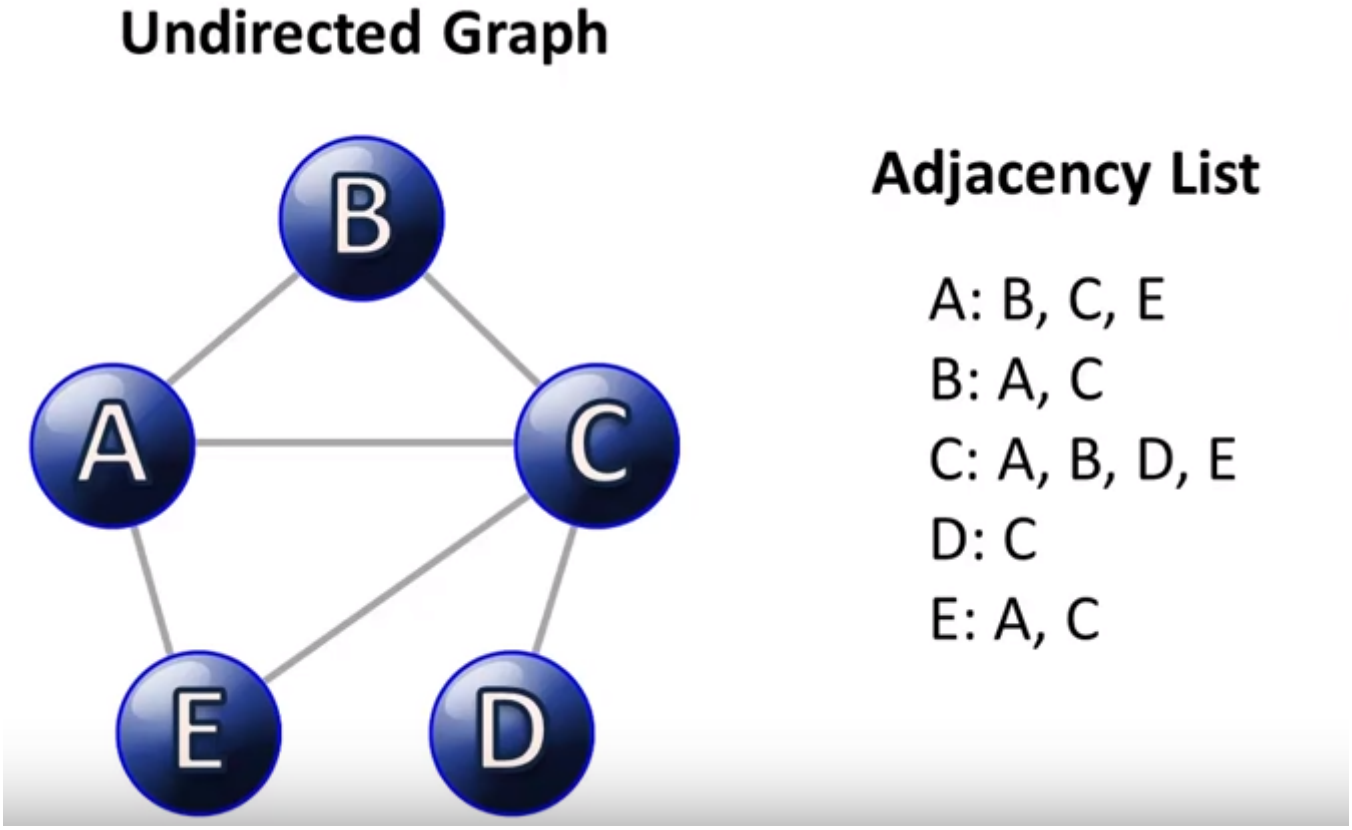
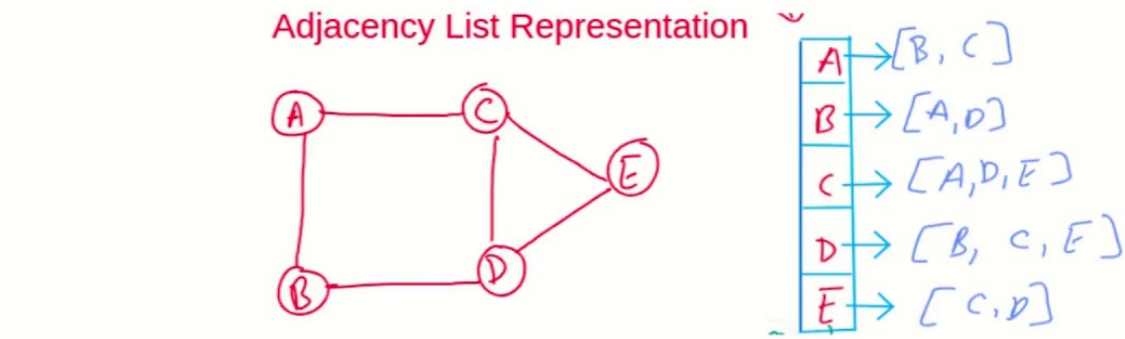
Data Structure for Graphs

