AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

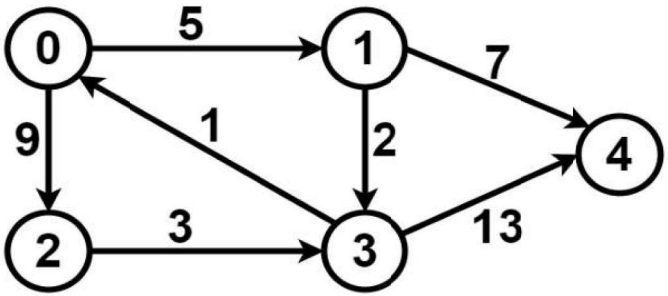
22BIO211: Intelligence of Biological Systems - 2

GRAPHS IN PYTHON

Dr. Manjusha Nair M
Amrita School of Computing, Amrita Vishwa Vidyapeetham
Email : manjushanair@am.amrita.edu
Contact No: 9447745519

Representing Graphs in Python

Adjacency Matrix Representation



	0	1	2	3	4
0	0	5	9	0	0
1	0	0	0	2	7
2	0	0	0	3	0
3	1	0	0	0	13
4	0	0	0	0	0

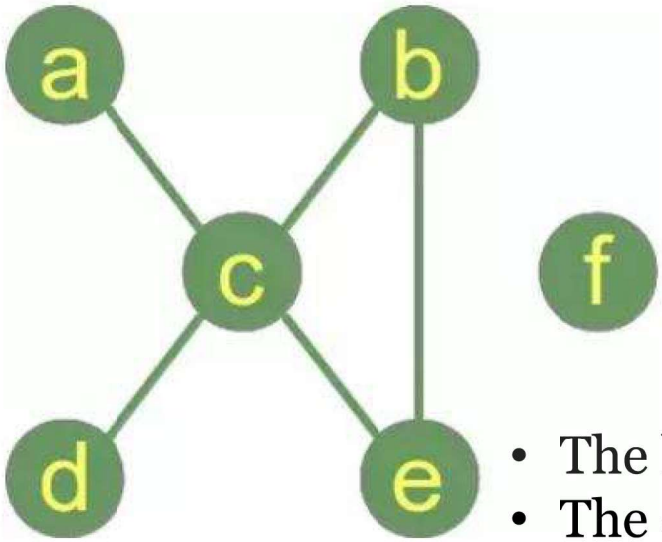
In Python, we can represent graphs like this using a two-dimensional array – ie by creating a **list of lists**.

```
Graph= [
       [0, 5, 9, 0, 0],
       [0, 0, 0, 2, 7],
       [0, 0, 0, 3, 0],
       [1, 0, 0, 0, 13],
       [0, 0, 0, 0, 0]
       ]
```

Representing Graphs in Python

Adjacency List Representation

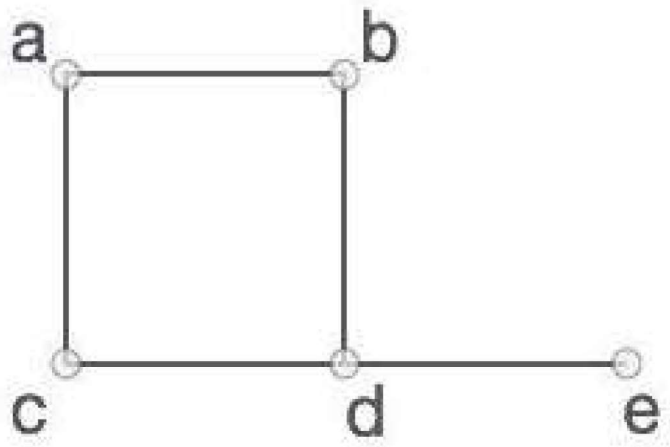
- The Python data structure 'Dictionary' is ideal for representing such graphs in Python




```
graph = { "a" : ["c"],
          "b" : ["c", "e"],
          "c" : ["a", "b", "d","e"],
          "d" : ["c"],
          "e" : ["c", "b"],
          "f" : []
        }
```

