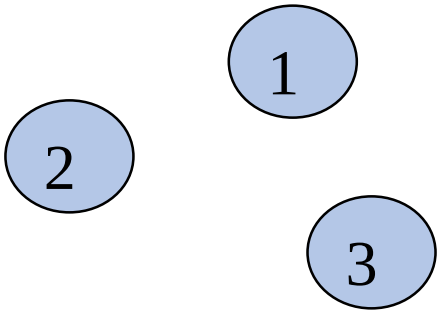


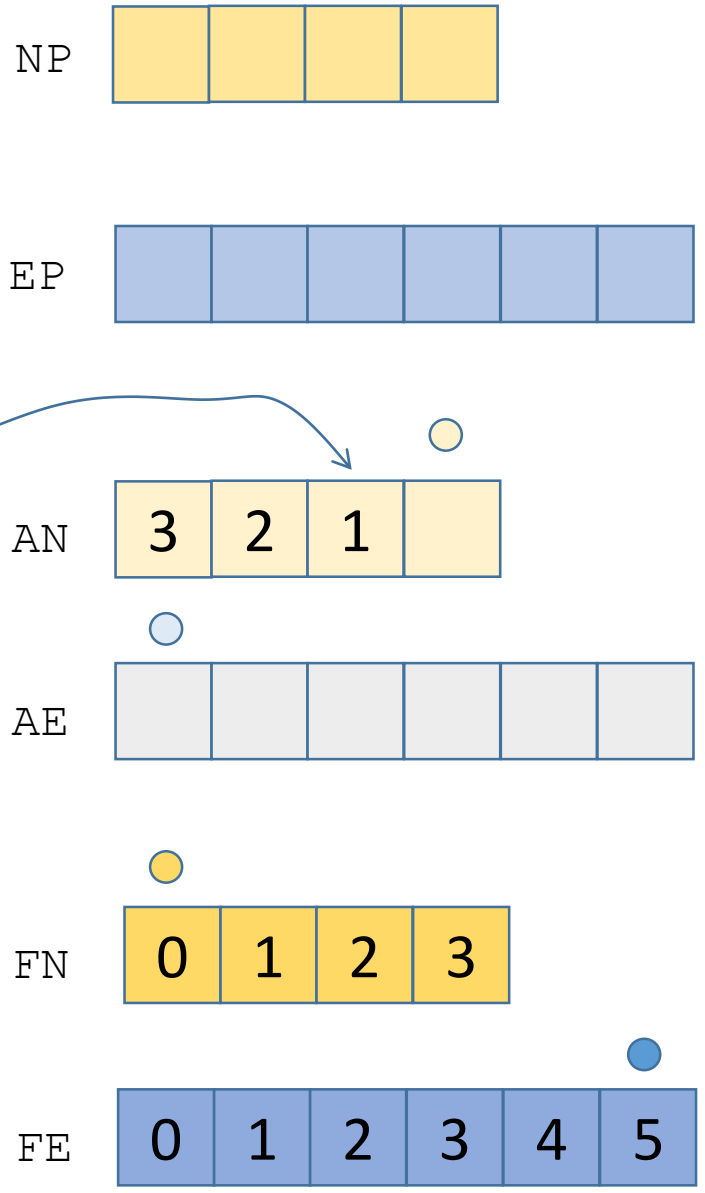
Add 3rd Node Step 3:4

index = addNode(…)

```
freeNodeIndex = 1

NODE {
 id = 1
 dynamicID = 2
 edgeID = { }
};
```



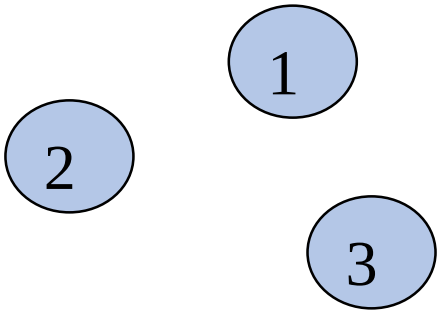


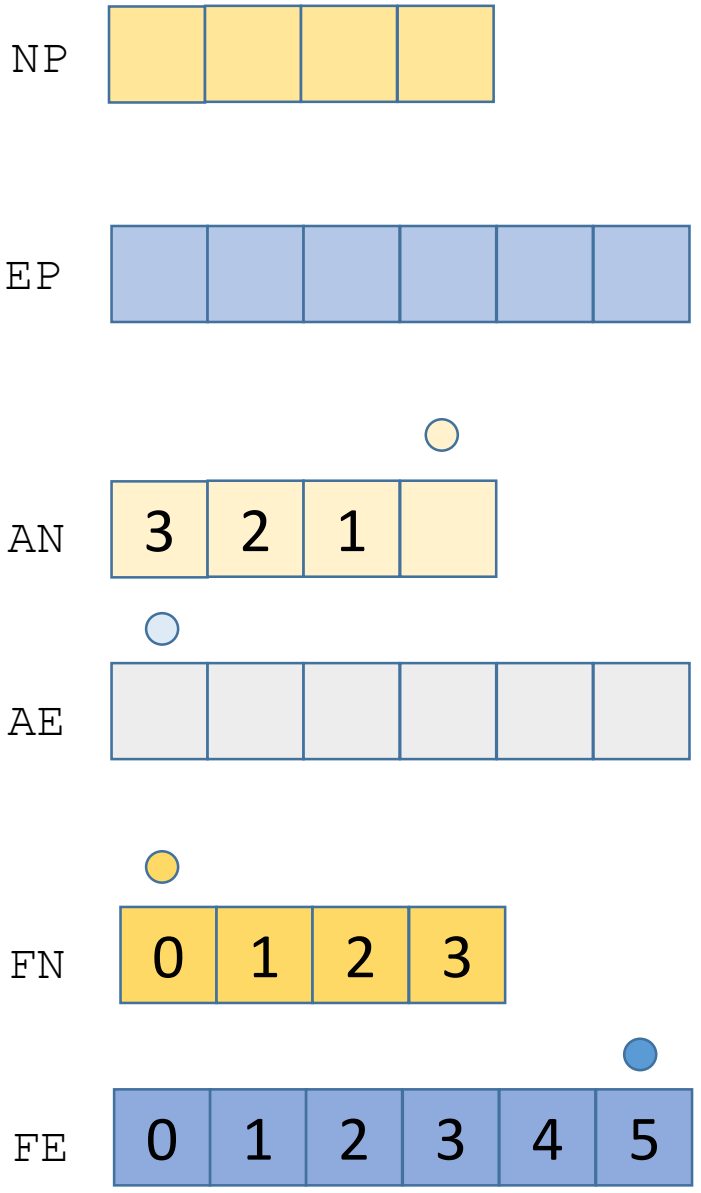
Add 4th Node Step 4:0

index = addNode(…)

```
freeNodeIndex = ?

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



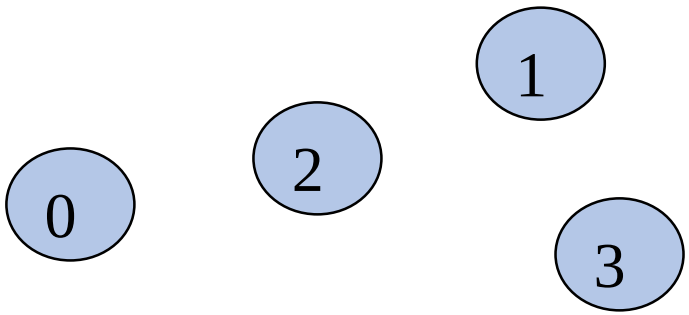


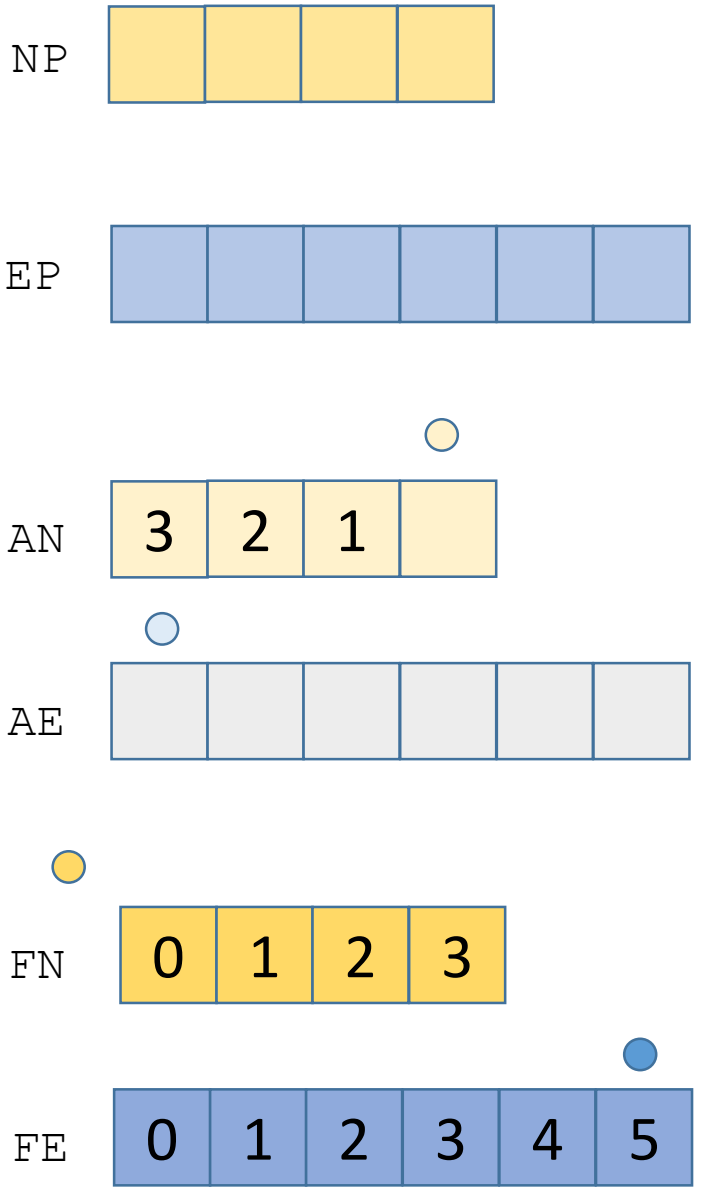
Add 4th Node Step 4:1

index = addNode(…)

```
freeNodeIndex = 0

NODE {
 id = ?
 dynamicID = ?
 edgeID = ?
};
```



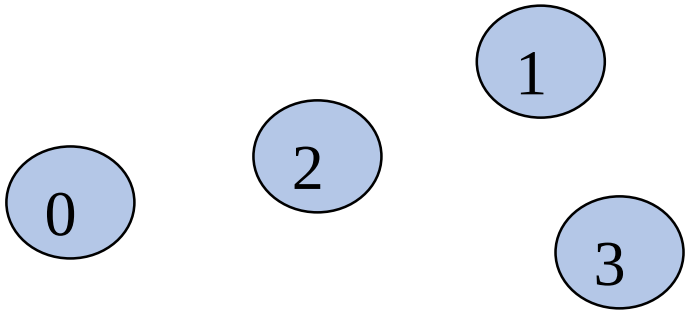


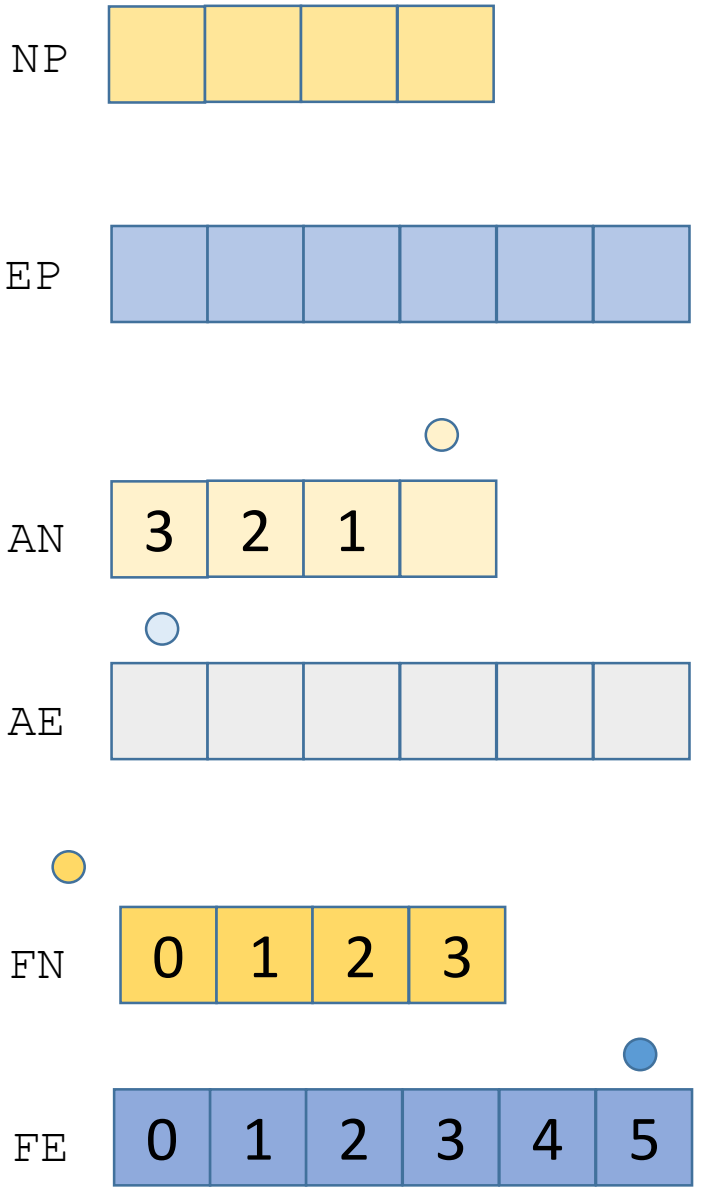
Add 4th Node Step 4:2

index = addNode(…)