- There are four data structures for Representing Grapgh
- Adjacency List
- Adjacency Matrix
- Edge List
- Adjacency map

Adjacency List





In [41]:
```
adj_list = {
    "A":["B","C"],
    "B":["A","D"],
    "C":["A","D","E"],
    "D":["B","C","E"],
    "E":["C","D"]
}
```

In [51]:
```
class Graph:
    def __init__(self, Nodes):
        self.nodes = Nodes
        self.adj_list = {}
```

In [ ]:

In [ ]:

In [8]:
```
adj_list={
          'A':['B','C'],
          'B':['A','D'],
          'C':['A','D','E'],
          'D':['B','C','E'],
          'E':['C','D']
}
```

In [9]:
```
adj_list['B']
```

Out[9]:
```
['A', 'D']
```

In [10]:
```
adj_list['A'].append('D')
# adj_list['D'].append('A')
print(adj_list)
```

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

In [15]:
```
edges=[('A','B'),('A','C'),('B','D'),('C','D'),('C','E'),('D','E')]
```

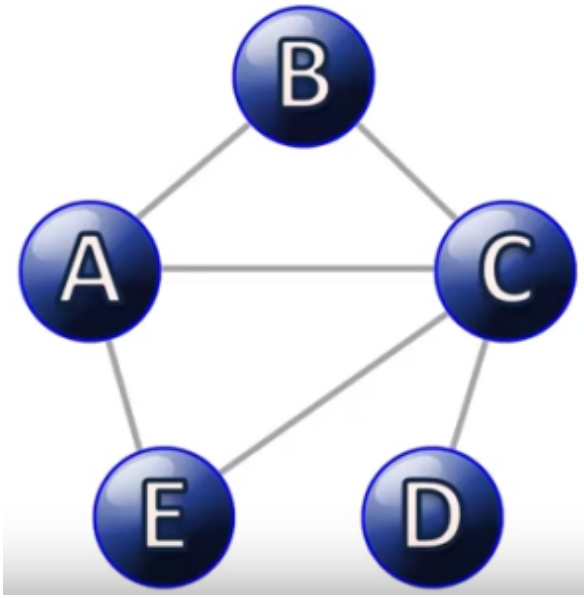
In [20]:

```
class Graph:
    def __init__(self,Nodes):
        self.nodes=Nodes
        self.adj_list={}
        for node in self.nodes:
            self.adj_list[node]=[]
    def add_edges(self,u,v):
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)
    def print_adj_list(self):
        for node in self.nodes:
            print(node,"-->",self.adj_list[node])
    def degree(self,node):
        deg=len(self.adj_list[node])
        return deg
nodes=['A','B','C','D','E']
g=Graph(nodes) # object creation
# print adj_list
for u,v in edges:
    g.add_edges(u,v)
print(g.print_adj_list())
for d in nodes:
    print(d,"Degree of a Node:",g.degree(d))
```

```
A --> ['B', 'C']
B --> ['A', 'D']
C --> ['A', 'D', 'E']
D --> ['B', 'C', 'E']
E --> ['C', 'D']
None
A Degree of a Node: 2
B Degree of a Node: 2
C Degree of a Node: 3
D Degree of a Node: 3
E Degree of a Node: 2
```