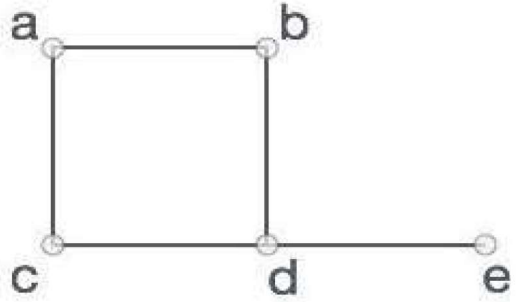
- The keys of the dictionary are the nodes of graph
- The corresponding values are **list** with the nodes, which are connected by an edge

Representing Graphs in Python- Example




```
# Create the dictionary with
graph elements
graph = {   "a" : ["b","c"],
            "b" : ["a", "d"],
            "c" : ["a", "d"],
            "d" : [“e”,”c”,”b”],
            "e" : ["d"]
        }
 # Print the graph
  print(graph)
```

Representing Graphs in Python - Example



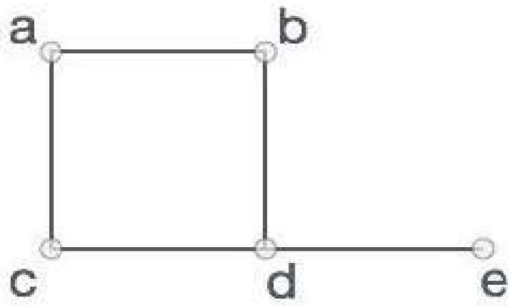
Display vertices of the graph

```
print("The vertices of the graph are")
   print(list(graph.keys()))
```

Display edges of the graph

```
print("The edges of the graph are")
edges=[]
# for each node(key) in graph
for node in graph:
   # for each neighbour node of a single node
   for neighbour in graph[node]:
      # if edge exists then append
      edges.append((node, neighbour))
print(edges)
```

Representing Graphs in Python- Example

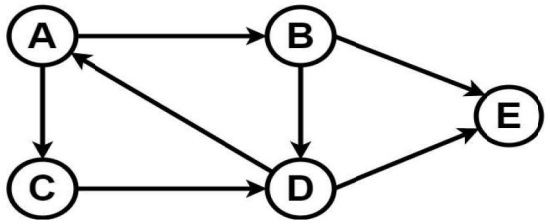


Output

```
{'a': ['b', 'c'], 'b': ['a', 'd'], 'c': ['a', 'd'], 'd': ['e'], 'e': ['d']}
The vertices of the graph are
['a', 'b', 'c', 'd', 'e']
The edges of the graph are
[('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'd'), ('c', 'a'), ('c', 'd'),
    ('d', 'e'), ('e', 'd')]
```

Representing Graphs in Python : Another example

Adjacency List Representation



A : B, C
B : D, E
C : D
D : E, A
E :

We associate all the neighbor nodes of a node together.

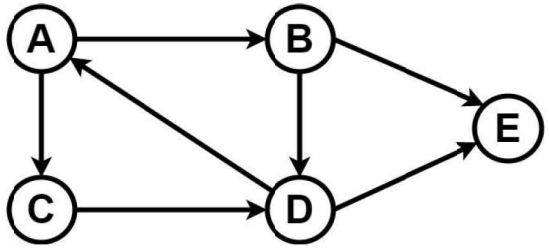
```
graph = {  'A': ['B', 'C'],
           'B': ['D', 'E'],
           'C': ['D'],
           'D': ['E', 'A'],
           'E': [],
           }
```

```
for key, val in graph.items():
    print(f"{key}-->{val}")
```

Construct Graph in Python using user input

- graph = dict()

- addEdge('A', 'B')
- addEdge('A', 'C')
- addEdge('B', 'D')
- addEdge('B', 'E')
- addEdge('C', 'D')
- addEdge('D', 'A')
- addEdge('D', 'E')



A : B, C
B : D, E
C : D
D : E, A
E :