```
freeNodeIndex = 0

NODE {
 id = 0
 dynamicID = ?
 edgeID = ?
};
```



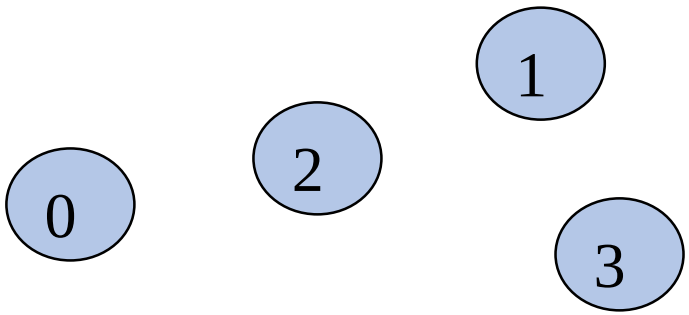


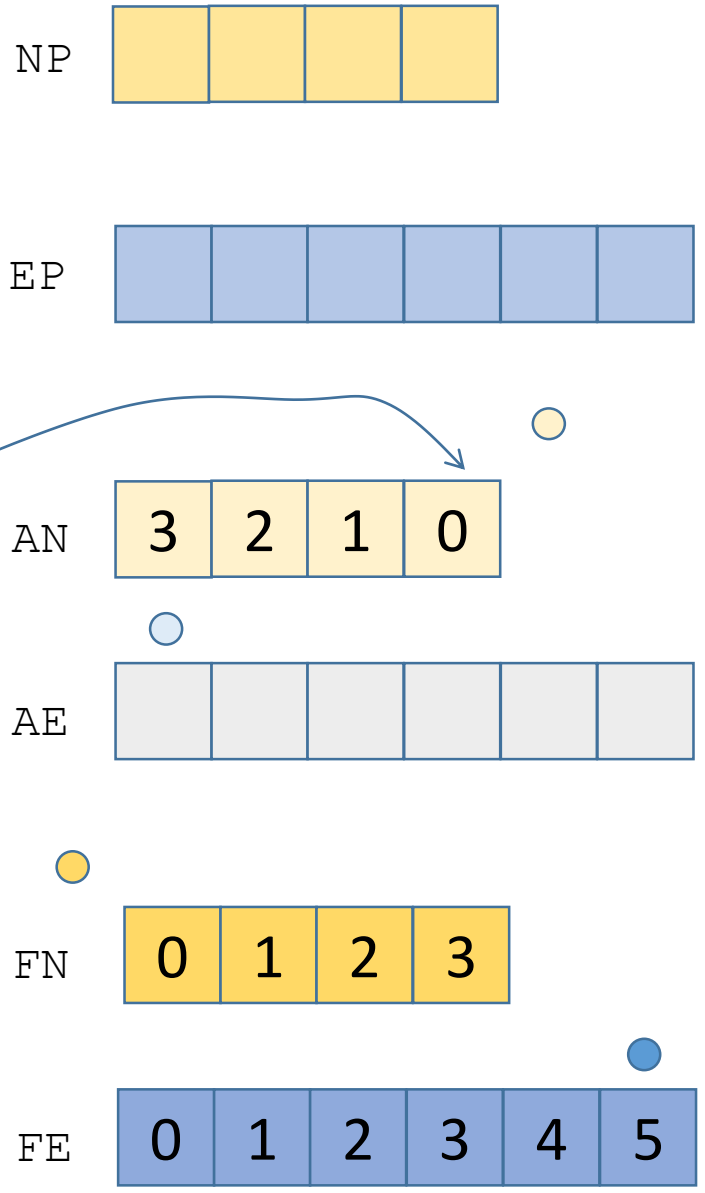
Add 4th Node Step 4:3

index = addNode(…)

```
freeNodeIndex = 0

NODE {
 id = 0
 dynamicID = 3
 edgeID = ?
};
```



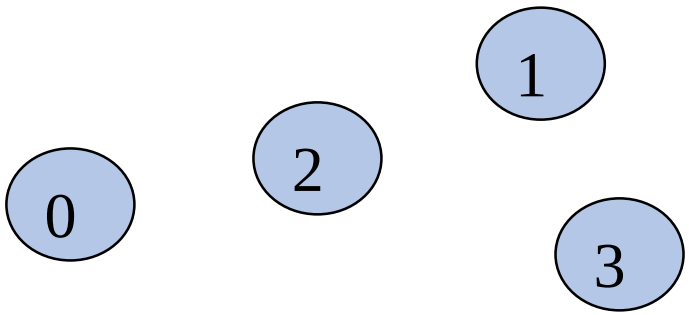


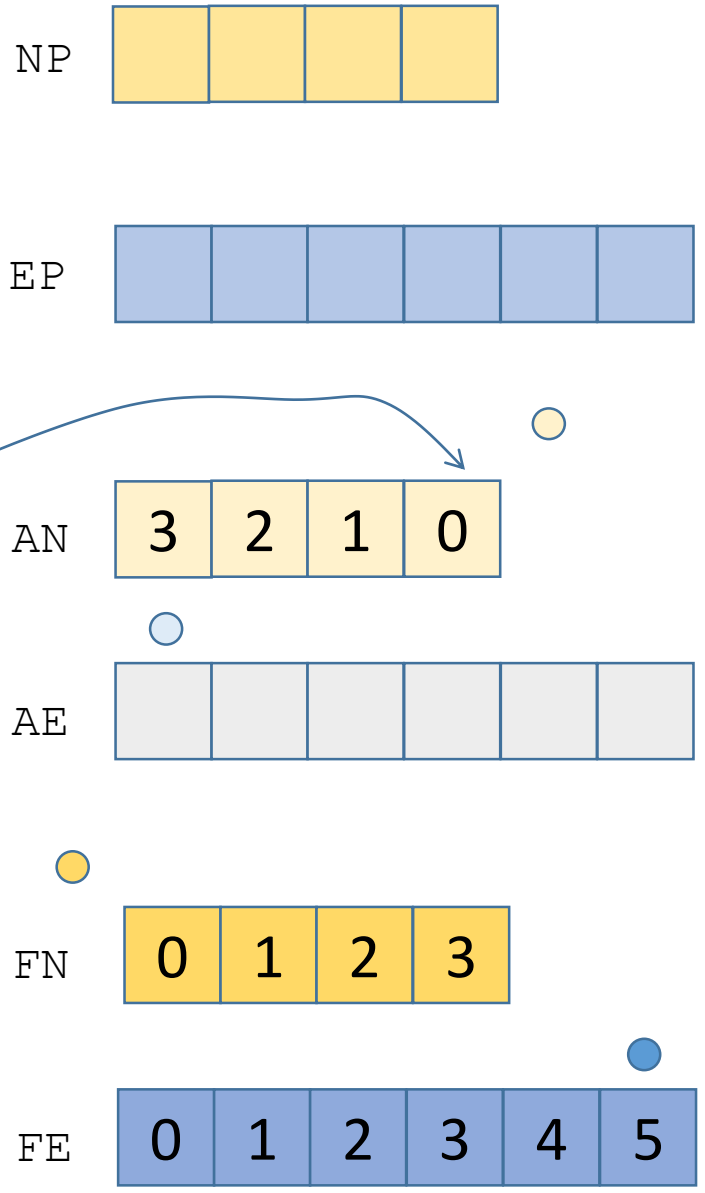
Add 4th Node Step 4:4

index = addNode(…)

```
freeNodeIndex = 0

NODE {
 id = 0
 dynamicID = 3
 edgeID = { }
};
```



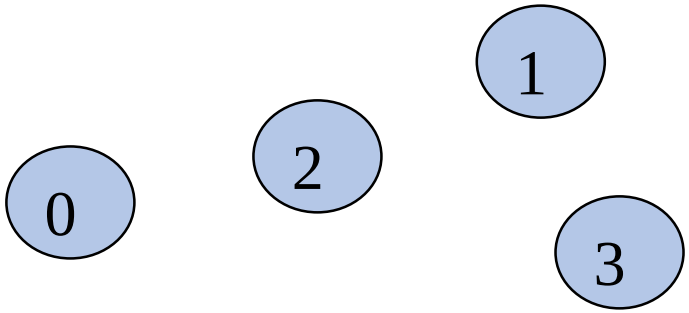


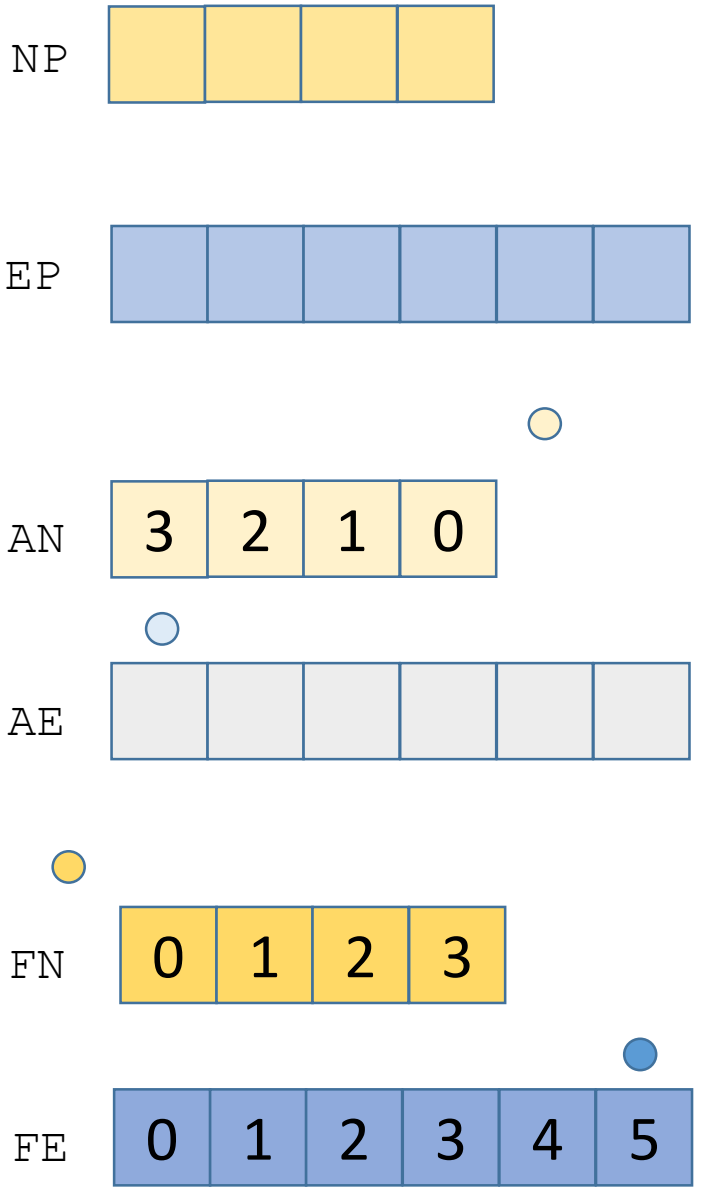
Add an edge Step 5:0

index = addEdge(…)