Adjacency Matrix

Undirected Graph



Adjacency Matrix

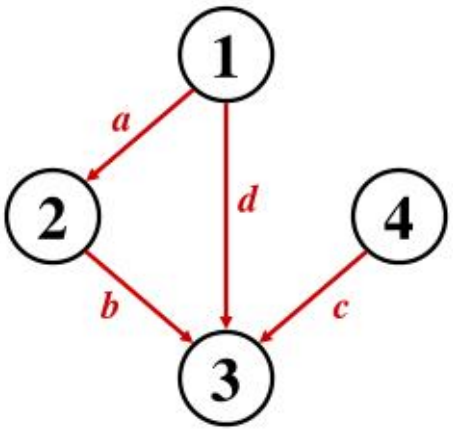
from \ to	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	0
C	1	1	0	1	1
D	0	0	1	0	0
E	1	0	1	0	0

- It is a matrix A[n][n] where, n is number of vertices and {A[i][j]=1 if i&j are Adjacent

=0, Otherwise }

Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1				
2				
3			??	
4				

zhengjin 1 2014/11/7

In [30]:

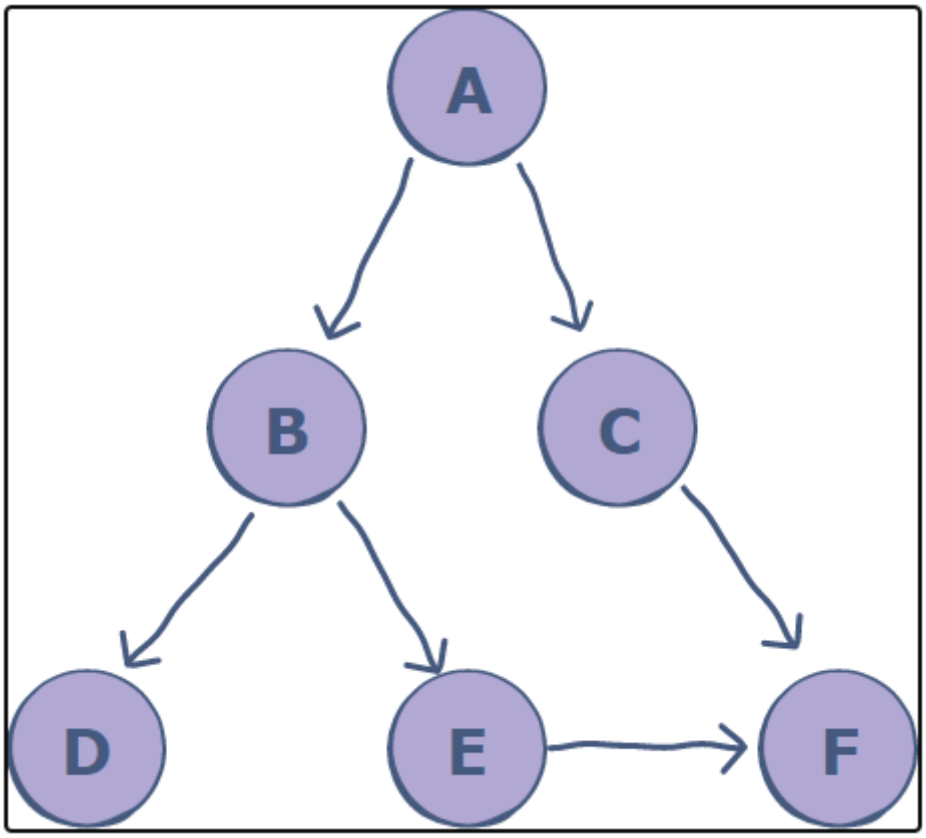
```
class MGraph:
    # Initialize a matrix
    def __init__(self,size):
        self.adjmatrix=[]
        for i in range(size):
            self.adjmatrix.append([0 for i in range(size)])
        self.size=size
# Add edges
    def add_edges(self,v1,v2):
        if v1==v2:
            print("Same vertex")
        self.adjmatrix[v1][v2]=1
        self.adjmatrix[v2][v1]=1
    def __len__(self):
        return self.size
    def print_matrix(self):
        print(self.adjmatrix)

#         for row in self.adjmatrix:
#             for val in row:
#                 print('{:4}'.format(val))
#             print()
mg=MGraph(4)
mg.add_edges(0,1)
mg.add_edges(0,2)
mg.add_edges(1,2)
mg.add_edges(2,0)
mg.add_edges(2,3)
mg.print_matrix()
```

```
[[0, 1, 1, 0], [1, 0, 1, 0], [1, 1, 0, 1], [0, 0, 1, 0]]
```