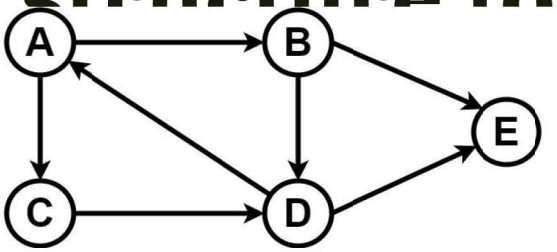
```
for key, val in graph.items():
    print(f"{key}-->{val}")
```

Construct Graph in Python using user input

- def addEdge(node1, node2):

```
# create an empty list for a key node
        if node1 not in graph:
                graph[node1] = []
        if node2 not in graph:
                graph[node2] = []
# append the neighbor node to its corresponding key node
        graph[node1].append(node2)
```

Construct Graph in Python using user input – The whole graph structure together



A : B, C
B : D, E
C : D
D : E, A
E :

- First input the number of nodes and edges of the graph

  5 7 - > nodes, edges = input().split()

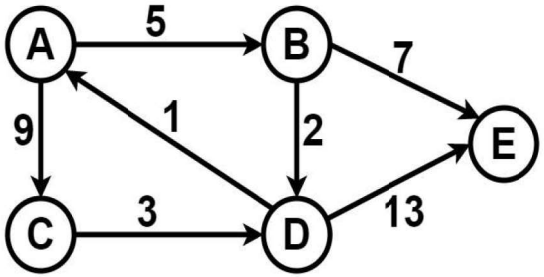
- Then, input each pair of nodes having an edge between them

```
for x in range(int(edges)):
    node1, node2 = input().split()
    addEdge(node1, node2)
```

A B
A C
B D
B E
C D
D A
D E

Construct Graph in Python using user input – The whole graph structure together

- **Adjacency list of a graph with path costs:**



A : (B, 5), (C, 9)
B : (D, 2), (E, 7)
C : (D, 3)
D : (A, 1), (E, 13)
E :

To create this graph, in each input we will take two neighbor nodes along with their path cost:

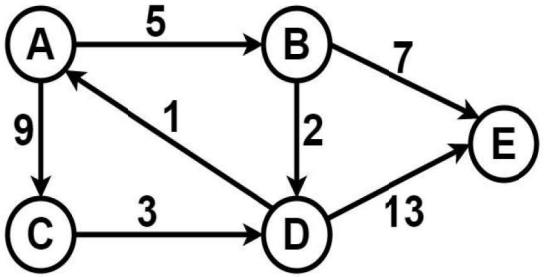
input

```
5 7
A B 5
A C 9
B D 2
B E 7
C D 3
D A 1
D E 13
```

```
for x in range(int(edges)):
        node1, node2,cost = input().split()
        addEdge(node1, node2,cost)
```

Construct Graph in Python using user input – The whole graph structure together

- **Adjacency list of a graph with path costs:**



A : (B, 5), (C, 9)
B : (D, 2), (E, 7)
C : (D, 3)
D : (A, 1), (E, 13)
E :

Each append operation we will take the neighbor node and path cost as a tuple

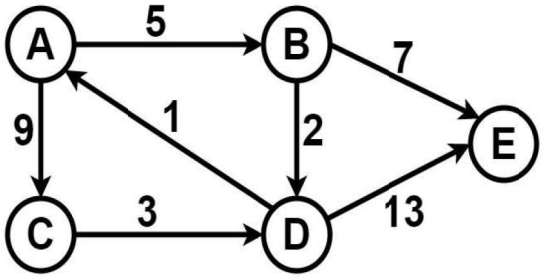
input

5 7
A B 5
A C 9
B D 2
B E 7
C D 3
D A 1
D E 13

```
def addEdge(node1, node2,cost):
# create an empty list for a key node
    if node1 not in graph:
        graph[node1] = []
    if node2 not in graph:
        graph[node2] = []
# append the neighbor node to its corresponding key node
    graph[node1].append(node2,(int)cost)
```

Construct Graph in Python using user input – From File

- **Adjacency list of a graph with path costs:**



A : (B, 5), (C, 9)
B : (D, 2), (E, 7)
C : (D, 3)
D : (A, 1), (E, 13)
E :

Create a file **input.txt** with the following content

```
5 7
A B 5
A C 9
B D 2
B E 7
C D 3
D A 1
D E 13
```

Reading from the file:

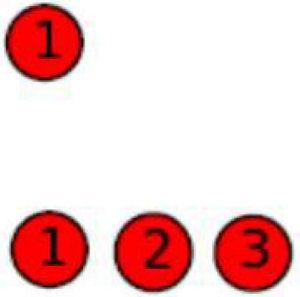
```
with open("input.txt") as f:
    lines = f.readlines()
nodes, edges = lines[0].split()
#first line: number of nodes and edges.
 # Pair of nodes starts from second line
 for i in range(1, len(lines)):
     node1, node2, cost = lines[i].split()
     g.addEdge(node1, node2, int(cost))
```

Software for Complex networks : NetworkX

- It is possible to draw small graphs with NetworkX
  - *https://networkx.org/*
  - *https://networkx.org/documentation/stable/tutorial.html*
- Different classes exist for directed and undirected networks.
  - *>>> import networkx as nx*
- Let's create a basic undirected Graph:
  - *>>> g = nx.Graph() # empty graph*
- The graph g can be grown in several ways. NetworkX provides many generator functions and facilities to read and write graphs in many formats.

NetworkX : Add nodes

- # One node at a time
  - >>> *g.add_node(1)*
- # A list of nodes
  - >>> *g.add_nodes_from([2, 3])*



NetworkX : Add edges

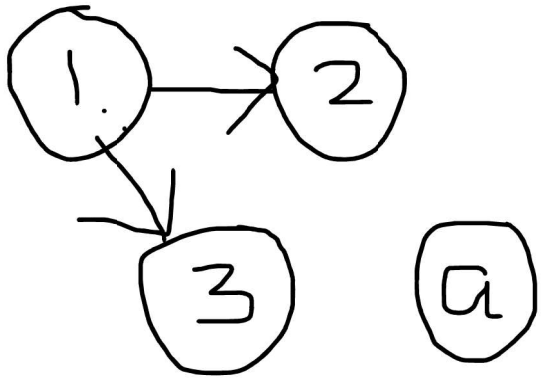
- # Single edge
  - *>>> g.add_edge(1, 2)*
  - *>>> e = (2, 3)*
  - *>>> g.add_edge(*e) # unpack tuple*
- # List of edges
  - *>>> g.add_edges_from([(1, 2), (1, 3)])*

NetworkX : Remove nodes and edges

- Remove any node of the graph
  - >>> *g.remove_node(2)*
- # Remove any edge
  - >>> *g.remove_edge(1, 2)*

NetworkX : Accessing Nodes and Edges

- >>>g.add_nodes_from([1, 2, 3])
- >>> g.add_edges_from([(1, 2), (1, 3)])
- >>> g.add_node('a')
- >>> g.number_of_nodes() # also g.order()
  - *4*
- >>> g.number_of_edges() # also g.size()
  - *2*
- >>> g.nodes()
  - *['a', 1, 2, 3]*
- >>> g.edges()
  - *[(1, 2), (1, 3)]*




```
>>> g.neighbors(1)
    [2, 3]
>>> g.degree(1)
    2
```

NetworkX : Visualizing Graphs

- ***NetworkX*** is not a graph visualizing package but basic drawing with Matplotlib is included in the software package.
- **Step 1 - Import networkx and matplotlib.pyplot**
  - *# importing networkx*
    - import networkx as nx
  - *# importing matplotlib.pyplot*
    - import matplotlib.pyplot as plt

Step 2 : Generate a graph using networkx.

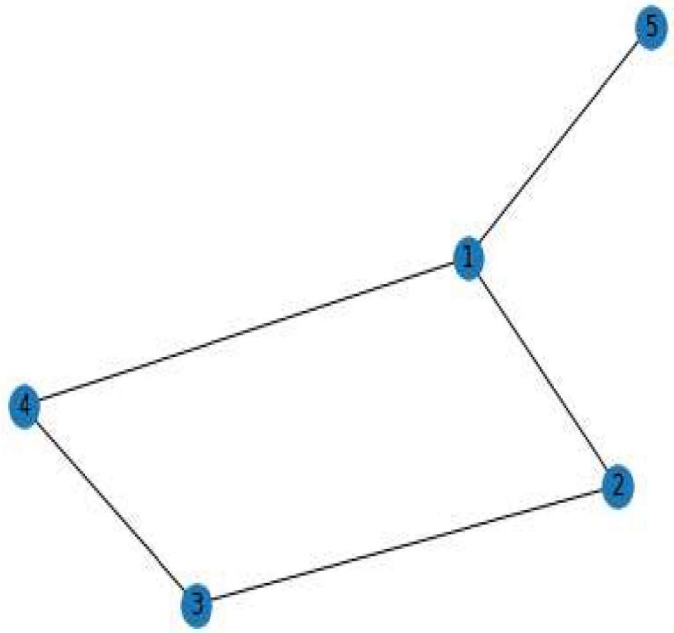
```
g = nx.Graph()
g.add_nodes_from([1,2,3,4,5])
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 4)
g.add_edge(1, 4)
g.add_edge(1, 5)
```

NetworkX : Visualizing Graphs

- **Step 3 :** Now use draw() function of networkx.drawing to draw the graph.
  - nx.draw(g)
- **Step 4 :** Use savefig("first.png") function of matplotlib.pyplot to save the drawing of graph in filename.png file.
  - *plt.savefig("first.png")*
- To add numbering in the node add one argument **with_labels=True** in draw() function.
  - *nx.draw(g,with_labels=True)*

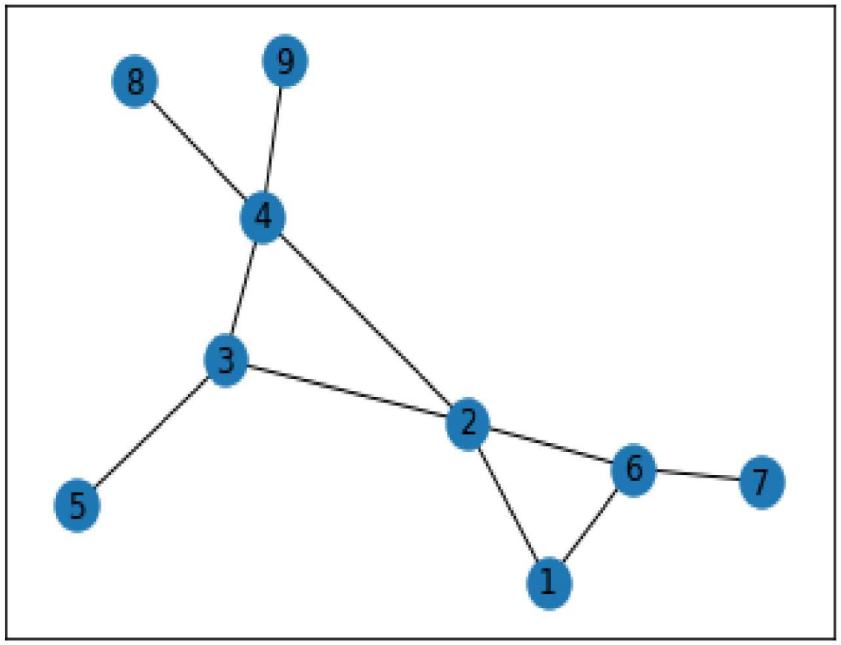
NetworkX : Visualizing Graphs

```
# importing networkx
import networkx as nx
# importing matplotlib.pyplot
import matplotlib.pyplot as plt

g = nx.Graph()
g.add_nodes_from([1,2,3,4,5])
g.add_edge(1, 2)
g.add_edge(2, 3)
g.add_edge(3, 4)
g.add_edge(1, 4)
g.add_edge(1, 5)
nx.draw(g,with_labels=True)
plt.savefig("first.png")
```



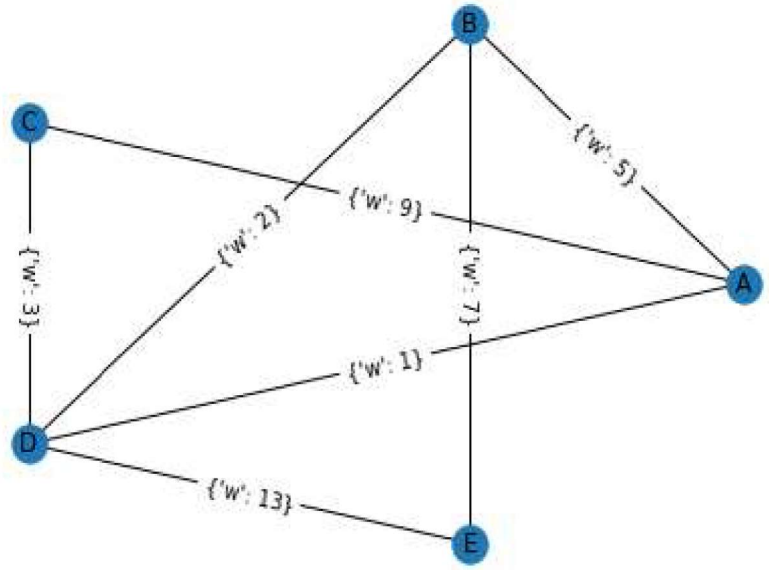
NetworkX : Creating weighted undirected Graphs

```
import networkx as nx
G = nx.Graph()

edges = [(1, 2, 19), (1, 6, 15), (2, 3, 6), (2, 4, 10),
         (2, 6, 22), (3, 4, 51), (3, 5, 14), (4, 8, 20),
         (4, 9, 42), (6, 7, 30)]

G.add_weighted_edges_from(edges)
nx.draw_networkx(G, with_labels = True)
```



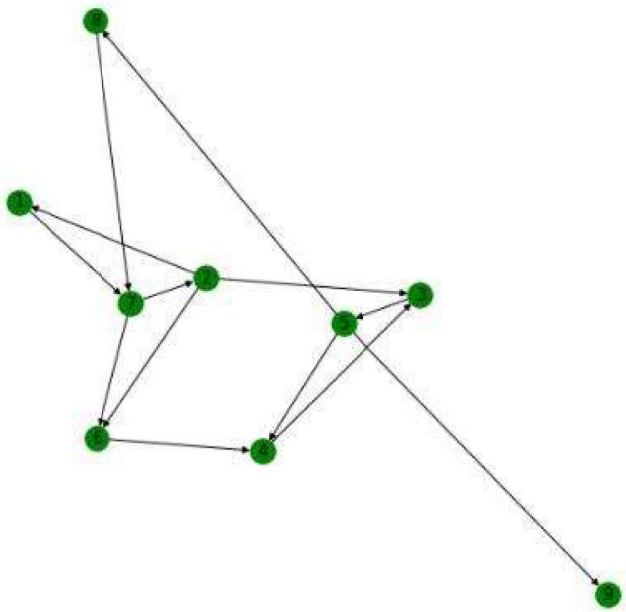
NetworkX : Display the edge weights

```
import networkx as nx
G = nx.Graph()
G.add_nodes_from(["A","B","C","D","E"
])
G.add_edges_from([("A", "B", {"w" : 5}),
                  ("A", "C", {"w" : 9}),
                  ("B", "D", {"w" : 2}),
                  ("B", "E", {"w" : 7}),
                  ("C", "D", {"w" : 3}),
                  ("D", "A", {"w" : 1}),
                  ("D", "E", {"w" : 13})])
pos = nx.circular_layout(G)
labels = nx.get_edge_attributes(G,'w')
nx.draw(G,pos,with_labels=True)
nx.draw_networkx_edge_labels(G,pos=
pos)
```



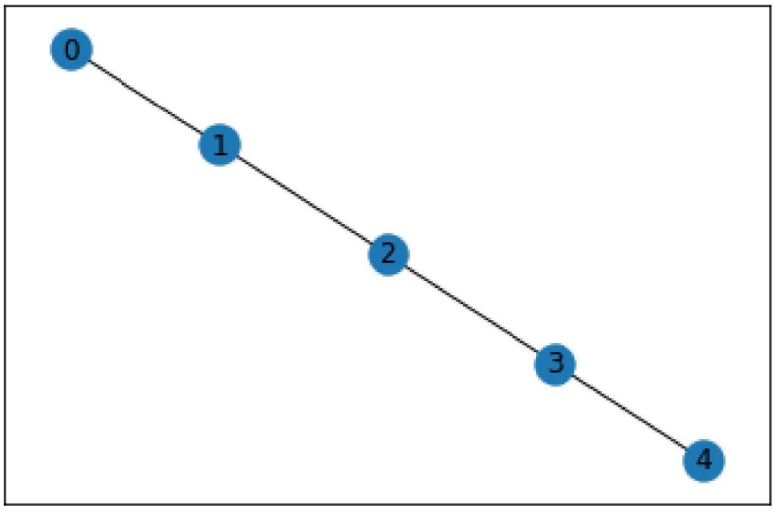
NetworkX :Creating Directed Graphs

```
import networkx as nx
G = nx.DiGraph()
G.add_edges_from([(1, 1), (1, 7), (2, 1), (2, 2), (2, 3
                  (2, 6), (3, 5), (4, 3), (5, 4), (5, 8),
                  (5, 9), (6, 4), (7, 2), (7, 6), (8, 7)])

nx.draw_networkx(G, with_label = True, node_cc
='green')
```

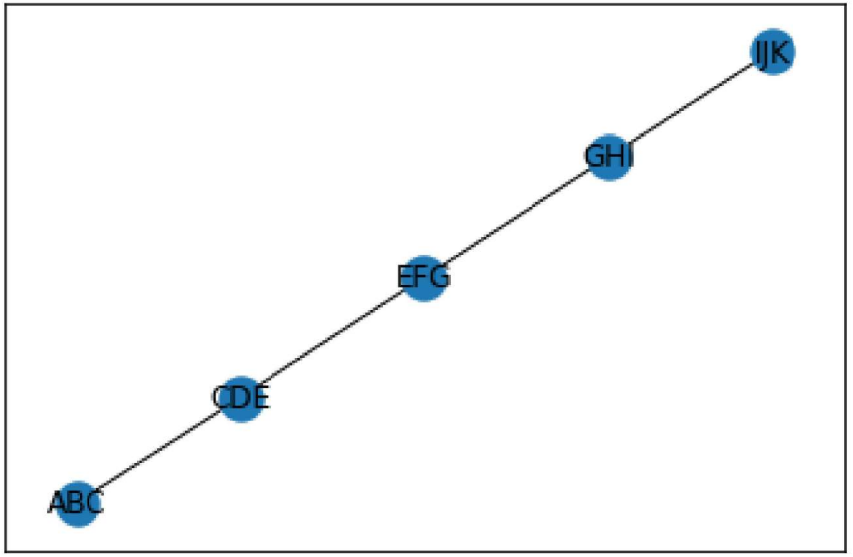


NetworkX :Creating Path Graphs

```
g = nx.path_graph(5)
nx.draw_networkx(g, with_labels = True)
```




```
g = nx.path_graph(5)
new = {0:"ABC", 1:"CDE", 2:"EFG", 3:"GHI",
4:"IJK"}
g = nx.relabel_nodes(g, new)
nx.draw_networkx(g, with_labels = True)
```



Summary

- Construct Graphs in Python
  - *Adjacency Matrix representation – List of lists*
  - *Adjacency List representation using dictionaries*
- Construct graphs using user inputs
- Construct graphs by reading from file
- NetworkX
  - *Add Nodes*
  - *Add Edges*
  - *Remove Nodes and Edges*
  - *Accessing Nodes and Edges*
  - *Visualizing Graphs*
  - *Weighted Graphs*
  - *Directed Graphs*
  - *Path Graphs*