```
freeEdgeIndex =

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



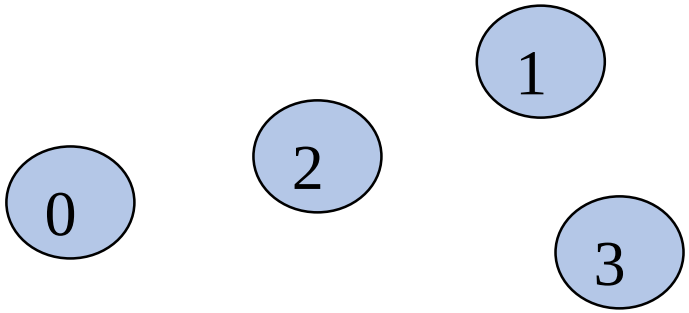


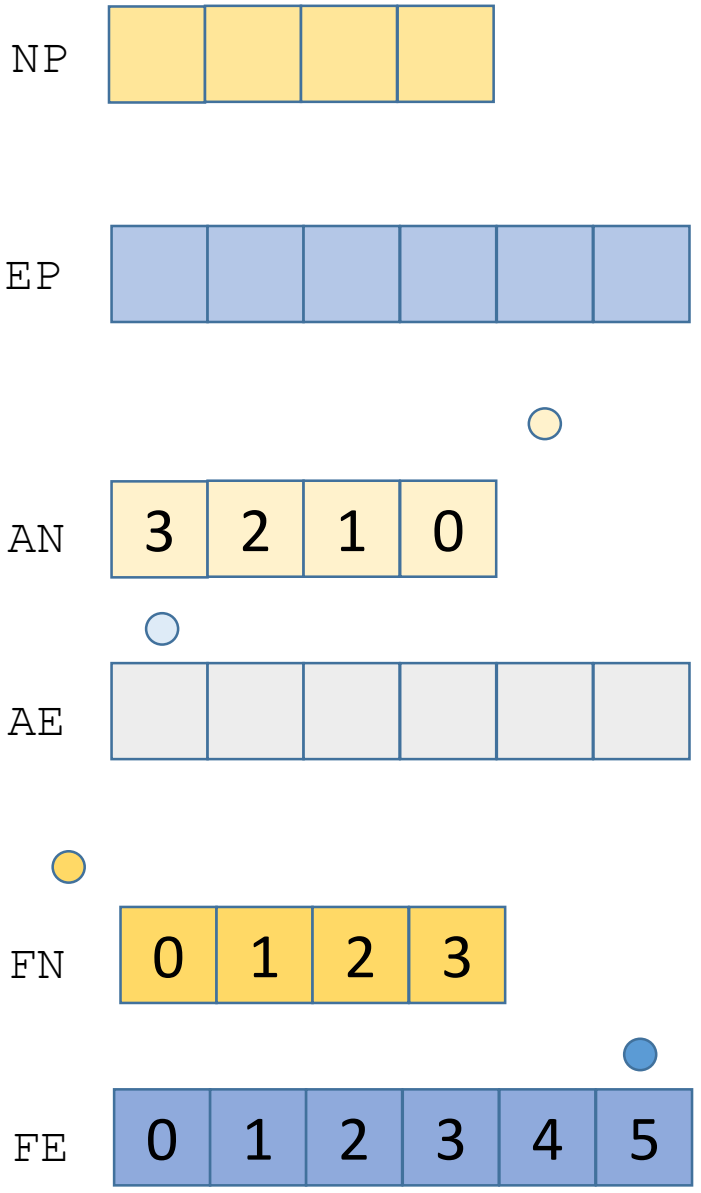
Add an edge Step 5:0

index = addEdge(2, 3)

```
freeEdgeIndex =

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



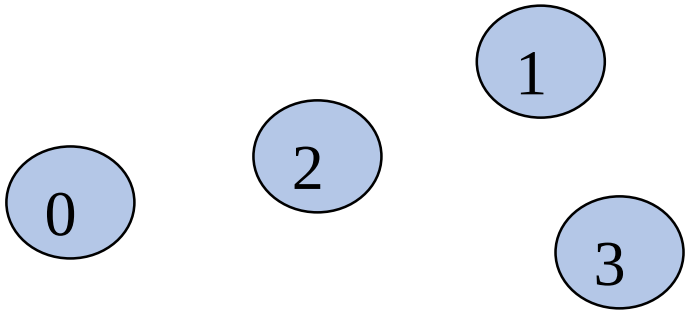


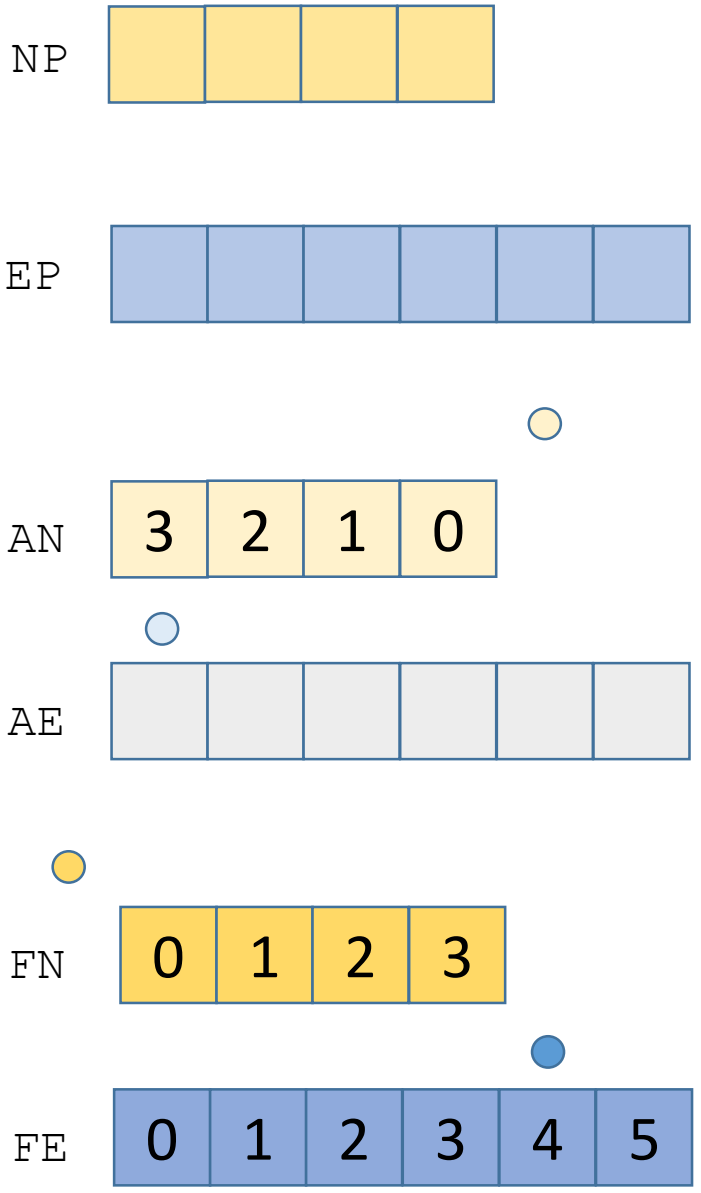
Add an edge Step 5:1

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



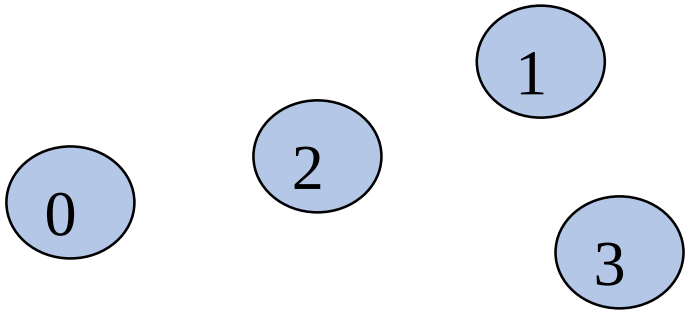


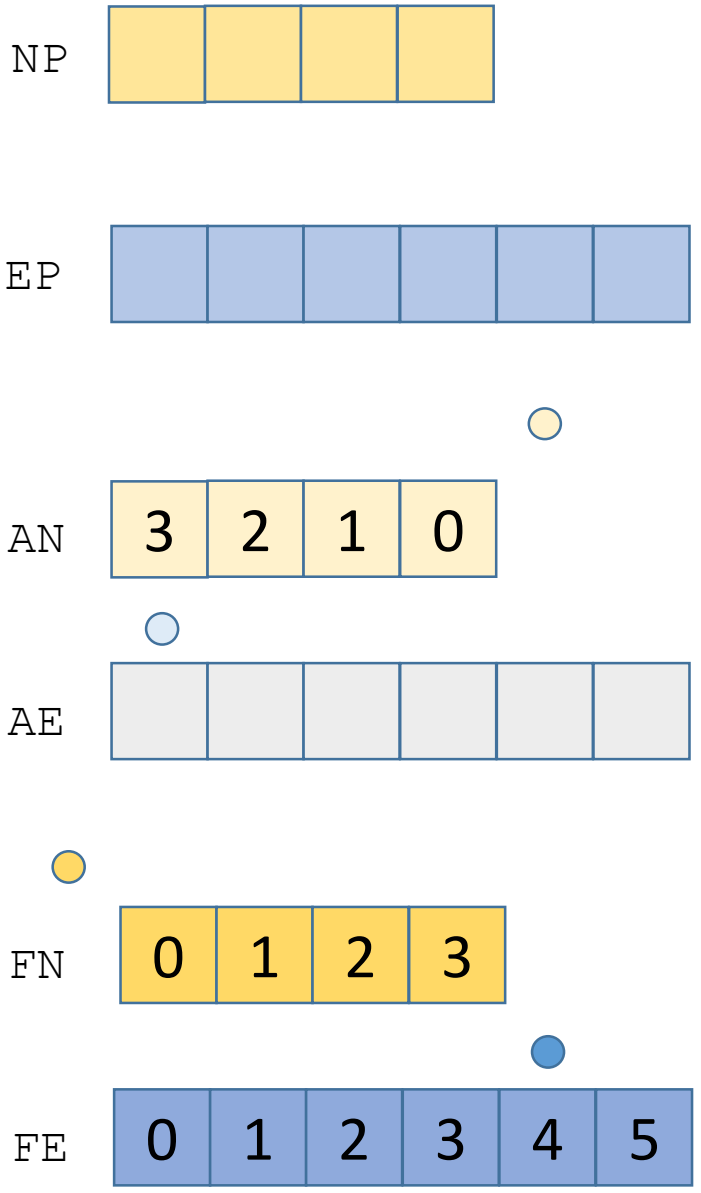
Add an edge Step 5:2

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
    id = 5
    dynamicID = ?
    nodeID[2] = ?
};
```



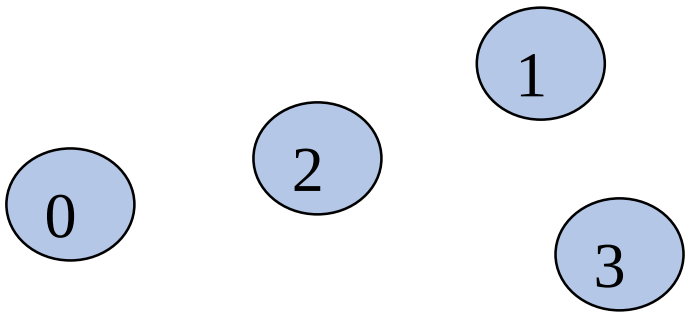


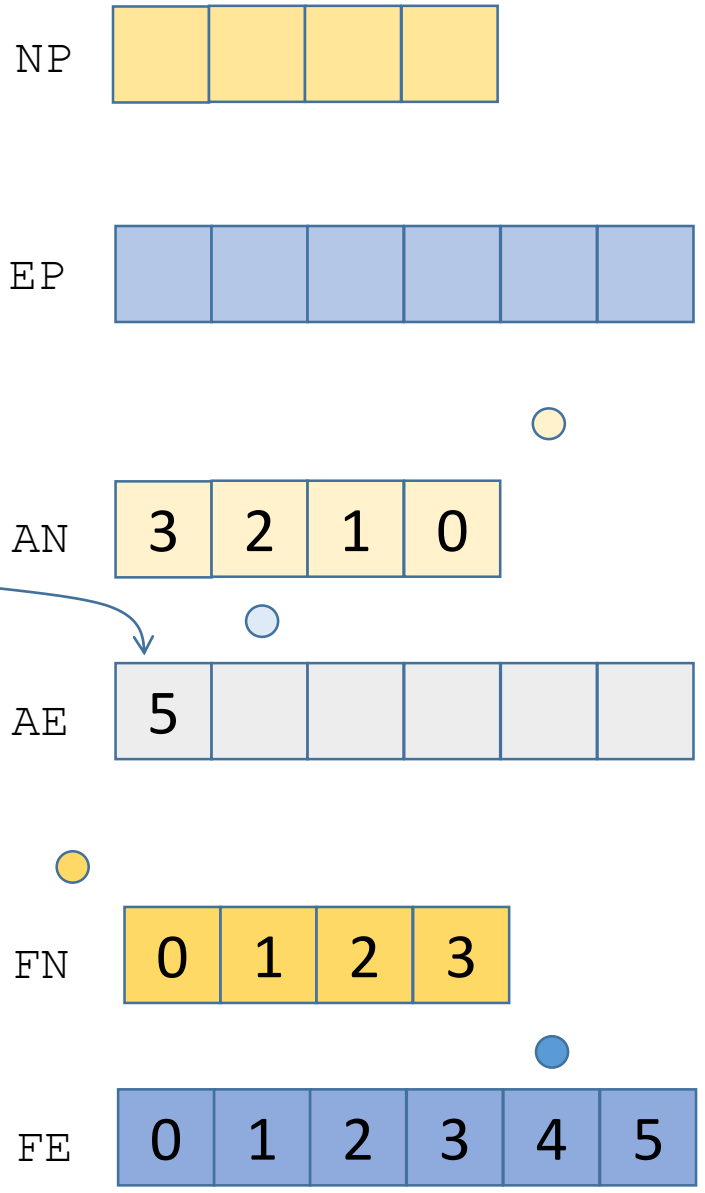
Add an edge Step 5:3

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = ?
};
```



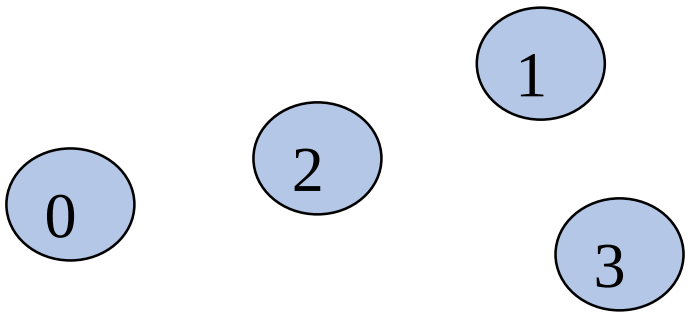


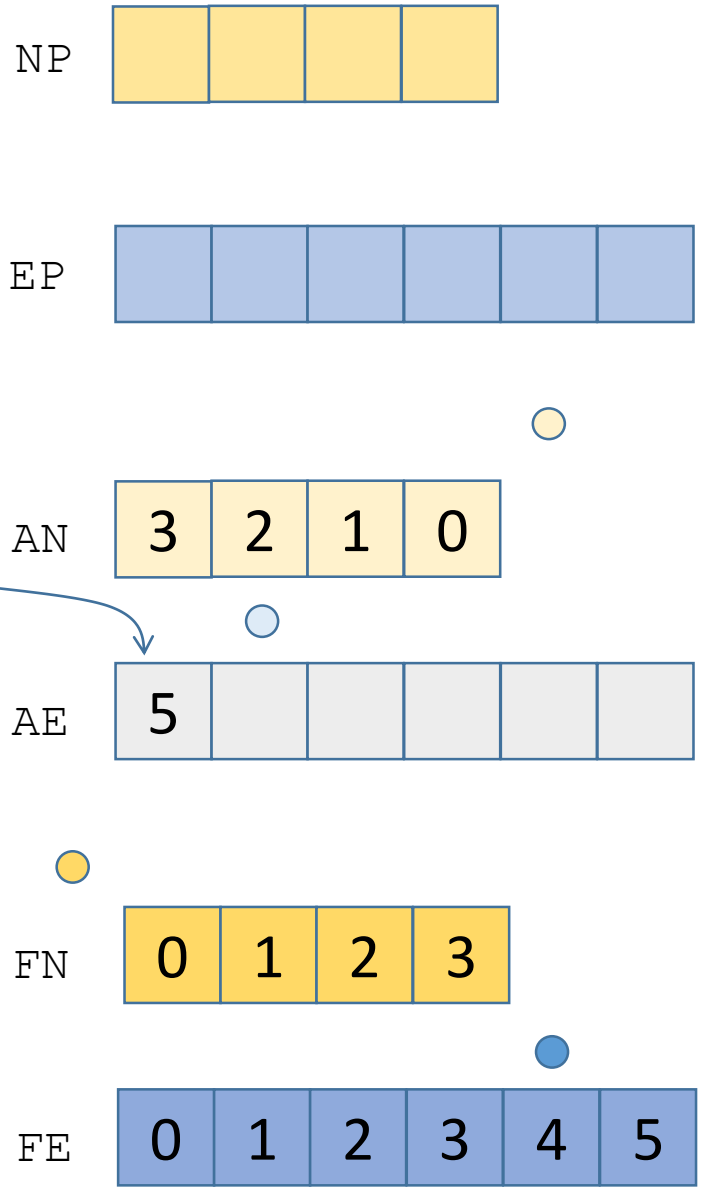
Add an edge Step 5:4

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
    id = 5
    dynamicID = 0
    nodeID[2] = {2, 3}
};
```



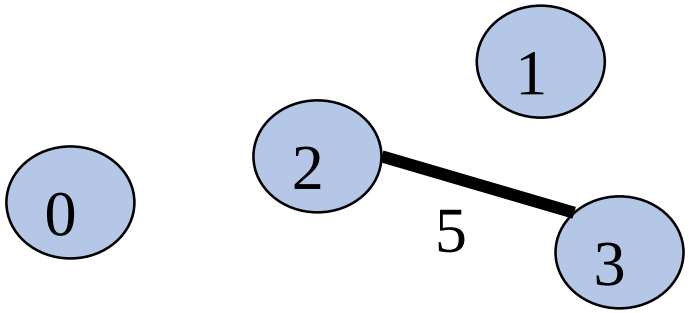


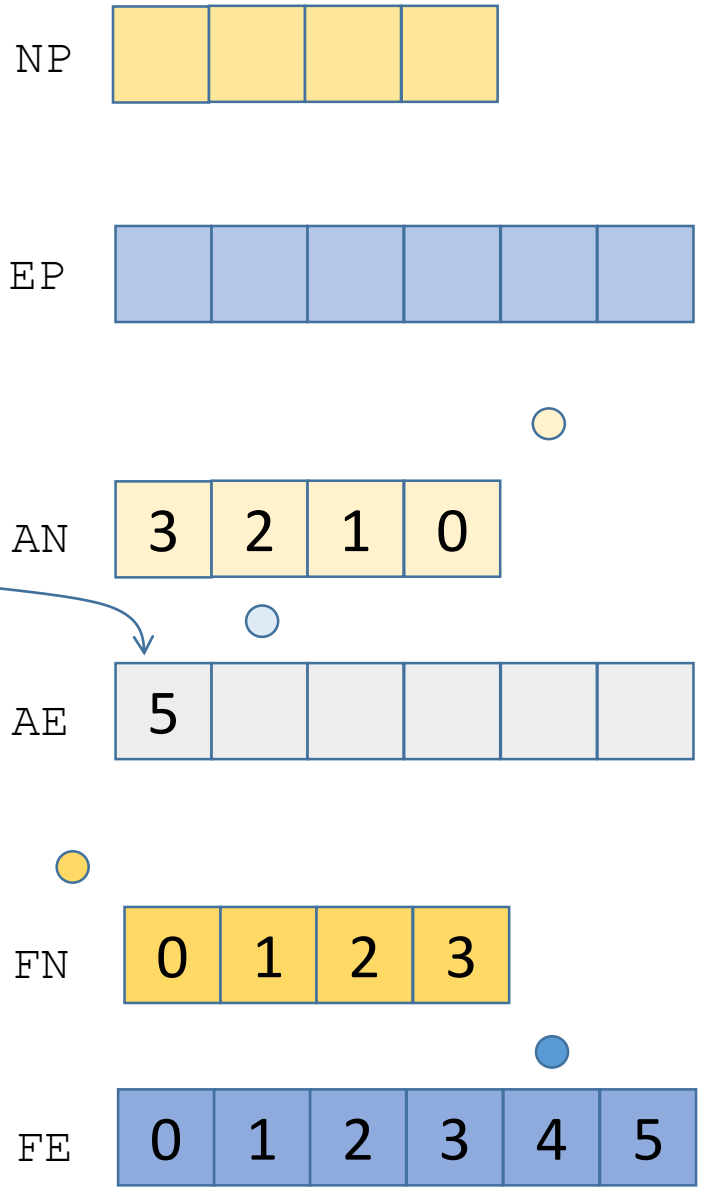
Add an edge
Step 5:4

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
    id = 5
    dynamicID = 0
    nodeID[2] = {2, 3}
};
```





We draw the edge for visualization.

Add an edge Step 5:5

Add edge ID to the two nodes.

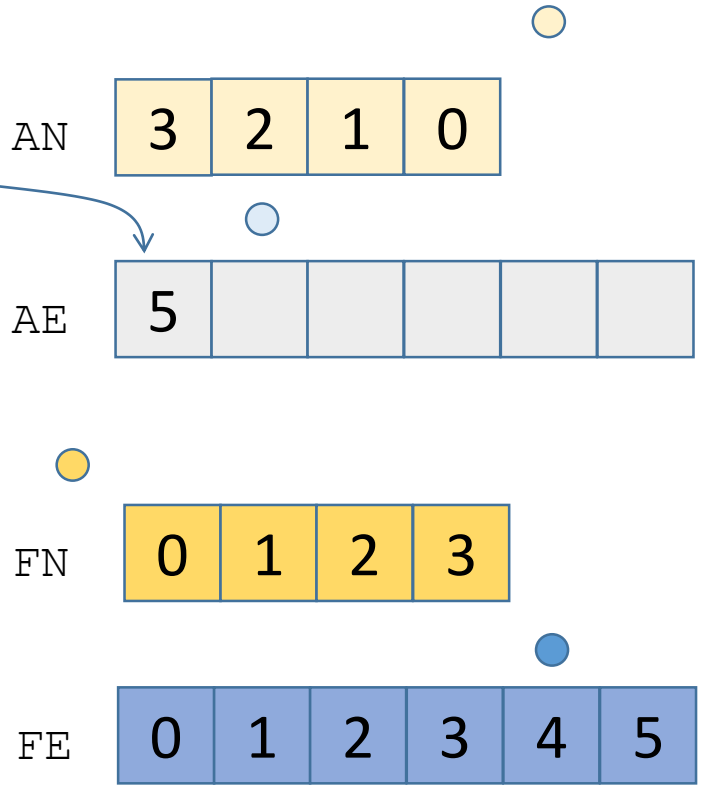
```
NODE {
 id = 3
 dynamicID = 0
 edgeID = {5}
};
```

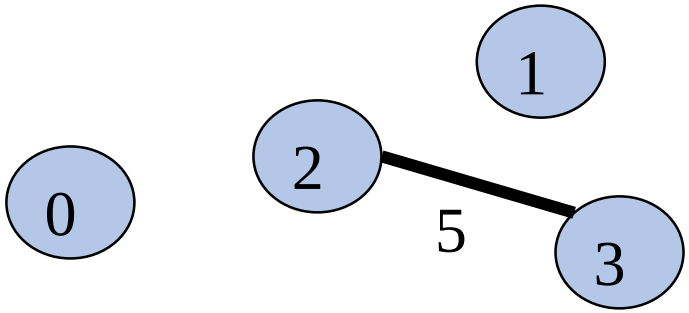
```
NODE {
 id = 2
 dynamicID = 1
 edgeID = {5}
};
```

index = addEdge(2, 3)

```
freeEdgeIndex = 5

EDGE {
    id = 5
    dynamicID = 0
    nodeID[2] = {2, 3}
};
```





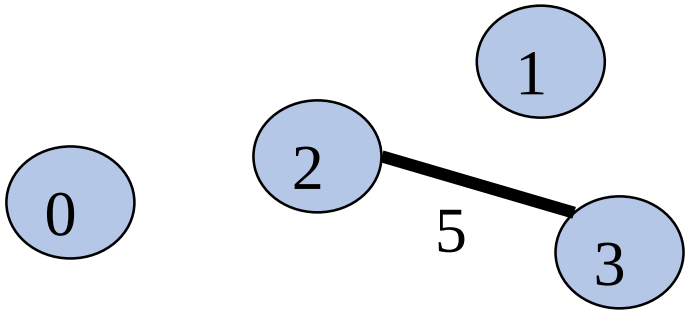
We draw the edge for visualization.

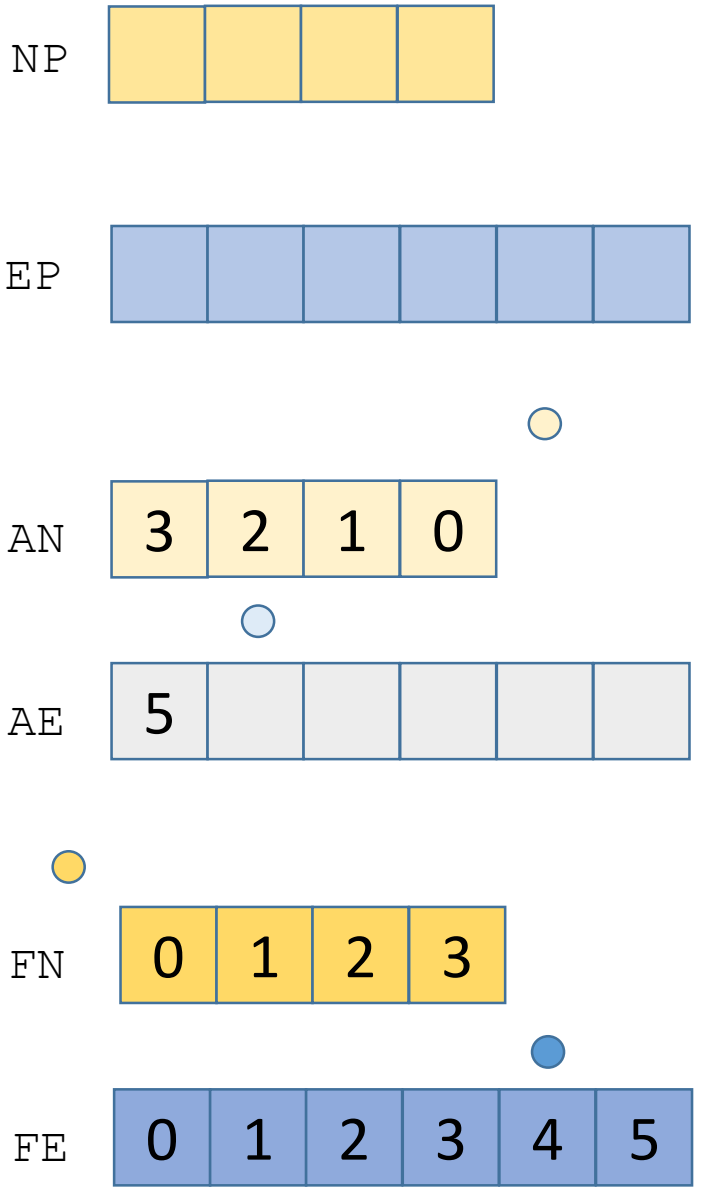
Add 2nd edge Step 6:0

index = addEdge(0, 1)

```
freeEdgeIndex = ?

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



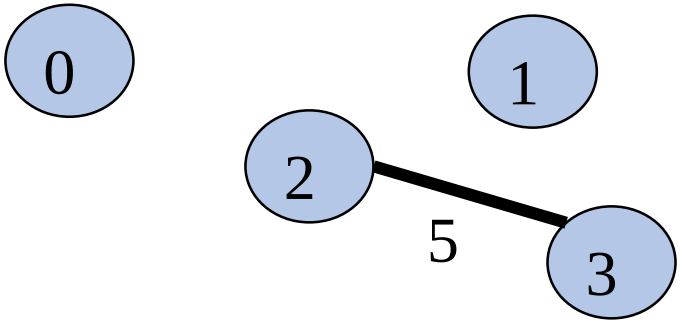


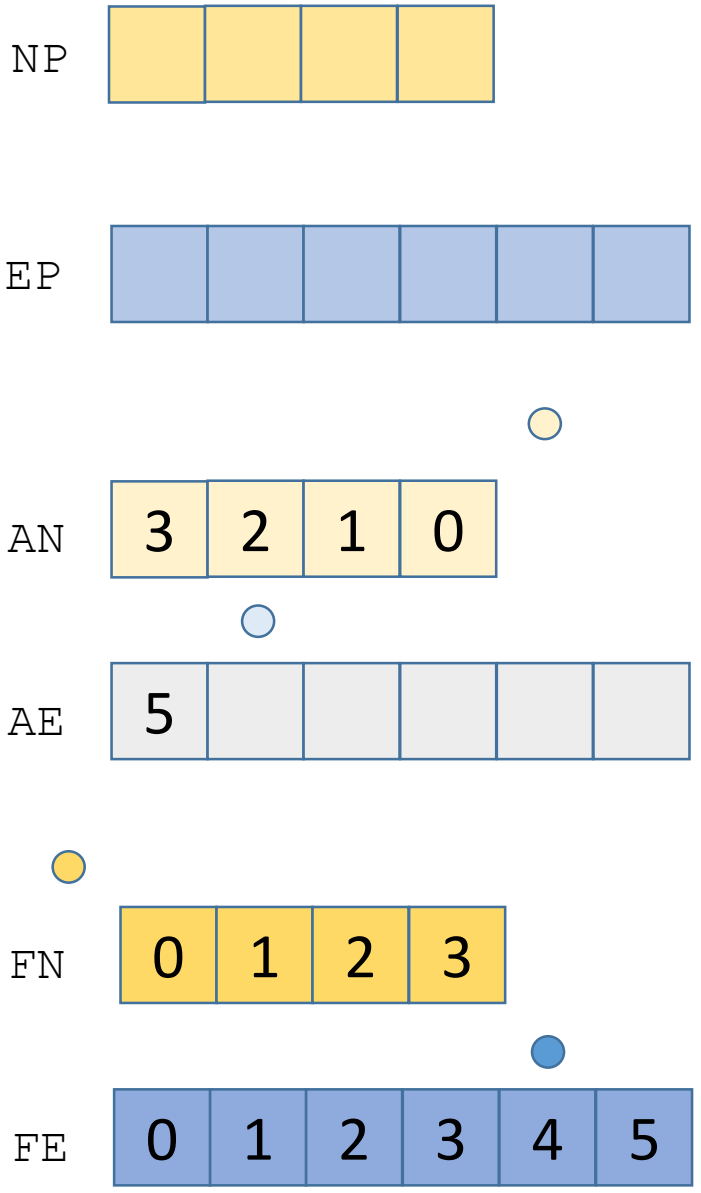
Add 2nd edge Step 6:0

index = addEdge(0, 1)

```
freeEdgeIndex = ?

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



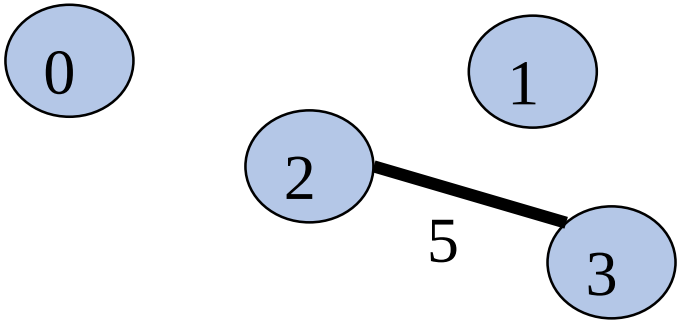


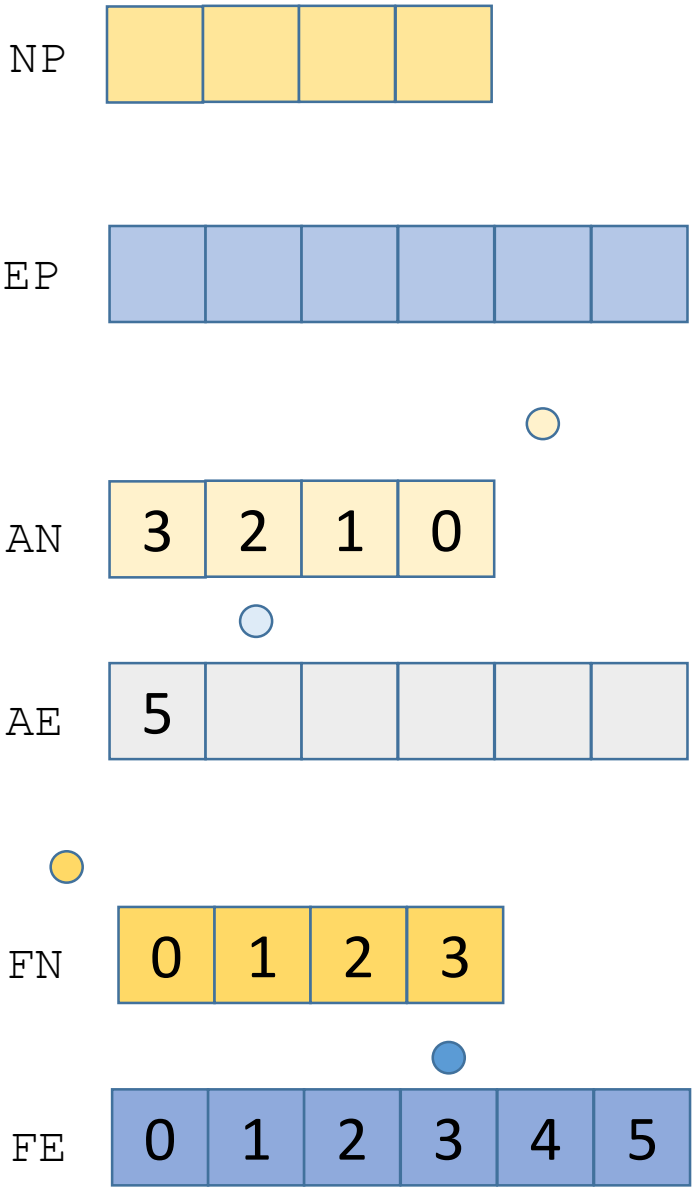
Add 2nd edge Step 6:1

index = addEdge(0, 1)

```
freeEdgeIndex = 4

EDGE {
    id = ?
    dynamicID = ?
    nodeID[2] = ?
};
```



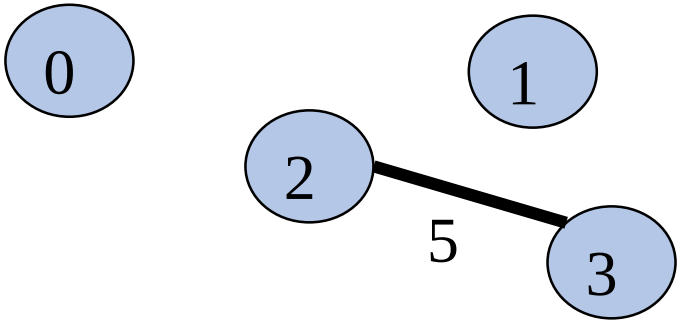


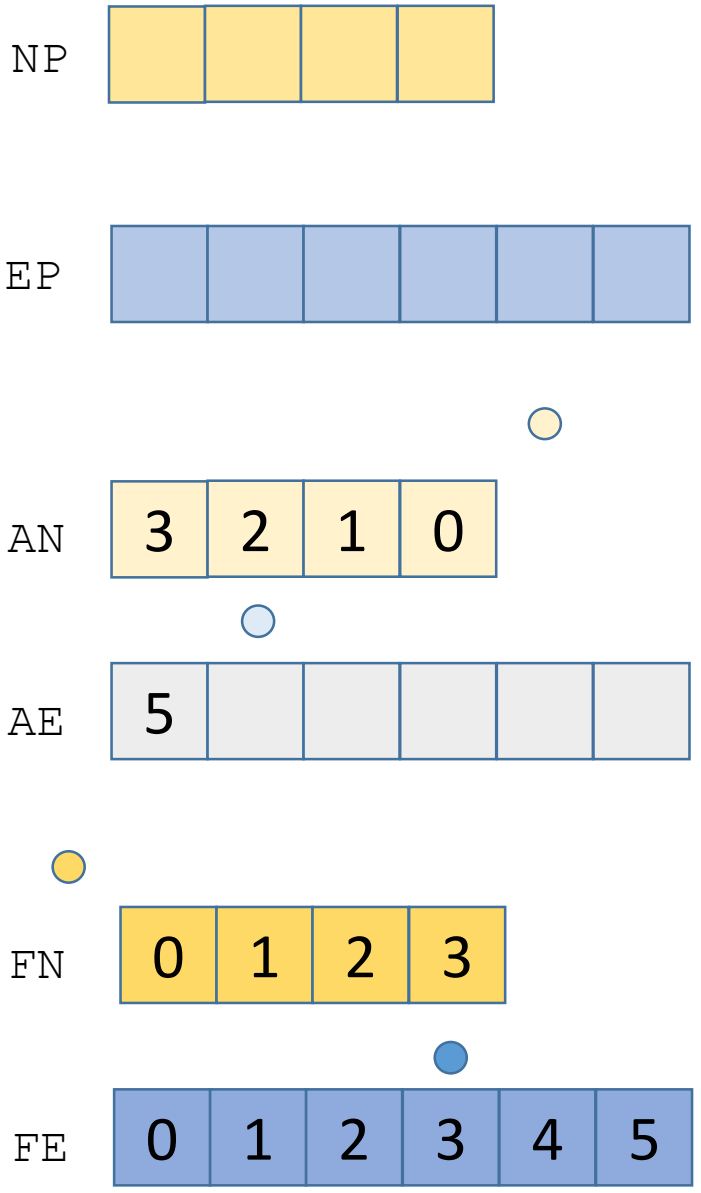
Add 2nd edge Step 6:2

index = addEdge(0, 1)

```
freeEdgeIndex = 4

EDGE {
    id = 4
    dynamicID = ?
    nodeID[2] = ?
};
```



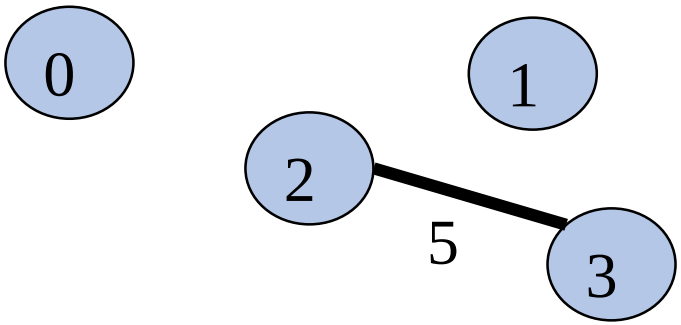


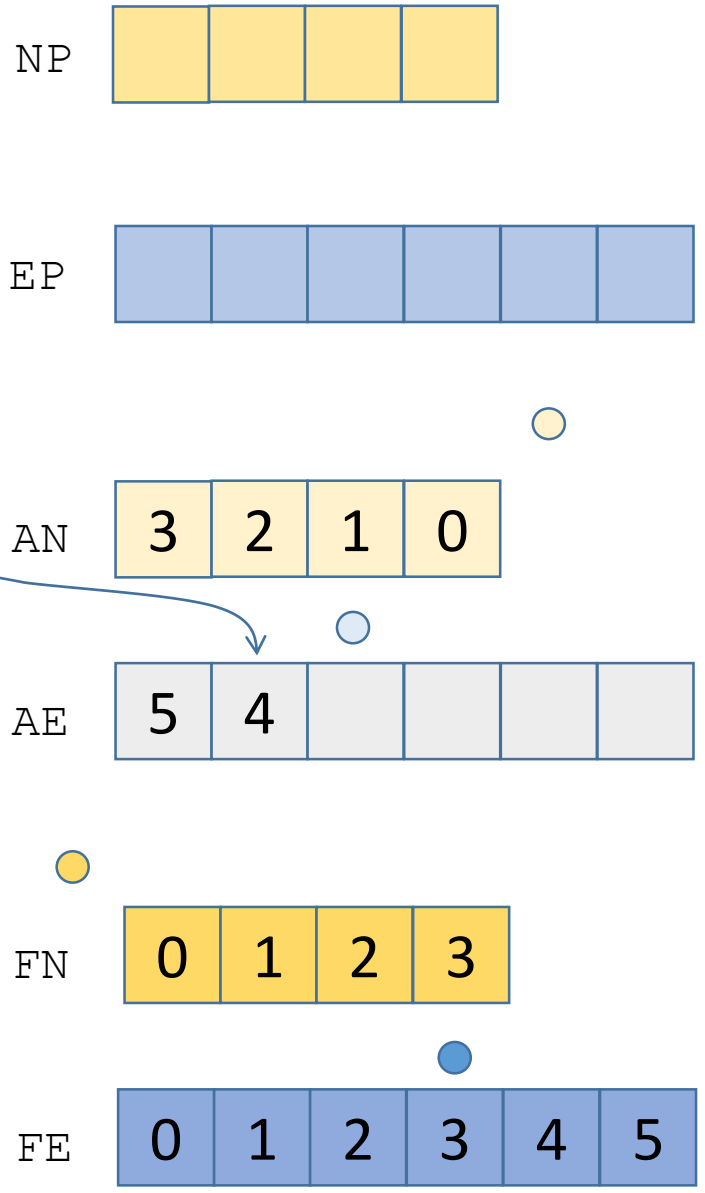
Add 2nd edge Step 6:3

index = addEdge(0, 1)

```
freeEdgeIndex = 4

EDGE {
    id = 4
    dynamicID = 1
    nodeID[2] = ?
};
```



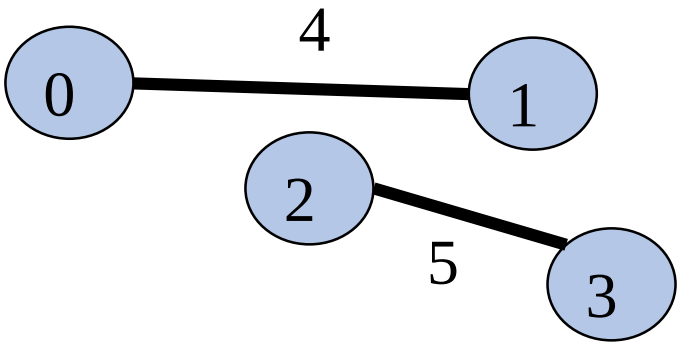


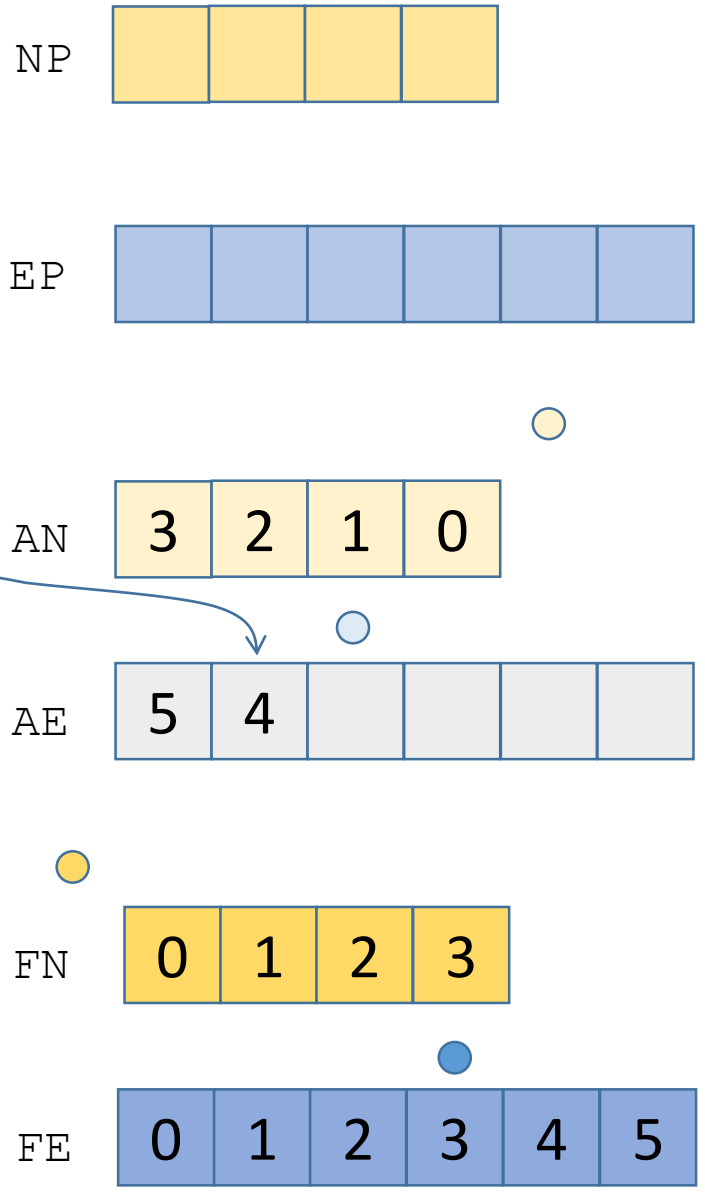
Add 2nd edge Step 6:4

index = addEdge(0, 1)

```
freeEdgeIndex = 4

EDGE {
    id = 4
    dynamicID = 1
    nodeID[2] = {0, 1}
};
```





Add 2nd edge Step 6:5

Add edge ID to the two nodes.

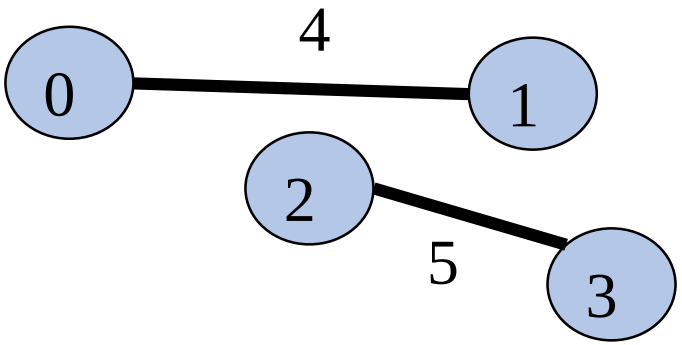
```
NODE {
 id = 1
 dynamicID = 2
 edgeID = {4}
};
```

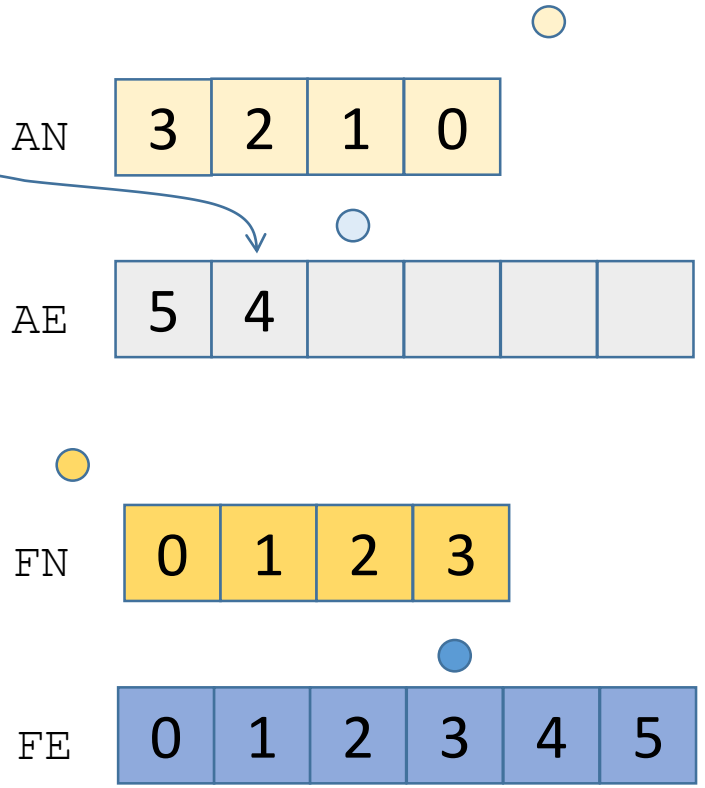
```
NODE {
 id = 0
 dynamicID = 3
 edgeID = {4}
};
```

index = addEdge(0, 1)

```
freeEdgeIndex = 4

EDGE {
    id = 4
    dynamicID = 1
    nodeID[2] = {0, 1}
};
```





Exercises: Implement functions for performing edge deletion and node deletion?

- When we delete a node, collect the adjacency edges.
  - Delete the edges first. Then delete the node.

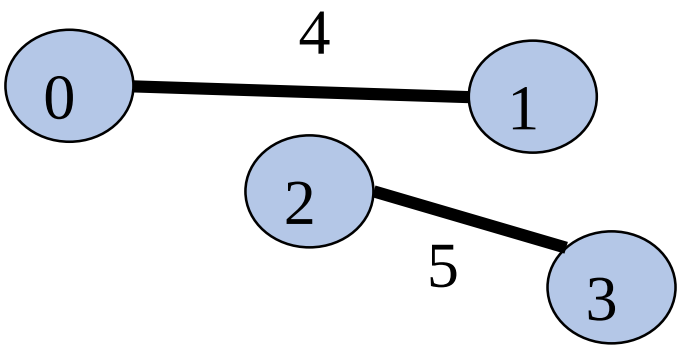
- When we delete an edge, simply delete edge. Get its dynamic ID, and then free its unique ID.

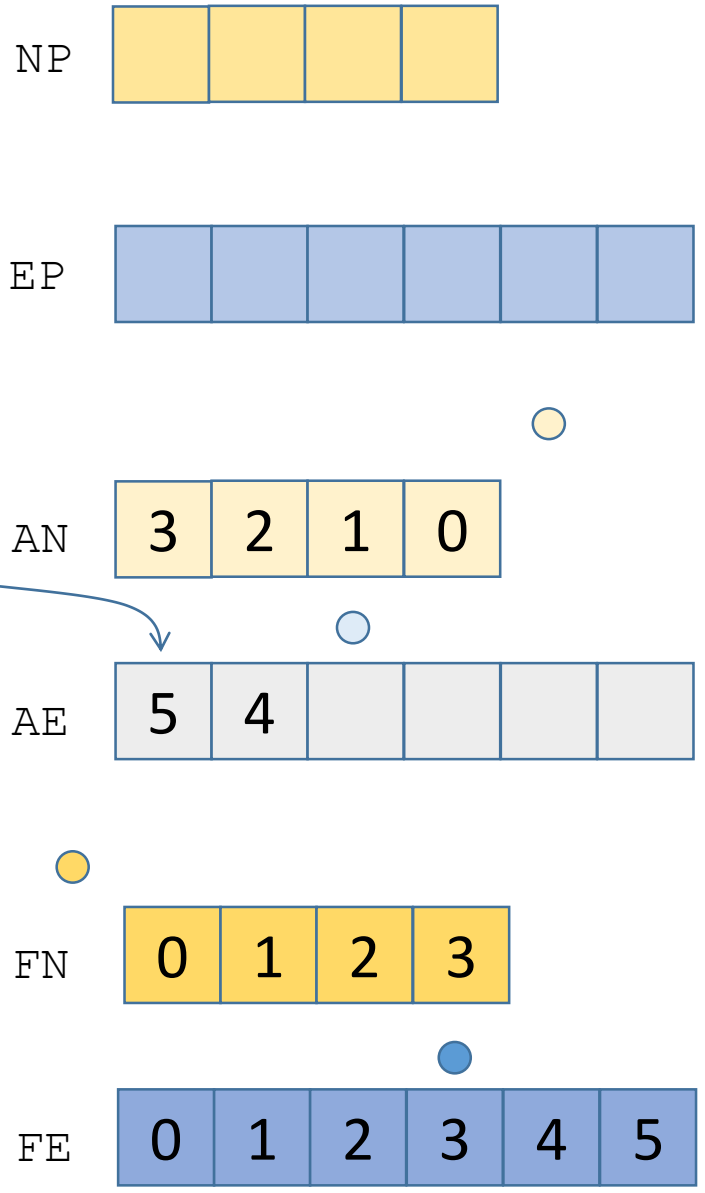
- Note that we do not really free the memory space allocated for the nodes or edges. We only collect their used IDs and assign them back to the free node array or the free edge array.

Delete edge 5
Step 7:0

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```



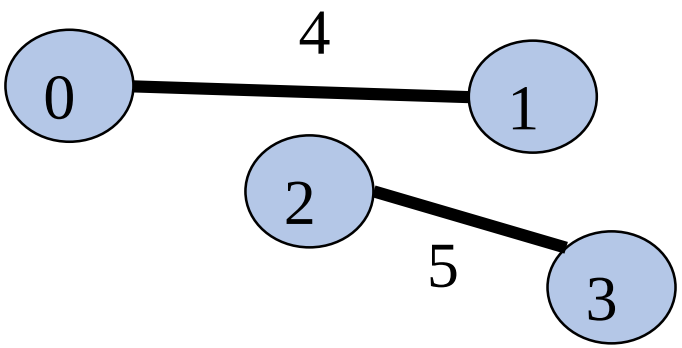


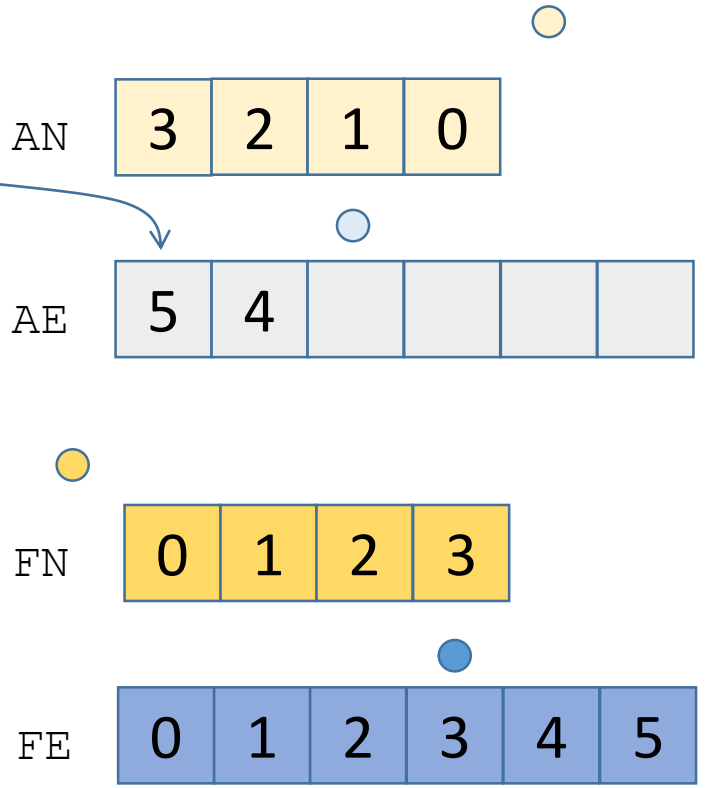
Delete edge 5 Step 7:0

We need to maintain the correctness of the data in the data structure.

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```



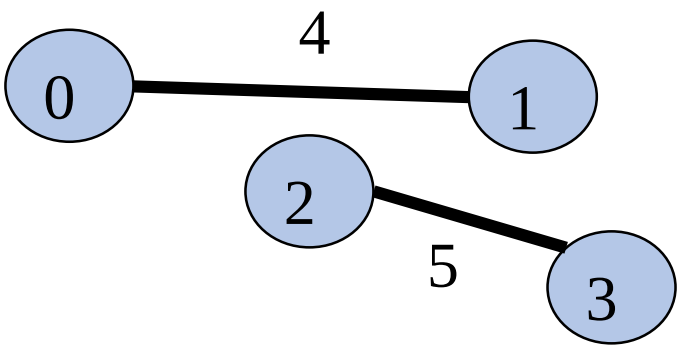


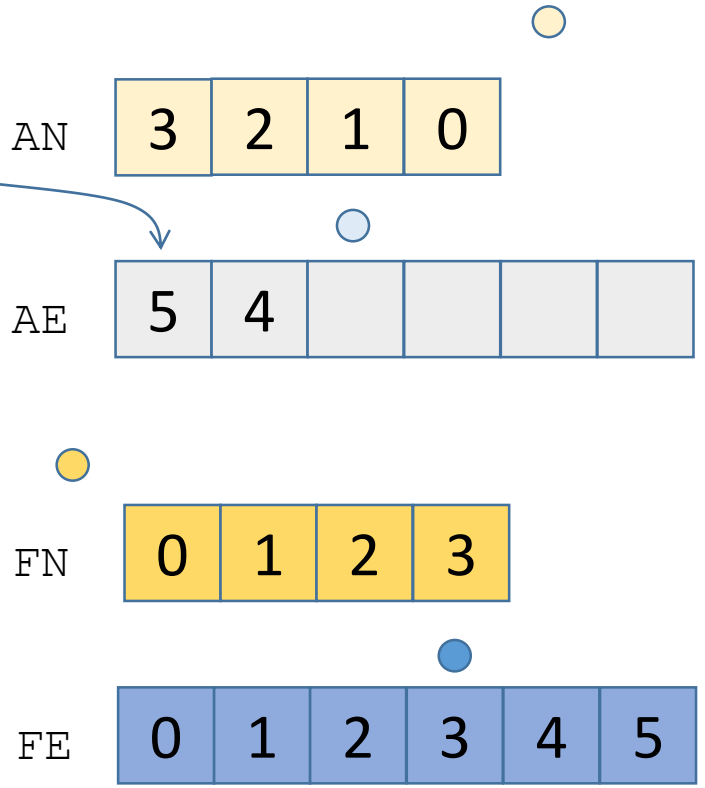
Delete edge 5 Step 7:0

Get the ID of the edge. Assign it back to FE (FreeEdgeArr).

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```



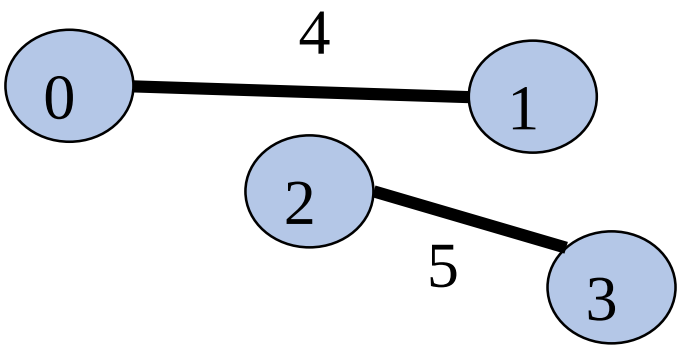


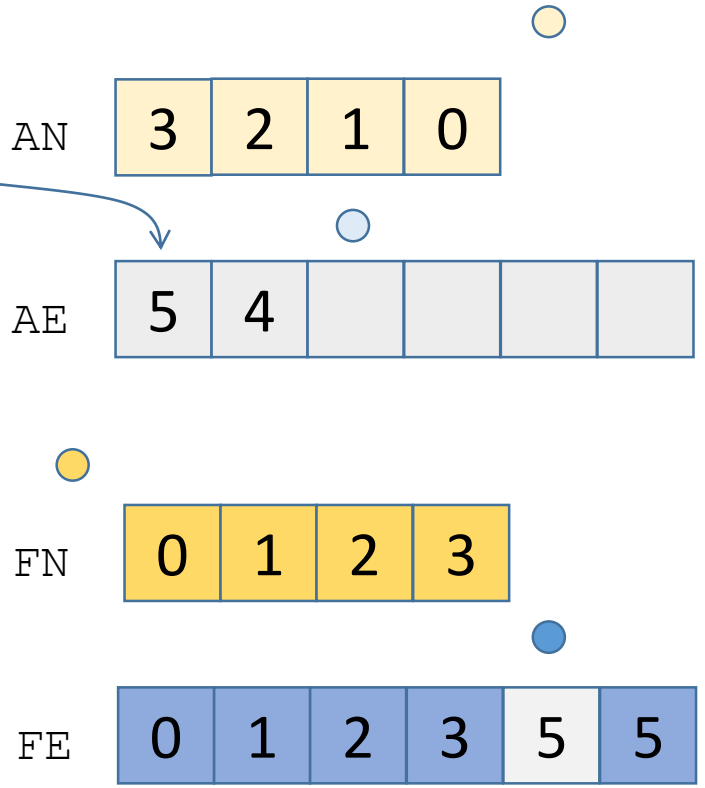
Delete edge 5 Step 7:1

Get the ID of the edge. Assign it back to FE (FreeEdgeArr).

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```



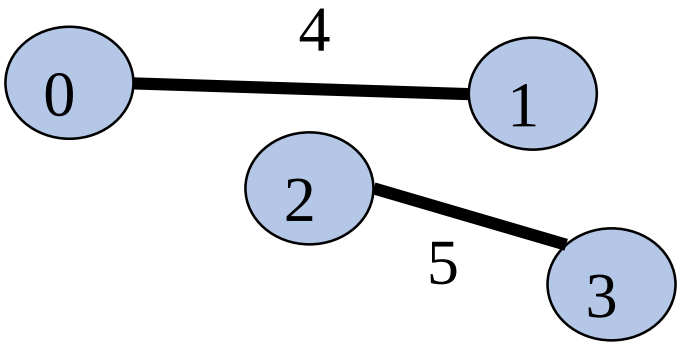


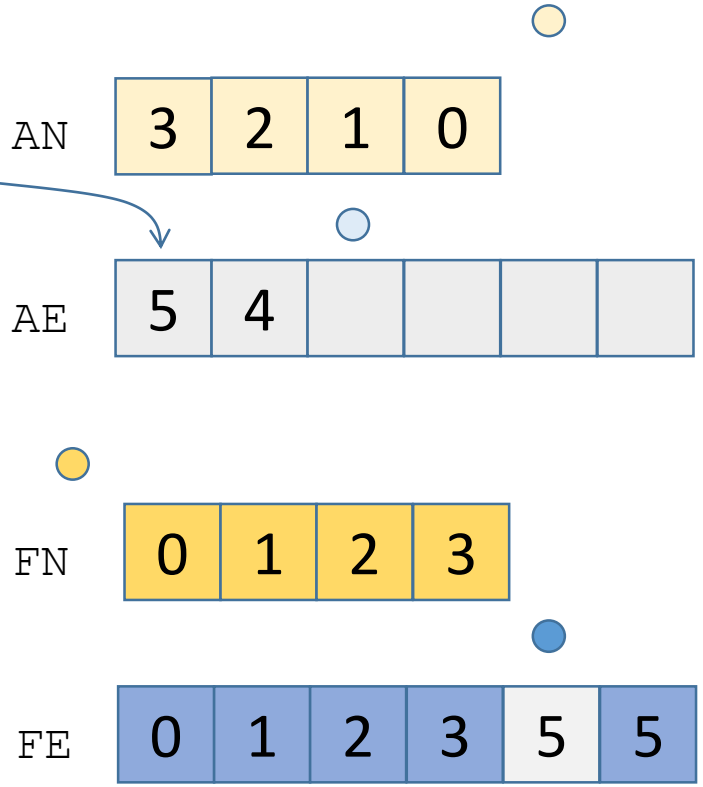
Delete edge 5 Step 7:2

Get the dynamic ID of the edge. Need to delete it from AE (active edge array).

deleteEdge(2)

```
EDGE {
    id = 5
    dynamicID = 0
    nodeID[2] = {2, 3}
};
```



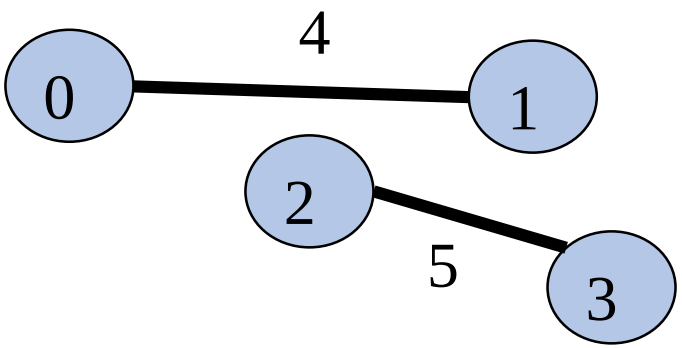


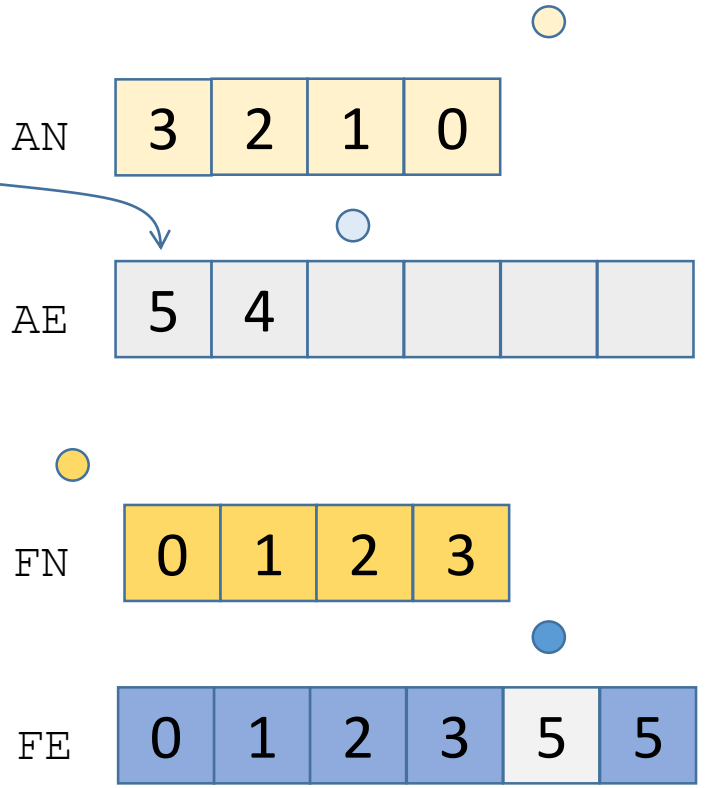
Delete edge 5 Step 7:2

deleteEdge(2)

Get the dynamic ID of the edge. Need to delete it from AE (active edge array). **Move the last element to its position. Then decrease the count.**

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```



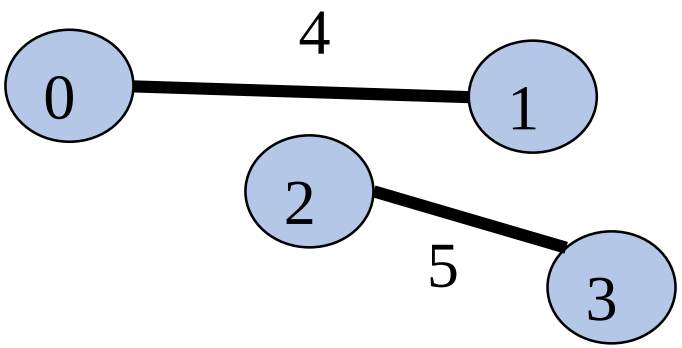


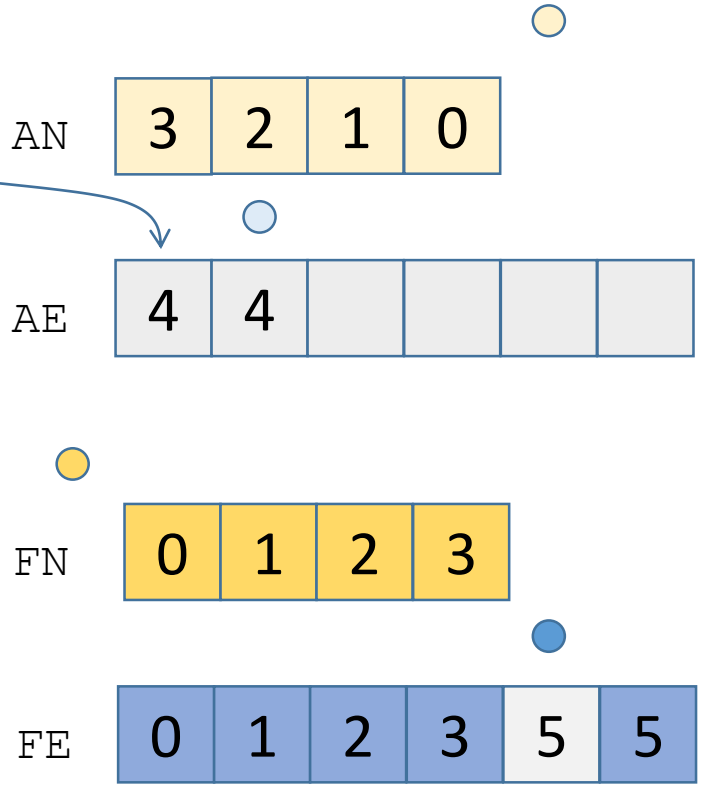
Delete edge 5 Step 7:2

deleteEdge(2)

Get the dynamic ID of the edge. Need to delete it from AE (active edge array). **Move the last element to its position. Then decrease the count.**

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```





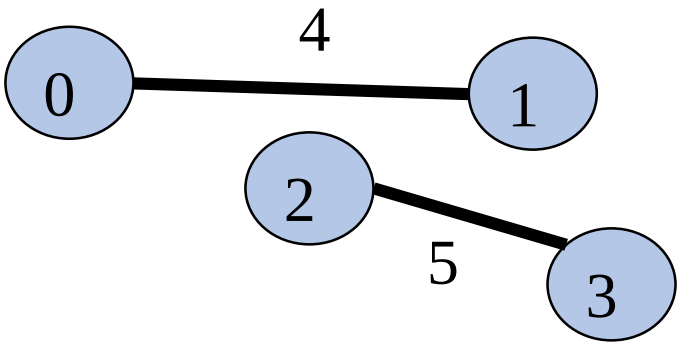
Delete edge 5 Step 7:2

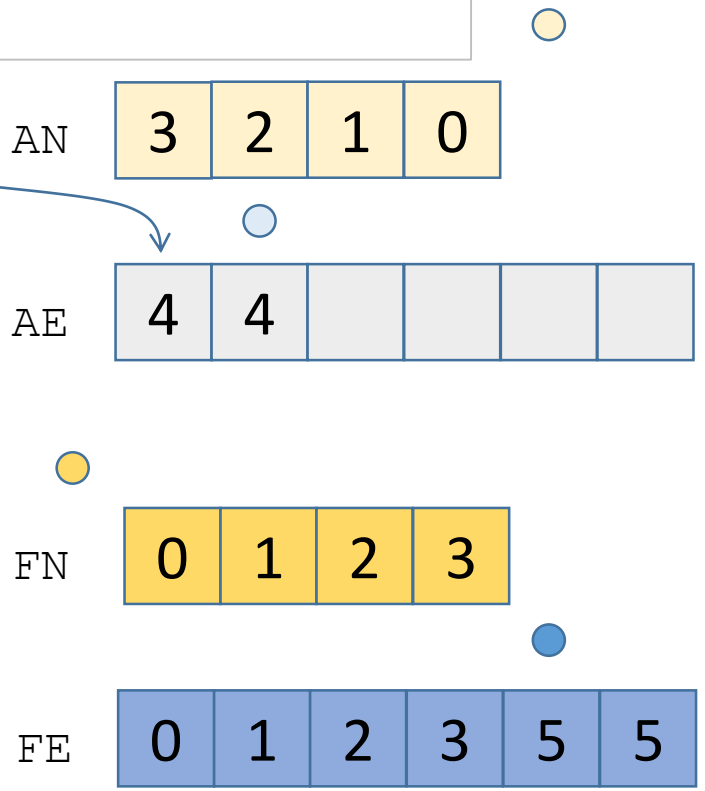
Update the dynamic ID of the edge 4 because its dynamic ID is changed to 0.

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```

```
EDGE {
     id = 4
     dynamicID = 1 -> 0
     nodeID[2] = {0, 1}
};
```





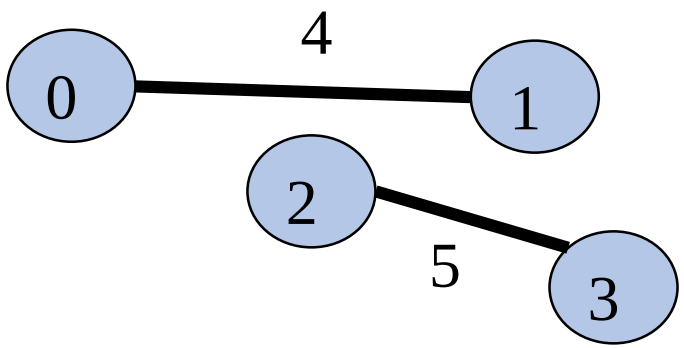
Delete edge 5 Step 7:2

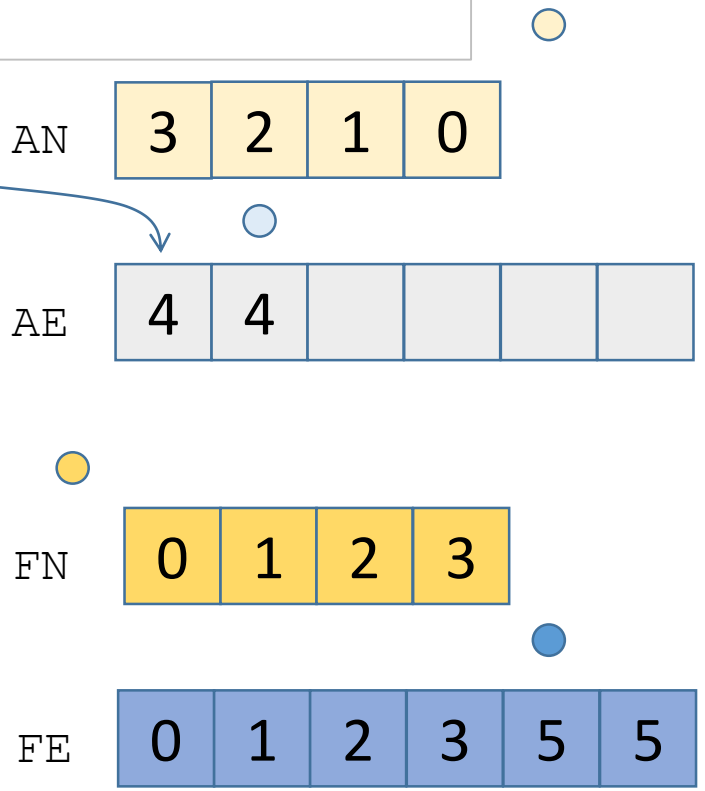
Update the dynamic ID of the edge 4 because its dynamic ID is changed to 0.

deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```

```
EDGE {
     id = 4
     dynamicID = 0
     nodeID[2] = {0, 1}
};
```





Delete edge 5 Step 7:3

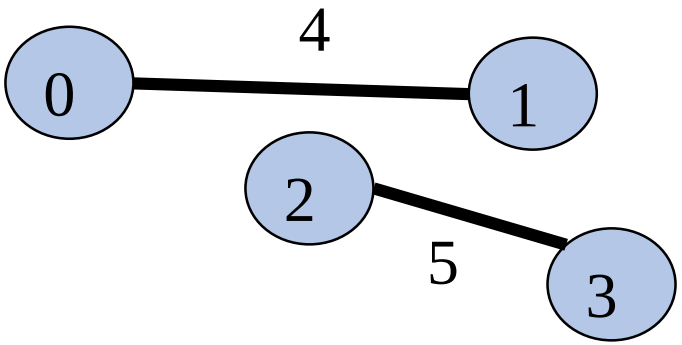
Now, delete the edge IDs in the nodes connecting the edge.

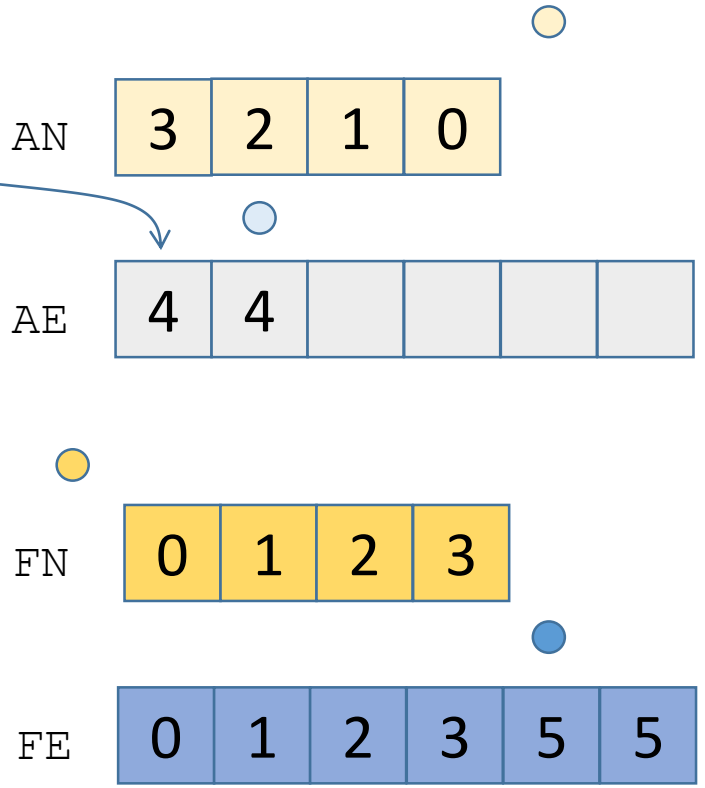
deleteEdge(2)

```
NODE {
 id = 2
 dynamicID = 1
 edgeID = {5}
};
```

```
NODE {
 id = 3
 dynamicID = 0
 edgeID = {5}
};
```

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```





Delete edge 5 Step 7:3

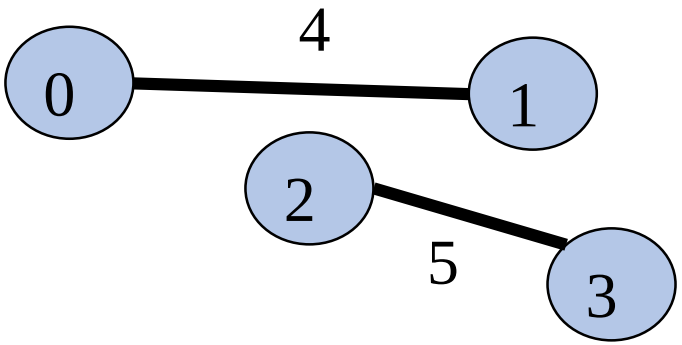
Now, delete the edge IDs in the nodes connecting the edge.

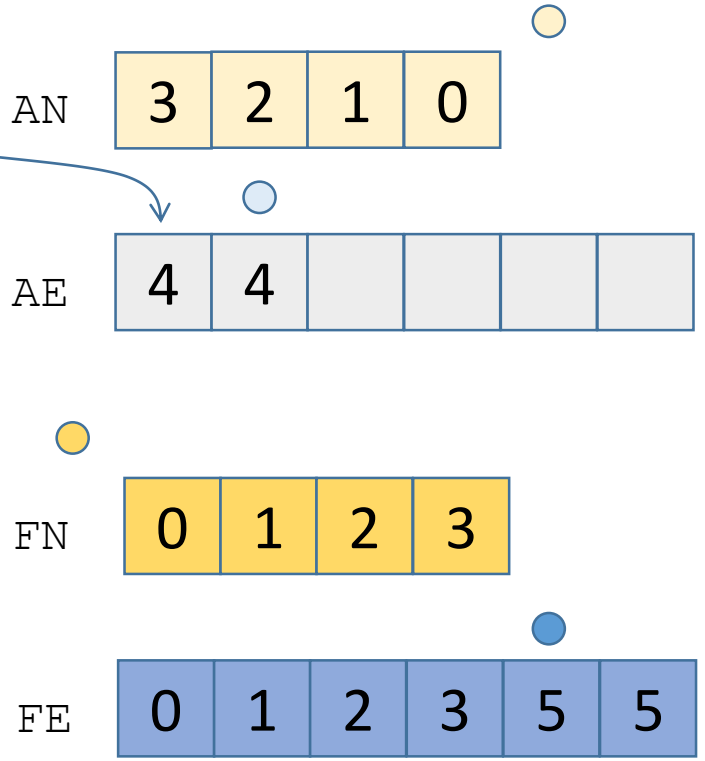
```
NODE {
 id = 2
 dynamicID = 1
 edgeID = { }
};
```

```
NODE {
 id = 3
 dynamicID = 0
 edgeID = { }
};
```

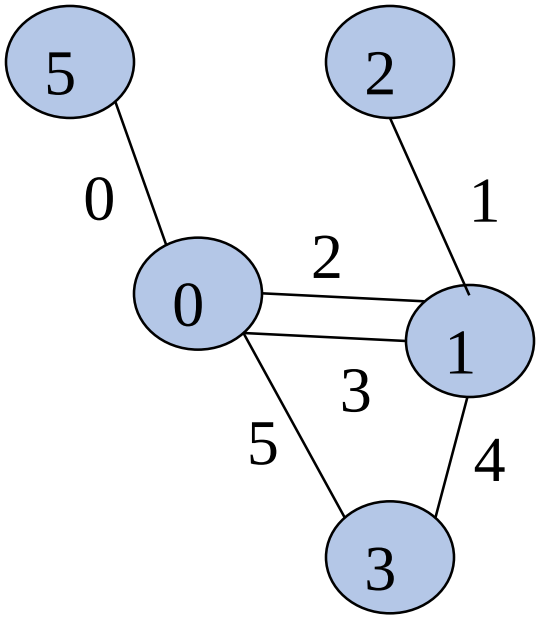
deleteEdge(2)

```
EDGE {
     id = 5
     dynamicID = 0
     nodeID[2] = {2, 3}
};
```

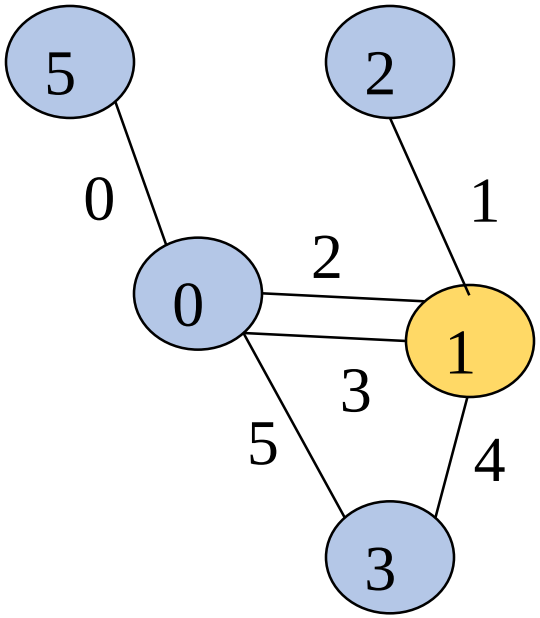




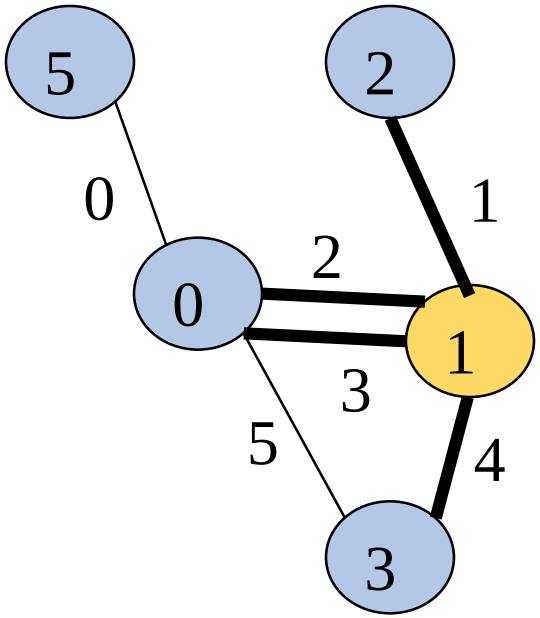
Node deletion



Node deletion

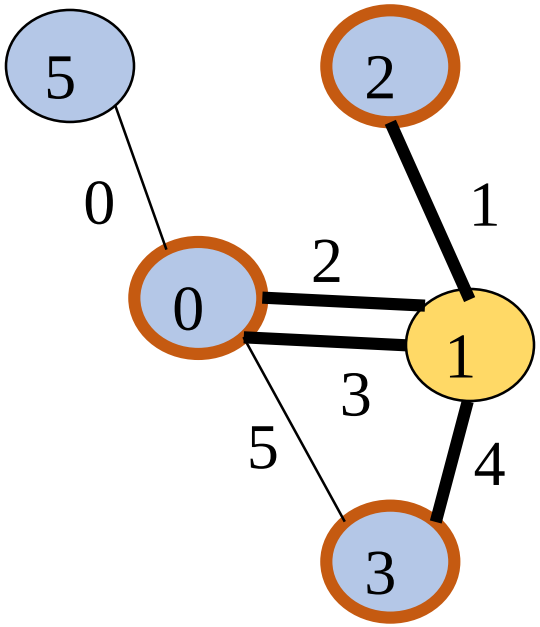


Node deletion



Edges that connect the node.

Node deletion



Nodes that are adjacent to the node.

Enjoy programming