Graph Traversal

- DFS(Depth First Search)
- BFS(Breadth First Search)

Depth First Search(DFS)



In [45]:

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

In [48]:

```
adj_list
```

Out[48]:

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

In [54]:

```
color={} # W-white, G-Gray, B-Black
parent={}
trav_time={} # [start ,end]
dfs_traversal_list=[] #[A,B,D,E,F,C]
# Initilization
for node in adj_list.keys():
    color[node]="White"
    parent[node]=None
    trav_time[node]=[-1, -1]
time=0
def dfs_util(u):
    global time
    color[u]="Gray"
    trav_time[u][0]=time
    dfs_traversal_list.append(u)

    time+=1
    for v in adj_list[u]:
        if color[v]=="White":
            parent[v]=u
            dfs_util(v)
    color[u]="Black"
    trav_time[u][1]=time
    time+=1
dfs_util("A")
print(dfs_traversal_list)
print(parent)
print(trav_time)
```

```
['A', 'B', 'D', 'E', 'F', 'C']
{'A': None, 'B': 'A', 'C': 'A', 'D': 'B', 'E': 'B', 'F': 'E'}
{'A': [0, 11], 'B': [1, 8], 'C': [9, 10], 'D': [2, 3], 'E': [4, 7], 'F': [5,
6]}
```

In [55]:

```
for node in adj_list.keys():
    print(node,"-->",trav_time[node])
```

```
A --> [0, 11]
B --> [1, 8]
C --> [9, 10]
D --> [2, 3]
E --> [4, 7]
F --> [5, 6]
```

BFS(Breadth First Search)

In [1]:

```
class Vertex:
    def __init__(self, n):
        self.name = n
        self.neighbors = list()

        self.distance = 9999
        self.color = 'black'

    def add_neighbor(self, v):
        if v not in self.neighbors:
            self.neighbors.append(v)
            self.neighbors.sort()

class Graph:
    vertices = {}

    def add_vertex(self, vertex):
        if isinstance(vertex, Vertex) and vertex.name not in self.vertices:
            self.vertices[vertex.name] = vertex
            return True
        else:
            return False

    def add_edge(self, u, v):
        if u in self.vertices and v in self.vertices:
            for key, value in self.vertices.items():
                if key == u:
                    value.add_neighbor(v)
                if key == v:
                    value.add_neighbor(u)
            return True
        else:
            return False

    def print_graph(self):
        for key in sorted(list(self.vertices.keys())):
            print(key + str(self.vertices[key].neighbors) + "  " + str(self.vertices[key].d

    def bfs(self, vert):
        q = list()
        vert.distance = 0
        vert.color = 'red'
        for v in vert.neighbors:
            self.vertices[v].distance = vert.distance + 1
            q.append(v)

        while len(q) > 0:
            u = q.pop(0)
            node_u = self.vertices[u]
            node_u.color = 'red'

            for v in node_u.neighbors:
                node_v = self.vertices[v]
                if node_v.color == 'black':
                    q.append(v)
                    if node_v.distance > node_u.distance + 1:
                        node_v.distance = node_u.distance + 1

g = Graph()
```

```
a = Vertex('A')
g.add_vertex(a)
g.add_vertex(Vertex('B'))
for i in range(ord('A'), ord('K')):
    g.add_vertex(Vertex(chr(i)))

edges = ['AB', 'AE', 'BF', 'CG', 'DE', 'DH', 'EH', 'FG', 'FI', 'FJ', 'GJ', 'HI']
for edge in edges:
    g.add_edge(edge[:1], edge[1:])

g.bfs(a)
g.print_graph()
```

```
A['B', 'E']  0
B['A', 'F']  1
C['G']  4
D['E', 'H']  2
E['A', 'D', 'H']  1
F['B', 'G', 'I', 'J']  2
G['C', 'F', 'J']  3
H['D', 'E', 'I']  2
I['F', 'H']  3
J['F', 'G']  3
```

In [ ]: